UNIVERSITAT POLITÈCNICA DE VALÈNCIA

UNIVERSITAT POLITÈCNICA DE VALÈNCIA
CAMPUS D'ALCOI

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Politécnica Superior de Alcoy

Métodos de Clasificación con Python: Aplicaciones Empresariales

Trabajo Fin de Máster

Máster Universitario en Dirección de Empresas (MBA)

AUTOR/A: Jiang , Qinhan

Tutor/a: Juan Pérez, Ángel Alejandro

Cotutor/a: Pérez Bernabeu, Elena

CURSO ACADÉMICO: 2022/2023

Resumen

Este trabajo proporciona una visión general de los diferentes métodos de aprendizaje automático y su aplicación en el campo de la ciencia de datos en una empresa. Se definen los objetivos del estudio y se describe la metodología empleada. El trabajo abarca varios tipos de aprendizaje automático, incluyendo el aprendizaje supervisado y el no supervisado.

Los métodos de aprendizaje supervisado discutidos en este artículo incluyen la regresión lineal, la regresión logística, los árboles de decisión, el bosque aleatorio, las máquinas de vectores de soporte (SVM) y los vecinos más cercanos (KNN). Se explica cada método en detalle, destacando sus conceptos clave y algoritmos.

Los métodos de aprendizaje no supervisado cubiertos en este trabajo son el análisis de clusters y el análisis de componentes principales (PCA). Estos métodos se utilizan para descubrir patrones y relaciones dentro de los datos sin necesidad de ejemplos etiquetados.

Además, el trabajo presenta aplicaciones del mundo real de la ciencia de datos en una empresa. El análisis se centra en el comportamiento del cliente, como la distribución de pedidos, la frecuencia de compra y las preferencias del cliente. Los resultados revelan información sobre las preferencias del cliente y resaltan la importancia de comprender los patrones de compra del cliente.

El análisis de los datos de productos proporciona información valiosa sobre los productos más vendidos, los productos con altas tasas de recompra y los períodos de tiempo en los que ciertos productos son más populares. Estas ideas pueden ayudar a la empresa a tomar decisiones informadas sobre estrategias de productos y campañas de marketing.

El trabajo concluye con una discusión sobre la aplicación del aprendizaje supervisado y no supervisado en la empresa. Explora cómo se puede utilizar la regresión logística para predecir las recompras de los clientes y cómo se puede lograr la segmentación de clientes utilizando técnicas de clustering.

En general, este estudio demuestra la aplicación práctica de técnicas de aprendizaje automático en el contexto de una empresa, resaltando su potencial para mejorar la toma de decisiones y mejorar el rendimiento empresarial.

Palabras clave:
Aprendizaje automático, ciencia de datos, aprendizaje automático supervisado, aprendizaje automático no supervisado, empresa.

Abstract

This paper provides an overview of the different machine learning methods and their application in the field of data science in a company. The objectives of the study are defined, and the methodology employed is described. The paper covers various types of machine learning, including supervised and unsupervised learning.

The supervised learning methods discussed in this paper include linear regression, logistic regression, decision trees, random forest, support vector machines (SVM), and k-nearest neighbors (KNN). Each method is explained in detail, highlighting its key concepts and algorithms.

The unsupervised learning methods covered in this paper are cluster analysis and principal component analysis (PCA). These methods are used to discover patterns and relationships within the data without the need for labeled examples.

Furthermore, the paper presents real-world applications of data science in a company. The analysis focuses on customer behavior, such as order distribution, purchase frequency, and customer preferences. The results reveal insights into customer preferences and highlight the importance of understanding customer buying patterns.

The analysis of product data provides valuable information about the best-selling products, products with high repurchase rates, and the time periods when certain products are most popular. These insights can help the company make informed decisions regarding product strategies and marketing campaigns.

The paper concludes with a discussion on the application of supervised and unsupervised learning in the company. It explores how logistic regression can be used to predict customer repurchases and how customer segmentation can be achieved using clustering techniques.

Overall, this study demonstrates the practical application of machine learning techniques in the context of a company, highlighting their potential for improving decision-making and enhancing business performance.

Key words:
Machine learning, data science, supervised machine learning, unsupervised machine learning, company

Índice

1. Introducción

Hoy en día las aplicaciones de Python en empresas pueden favorecer las empresas analizar los resultados obtenidos y hacer planes. En general, la aplicación de la ciencia de datos en las empresas puede proporcionar una ventaja competitiva significativa al permitir la toma de decisiones más informadas y efectivas.

2. Objetivos

El objetivo principal de este trabajo es explorar las aplicaciones de Data Science en empresas y cómo pueden contribuir al crecimiento y la eficiencia de las mismas. Para ello, se analizarán casos prácticos de empresas que han implementado técnicas de Data Science en su gestión empresarial.

También se explicarán los modelos de aprendizaje supervisado por los ejemplos.

Además, se pretende identificar las principales dificultades que se pueden presentar en la aplicación de estas técnicas y cómo pueden ser superadas.

3. Metodología

Para llevar a cabo este trabajo se realizará una revisión bibliográfica del estado de la aplicación de la ciencia de datos en las empresas. Se buscarán ejemplos prácticos de empresas que utilicen técnicas de ciencia de datos y se analizarán los resultados obtenidos.

El ciclo de vida de la ciencia de datos es un proceso iterativo para ejecutar proyectos de ciencia de datos. El proceso se compone de varias etapas, cada una de las cuales se enfoca en una tarea específica que debe completarse antes de pasar a la siguiente etapa. Las principales fases del ciclo de vida de la ciencia de datos son las siguientes:

Entender el problema: En esta fase se establecen los objetivos del proyecto y se definen las variables relevantes para el análisis.

Recopilación de datos: Se buscarán fuentes de datos relevantes para la pregunta que se analizará. Los datos pueden ser internos o externos y pueden incluir datos estructurados (por ejemplo, datos de ventas) o datos no estructurados (por ejemplo,

reseñas de clientes en las redes sociales).

Análisis exploratorio de datos: los datos recopilados se analizarán inicialmente para comprender mejor su estructura y patrones de detección. Esto puede incluir mirar datos y realizar estadísticas descriptivas.

Preprocesamiento de datos: en esta fase, los datos se preparan para el análisis mediante la eliminación de valores atípicos, la imputación de valores faltantes y la transformación de los datos si es necesario.

Selección y entrenamiento del modelo: los datos recopilados se utilizarán para seleccionar y entrenar el modelo de aprendizaje automático que mejor se adapte al problema en cuestión.

Evaluación del modelo: el rendimiento del modelo seleccionado se evaluará utilizando métricas relevantes para el problema en cuestión, como la precisión o la puntuación F1.

Implementación del modelo: una vez que se ha evaluado un modelo y se ha determinado que funciona bien, se puede implementar en producción para su uso continuo en el negocio.

4. Tipos de Aprendizaje Automático

En la tesis, se introducen dos tipos de aprendizaje automático, el aprendizaje supervisado y el aprendizaje no supervisado. La diferencia principal es que el aprendizaje supervisado es si los datos son etiquetados.

En cuanto al aprendizaje, por ejemplo (Figura 1), se clasifican las comidas y bebidas por etiquetas ´helado´, ´tarta´,´café´, así que la máquina pueda clasificarlas según las etiquetas.

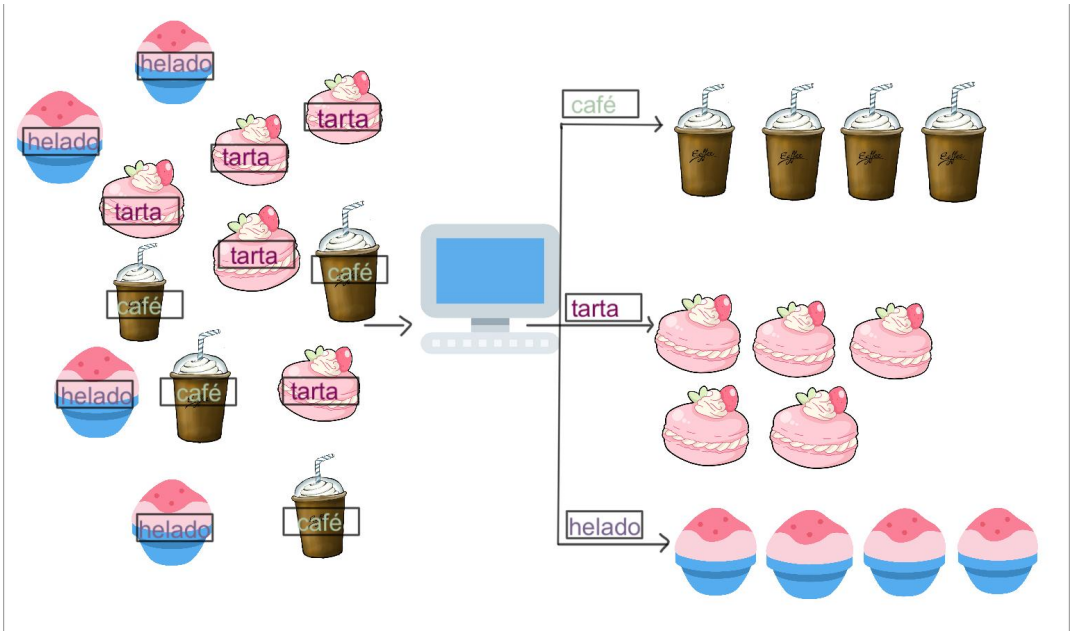


Figura 1 Ejemplo de aprendizaje supervisado Fuente: Elaboración propia

Respecto al aprendizaje no supervisado (Figura 2), se observa que no hay etiquetas en los datos originales, pero la máquina clasifica los datos según las características similares.

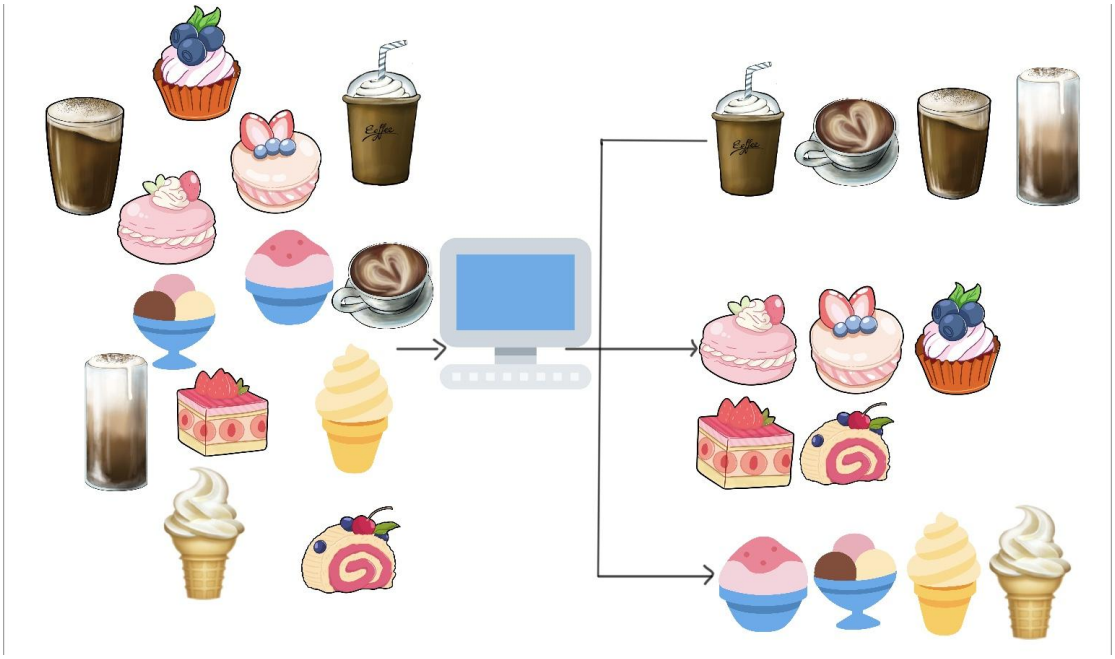


Figura 2 Ejemplo de aprendizaje no supervisado Fuente: Elaboración propia

El aprendizaje supervisado

El aprendizaje supervisado es una forma de aprendizaje automático en la que el conjunto de datos etiquetados, es decir, cada observación tiene asociada una clase específica. El objetivo del aprendizaje supervisado es **establecer una función** que pueda describir la relación entre **las entradas dadas** (las características presentes en nuestro conjunto de datos) y **la variable de salida** de la mejor manera posible.

Por lo general, si x es nuestra variable de entrada y y es nuestra variable de salida, una ecuación de aprendizaje supervisado se vería algo así:

$y = f(x)$

Donde f() es la función que nos da la relación entre la(s) variable(s) de entrada (x) y la variable de salida (y).

Para predecir los resultados de un conjunto de datos, se pueden utilizar diferentes técnicas de aprendizaje automático. Una de las categorías principales es el aprendizaje supervisado, que se divide en dos tipos: **clasificación y regresión.**

La clasificación

La clasificación en el aprendizaje supervisado es **cuando puede asignar etiquetas específicas a sus observaciones**. El objetivo es encontrar una función que asigne una etiqueta a cada entrada.

La regresión es un algoritmo utilizado en el aprendizaje supervisado cuando sus etiquetas son **variables continuas**, también conocidas como etiquetas de valor real.

El aprendizaje no supervisado

El aprendizaje no supervisado es una forma de aprendizaje automático en la que el conjunto de datos no etiquetados; es decir, no hay clases asociadas con las

observaciones inicialmente. El objetivo del aprendizaje no supervisado es **explorar las posibles relaciones dentro de las observaciones dadas** y **agruparlas en grupos similares** (Figura 3).

Por lo general, si x es nuestra entrada, **usamos varios algoritmos exploratorios para descubrir una función f()**, que puede crear grupos y asignar estas entradas a un grupo apropiado.

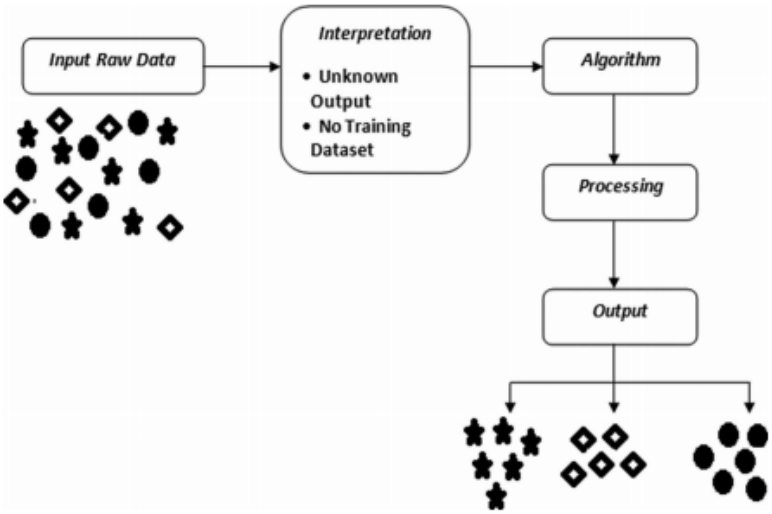


Figura 3: Block diagram of unsupervised learning. Fuente: Sumathi 2022 - Machine Learning for Decision Sciences with Case Studies in Python

En resumen, en el aprendizaje supervisado, el modelo se entrena con datos etiquetados para aprender a predecir la variable de salida correcta, mientras que en el aprendizaje no supervisado, el modelo se entrena con datos no etiquetados para encontrar patrones y relaciones en los datos.

5. Métodos de Aprendizaje Supervisado

5.1. Regresión Lineal

La regresión lineal simple es una forma de regresión lineal donde hay **una variable dependiente y una variable independiente**. Es la forma simple de una regresión lineal que está representada por la ecuación de una línea de la siguiente manera:

$Y = mX + c$

Donde X es la variable independiente e Y es la variable dependiente.

Algunas de las otras terminologías asociadas con X e Y son variables predictoras, variables de características, etc. (para X) y variable objetivo, variable de criterio, etc. (para Y).

Los modelos lineales, si bien son una excelente primera opción debido a su naturaleza intuitiva, también son muy poderosos en su poder predictivo, suponiendo que los conjuntos de datos contengan algún grado de relación lineal o polinomial entre las características y los valores de entrada. La naturaleza intuitiva de los modelos lineales a menudo surge de la capacidad de ver los datos tal como se trazan en un gráfico y observar un patrón de tendencia en los datos con, digamos, la salida (el valor del eje y para los datos) con una tendencia positiva o negativa con la entrada (valor del eje x).

El objetivo de regresión lineal es buscar una línea y hacer que la pase la mayoría de datos lo posible.

La aplicación de regresión lineal en empresas:

Pronóstico de ventas: las empresas pueden usar la regresión lineal para modelar la relación entre las ventas y varios factores, como el gasto en publicidad, la estacionalidad, el tamaño del mercado, etc., para predecir las ventas futuras. Esto es muy útil para formular estrategias de marketing, hacer planes de producción y administrar cadenas de suministro.

Fijación de precios: la regresión lineal se puede utilizar para analizar la relación entre los precios de los productos y varios factores, como el costo, los precios de la competencia, la demanda del mercado, etc., y ayudar a las empresas a determinar la mejor estrategia de precios para maximizar las ganancias o la participación de mercado.

Investigación de mercado: las empresas pueden utilizar la regresión lineal para analizar los datos de investigación de mercado para comprender el comportamiento y las preferencias del consumidor, optimizando así el diseño del producto, mejorando las estrategias de marketing y aumentando la satisfacción del cliente.

Ejemplo de la regresión lineal simple:

```python
import numpy as np
import matplotlib.pyplot as plt

# Datos de ejemplo
x = np.array([1, 2, 3, 4, 5, 6])
y = np.array([2, 3, 4, 4, 5, 7])

# Ajuste de la regresión lineal
coefficients = np.polyfit(x, y, 1)
m = coefficients[0]  # Pendiente
b = coefficients[1]  # Intercepto

# Calcular los valores ajustados de y
y_fit = m * x + b

# Crear el gráfico
fig, ax = plt.subplots()

# Graficar los puntos de datos
ax.scatter(x, y, color='blue', label='Datos')

# Graficar la línea de regresión
ax.plot(x, y_fit, color='red', label='Regresión lineal')

# Configurar los ejes y el título
ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_title('Regresión lineal')

# Mostrar la leyenda
ax.legend()

# Mostrar el gráfico
plt.show()
```

Figura 4: Código de un ejemplo de regresión lineal Elaboración propia

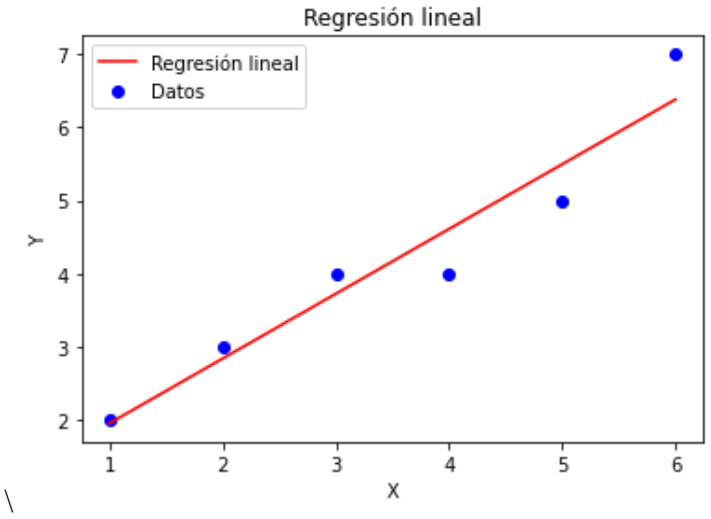


Figura 5: Gráfico de regresión lineal Elaboración propia)

La regresión lineal múltiple (Figura 5) es una forma de regresión lineal en la que hay **una variable dependiente y varias variables independientes**. Es una de las formas más utilizadas de una regresión lineal que se representa mediante la ecuación de una línea de la siguiente manera:

Y = m(X1+X2+X3+X4+X5…) + c

También escrito como:

Y = m1X1+ m2X2+m3X3+m4X4+m5X5+…+c

Donde X1, X2, X3, X4, X5 son las variables independientes e Y es la

variables dependientes. En este caso, m1 m2, m3, m4, m5 son los coeficientes de las variables independientes que deben calcularse utilizando nuestro modelo. Algunas de las otras terminologías asociadas con X e Y son variables explicativas (para X) y la variable de respuesta (para Y).

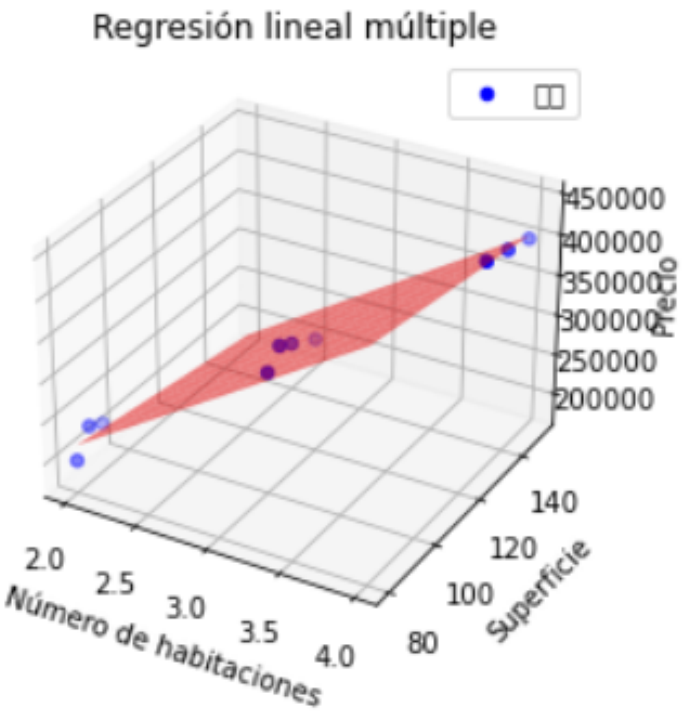
Ejemplo de la regresión lineal múltiple

El precio de una casa en función de sus características como habitaciones y la superficie. Es obvio que el precio de vivienda es más caro cuando tiene más habitaciones y más superficies(Figura 6,7).

```python
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

# Datos de ejemplo
habitaciones = [2, 3, 4, 2, 3, 4, 3, 2, 3, 4]
superficie = [80, 120, 150, 90, 110, 130, 100, 85, 105, 140]
precios = [200000, 300000, 400000, 220000, 320000, 420000, 310000, 230000, 330000, 410000]

# Convertir las características en una matriz de dos dimensiones
X = np.column_stack((np.ones(len(habitaciones)), habitaciones, superficie))

# Convertir los precios en un array unidimensional
y = np.array(precios)

# Calcular los coeficientes de la regresión
coefficients = np.linalg.inv(X.T @ X) @ X.T @ y

# Crear los puntos de la malla para graficar la superficie
x_surf, y_surf = np.meshgrid(np.linspace(2, 4, 10), np.linspace(80, 150, 10))
z_surf = coefficients[0] + coefficients[1] * x_surf + coefficients[2] * y_surf

# Crear la figura
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

# Graficar los puntos de datos
ax.scatter(habitaciones, superficie, precios, color='blue', label='Datos')
```

```python
# Graficar la superficie de regresión
ax.plot_surface(x_surf, y_surf, z_surf, color='red', alpha=0.5)

# Configurar los ejes y el título
ax.set_xlabel('Número de habitaciones')
ax.set_ylabel('Superficie')
ax.set_zlabel('Precio')
ax.set_title('Regresión lineal múltiple')

# Mostrar la leyenda
ax.legend()

# Mostrar el gráfico
plt.show()
```

(Figura 6: Código de un ejemplo de regresión multilineal Elaboración propia)

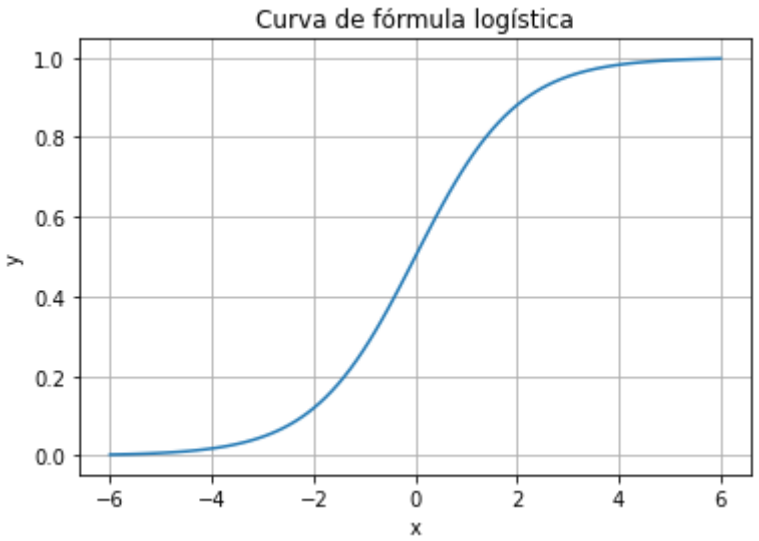


(Figura 7: gráfico de regresión multilineal Elaboración propia)

5.2 Regresión logística

La regresión logística(Figura 8) es un algoritmo de aprendizaje supervisado se utiliza para la clasificación binaria.

La regresión logística utiliza una función logística para modelar la relación entre las variables predictoras y la variable objetivo binaria. La función logística transforma la salida lineal del modelo en un valor entre 0 y 1, que se interpreta como la probabilidad de que el objeto pertenezca a la clase positiva.



(Figura 8: Modelo de regresión logística Fuente: elaboración propia)

La aplicación de regresión logística en empresas

Predicción de comportamientos: La regresión logística puede ayudar a predecir comportamientos de interés para una empresa, como la probabilidad de cancelación de un servicio, el abandono de un carrito de compras en línea o la probabilidad de respuesta a una campaña de marketing.

Análisis de satisfacción del cliente: La regresión logística se puede utilizar para analizar encuestas o comentarios de los clientes y predecir la probabilidad de satisfacción o insatisfacción. Esto permite a las empresas identificar los factores que influyen en la satisfacción del cliente y tomar medidas para mejorarla.

Ejemplo 1 de regresión logística: :

La aprobación de la tarjeta crédito(Figura 9), supongamos que si llega cierto ingreso anual será más fácil aprobar la solicitud de la tarjeta crédito. Aprobación es 1, y no aprobación es 0.

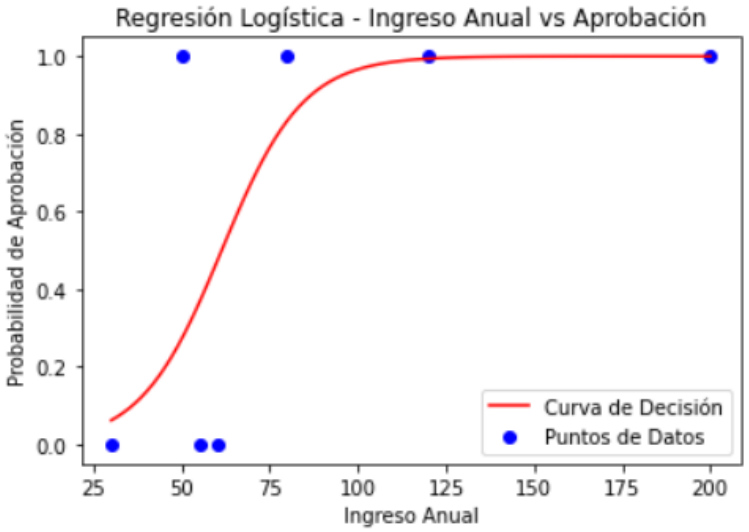
Clientes	Ingreso Anual (en miles de dólares)	Aprobación
1	50	1
2	120	1
3	30	0
4	80	1
5	55	0
6	200	1
7	60	0

(Figura 9: Aprobación de tarjeta crédito Fuente: elaboración propia)

Los códigos(Figura 10) siguientes realizar el modelo de regresión logística y según el modelo predecir la posibilidad de aprobación de clientes con ingreso de 70,40,20 miles de dólares respectivamente.

```python
# Graficar la curva de decisión
x_values = np.linspace(X.min(), X.max(), 100)
probabilidades = modelo.predict_proba(x_values.reshape(-1, 1))
plt.plot(x_values, probabilidades[:, 1], color='red', label='Curva de Decisión')

# Establecer el título y las etiquetas de los ejes
plt.title('Regresión Logística - Ingreso Anual vs Aprobación')
plt.xlabel('Ingreso Anual')
plt.ylabel('Probabilidad de Aprobación')

# Mostrar la leyenda
plt.legend()

# Mostrar el gráfico
plt.show()


# Realizar predicciones
nuevos_datos = {
    'Ingreso Anual': [70, 20, 40],
}

nuevo_df = pd.DataFrame(nuevos_datos)
predicciones = modelo.predict(nuevo_df)

# Imprimir los resultados de las predicciones
for i in range(len(predicciones)):
    print(f"Predicción {i+1}: {predicciones[i]}")
```

(Figura 10: Códigos de aprobación de tarjeta crédito Fuente: elaboración propia)




```
Predicción 1: 1
Predicción 2: 0
Predicción 3: 0
```

(Figura 11: Resultado y predicción de aprobación de tarjeta crédito Fuente: elaboración propia)

Según el resultado(Figura 11), 1 es aprobar mientras 0 es no aprobar, se observa que

solo se predice que el cliente con ingreso de 70 miles dólares puede aprobar, y otros con ingreso de 20 miles de dólares y 40 miles de dólares no pueden aprobar.

Ejemplo 2 de regresión logística:

El objetivo es predecir una variable objetivo binaria "RainTomorrow" basada en el conocimiento existente (por ejemplo, temperatura, humedad, velocidad del viento, etc.).

Paso 1: Importar y procesar los datos

- **Resumen de datos**

Se importan los datos y se echan un vistazo de los datos. En columnas se ven los nombres de los variables,como fecha, lugar,temperatura mínima, temperatura máxima y etc, algunos de ellos se van a utilizar para predecir la lluvia de mañana (Figura 12).

Data preprocessing

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from pandas.api.types import is_string_dtype,is_numeric_dtype

df = pd.read_csv("weatherAUS.csv")
df.head()
```

	Date	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir	WindGustSpeed	WindDir9am	...	Humidity9am
0	2008-12-01	Albury	13.4	22.9	0.6	NaN	NaN	W	44.0	W	...	71.0
1	2008-12-02	Albury	7.4	25.1	0.0	NaN	NaN	WNW	44.0	NNW	...	44.0
2	2008-12-03	Albury	12.9	25.7	0.0	NaN	NaN	WSW	46.0	W	...	38.0
3	2008-12-04	Albury	9.2	28.0	0.0	NaN	NaN	NE	24.0	SE	...	45.0
4	2008-12-05	Albury	17.5	32.3	1.0	NaN	NaN	W	41.0	ENE	...	82.0

5 rows × 23 columns

(Figura 12: procesar los datos Fuente: elaboración propia)

- import **pandas as pd** importa la biblioteca Pandas y se le asigna el alias "pd". Pandas es una biblioteca popular en Python para la manipulación y análisis de datos.
- import **seaborn as sns** importa la biblioteca Seaborn y se le asigna el alias "sns". Seaborn es una biblioteca de visualización de datos basada en

Matplotlib que proporciona una interfaz de alto nivel para crear gráficos estadísticos atractivos e informativos.

- import **matplotlib.pyplot as plt** importa la biblioteca Matplotlib y se le asigna el alias "plt". Matplotlib es otra biblioteca de visualización de datos popular en Python y se usa para crear gráficos estáticos y dinámicos.
- from **pandas.api.types import is_string_dtype,is_numeric_dtype** importa dos funciones de Pandas que se utilizan más adelante en el código.
- **df = pd.read_csv("weatherAUS.csv")** carga el conjunto de datos "weatherAUS.csv" en un DataFrame de Pandas llamado "df". "read_csv()" es una función de Pandas que lee archivos CSV y los convierte en un DataFrame. En este caso, el archivo CSV contiene datos climáticos de Australia.
- **df.head()** muestra las primeras cinco filas del DataFrame "df". Esto es útil para ver cómo se ve el conjunto de datos y asegurarse de que se haya cargado correctamente.

Luego se usa la función **df.describe()** (Figura 13) para proporcionar un resumen estadístico de este dataframe de pandas, incluyendo el conteo, la media, la desviación estándar, los valores mínimo y máximo, y los percentiles de los datos numéricos en el dataframe.

```
In [2]: df.describe()
```

Out[2]:

	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustSpeed	WindSpeed9am	WindSpeed3pm	Humidity9am	Humidity3p
count	143975.000000	144199.000000	142199.000000	82670.000000	75625.000000	135197.000000	143693.000000	142398.000000	142806.000000	140953.0000(
mean	12.194034	23.221348	2.360918	5.468232	7.611178	40.035230	14.043426	18.662657	68.880831	51.5391
std	6.398495	7.119049	8.478060	4.193704	3.785483	13.607062	8.915375	8.809800	19.029164	20.7959(
min	-8.500000	-4.800000	0.000000	0.000000	0.000000	6.000000	0.000000	0.000000	0.000000	0.0000(
25%	7.600000	17.900000	0.000000	2.600000	4.800000	31.000000	7.000000	13.000000	57.000000	37.0000(
50%	12.000000	22.600000	0.000000	4.800000	8.400000	39.000000	13.000000	19.000000	70.000000	52.0000(
75%	16.900000	28.200000	0.800000	7.400000	10.600000	48.000000	19.000000	24.000000	83.000000	66.0000(
max	33.900000	48.100000	371.000000	145.000000	14.500000	135.000000	130.000000	87.000000	100.000000	100.0000(

(Figura 13: Decripción de datos Fuente: elaboración propia)

- **df.describe()** es útil para tener una visión general rápida de los datos numéricos y para detectar posibles valores atípicos y errores en los datos.

- **Abordar los datos perdidos**

A continuación, se abordarán los datos perdidos que pueden ser resultado de las limitaciones de la medición. Primero, se calculará cuántos datos están perdidos y qué

porcentaje representan del total de los datos(Figura 14).

```
df.isnull().sum()
missing_count=df.isnull().sum()
value_count=df.isnull().count()
missing_percentage=round(missing_count/value_count*100,1)
missing_df=pd.DataFrame({'count':missing_count,'perentage':missing_percentage})
print(missing_df)
```

	count	perentage
Date	0	0.0
Location	0	0.0
MinTemp	1485	1.0
MaxTemp	1261	0.9
Rainfall	3261	2.2
Evaporation	62790	43.2
Sunshine	69835	48.0
WindGustDir	10326	7.1
WindGustSpeed	10263	7.1
WindDir9am	10566	7.3
WindDir3pm	4228	2.9
WindSpeed9am	1767	1.2
WindSpeed3pm	3062	2.1
Humidity9am	2654	1.8
Humidity3pm	4507	3.1
Pressure9am	15065	10.4
Pressure3pm	15028	10.3
Cloud9am	55888	38.4
Cloud3pm	59358	40.8
Temp9am	1767	1.2
Temp3pm	3609	2.5
RainToday	3261	2.2
RainTomorrow	3267	2.2

(Figura 14: resolver los datos perdidos Fuente: elaboración propia)

- El código **df.isnull().sum()** calcula el número de valores faltantes (valores nulos) en cada columna de un dataframe df.
- Luego, se guarda el resultado en una variable **missing_count.**
- Después, se cuenta el número total de valores en cada columna con **df.isnull().count().** El resultado se guarda en value_count.
- A continuación, se calcula el porcentaje de valores faltantes en cada columna mediante la fórmula **(missing_count/value_count)*100**, y se redondea a un decimal.
- Por último, se crea un nuevo dataframe llamado **missing_df** con dos columnas: una con el número de valores faltantes y otra con el porcentaje de valores faltantes para cada columna.
- Finalmente, se imprime el dataframe missing_df con el número y el porcentaje de valores faltantes en cada columna del dataframe original df.

```
print(df.columns)
```

```
Index(['Date', 'Location', 'MinTemp', 'MaxTemp', 'Rainfall', 'Evaporation',
       'Sunshine', 'WindGustDir', 'WindGustSpeed', 'WindDir9am', 'WindDir3pm',
       'WindSpeed9am', 'WindSpeed3pm', 'Humidity9am', 'Humidity3pm',
       'Pressure9am', 'Pressure3pm', 'Cloud9am', 'Cloud3pm', 'Temp9am',
       'Temp3pm', 'RainToday', 'RainTomorrow'],
      dtype='object')
```

(Figura 15: Resultado y predicción de aprobación de tarjeta crédito Fuente: elaboración propia)

- Este comando(Figura 16) imprime los nombres de las columnas del DataFrame df.Estos dos comandos eliminan algunas columnas del dataframe df y luego eliminan todas las filas donde la columna RainTomorrow es nula

```
df = df.drop(['Evaporation', 'Sunshine', 'Cloud3pm', 'Cloud9am'],axis=1)
df = df.dropna(subset=["RainTomorrow"])
```

.(Figura 16: Eliminar los datos perdidos de ´RainTomorrow´ Fuente: elaboración propia)

- **df = df.drop(['Evaporation', 'Sunshine', 'Cloud3pm', 'Cloud9am'],axis=1)** elimina las columnas 'Evaporation', 'Sunshine', 'Cloud3pm', 'Cloud9am' del dataframe df. El argumento axis=1 indica que se debe eliminar las columnas y no las filas.
- **df = df.dropna(subset=["RainTomorrow"])** elimina todas las filas donde la columna RainTomorrow es nula. La función dropna elimina las filas donde al menos una columna contiene un valor nulo por defecto, **pero con el argumento subset=["RainTomorrow"] se restringe la eliminación a las filas donde la columna RainTomorrow es nula.**

- Este código(Figura 17) itera a través de todas las columnas en un DataFrame llamado `df`, y

```
num_list=[]
cat_list=[]

for column in df:
    if column !='RainTomorrow':
        if is_numeric_dtype(df[column]):
            num_list.append(column)
        elif is_string_dtype(df[column]):
            cat_list.append(column)

print(num_list)
print(cat_list)
```

```
['MinTemp', 'MaxTemp', 'Rainfall', 'WindGustSpeed', 'WindSpeed9am', 'WindSpeed3pm', 'Humidity9am',
re3pm', 'Temp9am', 'Temp3pm']
['Date', 'Location', 'WindGustDir', 'WindDir9am', 'WindDir3pm', 'RainToday']
```

(Figura 17:Crear listas y clasificar según el tipo de datos Fuente: elaboración propia)

- si una columna no se llama "RainTomorrow", comprueba si es de tipo numérico o de tipo de cadena.
- Si es de tipo numérico, agrega el nombre de la columna a una lista llamada `**num_list**`. Si es de tipo de cadena, agrega el nombre de la columna a una lista llamada `**cat_list**`.
- La salida son dos listas: `**num_list**` que contiene los nombres de las columnas numéricas en `df`, y `**cat_list**` que contiene los nombres de las columnas de cadena en `df`.

Luego se reemplazan los valores faltantes con la media de la columna correspondiente.(Figura 18)

```
df.fillna(df.mean(),inplace=True)
```

```
C:\Users\qinhan\AppData\Local\Temp/ipykernel_380/2085774198.py:1: FutureWarning: Dropping of nu
ith 'numeric_only=None') is deprecated; in a future version this will raise TypeError.  Select
uction.
  df.fillna(df.mean(),inplace=True)
```

```
for i in (cat_list):
    if df[i].isnull().any():
        df[i].fillna("Unknown",inplace=True)
```

(Figura 18: Replacer los datos perdidos por el promedio Fuente: elaboración propia)

- Se ultiliza **Df.fillna()** rellena los valores faltantes (NaN) en un DataFrame df con el valor medio de la columna correspondiente y luego modifica el DataFrame original inplace.

Paso 2: Feature Engineering y EDA

EDA (Exploratory Data Analysis)(Figura 19) se define como un enfoque para analizar y visualizar un conjunto de datos resumiendo sus características principales.
El propósito de EDA es:

• Descubrir patrones dentro de un conjunto de datos

• Detectar anomalías

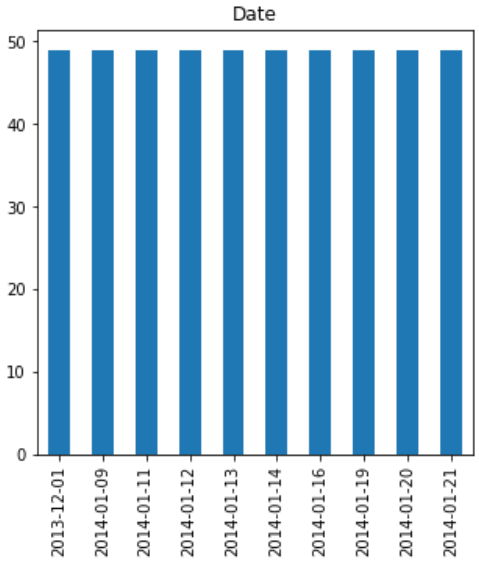
• Formular hipótesis sobre el comportamiento de los datos.

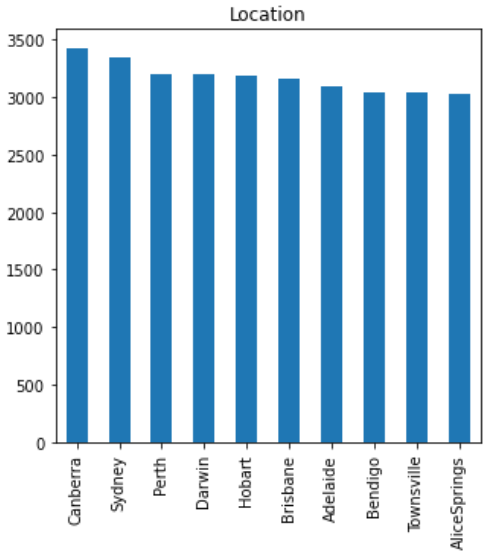
• Validar suposiciones

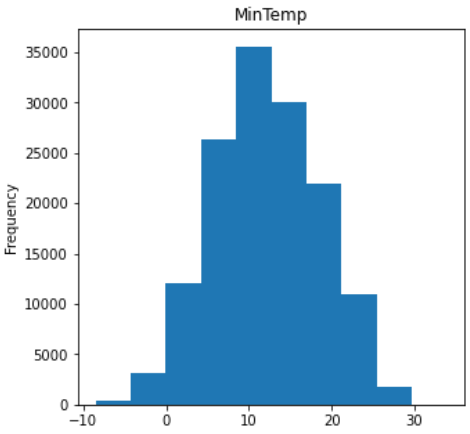
- Este código(Figura 19) genera un gráfico para cada columna del dataframe df. Si la columna es numérica, se dibuja un histograma de la distribución de valores de la columna. Si la columna es de tipo string, se dibuja un gráfico de barras que muestra la cantidad de valores únicos de la columna y su frecuencia en el dataframe. La figura se nombra con el nombre de la columna y se establece un tamaño de figura de 5 por 5.

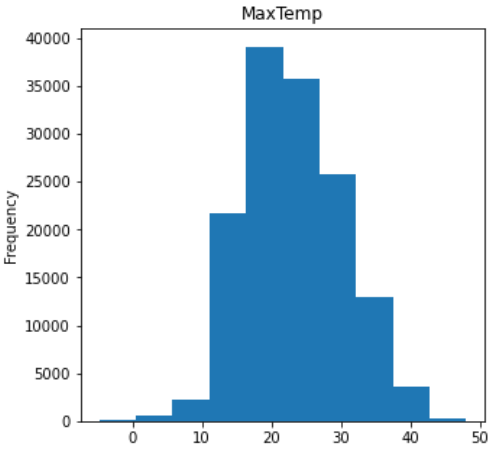
Feature Engeering and EDA

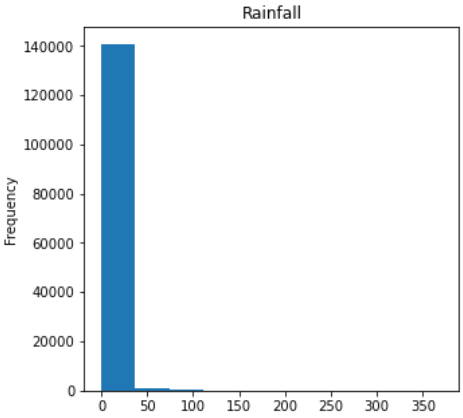
```
for column in df:
    plt.figure(column, figsize=(5, 5))
    plt.title(column)
    if is_numeric_dtype(df[column]):
        df[column].plot(kind='hist')
    elif is_string_dtype(df[column]):
        df[column].value_counts()[:10].plot(kind='bar')
```

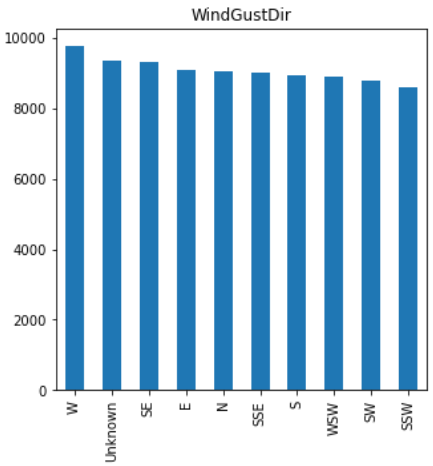

Date
50
40
30
20
10
0
2013-12-01
2014-01-09
2014-01-11
2014-01-12
2014-01-13
2014-01-14
2014-01-16
2014-01-19
2014-01-20
2014-01-21

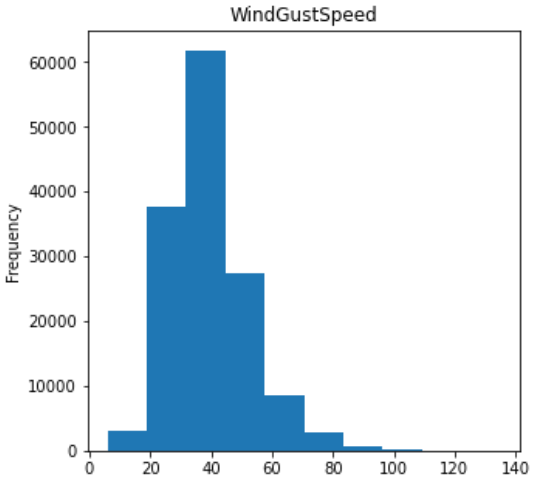

Location
3500
3000
2500
2000
1500
1000
500
0
Canberra
Sydney
Perth
Darwin
Hobart
Brisbane
Adelaide
Bendigo
Townsville
AliceSprings

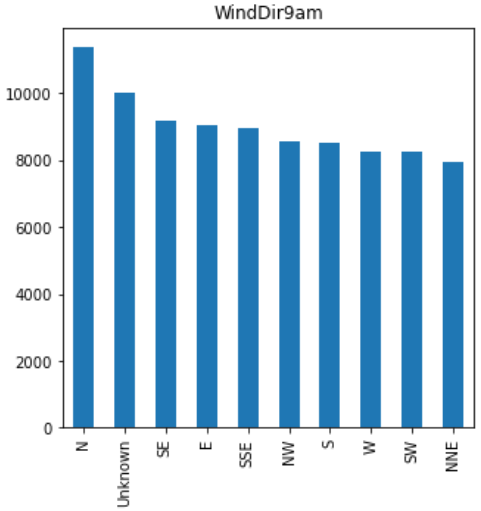

MinTemp
Frequency
35000
30000
25000
20000
15000
10000
5000
0
−10
0
10
20
30

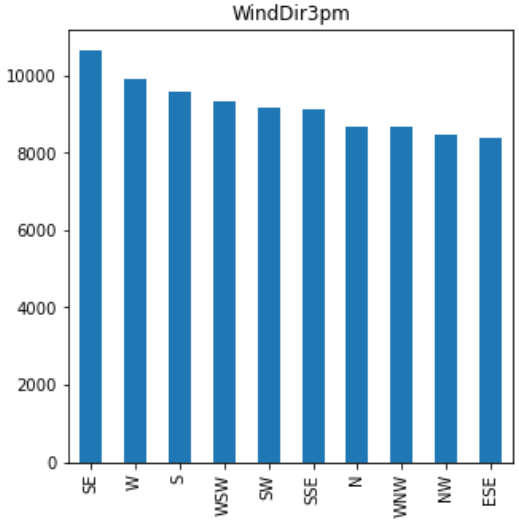

MaxTemp
Frequency
40000
35000
30000
25000
20000
15000
10000
5000
0
0
10
20
30
40
50

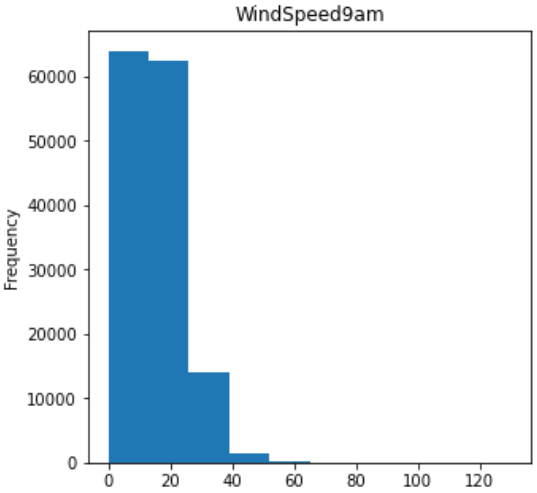

Rainfall
Frequency
140000
120000
100000
80000
60000
40000
20000
0
0
50
100
150
200
250
300
350

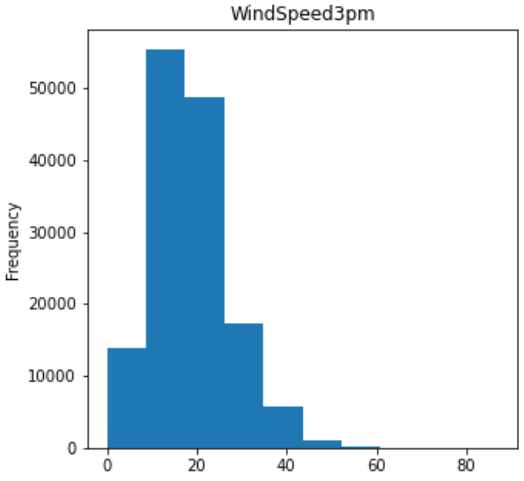

WindGustDir
10000
8000
6000
4000
2000
0
W
Unknown
SE
E
N
SSE
S
WSW
SW
SSW

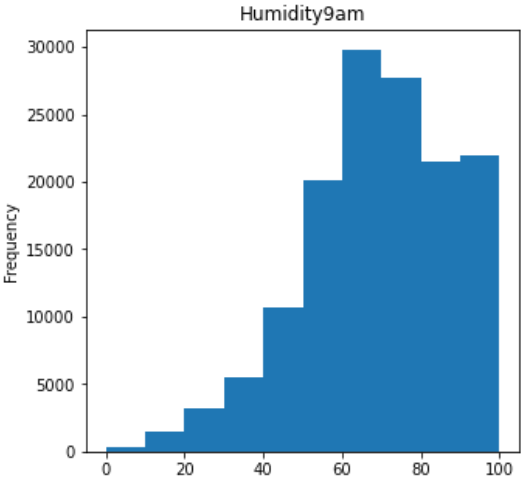

WindGustSpeed
Frequency
0
10000
20000
30000
40000
50000
60000
0
20
40
60
80
100
120
140

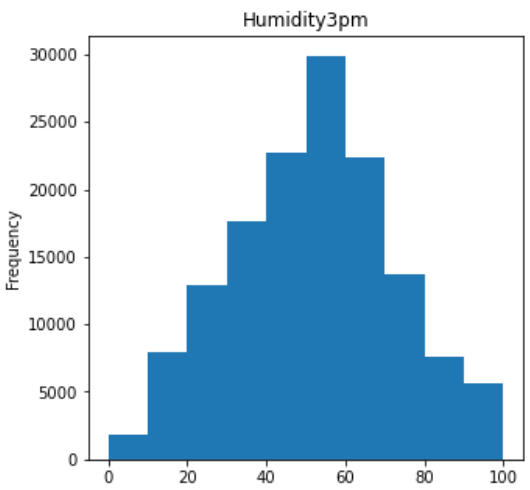

WindDir9am
0
2000
4000
6000
8000
10000
N
Unknown
SE
E
SSE
NW
S
W
SW
NNE

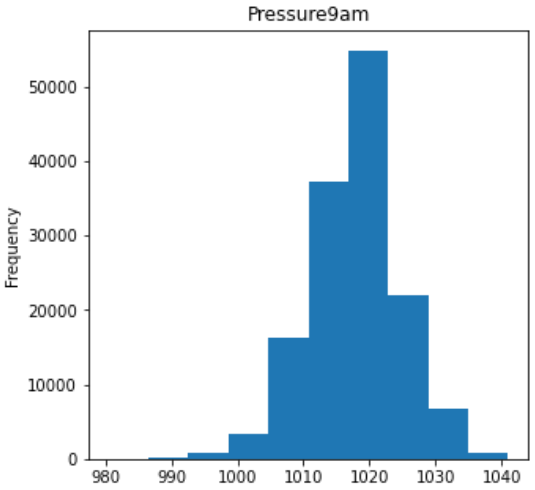

WindDir3pm
0
2000
4000
6000
8000
10000
SE
W
S
WSW
SW
SSE
N
WNW
NW
ESE

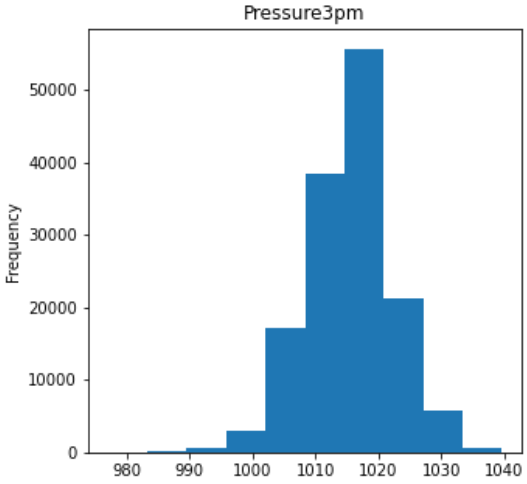

WindSpeed9am
Frequency
0
10000
20000
30000
40000
50000
60000
0
20
40
60
80
100
120

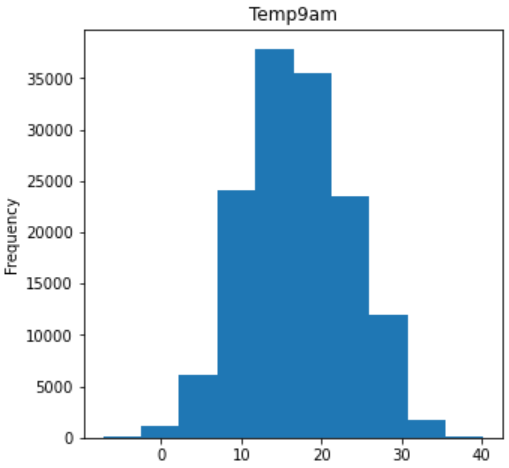

WindSpeed3pm
Frequency
0
10000
20000
30000
40000
50000
0
20
40
60
80

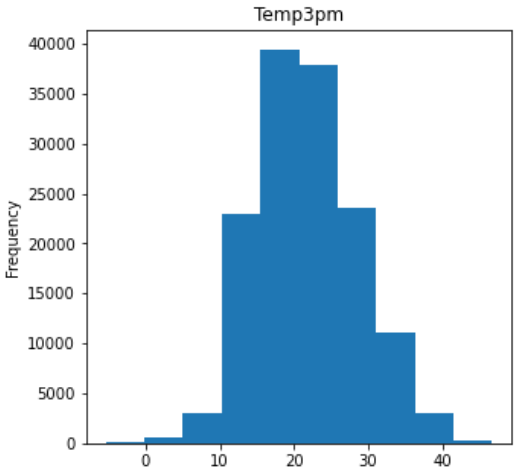

Humidity9am
Frequency
0
5000
10000
15000
20000
25000
30000
0
20
40
60
80
100

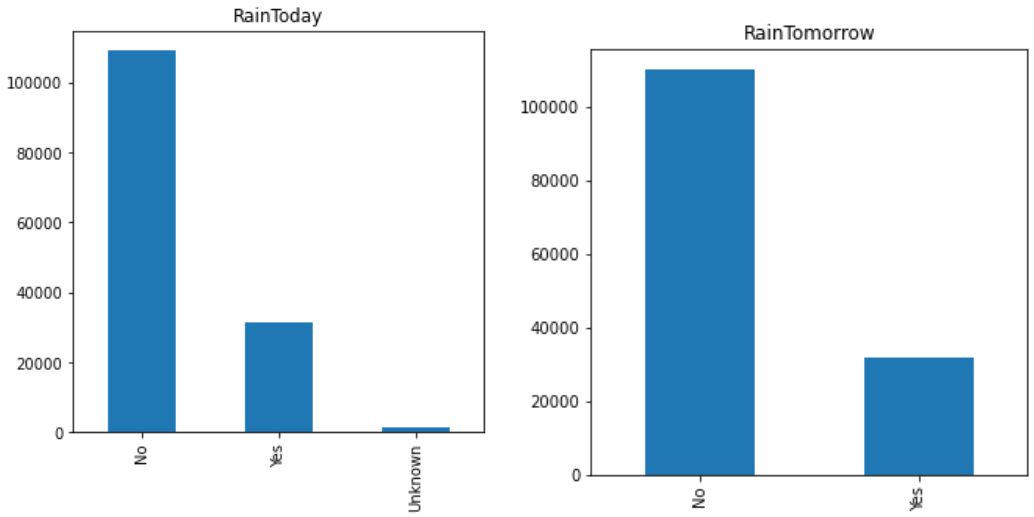

Humidity3pm
Frequency
30000
25000
20000
15000
10000
5000
0
0
20
40
60
80
100


Pressure9am
Frequency
50000
40000
30000
20000
10000
0
980
990
1000
1010
1020
1030
1040


Pressure3pm
Frequency
50000
40000
30000
20000
10000
0
980
990
1000
1010
1020
1030
1040


Temp9am
Frequency
35000
30000
25000
20000
15000
10000
5000
0
0
10
20
30
40

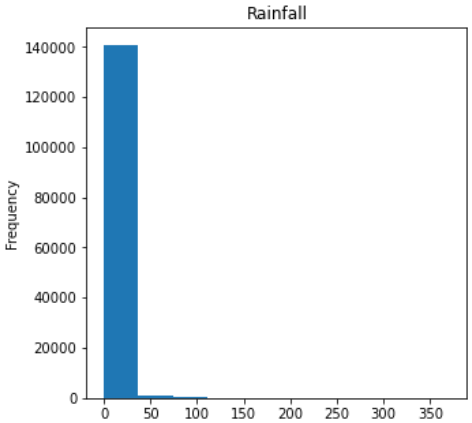

Temp3pm
Frequency
40000
35000
30000
25000
20000
15000
10000
5000
0
0
10
20
30
40



(Figura 19: Histograma y gráfico de barra Fuente: elaboración propia)

- **Abordar valores atípicos**

Como lo que muestra en el gráfico(Figura 20), por ejemplo, Rainfall tiene una distribución muy sesgada hacia la derecha, lo que indica que hay al menos un registro significativamente alto.



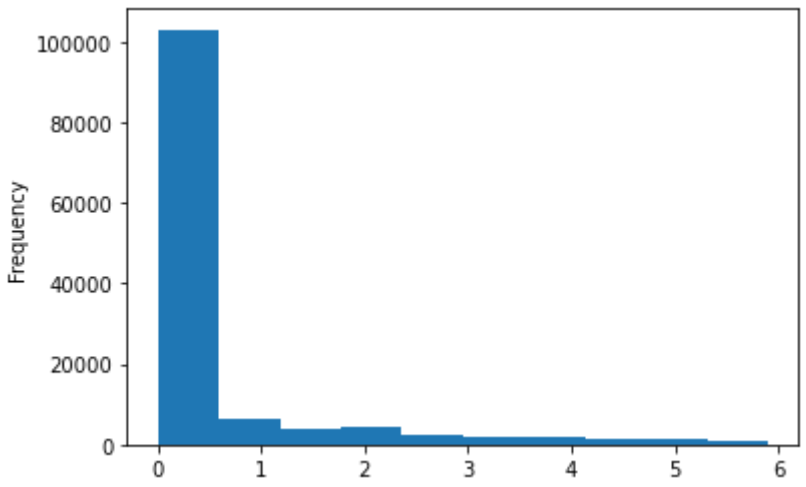
(Figura 20: Valor atípico Fuente: elaboración propia)

Para eliminar los valores atípicos(Figura 20), se utiliza el cuantil (0,9) para limitar el conjunto de datos a aquellos que caen en el cuantil del 90 % del conjunto de datos. Como resultado, el límite superior de los valores de lluvia se redujo significativamente de 350 a 6.

```
In [13]: #1) Address Outliers

         #address outlier in "Rainfall"

         maximum = df['Rainfall'].quantile(0.9)
         df=df[df["Rainfall"]<maximum]
         df["Rainfall"].plot(kind='hist')
         df.shape
```

(Figura 21: Resolver el valor atípico Fuente: elaboración propia)

- Este código (Figura 21)agrega una nueva columna "Month" al dataframe df, convirtiendo la columna "Date" a un objeto datetime, extrayendo el mes y convirtiéndolo en una cadena.
- Luego, cuenta cuántas Este código elimina los valores atípicos en la columna "Rainfall" del conjunto de datos.
- Primero, el valor máximo para la columna se calcula utilizando el método quantile(0.9), que devuelve el valor del percentil 90. Los valores por encima de este percentil se consideran valores atípicos y se eliminan del conjunto de datos utilizando el operador de comparación <.
- Luego, el histograma de la columna se traza para verificar la distribución de los datos después de la eliminación de valores atípicos.
- Finalmente, el número de filas y columnas del conjunto de datos se devuelve utilizando el método shape.

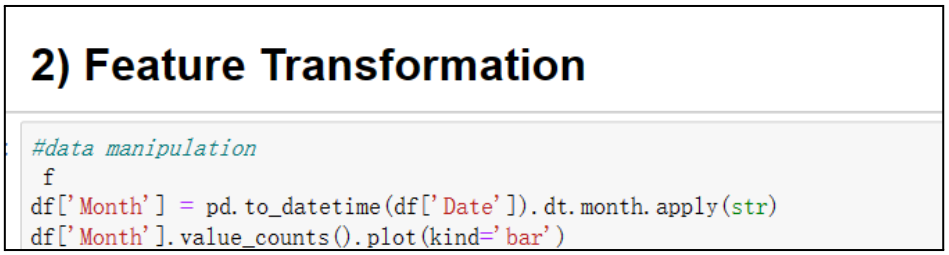


(Figura 22: Resultado de resolver el valor atípico Fuente: elaboración propia)

- **La transformación de características**

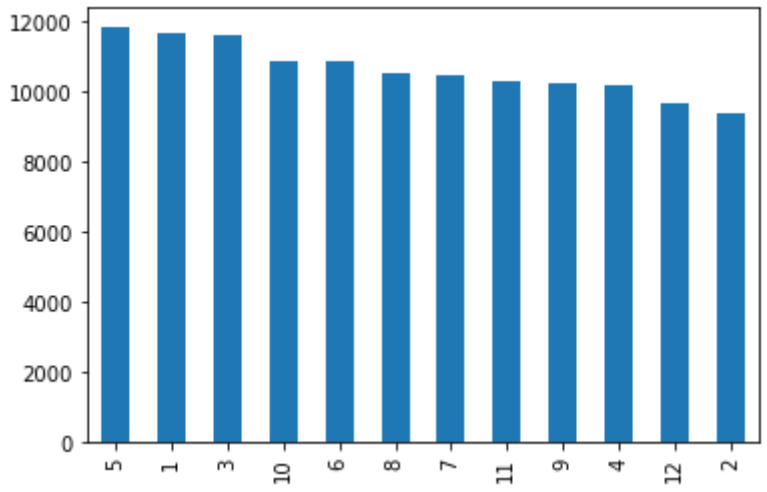
La transformación de características, también conocida como ingeniería de características, es el proceso de transformar los datos crudos en un formato que sea más fácil de entender y que tenga un mayor poder predictivo. Esto implica la selección de características relevantes y la creación de nuevas características a partir de las existentes.

En este caso(Figura 23), se transforma 'Month'(Mes) a 'Date'(Fecha).Esto se debe a que Fecha tiene una cardinalidad tan alta que hace que sea imposible resaltar patrones. Mientras que el uso de month puede dar sugerencias sobre si es más probable que llueva en ciertos meses del año.



(Figura 23: Transformación de 'Month'(Mes) a 'Date'(Fecha) Fuente: elaboración propia)

Este código(Figura 24) agrega una nueva columna "Month" al dataframe df, convirtiendo la columna "Date" a un objeto datetime, extrayendo el mes y convirtiéndolo en una cadena. Luego, cuenta cuántas veces aparece cada mes y muestra el resultado en un gráfico de barras.



(Figura 24: Gráfico de barras de frecuencia de lluvia Fuente: elaboración propia)

La codificación de características categóricas

La codificación de características categóricas se refiere al proceso de convertir características categóricas en una forma que pueda ser utilizada por los algoritmos de aprendizaje automático. Las características categóricas son aquellas que representan una categoría o una etiqueta, como el género (masculino/femenino), el estado civil (soltero/casado/divorciado), la ciudad de origen, etc.

La mayoría de los algoritmos de aprendizaje automático requieren que los datos de entrada sean numéricos. Por lo tanto, las características categóricas deben ser convertidas a valores numéricos para poder ser utilizadas por estos algoritmos.

En este caso, el código (Figura 25)está trabajando con un conjunto de datos que tiene características categóricas (por ejemplo, la ubicación, la dirección del viento, si llovió hoy o no, etc.). Para utilizar estos datos en modelos de aprendizaje automático, se necesita una forma de codificarlos numéricamente. Esto se puede hacer usando la técnica de encoding categórico, que transforma cada valor categórico en una serie de valores numéricos.

3) Categorical Feature Encoding

```
# encoding categorical data using dummies

from sklearn.preprocessing import LabelEncoder

categorical_features=['Location','WindGustDir','WindDir9am','WindDir3pm','RainToday','Month','RainTomorrow']

for i in categorical_features:
    df[i]=LabelEncoder().fit_transform(df[i])
```

(Figura 25: La codificación de características categóricas Fuente: elaboración propia)

- En este código(Figura 26), se está utilizando la función **LabelEncoder()** de Scikit-Learn para codificar cada valor categórico. Se está aplicando esta función a cada una de las características categóricas en la lista **"categorical_features"**. Después de aplicar la función **LabelEncoder()**, las características categóricas se transformarán en una serie de valores numéricos que se pueden utilizar para el modelado de datos.

```
In [19]: df.info()

<class 'pandas.core.frame.DataFrame'>
Int64Index: 127798 entries, 0 to 145458
Data columns (total 20 columns):
 #   Column         Non-Null Count   Dtype
---  ------         --------------   -----
 0   Date           127798 non-null  object
 1   Location       127798 non-null  int64
 2   MinTemp        127798 non-null  float64
 3   MaxTemp        127798 non-null  float64
 4   Rainfall       127798 non-null  float64
 5   WindGustDir    127798 non-null  int32
 6   WindGustSpeed  127798 non-null  float64
 7   WindDir9am     127798 non-null  int32
 8   WindDir3pm     127798 non-null  int32
 9   WindSpeed9am   127798 non-null  float64
 10  WindSpeed3pm   127798 non-null  float64
 11  Humidity9am    127798 non-null  float64
 12  Humidity3pm    127798 non-null  float64
 13  Pressure9am    127798 non-null  float64
 14  Pressure3pm    127798 non-null  float64
 15  Temp9am        127798 non-null  float64
 16  Temp3pm        127798 non-null  float64
 17  RainToday      127798 non-null  int32
 18  RainTomorrow   127798 non-null  int32
 19  Month          127798 non-null  int32
dtypes: float64(12), int32(6), int64(1), object(1)
memory usage: 17.6+ MB
```

(Figura 26:Información de los datos Fuente: elaboración propia)

La selección de características

La selección de características (también conocida como selección de variables) **es un proceso en el aprendizaje automático que se utiliza para seleccionar el subconjunto más relevante de características (variables, atributos) para ser utilizadas en el modelo predictivo**. El objetivo de la selección de características es mejorar la precisión del modelo, reducir el tiempo de entrenamiento y aumentar la comprensión del modelo al **eliminar características irrelevantes o redundantes**.

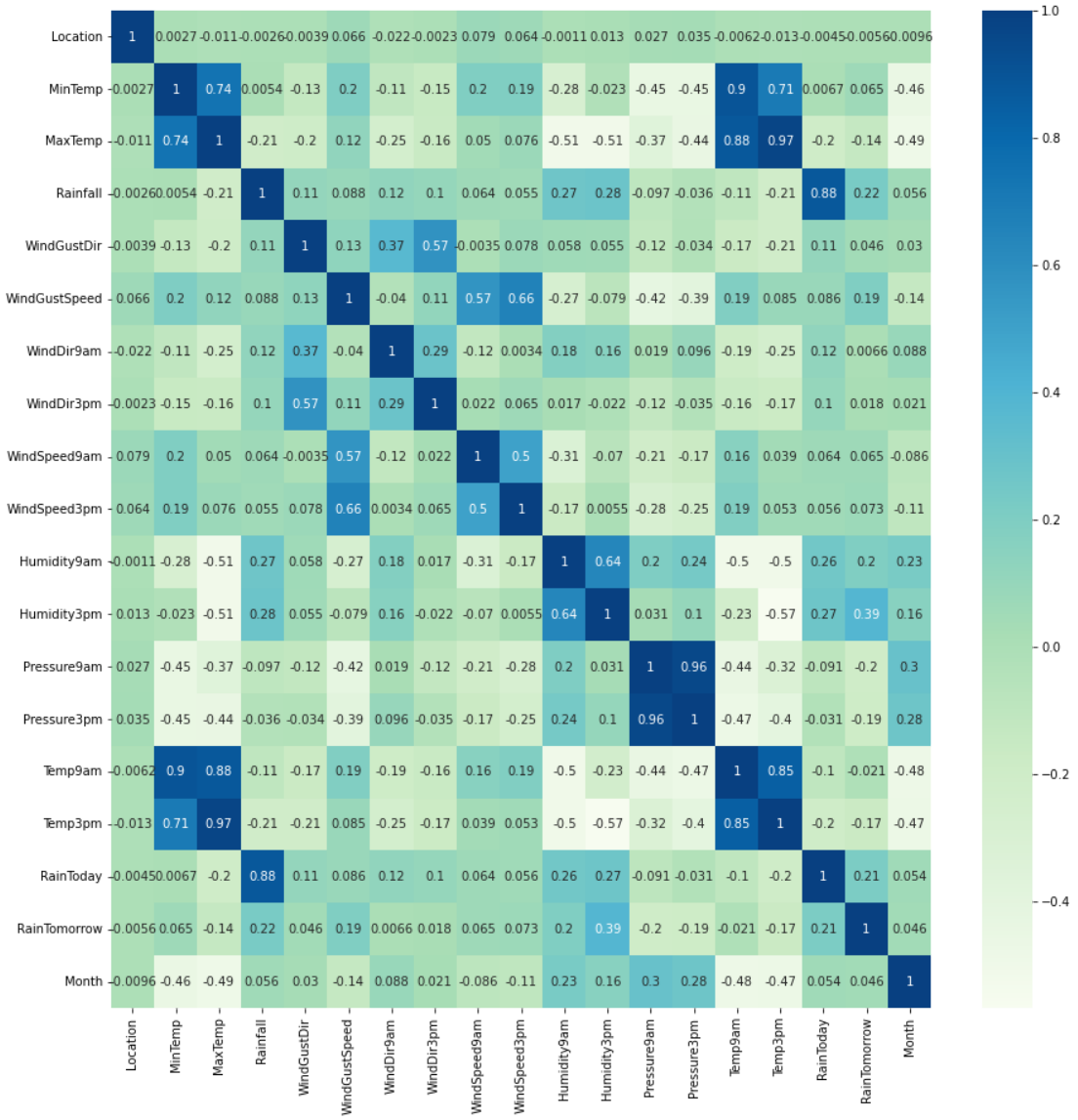
4) Feature Selection

```
# correlation analysis
plt.figure(1,figsize=(15,15))
correlation =df.corr()
sns.heatmap(correlation, cmap='GnBu',annot=True)
```

(Figura 27:Crear un heatmap de correlación de las variables Fuente: elaboración propia)

- Este código (Figura 27)genera una matriz de correlación entre todas las variables numéricas del conjunto de datos. La matriz de correlación muestra cómo cada variable numérica se correlaciona con todas las demás variables numéricas en el conjunto de datos.La función **plt.figure()** se utiliza para crear una figura con un tamaño de 15x15 pulgadas.
- La matriz de correlación se calcula utilizando el método **corr()** de Pandas en el conjunto de datos "df".

La biblioteca Seaborn se utiliza para crear una representación visual de la matriz de correlación. La función **sns.heatmap()** traza una matriz de colores en la que cada celda representa el nivel de correlación entre dos variables. Los valores altos están representados en azul oscuro y los valores bajos en azul claro. Además, la función annot=True agrega el valor numérico de la correlación en cada celda.



(Figura 28:Heatmap de correlación de las variables Fuente: elaboración propia)

Se nota que hay variables que tienen una alta correlación(Figura 28):

MinTemp, MaxTemp, Temp9am and Temp3pm

RainFall and RainToday

Pressure9am and Pressure3am

Por eso hay que solo mantener un variable de alta correlación.

```
df = df[['Month', 'Location', 'MinTemp', 'MaxTemp', 'WindGustDir', 'WindGustSpeed', 'WindDir9am', 'WindDir3pm', 'WindSpeed9am', 'WindSpeed
```

(Figura 29: Reorganiza el dataframe df Fuente: elaboración propia)

- Este código reorganiza el dataframe **df** para incluir solamente las siguientes columnas en el orden dado. **Dado que la regresión logística requiere que haya poca multicolinealidad entre los predictores, se mantiene solo una variable en cada grupo de variables altamente correlacionadas.**

Paso 3: Establecer un modelo

3. Model Building

```
X=df.iloc[:,:-1]
y=df["RainTomorrow"]
```

```
# split into train and test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42)
print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
```

```
(85624, 14) (42174, 14) (85624,) (42174,)
```

(Figura 30: Reorganiza el dataframe df Fuente: elaboración propia)

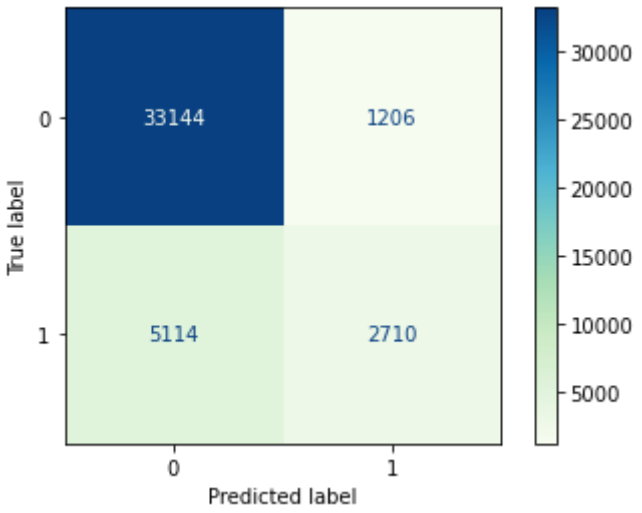
- El código anterior (Figura 30) divide los datos en conjuntos de entrenamiento y prueba utilizando la función **train_test_split** de la biblioteca Scikit-learn. Los argumentos X y y son las variables de características y la variable objetivo respectivamente. **test_size** indica la proporción de los datos que se deben utilizar como conjunto de prueba, en este caso, el 33% se asigna a los datos de prueba y el 67% restante se usa para entrenar el modelo. **random_state** es una semilla aleatoria para garantizar que la división de los datos sea reproducible. El código imprime las formas de los conjuntos de entrenamiento y prueba (X_train.shape, X_test.shape, y_train.shape y y_test.shape) para confirmar que los datos se han dividido correctamente

```
from sklearn import metrics

#confusion matrix
confusion_matrix=metrics.plot_confusion_matrix(reg, X_test, y_test, cmap='GnBu')
print(confusion_matrix)
```

(Figura 31: Crear la matriz de confusión df Fuente: elaboración propia)

- Para usar la función metrics.**plot_confusion_matrix()**(Figura 32), se necesita tener un modelo ya entrenado y una serie de datos de prueba **X_test** e **y_test**. La función muestra la matriz de confusión del modelo en forma de gráfico. La matriz de confusión es una herramienta para evaluar el rendimiento de un modelo de clasificación, que muestra **la frecuencia de predicciones correctas e incorrectas**. La diagonal principal de la matriz de confusión muestra las predicciones correctas, mientras que las celdas fuera de la diagonal principal muestran las predicciones incorrectas. La intensidad del color de cada celda del gráfico indica el número de observaciones en esa celda.

Verdadero positivo: llueve mañana cuando se pronostica que lloverá
Verdadero negativo: no llueve mañana cuando se predijo que no llovería
Falso positivo: no llueve mañana cuando se pronostica que lloverá
Falso negativo: llueve cuando se pronostica que no lloverá



(Figura 32: La matriz de confusión Fuente: elaboración propia)

Como se muestra, el verdadero negativo es 33122 casos, lo que sugiere que el modelo es bueno para predecir que mañana no lloverá cuando en realidad no va a llover. Sin embargo, todavía necesita mejorar la tasa de verdaderos positivos, por lo tanto, predecir con éxito la lluvia mañana (solo 2756 casos).(Figura 33)

```
#accuracy
print("Accuracy",metrics.accuracy_score(y_test, y_pred))

Accuracy 0.8501446388770333
```

(Figura 33:Precisión del modelo Fuente: elaboración propia)

La curva **ROC** (Receiver Operating Characteristic) es una herramienta que se utiliza para evaluar la capacidad de un modelo de clasificación binario para distinguir entre dos clases (positiva y negativa). Representa la tasa de verdaderos positivos (sensibilidad) en el eje y y la tasa de falsos positivos (1 - especificidad) en el eje x.

La **AUC** (Área bajo la curva ROC) es una medida de la capacidad de un modelo para distinguir entre las clases positiva y negativa. Un modelo con un AUC de 1 es un modelo perfecto que puede distinguir con precisión entre las dos clases en todas las instancias. Un modelo con un AUC de 0,5 es básicamente un modelo aleatorio y no tiene capacidad de discriminación. Por lo tanto, cuanto mayor sea el valor de AUC, mejor será el modelo.

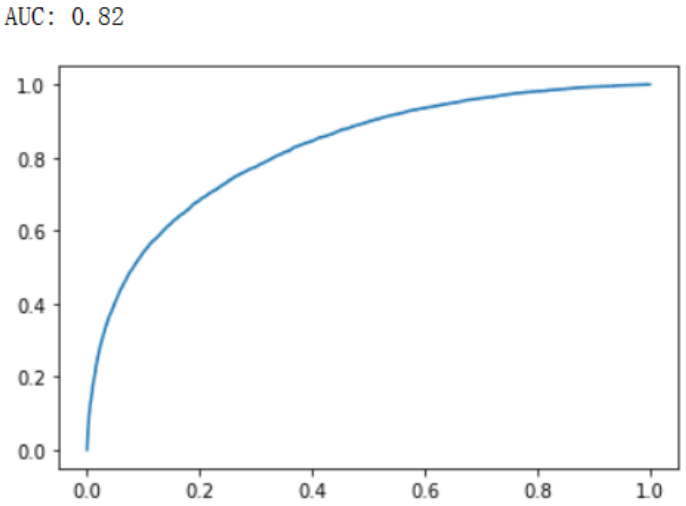
- Si **y_pred** es la variable que contiene las predicciones del modelo, este código está correcto para obtener la exactitud (accuracy) del modelo. La función **accuracy_score** de sklearn.metrics toma como argumentos el conjunto de etiquetas verdaderas **(y_test)** y el conjunto de etiquetas predichas **(y_pred)** y devuelve la exactitud.
- La exactitud es la proporción de predicciones correctas en relación al total de predicciones.

```
In [41]: #ROC curve and AUC

         y_pred_prob = reg.predict_proba(X_test)[:, 1]
         fpr, tpr, threshold=metrics.roc_curve(y_test,  y_pred_prob)
         plt.plot(fpr, tpr)

         auc = metrics.roc_auc_score(y_test, y_pred_prob)
         print("AUC:",round(auc, 2))
```

(Figura 34:Crear la curva de ROC y AUC Fuente: elaboración propia)

- En el código que has proporcionado, se ha utilizado metrics.roc_curve() para obtener la tasa de falsos positivos (fpr), la tasa de verdaderos positivos (tpr) y los umbrales, y se ha trazado la curva **ROC** utilizando **plt.plot()**. Luego, se ha calculado la **AUC** utilizando **metrics.roc_auc_score()** y se ha impreso el

resultado en la consola mediante el uso de print().



(Figura 35:La curva de ROC y AUC Fuente: elaboración propia)

Como se mencionó antes, un modelo con un AUC de 1 es un modelo perfecto que puede distinguir con precisión entre las dos clases en todas las instancias. En este caso(Figura 35), el resultado es 0.82, está cerca de 1, la evaluación de este modelo es adecuada.

5.3 Decision Tree

Definición de Decision Tree:

Un árbol de decisión puede dividir los datos en varios conjuntos y, estos conjuntos se dividen en más subconjuntos aumentando las preguntas para llegar a una decisión final.

Una de las ventajas más significativas que ofrecen los árboles de decisión es la capacidad de explicar las decisiones o la clasificación que se hizo con pruebas y ejemplos concretos.

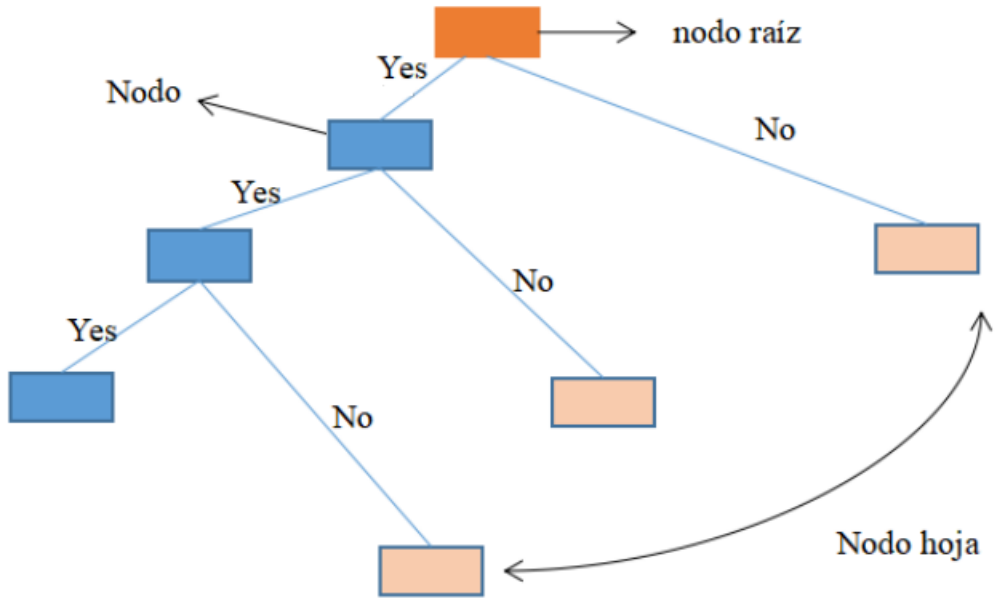
Aplicación de árbol de decisión en empresas:

Planificación de la producción: La decisión de árbol puede ser utilizada en la planificación de la producción según los existentes rescurso para optimizar los procesos y tomar decisiones eficientes. Los árboles de decisión pueden ayudar a determinar la secuencia de producción, la asignación de recursos y la planificación de la capacidad.

Segmentación de clientes: los árboles de decisión se pueden utilizar para segmentar a los clientes en diferentes grupos con características similares. La segmentación de clientes es una estrategia común en marketing que permite adaptar las estrategias y mensajes específicamente a cada grupo, lo que puede mejorar la efectividad de las acciones de marketing y la satisfacción del cliente.

Construcción del árbol de decisión:

La construcción del árbol de decisión es el proceso de dividir el conjunto de datos en múltiples subconjuntos seleccionando recursivamente las características óptimas. La clasificación empieza desde el nodo raíz, cada nodo interno representa una propiedad o característica, cada rama representa una regla de decisión y cada nodo hoja representa una clase o resultado.(Figura 36)



(Figura 36:Modelo de árbol de decisión Fuente: elaboración propia)

Clasificación del árbol de decisión:

Los métodos comunes de selección de características incluyen **la ganancia de información, la relación de ganancia de información y el índice de Gini**.

Ejemplo de Decision Tree

Paso 1: Importar los datos

Primero, se introducen los datos y se echa un vistazo de los datos. (Figura 37)

Loading Data

```
# Load libraries
import pandas as pd
from sklearn.tree import DecisionTreeClassifier # Import Decision Tree Classifier
from sklearn.model_selection import train_test_split # Import train_test_split function
from sklearn import metrics #Import scikit-learn metrics module for accuracy calculation
```

```
col_names = ['pregnant', 'glucose', 'bp', 'skin', 'insulin', 'bmi', 'pedigree', 'age', 'label']
# load dataset
pima = pd.read_csv("diabetes.csv", header=None, names=col_names)
```

```
pima.head()
```

	pregnant	glucose	bp	skin	insulin	bmi	pedigree	age	label
0	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
1	6	148	72	35	0	33.6	0.627	50	1
2	1	85	66	29	0	26.6	0.351	31	0
3	8	183	64	0	0	23.3	0.672	32	1
4	1	89	66	23	94	28.1	0.167	21	0

(Figura 37: procesar los datos Fuente: elaboración propia)

- Este código carga las bibliotecas necesarias para implementar un clasificador de árbol de decisiones en Python utilizando scikit-learn.
- El clasificador de árbol de decisiones es una técnica de aprendizaje supervisado utilizada para resolver problemas de clasificación y regresión.
- El código también importa la función train_test_split que divide un conjunto de datos en conjuntos de entrenamiento y prueba para evaluar la precisión del modelo.
- La biblioteca metrics de scikit-learn se utiliza para calcular la precisión del modelo.

Paso 2: La selección de características

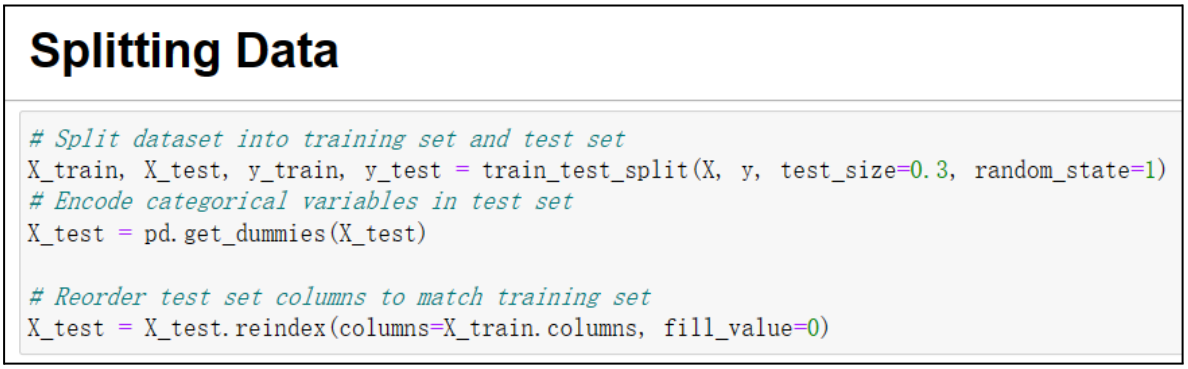
La selección de características (también conocida como selección de variables) **es un proceso en el aprendizaje automático que se utiliza para seleccionar el subconjunto más relevante de características (variables, atributos) para ser utilizadas en el modelo predictivo**. El objetivo de la selección de características es mejorar la precisión del modelo, reducir el tiempo de entrenamiento y aumentar la

comprensión del modelo al eliminar características irrelevantes o redundantes.

(Figura 38: Selección de características Fuente: elaboración propia)

- En este código, se dividen las variables del conjunto de datos en variables de características y variables de destino.
- La lista feature_cols contiene los nombres de las columnas que se utilizarán como características en el modelo. En este caso, se consideran las características más relevantes son 'pregnant', 'glucose', 'bp', 'skin', 'insulin', 'bmi', 'pedigree'.
- Estas características se seleccionan del conjunto de datos pima y se asignan a la variable X.
- La variable y contiene los valores de destino que el modelo intentará predecir. En este caso, la columna label del conjunto de datos pima se asigna a y.

Paso3: Dividir los datos



(Figura 39: Dividir los datos a dos partes: entrenamiento y prueba Fuente: elaboración propia)

- Este código (Figura 38)divide el conjunto de datos en un conjunto de entrenamiento y un conjunto de prueba utilizando la función train_test_split de la biblioteca sklearn.model_selection. El conjunto de entrenamiento se usará

para entrenar el modelo y el conjunto de prueba se usará para evaluar el rendimiento del modelo. La división se realiza utilizando una proporción de 70% para el conjunto de entrenamiento y 30% para el conjunto de prueba.

- A continuación, se codifica las variables categóricas en el conjunto de prueba utilizando la función get_dummies de la biblioteca pandas. Esta función convierte las variables categóricas en variables ficticias para que puedan ser utilizadas en el modelo.
- Después, se reordenan las columnas del conjunto de prueba para que coincidan con las del conjunto de entrenamiento utilizando la función reindex de pandas. Se establece fill_value=0 para rellenar las columnas que puedan estar presentes en el conjunto de entrenamiento pero no en el conjunto de prueba con ceros.

Paso 4:Establecer un modelo de árbol de decisión

Building Decision Tree Model

```
# Create Decision Tree classifer object
clf = DecisionTreeClassifier()

# Train Decision Tree Classifer
clf = clf.fit(X_train, y_train)

#Predict the response for test dataset
y_pred = clf.predict(X_test)
```

(Figura 40: Crear el árbol de decisión Fuente: elaboración propia)

- El código(Figura 40) anterior corresponde a la implementación de un modelo de árbol de decisión en Python utilizando la biblioteca Scikit-learn. Primero, se crea un objeto de clasificador de árbol de decisión vacío con la línea **"clf = DecisionTreeClassifier()"**.
- Luego, se entrena el clasificador utilizando el conjunto de entrenamiento con la línea **"clf = clf.fit(X_train,y_train)"**, donde **X_train** es el conjunto de características y y_train es el conjunto de etiquetas correspondientes. Finalmente, se realiza la predicción en el conjunto de prueba utilizando la línea **"y_pred = clf.predict(X_test)"**, y se obtiene una matriz de predicciones de clasificación para el conjunto de prueba.

Paso5: Evaluar el modelo

(Figura 41: Evaluar el árbol de decisión Fuente: elaboración propia)

- Este códig(Figura 41)o calcula la precisión del modelo clasificador. Compara las etiquetas reales de la variable objetivo en el conjunto de prueba con las etiquetas predichas por el modelo y devuelve la precisión del modelo como un valor de puntuación. En otras palabras, determina con qué frecuencia el modelo clasificador predice correctamente la etiqueta de la variable objetivo.

```
Accuracy: 0.6493506493506493
```

(Figura 42: La precisión del árbol de decisión Fuente: elaboración propia)

Tiene una tasa de clasificación del 67,53%, que se considera una buena precisión. Puede mejorar esta precisión ajustando los parámetros en el algoritmo del árbol de decisiones.

Paso6: Visualizar el árbol de decisión

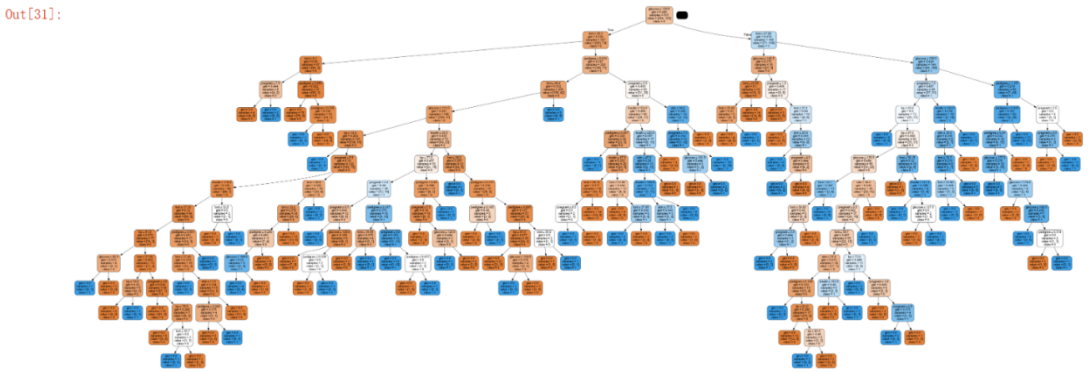
La visualización (Figura 43,44)resultante muestra el árbol de decisión y las divisiones que se han hecho en las diferentes características para clasificar las muestras en las diferentes clases.

Visualizing Decision Trees

```
import os
os.environ["PATH"] += os.pathsep + 'C:/Program Files (x86)/Graphviz/bin/'

from sklearn.tree import export_graphviz
from IPython.display import Image
import pydotplus
from io import StringIO

dot_data = StringIO()
export_graphviz(clf, out_file=dot_data,
                filled=True, rounded=True,
                special_characters=True,feature_names = feature_cols,class_names=['0','1'])
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
graph.write_png('diabetes.png')
Image(graph.create_png())
```

(Figura 43:Visualizar el árbol de decisión Fuente: elaboración propia)

- Este código es una forma de visualizar el árbol de decisión generado por el modelo de clasificación de árbol de decisión. Utiliza la biblioteca Graphviz para crear una representación visual del árbol de decisión y la biblioteca pydotplus para mostrar la imagen en formato png.
- Primero, se agrega la ruta al archivo binario de Graphviz para que se pueda usar para crear la imagen. Luego, se importan las bibliotecas necesarias y se crea un objeto StringIO que se utilizará para almacenar los datos del gráfico. Se llama a la función export_graphviz y se le pasa el modelo de árbol de decisión entrenado y los nombres de las características y clases. La función crea un archivo dot que contiene los datos del gráfico.
- Luego, se utiliza la biblioteca pydotplus para leer los datos del gráfico del archivo dot y crear una imagen png. Finalmente, se muestra la imagen en el notebook de Jupyter utilizando la biblioteca IPython.display.

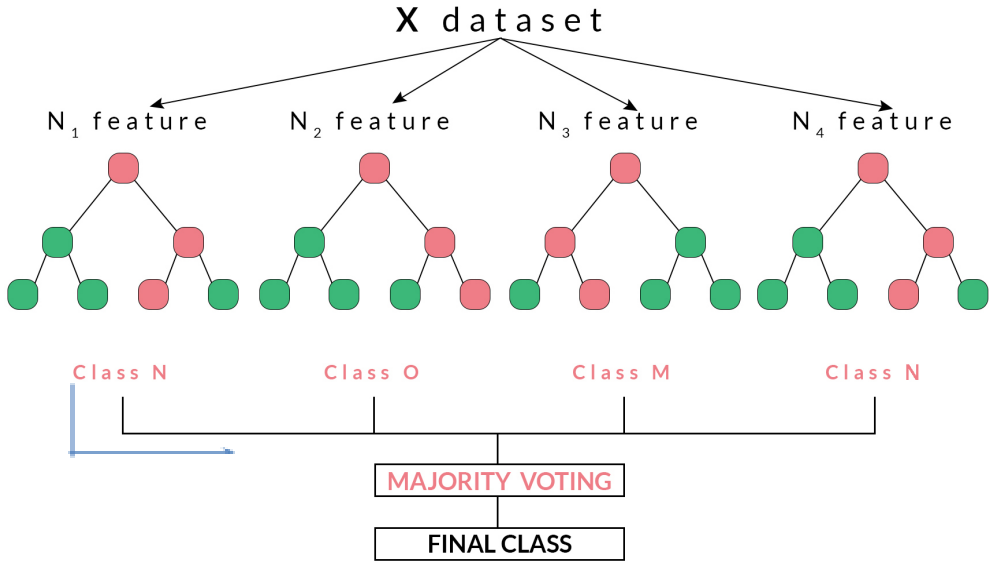


(Figura 44:El árbol de decisión Fuente: elaboración propia)

5.4 Bosque aleatorio

Bosque aleatorio(Figura 45) es una combinación de múltiples árboles de decisión aleatorios. Como se mencionó antes los modelos de árbol de decisión son uno de los algoritmos de aprendizaje supervisado más famosos en el aprendizaje automático, tanto para propósitos de regresión como de clasificación.

Tener múltiples árboles de decisión garantiza la explicabilidad del modelo y la facilidad para interpretar un modelo de aprendizaje automático.



(Figura 45:Bosque aleatorio Fuente:Sasiwut Chaiyadecha)

Hay dos formas de crear aleatoriedad en un random forest:

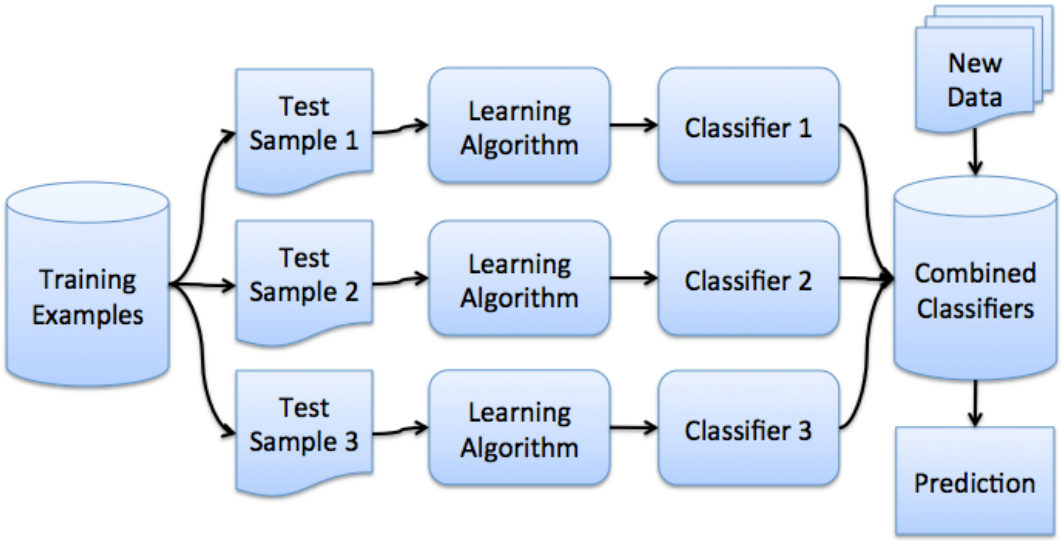
- Cada árbol se construye utilizando una muestra elegida al azar tomada de nuestro conjunto de datos de entrenamiento.
- En cada nodo de decisión, el árbol se divide aún más en función de una serie de atributos elegidos al azar.

La técnica de bagging

Bagging es un tipo de aprendizaje en conjunto que ayuda a reducir la varianza de diferentes tipos de modelos. Bagging significa agregación bootstrap. Sabemos que los bosques aleatorios utilizan un conjunto de árboles de decisión, cada uno de los cuales produce una salida y, por lo tanto, puede considerarse como aprendices que cometen errores independientes. Para asegurarnos de que los aprendices cometan errores independientes, podemos proporcionarles diferentes muestras de datos, diferentes parámetros de algoritmo, etc.

En bootstrap aggregation/bagging, se seleccionan los conjuntos de muestras de tal manera que sean diferentes entre sí.

El proceso de bagging funciona de la siguiente manera(Figura 46):



(Figura 46: Método de bagging Fuente:Custom Ensemble approach - vision - PyTorch Forums)

Supongamos que tenemos un conjunto de datos de entrenamiento T.

Si tenemos n árboles de decisión/aprendices, podemos seleccionar aleatoriamente un tamaño de muestra k (este tamaño de muestra debe ser lo suficientemente grande para

evitar un sobreajuste excesivo).
Ahora repetidamente elegiremos k elementos del conjunto de datos de entrenamiento, con reemplazo, para crear los conjuntos de entrenamiento T1, T2, T3, ..., hasta que estén disponibles los conjuntos de entrenamiento Tn.
El promedio de todos los árboles se considerará como la predicción final.
Debido a la diferencia en los subconjuntos de muestras, los árboles individuales no estarán correlacionados entre sí y podrán ofrecer ideas únicas sobre los patrones de los datos.

Aplicación de bosque aleteatorio en empresa

Modelo de pronóstico: los árboles de decisión se pueden usar para construir modelos de pronóstico, como pronóstico de ventas, pronóstico de demanda del mercado, pronóstico de comportamiento del usuario, etc. Mediante el análisis de diferentes características y caminos de decisión, los árboles de decisión pueden predecir resultados o clasificaciones futuras.

Clasificación y segmentación de clientes: El bosque aleatorio puede utilizarse para clasificar clientes en diferentes segmentos en función de sus características y comportamientos. Esto permite a las empresas personalizar sus estrategias de marketing y adaptar sus mensajes y ofertas a cada segmento de manera más efectiva.

Recomendación de productos: Los árboles de decisión se pueden utilizar para recomendaciones personalizadas de productos.De acuerdo con las características del usuario y el comportamiento histórico, se construye un modelo de árbol de decisión para recomendar productos que mejor satisfagan los intereses y necesidades del usuario.

Ejemplo de Bosque aleatorio

Se predice la temperatura máxima para 2017 en Seattle utilizando un año de datos meteorológicos de 2016.

Paso 1: preparar los datos

Se introducen los datos y se echa un vistazo de los datos(Figura 47):

Data Preparation

```
# Pandas is used for data manipulation
import pandas as pd

# Read in data as pandas dataframe and display first 5 rows
features = pd.read_csv('temps.csv')
features.head(5)
```

	year	month	day	week	temp_2	temp_1	average	actual	friend
0	2016	1	1	Fri	45	45	45.6	45	29
1	2016	1	2	Sat	44	45	45.7	44	61
2	2016	1	3	Sun	45	44	45.8	41	56
3	2016	1	4	Mon	44	41	45.9	40	53
4	2016	1	5	Tues	41	40	46.0	44	41

```
print('The shape of our features is:', features.shape)
```

```
The shape of our features is: (348, 9)
```

(Figura 47:Procesar los datos Fuente: elaboración propia)

Paso2: codificación

La codificación de características categóricas se refiere al proceso de convertir características categóricas en una forma que pueda ser utilizada por los algoritmos de aprendizaje automático.

En este caso(Figura 48), **se convierte la semana a un formato numérico** para que la máquina lo entienda mejor.

One-Hot Encoding

One hot encoding takes this:

week
Mon
Tue
Wed
Thu
Fri

and converts it into:

Mon	Tue	Wed	Thu	Fri
1	0	0	0	0
0	1	0	0	0
0	0	1	0	0
0	0	0	1	0
0	0	0	0	1

(Figura 48: Explicación de codificación Fuente: elaboración propia)

- Cada valor distinto se convierte en una nueva columna, y para cada muestra se etiqueta el valor al que pertenece 1 en la columna correspondiente y 0 en el resto. El propósito de esto es permitir que los algoritmos de aprendizaje automático comprendan y procesen mejor los datos de características discretas, como el día de la semana y etc.

```
# One-hot encode categorical features
features = pd.get_dummies(features)
features.head(5)
```

	year	month	day	temp_2	temp_1	average	actual	friend	week_Fri	week_Mon	week_Sat	week_Sun	week_Thurs
0	2016	1	1	45	45	45.6	45	29	1	0	0	0	0
1	2016	1	2	44	45	45.7	44	61	0	0	1	0	0
2	2016	1	3	45	44	45.8	41	56	0	0	0	1	0
3	2016	1	4	44	41	45.9	40	53	0	1	0	0	0
4	2016	1	5	41	40	46.0	44	41	0	0	0	0	0

```
print('Shape of features after one-hot encoding:', features.shape)
```

```
Shape of features after one-hot encoding: (348, 15)
```

(Figura 49: codificación Fuente: elaboración propia)

Paso 3: Características y etiquetas

Se enfoca en la preparación de datos para su uso en modelos de aprendizaje automático.(Figura 50) Convierte un conjunto de datos de características y etiquetas en dos arreglos de NumPy separados, uno para las características y otro para las etiquetas.

Además, elimina la columna "actual" del conjunto de características y la guarda como una variable separada "etiquetas". Finalmente, guarda los nombres de las características en una lista llamada "feature_list".

Features and Labels

```
# Use numpy to convert to arrays
import numpy as np

# Labels are the values we want to predict
labels = np.array(features['actual'])

# Remove the labels from the features
# axis 1 refers to the columns
features= features.drop('actual', axis = 1)

# Saving feature names for later use
feature_list = list(features.columns)

# Convert to numpy array
features = np.array(features)
```

(Figura 50:Estrenar y probar los datos Fuente: elaboración propia)

- Se utiliza la librería Numpy para convertir datos a arreglos. Primero, se define un arreglo llamado "labels" que contiene los valores que se quieren predecir. Estos valores se obtienen a partir de una columna de la variable "features".
- Luego, se remueve la columna de los valores a predecir de la variable "features" usando el método "drop" y especificando que se quiere eliminar la columna llamada "actual" en el eje 1 (columnas).
- Después, se guarda el nombre de las columnas restantes en la variable "feature_list" para usarlas más adelante. Finalmente, la variable "features" se convierte a un arreglo de Numpy usando el método "array" de Numpy.

Paso 4: entrenamiento y prueba

Se utiliza la biblioteca Scikit-learn para dividir los datos en conjuntos de entrenamiento y prueba. (Figura 51)

- La función `train_test_split` toma como entrada dos matrices NumPy:

`features` y `labels`.

- Luego, divide estos datos en cuatro conjuntos de datos: `train_features`, `test_features`, `train_labels` y `test_labels`. La relación entre los conjuntos de datos de entrenamiento y prueba se establece con el argumento `test_size`, que se establece en 0,25 para que el conjunto de prueba sea el 25% del tamaño total de los datos. El argumento `random_state` se establece en 42 para que los resultados sean reproducibles.

Training and Testing Sets

```
# Using Skicit-learn to split data into training and testing sets
from sklearn.model_selection import train_test_split

# Split the data into training and testing sets
train_features, test_features, train_labels, test_labels = train_test_split(features, labels, test_size = 0.25,
                                                                            random_state = 42)
```

```
print('Training Features Shape:', train_features.shape)
print('Training Labels Shape:', train_labels.shape)
print('Testing Features Shape:', test_features.shape)
print('Testing Labels Shape:', test_labels.shape)
```

```
Training Features Shape: (261, 14)
Training Labels Shape: (261,)
Testing Features Shape: (87, 14)
Testing Labels Shape: (87,)
```

(Figura 51:Estrenar y probar los datos Fuente: elaboración propia)

Paso 5: Establecer línea de base

Se hacen predicciones basadas en el promedio histórico de los datos de prueba. (Figura 52)

Establish Baseline

```
# The baseline predictions are the historical averages
baseline_preds = test_features[:, feature_list.index('average')]

# Baseline errors, and display average baseline error
baseline_errors = abs(baseline_preds - test_labels)
print('Average baseline error: ', round(np.mean(baseline_errors), 2), 'degrees.')
```

```
Average baseline error:  5.06 degrees.
```

(Figura 52:Establecer la línea base Fuente: elaboración propia)

- ❖ Primero se define la variable `baseline_preds` que contiene las predicciones de la línea base, que en este caso es el promedio histórico de la característica "average" de los datos de prueba.
- ❖ Luego se calculan los errores de línea base comparando las predicciones de línea base con las etiquetas de prueba reales.
- ❖ Finalmente, se muestra el error promedio de la línea base.

Training the Forest

```
# Import the model we are using
from sklearn.ensemble import RandomForestRegressor

# Instantiate model
rf = RandomForestRegressor(n_estimators= 1000, random_state=42)

# Train the model on training data
rf.fit(train_features, train_labels);
```

(Figura 53:Estrenar el bosque Fuente: elaboración propia)

- Este código importa la clase `RandomForestRegressor` del módulo `sklearn.ensemble`, que se utiliza para implementar el algoritmo de Random Forest para regresión.
- Luego se crea una instancia de la clase `RandomForestRegressor` y se especifica que se usarán 1000 árboles (`n_estimators=1000`) y se establece el estado aleatorio en 42 (`random_state=42`).
- Finalmente, se ajusta el modelo a los datos de entrenamiento utilizando el método `fit()` de la clase `RandomForestRegressor`.

Paso6: Hacer predicciones en datos de prueba

Se implementa un modelo de Regresión Forestal Aleatoria (Random Forest Regressor) para predecir la temperatura máxima diaria. (Figura 54)

Make Predictions on Test Data

```
# Use the forest's predict method on the test data
predictions = rf.predict(test_features)

# Calculate the absolute errors
errors = abs(predictions - test_labels)

# Print out the mean absolute error (mae)
print('Mean Absolute Error:', round(np.mean(errors), 2), 'degrees.')
```

```
Mean Absolute Error: 3.83 degrees.
```

```
# Calculate mean absolute percentage error (MAPE)
mape = 100 * (errors / test_labels)

# Calculate and display accuracy
accuracy = 100 - np.mean(mape)
print('Accuracy:', round(accuracy, 2), '%.')
```

```
Accuracy: 93.99 %.
```

(Figura 54:Hacer la predicción en data de prueba base Fuente: elaboración propia)

- Primero, el código divide los datos en un conjunto de entrenamiento y un conjunto de la prueba. Luego, se entrena el modelo con el conjunto de entrenamiento y se ajustan los hiperparámetros del modelo mediante el uso de una validación cruzada.
- A continuación, se hacen predicciones utilizando el modelo entrenado en el conjunto de prueba y se calcula el error absoluto medio (MAE) para evaluar la precisión del modelo.
- Por último, se muestran los resultados de la predicción y el error absoluto medio.

```
# Calculate mean absolute percentage error (MAPE)
mape = 100 * (errors / test_labels)

# Calculate and display accuracy
accuracy = 100 - np.mean(mape)
print('Accuracy:', round(accuracy, 2), '%.')
```

```
Accuracy: 93.93 %.
```

(Figura 55:Calcular la precisión Fuente: elaboración propia)

Este código (Figura 55) calcula el error porcentual absoluto medio (MAPE) y luego calcula la precisión de las predicciones del modelo. La precisión se calcula restando el MAPE del 100%. Luego, se imprime la precisión en forma de porcentaje.

Paso 7: Visualizar el árbol de decisión

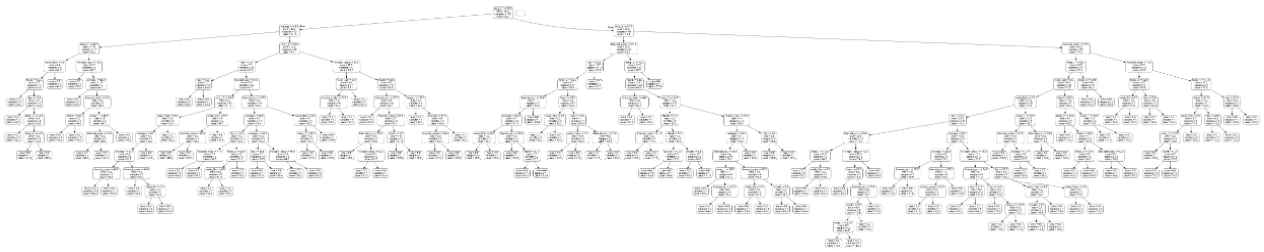
Visualizing a Single Decision Tree

```
# Import tools needed for visualization
import os
os.environ["PATH"] += os.pathsep + 'C:/Program Files (x86)/Graphviz/bin/'

!pip install pydot

from sklearn.tree import export_graphviz
import pydot

# Pull out one tree from the forest
tree = rf.estimators_[5]

# Export the image to a dot file
export_graphviz(tree, out_file = 'tree.dot', feature_names = feature_list, rounded = True, precision = 1)

# Use dot file to create a graph
(graph, ) = pydot.graph_from_dot_file('tree.dot')

# Write graph to a png file
graph.write_png('tree.png');
```

```
Requirement already satisfied: pydot in c:\programdata\anaconda3\lib\site-packages (1.4.2)
Requirement already satisfied: pyparsing>=2.1.4 in c:\programdata\anaconda3\lib\site-packages (from pydot)
```

(Figura 56:Visualizar el árbol de decisión Fuente: elaboración propia)

- Se importan las herramientas necesarias (Figura 56) para la visualización, incluyendo **Graphviz**, que es un software de visualización de gráficos.
- Se instala la librería **pydot** para poder trabajar con archivos .dot.
- A continuación, se selecciona uno de los árboles del bosque de regresión y se exporta la imagen del árbol a un **archivo .dot** utilizando la función export_graphviz de sklearn.
- Se especifican los nombres de las características y se establecen algunos parámetros para la visualización, como el redondeo y la precisión.
- Luego, se crea el gráfico utilizando el **archivo .dot** y se guarda la imagen resultante en un archivo .png.



(Figura 57:El árbol de decisión Fuente: elaboración propia)

```
print('The depth of this tree is:', tree.tree_.max_depth)
```

```
The depth of this tree is: 13
```

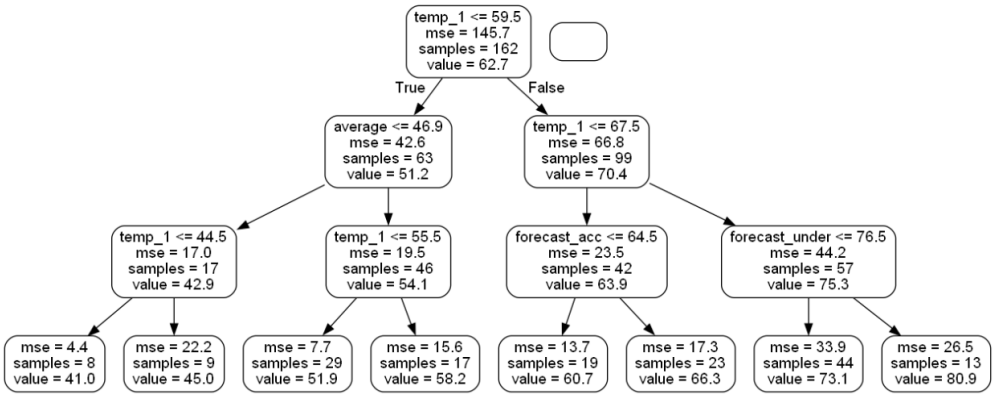
(Figura 58: La profundidad del árbol de decisión Fuente: elaboración propia)

Se puede ver la profundidad del árbol de decisión es 13 (Figura 58).

```
# Limit depth of tree to 2 levels
rf_small = RandomForestRegressor(n_estimators=10, max_depth = 3, random_state=42)
rf_small.fit(train_features, train_labels)

# Extract the small tree
tree_small = rf_small.estimators_[5]

# Save the tree as a png image
export_graphviz(tree_small, out_file = 'small_tree.dot', feature_names = feature_list, 

(graph, ) = pydot.graph_from_dot_file('small_tree.dot')

graph.write_png('small_tree.png')
```

(Figura 59: Crear una pequeña arbol de decisión Fuente: elaboración propia)

- Se crea una instancia de un modelo de bosque aleatorio con n_estimators = 10 y una profundidad máxima de árbol de 3 niveles. A continuación, se ajusta este modelo a los datos de entrenamiento.
- Luego, se extrae uno de los 10 árboles del modelo resultante y se guarda como imagen PNG utilizando la biblioteca pydot para visualizarlo. Este árbol es más pequeño que el modelo original y se utiliza para ilustrar cómo la profundidad del árbol afecta la capacidad del modelo para capturar la complejidad del conjunto de datos.



(Figura 60: Una pequeña arbol de decisión Fuente: elaboración propia)

Variable Importances

```
# Get numerical feature importances
importances = list(rf.feature_importances_)

# List of tuples with variable and importance
feature_importances = [(feature, round(importance, 2)) for feature, importance in zip(feature_list

# Sort the feature importances by most important first
feature_importances = sorted(feature_importances, key = lambda x: x[1], reverse = True)

# Print out the feature and importances
[print('Variable: {:20} Importance: {}'.format(*pair)) for pair in feature_importances];
```

```
Variable: temp_1               Importance: 0.66
Variable: average              Importance: 0.15
Variable: forecast_noaa        Importance: 0.05
Variable: forecast_acc         Importance: 0.03
Variable: day                  Importance: 0.02
Variable: temp_2               Importance: 0.02
Variable: forecast_under       Importance: 0.02
Variable: friend               Importance: 0.02
Variable: month                Importance: 0.01
Variable: year                 Importance: 0.0
Variable: week_Fri             Importance: 0.0
Variable: week_Mon             Importance: 0.0
Variable: week_Sat             Importance: 0.0
Variable: week_Sun             Importance: 0.0
Variable: week_Thurs           Importance: 0.0
Variable: week_Tues            Importance: 0.0
Variable: week_Wed             Importance: 0.0
```

(Figura 61: Variables importantes Fuente: elaboración propia)

- Se calcula la importancia de cada variable en el modelo de Random Forest. Primero, se extrae la importancia de cada variable utilizando la función `**feature_importances_**` de Random Forest.
- Luego, se crea una lista de tuplas que contienen el nombre de cada variable y su importancia, redondeado a dos decimales. Esta lista se ordena por

importancia, de mayor a menor, utilizando la función `**sorted**`.

- Por último, se imprime el nombre de cada variable y su importancia en orden descendente de importancia. (Figura 61)

Two Most Important Features

```
# New random forest with only the two most important variables
rf_most_important = RandomForestRegressor(n_estimators= 1000, random_state=42)

# Extract the two most important features
important_indices = [feature_list.index('temp_1'), feature_list.index('average')]
train_important = train_features[:, important_indices]
test_important = test_features[:, important_indices]

# Train the random forest
rf_most_important.fit(train_important, train_labels)

# Make predictions and determine the error
predictions = rf_most_important.predict(test_important)

errors = abs(predictions - test_labels)

# Display the performance metrics
print('Mean Absolute Error:', round(np.mean(errors), 2), 'degrees.')

mape = np.mean(100 * (errors / test_labels))
accuracy = 100 - mape

print('Accuracy:', round(accuracy, 2), '%.')
```

```
Mean Absolute Error: 3.92 degrees.
Accuracy: 93.76 %.
```

(Figura 62: Dos características importantes Fuente: elaboración propia)

- Este código (Figura 62) crea un nuevo modelo de bosque aleatorio que utiliza solo las dos variables más importantes, `**temp_1**` y `**average**`, en lugar de las 10 variables originales.
- Primero, se usa el nuevo modelo `**rf_most_important**` con 1000 árboles y el mismo valor `**random_state**` que el modelo anterior. Luego, se extraen las columnas `**temp_1**` y `**average**` del conjunto de datos de entrenamiento y de prueba y se utilizan para entrenar el modelo `**rf_most_important**`.
- Se hacen predicciones con el nuevo modelo en el conjunto de prueba y se calcula el error absoluto medio (MAE) y la precisión. La precisión se calcula como el porcentaje de las etiquetas de prueba que se predijeron correctamente.

Se observa que la precesión ya es muy alta, tiene 93.76%

Paso 8: Visualización

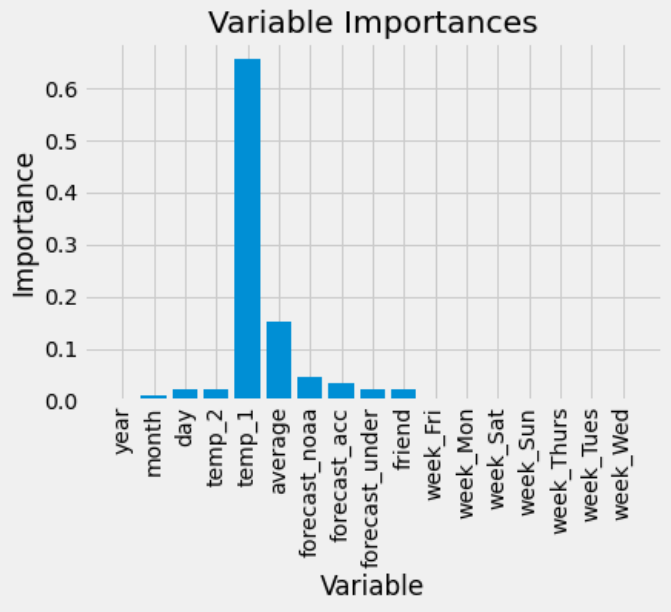
cómo graficar la importancia de las variables utilizadas en el modelo de Random Forest.（Figura 63)

Visualizations

```
# Import matplotlib for plotting and use magic command for Jupyter Notebooks
import matplotlib.pyplot as plt

%matplotlib inline

# Set the style
plt.style.use('fivethirtyeight')

# list of x locations for plotting
x_values = list(range(len(importances)))

# Make a bar chart
plt.bar(x_values, importances, orientation = 'vertical')

# Tick labels for x axis
plt.xticks(x_values, feature_list, rotation='vertical')

# Axis labels and title
plt.ylabel('Importance'); plt.xlabel('Variable'); plt.title('Variable Importances');
```

(Figura 63: Vizualización Fuente: elaboración propia)

- Se importa la librería matplotlib para graficar, y se utiliza el comando mágico `%matplotlib inline` para que las gráficas se muestren en el cuaderno.
- Se establece el estilo de la gráfica con **`plt.style.use('fivethirtyeight')`**.
- Se crean las ubicaciones para las barras de la gráfica en el eje x con **`x_values = list(range(len(importances)))`**.
- Se realiza el gráfico de barras con **`plt.bar(x_values, importances, orientation = 'vertical')`**, donde **`x_values`** es la ubicación de cada barra, `importances` es la altura de cada barra y `orientation = 'vertical'` indica que las barras son verticales. Para añadir las etiquetas en el eje x, se utiliza **`plt.xticks(x_values, feature_list, rotation='vertical')`**, donde **`feature_list`** son los nombres de las variables y `rotation='vertical'` se utiliza para que los nombres se muestren de manera vertical en el eje x. Finalmente, se agregan las etiquetas de los ejes y el título de la gráfica con **`plt.ylabel('Importance')`**; **plt.xlabel('Variable')**; **plt.title('Variable Importances')`**.



(Figura 64: Variables importantes Fuente: elaboración propia)

```
import datetime

# Dates of training values
months = features[:, feature_list.index('month')]
days = features[:, feature_list.index('day')]
years = features[:, feature_list.index('year')]

# List and then convert to datetime object
dates = [str(int(year)) + '-' + str(int(month)) + '-' + str(int(day)) for year, month, day in
dates = [datetime.datetime.strptime(date, '%Y-%m-%d') for date in dates]

# Dataframe with true values and dates
true_data = pd.DataFrame(data = {'date': dates, 'actual': labels})

# Dates of predictions
months = test_features[:, feature_list.index('month')]
days = test_features[:, feature_list.index('day')]
years = test_features[:, feature_list.index('year')]

# Column of dates
test_dates = [str(int(year)) + '-' + str(int(month)) + '-' + str(int(day)) for year, month, da

# Convert to datetime objects
test_dates = [datetime.datetime.strptime(date, '%Y-%m-%d') for date in test_dates]

# Dataframe with predictions and dates
predictions_data = pd.DataFrame(data = {'date': test_dates, 'prediction': predictions})
```

(Figura 65: Transformación de características Fuente: elaboración propia)

Este código (Figura 65) convierte las fechas en las que se realizaron las predicciones y las fechas en las que se conocen los valores reales a objetos de fecha en Python,

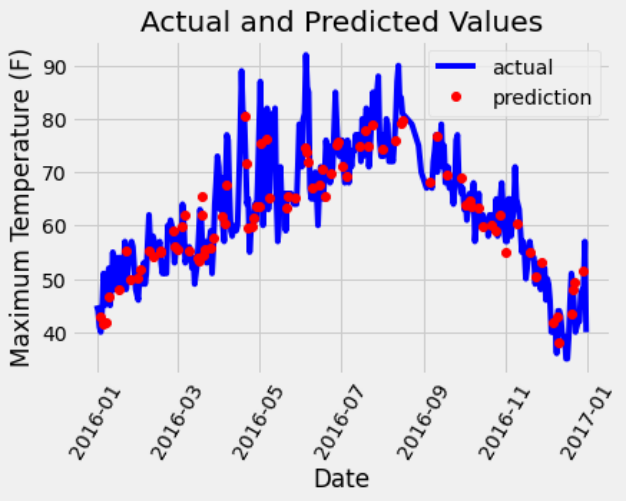
para poder graficarlas en un formato legible.

- Primero, se extraen los valores de mes, día y año de las características del conjunto de datos original, que se almacenan en las variables `**months**`, `**days**` y `**years**`. A continuación, se crea una lista de fechas concatenando estos valores y se convierte esta lista en objetos de fecha utilizando la función `**strptime**` de Python. El objeto de fecha resultante se almacena en la variable `**dates**`.
- Se crea un marco de datos de Pandas denominado `**true_data**` para almacenar las fechas reales y los valores correspondientes. Los valores reales se extraen del conjunto de etiquetas `**labels**`.
- Luego, se repiten los mismos pasos para las fechas en las que se realizaron las predicciones. Los valores de mes, día y año se extraen de las características del conjunto de prueba y se concatenan en una lista de fechas llamada `**test_dates**`.
- Finalmente, se crea otro marco de datos de Pandas llamado `**predictions_data**` para almacenar las fechas de predicción y los valores predichos correspondientes. Los valores predichos se almacenan en la variable `**predictions**`.

```
# Plot the actual values
plt.plot(true_data['date'], true_data['actual'], 'b-', label = 'actual')

# Plot the predicted values
plt.plot(predictions_data['date'], predictions_data['prediction'], 'ro', label = 'prediction')
plt.xticks(rotation = '60');
plt.legend()

# Graph labels
plt.xlabel('Date'); plt.ylabel('Maximum Temperature (F)'); plt.title('Actual and Predicted Values'
```

(Figura 66: Trazar el gráfico Fuente: elaboración propia)

- Los valores se representan en función de la fecha en el eje x y de la temperatura máxima en grados Fahrenheit en el eje y.
- La leyenda indica qué línea corresponde a los valores reales y cuál corresponde a los valores predichos.
- Los títulos de los ejes y el título del gráfico están incluidos para describir los datos y ayudar a interpretar la gráfica.



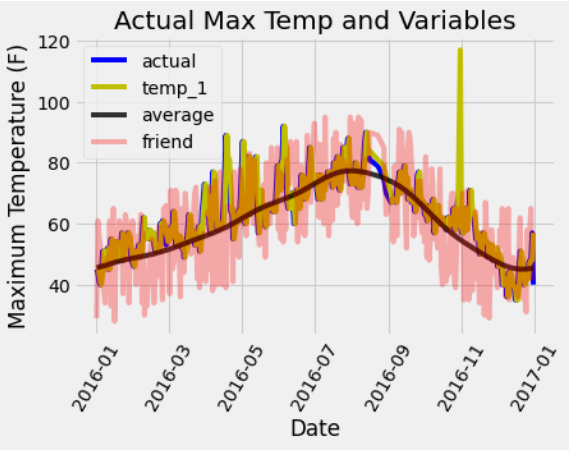
(Figura 67: Valores actuales y predich Fuente: elaboración propia)

```
# Make the data accessible for plotting
true_data['temp_1'] = features[:, feature_list.index('temp_1')]
true_data['average'] = features[:, feature_list.index('average')]
true_data['friend'] = features[:, feature_list.index('friend')]

# Plot all the data as lines
plt.plot(true_data['date'], true_data['actual'], 'b-', label  = 'actual', alpha = 1.0)
plt.plot(true_data['date'], true_data['temp_1'], 'y-', label  = 'temp_1', alpha = 1.0)
plt.plot(true_data['date'], true_data['average'], 'k-', label = 'average', alpha = 0.8)
plt.plot(true_data['date'], true_data['friend'], 'r-', label = 'friend', alpha = 0.3)

# Formatting plot
plt.legend(); plt.xticks(rotation = '60');

# Lables and title
plt.xlabel('Date'); plt.ylabel('Maximum Temperature (F)'); plt.title('Actual Max Temp and Var
```

(Figura 68: Valores actuales y predichos Fuente: elaboración propia)

- En esta sección del código, se están preparando los datos para ser visualizados en una gráfica que muestra la temperatura máxima real, junto con las variables **"temp_1", "average" y "friend"** que se utilizaron en el modelo para hacer predicciones.
- Para esto, se utiliza el módulo "datetime" para convertir las fechas en una lista "dates" a objetos de fecha y hora, que luego se convierten en un objeto de datos pandas "true_data". Se realiza lo mismo para los datos de predicción en "test_dates" y "predictions_data".
- Luego, se agregan las variables **"temp_1", "average" y "friend" a "true_data"** para que se puedan graficar junto con la temperatura máxima real. Estas variables se representan con diferentes colores en la gráfica.
- Finalmente, se utiliza el método **"plot"** de matplotlib para dibujar la gráfica

con las diferentes líneas de los datos de temperatura y las variables. Se establecen las etiquetas y el título del gráfico y se muestra la leyenda.



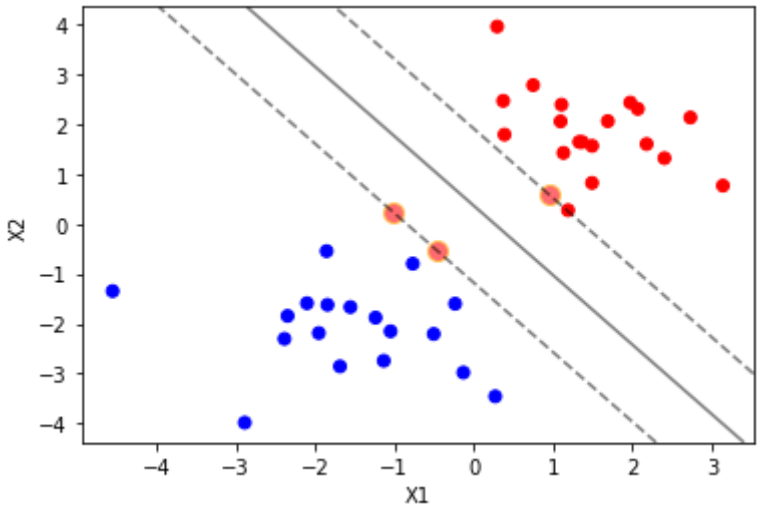
(Figura 69: Temperatura y variables máximas Fuente: elaboración propia)

5.5 SVM

Las Máquinas de Vectores de Soporte (SVM) (Figura 70), también conocidas como SVMs, son más adecuadas para escenarios en los que es necesario examinar cuidadosamente el conjunto de datos. Funcionan mejor en casos en los que se necesite segregar dos clases, considerando las características extremas de ambas clases.

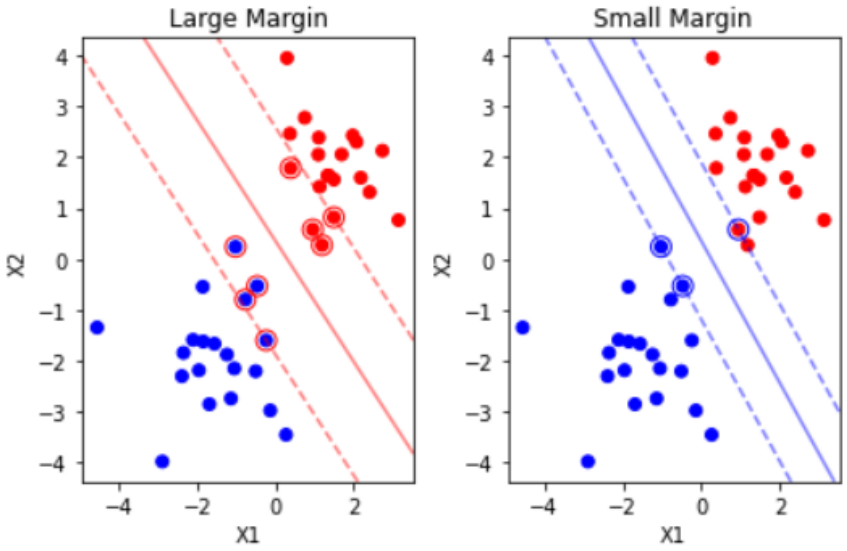
En términos simples, también podemos decir que las SVM actúan como una frontera que puede segregar dos clases de la mejor manera posible. La frontera que separa dos clases también se conoce como límite de decisión, y las SVM nos ayudan a obtener un límite de decisión óptimo.

Las SVM, como mencionamos anteriormente, introducen discriminadores lineales/hiperplanos que maximizan la distancia entre el hiperplano y los "puntos difíciles" cerca del hiperplano.



(Figura 70: Ejemplo de SVM Fuente: elaboración propia)

El hiperplano óptimo (Figura 71) es el que tiene la máxima separación entre las clases. En casos en los que los datos no son linealmente separables, se utiliza una técnica conocida como kernel trick, que permite transformar los datos de entrada a un espacio de mayor dimensión, donde se pueden separar linealmente. Esto es posible gracias a la utilización de funciones kernel, que permiten trabajar con datos no lineales en el espacio original.



(Figura 71: Margen de decisión de SVM Fuente: elaboración propia)

SVM es útil en una variedad de aplicaciones, como la clasificación de correos

electrónicos, la detección de spam, la clasificación de imágenes y la detección de fraudes. Además, es conocido por su capacidad para manejar grandes conjuntos de datos y alta precisión.

Sin embargo, el tiempo de entrenamiento de SVM puede ser bastante largo en conjuntos de datos grandes y complejos, y la selección adecuada de los parámetros de SVM puede ser una tarea difícil. Aun así, SVM sigue siendo uno de los algoritmos más populares en el aprendizaje supervisado debido a su capacidad para clasificar datos con alta precisión y su capacidad para trabajar en conjuntos de datos grandes.

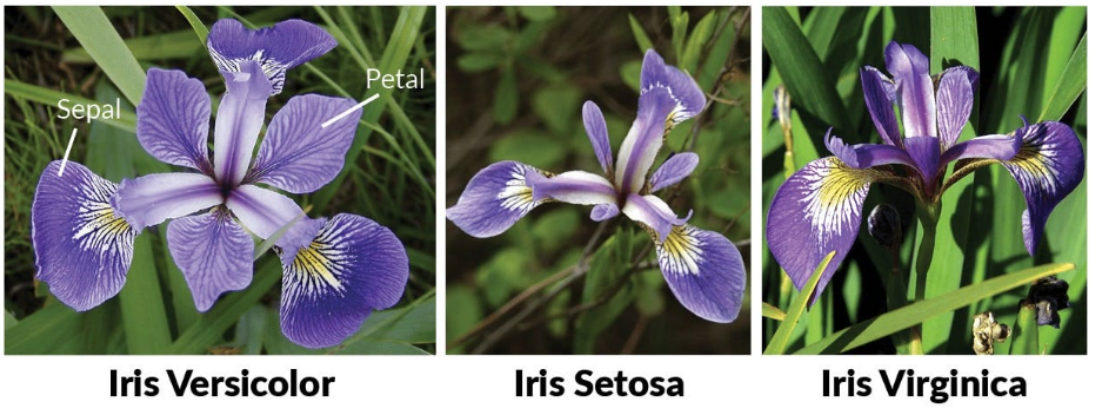
Aplicación de SVM en empresas

Segmentación de clientes: Random Forest se puede utilizar para la segmentación de clientes, que se pueden clasificar según los atributos y comportamientos de los clientes, para personalizar las estrategias de marketing y mejorar la satisfacción del cliente y la eficacia de las ventas.

Evaluación crediticia: los bosques aleatorios se pueden utilizar para la evaluación crediticia y la gestión de riesgos, lo que ayuda a las empresas a juzgar el estado crediticio de los clientes, reducir el riesgo de deudas incobrables y optimizar el proceso de aprobación de préstamos.

Ejemplo de SVM

Se ultiliza los datos de Iris para explicar el árbol de decisión. "datos de Iris" se trata de un conjunto de datos de Iris, también conocido como conjunto de datos de Fisher's Iris. Este conjunto de datos es uno de los más famosos en el campo de la minería de datos y el aprendizaje automático. Consiste en 50 muestras de cada una de las tres especies de iris (Iris setosa, Iris virginica e Iris versicolor)（Figura 72), lo que hace un total de 150 muestras. Para cada muestra, se miden cuatro características: longitud del sépalo, ancho del sépalo, longitud del pétalo y ancho del pétalo. Estos datos se utilizan comúnmente para entrenar y probar modelos de clasificación y clustering en el aprendizaje automático.



*(Figura 72: 3 clases de Iris Fuente https://miro.medium.com/max/2550/0*GVjzZeYrir0R_6-X.png)*

Paso1: Importar los datos

En este caso（Figura 73) como los datos son muy completos, es decir, no hay datos perdidos por eso no hace falta abordar los datos perdidos. Solo se importan los datos y se echa un vistazo de los.

```
#Import the libraries

import pandas as pd
import numpy as np
from sklearn.svm import SVC
from sklearn.metrics import classification_report, confusion_matrix
import matplotlib.pyplot as plt
%matplotlib inline
```

```
#Read the input data from the external CSV

irisdata = pd.read_csv('iris.csv')
```

```
#Take a look at the data

irisdata.head()
```

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

(Figura 73: Introducir los datos Fuente: elaboración propia)

```
irisdata.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   sepal_length  150 non-null    float64
 1   sepal_width   150 non-null    float64
 2   petal_length  150 non-null    float64
 3   petal_width   150 non-null    float64
 4   species       150 non-null    object
dtypes: float64(4), object(1)
memory usage: 6.0+ KB
```

(Figura 74: Información de datos Fuente: elaboración propia)

Paso 2: Visualizar los datos

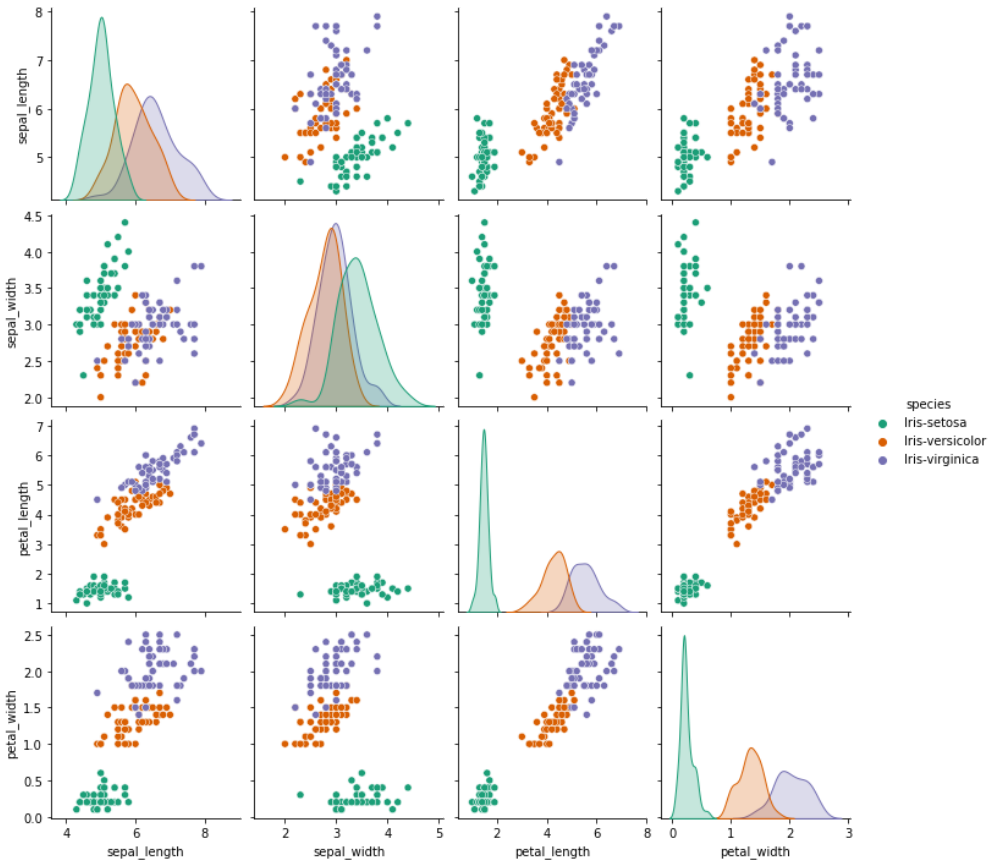
Se utiliza la biblioteca Seaborn de Python (Figura 75) para crear una matriz de gráficos de dispersión (scatter plots) y histogramas que muestran la relación entre diferentes pares de características de un conjunto de datos de flores Iris.

```
#Visualise Data with Pairs Plots
import seaborn as sns
sns.pairplot(irisdata,hue='species',palette='Dark2')
```

(Figura 75:Visualización de datos Fuente: elaboración propia)

- La función `**pairplot**` de Seaborn crea una cuadrícula de gráficos que muestran la distribución de cada característica y la relación entre cada par de características.
- El argumento `**hue**` se utiliza para distinguir entre diferentes clases de flores Iris y el argumento `**palette**` se utiliza para establecer la paleta de colores utilizada para cada clase.

<seaborn.axisgrid.PairGrid at 0x2c8197ed190>



(Figura 76: Resultado de visualización de datos Fuente: elaboración propia)

```
#Train Test Split — Split your data into a training set and a testing set.

from sklearn.model_selection import train_test_split
X = irisdata.drop('species', axis=1)
y = irisdata['species']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20)
```

(Figura 77:Dividir datos y estrenarlos Fuente: elaboración propia)

Este Código (Figura 77) se utiliza 'train Test La función Split 'divide aleatoriamente una parte de los datos del conjunto de datos' irisdata' como conjunto de prueba, y los datos restantes como conjunto de entrenamiento. Entre ellos:

- **'x = irisdata.drop (' species', Axis = 1) '**: eliminar la columna' species' del conjunto de datos' irisdata 'y obtener el conjunto de datos' x' que solo contiene columnas características.
- **'y = irisdata (' species') '**: utilizar la columna' species' del conjunto de datos' irisdata 'como variable objetivo' y '.

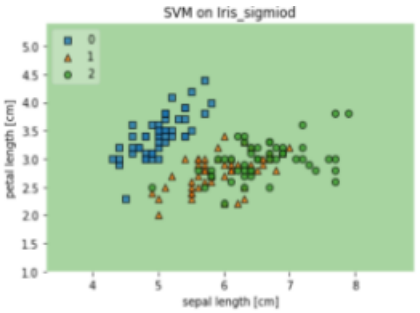
- **'train_Test_Split (x, y, Test_size = 0,20) '**: siga los datos característicos' x' y los datos objetivo **'test_size** 'la proporción del parámetro se divide aleatoriamente, de los cuales**' test_size = 0,20** 'significa que el 20% de los datos se utilizan como conjunto de prueba y el 80% de los datos se utilizan como conjunto de entrenamiento.
- **'X_train, X_test, y_train, y_test = ... ':** Asignar el conjunto de datos dividido a **X_train、X_test、y_train e y_test** . Entre ellos, 'x Tren 'y' y La línea 'es la característica y la variable objetivo del conjunto de entrenamiento,' X Test 'y' y La prueba 'es la característica y la variable objetivo del conjunto de pruebas, respectivamente.

```
#Apply kernels to transform the data to a higher dimension
from sklearn.metrics import classification_report


kernels = ['Polynomial', 'RBF', 'Sigmoid','Linear']

#A function which returns the corresponding SVC model
def getClassifier(ktype):
    if ktype == 0:
        # Polynomial kernal
        return SVC(kernel='poly', degree=8, gamma="auto")
    elif ktype == 1:
        # Radial Basis Function kernal
        return SVC(kernel='rbf', gamma="auto")
    elif ktype == 2:
        # Sigmoid kernal
        return SVC(kernel='sigmoid', gamma="auto")
    elif ktype == 3:
        # Linear kernal
        return SVC(kernel='linear', gamma="auto")

for i in range(4):
    # Separate data into test and training sets
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20)
    svclassifier = getClassifier(i)
    svclassifier.fit(X_train, y_train)
    y_pred = svclassifier.predict(X_test)
    print("Evaluation:", kernels[i], "kernel")
    print(classification_report(y_test,y_pred, zero_division=1))
```

(Figura 78:Clasificar los datos Fuente: elaboración propia)

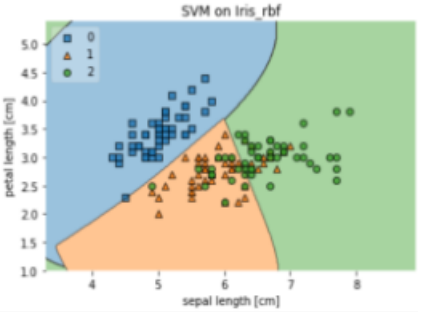
Evaluation: Sigmoid kernel

	precision	recall	f1-score	support
Iris-setosa	0.00	0.00	0.00	10
Iris-versicolor	0.00	0.00	0.00	11
Iris-virginica	0.30	1.00	0.46	9
accuracy			0.30	30
macro avg	0.10	0.33	0.15	30
weighted avg	0.09	0.30	0.14	30



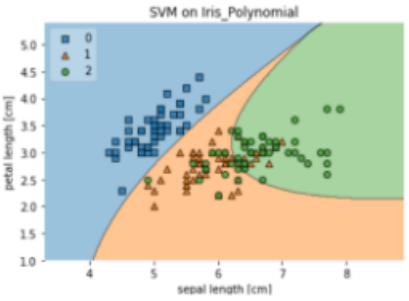
Evaluation: RBF kernel

	precision	recall	f1-score	support
Iris-setosa	1.00	1.00	1.00	10
Iris-versicolor	0.91	1.00	0.95	10
Iris-virginica	1.00	0.90	0.95	10
accuracy			0.97	30
macro avg	0.97	0.97	0.97	30
weighted avg	0.97	0.97	0.97	30



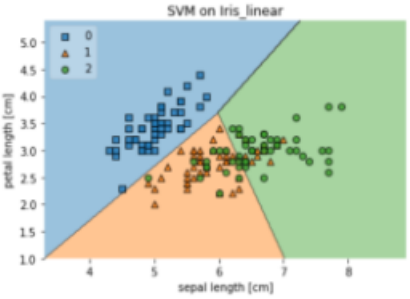
Evaluation: Polynomial kernel

	precision	recall	f1-score	support
Iris-setosa	1.00	1.00	1.00	13
Iris-versicolor	1.00	0.88	0.93	8
Iris-virginica	0.90	1.00	0.95	9
accuracy			0.97	30
macro avg	0.97	0.96	0.96	30
weighted avg	0.97	0.97	0.97	30



Evaluation: linear kernel

	precision	recall	f1-score	support
Iris-setosa	1.00	1.00	1.00	6
Iris-versicolor	1.00	1.00	1.00	9
Iris-virginica	1.00	1.00	1.00	15
accuracy			1.00	30
macro avg	1.00	1.00	1.00	30
weighted avg	1.00	1.00	1.00	30

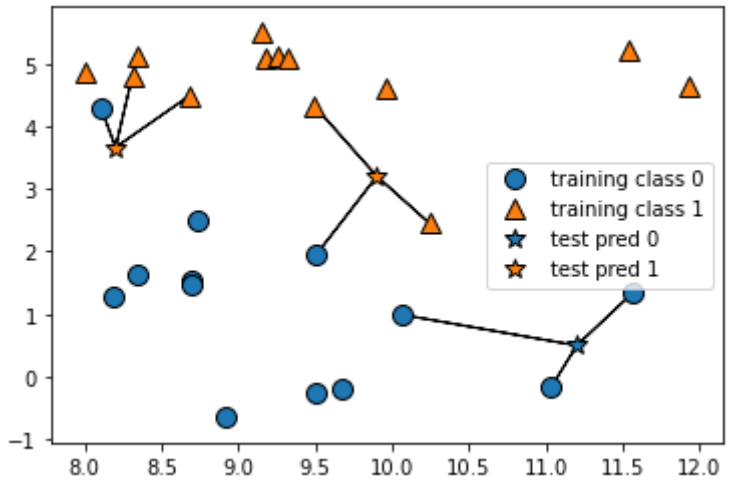


(Figura 79:Resultado de clasificación Fuente: elaboración propia)

5.6 K-Vecinos

El algoritmo K-Vecinos (Figura 80) más Cercanos o K-Nearest Neighbors (KNN) es un algoritmo de aprendizaje supervisado utilizado para la clasificación y regresión. En términos simples, el algoritmo KNN clasifica un objeto en función de los objetos etiquetados más cercanos en el conjunto de datos.

La "K" en KNN representa el número de vecinos más cercanos que se utilizan para clasificar el objeto desconocido. Para clasificar un nuevo objeto, el algoritmo KNN busca los K vecinos más cercanos en el conjunto de entrenamiento y asigna la etiqueta más común entre esos K vecinos al objeto desconocido. En otras palabras, la etiqueta del objeto desconocido se decide por mayoría de votos entre sus K vecinos más cercanos.



(Figura 80:Modelo de K-vecinos Fuente: elaboración propia)

La distancia se mide generalmente en términos de distancia euclidiana, pero también se pueden utilizar otras medidas de distancia. El valor de K se elige mediante validación cruzada o según el conocimiento del dominio.

Una de las principales ventajas del algoritmo KNN es que es fácil de entender e implementar. Además, es útil para clasificar datos no lineales. Sin embargo, el rendimiento del algoritmo se puede ver afectado negativamente por datos con muchos atributos o características, y puede ser costoso en términos de tiempo de cálculo en

grandes conjuntos de datos.

Aplicación de K vecino en empresas

Servicio al cliente y soporte: El algoritmo K-vecino más cercano puede asignar clientes a los representantes de servicio al cliente más relevantes de acuerdo con sus necesidades y problemas, mejorando la calidad y la eficiencia del servicio al cliente.

Marketing: K-Nearest Neighbor Algorithm puede encontrar grupos de usuarios similares en función de los intereses de los usuarios y los comportamientos de compra, llevar a cabo actividades precisas de marketing y promoción, y aumentar las ventas y la cuota de mercado.

Paso 1: Procesar los datos

Se importan los datos para echar un vistazo de los datos y luego se abordan los datos perdidos, en este caso se puede ver que no existen datos perdidos, es un dataset completo.

Import Library

```
import pandas as pd
import numpy as np

import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_auc_score
from sklearn.model_selection import GridSearchCV

import warnings
warnings.filterwarnings('ignore')
```

Load Dataset

```
df = pd.read_csv('heart.csv')
```

(Figura 81: Introducir los datos Fuente: elaboración propia)

```
#Melihat 5 data teratas
df.head()
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	tar
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2	
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2	
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2	

```
#Melihat 5 data terbawah
df.tail()
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	t
298	57	0	0	140	241	0	1	123	1	0.2	1	0	3	
299	45	1	3	110	264	0	1	132	0	1.2	1	0	3	
300	68	1	0	144	193	1	1	141	0	3.4	1	2	3	
301	57	1	0	130	131	0	1	115	1	1.2	1	1	3	
302	57	0	1	130	236	0	0	174	0	0.0	1	1	2	

(Figura 82:Mostrar las primera y últimas filas de datos Fuente: elaboración propia)

```
#Melihat jumlah rows (303) dan jumlah kolom/features (14)
df.shape
```

```
(303, 14)
```

(Figura 83:Mostrar la forma de datos Fuente: elaboración propia)

```
#General information dari data
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 303 entries, 0 to 302
Data columns (total 14 columns):
age         303 non-null int64
sex         303 non-null int64
cp          303 non-null int64
trestbps    303 non-null int64
chol        303 non-null int64
fbs         303 non-null int64
restecg     303 non-null int64
thalach     303 non-null int64
exang       303 non-null int64
oldpeak     303 non-null float64
slope       303 non-null int64
ca          303 non-null int64
thal        303 non-null int64
target      303 non-null int64
dtypes: float64(1), int64(13)
memory usage: 33.2 KB
```

(Figura 84 :Mostrar la información de datos Fuente: elaboración propia)

Handling Missing Values

```
#Data sudah clean dan bisa dilanjutkan ke tahap EDA
df.isnull().sum()
```

```
age         0
sex         0
cp          0
trestbps    0
chol        0
fbs         0
restecg     0
thalach     0
exang       0
oldpeak     0
slope       0
ca          0
thal        0
target      0
dtype: int64
```

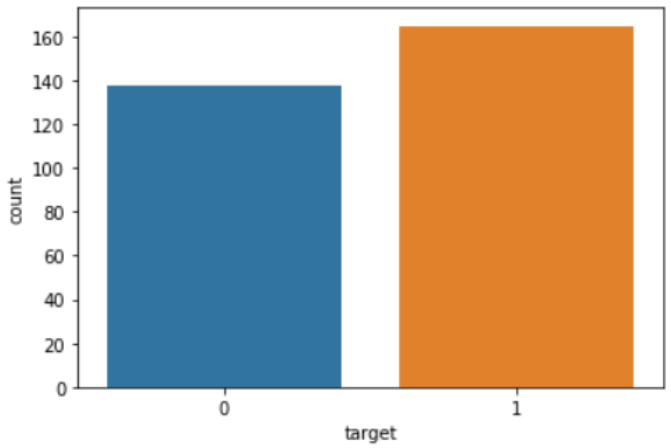
(Figura 85: Resolver datos perdidos Fuente: elaboración propia)

Paso 2: Exploración de datos (EDA)

Exploratory Data Analysis

```
#Univariate analysis target.
sns.countplot(df['target'])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x25dae0587f0>
```



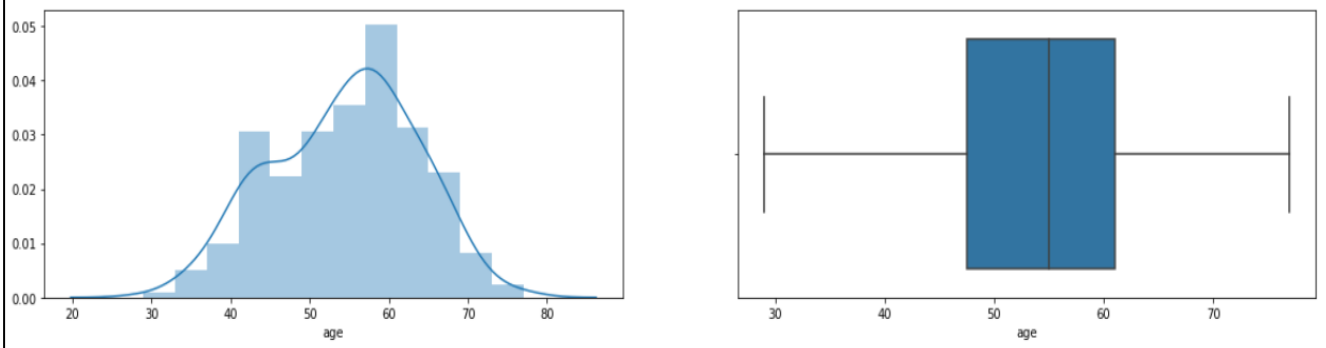
- Sepertinya sudah balance karena jumlah nilai 0 dan 1 tidak berbeda jauh.
- Nilai 0 untuk Heart Disease.
- Nilai 1 untuk No Heart Disease.

(Figura 86: La histograma de distribución de la enfermedad cardíaca Fuente: elaboración propia)

- Este código (Figura 86) utiliza la función ´**countplot**´ de la biblioteca **Seaborn**, que se utiliza para dibujar un histograma y calcular la frecuencia de aparición de cada categoría.
- Aquí, se dibuja la distribución de conteos de la columna **'target'** del dataframe df, que representa el objetivo de una variable de clasificación binaria, donde **1 indica una enfermedad cardiovascular y 0 indica la ausencia de ella**. El objetivo de este código es proporcionar información sobre la distribución de categorías de la variable objetivo.

```
#Univariate analysis age.

f = plt.figure(figsize=(20, 4))

f.add_subplot(1, 2, 1)
sns.distplot(df['age'])

f.add_subplot(1, 2, 2)
sns.boxplot(df['age'])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x25dae36f978>
```



(Figura 87: La histograma y boxplot cardíaca Fuente: elaboración propia)

Este código se utiliza para analizar una única variable en el conjunto de datos, en este caso se selecciona la columna 'edad' o 'age'.

- Primero, se crea una figura que contiene dos subgráficos, cada uno con una anchura de la mitad de la figura completa. El subgráfico de la izquierda utiliza la función **distplot** de la biblioteca Seaborn para dibujar un histograma y una estimación de densidad de núcleo de la columna 'age'.
- El subgráfico de la derecha utiliza la función **boxplot de Seaborn** para dibujar un diagrama de caja de la columna 'age'.
- El diagrama de caja puede mostrar información sobre los cuartiles, la mediana, los valores atípicos y otros datos. Con este código, se puede obtener información sobre la distribución, la mediana, los cuartiles, los valores atípicos y otros datos de la columna 'age', lo que puede ayudar en un análisis y procesamiento de datos posteriores.

Desde la histograma, se observa que la distribución de los datos concentran en la gente con la edad entre 50 y 60 años

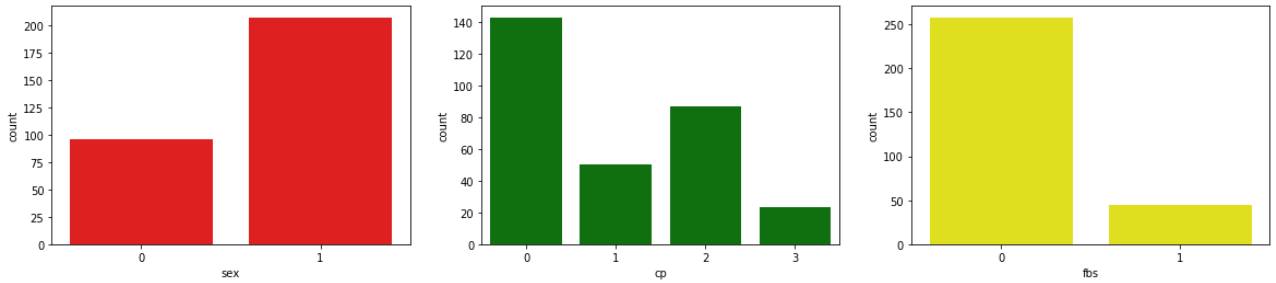
En boxplot muestra que no hay valores atípicos. (Figura 87)

```
#Univariate analysis sex: 1=male; 0=female.
#Univariate analysis chest pain type (cp): 0=typical angina; 1=atypical angi
#Univariate analysis fasting blood sugar: 1 if > 120 mg/dl; 0 otherwise.

f = plt.figure(figsize=(20,4))

f.add_subplot(1,3,1)
sns.countplot(df['sex'], color='red')

f.add_subplot(1,3,2)
sns.countplot(df['cp'], color='green')

f.add_subplot(1,3,3)
sns.countplot(df['fbs'], color='yellow')
```

(Figura 88: Análisis univariat Fuente: elaboración propia)

Este código (Figura 88) se utiliza para analizar una única variable (característica) del conjunto de datos, en el que hemos elegido la columna 'age'.

- Primero, se crea una figura que contiene dos subgráficos, cada uno con una anchura igual a la mitad de la figura completa.
- El subgráfico izquierdo utiliza la función ´**distplot**´ de **Seaborn** para trazar un histograma y una estimación de densidad de núcleo de la columna 'age'. El subgráfico derecho utiliza la función ´**boxplot**´ de **Seaborn** para trazar un diagrama de caja de la columna **'age'**. El diagrama de caja puede mostrar información como los cuartiles, la mediana, los valores atípicos, entre otros.



(Figura 89: Gráficos de Análisis univariat Fuente: elaboración propia)

Los pacientes masculinos resultan tener más números o incluso 2 veces el número de pacientes femeninas.

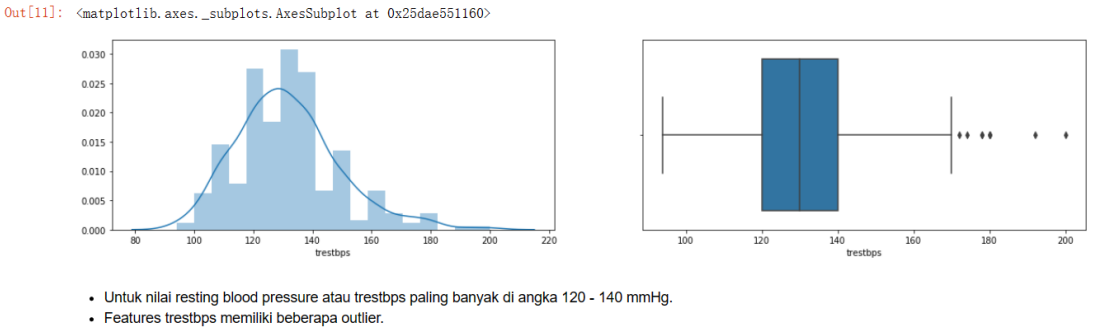
La mayoría de los pacientes tienen el tipo CP 0, que es la angina típica y el tipo mínimo es el 3, que es asintomático.

El gráfico anterior muestra que hay muchos valores de azúcar en la sangre en ayunas por debajo de 120 o 0. (Figura 89)

```
#Univariate analysis resting blood pressure (mm Hg) atau trestbps.

f = plt.figure(figsize=(20, 4))

f.add_subplot(1, 2, 1)
sns.distplot(df['trestbps'])

f.add_subplot(1, 2, 2)
sns.boxplot(df['trestbps'])
```

(Figura 90: Gráficos de Análisis univariat Fuente: elaboración propia)

- Se crea una figura con dos subtramas, cada una con un ancho de la mitad de la figura completa. La subparcela izquierda usa la función ´**distplot**´ de la biblioteca Seaborn para trazar un histograma y una estimación de la densidad del núcleo de la columna **'trestbps'**.
- La subparcela derecha usa la función boxplot de la biblioteca Seaborn para trazar una gráfica de caja de la columna **'trestbps'**. El diagrama de caja puede mostrar información sobre los cuartiles, la mediana, los valores atípicos y más de los datos. Se puede obtener la información sobre la distribución, la mediana, los cuartiles y los valores atípicos de la columna **'trestbps'**, que puede ser útil para el análisis y preprocesamiento de datos posteriores.



(Figura 91: La histograma y boxplot cardíaca Fuente: elaboración propia)

Para el valor de la presión arterial en reposo o trestbps, la mayoría de los números oscilan entre 120 y 140 mmHg.

La función trestbps tiene varios valores atípicos.

Paso 3: Modelar

Modelling

```
#Membuat object KNN
knn = KNeighborsClassifier()

#Membuat variable x dan y
x = df.drop(columns=['target'])
y = df['target']

#Split data kedalam training dan testing
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=4)

#Training the model
knn.fit(x_train, y_train)

KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                     weights='uniform')

#Predict testing set
y_pred = knn.predict(x_test)

#Check performa model menggunakan classification_report
print(classification_report(y_test, y_pred))
```

(Figura 92:Modelar por K vecinos Fuente: elaboración propia)

Este código (Figura 92) es para crear un modelo de clasificación utilizando el algoritmo K-Vecinos más cercanos (KNN).

- Primero se crea un objeto KNN utilizando la clase KNeighborsClassifier de la biblioteca Scikit-learn. Luego, se divide el conjunto de datos en conjuntos de entrenamiento y prueba utilizando la función train_test_split de la biblioteca Scikit-learn.
- Se ajusta el modelo KNN al conjunto de entrenamiento utilizando el método fit y se utiliza el modelo entrenado para hacer predicciones sobre el conjunto de prueba utilizando el método ´predict´.
- Finalmente, se imprime un informe de clasificación que muestra la precisión, la recuperación y el valor F1 del modelo en el conjunto de prueba utilizando la función classification_report de la biblioteca Scikit-learn.

	precision	recall	f1-score	support
0	0.48	0.52	0.50	25
1	0.65	0.61	0.63	36
accuracy			0.57	61
macro avg	0.56	0.57	0.56	61
weighted avg	0.58	0.57	0.58	61

(Figura 93: Informe de clasificación Fuente: elaboración propia)

En el informe de clasificación (Figura 93), se puede ver que el modelo tiene un rendimiento promedio de alrededor del 57 % que va desde precisión, recuperación, puntaje f1 y soporte. La precisión también se muestra en un valor del 57%.

Luego, para la puntuación AUC, se puede ver que el valor está alrededor del 56,5%.

Hyperparameter Tuning Menggunakan Grid Search

```
#List Hyperparameters yang akan diuji
leaf_size = list(range(1,50))
n_neighbors = list(range(1,30))
p=[1,2]
```

```
#Menjadikan ke dalam bentuk dictionary
hyperparameters = dict(leaf_size=leaf_size, n_neighbors=n_neighbors, p=p)
```

```
knn_2 = KNeighborsClassifier()
```

```
#cv itu cross validation
clf = GridSearchCV(knn_2, hyperparameters, cv=10)
```

```
best_model = clf.fit(x,y)
```

```
#Nilai hyperpaameters terbaik
print('Best leaf_size:', best_model.best_estimator_.get_params()['leaf_size'])
print('Best p:', best_model.best_estimator_.get_params()['p'])
print('Best n_neighbors:', best_model.best_estimator_.get_params()['n_neighbors'])
```

(Figura 94: Informe de clasificación Fuente: elaboración propia)

- Este código (Figura 94) muestra cómo realizar una sintonización de hiperparámetros utilizando Grid Search para un modelo KNN. Primero, se crea una lista de hiperparámetros para probar, incluyendo los valores de 'leaf_size', 'n_neighbors' y 'p'.

- Luego, se define un objeto KNN y se utiliza GridSearchCV de scikit-learn para realizar la búsqueda de hiperparámetros utilizando una validación cruzada de 10 pliegues.
- El mejor modelo se selecciona en función de su puntaje de precisión media. Finalmente, se muestran los valores óptimos de los hiper parámetros encontrados por Grid Search.

```
Best leaf_size: 5
Best p: 1
Best n_neighbors: 7
```

(Figura 95: Los valores óptimos de los hiperparámetros Fuente: elaboración propia)

De GridSearch (Figura 96)**, se puede ver que el mejor número de leaf_size es 5 mientras que el método de distancia óptimo es Manhattan o p = 1.**
Entonces el número más óptimo de K es 7.

```
y_pred = best_model.predict(x_test)

print(classification_report(y_test, y_pred))
```

(Figura 96: Los valores óptimos de los hiperparámetros Fuente: elaboración propia)

- Se usa el modelo KNN entrenado y optimizado **(best_model)** para predecir los valores objetivo **(y_pred)** para el conjunto de prueba **(x_test)**. Luego, los valores predichos se comparan con los valores objetivo reales **(y_test)** para evaluar el rendimiento del modelo.
- **y_test** es la variable objetivo en el conjunto de prueba e **y_pred** es la variable objetivo predicha por el modelo. Esto imprimirá detalles sobre el rendimiento del modelo, como precisión, recuperación, puntaje F1 y soporte.

	precision	recall	f1-score	support
0	0.72	0.72	0.72	25
1	0.81	0.81	0.81	36
accuracy			0.77	61
macro avg	0.76	0.76	0.76	61
weighted avg	0.77	0.77	0.77	61

(Figura 97:Optimizar el modelo Fuente: elaboración propia)

El uso de Hyperparameters Tuning(Figura 97) **puede mejorar el rendimiento del modelo en aproximadamente un 20 % hasta un rango del 77 % para todas las matrices de evaluación.**

```
Out[33]: 0.7627777777777778
```

(Figura 98:Resultado de ROC Fuente: elaboración propia)

El valor ROC(Figura 98) **también aumentó a 76%**

5.7 El Teorema de Bayes

El Teorema de Bayes es una fórmula matemática que se utiliza en Machine Learning y otras áreas para **calcular la probabilidad de un evento en función de la probabilidad de otro evento relacionado**. En otras palabras, nos permite actualizar nuestras creencias sobre la probabilidad de un evento a medida que obtenemos nueva información.

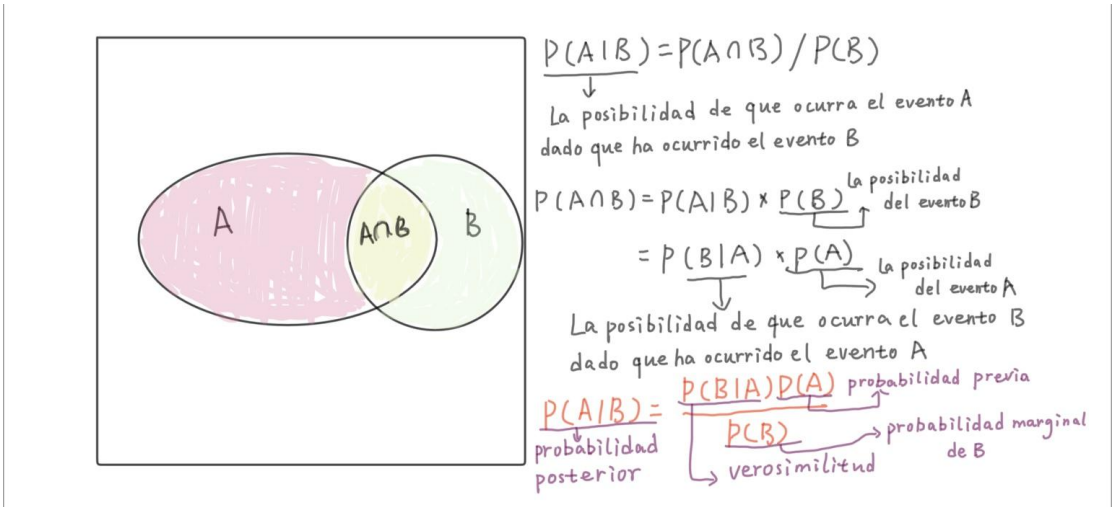
El Teorema de Bayes se compone de dos partes: **la probabilidad previa** y la **probabilidad posterior**. La probabilidad a priori es la probabilidad que tenemos de que ocurra un evento **antes de obtener nueva información**. La probabilidad a posteriori es la probabilidad actualizada que tenemos del evento **después de obtener nueva información**.

Formúla:

P(A|B) = (P(B|A) * P(A)) / P(B)

Donde:

P(A|B) es la probabilidad de que ocurra el evento A dado que ha ocurrido el evento B.
P(B|A) es la probabilidad de que ocurra el evento B dado que ha ocurrido el evento A.
P(A) es la probabilidad del evento A.
P(B) es la probabilidad del evento B.
El teorema de Bayes es especialmente útil cuando se desea actualizar una probabilidad inicial a la luz de nueva evidencia. Se puede aplicar en diversos contextos, como la medicina, la ingeniería, la inteligencia artificial y muchos otros campos donde es necesario hacer inferencias basadas en información previa y observaciones actuales.



(Figura 99:La deducción de teorema de Bayes Fuente: elaboración propia)

La relación entre El teorema de Bayes y la ley de la probabilidad total:

P(B)=P(A∩B)+[[P(H)-P(A)]∩P(B)]
P(B)=P(A∩B)+P(~A)∩P(B)]
=P(A)*[P(A∩B)/P(A)]+P(B)*[P(~A)∩P(B)/P(B)]
P(A)*P(A|B)+P(~A)*P(~A|B)

P(A|B)=P(B|A)*P(A)/P(B)

=P(B|A)*P(A)/P(A)*P(A|B)+P(~A)*P(~A|B)

Aplicación del Teorema de Bayes en empresas

Control de calidad del producto: la fórmula bayesiana se puede utilizar para el control de calidad del producto y la detección de defectos. A través de la actualización bayesiana basada en la probabilidad previa y los resultados de las pruebas, se puede calcular la probabilidad de calidad de los productos en diferentes condiciones para ayudar a las empresas a formular estrategias de control de calidad razonables.

Retrato del usuario y recomendación personalizada: la fórmula bayesiana se puede utilizar para construir el retrato del usuario y hacer una recomendación personalizada. Al combinar el comportamiento histórico del usuario y el conocimiento previo, se puede calcular la probabilidad de preferencia del usuario por diferentes productos o servicios, para lograr recomendaciones personalizadas precisas.

Gestión de la cadena de suministro: la fórmula bayesiana se puede aplicar a la previsión de la demanda y la optimización del inventario en la gestión de la cadena de suministro. Al combinar los datos históricos de ventas y la probabilidad anterior, se puede calcular la probabilidad posterior de la demanda futura para ayudar a las empresas a tomar decisiones razonables de planificación de producción y gestión de inventario.

Ejemplo de teorema de Bayes

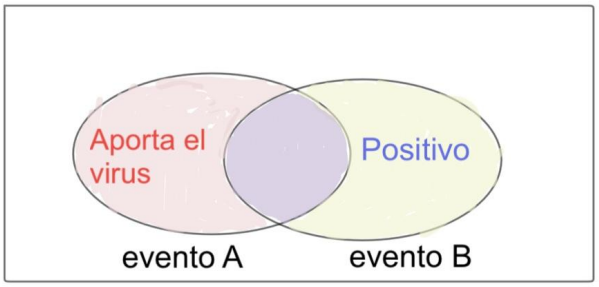
Un ejemplo excelente（Figura 100) será la prueba de covid. Como siempre sabemos el resultado de la prueba primero que es positivo o negativo, sin embargo, eso no significa que aportamos el covid.

La tasa de portadores del virus del nuevo coronavirus en la población es de 0.03, pero debido a varias razones, aquellos que portan el virus pueden no ser necesariamente

positivos, y aquellos que no portan el virus también pueden ser positivos. Supongamos que P(positivo |portador de virus)=0.95, P(negativo| portador de virus) = 0.05, P(positivo | sin virus) = 0.01, P(negativo | sin virus) = 0.99, entonces si una persona da positivo, ¿cuánto es la probabilidad de que tenga el covid?

Trece evaluaciones de cuatro pruebas moleculares diferentes detectaron correctamente un promedio del 95 % de las muestras infectadas con COVID-19. Alrededor del 1% de las muestras dieron resultados falsos positivos

Supongamos que **portar el virus** sea el **evento A**, y esta persona es **positiva** es el **evento B**, entonces :



(Figura 100: Ejemplo ´covid´ de Teorema de Bayes Fuente: elaboración propia)

P(A|B) = (**A∩B**) / P(B)

(**A∩B**) = Positivo y aporta el virus

P(B) = Aporta el virus

P(A|B) = (P(B|A) * P(A)) / P(B)

- P(A|B) es la probabilidad condicional de aportar el virus (A) dado resultado positivo(B).
- P(B|A) es la probabilidad condicional de resultado positivo(B) dado aportar el virus(A).
- P(A) es la probabilidad de aportar el virus.
- P(B) es la probabilidad de resultado positivo.

P(B|A)=0.95

P(B)=0.03*0.95+0.97*0.01=0.382

P(A|B) = 0.03*0.95/(0.03*0.95+0.97*0.01)=74.60%

Es decir, la prueba de covid no siempre es válida, aunque es posible que detecte el covid.

```
]: # calculate the probability of cancer patient and diagnostic test

   # calculate P(A|B) given P(A), P(B|A), P(B|not A)
   def bayes_theorem(p_a, p_b_given_a, p_b_given_not_a):
   ⟶# calculate P(not A)
   ⟶not_a = 1 - p_a
   ⟶# calculate P(B)
   ⟶p_b = p_b_given_a * p_a + p_b_given_not_a * not_a
   ⟶# calculate P(A|B)
   ⟶p_a_given_b = (p_b_given_a * p_a) / p_b
   ⟶return p_a_given_b

   # P(A)
   p_a = 0.03
   # P(B|A)
   p_b_given_a = 0.95
   # P(B|not A)
   p_b_given_not_a = 0.01
   # calculate P(A|B)
   result = bayes_theorem(p_a, p_b_given_a, p_b_given_not_a)
   # summarize
   print('P(A|B) = %.3f%%' % (result * 100))

   P(A|B) = 74.607%
```

(Figura 101: Precisión de la prueba de covid Fuente: Elaboración propia)

En este caso, se supone que la probabilidad de A es p_a, la probabilidad de B dado A es p_b_given_a y la probabilidad de B dado no A es p_b_given_not_a. A partir de estas probabilidades, se puede calcular la probabilidad condicional de A dado B utilizando la función bayes_theorem.

La función bayes_theorem primero calcula la probabilidad de no A como 1 - p_a. Luego, calcula la probabilidad de B como la suma de la probabilidad de B dado A multiplicado por la probabilidad de A y la probabilidad de B dado no A multiplicado por la probabilidad de no A. Finalmente, calcula la probabilidad condicional de A dado B como la probabilidad de B dado A multiplicado por la probabilidad de A

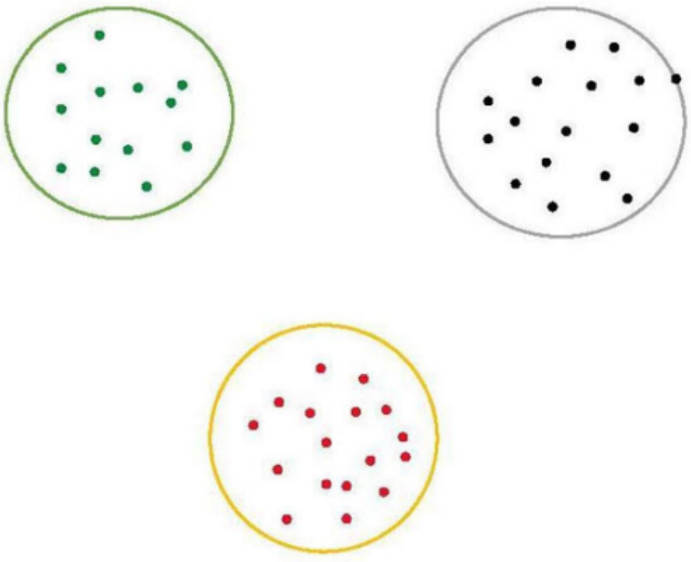
dividido por la probabilidad de B.

En el ejemplo proporcionado, se supone que P(A) = 0.03, P(B|A) = 0.95 y P(B|not A) = 0.01. Luego, se utiliza la función bayes_theorem para calcular P(A|B), que se imprime en la consola como un porcentaje.

6. Métodos de Aprendizaje no Supervisado

6.1. Análisis Clúster

El clustering (Figura 102), como sugiere su nombre, es una agrupación de puntos. Los puntos que tienen características similares se agrupan en un mismo cluster. Se basa en el hecho de que los puntos de datos que tienen propiedades similares deben estar en un grupo y tener propiedades diferentes a los puntos de datos en otros grupos.



(Figura 102: Clustering data into three different groups Fuente: Lakshmi 2020)

La similitud entre los puntos de datos se calcula utilizando una métrica de distancia

basada en un tipo de conjunto de variables de características.

El clustering es un algoritmo simple que proporciona información valiosa sobre nuestros datos. Tiene varias aplicaciones en la industria, por ejemplo:

Biólogía, agrupación de especies

Imágenes médicas

Sistemas de recomendación

Aquí están algunos de los algoritmos de clustering más famosos utilizados actualmente en la industria:

K-means

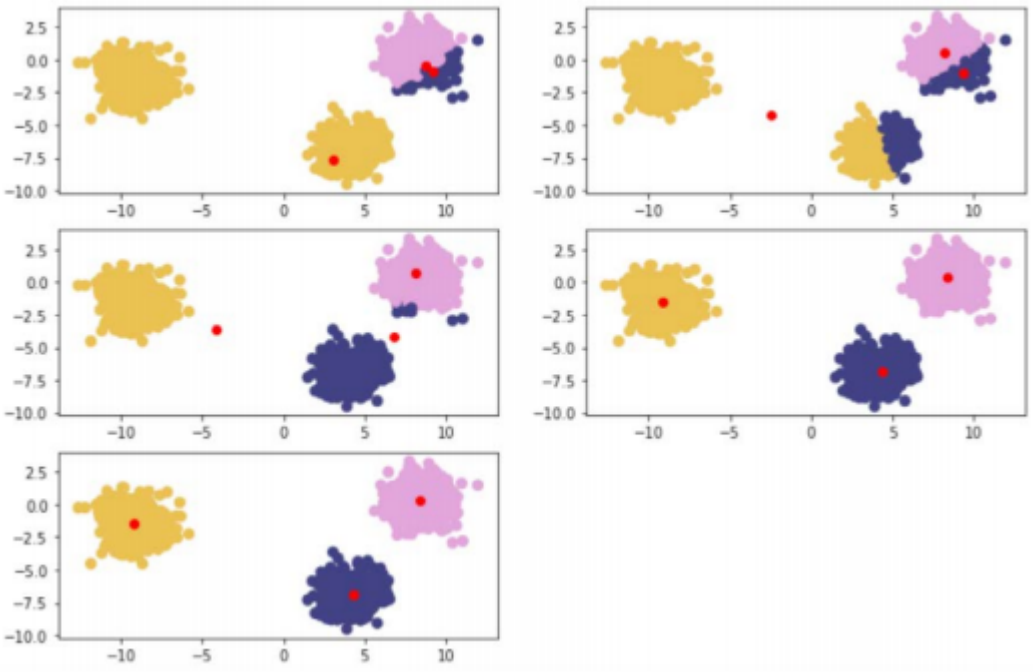
Una de las aproximaciones más básicas pero populares es utilizar un análisis de agrupamiento llamado clustering k-means. El k-means funciona buscando K grupos en tus datos y el flujo de trabajo es bastante intuitivo. Comenzaremos con una introducción sin matemáticas al k-means, seguida de una implementación en Python.

Descripción sin matemáticas del k-means:

Aquí tienes el algoritmo sin matemáticas del clustering k-means (Figura 103):

1. Escoge K centroides (K = número esperado de grupos distintos).
2. Coloca aleatoriamente K centroides en cualquier lugar entre tus datos de entrenamiento existentes.
3. Calcula la distancia euclidiana desde cada centroide hasta todos los puntos en tus datos de entrenamiento.
4. Los puntos de datos de entrenamiento se agrupan con su centroide más cercano.
5. Entre los puntos de datos agrupados en cada centroide, calcula el punto de datos promedio y mueve tu centroide a esa ubicación.
6. Repite este proceso hasta la convergencia, o cuando la pertenencia a cada grupo ya no cambie.

Repite este proceso hasta que los centroides ya no cambien o se alcance una convergencia, es decir, cuando la membresía en cada grupo ya no cambie.



(Figura 103 Proceso de clasificar por K-means Fuente: Johnston 2019)

Clustering jerárquico aglomerativo

Una forma de abordar el clustering jerárquico es comenzar con cada punto de datos como su propio clúster y unir recursivamente los puntos similares para formar clústeres. Esto se conoce como clustering jerárquico aglomerativo. Entraremos en más detalles sobre las diferentes formas de abordar el clustering jerárquico en una sección posterior.

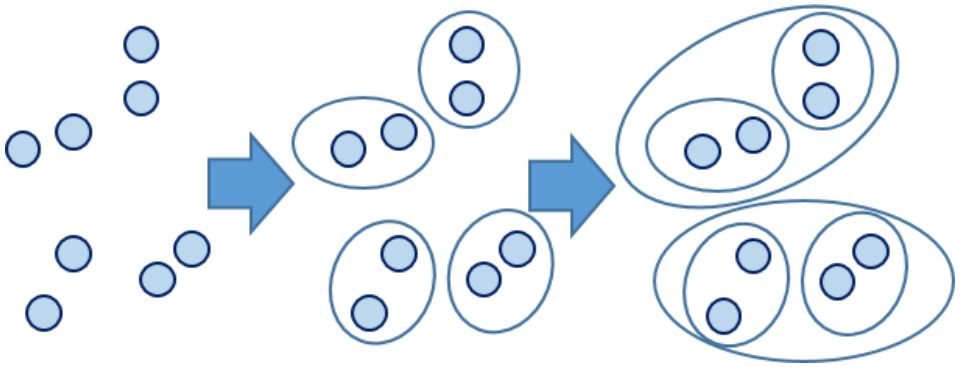
En el enfoque aglomerativo del clustering jerárquico, el concepto de similitud entre puntos de datos se puede pensar en el paradigma que vimos en k-means. En k-means, utilizamos la distancia euclidiana para calcular la distancia entre los puntos individuales y los centroides de los "k" clústeres esperados. Para este enfoque de clustering jerárquico, reutilizaremos la misma métrica de distancia para determinar la similitud entre los registros de nuestro conjunto de datos.

Eventualmente, al agrupar registros individuales con sus registros más similares de

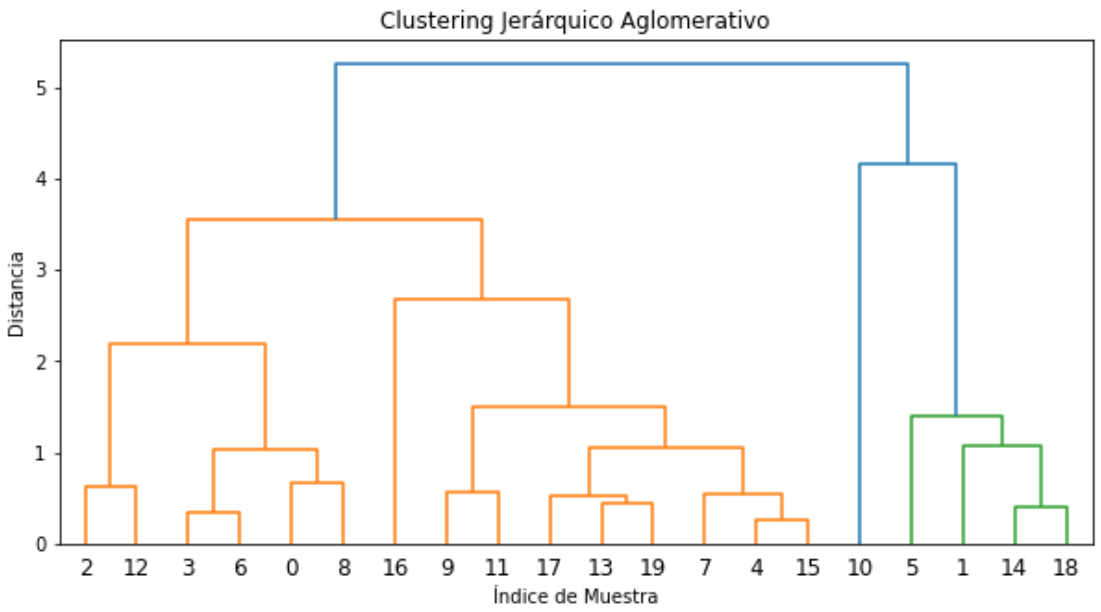
manera recursiva, se construye una jerarquía de abajo hacia arriba. Los clústeres individuales de un solo miembro se unen en un solo clúster en la parte superior de nuestra jerarquía.

Pasos para realizar el clustering jerárquico (Figura 104,105)**:**

1. Dado un conjunto de datos con n puntos de muestra, se considera cada punto como un "clúster" individual con solo ese punto como miembro.
2. Calcular la distancia euclidiana entre los centroides de todos los clústeres en sus datos.
3. Agrupar los pares de puntos más cercanos.
4. Repetir el paso 2 y el paso 3 hasta llegar a un solo clúster que contenga todos los datos de su conjunto.
5. Graficar un dendrograma para mostrar cómo sus datos se han unido en una estructura jerárquica. Un dendrograma es simplemente un diagrama que se utiliza para representar una estructura de árbol, mostrando una disposición de clústeres de arriba a abajo.
6. Decidir en qué nivel desea crear los clústeres.



(Figura 104: Clustering jerárquico aglomerativo Fuente:Hendra 2015)



(Figura 105: Clustering jerárquico aglomerativo Fuente:Elaboración propia)

DBSCAN

El agrupamiento espacial basado en la densidad con aplicaciones de ruido (DBSCAN) es un algoritmo de agrupamiento ML no supervisado. Sin supervisión significa que no utiliza objetos preetiquetados para agrupar puntos de datos. La agrupación se refiere al intento de agrupar puntos de datos similares en grupos o agrupaciones determinadas artificialmente. Puede reemplazar algoritmos de agrupamiento populares como KMeans y agrupamiento jerárquico.

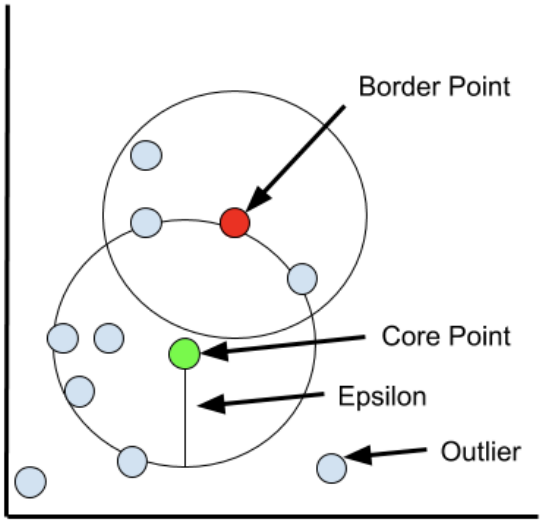
Primero, definamos el Epsilon y el punto mínimo, los dos parámetros requeridos al aplicar el algoritmo DBSCAN y algunos parámetros adicionales.（Figura 106)

Epsilon (ε): El radio máximo de la comunidad. Los puntos de datos serán de la misma clase si su distancia mutua es menor o igual al épsilon especificado. En otras palabras, es la distancia que utiliza DBSCAN para determinar si dos puntos son similares y pertenecen a la misma clase. Un épsilon más grande producirá clústeres más grandes (que contienen más puntos de datos) y un épsilon más pequeño generará clústeres más pequeños. En general, nos gustan los valores más pequeños porque solo necesitamos una pequeña fracción de los puntos de datos para estar a una distancia entre sí. Pero si

es demasiado pequeño, dividirá el grupo cada vez más pequeño.

MinPts (minPts): los vecindarios con un número de minPts dentro del radio de un vecindario se consideran un clúster. Recuerda que los puntos iniciales están incluidos en minPts. Un minPts más bajo ayuda al algoritmo a crear más clústeres con más ruido o valores atípicos. Un minPts más alto garantizará un clúster más sólido, pero si el clúster es demasiado grande, los clústeres más pequeños se fusionarán con el más grande.

Si "puntos mínimos" = 4, 4 o más puntos dentro de una distancia entre sí se consideran un grupo.



Expandir el clustering Fuente:Kamil Mysiak, 2020)

Otros parámetros

Punto central: el punto de datos central tiene al menos la cantidad mínima de puntos de datos dentro de la distancia de su vecino.

Puntos de límite: los puntos de datos de límite se ubican en las afueras como si pertenecieran a los vecinos más cercanos. (por ejemplo, con puntos centrales dentro de la distancia épsilon), pero debe ser inferior a minPts.

Valores atípicos: estos puntos no son vecinos ni puntos fronterizos. Estos puntos están ubicados en áreas de baja densidad.

Ejemplo de DBSCAN

Para el algoritmo DBSCAN, podemos suponer que tenemos un conjunto de datos en coordenadas bidimensionales con la siguiente distribución de puntos:
Punto A: (2, 2)
Punto B: (2, 4)
Punto C: (4, 3)
Punto D: (6, 2)
Punto E: (8, 8)
Punto F: (9, 9)
Ahora, utilizaremos el algoritmo DBSCAN para agrupar estos puntos. **Especificamos un número mínimo de puntos (minPts) de 3** y elegimos **un radio de vecindad (epsilon) de 3**. Con estos parámetros, podemos dividir los puntos en diferentes grupos.
Primero, comenzamos con un punto (por ejemplo, el punto A) y examinamos los puntos en su vecindad. Si el número de puntos en la vecindad es mayor o igual al número mínimo de puntos (minPts), se agrupan estos puntos en un grupo y se continúa explorando la vecindad de estos puntos. Si el número de puntos en la vecindad es menor que el número mínimo de puntos (minPts), se etiqueta ese punto como ruido o sin clasificar y se procede al siguiente punto.
En nuestro ejemplo, podemos realizar el siguiente agrupamiento (Figura 107):

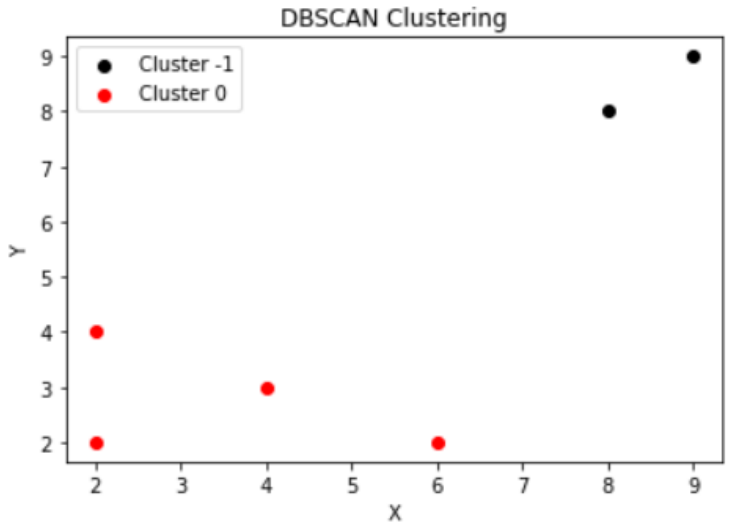
Distancia	A	B	C	D	E	F
A	0	2	√5	4	6√2	4
B	2	0	√5	2√5	2√13	2√5
C	√5	√5	0	√5	√41	√5
D	4	2√5	√5	0	2√10	2√10
E	6√2	2√13	√41	√40	0	2√10
F	7√2	√2 * √37	√61	√58	√2	0

(Figura 107: Proceso de buscar el punto core- cálculo de la distancia de los puntos Fuente: Elaboración propia)

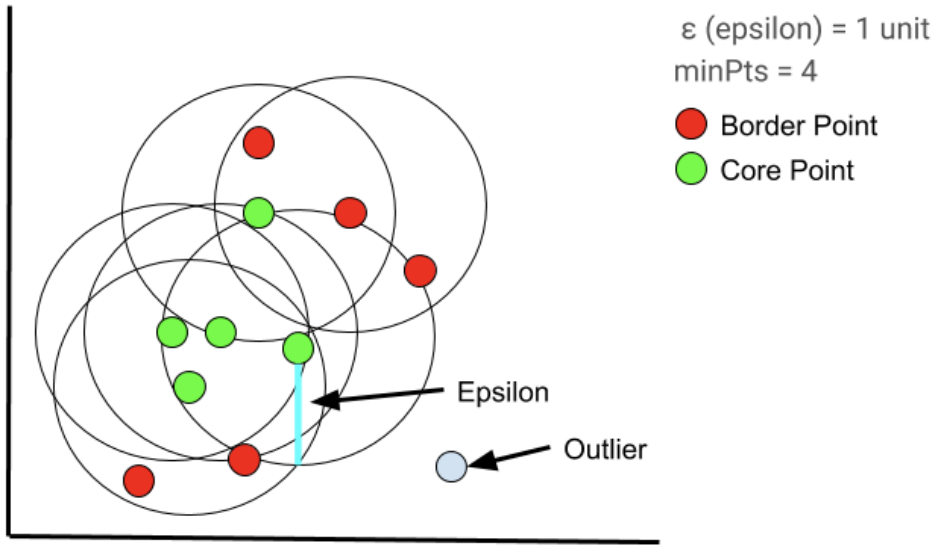
Se observa que C es el punto core, **como ya sabemos** el **radio de vecindad (epsilon) de 3, C tiene una distancia menos de 3 con A, B y D, también el número mínimo de puntos (minPts) de 3**, por eso C se puede considerar como el punto core.

Grupo 1: Contiene los puntos A, B, C, D.

Grupo 2: Contiene los puntos E, F.



(Figura 108 Resultado de DBSCAN clustering Elaboración propia)



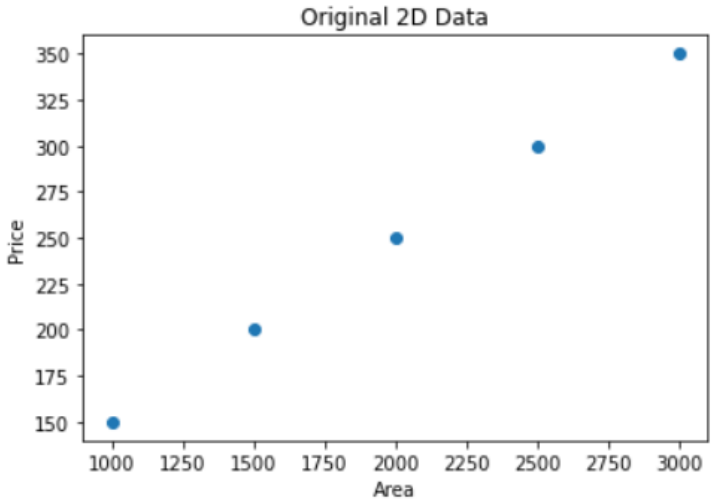
(Figura 109 Expandir el clustering Fuente:Kamil Mysiak, 2020)

Primero, se elige un punto aleatorio que tenga al menos minPts dentro de su radio. Luego, se evalúa cada punto dentro de la vecindad del punto central para ver si tiene minPts dentro de la distancia épsilon (minPts incluye el punto en sí). Si el punto cumple con el criterio minPts, se convertirá en otro punto central y el grupo se expandirá. Si un punto no cumple el criterio minPts, se convierte en un punto límite. A medida que continúa el proceso, el algoritmo comienza a desarrollarse de tal

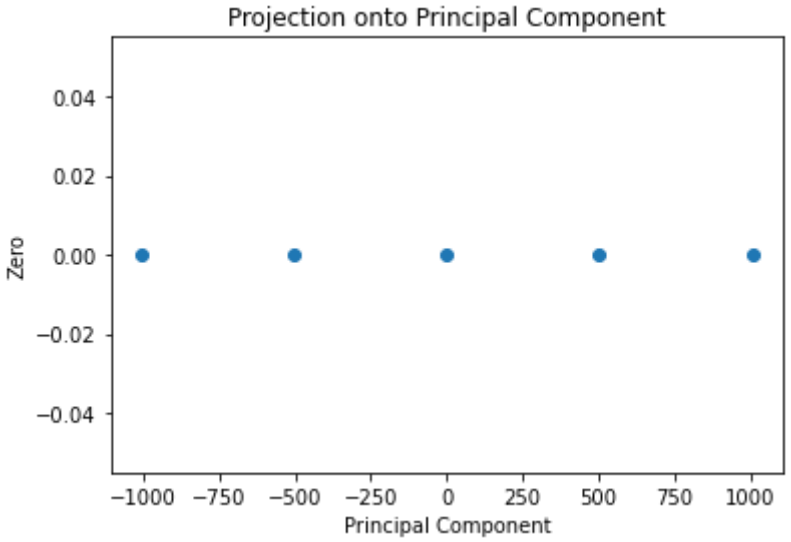
manera que el punto central "a" es vecino de "b", que a su vez es vecino de "c", y así sucesivamente. Cuando un conglomerado está rodeado de puntos límite, el conglomerado se ha buscado por completo porque no hay más puntos dentro de la distancia. Elija un nuevo punto aleatorio y repita el proceso para identificar el siguiente grupo

6.2. Análisis de Componentes Principales

PCA (Figura 110)es una técnica de reducción de dimensionalidad comúnmente utilizada y muy efectiva, que a menudo forma parte de la etapa de preprocesamiento de varios modelos y técnicas de aprendizaje automático. PCA reduce la dispersión en el conjunto de datos al separar los datos en una serie de componentes donde cada componente representa una fuente de información dentro de los datos. Como su nombre sugiere, el primer componente producido en PCA, el componente principal, comprende la mayoría de la información o varianza dentro de los datos. El componente principal a menudo se puede considerar como la contribución de la mayor cantidad de información interesante además de la media. Con cada componente subsiguiente, se contribuye menos información pero con más sutileza a los datos comprimidos. Si consideramos todos estos componentes



(Figura 110 Data de 2D Fuente: Elaboración propia)



(*Figura 111 Transformar los datos de 2D a 1D Fuente: Elaboración propia*)

7. Ejemplo de Aplicaciones de Data Science en empresa

7.1 Introducción de dataset

El dataset (Figura 112) es los datos de la compra de los clientes en 2023 en una empresa alimentaria en línea. Se analizan los datos utilizando Python para saber qué decisiones la empresa puede hacer para conseguir la mayor venta en el futuro.

order_id	user_id	order_num	order_dow	order_hou	days_since	product_id	add_to_cart_order	reordered	department	department	product_name
2425083	49125	1	2	18		17	1	0	13	pantry	baking ingredients
2425083	49125	1	2	18		91	2	0	16	dairy eggs	soy lactosefree
2425083	49125	1	2	18		36	3	0	16	dairy eggs	butter
2425083	49125	1	2	18		83	4	0	4	produce	fresh vegetables
2425083	49125	1	2	18		83	5	0	4	produce	fresh vegetables
2425083	49125	1	2	18		91	6	0	16	dairy eggs	soy lactosefree
2425083	49125	1	2	18		120	7	0	16	dairy eggs	yogurt
2425083	49125	1	2	18		59	8	0	15	canned gooc	canned meals beans
2425083	49125	1	2	18		35	9	0	12	meat seafoc	poultry counter
1944304	162867	1	3	17		37	1	0	1	frozen	ice cream ice
1944304	162867	1	3	17		24	2	0	4	produce	fresh fruits
1944304	162867	1	3	17		83	3	0	4	produce	fresh vegetables
1944304	162867	1	3	17		84	4	0	16	dairy eggs	milk
1944304	162867	1	3	17		91	5	0	16	dairy eggs	soy lactosefree
1944304	162867	1	3	17		24	6	0	4	produce	fresh fruits
1944304	162867	1	3	17		24	7	0	4	produce	fresh fruits
1944304	162867	1	3	17		24	8	0	4	produce	fresh fruits
1944304	162867	1	3	17		21	9	0	16	dairy eggs	packaged cheese
1944304	162867	1	3	17		112	10	0	3	bakery	bread
1944304	162867	1	3	17		24	11	0	4	produce	fresh fruits
1944304	162867	1	3	17		24	12	0	4	produce	fresh fruits
1944304	162867	1	3	17		24	13	0	4	produce	fresh fruits
1201011	147243	14	0	16	3	94	1	0	7	beverages	tea
1201011	147243	14	0	16	3	83	2	0	4	produce	fresh vegetables

(Figura 112: Una parte de los datos)

Si analizamos los datos por horizontal, podemos tener los combinaciones como los siguientes:

1. **order_number y order_hour_of_day**: Puede analizar la variación de la cantidad de pedidos a lo largo del tiempo, comprender los niveles de pedidos en diferentes momentos del día y ajustar las estrategias de marketing o la gestión de la cadena de suministro.
2. **product_name y order_number:** Puede verificar la cantidad de pedidos para diferentes productos, comprender qué productos son populares y cuáles tienen una demanda baja, lo que puede ayudar en la gestión de inventario y la estrategia de productos.

3. **product_name y order_hour_of_day:** Puede analizar las ventas de diferentes productos en diferentes momentos del día, identificar los picos y valles de ventas y optimizar las estrategias de lanzamiento de productos y promociones.
4. **product_name y department:** Puede entender el desempeño de ventas de diferentes departamentos o categorías de productos, lo que ayuda en la gestión de productos y la segmentación del mercado.
5. **order_dow y order_hour_of_day:** Puede analizar la cantidad de pedidos en diferentes días de la semana y momentos del día, identificar los picos y valles de ventas semanales y diarios para planificar la asignación de personal y recursos.
6. **product_name y reordered:** Puede identificar los productos que tienen una alta tasa de recompra, lo que puede indicar que estos productos son populares y confiables entre los clientes.
7. **department y reordered:** Puede analizar la tasa de recompra de productos en diferentes departamentos, comprender qué departamentos tienen una demanda estable y leal.
8. **user_id y days_since_prior_order:** Puede comprender los intervalos de tiempo entre los pedidos de diferentes usuarios, lo que ayuda a determinar la frecuencia de compra y la lealtad de los clientes.
9. **department y add_to_cart_order:** Puede analizar el orden en que los productos de diferentes departamentos se agregan al carrito de compras, lo que ayuda a comprender las preferencias y hábitos de compra de los usuarios para diferentes productos.
10. **Frecuencia de compra de usuarios y tasa de recompra de productos:** Al analizar las columnas user_id y reordered, puede comprender la frecuencia de compra de los usuarios y la tasa de recompra de diferentes productos. Esto ayuda a comprender la lealtad de los usuarios y la popularidad de los productos, lo que a su vez permite optimizar las estrategias de retención de clientes y marketing.
11. **Análisis de tendencias de ventas:** Al combinar las columnas order_dow (día de la semana) y order_hour_of_day (hora del día), puede analizar el volumen de pedidos y las ventas en diferentes momentos del día. Esto ayuda a comprender los patrones y tendencias de compra de los consumidores, lo que a su vez permite planificar mejor el inventario, las actividades de marketing y el

servicio al cliente.

Si analizamos los datos por vertical, podemos tener los combinaciones como los siguientes:

1.Efecto sinérgico entre departamentos: Al combinar la columna department, puede analizar el efecto sinérgico entre diferentes departamentos. Por ejemplo, puede calcular la frecuencia con la que los productos de un departamento se compran junto con los productos de otros departamentos para descubrir asociaciones potenciales y oportunidades de venta cruzada.

2. **Combinación de productos**: se calcula las combinación de los productos. Eso es más importante para analizar cuáles productos podemos recomendar a nuestros clientes después de comprar un producto o podemos hacer una combinación para promocionar los productos.

Luego se usa el aprendizaje supervisado para predecir los comportamientos de clientes en el futuro y para clasificar los clientes también.

1.Procesar datos

```
# En esta parte, se procesan los datos para obtener la información básica de los datos
```

```
import numpy as np
import pandas as pd
df=pd.read_csv('Ebusiness.csv')
df.head()
```

	order_id	user_id	order_number	order_dow	order_hour_of_day	days_since_prior_order	product_id	add_to_cart_order	reordered
0	2425083	49125	1	2	18	NaN	17	1	0
1	2425083	49125	1	2	18	NaN	91	2	0
2	2425083	49125	1	2	18	NaN	36	3	0
3	2425083	49125	1	2	18	NaN	83	4	0
4	2425083	49125	1	2	18	NaN	83	5	0

(Figura 113 :Introducir los datos Fuente: Elaboración propia)

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2019501 entries, 0 to 2019500
Data columns (total 12 columns):
 #   Column                  Dtype
---  ------                  -----
 0   order_id                int64
 1   user_id                 int64
 2   order_number            int64
 3   order_dow               int64
 4   order_hour_of_day       int64
 5   days_since_prior_order  float64
 6   product_id              int64
 7   add_to_cart_order       int64
 8   reordered               int64
 9   department_id           int64
 10  department              object
 11  product_name            object
dtypes: float64(1), int64(9), object(2)
memory usage: 184.9+ MB
```

(Figura 114 :Información de los datos Fuente: Elaboración propia)

```
df.describe()
```

order_id	user_id	order_number	order_dow	order_hour_of_day	days_since_prior_order
19501e+06	2.019501e+06	2.019501e+06	2.019501e+06	2.019501e+06	1.895159e+06
07013e+06	1.030673e+05	1.715138e+01	2.735367e+00	1.343948e+01	1.138603e+01
59832e+05	5.949117e+04	1.752576e+01	2.093882e+00	4.241008e+00	8.970980e+00
00000e+01	2.000000e+00	1.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00
26490e+05	5.158400e+04	5.000000e+00	1.000000e+00	1.000000e+01	5.000000e+00
05004e+06	1.026900e+05	1.100000e+01	3.000000e+00	1.300000e+01	8.000000e+00
59031e+06	1.546000e+05	2.400000e+01	5.000000e+00	1.600000e+01	1.500000e+01
21080e+06	2.062090e+05	1.000000e+02	6.000000e+00	2.300000e+01	3.000000e+01

(Figura 115 : Descripción de los datos Fuente: Elaboración propia)

```
df.isnull().sum()
```

```
order_id                      0
user_id                       0
order_number                  0
order_dow                     0
order_hour_of_day             0
days_since_prior_order    124342
product_id                    0
add_to_cart_order             0
reordered                     0
department_id                 0
department                    0
product_name                  0
dtype: int64
```

(Figura 116: Calcular los datos perdidos Fuente: Elaboración propia)

```
# Se usa el método de interpolación, porque hay muchos datos perdidos
df['days_since_prior_order'].fillna(-1, inplace=True)
```

(Figura 117: Resolver los datos perdidos con interpolación Fuente: Elaboración propia)

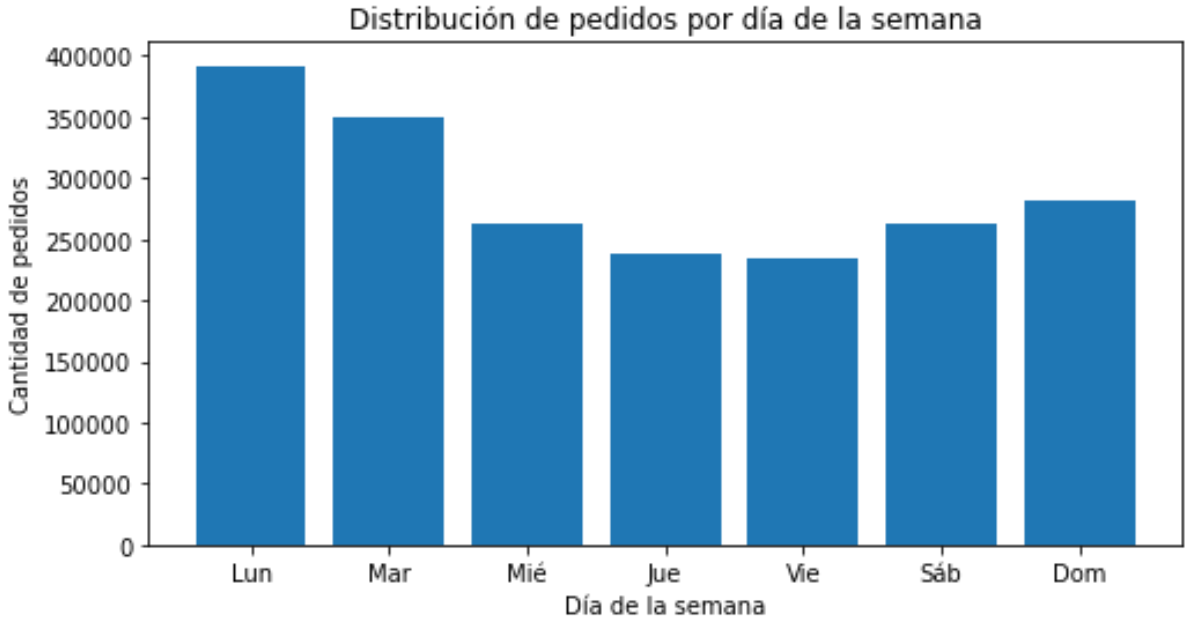
7.2 Análisis de comportamiento de clientes

7.2.1 Distribución de pedidos

Al analizar el día de la semana (order_dow) y las horas (order_hour_of_day) del pedido, se pueden comprender los hábitos de compra del usuario y las preferencias de tiempo de compra dentro de una semana. Esto puede ayudar a determinar el mejor momento para las promociones y optimizar las cadenas de suministro. Eso es muy importante en la promoción también, porque el horario de publicidad influye los pedidos.

```
import pandas as pd
import matplotlib.pyplot as plt

# Estadísticas de la distribución de los días de la semana de los pedidos
order_dow_counts = df['order_dow'].value_counts().sort_index()

# Estadísticas de la distribución de las horas del día de los pedidos
order_hour_counts = df['order_hour_of_day'].value_counts().sort_index()

# Gráfico de la distribución de los días de la semana de los pedidos
plt.figure(figsize=(8, 4))
plt.bar(order_dow_counts.index, order_dow_counts.values)
plt.xlabel('Día de la semana')
plt.ylabel('Cantidad de pedidos')
plt.title('Distribución de pedidos por día de la semana')
plt.xticks(order_dow_counts.index, ['Lun', 'Mar', 'Mié', 'Jue', 'Vie', 'Sáb', 'Dom'])
plt.show()

# Gráfico de la distribución de las horas del día de los pedidos
plt.figure(figsize=(12, 4))
plt.bar(order_hour_counts.index, order_hour_counts.values)
plt.xlabel('Hora del día')
plt.ylabel('Cantidad de pedidos')
plt.title('Distribución de pedidos por hora del día')
plt.show()
```

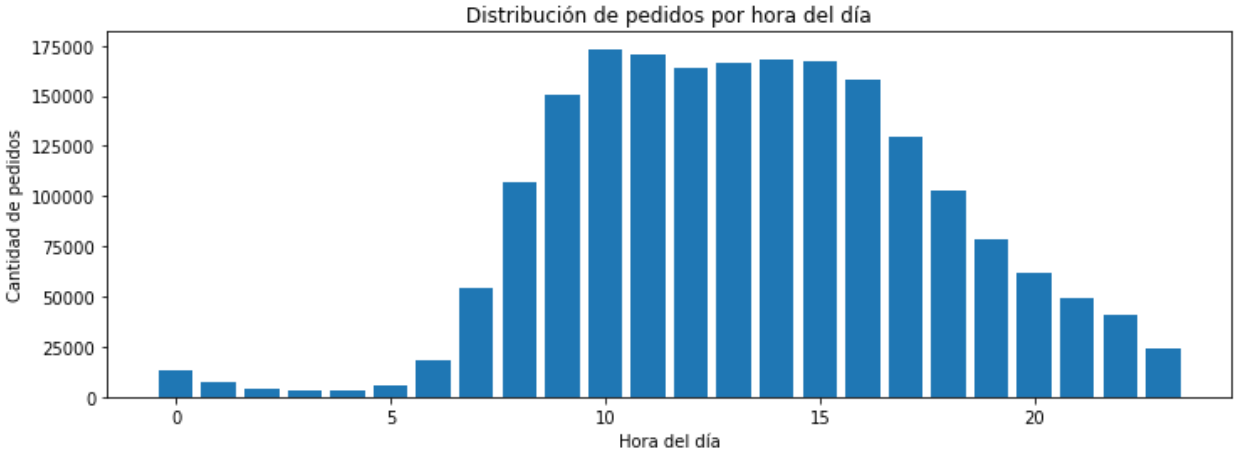
(Figura 118: Trazar histograma de distribución de pedidos Fuente: Elaboración propia)



(Figura 119: Distribución de pedidos por día Fuente: Elaboración propia)

El gráfico（Figura 119) muestra que dentro una semana, el lunes y el martes la

empresa recibe más pedidos. Según el análisis, la empresa puede preparar los productos para estos días.



(Figura 120: Distribución de pedidos por hora del día Fuente: Elaboración propia)

Se puede ver que la hora de la mayor venta es las 9, dentro 24 horas, de 9 a 16 es un período de venta（Figura 120）. Eso es importante para promocionar los productos según la hora del día.

7.2.2 Frecuencia de compras

```python
import pandas as pd

# Estadísticas de la frecuencia de pedidos y cantidad de pedidos por usuario
order_frequency = df.groupby('user_id')['order_number'].nunique()
order_quantity = df.groupby('user_id')['order_number'].count()

# Definir criterios de clasificación
frequency_criteria = [1, 3, 5]  # Criterios de frecuencia de pedidos, por ejemplo: menos
quantity_criteria = [10, 20, 30]  # Criterios de cantidad de pedidos, por ejemplo: menos

# Asignar clasificación a cada usuario según los criterios establecidos
def get_frequency_level(frequency):
    if frequency <= frequency_criteria[0]:
        return 'Baja frecuencia'
    elif frequency <= frequency_criteria[1]:
        return 'Frecuencia media'
    else:
        return 'Alta frecuencia'

def get_quantity_level(quantity):
    if quantity <= quantity_criteria[0]:
        return 'Baja cantidad'
    elif quantity <= quantity_criteria[1]:
        return 'Cantidad media'
    else:
        return 'Alta cantidad'

# Asignar clasificación de frecuencia y cantidad de pedidos a cada usuario
df['Nivel de frecuencia'] = order_frequency.apply(get_frequency_level)
df['Nivel de cantidad'] = order_quantity.apply(get_quantity_level)

# Estadísticas de la cantidad de usuarios en cada nivel de clasificación
frequency_counts = df['Nivel de frecuencia'].value_counts()
quantity_counts = df['Nivel de cantidad'].value_counts()

print('Cantidad de usuarios por nivel de frecuencia:')
print(frequency_counts)
print('\nCantidad de usuarios por nivel de cantidad:')
```

(Figura 121: Clasificar los clientes según la cantidad y la frecuencia Fuente: Elaboración propia)

```
Cantidad de usuarios por nivel de frecuencia:
Baja frecuencia     57803
Frecuencia media    35867
Alta frecuencia     11603
Name: Nivel de frecuencia, dtype: int64

Cantidad de usuarios por nivel de cantidad:
Baja cantidad     43466
Alta cantidad     33142
Cantidad media    28665
Name: Nivel de cantidad, dtype: int64
```

(Figura 122: La cantidad y la frecuencia de compras de clientes Fuente: Elaboración propia)

Se ve que la mayoría de clientes tiene baja frecuencia y cantidad, pero también hay una parte de clientes de frecuencia alta y media. La empresa tiene que mejorar la lealtad de los clientes.

7.2.3 Preferencia de clientes

Se trata de cuál producto que los clientes quieren añadir al carrito al primero. En esta parte se calcula el orden promedio de añadir los productos al carrito para saber la preferencia de nuestros clientes.

```
# El orden promedio de departamento que se añade al carrito
department_order = df.groupby('department')['add_to_cart_order'].mean()

# Imprimir el resultado
print(department_order)
```

```
department
alcohol            5.167497
babies            10.489977
bakery             8.072469
beverages          6.931194
breakfast          9.237103
bulk               8.436474
canned goods       9.891451
dairy eggs         7.529466
deli               8.718531
dry goods pasta   10.200337
frozen             9.052431
household          8.486716
international      9.983033
meat seafood       8.540670
missing            9.211202
other              8.126339
pantry             9.633302
personal care      8.822244
pets               7.627806
produce            8.034465
snacks             9.206401
Name: add_to_cart_order, dtype: float64
```

(Figura 123: El orden promedio de los productos que añaden al carrito Fuente: Elaboración propia)

Según el resultado (Figura 123) , se observa que el alcohol es un producto que se añade al carrito muy rápido,es decir, es un producto preferido de los clientes, sin embargo, no es el producto de la mayor venta, hay que reflexionar cómo hace los

clientes comprar sus productos preferidos.

Recomendaciones:

Recomendación de productos: al analizar el orden de producto del carrito de compras del cliente, puede comprender los productos que les interesan a los clientes y sus hábitos de compra. Con base en esta información, la empresa puede implementar una estrategia personalizada de recomendación de productos, recomendando a los clientes otros productos que les puedan gustar. Esto ayuda a aumentar las conversiones de ventas y la satisfacción del cliente.

Estrategia de promoción: el pedido del carrito también puede guiar la estrategia de promoción de una empresa. Si un producto se agrega a menudo al carrito de compras como el primer o último artículo, la empresa puede considerar diseñar promociones para estos productos para atraer a más clientes a comprar. Además, el orden de los productos en un carrito de compras puede arrojar luz sobre el proceso de decisión de compra de un cliente, lo que ayuda a las empresas a optimizar la forma en que se muestran y presentan las promociones.

Gestión de inventario y decisiones de reabastecimiento: el pedido del carrito de compras también puede proporcionar información útil sobre la demanda de productos. Si un producto se agrega con frecuencia a los carritos de compras pero está agotado con frecuencia, las empresas pueden usar los datos del carrito para tomar decisiones más precisas de administración de inventario y reabastecimiento para satisfacer la demanda de los clientes y evitar la pérdida de ventas.

Análisis del comportamiento del usuario: el pedido del carrito de la compra es una manifestación del comportamiento del cliente. Al analizar los datos del carrito de la compra, podemos obtener información sobre las preferencias del cliente, los hábitos de compra y las intenciones de compra. Esto ayuda a las empresas a comprender mejor los grupos de clientes objetivo y a realizar actividades de marketing y

posicionamiento de mercado perfeccionadas en función de estos conocimientos.

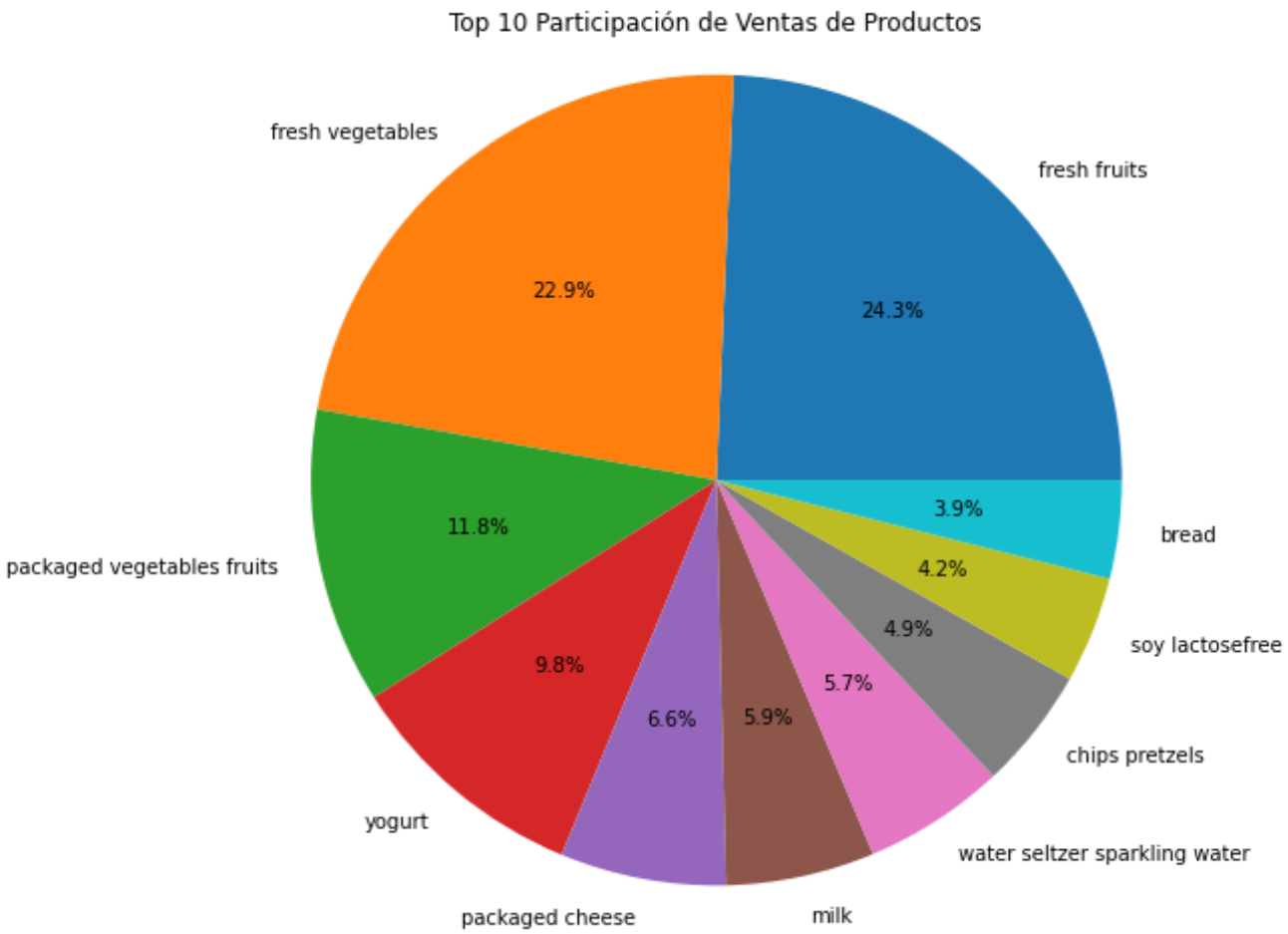
7.3 Análisis de productos

7.3.1 Productos más vendidos(Star product)

Al analizar la columna "product_name", puede determinar los productos más vendidos. Esto puede ayudar a comprender las preferencias y necesidades de los usuarios, y ajustar las estrategias de inventario y adquisición para satisfacer la demanda del mercado.

```
# Calcular la cantidad de compras por categoría de producto
category_counts = df.groupby(['department_id', 'department'])['order_id'].count().reset_index()
category_counts = category_counts.sort_values('order_id', ascending=False)

# Obtener las 10 categorías principales
top_categories = category_counts.head(10)
print(top_categories)

# Graficar las preferencias de categoría de producto
import matplotlib.pyplot as plt

plt.bar(top_categories['department'], top_categories['order_id'])
plt.xlabel('Categoría de Producto')
plt.ylabel('Cantidad de Compras')
plt.title('Top 10 Categorías de Producto Preferidas')
plt.xticks(rotation=45)
plt.show()
```

(Figura 124: Calcular Top 10 catelogías más vendidas Fuente: Elaboración propia)



(Figura 125 Top 10 catelogías más vendidas Fuente: Elaboración propia)

En el gráfico (Figura 125) , se ve el producto más vendido es fruta fresca, el siguiente es verdura fresca y agua sparkling, luego es verduras y frutas empaquetadas.

Según los productos más vendidos, la empres puede:

Estrategia de ventas y toma de decisiones: desarrollar estrategias de ventas y tomar decisiones. Pueden ajustar estrategias como posicionamiento en el mercado, actividades promocionales y administración de inventario para aumentar aún más las ventas y la participación en el mercado.

Gestión de inventario: La empresa puede mejorar la gestión de inventario. Al saber qué productos son los más vendidos, las empresas pueden planificar y administrar adecuadamente los niveles de inventario para evitar el exceso o la escasez, aumentando así la rotación de inventario y reduciendo los costos de inventario.

Desarrollo e innovación de productos: La empresa puede obtener información importante sobre la demanda del mercado y las preferencias de los consumidores. Estos conocimientos pueden guiar el desarrollo de productos y la dirección de la innovación de una empresa, ayudándoles a diseñar y lanzar productos más competitivos que satisfagan las necesidades de los clientes y mejoren su posición en el mercado.

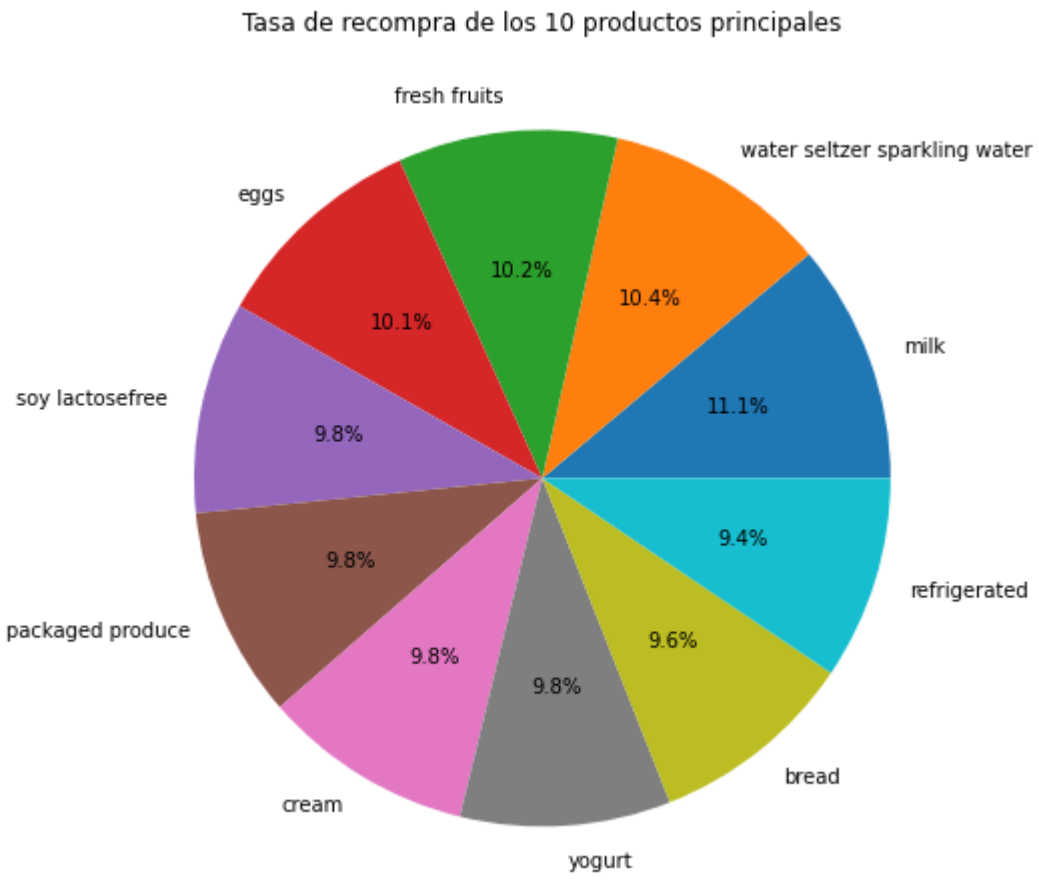
Análisis de la competencia en el mercado: La empresa puede destacar sus ventajas de competencia en el mercado, también se puede diferenciar con los competidores y promocionar los productos para tener la mayor venta.

Actividades de marketing y promoción: conocer los diez productos más vendidos de la empresa puede orientar la planificación de las actividades de marketing y promoción. La empresa puede poner más recursos e inversiones en los productos más vendidos, llevar a cabo una promoción de marketing dirigida y aumentar la popularidad y las ventas del producto.

7.3.2 Productos más recomprados

```
import pandas as pd
import matplotlib.pyplot as plt

# Calcular la tasa de recompra de productos
product_reorder_ratio = df.groupby('product_id')['reordered'].mean().reset_index()
product_reorder_ratio.columns = ['product_id', 'reorder_ratio']

# Combinar la información del nombre del producto
product_reorder_ratio_with_name = pd.merge(product_reorder_ratio, df[['product_id', 'product_name']],

# Ordenar los productos según la tasa de recompra
sorted_products = product_reorder_ratio_with_name.sort_values(by='reorder_ratio', ascending=False)

# Imprimir los resultados
print(sorted_products[['product_id', 'product_name', 'reorder_ratio']])

# Obtener la tasa de recompra de cada producto
reorder_ratio = df.groupby('product_id')['reordered'].mean()

# Obtener la información única de productos
unique_products = df[['product_id', 'product_name']].drop_duplicates()

# Combinar la tasa de recompra con la información única de productos
unique_products = unique_products.merge(reorder_ratio, on='product_id')

# Ordenar los productos según la tasa de recompra en orden descendente
unique_products = unique_products.sort_values(by='reordered', ascending=False)

# Mantener los 10 primeros productos
top_10_products = unique_products.head(10)

# Crear un gráfico de pastel
plt.figure(figsize=(8, 8))
plt.pie(top_10_products['reordered'], labels=top_10_products['product_name'], autopct='%1.1f%%')
plt.title('Tasa de recompra de los 10 productos principales')
plt.show()
```

(Figura 126 Calcular Top 10 productos más vendidas Fuente: Elaboración propia)



(Figura 127 Top 10 productos más vendidas Fuente: Elaboración propia)

En el gráfico (Figura 127) , se ve el producto más recomprado es la lecha, el siguiente es agua seltzer y agua sparkling, luego es frutos frescos.

Según el resultado de los productos más recomprados, las estrategias y decisiones que la empresa puede hacer son las siguientes:

Evaluación de la lealtad del cliente: La recompra de productos es uno de los indicadores importantes para medir la lealtad del cliente. Al analizar qué productos compran repetidamente los clientes, es posible evaluar la lealtad de los clientes hacia la marca y los productos de la empresa. Una tasa de recompra más alta indica que los clientes están más satisfechos con el producto y tienden a mantener una relación transaccional a largo plazo con la empresa. Esto ayuda a evaluar la lealtad del cliente e informa la estrategia de retención de clientes de la empresa.

Ventas de productos y gestión de inventario: productos de recompra proporciona información importante sobre las tendencias y la demanda de ventas de productos. Mediante el análisis de los datos de ventas de los productos recomprados, es posible determinar qué productos tienen una demanda de mercado estable y un potencial de ventas, lo que orienta las estrategias de gestión de ventas e inventario de la empresa. Para los productos que se compran con frecuencia, las empresas pueden aumentar el inventario e impulsar las ventas para satisfacer la demanda de los clientes y aumentar los ingresos.

Mejora e innovación de productos: al comprender las características y preferencias de los clientes que vuelven a comprar productos, las empresas pueden obtener comentarios útiles sobre los productos. Al analizar las características, la frecuencia y los comentarios de los clientes sobre los productos de recompra, se pueden identificar las fortalezas del producto y las posibles áreas de mejora. Esto ayuda a la empresa a optimizar el diseño del producto, mejorar la calidad del producto, satisfacer las necesidades de los clientes y desarrollar nuevos productos innovadores para aumentar la competitividad en el mercado.

Segmentación de clientes y marketing personalizado: la segmentación efectiva de clientes se puede llevar a cabo analizando las características de los grupos de clientes de productos recomprados. Diferentes productos de recompra pueden atraer a diferentes grupos de clientes, con diferentes características y preferencias. Al dividir a los clientes en diferentes segmentos, las empresas pueden diseñar estrategias de marketing personalizadas para cada grupo, proporcionar recomendaciones y promociones de productos personalizados y aumentar la participación del cliente y la intención de compra.

7.3.3 Período de mayor venta de productos

Las demandas de consumidores se varian según la hora. Por ejemplo, por la mañana, pueden comprar algo para el desayuno y por la tarde, es posible que compren tapas. Por eso hay que saber el período cuando se vende la mayoría de cada producto.

```python
import pandas as pd

# Calcular el período más vendido para cada producto
best_selling_periods = df.groupby('product_id').apply(lambda x: x.groupby(['order_dow', 'order_hour_of_d
best_selling_periods = best_selling_periods.reset_index().rename(columns={0: 'period'})

# Eliminar valores duplicados
best_selling_periods = best_selling_periods.drop_duplicates()

# Combinar el DataFrame best_selling_periods con el DataFrame products para obtener el nombre del produc
best_selling_periods = pd.merge(best_selling_periods, df[['product_id', 'product_name']], on='product_id'

# Mostrar el período más vendido para cada producto
print(best_selling_periods)
```

(Figura 128 Top 10 calcular el período de mayor de productos Fuente: Elaboración propia)

	product_id	period	product_name
0	1	(1, 9)	prepared soups salads
1	1	(1, 9)	prepared soups salads
2	1	(1, 9)	prepared soups salads
3	1	(1, 9)	prepared soups salads
4	1	(1, 9)	prepared soups salads
...	...	...	...
2019496	134	(5, 10)	specialty wines champagnes
2019497	134	(5, 10)	specialty wines champagnes
2019498	134	(5, 10)	specialty wines champagnes
2019499	134	(5, 10)	specialty wines champagnes
2019500	134	(5, 10)	specialty wines champagnes

[2019501 rows x 3 columns]

(Figura 129 El período de mayor de productos Fuente: Elaboración propia)

Por ejemplo, se ve que la sopa preparada se vende al mayor durante 1 y 9 , y las champanes se venden muy bien desde las 5 hasta las 10.

Estrategias que la empresa puede hacer:

Optimización de la estrategia de marketing: conocer el período de mayor venta de

cada producto puede ayudar a las empresas a optimizar sus estrategias de marketing. Al centrar las promociones, la publicidad y el marketing en los intervalos de tiempo de mayor venta, puede aumentar la exposición del producto y las oportunidades de venta. Las empresas pueden formular planes de marketing específicos basados en los períodos de mayor venta de productos para atraer a más clientes y aumentar las ventas.

Gestión de inventario: conocer los períodos de tiempo de mayor venta de un producto ayuda a una empresa en la gestión de inventario. Las empresas pueden planificar y administrar el inventario de manera racional en función de las tendencias de ventas de productos y los niveles de demanda durante los períodos de mayor venta. Cree un inventario antes de sus tiempos de mayor venta para satisfacer la demanda de los clientes y evitar desabastecimientos o excesos. Esto ayuda a aumentar la rotación de inventario, reducir los costos de inventario y garantizar la estabilidad del suministro de productos.

Planificación de la producción y asignación de recursos: el período de mayor venta de un producto puede guiar la planificación de la producción y la asignación de recursos de la empresa. De acuerdo con la demanda de ventas de productos en el período de mayor venta, la empresa puede organizar el plan de producción para garantizar que los pedidos de los clientes se puedan cumplir a tiempo. Además, la empresa puede priorizar la asignación de recursos (como mano de obra, equipos, etc.) para la producción de productos durante el período de mayor venta para mejorar la eficiencia de la producción y la flexibilidad del suministro de productos.

Desarrollo de nuevos productos y expansión del mercado: conocer el período de mayor venta de un producto puede ayudar a una empresa en su estrategia de **desarrollo de nuevos productos y expansión del mercado.** Al analizar el ciclo de demanda del mercado y los cambios estacionales de diferentes productos, la empresa puede lanzar nuevos productos con oportunidad y potencial de mercado de acuerdo con las características del producto del período de mayor venta. Además, las empresas pueden considerar expandir las ventas de productos a otras regiones o mercados objetivo para maximizar el uso de los intervalos de tiempo más vendidos del producto.

Optimización de la cadena de suministro: conocer los períodos de mayor venta de un producto puede ayudar a las empresas a optimizar la gestión de la cadena de suministro. Las empresas pueden trabajar con proveedores, socios de logística, etc. para garantizar la disponibilidad del producto y las capacidades de entrega durante los intervalos de tiempo de mayor venta. Al optimizar el proceso de la cadena de suministro y reducir los cuellos de botella y los retrasos en la cadena de suministro, se puede mejorar la velocidad de entrega del producto y la satisfacción del cliente.

7.3.4 Combinación de productos

Cuando compramos productos, siempre vemos que hay productos recomendados para nosotros. Aquí se calculan las combinaciones de los productos (Figura 130) . Eso es más importante para analizar cuáles productos podemos recomendar a nuestros clientes después de comprar un producto o podemos hacer una combinación para promocionar los productos.

Por ejemplo, cuando compramos el pan para desayuno, queremos comprar la leche también.

```python
import pandas as pd
from itertools import combinations
from collections import Counter

# Crear un diccionario vacío para almacenar el recuento de combinaciones de productos
combination_counts = Counter()

# Iterar sobre cada orden y su lista de IDs de productos
for order_id, group in df.groupby('order_id'):
    product_ids = group['product_id'].tolist()

    # Generar todas las posibles combinaciones de productos utilizando combinations
    product_combinations = list(combinations(product_ids, 2))  # Selecciona 2 productos para formar las combinaci

    # Actualizar el recuento de combinaciones de productos
    combination_counts.update(product_combinations)

# Seleccionar las combinaciones de productos más frecuentes
frequent_combinations = {combination: count for combination, count in combination_counts.items() if count > 1}

# Imprimir los resultados
for combination, count in frequent_combinations.items():
    print(f"Combination: {combination}, Count: {count}")
```

(Figura 130 Calcular la combinación de productos Fuente: Elaboración propia)

```
Combination: (24, 83), Count: 239305
Combination: (24, 16), Count: 25865
Combination: (24, 24), Count: 211266
Combination: (24, 92), Count: 25619
Combination: (24, 59), Count: 17725
Combination: (24, 53), Count: 13533
Combination: (24, 122), Count: 4575
Combination: (83, 16), Count: 38717
Combination: (83, 24), Count: 170894
Combination: (83, 83), Count: 286160
Combination: (83, 92), Count: 15918
Combination: (83, 59), Count: 20776
Combination: (83, 53), Count: 10785
Combination: (83, 122), Count: 4960
Combination: (16, 24), Count: 17895
Combination: (16, 83), Count: 31808
Combination: (16, 16), Count: 5948
Combination: (16, 92), Count: 1155
Combination: (16, 59), Count: 2037
```

(Figura 131 combinación de productos Fuente: Elaboración propia)

7.4 La aplicación del aprendizaje supervisado y el aprendizaje no supervisado en la empresa

7.4.1 Predicción de recompra de clientes basada en Regresión logística

Codificación de características categóricas

En esta parte, se cambian las variables no numéricas por las variables numéricas usando la codificación one-hot（Figura 132）.Esta codificación se usa comúnmente en el aprendizaje automático para tratar con variables categóricas.

Codificación de características categóricas

```
import pandas as pd

# Extraer las columnas de características que deben codificarse de forma one-hot
categorical_features = ['product_name', 'department','Nivel de frecuencia','Nivel de cantidad',

# Aplicar la codificación one-hot
data_encoded = pd.get_dummies(df, columns=categorical_features)

# Imprimir los datos codificados
print(data_encoded.head())
```

```
   order_id  user_id  order_number  order_dow  order_hour_of_day  \
0   2425083    49125             1          2                 18
1   2425083    49125             1          2                 18
2   2425083    49125             1          2                 18
3   2425083    49125             1          2                 18
4   2425083    49125             1          2                 18

   days_since_prior_order  product_id  add_to_cart_order  reordered  \
0                    -1.0          17                  1          0
1                    -1.0          91                  2          0
2                    -1.0          36                  3          0
3                    -1.0          83                  4          0
4                    -1.0          83                  5          0
```

(Figura 132 Codificación de características categóricas

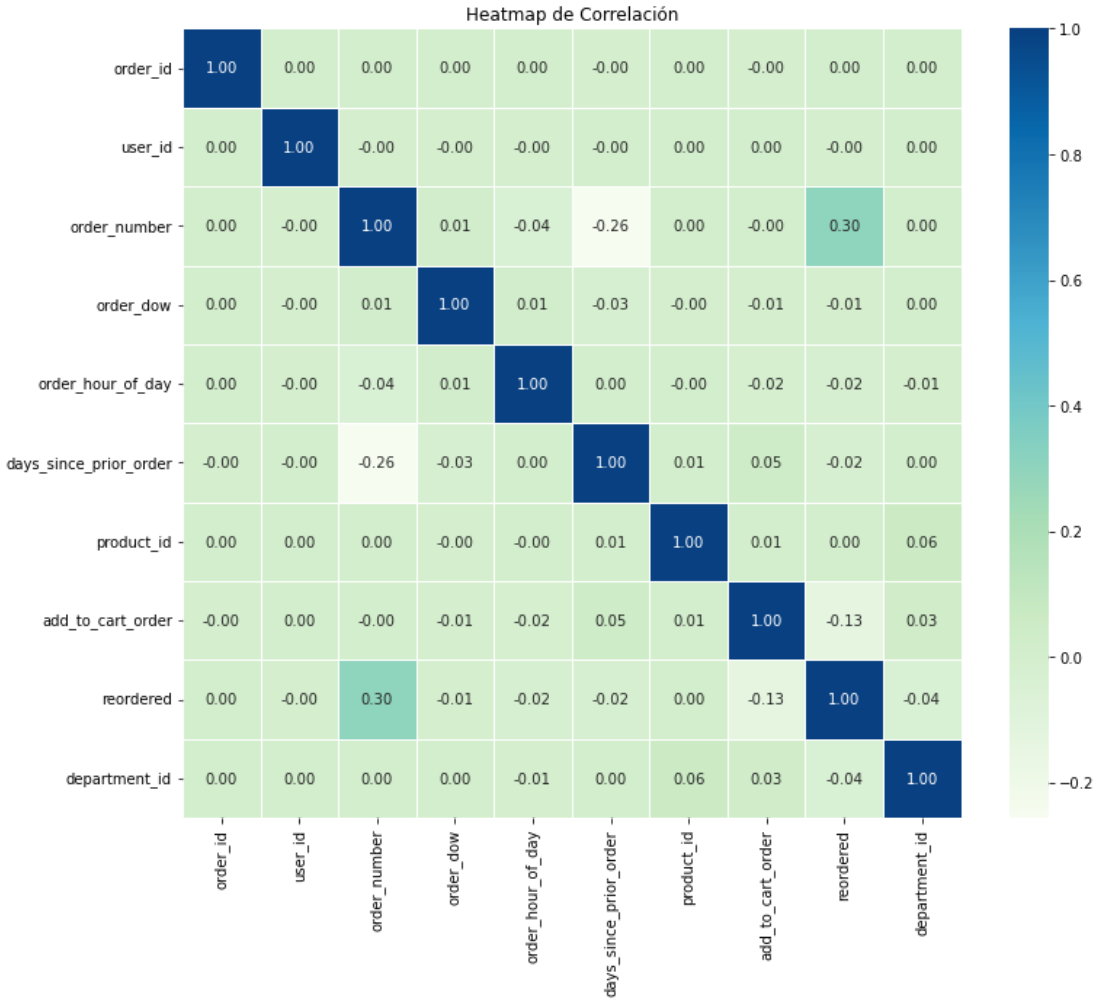
Fuente: Elaboración propia)

Selección de características

En esta parte, se analizan las características de los datos para saber si hay correlación entre las características

Selección de características

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Seleccionar las características para calcular la matriz de correlación
features = ['order_id', 'user_id', 'order_number', 'order_dow', 'order_hour_of_day', 'days_since_
            'product_id', 'add_to_cart_order', 'reordered', 'department_id']

# Extraer las características del DataFrame
X = df[features]

# Calcular la matriz de correlación
correlation_matrix = X.corr()

# Crear el heatmap de correlación con esquema de colores de macaron
plt.figure(figsize=(12, 10))
sns.heatmap(correlation_matrix, annot=True, cmap='GnBu', fmt=".2f", linewidths=0.5)
plt.title('Heatmap de Correlación')
plt.show()
```



(Figura 133 El heatmapa de correlación

Fuente: Elaboración propia)

Según el heatmapa (Figura 133) , no existe multi correlación entre los datos.

Establecer el modelo de regresión logística

Establecer el modelo

```
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

#Realizar la codificación one-hot
data_encoded = pd.get_dummies(df, columns=características_categóricas)

#Separar las características y la variable objetivo
X = data_encoded.drop('reordered', axis=1)
y = data_encoded['reordered']

#Dividir el conjunto de entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
```

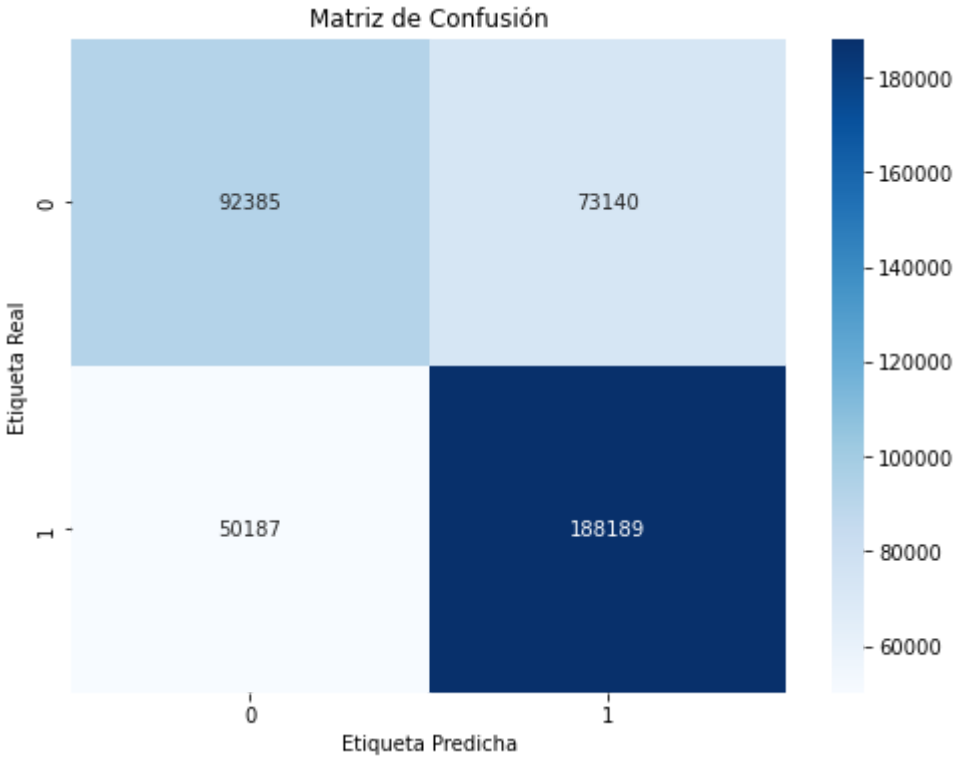
```
(1615600, 171) (403901, 171) (1615600,) (403901,)
```

(Figura 134 Establecer el modelo de regresión logística Fuente: Elaboración propia)

Se dividen los datos en dos partes (Figura 134) , una parte para entrenar y otra parte para probar. Se imprime la forma de las partes de trenamiento y las partes de prueba.

```python
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

# Definir y entrenar el modelo
reg = LogisticRegression()
reg.fit(X_train, y_train)

# Obtener las predicciones del modelo
predicciones = reg.predict(X_test)

# Calcular la matriz de confusión
cm = confusion_matrix(y_test, predicciones)

# Crear el heatmap de la matriz de confusión
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt="d", cmap='Blues')
plt.xlabel('Etiqueta Predicha')
plt.ylabel('Etiqueta Real')
plt.title('Matriz de Confusión')
plt.show()
```

(Figura 135 Crear matriz de confusión Fuente: Elaboración propia)



(Figura 136 Matriz de confusión Fuente: Elaboración propia)

Verdadero positivo(TP): recompran productos cuando se pronostica que recomprarán

Verdadero negativo(TN): no recompran productos cuando se predijo que no recomprarán

Falso positivo(FP): no recompran cuando se pronostica que recomprarán

Falso negativo(FN): recompan cuando se pronostica que no recomprán

Exactitud (Accuracy) = (TP + TN) /(TP+TN+FP+FN) = (92385+188189)/(92385+188189+50187+73140)=0.69

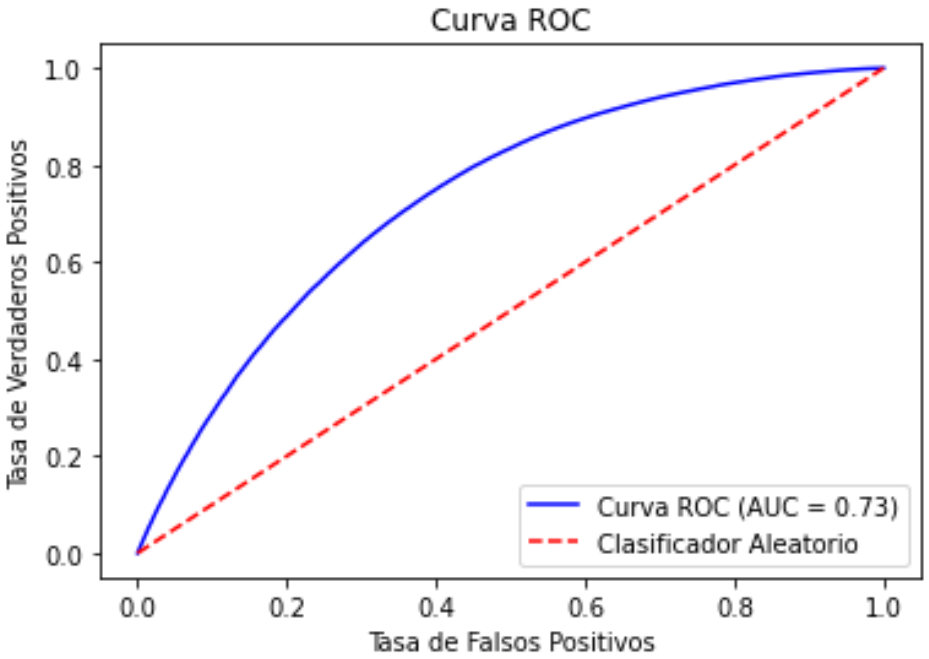
Precisión (Precision) = (TP) /(TP+FP)=92385/(92385+50187)= 0.65

Exhaustividad o recall (Recall)=(TP) /(TP+FN)=92385/(92385+73140) = 0.56

Como se muestra, el verdadero negativo es 188189 casos, lo que sugiere que el modelo es bueno para predecir que los clientes no vuelven a comprar cuando en realidad no van a recomprar. Sin embargo, todavía necesita mejorar la tasa de verdaderos positivos, por lo tanto, predecir con éxito la lluvia mañana (solo 92385 casos).

```python
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

# Definir y entrenar el modelo
reg = LogisticRegression()
reg.fit(X_train, y_train)

# Obtener las probabilidades de las clases positivas
probas = reg.predict_proba(X_test)[:, 1]

# Calcular la curva ROC y el AUC
fpr, tpr, thresholds = roc_curve(y_test, probas)
roc_auc = auc(fpr, tpr)

# Graficar la curva ROC
plt.plot(fpr, tpr, color='blue', label='Curva ROC (AUC = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='red', linestyle='--', label='Clasificador Aleatorio')
plt.xlabel('Tasa de Falsos Positivos')
plt.ylabel('Tasa de Verdaderos Positivos')
plt.title('Curva ROC')
plt.legend(loc='lower right')
plt.show()
```

(Figura 137 Trazar la curva de ROC y AUC Fuente: Elaboración propia)



(Figura 138 La curva de ROC y AUC Fuente: Elaboración propia)

La curva **ROC** (Receiver Operating Characteristic) es una herramienta que se utiliza para evaluar la capacidad de un modelo de clasificación binario para distinguir entre dos clases (positiva y negativa). Representa la tasa de **verdaderos positivos** (sensibilidad) en el eje y y **la tasa de falsos positivos** (1 - especificidad) en el eje x.

La **AUC** (Área bajo la curva ROC) es una medida de la capacidad de un modelo para distinguir entre las clases positiva y negativa. Un modelo con un AUC de 1 es un modelo perfecto que puede distinguir con precisión entre las dos clases en todas las instancias. Un modelo con un AUC de 0,5 es básicamente un modelo aleatorio y no tiene capacidad de discriminación. Por lo tanto, cuanto mayor sea el valor de AUC, mejor será el modelo.
En este caso, la ROC está cerca de 1, AUC es 0.73, es decir, el modelo está bien.

7.4.2 Segmentación de clientes

En este ejemplo, las características seleccionadas son:

order_number: número de pedido. Esta función puede estar relacionada con el comportamiento y las preferencias de compra de los clientes, y puede usarse para identificar grupos de clientes con patrones de compra similares.

days_since_prior_order: El número de días desde el último pedido. Esta función puede reflejar la frecuencia de compra del cliente y puede estar relacionada con la lealtad y los hábitos de compra del cliente.

add_to_cart_order: Agregar pedido al carrito. Esta característica puede estar relacionada con la preferencia del cliente por el producto, por ejemplo, los productos agregados al carrito de compras antes pueden ser más populares.

product_name_encoded: La codificación del nombre del producto. Esta función se obtiene codificando el nombre del producto a través de LabelEncoder y puede estar relacionada con la categoría o atributo del producto.

El propósito de seleccionar estas funciones es agrupar clientes o productos en función de la información de estas funciones y encontrar grupos con comportamientos o atributos de compra similares.

El propósito de este código es usar el algoritmo de agrupamiento de K-means para agrupar las muestras en el conjunto de datos y agregar los resultados del agrupamiento al conjunto de datos como una nueva columna de características. El número de grupos es 4 en este ejemplo, puede ajustarlo según la situación y las necesidades reales. Los resultados de la agrupación pueden ayudarnos a identificar subconjuntos de datos con características o patrones de comportamiento similares para un análisis y una comprensión más profundos.

Segmentación de clientes

```
import pandas as pd
from sklearn.cluster import KMeans

from sklearn.preprocessing import LabelEncoder

# Crear el objetivo de LabelEncoder
label_encoder = LabelEncoder()

# Transfomar el tipo
df['product_name_encoded'] = label_encoder.fit_transform(df['product_name'])

# Seleccionar las características
features = ['order_number', 'days_since_prior_order', 'add_to_cart_order', 'product_name_en


# Extraer los datos de características
X = df[features]

# Instanciar el modelo de clustering K-means
kmeans = KMeans(n_clusters=4, random_state=42)

# Realizar el clustering en los datos
kmeans.fit(X)

# Obtener las etiquetas de clustering para cada muestra
labels = kmeans.labels_

# Agregar las etiquetas de clustering al conjunto de datos
df['cluster'] = labels

# Imprimir la información estadística de cada cluster
cluster_counts = df['cluster'].value_counts()
print("Información de clustering:")
print(cluster_counts)
```

(Figura 139 El uso de K-Means para clasificar clientes Fuente: Elaboración propia)

Luego se traza la clasificación de clientes:

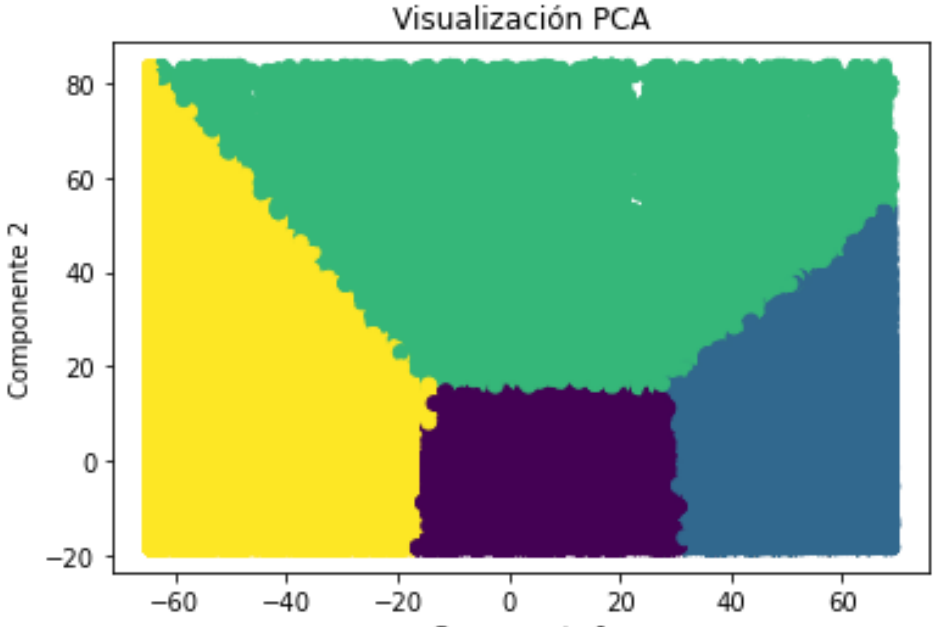
```python
# import pandas as pd
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA

# Seleccionar las características para reducción de dimensionalidad
features = ['order_number', 'days_since_prior_order', 'add_to_cart_order','product_name_encoded']

# Extraer los datos de características
X = df[features]

# Instanciar el modelo de PCA y realizar la reducción de dimensionalidad
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X)

# Crear un nuevo dataframe para almacenar los datos reducidos y las etiquetas de clustering
df_pca = pd.DataFrame(X_pca, columns=['Componente 1', 'Componente 2'])
df_pca['cluster'] = labels

# Graficar el scatter plot en 2D
plt.scatter(df_pca['Componente 1'], df_pca['Componente 2'], c=df_pca['cluster'])
plt.xlabel('Componente 1')
plt.ylabel('Componente 2')
plt.title('Visualización PCA')
plt.show()
```

(Figura 140 Trazar la clasificación de clientes Fuente: Elaboración propia)

```
Información de clustering:
0    763793
3    687364
1    371713
2    196631
Name: cluster, dtype: int64
```

(Figura 141 La clasificación de clientes Fuente: Elaboración propia)

Este código se utiliza para realizar la reducción de la dimensionalidad del análisis de componentes principales (PCA) en los datos y visualizar los datos de dimensionalidad reducida.Los clusters son los siguientes:



(Figura 142 La clasificación de clientes Fuente: Elaboración propia)

8. Conclusión

A través de la aplicación de Data Science en una empresa, se puede concluir que optimiza las decisiones y mejora la gestión de la empresa. Con Data Science, la empresa puede obtener un perfil más claro de sus clientes, incluyendo su comportamiento y preferencias. El análisis de los clientes y los productos puede ser útil para el desarrollo de estrategias de marketing y ventas más efectivas.

Bibliografía

Analytics Vidhya. (2021, January 22). Naive Bayes | Gaussian Naive Bayes with Hyperpameter Tuning in Python. https://www.analyticsvidhya.com/blog/2021/01/gaussian-naive-bayes-with-hyperpameter-tuning/

Data Camp. (n.d.). Python Decision Tree Classification Tutorial: Scikit-Learn DecisionTreeClassifier. Retrieved May 2, 2023, from https://www.datacamp.com/tutorial/decision-tree-classification-python

Galea, A. (2018). Applied Data Science with Python and Jupyter: Use powerful industry-standard tools to unlock new, actionable insights from your data. Packt Publishing Ltd.

Gong, D. (2019, May 29). Top 6 Machine Learning Algorithms for Classification. Towards Data Science. https://towardsdatascience.com/top-machine-learning-algorithms-for-classification-2197870ff501

Gong, D. (2019, June 24). Simple Logistic Regression in Python. Towards Data Science.https://towardsdatascience.com/simple-logistic-regression-using-python-scikit-learn-86bf984f61f1

Gupta, P. (2019). Data Science with Jupyter. BPB Publications.

Koehrsen, W. (2018, May 29). Hyperparameter Tuning the Random Forest in Python. Towards Data Science. https://towardsdatascience.com/hyperparameter-tuning-the-random-forest-in-python-using-scikit-learn-28d2aa77dd74

Kuhn, T. (1962). The Structure of Scientific Revolutions. University of Chicago

Press.

Liu, Y. H. (2017). Python Machine Learning By Example. Packt Publishing Ltd.

Massaron, L., & Mueller, J. P. (2019). Machine learning for dummies. John Wiley & Sons.

Martulandi, A. (2021, February 10). K-Nearest Neighbors in Python + Hyperparameters Tuning. DataDrivenInvestor. https://medium.datadriveninvestor.com/k-nearest-neighbors-in-python-hyperparameters-tuning-716734bc557f

Müller, A. C., & Guido, S. (2016). Introduction to machine learning with Python: a guide for data scientists. O'Reilly Media, Inc.

Rogel-Salazar, J. (2018). Data science and analytics with Python. CRC Press.

Velocity Business Solutions Limited. (2020, March 17). SVM Hyperparameter Tuning using GridSearchCV. Vebuso.com. https://www.vebuso.com/2020/03/svm-hyperparameter-tuning-using-gridsearchcv/

Yan, Y., & Yan, J. (2018). Data Science with Anaconda: Utilize the right mix of tools to create high-performance data science applications. Packt Publishing Ltd.

Lakshmi, N. (2020). Hands-on Supervised Learning with Python: Learn How to Solve Machine Learning Problems with Supervised Learning Algorithms.

Sumathi, V. P. (2022). Machine Learning for Decision Sciences with Case Studies in Python.

Johnston, K. (2019). Applied Unsupervised Learning with Python.

Mysiak, K. (2020, July 16). Explaining DBSCAN Clustering.