**Back Propogation**

import numpy as np

X = np.array(([2, 9], [1, 5], [3, 6]), dtype=float)

y = np.array(([92], [86], [89]), dtype=float)

X = X/np.amax(X,axis=0) #maximum of X array longitudinally

y = y/100

#Sigmoid Function

def sigmoid (x):

return 1/(1 + np.exp(-x))

#Derivative of Sigmoid Function

def derivatives\_sigmoid(x):

return x \* (1 - x)

#Variable initialization

epoch=5 #Setting training iterations

lr=0.1 #Setting learning rate

inputlayer\_neurons = 2 #number of features in data set

hiddenlayer\_neurons = 3 #number of hidden layers neurons

output\_neurons = 1 #number of neurons at output layer

#weight and bias initialization

wh=np.random.uniform(size=(inputlayer\_neurons,hiddenlayer\_neurons))

bh=np.random.uniform(size=(1,hiddenlayer\_neurons))

wout=np.random.uniform(size=(hiddenlayer\_neurons,output\_neurons))

bout=np.random.uniform(size=(1,output\_neurons))

#draws a random range of numbers uniformly of dim x\*y

for i in range(epoch):

#Forward Propogation

hinp1=np.dot(X,wh)

hinp=hinp1 + bh

hlayer\_act = sigmoid(hinp)

outinp1=np.dot(hlayer\_act,wout)

outinp= outinp1+bout

output = sigmoid(outinp)

#Backpropagation

EO = y-output

outgrad = derivatives\_sigmoid(output)

d\_output = EO \* outgrad

EH = d\_output.dot(wout.T)

hiddengrad = derivatives\_sigmoid(hlayer\_act)#how much hidden layer wts contributed to error

d\_hiddenlayer = EH \* hiddengrad

wout += hlayer\_act.T.dot(d\_output) \*lr # dotproduct of nextlayererror and currentlayerop

wh += X.T.dot(d\_hiddenlayer) \*lr

print ("-----------Epoch-", i+1, "Starts----------")

print("Input: \n" + str(X))

print("Actual Output: \n" + str(y))

print("Predicted Output: \n" ,output)

print ("-----------Epoch-", i+1, "Ends----------\n")

print("Input: \n" + str(X))

print("Actual Output: \n" + str(y))

print("Predicted Output: \n" ,output)

**Candidate Elimination**

import csv

def is\_consistent(hypothesis, instance):

for i in range(len(hypothesis)):

if hypothesis[i] != '?' and hypothesis[i] != instance[i]:

return False

return True

def generalize\_specific(instance, general\_hypothesis):

new\_general\_hypothesis = list(general\_hypothesis)

for i in range(len(new\_general\_hypothesis)):

if general\_hypothesis[i] == '?':

new\_general\_hypothesis[i] = instance[i]

elif general\_hypothesis[i] != instance[i]:

new\_general\_hypothesis[i] = '?'

return new\_general\_hypothesis

def candidate\_elimination(data):

data\_reader = csv.reader(data)

instances = [row for row in data\_reader]

# Initialize general and specific hypotheses

specific\_hypothesis = instances[0][:-1]

general\_hypothesis = ['?' for \_ in range(len(specific\_hypothesis))]

for instance in instances:

if instance[-1] == 'Yes':

for i in range(len(specific\_hypothesis)):

if specific\_hypothesis[i] != instance[i]:

specific\_hypothesis[i] = '?'

for i in range(len(general\_hypothesis)):

if specific\_hypothesis[i] == '?':

general\_hypothesis[i] = '?'

else:

if is\_consistent(specific\_hypothesis, instance):

general\_hypothesis = generalize\_specific(instance, general\_hypothesis)

return specific\_hypothesis, general\_hypothesis

with open('enjoysport.csv', 'r') as csvfile:

specific, general = candidate\_elimination(csvfile)

print("Final Specific Hypothesis:", specific)

print("Final General Hypothesis:", general)

**Car Price Precdition**

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

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

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import mean\_squared\_error, r2\_score

# Load the dataset

data = pd.read\_csv("carprice.csv")

data.head()

data.shape

data.isnull().sum()

#So this dataset doesn’t have any null values, now let’s look at some of the other important insights to get

#an idea of what kind of data we’re dealing with:

data.info()

data.describe()

# Drop the 'carname' column

data = data.drop(columns=['carname'])

sns.set\_style("whitegrid")

plt.figure(figsize=(15, 10))

sns.histplot(data.price, kde=True)

plt.show()

'''

# Now let’s have a look at the correlation among all the features of this dataset:

print(data.corr())

plt.figure(figsize=(20, 15))

correlations = data.corr()

sns.heatmap(correlations, cmap="coolwarm", annot=True)

plt.show()'''

#Training a Car Price Prediction Model

predict = "price"

# Data preprocessing

selected\_features = ["symboling", "wheelbase", "carlength",

"carwidth", "carheight", "curbweight",

"enginesize", "boreratio", "stroke",

"compressionratio", "horsepower", "peakrpm",

"citympg", "highwaympg"]

X = data[selected\_features]

y = data["price"]

# Split the data into training and testing sets

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

# Create and train the Linear Regression model

model = LinearRegression()

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

# Make predictions

predictions = model.predict(X\_test)

# Evaluate the model

mse = mean\_squared\_error(y\_test, predictions)

r2 = r2\_score(y\_test, predictions)

print("Mean Squared Error:", mse)

print("R-squared:", r2)

# Example prediction

example\_data = np.array([[0, 95.1, 158.7, 63.6, 52.0, 2017, 141, 3.78, 3.15, 8.0, 95, 5000, 27, 34]])

example\_df = pd.DataFrame(example\_data, columns=selected\_features)

predicted\_price = model.predict(example\_df)

print("Predicted Car Price:", predicted\_price[0])

print(predictions)

**Comparison Of Classfication Algorithm**

import numpy

import pandas as pd

import numpy as np

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

from sklearn.tree import DecisionTreeClassifier

from sklearn.linear\_model import LogisticRegression

from sklearn.naive\_bayes import BernoulliNB

from sklearn.neighbors import KNeighborsClassifier

from sklearn.linear\_model import PassiveAggressiveClassifier

from sklearn.metrics import classification\_report

iris= pd.read\_csv("IRIS.csv")

print(iris.head())

x = iris.drop("species", axis=1)

y = iris["species"]

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

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y,test\_size=0.10,random\_state=42)

#x = np.array(data[["Age", "EstimatedSalary"]])

#y = np.array(data[["Purchased"]])

#xtrain, xtest, ytrain, ytest = train\_test\_split(x, y, test\_size=0.10, random\_state=42)

decisiontree = DecisionTreeClassifier()

logisticregression = LogisticRegression()

knearestclassifier = KNeighborsClassifier()

#svm\_classifier = SVC()

bernoulli\_naiveBayes = BernoulliNB()

passiveAggressive = PassiveAggressiveClassifier()

knearestclassifier.fit(x\_train, y\_train)

decisiontree.fit(x\_train, y\_train)

logisticregression.fit(x\_train, y\_train)

passiveAggressive.fit(x\_train, y\_train)

data1 = {"Classification Algorithms": ["KNN Classifier", "Decision Tree Classifier",

"Logistic Regression", "Passive Aggressive Classifier"],

"Score": [knearestclassifier.score(x,y), decisiontree.score(x, y),

logisticregression.score(x, y), passiveAggressive.score(x,y) ]}

score = pd.DataFrame(data1)

print(score)

**Car Price**

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

import numpy as np

from sklearn.preprocessing import OneHotEncoder, StandardScaler, MinMaxScaler

from sklearn.model\_selection import GridSearchCV, train\_test\_split, KFold, cross\_val\_score

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import r2\_score, mean\_squared\_error, accuracy\_score, mean\_absolute\_error, mean\_squared\_error

from sklearn.ensemble import GradientBoostingRegressor

from sklearn.neural\_network import MLPRegressor

from sklearn.svm import SVR

from sklearn.neighbors import KNeighborsRegressor

from sklearn.cross\_decomposition import PLSRegression

from sklearn.tree import DecisionTreeRegressor

import missingno as msno

from sklearn.utils import shuffle

from category\_encoders import TargetEncoder, OneHotEncoder

import warnings

warnings.filterwarnings("ignore")

sns.set(rc = {'figure.figsize': (20, 20)})

plt.figure(figsize = (20, 15))

sns.countplot(y = data.Make)

plt.title("Car companies with their cars", fontsize = 20)

plt.show()

plt.figure(figsize = (20, 15))

sns.countplot(data.Year, palette = 'viridis')

plt.title("Number of cars in different years", fontsize = 20)

plt.show()

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

sns.countplot(x = 'Vehicle Size', data = data, palette = 'Set1')

msno.matrix(data, color = (0.5, 0.5, 0.5))

plt.figure(figsize = (20, 10))

data.groupby('Year')['MSRP'].mean().plot(kind = 'bar', color = 'g')

plt.title("The Average Price of cars in different years", fontsize = 20)

plt.show()

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

data.groupby('Transmission Type')['MSRP'].mean().plot(kind = 'bar', color = 'y')

plt.title("The Average Price of cars in different tranmission types", fontsize = 20)

plt.show()

plt.figure(figsize = (20, 15))

data.groupby(['Make']).mean()['MSRP'].sort\_values(ascending = False).plot(kind = 'bar', fontsize = 15, color = 'black')

plt.title("The average price of cars of different companies", fontsize = 20)

plt.show()

plt.figure(figsize = (15, 15))

numeric\_columns = ['Engine HP', 'Engine Cylinders', 'Number of Doors', 'highway MPG', 'city mpg', 'Popularity']

heatmap\_data = data[numeric\_columns].corr()

sns.heatmap(heatmap\_data, cmap = 'BuPu', annot = True)

encoder = TargetEncoder(cols = 'Make')

encoder.fit(X\_train['Make'], y\_train.to\_frame()['MSRP'])

X\_train['Make'] = encoder.transform(X\_train['Make'])

X\_test['Make'] = encoder.transform(X\_test['Make'])

data['Present Year'] = 2021

data['Years Of Manufacture'] = data['Present Year'] - data['Year']

data.drop(['Present Year'], inplace = True, axis = 1)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.2, random\_state = 100)

encoder = OneHotEncoder()

encoder.fit(X\_train[['Engine Fuel Type', 'Transmission Type', 'Driven\_Wheels', 'Vehicle Size', 'Vehicle Style']])

one\_hot\_encoded\_output\_train = encoder.transform(X\_train[['Engine Fuel Type', 'Transmission Type', 'Driven\_Wheels', 'Vehicle Size', 'Vehicle Style']])

one\_hot\_encoded\_output\_test = encoder.transform(X\_test[['Engine Fuel Type', 'Transmission Type', 'Driven\_Wheels', 'Vehicle Size', 'Vehicle Style']])

scaler = MinMaxScaler()

scaler.fit(X\_train)

X\_train\_new = scaler.transform(X\_train)

X\_test\_new = scaler.transform(X\_test)

model = LinearRegression()

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

y\_predict = model.predict(X\_test\_new)

error\_mean\_square.append(int(mean\_squared\_error(y\_predict, y\_test)))

error\_mean\_absolute.append(int(mean\_absolute\_error(y\_predict, y\_test)))

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

sns.regplot(data = results, y = 'Predicted Output', x = 'MSRP', color = 'teal', marker = 'o')

plt.title("Comparision of predicted values and the actual values", fontsize = 20)

plt.savefig('images/linear\_regression\_outcome.png')

plt.show()

model = SVR()

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

y\_predict = model.predict(X\_test\_new)

y\_predict = pd.DataFrame(y\_predict, columns = ['Predicted Output'])

results = pd.concat([y\_predict, y\_test.to\_frame().reset\_index(drop = True)], axis = 1, ignore\_index = False)

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

sns.regplot(data = results, y = 'Predicted Output', x = 'MSRP', color = 'palevioletred', marker = 'o')

plt.title("Comparision of predicted values and the actual values", fontsize = 20)

plt.savefig('images/support\_vector\_machines\_outcome.png')

plt.show()

model = KNeighborsRegressor(n\_neighbors = 2)

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

y\_predict = model.predict(X\_test\_new)

y\_predict = pd.DataFrame(y\_predict, columns = ['Predicted Output'])

results = pd.concat([y\_predict, y\_test.to\_frame().reset\_index(drop = True)], axis = 1, ignore\_index = False)

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

sns.regplot(data = results, y = 'Predicted Output', x = 'MSRP', color = 'darkslateblue', marker = 'o')

plt.title("Comparision of predicted values and the actual values", fontsize = 20)

plt.savefig('images/k\_nearest\_neighbors\_outcome.png')

plt.show()

model = PLSRegression(n\_components = 20)

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

y\_predict = model.predict(X\_test\_new)

y\_predict = pd.DataFrame(y\_predict, columns = ['Predicted Output'])

results = pd.concat([y\_predict, y\_test.to\_frame().reset\_index(drop = True)], axis = 1, ignore\_index = False)

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

sns.regplot(data = results, y = 'Predicted Output', x = 'MSRP', color = 'firebrick', marker = 'o')

plt.title("Comparision of predicted values and the actual values", fontsize = 20)

plt.savefig('images/pls\_regression\_outcome.png')

plt.show()

model = DecisionTreeRegressor(splitter = 'random')

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

y\_predict = model.predict(X\_test\_new)

y\_predict = pd.DataFrame(y\_predict, columns = ['Predicted Output'])

results = pd.concat([y\_predict, y\_test.to\_frame().reset\_index(drop = True)], axis = 1, ignore\_index = False)

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

sns.regplot(data = results, y = 'Predicted Output', x = 'MSRP', color = 'coral', marker = 'o')

plt.title("Comparision of predicted values and the actual values", fontsize = 20)

plt.savefig('images/decision\_tree\_regressor\_outcome.png')

plt.show()

model = GradientBoostingRegressor()

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

y\_predict = model.predict(X\_test\_new)

y\_predict = pd.DataFrame(y\_predict, columns = ['Predicted Output'])

results = pd.concat([y\_predict, y\_test.to\_frame().reset\_index(drop = True)], axis = 1, ignore\_index = False)

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

sns.regplot(data = results, y = 'Predicted Output', x = 'MSRP', color = 'darkmagenta', marker = 'o')

plt.title("Comparision of predicted values and the actual values", fontsize = 20)

plt.savefig('images/gradient\_boosting\_regressor\_outcome.png')

plt.show()

model = MLPRegressor(hidden\_layer\_sizes = 50, alpha = 0.001, solver = 'lbfgs', learning\_rate = 'adaptive')

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

y\_predict = model.predict(X\_test\_new)

y\_predict = pd.DataFrame(y\_predict, columns = ['Predicted Output'])

results = pd.concat([y\_predict, y\_test.to\_frame().reset\_index(drop = True)], axis = 1, ignore\_index = False)

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

sns.regplot(data = results, y = 'Predicted Output', x = 'MSRP', color = 'steelblue', marker = 'o')

plt.title("Comparision of predicted values and the actual values", fontsize = 20)

plt.savefig('images/mlp\_regressor\_outcome.png')

plt.show()

plt.figure(figsize = (20, 10))

splot = sns.barplot(data = model\_dataframe, x = 'Models', y = 'Mean Absolute Error', palette = 'Paired')

for p in splot.patches:

splot.annotate(format(p.get\_height(), '.0f'),

(p.get\_x() + p.get\_width() / 2., p.get\_height()),

ha = 'center', va = 'center',

xytext = (0, 9),

textcoords = 'offset points')

plt.xticks(fontsize = 13)

plt.yticks(fontsize = 13)

plt.title("Barplot of various machine learning regression models with mean absolute error", fontsize = 20)

plt.savefig('images/machine\_learning\_models\_outcomes.png')

plt.show()

plt.figure(figsize = (20, 10))

splot = sns.barplot(data = model\_dataframe, x = 'Models', y = 'Mean Squared Error', palette = 'Spectral')

for p in splot.patches:

splot.annotate(format(p.get\_height(), '.0f'),

(p.get\_x() + p.get\_width() / 2., p.get\_height()),

ha = 'center', va = 'center',

xytext = (0, 9),

textcoords = 'offset points')

plt.xticks(fontsize = 13)

plt.yticks(fontsize = 13)

plt.title("Barplot of various machine learning regression models with mean squared error", fontsize = 20)

plt.savefig('images/machine\_learning\_models\_MSE\_outcomes.png')

plt.show()

**Credit Score**

import pandas as pd

import numpy as np

import plotly.express as px

import plotly.graph\_objects as go

import plotly.io as pio

pio.templates.default = "plotly\_white"

import plotly.io as io

io.renderers.default='browser'

data = pd.read\_csv('creditscore.csv')

print(data.head())

print(data.info())

#the dataset has any null values or not:

print(data.isnull().sum())

#The dataset doesn’t have any null values. As this dataset is labelled, let’s have a look at the Credit\_Score column values:

data["Credit\_Score"].value\_counts()

data.shape

#Data Exploration

"""

The dataset has many features that can train a Machine Learning model for credit score classification.

Let’s explore all the features one by one.

I will start by exploring the occupation feature to know if the occupation of the person affects credit scores:

"""

fig = px.box(data,

x="Occupation",

color="Credit\_Score",

title="Credit Scores Based on Occupation",

color\_discrete\_map={'Poor':'red',

'Standard':'yellow',

'Good':'green'})

fig.show()

"""

There’s not much difference in the credit scores of all occupations mentioned in the data. Now let’s explore

whether the Annual Income of the person impacts your credit scores or not:

"""

fig = px.box(data,

x="Credit\_Score",

y="Annual\_Income",

color="Credit\_Score",

title="Credit Scores Based on Annual Income",

color\_discrete\_map={'Poor':'red',

'Standard':'yellow',

'Good':'green'})

fig.update\_traces(quartilemethod="exclusive")

fig.show()

"""

let’s explore whether the monthly in-hand salary impacts credit scores or not:

"""

fig = px.box(data,

x="Credit\_Score",

y="Monthly\_Inhand\_Salary",

color="Credit\_Score",

title="Credit Scores Based on Monthly Inhand Salary",

color\_discrete\_map={'Poor':'red',

'Standard':'yellow',

'Good':'green'})

fig.update\_traces(quartilemethod="exclusive")

fig.show()

fig = px.box(data,

x="Credit\_Score",

y="Num\_Bank\_Accounts",

color="Credit\_Score",

title="Credit Scores Based on Number of Bank Accounts",

color\_discrete\_map={'Poor':'red',

'Standard':'yellow',

'Good':'green'})

fig.update\_traces(quartilemethod="exclusive")

fig.show()

# impact on credit scores based on the number of credit cards you have:

fig = px.box(data,

x="Credit\_Score",

y="Num\_Credit\_Card",

color="Credit\_Score",

title="Credit Scores Based on Number of Credit cards",

color\_discrete\_map={'Poor':'red',

'Standard':'yellow',

'Good':'green'})

fig.update\_traces(quartilemethod="exclusive")

fig.show()

fig = px.box(data,

x="Credit\_Score",

y="Interest\_Rate",

color="Credit\_Score",

title="Credit Scores Based on the Average Interest rates",

color\_discrete\_map={'Poor':'red',

'Standard':'yellow',

'Good':'green'})

fig.update\_traces(quartilemethod="exclusive")

fig.show()

data["Credit\_Mix"] = data["Credit\_Mix"].map({"Standard": 1,

"Good": 2,

"Bad": 0})

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

x = np.array(data[["Annual\_Income", "Monthly\_Inhand\_Salary",

"Num\_Bank\_Accounts", "Num\_Credit\_Card",

"Interest\_Rate", "Num\_of\_Loan",

"Delay\_from\_due\_date", "Num\_of\_Delayed\_Payment",

"Credit\_Mix", "Outstanding\_Debt",

"Credit\_History\_Age", "Monthly\_Balance"]])

y = np.array(data[["Credit\_Score"]])

xtrain, xtest, ytrain, ytest = train\_test\_split(x, y,

test\_size=0.33,

random\_state=42)

from sklearn.ensemble import RandomForestClassifier

model = RandomForestClassifier()

model.fit(xtrain, ytrain)

print("Credit Score Prediction : ")

a = float(input("Annual Income: "))

b = float(input("Monthly Inhand Salary: "))

c = float(input("Number of Bank Accounts: "))

d = float(input("Number of Credit cards: "))

e = float(input("Interest rate: "))

f = float(input("Number of Loans: "))

g = float(input("Average number of days delayed by the person: "))

h = float(input("Number of delayed payments: "))

i = input("Credit Mix (Bad: 0, Standard: 1, Good: 3) : ")

j = float(input("Outstanding Debt: "))

k = float(input("Credit History Age: "))

l = float(input("Monthly Balance: "))

features = np.array([[a, b, c, d, e, f, g, h, i, j, k, l]])

print("Predicted Credit Score = ", model.predict(features))

**Credit Score Classfication**

import pandas as pd

import numpy as np

import plotly.express as px

import plotly.graph\_objects as go

import plotly.io as pio

pio.templates.default = "plotly\_white"

import plotly.io as io

io.renderers.default='browser'

data = pd.read\_csv('creditscore.csv')

print(data.head())

print(data.info())

#the dataset has any null values or not:

print(data.isnull().sum())

#The dataset doesn’t have any null values. As this dataset is labelled, let’s have a look at the Credit\_Score column values:

data["Credit\_Score"].value\_counts()

data.shape

#Data Exploration

"""

The dataset has many features that can train a Machine Learning model for credit score classification.

Let’s explore all the features one by one.

I will start by exploring the occupation feature to know if the occupation of the person affects credit scores:

"""

fig = px.box(data,

x="Occupation",

color="Credit\_Score",

title="Credit Scores Based on Occupation",

color\_discrete\_map={'Poor':'red',

'Standard':'yellow',

'Good':'green'})

fig.show()

"""

There’s not much difference in the credit scores of all occupations mentioned in the data. Now let’s explore

whether the Annual Income of the person impacts your credit scores or not:

"""

fig = px.box(data,

x="Credit\_Score",

y="Annual\_Income",

color="Credit\_Score",

title="Credit Scores Based on Annual Income",

color\_discrete\_map={'Poor':'red',

'Standard':'yellow',

'Good':'green'})

fig.update\_traces(quartilemethod="exclusive")

fig.show()

"""

let’s explore whether the monthly in-hand salary impacts credit scores or not:

"""

fig = px.box(data,

x="Credit\_Score",

y="Monthly\_Inhand\_Salary",

color="Credit\_Score",

title="Credit Scores Based on Monthly Inhand Salary",

color\_discrete\_map={'Poor':'red',

'Standard':'yellow',

'Good':'green'})

fig.update\_traces(quartilemethod="exclusive")

fig.show()

fig = px.box(data,

x="Credit\_Score",

y="Num\_Bank\_Accounts",

color="Credit\_Score",

title="Credit Scores Based on Number of Bank Accounts",

color\_discrete\_map={'Poor':'red',

'Standard':'yellow',

'Good':'green'})

fig.update\_traces(quartilemethod="exclusive")

fig.show()

# impact on credit scores based on the number of credit cards you have:

fig = px.box(data,

x="Credit\_Score",

y="Num\_Credit\_Card",

color="Credit\_Score",

title="Credit Scores Based on Number of Credit cards",

color\_discrete\_map={'Poor':'red',

'Standard':'yellow',

'Good':'green'})

fig.update\_traces(quartilemethod="exclusive")

fig.show()

fig = px.box(data,

x="Credit\_Score",

y="Interest\_Rate",

color="Credit\_Score",

title="Credit Scores Based on the Average Interest rates",

color\_discrete\_map={'Poor':'red',

'Standard':'yellow',

'Good':'green'})

fig.update\_traces(quartilemethod="exclusive")

fig.show()

data["Credit\_Mix"] = data["Credit\_Mix"].map({"Standard": 1,

"Good": 2,

"Bad": 0})

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

x = np.array(data[["Annual\_Income", "Monthly\_Inhand\_Salary",

"Num\_Bank\_Accounts", "Num\_Credit\_Card",

"Interest\_Rate", "Num\_of\_Loan",

"Delay\_from\_due\_date", "Num\_of\_Delayed\_Payment",

"Credit\_Mix", "Outstanding\_Debt",

"Credit\_History\_Age", "Monthly\_Balance"]])

y = np.array(data[["Credit\_Score"]])

xtrain, xtest, ytrain, ytest = train\_test\_split(x, y,

test\_size=0.33,

random\_state=42)

from sklearn.ensemble import RandomForestClassifier

model = RandomForestClassifier()

model.fit(xtrain, ytrain)

print("Credit Score Prediction : ")

a = float(input("Annual Income: "))

b = float(input("Monthly Inhand Salary: "))

c = float(input("Number of Bank Accounts: "))

d = float(input("Number of Credit cards: "))

e = float(input("Interest rate: "))

f = float(input("Number of Loans: "))

g = float(input("Average number of days delayed by the person: "))

h = float(input("Number of delayed payments: "))

i = input("Credit Mix (Bad: 0, Standard: 1, Good: 3) : ")

j = float(input("Outstanding Debt: "))

k = float(input("Credit History Age: "))

l = float(input("Monthly Balance: "))

features = np.array([[a, b, c, d, e, f, g, h, i, j, k, l]])

print("Predicted Credit Score = ", model.predict(features))

**Decision Tree**

import pandas as pd

import math

import numpy as np

data = pd.read\_csv("enjoysport.csv")

features = [feat for feat in data]

features.remove("answer")

# Create a class named Node with four members children, value, isLeaf and pred.

class Node:

def \_\_init\_\_(self):

self.children = []

self.value = ""

self.isLeaf = False

self.pred = ""

# Define a function called entropy to find the entropy oof the dataset

def entropy(examples):

pos = 0.0

neg = 0.0

for \_, row in examples.iterrows():

if row["answer"] == "yes":

pos += 1

else:

neg += 1

if pos == 0.0 or neg == 0.0:

return 0.0

else:

p = pos / (pos + neg)

n = neg / (pos + neg)

return -(p \* math.log(p, 2) + n \* math.log(n, 2))

# Define a function named info\_gain to find the gain of the attribute

def info\_gain(examples, attr):

uniq = np.unique(examples[attr])

#print ("\n",uniq)

gain = entropy(examples)

#print ("\n",gain)

for u in uniq:

subdata = examples[examples[attr] == u]

#print ("\n",subdata)

sub\_e = entropy(subdata)

gain -= (float(len(subdata)) / float(len(examples))) \* sub\_e

#print ("\n",gain)

return gain

# Define a function named ID3 to get the decision tree for the given dataset

def ID3(examples, attrs):

root = Node()

max\_gain = 0

max\_feat = ""

for feature in attrs:

#print ("\n",examples)

gain = info\_gain(examples, feature)

if gain > max\_gain:

max\_gain = gain

max\_feat = feature

root.value = max\_feat

#print ("\nMax feature attr",max\_feat)

uniq = np.unique(examples[max\_feat])

#print ("\n",uniq)

for u in uniq:

#print ("\n",u)

subdata = examples[examples[max\_feat] == u]

#print ("\n",subdata)

if entropy(subdata) == 0.0:

newNode = Node()

newNode.isLeaf = True

newNode.value = u

newNode.pred = np.unique(subdata["answer"])

root.children.append(newNode)

else:

dummyNode = Node()

dummyNode.value = u

new\_attrs = attrs.copy()

new\_attrs.remove(max\_feat)

child = ID3(subdata, new\_attrs)

dummyNode.children.append(child)

root.children.append(dummyNode)

return root

# Define a function named printTree to draw the decision tree

def printTree(root: Node, depth=0):

for i in range(depth):

print("\t", end="")

print(root.value, end="")

if root.isLeaf:

print(" -> ", root.pred)

print()

for child in root.children:

printTree(child, depth + 1)

# Define a function named classify to classify the new example

def classify(root: Node, new):

for child in root.children:

if child.value == new[root.value]:

if child.isLeaf:

print ("Predicted Label for new example", new," is:", child.pred)

exit

else:

classify (child.children[0], new)

# Finally, call the ID3, printTree and classify functions

root = ID3(data, features)

print("Decision Tree is:")

printTree(root)

print ("------------------")

**Estimation And Maximum Algorithm**

#from sklearn.cluster import KMeans

from sklearn.mixture import GaussianMixture

import sklearn.metrics as metrics

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

names = ['Sepal\_Length','Sepal\_Width','Petal\_Length','Petal\_Width', 'Class']

dataset = pd.read\_csv("iris.csv", skiprows=1, names=names)

X = dataset.iloc[:, :-1]

label = {'Iris-setosa': 0,'Iris-versicolor': 1, 'Iris-virginica': 2}

y = [label[c] for c in dataset.iloc[:, -1]]

plt.figure(figsize=(14,7))

colormap=np.array(['red','lime','black'])

# REAL PLOT

plt.subplot(1,3,1)

plt.title('Real')

plt.scatter(X.Petal\_Length,X.Petal\_Width,c=colormap[y])

# GMM PLOT

gmm=GaussianMixture(n\_components=3, random\_state=0).fit(X)

y\_cluster\_gmm=gmm.predict(X)

plt.subplot(1,3,3)

plt.title('GMM Classification')

plt.scatter(X.Petal\_Length,X.Petal\_Width,c=colormap[y\_cluster\_gmm])

print('The accuracy score of EM: ',metrics.accuracy\_score(y, y\_cluster\_gmm))

print('The Confusion matrix of EM:\n ',metrics.confusion\_matrix(y, y\_cluster\_gmm))

**Find S**

import csv

a = []

with open("enjoysport.csv", 'r') as csvfile:

for row in csv.reader(csvfile):

a.append(row)

print(a)

print("\n The total number of training instances are : ",len(a))

num\_attribute = len(a[0])-1

print("\n The initial hypothesis is : ")

hypothesis = ['0']\*num\_attribute

print(hypothesis)

for i in range(0, len(a)):

if a[i][num\_attribute] == 'yes':

for j in range(0, num\_attribute):

if hypothesis[j] == '0' or hypothesis[j] == a[i][j]: hypothesis[j] = a[i][j]

else:

hypothesis[j] = '?'

print("\n The hypothesis for the training instance {} is :\n" .format(i+1),hypothesis)

print("\n The Maximally specific hypothesis for the training instance is ")

print(hypothesis)

**Future Sale Predicition**

import pandas as pd

import numpy as np

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

from sklearn.linear\_model import LinearRegression

import plotly.io as io

io.renderers.default='browser'

data = pd.read\_csv("futuresale prediction.csv")

print(data.head())

print(data.sample(5))

print(data.isnull().sum())

import plotly.express as px

import plotly.graph\_objects as go

figure = px.scatter(data\_frame = data, x="Sales",

y="TV", size="TV", trendline="ols")

figure.show()

figure = px.scatter(data\_frame = data, x="Sales",

y="Newspaper", size="Newspaper", trendline="ols")

figure.show()

figure = px.scatter(data\_frame = data, x="Sales",

y="Radio", size="Radio", trendline="ols")

figure.show()

correlation = data.corr()

print(correlation["Sales"].sort\_values(ascending=False))

x = np.array(data.drop(["Sales"], 1))

y = np.array(data["Sales"])

xtrain, xtest, ytrain, ytest = train\_test\_split(x, y,

test\_size=0.2,

random\_state=42)

model = LinearRegression()

model.fit(xtrain, ytrain)

print(model.score(xtest, ytest))

features = [[TV, Radio, Newspaper]]

features = np.array([[230.1, 37.8, 69.2]])

print(model.predict(features))

**House Predicition**

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.preprocessing import OneHotEncoder

from sklearn.metrics import mean\_absolute\_error, mean\_absolute\_percentage\_error

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

from sklearn.ensemble import RandomForestRegressor

from sklearn.linear\_model import LinearRegression

from sklearn.svm import SVR

# Load the dataset

dataset = pd.read\_csv("HousePricePrediction.csv")

# Display basic info about the dataset

print(dataset.head())

print(dataset.shape)

# Identify categorical and numeric columns

object\_cols = dataset.select\_dtypes(include=['object']).columns

num\_cols = dataset.select\_dtypes(include=['int', 'float']).columns

# Visualize correlation matrix

plt.figure(figsize=(12, 8))

sns.heatmap(dataset[num\_cols].corr(), cmap='BrBG', fmt='.2f', linewidths=2, annot=True)

plt.show()

# Visualize unique values of categorical columns

unique\_values = dataset[object\_cols].nunique()

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

plt.title('No. Unique values of Categorical Features')

plt.xticks(rotation=90)

sns.barplot(x=unique\_values.index, y=unique\_values.values)

plt.show()

# Visualize distribution of categorical features

plt.figure(figsize=(18, 36))

plt.title('Categorical Features: Distribution')

plt.xticks(rotation=90)

for i, col in enumerate(object\_cols, 1):

plt.subplot(11, 4, i)

plt.xticks(rotation=90)

sns.countplot(data=dataset, x=col)

plt.tight\_layout()

plt.show()

# Drop 'Id' column and handle missing values

dataset.drop(['Id'], axis=1, inplace=True)

dataset['SalePrice'].fillna(dataset['SalePrice'].mean(), inplace=True)

new\_dataset = dataset.dropna()

# One-hot encode categorical variables

OH\_encoder = OneHotEncoder(sparse=False)

OH\_cols = pd.DataFrame(OH\_encoder.fit\_transform(new\_dataset[object\_cols]))

OH\_cols.index = new\_dataset.index

OH\_cols.columns = OH\_encoder.get\_feature\_names(object\_cols)

df\_final = pd.concat([new\_dataset[num\_cols], OH\_cols], axis=1)

# Prepare data for modeling

X = df\_final.drop(['SalePrice'], axis=1)

Y = df\_final['SalePrice']

X\_train, X\_valid, Y\_train, Y\_valid = train\_test\_split(X, Y, train\_size=0.8, test\_size=0.2, random\_state=0)

# SVR model

model\_SVR = SVR()

model\_SVR.fit(X\_train, Y\_train)

Y\_pred = model\_SVR.predict(X\_valid)

mape\_SVR = mean\_absolute\_percentage\_error(Y\_valid, Y\_pred)

print("MAPE for SVR:", mape\_SVR)

# Random Forest model

model\_RFR = RandomForestRegressor(n\_estimators=10)

model\_RFR.fit(X\_train, Y\_train)

Y\_pred = model\_RFR.predict(X\_valid)

mape\_RFR = mean\_absolute\_percentage\_error(Y\_valid, Y\_pred)

print("MAPE for Random Forest:", mape\_RFR)

# Linear Regression model

model\_LR = LinearRegression()

model\_LR.fit(X\_train, Y\_train)

Y\_pred = model\_LR.predict(X\_valid)

mape\_LR = mean\_absolute\_percentage\_error(Y\_valid, Y\_pred)

print("MAPE for Linear Regression:", mape\_LR)

**Implement Naiive Bayes**

import numpy as np

import pandas as pd

#Importing the dataset

"""

Next, we import or read the dataset. Click here to download the breast cancer dataset used in this implementation

"""

dataset = pd.read\_csv("breastcancer.csv")

X = dataset.iloc[:, :-1].values

y = dataset.iloc[:, -1].values

"""

Splitting the dataset into the Training set and Test set

Once the dataset is read into the memory, next, divide the dataset into two parts, training and

testing using the train\_test\_split function from sklearn.

The test\_size and random\_state attributes are set to 0.25 and 0 respectively.

You can change these attributes as per your requirements.

"""

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

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.25, random\_state = 0)

#Feature Scaling

"""

Feature scaling is the process of converting the data into a min-max range. In this case,

the standard scalar method is used.

"""

from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

X\_train = sc.fit\_transform(X\_train)

X\_test = sc.transform(X\_test)

"""

Training the Naive Bayes Classification model on the Training set

The GaussianNB function is imported from sklearn.naive\_bayes library. The hyperparameters such as kernel,

and random\_state to linear, and 0 respectively. The remaining hyperparameters of the support vector machine

algorithm are set to default values.

"""

from sklearn.naive\_bayes import GaussianNB

classifier = GaussianNB()

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

#Naive Bayes classifier model

GaussianNB(priors=None, var\_smoothing=1e-09)

#Display the results (confusion matrix and accuracy)

"""

Here evaluation metrics such as confusion matrix and accuracy are used to evaluate the performance of

the model built using a decision tree classifier.

"""

from sklearn.metrics import confusion\_matrix, accuracy\_score

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

cm = confusion\_matrix(y\_test, y\_pred)

print(cm)

a=accuracy\_score(y\_test, y\_pred)

print(a)

**IRIS KNN**

import pandas as pd

import matplotlib.pyplot as plt

import numpy as np

iris = pd.read\_csv("iris.csv")

#first five rows of this dataset:

print(iris.head())

print(iris.describe())

#The target labels of this dataset are present in the species column, let’s have a quick look at the target labels:

print("Target Labels", iris["species"].unique())

#plot the data using a scatter plot which will plot the iris species according to the sepal length and sepal width:

import plotly.io as io

io.renderers.default='browser'

import plotly.express as px

fig = px.scatter(iris, x="sepal\_width", y="sepal\_length", color="species")

fig.show()

#Iris Classification Model

x = iris.drop("species", axis=1)

y = iris["species"]

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

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y,test\_size=0.2,random\_state=0)

from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n\_neighbors=1)

knn.fit(x\_train, y\_train)

x\_new = np.array([[6, 2.9, 1, 0.2]])

prediction = knn.predict(x\_new)

print("Prediction: {}".format(prediction))

**Breast Cancer KNN**

import numpy as np

import pandas as pd

dataset = pd.read\_csv("breastcancer.csv")

"""

The breast cancer dataset has the following features: Sample code number, Clump Thickness, Uniformity of Cell Size,

Uniformity of Cell Shape, Marginal Adhesion, Single Epithelial Cell Size, Bare Nuclei, Bland Chromatin,

Normal Nucleoli, Mitosis, Class.

"""

X = dataset.iloc[:, :-1].values

y = dataset.iloc[:, -1].values

dataset.shape

#splitting the dataset into the Training set 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 = 0.20, random\_state = 42)

#Feature Scaling

"""

Feature scaling is the process of converting the data into a given range.

In this case, the standard scalar technique is used.

"""

from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

X\_train = sc.fit\_transform(X\_train)

X\_test = sc.transform(X\_test)

#Training the K-Nearest Neighbors (K-NN) Classification model on the Training set

"""

Once the dataset is scaled, next, the K-Nearest Neighbors (K-NN) classifier algorithm is used to create a model.

The hyperparameters such as n\_neighbors, metric, and p are set to 5, Minkowski, and 2 respectively.

The remaining hyperparameters are set to default values.

"""

from sklearn.neighbors import KNeighborsClassifier

classifier = KNeighborsClassifier(n\_neighbors = 5, metric = 'minkowski', p = 2)

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

"""

Display the results (confusion matrix and accuracy)

Here evaluation metrics such as confusion matrix and accuracy are used to evaluate the performance of the model built using a decision tree classifier.

"""

from sklearn.metrics import confusion\_matrix, accuracy\_score

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

cm = confusion\_matrix(y\_test, y\_pred)

print(cm)

ac=accuracy\_score(y\_test, y\_pred)

print(ac)

**KNN**

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

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import classification\_report, confusion\_matrix

from sklearn import datasets

iris=datasets.load\_iris()

x = iris.data

y = iris.target

print ('sepal-length', 'sepal-width', 'petal-length', 'petal-width')

print(x)

print('class: 0-Iris-Setosa, 1- Iris-Versicolour, 2- Iris-Virginica')

print(y)

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x,y,test\_size=0.3)

#To Training the model and Nearest nighbors K=5

classifier = KNeighborsClassifier(n\_neighbors=5)

classifier.fit(x\_train, y\_train)

#To make predictions on our test data

y\_pred=classifier.predict(x\_test)

print('Confusion Matrix')

print(confusion\_matrix(y\_test,y\_pred))

print('Accuracy Metrics')

print(classification\_report(y\_test,y\_pred))

**Linear And Polynomial Regression**

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

dataset = pd.read\_csv('Position\_Salaries.csv')

X = dataset.iloc[:, 1:-1].values

y = dataset.iloc[:, -1].values

"""

Training the Linear Regression model on the Whole dataset

A Linear regression algorithm is used to create a model.

A LinearRegression function is imported from sklearn.linear\_model library.

"""

from sklearn.linear\_model import LinearRegression

lin\_reg = LinearRegression()

lin\_reg.fit(X, y)

"""

Training the Polynomial Regression model on the Whole dataset

A polynomial regression algorithm is used to create a model.

"""

from sklearn.preprocessing import PolynomialFeatures

poly\_reg = PolynomialFeatures(degree = 4)

X\_poly = poly\_reg.fit\_transform(X)

lin\_reg\_2 = LinearRegression()

lin\_reg\_2.fit(X\_poly, y)

"""

Visualising the Linear Regression results

Here scatter plot is used to visualize the results. The title of the plot is set to Truth or Bluff

(Linear Regression), xlabel is set to Position Level , and ylabel is set to Salary.

"""

plt.scatter(X, y, color = 'red')

plt.plot(X, lin\_reg.predict(X), color = 'blue')

plt.title('Truth or Bluff (Linear Regression)')

plt.xlabel('Position Level')

plt.ylabel('Salary')

plt.show()

#Visualising the Polynomial Regression results

"""

The title of the plot is set to Truth or Bluff (Polynomial Regression), xlabel is set to Position level,

and ylabel is set to Salary.

"""

plt.scatter(X, y, color = 'red')

plt.plot(X, lin\_reg\_2.predict(poly\_reg.fit\_transform(X)), color = 'blue')

plt.title('Truth or Bluff (Polynomial Regression)')

plt.xlabel('Position level')

plt.ylabel('Salary')

plt.show()

**Linear Regression**

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

dataset = pd.read\_csv('Position\_Salaries.csv')

X = dataset.iloc[:, 1:2].values

y = dataset.iloc[:, -1].values

dataset.head()

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

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 1/3, random\_state = 0)

from sklearn.linear\_model import LinearRegression

regressor = LinearRegression()

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

#LinearRegression(copy\_X=True, fit\_intercept=True, n\_jobs=None)

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

pd.DataFrame(data={'Actuals': y\_test, 'Predictions': y\_pred})

#Visualising the Training set results Here scatter plot is used to visualize the results.

plt.scatter(X\_train, y\_train, color = 'red')

plt.plot(X\_train, regressor.predict(X\_train), color = 'blue')

plt.title('Salary vs Experience (Training set)')

plt.xlabel('Years of Experience')

plt.ylabel('Salary')

plt.show()

**Logistic Regression**

import numpy as np

import pandas as pd

#"Importing the dataset

"""

After importing the necessary libraries, next, we import or read the dataset.

Click here to download the breast cancer dataset used in this implementation.

The breast cancer dataset has the following features:

Sample code number, Clump Thickness, Uniformity of Cell Size, Uniformity of Cell Shape, Marginal Adhesion,

Single Epithelial Cell Size, Bare Nuclei, Bland Chromatin, Normal Nucleoli, Mitosis, Class.

"""

# divide the dataset into concepts and targets. Store the concepts into X and targets into y.

dataset = pd.read\_csv("breastcancer.csv")

X = dataset.iloc[:, :-1].values

y = dataset.iloc[:, -1].values

#Splitting the dataset into the Training set and Test

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

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.30, random\_state = 2)

#Feature Scaling

"""

Feature scaling is the process of converting the data into a given range. In this case, the standard scalar technique is used.

from sklearn.preprocessing import StandardScaler

"""

from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

X\_train = sc.fit\_transform(X\_train)

X\_test = sc.transform(X\_test)

"""

Training the Logistic Regression (LR) Classification model on the Training set

Once the dataset is scaled, next, the Logistic Regression (LR) classifier algorithm is used to create a model.

The hyperparameters such as random\_state to 0 respectively.

The remaining hyperparameters Logistic Regression (LR) are set to default values.

"""

from sklearn.linear\_model import LogisticRegression

classifier = LogisticRegression(random\_state = 0)

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

#Logistic Regression (LR) classifier model

LogisticRegression(C=1.0, class\_weight=None, dual=False, fit\_intercept=True,

intercept\_scaling=1, l1\_ratio=None, max\_iter=100,

multi\_class='warn', n\_jobs=None, penalty='l2',

random\_state=0, solver='warn', tol=0.0001, verbose=0,

warm\_start=False)

#Display the results (confusion matrix and accuracy)

"""

Here evaluation metrics such as confusion matrix and accuracy are used to evaluate the performance of the model

built using a decision tree classifier.

"""

from sklearn.metrics import confusion\_matrix, accuracy\_score

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

cm = confusion\_matrix(y\_test, y\_pred)

print(cm)

accuracy\_score(y\_test, y\_pred)

**Mobile Price**

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

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

from sklearn.preprocessing import StandardScaler

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import accuracy\_score

sns.set()

import plotly.io as io

io.renderers.default='browser'

data = pd.read\_csv("mobile\_prices.csv")

print(data.head())

plt.figure(figsize=(12, 10))

sns.heatmap(data.corr(), annot=True, cmap="coolwarm", linecolor='white', linewidths=1)

#data preparation

x = data.iloc[:, :-1].values

y = data.iloc[:, -1].values

x = StandardScaler().fit\_transform(x)

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size=0.20, random\_state=0)

# Logistic Regression algorithm provided by Scikit-learn:

from sklearn.linear\_model import LogisticRegression

lreg = LogisticRegression()

lreg.fit(x\_train, y\_train)

y\_pred = lreg.predict(x\_test)

#accuracy of the model:

accuracy = accuracy\_score(y\_test, y\_pred) \* 100

print("Accuracy of the Logistic Regression Model: ",accuracy)

#predictions made by the model:

print(y\_pred)

#Let’s have a look at the number of mobile phones classified for each price range:

(unique, counts) = np.unique(y\_pred, return\_counts=True)

price\_range = np.asarray((unique, counts)).T

print(price\_range)

**Naiive IRIS Classfication**

from sklearn.naive\_bayes import GaussianNB

from sklearn.naive\_bayes import MultinomialNB

from sklearn import datasets

from sklearn.metrics import confusion\_matrix

iris = datasets.load\_iris()

gnb = GaussianNB()

mnb = MultinomialNB()

y\_pred\_gnb = gnb.fit(iris.data, iris.target).predict(iris.data)

cnf\_matrix\_gnb = confusion\_matrix(iris.target, y\_pred\_gnb)

print(cnf\_matrix\_gnb)

y\_pred\_mnb = mnb.fit(iris.data, iris.target).predict(iris.data)

cnf\_matrix\_mnb = confusion\_matrix(iris.target, y\_pred\_mnb)

print(cnf\_matrix\_mnb)

**Naiive Bayes**

import numpy as np

import pandas as pd

#Importing the dataset

"""

Next, we import or read the dataset. Click here to download the breast cancer dataset used in this implementation.

After reading the dataset, divide the dataset into concepts and targets. Store the concepts into X and

targets into y.

"""

dataset = pd.read\_csv("breastcancer.csv")

X = dataset.iloc[:, :-1].values

y = dataset.iloc[:, -1].values

"""

Splitting the dataset into the Training set and Test set

Once the dataset is read into the memory, next, divide the dataset into two parts, training and

testing using the train\_test\_split function from sklearn.

The test\_size and random\_state attributes are set to 0.25 and 0 respectively.

You can change these attributes as per your requirements.

"""

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

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.25, random\_state = 0)

#Feature Scaling

"""

Feature scaling is the process of converting the data into a min-max range. In this case,

the standard scalar method is used.

"""

from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

X\_train = sc.fit\_transform(X\_train)

X\_test = sc.transform(X\_test)

"""

Training the Naive Bayes Classification model on the Training set

Once the dataset is scaled, next, the Naive Bayes classifier algorithm is used to create a model.

The GaussianNB function is imported from sklearn.naive\_bayes library. The hyperparameters such as kernel,

and random\_state to linear, and 0 respectively. The remaining hyperparameters of the support vector machine

algorithm are set to default values.

"""

from sklearn.naive\_bayes import GaussianNB

classifier = GaussianNB()

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

#Naive Bayes classifier model

GaussianNB(priors=None, var\_smoothing=1e-09)

#Display the results (confusion matrix and accuracy)

"""

Here evaluation metrics such as confusion matrix and accuracy are used to evaluate the performance of

the model built using a decision tree classifier.

"""

from sklearn.metrics import confusion\_matrix, accuracy\_score

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

cm = confusion\_matrix(y\_test, y\_pred)

print(cm)

accuracy\_score(y\_test, y\_pred)

**Perception IRIS Classfication**

from sklearn import datasets

import numpy as np

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

from sklearn.linear\_model import Perceptron

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import accuracy\_score

iris = datasets.load\_iris()

X = iris.data[:, [2, 3]]

y = iris.target

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

X, y, test\_size=0.3, random\_state=1, stratify=y)

sc = StandardScaler()

sc.fit(X\_train)

X\_train\_std = sc.transform(X\_train)

X\_test\_std = sc.transform(X\_test)

ppn = Perceptron(eta0=0.1, random\_state=1)

ppn.fit(X\_train\_std, y\_train)

y\_pred = ppn.predict(X\_test\_std)

print('Accuracy: %.3f' % accuracy\_score(y\_test, y\_pred))

print('Accuracy: %.3f' % ppn.score(X\_test\_std, y\_test))