```
#CA-1 1st

import pandas as pd
import matplotlib.pyplot as plt

# Load the data from the local CSV file
file_path = "C:/Users/RITHVI/Desktop/pred_final/IPG2211A2N.csv"
data = pd.read_csv(file_path, parse_dates=['DATE'], index_col='DATE')

# a. Resample the series and choose an essential graph to visualize the month-wise five number summary.
monthly_summary = data.resample('M').apply(lambda x: x.describe().loc['50%'])
plt.figure(figsize=(10, 6))
plt.boxplot(monthly_summary['IPG2211A2N'])
plt.title('Month-wise Five Number Summary for Industrial Production Index')
plt.xlabel('Month')
plt.ylabel('Index Value')
plt.xticks(rotation=45)
plt.grid(True, axis='y')
plt.tight_layout()
plt.show()

# b. Enumerate the components of a time series and isolate the components using additive and multiplicative models.
# Decompose the time series using additive and multiplicative models
from statsmodels.tsa.seasonal import seasonal_decompose

# Use additive decomposition
result_add = seasonal_decompose(data['IPG2211A2N'], model='additive', extrapolate_trend='freq')

# Use multiplicative decomposition
result_mult = seasonal_decompose(data['IPG2211A2N'], model='multiplicative', extrapolate_trend='freq')

# Isolate the components
trend_add = result_add.trend
seasonal_add = result_add.seasonal
residual_add = result_add.resid

trend_mult = result_mult.trend
seasonal_mult = result_mult.seasonal
residual_mult = result_mult.resid

#CA1 - 2

import pandas as pd
import matplotlib.pyplot as plt

# Load the data from the local CSV file
file_path = "/content/IPG2211A2N.csv"
data = pd.read_csv(file_path, parse_dates=['DATE'], index_col='DATE')

from statsmodels.tsa.stattools import adfuller

# Perform ADF test on the original series
result = adfuller(data['IPG2211A2N'])
print('ADF Statistic:', result[0])
print('p-value:', result[1])

# Plot the original time series
plt.figure(figsize=(10, 6))
plt.plot(data['IPG2211A2N'])
plt.title('Original Time Series - Industrial Production Index')
plt.xlabel('Date')
plt.ylabel('Index Value')
plt.grid(True)
plt.tight_layout()
plt.show()

# Make the time series stationary using differencing
data_diff = data['IPG2211A2N'].diff().dropna()

# Plot the differenced time series
plt.figure(figsize=(10, 6))
plt.plot(data_diff)
plt.title('Differenced Time Series - Industrial Production Index')
plt.xlabel('Date')
plt.ylabel('Differenced Index Value')
plt.grid(True)
plt.tight_layout()
plt.show()

#Pyspark
```

```
pip install pyspark

from pyspark.sql import SparkSession
from pyspark.ml.feature import StringIndexer, VectorAssembler
from pyspark.ml.classification import LogisticRegression

# Step 1: Create a Spark session
spark = SparkSession.builder.appName("LogisticRegressionExample").getOrCreate()

# Step 2: Load the dataset
data = spark.read.csv("C:/Users/RITHVI/Desktop/pred_final/Log_Reg_dataset.csv", header=True, inferSchema=True)

# Step 3: Perform descriptive statistics
data.describe().show()

# Step 4: Assign probability values (0 and 1) for class 0 and class 1 in 'Status' column
data = data.withColumn("Status", data["Status"].cast("double"))

# Step 5: Convert categorical columns to numeric using StringIndexer
indexers = [
    StringIndexer(inputCol=col, outputCol=col+"_index", handleInvalid="keep")
    for col in ["Country", "Platform"]
]
indexers.append(StringIndexer(inputCol="Repeat_Visitor", outputCol="Repeat_Visitor_index", handleInvalid="keep"))

# Fit and transform the data with the StringIndexer
for indexer in indexers:
    data = indexer.fit(data).transform(data)

# Step 6: Prepare data for logistic regression
# Select features and label columns
feature_cols = ["Country_index", "Age", "Repeat_Visitor_index", "Platform_index", "Web_pages_viewed"]
label_col = "Status"

# Create a vector assembler to combine features into a single vector column
assembler = VectorAssembler(inputCols=feature_cols, outputCol="features")
data = assembler.transform(data)

# Step 7: Split the data into training and test datasets
(training_data, test_data) = data.randomSplit([0.8, 0.2], seed=42)

# Step 8: Build and fit the logistic regression model
lr = LogisticRegression(labelCol=label_col)
lr_model = lr.fit(training_data)

# Step 9: Make predictions on the test dataset
predictions = lr_model.transform(test_data)

# Confusion matrix
conf_matrix = predictions.groupBy("Status", "prediction").count()
print("Confusion Matrix:")
conf_matrix.show()

# Precision, recall, and F1 score
TP = predictions[(predictions.Status == 1) & (predictions.prediction == 1)].count()
TN = predictions[(predictions.Status == 0) & (predictions.prediction == 0)].count()
FP = predictions[(predictions.Status == 0) & (predictions.prediction == 1)].count()
FN = predictions[(predictions.Status == 1) & (predictions.prediction == 0)].count()

# Compute classification metrics
accuracy = (TP + TN) / (TP + TN + FP + FN)
precision = TP / (TP + FP)
recall = TP / (TP + FN)
f1_score = 2 * (precision * recall) / (precision + recall)

print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1 Score:", f1_score)

#Superstore

import pandas as pd
import matplotlib.pyplot as plt
from statsmodels.tsa.stattools import adfuller
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from statsmodels.tsa.arima.model import ARIMA
from sklearn.metrics import mean_squared_error


# Load data
df = pd.read_excel('/content/Superstore.xls')
```

```
office_supplies = df[df['Category'] == 'Office Supplies']
office_supplies['Order Date'] = pd.to_datetime(office_supplies['Order Date'])
# Monthly sales
monthly_sales = office_supplies.groupby(pd.Grouper(key='Order Date', freq='M')).sum()['Sales']

# a. Stationarity check
plt.plot(monthly_sales, label='Original')
plt.plot(monthly_sales.rolling(window=12).mean(), label='Rolling Mean')
plt.plot(monthly_sales.rolling(window=12).std(), label='Rolling Std')
plt.legend()
plt.title('Rolling Mean & Standard Deviation')
plt.show()

# Augmented Dickey-Fuller test
result = adfuller(monthly_sales)
print('ADF Statistic (a):', result[0])
print('p-value (a):', result[1])
print('Critical Values (a):', result[4])

# b. Determine order of differencing, d
d = 1

plt.plot(monthly_sales)
plt.title('Monthly Sales')
plt.xlabel('Order Date')
plt.ylabel('Sales')
plt.show()

# c. Determine order p for AR(p)
plt.figure(figsize=(10, 6))
plot_pacf(monthly_sales, lags=min(24, len(monthly_sales)-1), alpha=0.05)
plt.title('Partial Autocorrelation Function (PACF)')
plt.show()

# Assuming p = 2
p = 2

# d. Determine order q for MA(q)
plt.figure(figsize=(10, 6))
plot_acf(monthly_sales, lags=30, alpha=0.05)
plt.title('Autocorrelation Function (ACF)')
plt.show()

# Assuming q = 2
q = 2

# e. Fit ARIMA model and forecast
model = ARIMA(monthly_sales, order=(p, d, q))
model_fit = model.fit()

# Forecast
forecast = model_fit.forecast(steps=len(monthly_sales))

# Plot original data and forecast
plt.plot(monthly_sales, label='Original')
plt.plot(monthly_sales.index, forecast, label='Forecast', linestyle='--')
plt.legend()
plt.title('ARIMA Forecast')
plt.show()

# Evaluate model accuracy
mse = mean_squared_error(monthly_sales, forecast)
print('Mean Squared Error (MSE) (e):', mse)

#######
#e.
import pandas as pd
from statsmodels.tsa.arima.model import ARIMA

# Load data
data = pd.read_excel("C:\\Users\\91636\\Downloads\\Superstore (1).xls")
office_supplies_sales = data[data['Category'] == 'Office Supplies'][['Order Date', 'Sales']]
office_supplies_sales = office_supplies_sales.set_index('Order Date').sort_index()

# Fit ARIMA model
model = ARIMA(office_supplies_sales['Sales'], order=(2, 1, 2))
model_fit = model.fit()

# Summary
print(model_fit.summary())

# Forecasting
forecast = model_fit.forecast(steps=10)
```

```
forecast = model_fit.forecast(steps=10)
print(forecast)

#ANN

import pandas as pd
import numpy as np
import tensorflow as tf
from sklearn.preprocessing import LabelEncoder, StandardScaler

# Load the dataset
df = pd.read_csv("C:/Users/RITHVI/Desktop/pred_final/WA_Fn-UseC_-Telco-Customer-Churn.csv")

# Preprocess the data
encoder = LabelEncoder()
scaler = StandardScaler()
categorical_cols = df.select_dtypes(include=['object']).columns
df[categorical_cols] = df[categorical_cols].apply(encoder.fit_transform)
numerical_cols = ['tenure', 'MonthlyCharges', 'TotalCharges']
df[numerical_cols] = scaler.fit_transform(df[numerical_cols])

# Define features and target variable
X = df.drop(columns=['customerID', 'Churn'])
y = df['Churn']

# Define the ANN model
model = tf.keras.Sequential([
    tf.keras.layers.Dense(64, activation='relu', input_shape=(X.shape[1],)),
    tf.keras.layers.Dense(32, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Train the model
model.fit(X, y, batch_size=34, epochs=100)

new_customer_info = np.array([[10, 50.0, 500.0]])  # Example: Tenure=10, MonthlyCharges=50.0, TotalCharges=500.0

# Scale the new customer information
new_customer_info_scaled = scaler.transform(new_customer_info)

# Expand the new_customer_info_scaled array to match the input shape expected by the model
new_customer_info_scaled_expanded = np.hstack((new_customer_info_scaled, np.zeros((new_customer_info_scaled.shape[0], 16))))

# Predict churn for the new customer
prediction = model.predict(new_customer_info_scaled_expanded)

if prediction > 0.5:
    print("The new customer is predicted to churn.")
else:
    print("The new customer is predicted not to churn.")


#Spam
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

df = pd.read_csv("C:/Users/RITHVI/Desktop/pred_final/Spam.csv", encoding='latin-1')

df = df[['v1', 'v2']]
df.columns = ['label', 'message']

import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

df = pd.read_csv("C:/Users/RITHVI/Desktop/pred_final/Spam.csv", encoding='latin-1')

df = df[['v1', 'v2']]
df.columns = ['label', 'message']

vectorizer = CountVectorizer()
x = vectorizer.fit_transform(df['message'])


y = df['label']

clf = MultinomialNB()
clf.fit(x, y)

def classify(clf,text):
```

```
def evaluate_classifier(clf, x_test, y_test):
    y_pred = clf.predict(x_test)
    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred, pos_label='spam')
    recall = recall_score(y_test, y_pred, pos_label='spam')
    f1 = f1_score(y_test, y_pred, pos_label='spam')
    return accuracy, precision, recall, f1

accuracy, precision, recall, f1 = evaluate_classifier(clf, x, y)
print(f"Accuracy: {accuracy:.2f}")
print(f"precision: {precision:.2f}")
print(f"recall: {recall:.2f}")
print(f"F1 Score: {f1:.2f}")


#India exchange rate
import pandas as pd
import matplotlib.pyplot as plt
from statsmodels.tsa.stattools import adfuller
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf

data = pd.read_excel("C:/Users/RITHVI/Downloads/India_Exchange_Rate_Dataset.xls")

data['EXINUS'].plot(figsize=(12, 6), title='India Exchange Rate')
plt.show()

data['EXINUS'].rolling(window=12).mean().plot(figsize=(12, 6), title='India Exchange Rate with 12-month SMA')
plt.show()

data['EXINUS'].ewm(span=12).mean().plot(figsize=(12, 6), title='India Exchange Rate with 12-month EWMA')
plt.show()

result = adfuller(data['EXINUS'])
print("Adfuller statistics:", result[0])
print("p-value:", result[1])
#print(f'ADF Statistic: {result[0]}, p-value: {result[1]}')

fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(12, 8))
plot_acf(data['EXINUS'], lags=30, ax=ax1)
plot_pacf(data['EXINUS'], lags=30, ax=ax2)
plt.show()

#Fitbit

pip install pyspark

import pandas as pd
import sqlite3
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
from pyspark.sql import SparkSession
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.regression import RandomForestRegressor as SparkRandomForestRegressor, LinearRegression as SparkLinearRegression

# Step 1: Create SQLite database "fitbit.db" and load the fitbitdf.csv file
conn = sqlite3.connect('fitbit.db')

# Load the fitbitdf.csv file into the SQLite database
df = pd.read_csv('/content/fitbit_df.csv')
df.to_sql('fitbit_data', conn, if_exists='replace', index=False)

# Step 2: Perform exploratory analysis - calculate correlation measures
correlation_matrix = df.corr()
print("Correlation Matrix:")
print(correlation_matrix)

# Step 3: Predict calories using tree-based machine learning models
X = df.drop(columns=['Calories'])
y = df['Calories']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
rf_model = RandomForestRegressor().fit(X_train, y_train)
y_pred_rf = rf_model.predict(X_test)
mse_rf = mean_squared_error(y_test, y_pred_rf)
r2_rf = r2_score(y_test, y_pred_rf)
print("Random Forest Regression Results:")
print(f"MSE: {mse_rf}, R2: {r2_rf}")

# Step 4: Choose best performing model for PySpark implementation
spark = SparkSession.builder.appName("FitBitModel").getOrCreate()
df_spark = spark.createDataFrame(df)
assembler = VectorAssembler(inputCols=[str(col) for col in df.columns[:-1]], outputCol='features')
```

```
vector_df = assembler.transform(df_spark)
train_df, test_df = vector_df.randomSplit([0.8, 0.2], seed=42)
rf_model_spark = SparkRandomForestRegressor(featuresCol='features', labelCol='Calories').fit(train_df)
predictions_spark_rf = rf_model_spark.transform(test_df).select('Calories', 'prediction').toPandas()
mse_spark_rf = mean_squared_error(predictions_spark_rf['Calories'], predictions_spark_rf['prediction'])
r2_spark_rf = r2_score(predictions_spark_rf['Calories'], predictions_spark_rf['prediction'])
print("Random Forest Regression Results using PySpark:")
print(f"MSE: {mse_spark_rf}, R2: {r2_spark_rf}")

# Step 5: Predict calories using linear machine learning models
lr_model = LinearRegression().fit(X_train, y_train)
y_pred_lr = lr_model.predict(X_test)
mse_lr = mean_squared_error(y_test, y_pred_lr)
r2_lr = r2_score(y_test, y_pred_lr)
print("Linear Regression Results:")
print(f"MSE: {mse_lr}, R2: {r2_lr}")

# Step 6: Close Spark session and SQLite connection
spark.stop()
conn.close()
```

```
#r programming
#2
install.packages("readr")
install.packages("dplyr")
install.packages("Hmisc")
install.packages("ggplot2")
install.packages("datarium")
install.packages("caret")
library(readr)#reading a csv file
library(dplyr)#data wrangling
library(Hmisc)#data description
library(ggplot2)#data visualization
library(datarium)#dataset
library(caret)#macine learning library for splitting on training and test
data("marketing", package="datarium")
marketing_plan<-marketing
marketing_plan

#1
marketing_plan %>% ggplot(aes(x = youtube, y = sales)) +  geom_point() +
  labs(x = "Spending on YouTube ads",y = "Sales", title = "Graph 1: Relationship between YouTube ads and sales") +  stat_smooth(se = FALS
marketing_plan %>% ggplot(aes(x = facebook, y = sales)) +  geom_point() +
  labs(x = "facebook",y = "Sales", title = "Graph 2: Relationship between facebook and sales") +  stat_smooth(se = FALSE) +   theme(panel
marketing_plan %>% ggplot(aes(x = newspaper, y = sales)) +  geom_point() +
  labs(x = "newspaper",y = "Sales", title = "Graph 3: Relationship between newspaper and sales") +  stat_smooth(se = FALSE) +   theme(pan

#4
set.seed(1)
train_indices<-createDataPartition(y=marketing[["sales"]],
                                   p=0.8,
                                   list = FALSE)
train_listings<-marketing[train_indices,]
test_listings<-marketing[-train_indices,]

#3
#observation
#we can see that the p value for youtube and facebook is extremely small,
#which means that we reject the null hypothesis that the youtube and facbook donot impact sales
#on the other hand, p value for newspaper is greater than 0.05,
#which means it is not a significant value. we fail to reject the null hypothesis
#that there is any significant relationship between newspaper ads and sales.
#I will create second model to exclude the variable newspaper.

#5
#case1:considering youtube,facebook,newspaper versus sales
model_0<-lm(sales~youtube+facebook+newspaper,data=train_listings)
summary(model_0)
model_1<-lm(sales~youtube+facebook,data=train_listings)
summary(model_1)
model_2<-lm(sales~facebook+I(facebook^2)+youtube+I(youtube^2),data=train_listings)
summary(model_2)
model_3<-lm(sales~facebook+poly(youtube,5),data=train_listings)
summary(model_3)
model_4<-lm(sales~facebook+poly(youtube,3)+facebook*youtube,data=train_listings)
summary(model_4)
```