P-2

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.linear\_model import LinearRegression

data = {'Experiance' : [1,2,3,4,5,6],

'salary' : [1000,2000,3200,4000,5000,6400]

}

df = pd.DataFrame(data)

x = df[['Experiance']]

y = df['salary']

model = LinearRegression()

model.fit(x,y)

y\_pred = model.predict(x)

plt.scatter(x,y, color='blue',label='Actual Data')

plt.plot(x,y\_pred,color='red', label='Regression Line')

plt.xlabel('Years of Experiance')

plt.ylabel('salary')

plt.title('Simple Linear Regression')

plt.legend()

plt.show()

P-3

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LinearRegression

from sklearn.datasets import fetch\_california\_housing

# Load the California Housing dataset

california\_housing = fetch\_california\_housing()

# Assign the data (features) and target (house prices)

X = pd.DataFrame(california\_housing.data, columns=california\_housing.feature\_names)

y = pd.Series(california\_housing.target)

X = X[['MedInc', 'AveRooms']]

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

model = LinearRegression()

model.fit(X\_train, y\_train)

y\_pred = model.predict(X\_test)

fig = plt.figure(figsize=(10, 7))

ax = fig.add\_subplot(111, projection='3d')

ax.scatter(X\_test['MedInc'], X\_test['AveRooms'], y\_test, color='blue', label='Actual Data')

x1\_range = np.linspace(X\_test['MedInc'].min(), X\_test['MedInc'].max(), 100)

x2\_range = np.linspace(X\_test['AveRooms'].min(), X\_test['AveRooms'].max(), 100)

x1, x2 = np.meshgrid(x1\_range, x2\_range)

z = model.predict(np.c\_[x1.ravel(), x2.ravel()]).reshape(x1.shape)

ax.plot\_surface(x1, x2, z, color='red', alpha=0.5, rstride=100, cstride=100)

ax.set\_xlabel('Median Income')

ax.set\_ylabel('Average Rooms')

ax.set\_zlabel('House Price')

ax.set\_title('Multiple Linear Regression Best Fit Line (3D)')

plt.show()

P-4

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

# Sample time series data

np.random.seed(42)

data = np.random.randn(100).cumsum() # Simulated stock price data

# Convert to Pandas DataFrame

df = pd.DataFrame(data, columns=['Price'])

# Compute 10-day,20-day,2-day Moving Averages

df['SMA\_10'] = df['Price'].rolling(window=10).mean()

df['SMA\_20'] = df['Price'].rolling(window=20).mean()

df['SMA\_2'] = df['Price'].rolling(window=2).mean()

# Plot the results

plt.figure(figsize=(12, 6))

plt.plot(df['Price'], label="Price", linestyle='dotted', color='black')

plt.plot(df['SMA\_10'], label="10-day SMA", color='blue')

plt.plot(df['SMA\_20'], label="20-day SMA", color='red')

plt.plot(df['SMA\_2'], label="2-day SMA", color='yellow')

plt.legend()

plt.title("Moving Averages")

plt.show()

P-5

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import statsmodels.api as sm

from statsmodels.tsa.stattools import adfuller

from statsmodels.graphics.tsaplots import plot\_acf, plot\_pacf

from statsmodels.tsa.arima.model import ARIMA

import pmdarima as pm # Auto-ARIMA

# Generate Sample Time Series Data (Replace with Actual Data)

np.random.seed(42)

data = np.cumsum(np.random.randn(100)) # Simulated non-stationary data

# Convert to Pandas DataFrame

df = pd.DataFrame(data, columns=['Value'])

# Step 1: ADF Test to Check Stationarity

def adf\_test(series):

result = adfuller(series.dropna())

print(f'ADF Statistic: {result[0]}')

print(f'p-value: {result[1]}')

return result[1] # Return p-value

# Determine Differencing Order (d)

p\_value = adf\_test(df['Value'])

d = 0

diff\_series = df['Value'].copy()

while p\_value > 0.05 and d < 2:

diff\_series = diff\_series.diff().dropna()

d += 1

p\_value = adf\_test(diff\_series)

print(f"Optimal differencing order (d): {d}")

#Step 2: Plot ACF & PACF to Determine p and q

plt.figure(figsize=(12, 5))

plt.subplot(121)

plot\_acf(diff\_series.dropna(), ax=plt.gca(), lags=20)

plt.subplot(122)

plot\_pacf(diff\_series.dropna(), ax=plt.gca(), lags=20)

plt.show()

# Step 3: Use Auto-ARIMA with Constraints

auto\_arima\_model = pm.auto\_arima(

diff\_series.dropna(),

seasonal=False,

stepwise=True,

trace=True,

max\_p=3,

max\_q=3,

max\_d=2,

start\_p=1, # Prevent (0,0,0)

start\_q=1, # Prevent (0,0,0)

suppress\_warnings=True

)

p, d, q = auto\_arima\_model.order

print(f"Best (p, d, q): ({p}, {d}, {q})")

# Step 4: Ensure p and q Are Not Both 0

if p == 0 and q == 0:

print("Auto-ARIMA chose (0,0,0), adjusting to (1,0,1) to avoid flat line.")

p, q = 1, 1

# Step 5: Fit ARIMA Model

model = ARIMA(df['Value'].dropna(), order=(p, d, q))

model\_fit = model.fit()

# Step 6: Forecasting

forecast\_steps = 5

forecast = model\_fit.forecast(steps=forecast\_steps)

# Step 7: Avoid Flat Forecast (Adjust if Needed)

last\_value = df['Value'].iloc[-1]

if abs(forecast.iloc[0] - last\_value) > 2 \* df['Value'].std():

print("Forecast is too far from last observed value. Consider adjusting (p, q).")

# Plot Forecast

plt.figure(figsize=(10, 5))

plt.plot(df['Value'], label="Original Data")

plt.plot(range(len(df), len(df) + forecast\_steps), forecast, label="Forecast", color='red')

plt.legend()

plt.title("ARIMA Forecast with Optimized (p, d, q)")

plt.sh

P-6

# import libraries

import pandas as pd

import nltk

from nltk.sentiment.vader import SentimentIntensityAnalyzer

from nltk.corpus import stopwords

from nltk.tokenize import word\_tokenize

from nltk.stem import WordNetLemmatizer

# download nltk corpus (first time only)

import nltk

nltk.download('all')

# Load the amazon review dataset

df =

pd.read\_csv('https://raw.githubusercontent.com/pycaret/pycaret/master/datasets/amazon.csv'

)

df

# create preprocess\_text function

def preprocess\_text(text):

# Tokenize the text

tokens = word\_tokenize(text.lower())

# Remove stop words

filtered\_tokens = [token for token in tokens if token not in stopwords.words('english')]

# Lemmatize the tokens

lemmatizer = WordNetLemmatizer()

lemmatized\_tokens = [lemmatizer.lemmatize(token) for token in filtered\_tokens]

# Join the tokens back into a string

processed\_text = ' '.join(lemmatized\_tokens)

return processed\_text

# apply the function df

df['reviewText'] = df['reviewText'].apply(preprocess\_text)

df

# initialize NLTK sentiment analyzer

analyzer = SentimentIntensityAnalyzer()

# create get\_sentiment function

def get\_sentiment(text):

scores = analyzer.polarity\_scores(text)

sentiment = 1 if scores['pos'] > 0 else 0

return sentiment

# apply get\_sentiment function

df['sentiment'] = df['reviewText'].apply(get\_sentiment)

df

from sklearn.metrics import confusion\_matrix

print(confusion\_matrix(df['Positive'], df['sentiment']))

from sklearn.metrics import classification\_report

print(classification\_report(df['Positive'], df['sentiment']))

P-8

Code for Correlation :

df <- mtcars[1:4]

# printing the head of the data

print ("Original Data")

head(df)

# computing correlation matrix

cor\_data = cor(df)

print("Correlation matrix")

print(cor\_data)

Code For Scatter Plot :

df <- mtcars[1:4]

# You can either use this version for doing the same thing: pairs(~ mpg + hp + wt + disp, data=

# df, ...)

pairs(~ mtcars$mpg + mtcars$hp + mtcars$wt + mtcars$disp , data = df, pch = 19,col = "blue",

main = "Plot")

Code for Heatmap:

library(ggplot2)

library(ggcorrplot)

mydata.cor = cor(mtcars)

ggcorrplot(mydata.cor,

hc.order = TRUE,

type = "full",

lab = TRUE)

P-9

import pandas as pd

import numpy as np

df = pd.read\_csv('/content/synthetic\_dataset.csv')

print("Loaded Dataset:")

print(df.head())

df['BMI'] = df['Weight'] / (df['Height'] / 100) \*\* 2

print("\nData Info (Data types, non-null counts):")

print(df.info())

print("\nSummary Statistics:")

print(df.describe())

print("\nMissing Values (null counts):")

print(df.isnull().sum())

df\_older\_than\_30 = df[df['Age'] > 30]

print("\nFiltered Data (Age > 30):")

print(df\_older\_than\_30)

print("\nDescriptive Statistics (Mean, Median, Mode):")

mean\_values = df.mean()

median\_values = df.median()

mode\_values = df.mode().iloc[0]

print("Mean Values:")

print(mean\_values)

print("\nMedian Values:")

print(median\_values)

print("\nMode Values:")

print(mode\_values)

print("\nStandard Deviations and Variances using NumPy:")

std\_devs = np.std(df[['Age', 'Height', 'Salary', 'Weight', 'BMI']], axis=0)

variances = np.var(df[['Age', 'Height', 'Salary', 'Weight', 'BMI']], axis=0)

percentiles\_25 = {col: np.percentile(df[col], 25) for col in ['Age', 'Height', 'Salary', 'Weight',

'BMI']}

percentiles\_75 = {col: np.percentile(df[col], 75) for col in ['Age', 'Height', 'Salary', 'Weight',

'BMI']}

print("Standard Deviations:")

print(std\_devs)

print("\nVariances:")

print(variances)

print("\n25th Percentiles:")

print(percentiles\_25)

print("\n75th Percentiles:")

print(percentiles\_75)

print("\nCorrelation Matrix:")

print(df.corr())

print("\nCovariance Matrix:")

print(df.cov())

ages\_first\_5 = df['Age'].values[5:7]

print("First 7 Ages:")

print(ages\_first\_5)

age\_array = df['Age'].values.reshape(-1, 1)

print("\nReshaped Age Array (10 rows, 1 column):")

print(age\_array)

salary\_array = df['Salary'].values

concatenated\_data = np.column\_stack((age\_array, salary\_array))

print("\nConcatenated Age and Salary:")

print(concatenated\_data)

import matplotlib.pyplot as plt

import seaborn as sns

tips = sns.load\_dataset('tips')

print(tips.head()) # Show dataset structure

plt.figure(figsize=(10, 6))

sns.histplot(data=tips, x='total\_bill', kde=True, color='royalblue', bins=15)

plt.title('Distribution of Total Bills', fontsize=14, pad=20)

plt.xlabel('Total Bill ($)', fontsize=12)

plt.ylabel('Frequency', fontsize=12)

plt.grid(alpha=0.3)

plt.show()

plt.figure(figsize=(10, 6))

sns.barplot(x='day', y='total\_bill', hue='day', data=tips, errorbar=None, palette='coolwarm',

legend=False)

plt.title('Average Total Bill by Day', fontsize=14, pad=20)

plt.xlabel('Day of Week', fontsize=12)

plt.ylabel('Average Bill ($)', fontsize=12)

plt.xticks(rotation=45)

plt.grid(axis='y', alpha=0.3)

plt.show()

plt.figure(figsize=(8, 8))

day\_counts = tips['day'].value\_counts()

plt.pie(day\_counts, labels=day\_counts.index, autopct='%1.1f%%',

startangle=90, colors=sns.color\_palette('pastel'),

explode=(0.1, 0, 0, 0)) # Explode biggest slice

plt.title('Proportion of Orders by Day', fontsize=14, pad=20)

plt.show()

plt.figure(figsize=(10, 6))

sns.boxplot(data=tips, x='day', y='total\_bill', hue='smoker', palette='Set2')

plt.title('Total Bill Distribution by Day/Smoker Status', fontsize=14, pad=20)

plt.xlabel('Day of Week', fontsize=12)

plt.ylabel('Total Bill ($)', fontsize=12)

plt.legend(title='Smoker')

plt.grid(alpha=0.3)

plt.show()

plt.figure(figsize=(10, 6))

sns.violinplot(data=tips, x='day', y='total\_bill', split=True,

hue='smoker', palette='magma', inner='quartile')

plt.title('Density Distribution of Bills by Day/Smoker Status', fontsize=14, pad=20)

plt.xlabel('Day of Week', fontsize=12)

plt.ylabel('Total Bill ($)', fontsize=12)

plt.legend(title='Smoker')

plt.show()

sns.lmplot(data=tips, x='total\_bill', y='tip', hue='time',

height=6, aspect=1.5, markers=['o', 's'], palette='husl')

plt.title('Total Bill vs Tips with Regression Lines', fontsize=14, pad=20)

plt.xlabel('Total Bill ($)', fontsize=12)

plt.ylabel('Tip ($)', fontsize=12)

plt.grid(alpha=0.3)

plt.show()