#Linear Regression in Python ---------------🡪

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.metrics import mean\_squared\_error, mean\_absolute\_error, r2\_score

# Load the data

df= pd.read\_csv("Experience-Salary.csv")

print(df)

# Prepare the data

X = df["exp(in months)"].values.reshape(-1, 1)

Y = df["salary(in thousands)"].values.reshape(-1, 1)

# Split the data

xtrain, xtest, ytrain, ytest = train\_test\_split(X, Y, test\_size=0.2, shuffle=True)

# Train the model

model = LinearRegression()

model.fit(xtrain, ytrain)

ypred = model.predict(xtest)

# Calculate and display evaluation metrics

print('Mean Squared Error:', mean\_squared\_error(ytest, ypred))

print('Mean Absolute Error:', mean\_absolute\_error(ytest, ypred))

print('R-squared:', r2\_score(ytest, ypred))

# Display the model's intercept and coefficients

print("Intercept:", model.intercept\_)

print("Coefficients:", model.coef\_)

# Plot the results

plt.scatter(xtest, ytest, color='red')

plt.plot(xtest, ypred, color='blue')

plt.xlabel("Years of Experience in month")

plt.ylabel("Salary in thousands")

plt.title("Salary Prediction")

plt.show()

#Linear Regression in R ---------------🡪

df<-read.csv("Experience-Salary.csv")

df

df<-na.omit(df)

df

install.packages("caTools")

library(caTools)

s<-sample.split(df$exp.in.months., df$salary.in.thousands., SplitRatio=0.6)

training\_set<-subset(df, s==TRUE)

training\_set

testing\_set<-subset(df, s==FALSE)

testing\_set

model <- lm(salary.in.thousands.~ exp.in.months., data = training\_set)

summary(model)

pred\_sal=predict(model, testing\_set)

pred\_sal

plot(training\_set$exp.in.months., training\_set$salary.in.thousands.,xlab = "YearsExperience", ylab = "Salary", main = "Linear Regression")

abline(model, col = "red")

#Multiple Linear Regression in Python ---------------🡪

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.metrics import mean\_squared\_error, mean\_absolute\_error, r2\_score

# Load the dataset

df = pd.read\_csv("Student\_Performance.csv")

print(df)

# Define the independent variables (features) and the dependent variable (target)

X = df[["Hours Studied","Sample Question Papers Practiced"]]

Y = df["Performance Index"]

# Display the dependent variable name

print("Dependent variable (Target):", Y.name)

# Split the data into training and test sets

xtrain, xtest, ytrain, ytest = train\_test\_split(X, Y, test\_size=0.2, shuffle=True)

# Initialize and fit the linear regression model

model = LinearRegression()

model.fit(xtrain, ytrain)

# Predict the test set results

ypred = model.predict(xtest)

# Calculate and display evaluation metrics

print('Mean Squared Error:', mean\_squared\_error(ytest, ypred))

print('Mean Absolute Error:', mean\_absolute\_error(ytest, ypred))

print('R-squared:', r2\_score(ytest, ypred))

# Display the model's intercept and coefficients

print("Intercept:", model.intercept\_)

print("Coefficients:", model.coef\_)

# Display the coefficient for each feature

features = X.columns

for feature, coef in zip(features, model.coef\_):

    print(f"Coefficient for {feature}: {coef}")

# Plotting the results

plt.scatter(ytest,ypred,color="red")

plt.plot([min(ytest),max(ytest)],[min(ypred),max(ypred)])

plt.xlabel("Features")

plt.ylabel("Performance Index")

plt.title("Multiple Regression Prediction")

plt.show()

ax=plt.axes(projection="3d")

ax.plot3D(xtest["Hours Studied"],xtest["Sample Question Papers Practiced"],ytest)

plt.show()

#Multiple Linear Regression in R ---------------🡪

df<-read.csv("Student\_Performance.csv")

df

df<-na.omit(df)

df

install.packages("caTools")

library(caTools)

install.packages("scatterplot3d")

library(scatterplot3d)

s<-sample.split(df$Hours.Studied+df$Sleep.Hours, df$Performance.Index, SplitRatio=0.6)

training\_set<-subset(df, s==TRUE)

training\_set

testing\_set<-subset(df, s==FALSE)

testing\_set

model <- lm(Performance.Index~Hours.Studied+Sleep.Hours, data = training\_set)

summary(model)

scatterplot3d(training\_set$Hours.Studied, training\_set$Sleep.Hours, training\_set$Performance.Index, xlab = "Hours Studied", ylab = "Sleep Hours", zlab = "Performance Index", main = "3D Scatter Plot of Performance Index")

#Time Series in Python-------------------🡪

import pandas as pd

import matplotlib.pyplot as plt

from statsmodels.tsa.stattools import adfuller

from statsmodels.tsa.seasonal import seasonal\_decompose

from statsmodels.tsa.arima.model import ARIMA

df = pd.read\_csv("Month\_Value\_1.csv")

print(df)

df = df.dropna()

print(df)

ts\_data = pd.Series(df['Revenue'].values, index=pd.date\_range(start='2015-01-01', periods=len(df), freq='ME'))

ts\_data.plot(title="TSeries", xlabel="Month-Year", ylabel="Revenue")

plt.show()

decomposition = seasonal\_decompose(ts\_data, model='additive', period=12)

decomposition.plot()

plt.show()

ad\_test = adfuller(ts\_data)

print(f"ADF Statistic: {ad\_test[0]}")

print(f"p-value: {ad\_test[1]}")

if ad\_test[1] <= 0.05 :

    print("Time Series is stationary")

else:

    print("Time Series is not stationary")

ts\_data1 = ts\_data.diff().dropna()

ts\_data1.plot(title="Differenced TSeries", xlabel="Month-Year", ylabel="Revenue")

plt.show()

decomposition1 = seasonal\_decompose(ts\_data1, model='additive', period=12)

decomposition1.plot()

plt.show()

model = ARIMA(ts\_data1, order=(2, 1, 1))

model\_fit = model.fit()

forecast\_data = model\_fit.forecast(steps=20)

plt.plot(ts\_data1, label='Observed')

plt.plot(forecast\_data, label='Forecast', color='red')

plt.title("ARIMA Forecast")

plt.xlabel("Month-Year")

plt.ylabel("Revenue")

plt.legend()

plt.show()

#Time Series in R-------------------🡪

df<-read.csv("Month\_Value\_1.csv")

df

df<-na.omit(df)

df

ts\_data<-ts(df$Revenue, start=c(2015,1), frequency=12)

ts\_data

plot(ts\_data)

de<-decompose(ts\_data)

plot(de)

install.packages("tseries")

library(tseries)

adf\_result <- adf.test(ts\_data)

print(adf\_result)

ts\_data1<-diff(ts\_data)

plot(ts\_data1)

de1<-decompose(ts\_data1)

plot(de1)

model<-arima(ts\_data1, order=c(2,1,1))

pred\_data<-forecast(model, h=20)

plot(pred\_data)

#ARIMA in Python-------------------🡪

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from statsmodels.tsa.stattools import adfuller

from statsmodels.tsa.arima.model import ARIMA

df = pd.read\_csv("Month\_Value\_1.csv")

print(df)

df = df.dropna()

print(df)

ts\_data = pd.Series(df['Revenue'].values, index=pd.date\_range(start='2015-01-01', periods=len(df), freq='ME'))

ts\_data.plot(title="TSeries", xlabel="Month-Year", ylabel="Revenue")

plt.show()

ad\_test = adfuller(ts\_data)

print(f"ADF Statistic: {ad\_test[0]}")

print(f"p-value: {ad\_test[1]}")

if ad\_test[1] <= 0.05 :

    print("Time Series is stationary")

else:

    print("Time Series is not stationary")

ts\_data1 = ts\_data.diff().dropna()

ts\_data1.plot(title="Differenced TSeries", xlabel="Month-Year", ylabel="Revenue")

plt.show()

ad\_test = adfuller(ts\_data1)

print(f"ADF Statistic: {ad\_test[0]}")

print(f"p-value: {ad\_test[1]}")

if ad\_test[1] <= 0.05 :

    print("Time Series is stationary")

else:

    print("Time Series is not stationary")

model = ARIMA(ts\_data1, order=(2, 1, 1))

model\_fit = model.fit()

forecast\_data = model\_fit.forecast(steps=20)

plt.plot(ts\_data1, label='Observed')

plt.plot(forecast\_data, label='Forecast', color='red')

plt.title("ARIMA Forecast")

plt.xlabel("Month-Year")

plt.ylabel("Revenue")

plt.legend()

plt.show()

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

plot\_acf(ts\_data1, lags=6)

plt.show()

plot\_pacf(ts\_data1, lags=6)

plt.show()

#ARIMA in R-------------------🡪

df<-read.csv("Month\_Value\_1.csv")

df

df<-na.omit(df)

df

ts\_data<-ts(df$Revenue, start=c(2015,1), frequency=12)

ts\_data

plot(ts\_data)

install.packages("tseries")

library(tseries)

adf\_result <- adf.test(ts\_data)

print(adf\_result)

ts\_data1<-diff(ts\_data)

plot(d\_data)

adf\_result <- adf.test(ts\_data1)

print(adf\_result)

install.packages("forecast")

library(forecast)

model<-arima(ts\_data1, order=c(2,1,1))

pred\_data<-forecast(model, h=20)

plot(pred\_data)

acf(d\_data, lag.max=6)

pacf(d\_data, lag.max=6)

#Text Analysis ----------------------------🡪

#stopword removal

from nltk.corpus import stopwords

import nltk

nltk.download('stopwords')

nltk.download('punkt')

s\_word = set(stopwords.words('english'))

print(s\_word)

text = "Hello, I am in Ratnagiri, where are you ?"

text = text.lower()

words = text.split()

for word in words:

    if(word not in s\_word):

        print(word)

#lemmatization

import nltk

from nltk.stem import WordNetLemmatizer

nltk.download('wordnet')

lem = WordNetLemmatizer()

print(lem.lemmatize("are", pos='v'))

print(lem.lemmatize("running", pos='v'))

print(lem.lemmatize("playing", pos='v'))

print(lem.lemmatize("effectively", pos='r'))

print(lem.lemmatize("best", pos='a'))

#Stemmer

from nltk.stem import PorterStemmer

ps=PorterStemmer()

text = "Running playing effectively happily is am best"

words = text.split()

for word in words:

    print(ps.stem(word))

#Remove punctuations

import string

text = "Hello, I am in Ratnagiri, where are you ?"

print(text)

text = text.translate(str.maketrans("", "", string.punctuation))

print(text)

#Bag of words

from sklearn.feature\_extraction.text import CountVectorizer

text = [

    "Hello world! This is a sample text.",

    "Bag of words model is simple.",

    "This is another example of text processing."

]

vectorize = CountVectorizer()

x = vectorize.fit\_transform(text)

print(vectorize.get\_feature\_names\_out(), x.toarray())

#TFIDF

from sklearn.feature\_extraction.text import TfidfVectorizer

text = [

    "Hello world! This is a sample text.",

    "Bag of words model is simple.",

    "This is another example of text processing."

]

vectorize = TfidfVectorizer()

x = vectorize.fit\_transform(text)

print(vectorize.get\_feature\_names\_out(), x.toarray())