**Hindi Vidya Prachar Samiti’s**

**RAMNIRANJAN JHUNJHUNWALA COLLEGE OF ARTS, SCIENCE, AND COMMERCE**

**(EMPOWERED AUTONOMOUS)**

**Introduction to Supervised Learning**

****

**Name: Surajkumar Yadav**

**Roll no: 10302**

**Class: MSc. Data Science and Artificial Intelligence Part I**



**Ramniranjan Jhunjhunwala College**

**of Arts, Science and Commerce**

**Department of Data Science and Artificial Intelligence**

**CERTIFICATE**

**This is to certify Surajkumar Yadav of MSc. Data Science and Artificial Intelligence Roll No. 10302 has successfully completed the practical of Introduction to Supervised Learning during the Academic Year 2024-2025.**

**Date:**

**(Prof. Mujtaba Shaikh) External Examiner Prof-In-Charge**

**INDEX**

| **Sr No.** | **Practical Name** | **Date** |
| --- | --- | --- |
| **1** | **Feature Engineering** | **-** |
| **1.1** | **Data Imputation** | **18/7/24** |
| **1.2** | **Handling Categorical Variable** | **18/7/24** |
| **1.3** | **Feature Scaling** | **19/7/24** |
| **1.4** | **Feature Selection** | **20/7/24** |
| **2** | **Linear Regression** | **-** |
| **2.1** | **Simple Linear Regression** | **25/7/24** |
| **2.2** | **Multiple Linear Regression** | **27/7/24** |
| **2.3** | **Polynomial Linear Regression** | **27/7/24** |
| **3** | **Regularization** | **-** |
| **3.1** | **L1 – Lasso Regression** | **1/8/24** |
| **3.2** | **L2 – Ridge Regression** | **1/8/24** |
| **4** | **Classification** | **-** |
| **4.1** | **Logistic Regression** | **1/8/24** |
| **4.2** | **Decision Tree** | **8/8/24** |
| **4.3** | **Random Forest** | **8/8/24** |
| **4.4** | **Naïve Bayes** | **9/8/24** |

**Practical 1**

**Feature Engineering**

**1.1 Data Imputation**

**Code:**

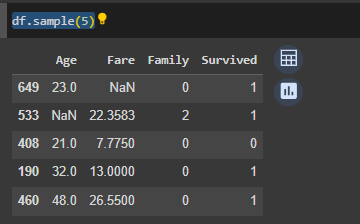
import pandas as pd

import matplotlib.pyplot as plt

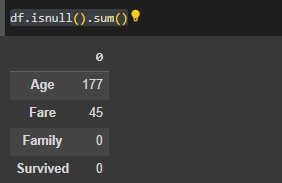
import seaborn as sns

df=pd.read\_csv('/content/drive/MyDrive/supervisedlearning/datasets /titanic\_toy.csv')

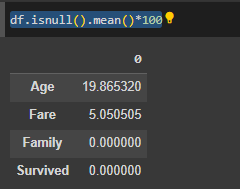
df.sample(5)

****

df.isnull().sum()



df.isnull().mean()\*100



* **finding mean median mode for age and fare column**

age\_mean= df['Age'].mean()

age\_median=df['Age'].median()

age\_mode=df['Age'].mode()

fare\_mean=df['Fare'].mean()

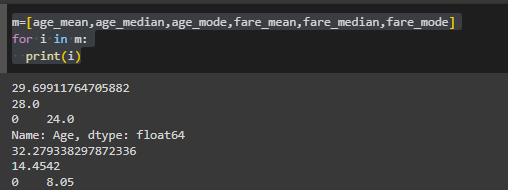
fare\_median=df['Fare'].median()

fare\_mode=df['Fare'].mode()

m=[age\_mean,age\_median,age\_mode,fare\_mean,fare\_median,fare\_mode]

for i in m:

print(i)

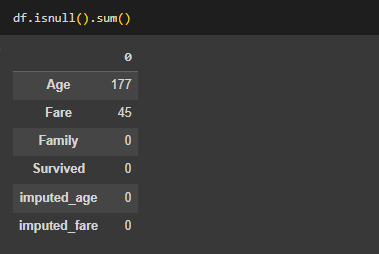


* **Replacing null value**

df['imputed\_age\_mean']=df['Age'].fillna(age\_mean)

df['imputed\_fare\_mean']=df['Fare'].fillna(fare\_mean)

df.sample(5)

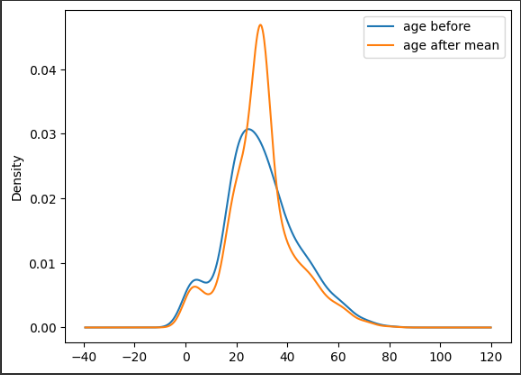


* **using mean**

df['Age'].plot(kind='kde',label='age before')

df['imputed\_age\_mean'].plot(kind='kde',label='age after mean')

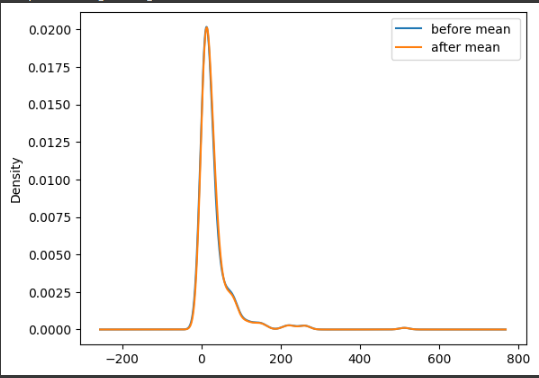
plt.legend()

****

df['Fare'].plot(kind='kde',label='before mean ')

df['imputed\_fare\_mean'].plot(kind='kde',label='after mean ')

plt.legend()



* **using median**

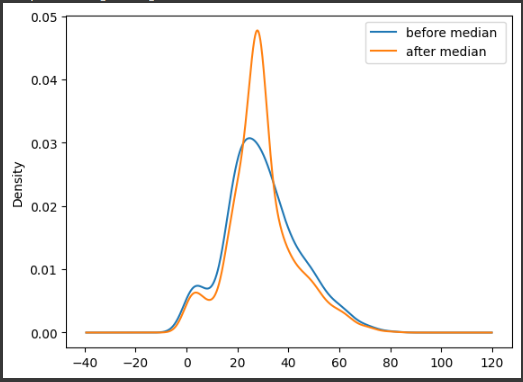
df['imputed\_age\_median']=df['Age'].fillna(age\_median)

df['imputed\_fare\_median']=df['Fare'].fillna(fare\_median)

df['Age'].plot(kind='kde',label='before median ')

df['imputed\_age\_median'].plot(kind='kde',label='after median ')

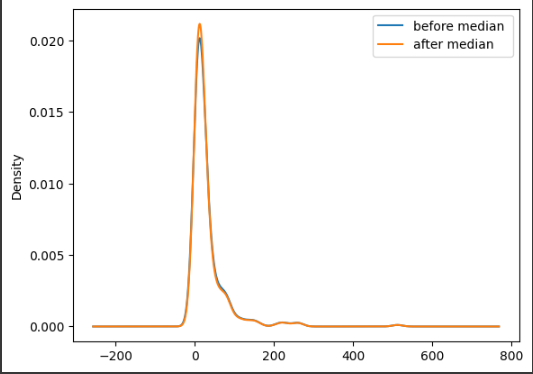
plt.legend()



df['Fare'].plot(kind='kde',label='before median ')

df['imputed\_fare\_median'].plot(kind='kde',label='after median ')

plt.legend()



* **using mode**

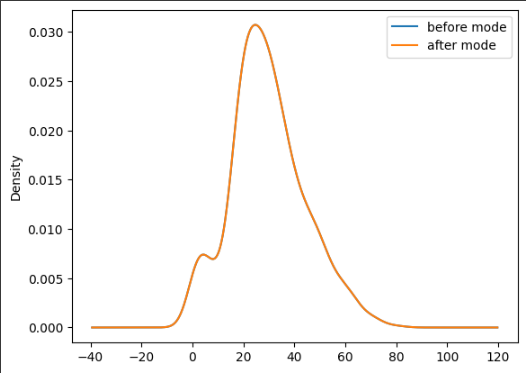
df['imputed\_age\_mode']=df['Age'].fillna(age\_mode)

df['imputed\_fare\_mode']=df['Fare'].fillna(fare\_mode)

df['Age'].plot(kind='kde',label='before mode')

df['imputed\_age\_mode'].plot(kind='kde',label='after mode')

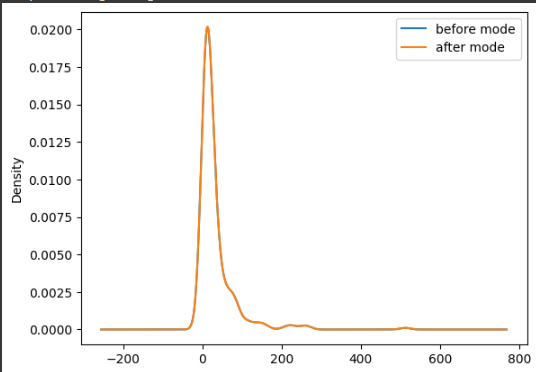
plt.legend()



df['Fare'].plot(kind='kde',label='before mode')

df['imputed\_fare\_mode'].plot(kind='kde',label='after mode')

plt.legend()



for age, we will choose the mode as it fits the perfect curve and for fare, we will choose either median or mode as it fits the curve

**1.2 Handling Categorical Variable**

import pandas as pd

from sklearn.preprocessing import LabelEncoder

data=pd.DataFrame({

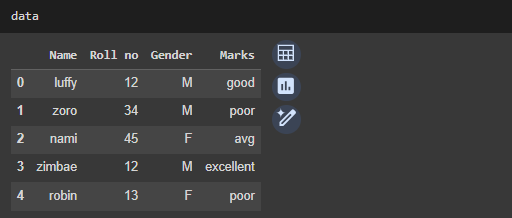
'Name':['luffy','zoro','nami','zimbae','robin'],

'Roll no':[12,34,45,12,13],

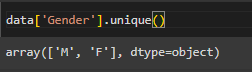
'Gender':['M','M','F','M','F'],

'Marks':['good','poor','avg','excellent','poor']

})

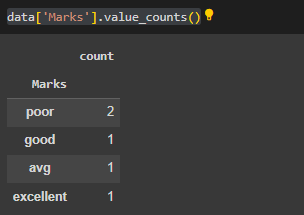
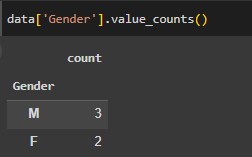


data['Gender'].unique()



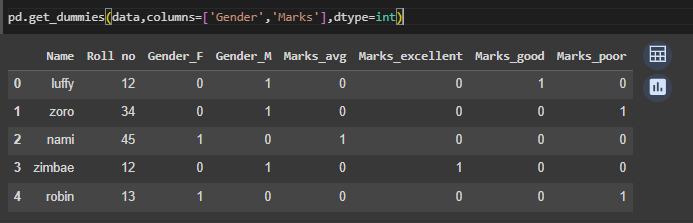
data['Gender'].value\_counts()

data['Marks'].value\_counts()



* **using pandas to perform one hot encoding**

pd.get\_dummies(data,columns=['Gender','Marks'],dtype=int)



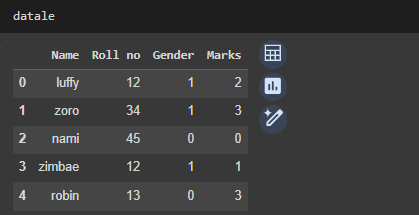
* **using scikit-learn to perform one hot encoding**

le= LabelEncoder()

datale=data

datale['Gender']=le.fit\_transform(data['Gender'])

datale['Marks']=le.fit\_transform(data['Marks'])



**1.3 Feature Scaling**

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.metrics import confusion\_matrix

from sklearn.preprocessing import StandardScaler,MinMaxScaler

df=pd.read\_csv('/content/drive/MyDrive/supervisedlearning/datasets /Social\_Network\_Ads.csv')

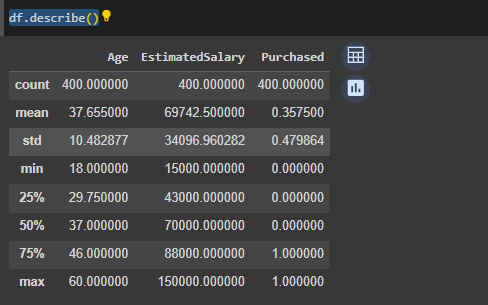




* **Dropping Columns**

df.drop(columns=['User ID','Gender'],axis=0,inplace=True)

df.describe()



* **Performing Normalization and Standardization**

std=StandardScaler()

norm=MinMaxScaler()

df[['std\_age','std\_salary']]=std.fit\_transform(df.iloc[:,0:2])

df[['norm\_age','norm\_salary']]=norm.fit\_transform(df.iloc[:,0:2])

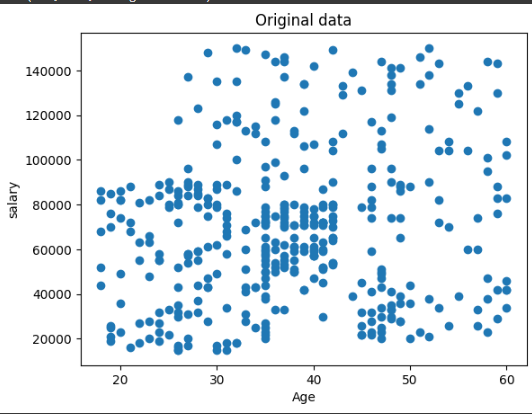


plt.scatter(df['Age'],df['std\_salary'])

plt.xlabel('Age')

plt.ylabel('salary')

plt.title("Original data")

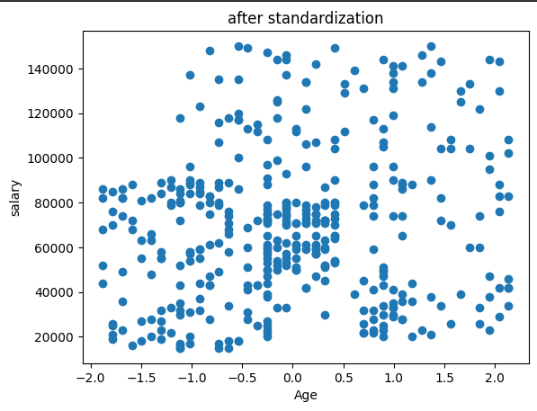


plt.scatter(df['std\_age'],df['EstimatedSalary'])

plt.xlabel('Age')

plt.ylabel('salary')

plt.title("after standardization")

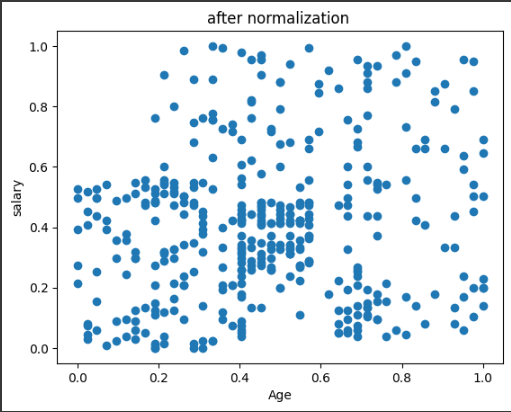


plt.scatter(df['norm\_age'],df['norm\_salary'])

plt.xlabel('Age')

plt.ylabel('salary')

plt.title("after normalization ")



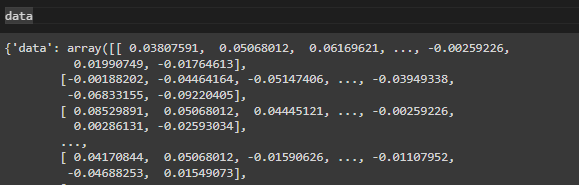
**1.4 Feature Selection**

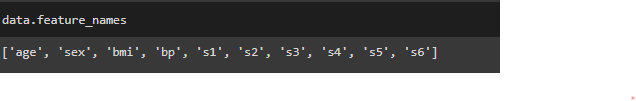
import pandas as pd

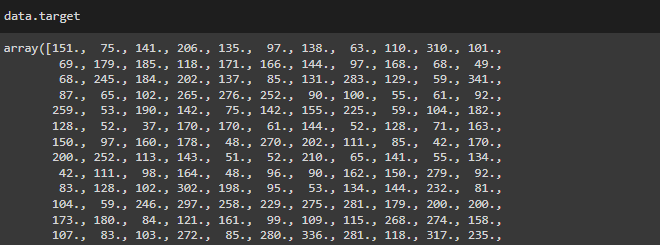
from sklearn.datasets import load\_diabetes

import seaborn as sns

data=load\_diabetes()



****

****

* **Creating dataframe**

ind=data.data

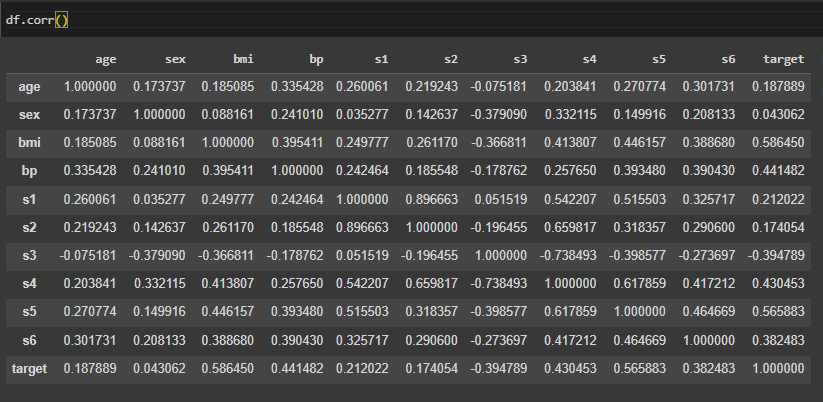
dep=data.target

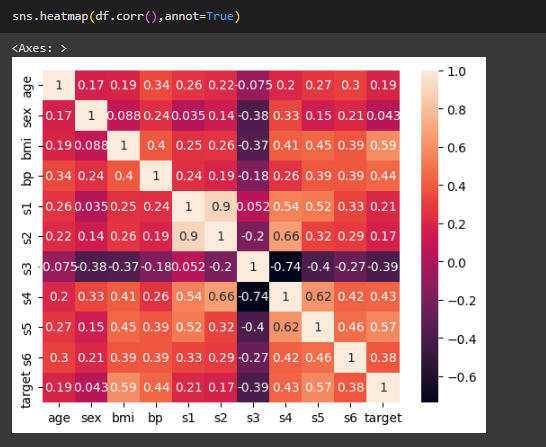
x=pd.DataFrame(ind,columns=data.feature\_names)

y=pd.DataFrame(dep,columns=['target'])

df=pd.concat([x,y],axis=1)

df.corr()





**Practical 2**

**Linear Regression**

**2.1 Simple Linear Regression**

import pandas as pd

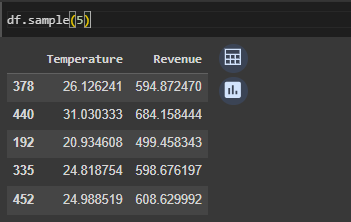
import matplotlib.pyplot as plt

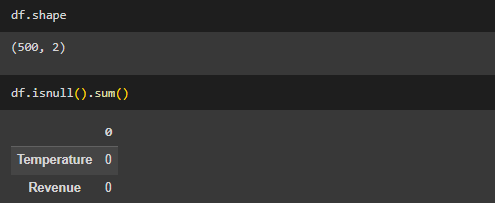
import seaborn as sns

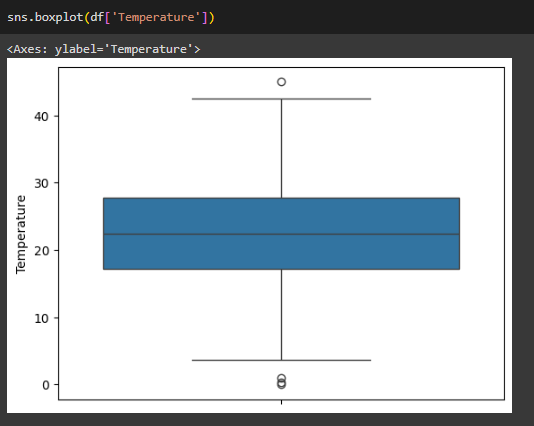
from sklearn.preprocessing import StandardScaler,MinMaxScaler

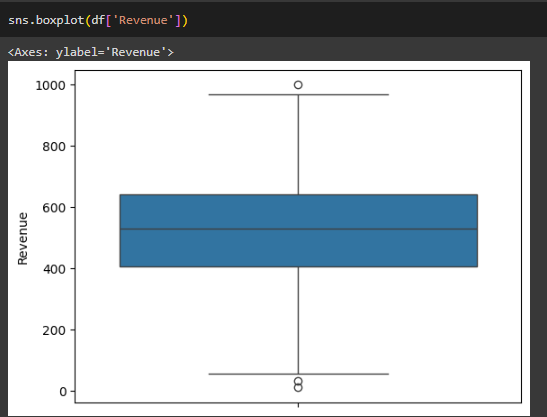
import numpy as np

df = pd.read\_csv('/content/drive/MyDrive/supervisedlearning/datasets /IceCreamData.csv')

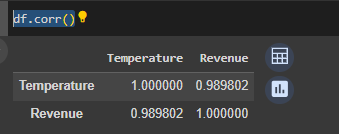








df.corr()

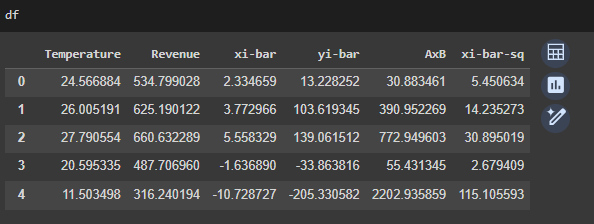


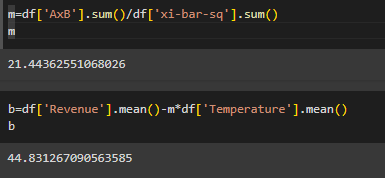
df['xi-bar']=df['Temperature']-df['Temperature'].mean()

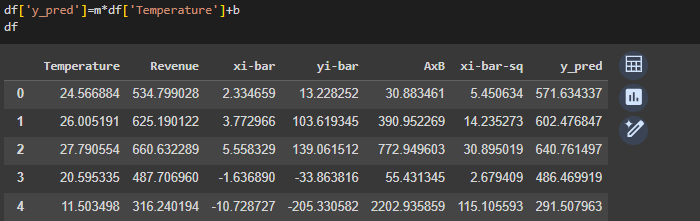
df['yi-bar']=df['Revenue']-df['Revenue'].mean()

df['AxB']=df['xi-bar']\*df['yi-bar']

df['xi-bar-sq']=np.square(df['xi-bar'])







* **Linear Regression model**

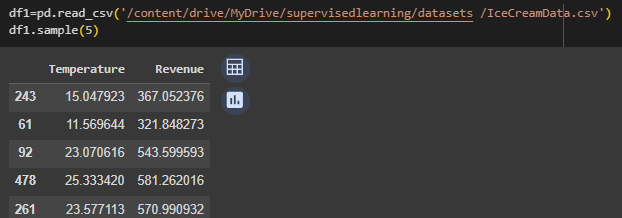
from sklearn.linear\_model import LinearRegression

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

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

mse=mean\_squared\_error(df['Temperature'],df['y\_pred'])

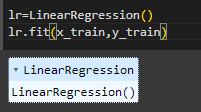
mae=mean\_absolute\_error(df['Temperature'],df['y\_pred'])



x\_train,x\_test,y\_train,y\_test = train\_test\_split(df1.drop('Revenue',axis=1),df1['Revenue'],test\_size=0.3,random\_state=42)

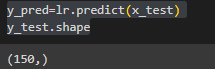
lr=LinearRegression()

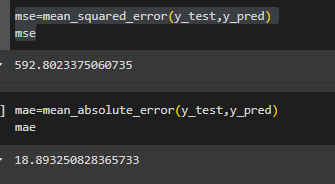
lr.fit(x\_train,y\_train)



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

y\_test.shape





plt.scatter(df['Temperature'],df['Revenue'],color='blue',label='Actual data ')

plt.plot(x\_test,y\_pred,color='red',linewidth=2)

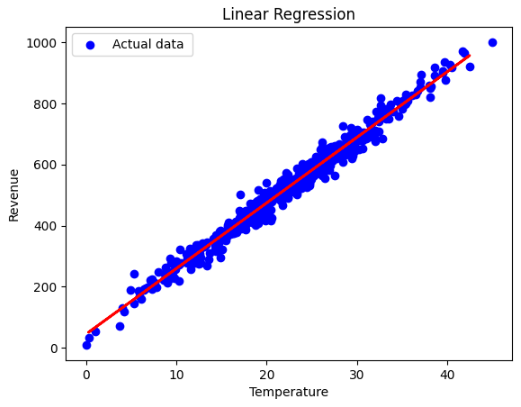
plt.xlabel('Temperature')

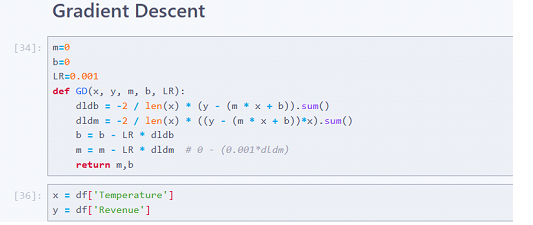
plt.ylabel('Revenue')

plt.title('Linear Regression')

plt.legend()

plt.show()







**2.2 Multiple Linear Regression**

import pandas as pd

import numpy as np

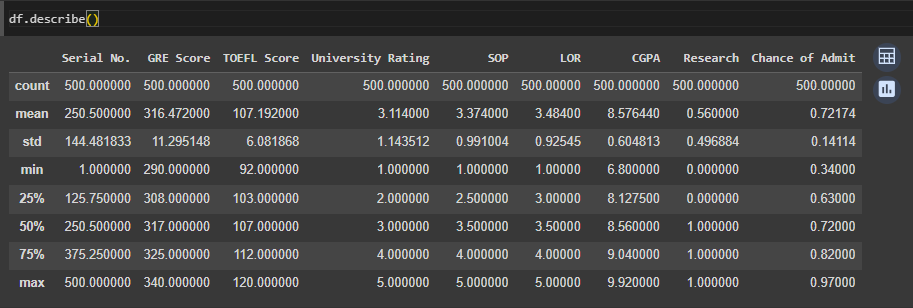
from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LinearRegression

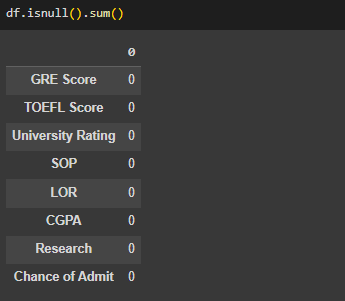
from sklearn.metrics import mean\_absolute\_error,mean\_squared\_error,r2\_score

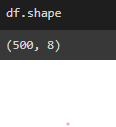
import sklearn.metrics as metrics

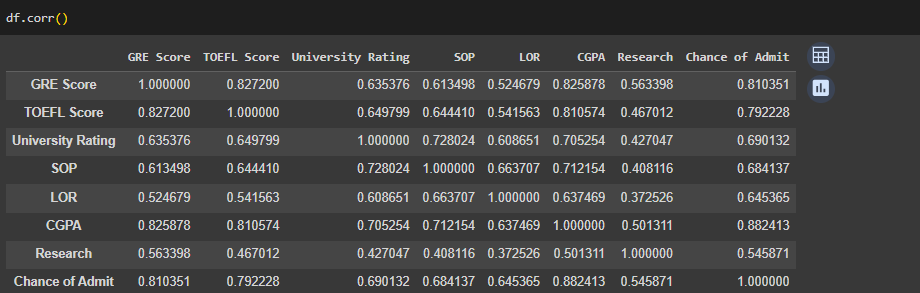
df=pd.read\_csv('/content/drive/MyDrive/supervisedlearning/datasets /Admission.csv')



df.drop('Serial No.',axis=1, inplace=True)







x\_train,x\_test,y\_train,y\_test=train\_test\_split(df.drop('Chance of Admit',axis=1),df['Chance of Admit'],test\_size=0.3,random\_state=42)

lr=LinearRegression()

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

#beta values m

lr.coef\_



#intercept value

lr.intercept\_



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

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

mae=mean\_absolute\_error(y\_test,y\_pred)

print(mse)

print(mae)



rmse=np.sqrt(mse)

r2 = metrics.r2\_score(y\_test,y\_pred)

print(rmse)

print(r2)



**2.3 Polynomial Linear Regression**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.preprocessing import PolynomialFeatures

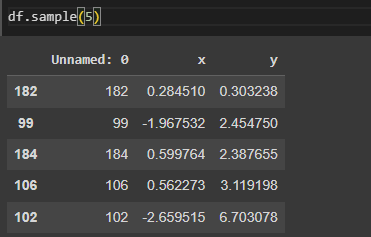
from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LinearRegression

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

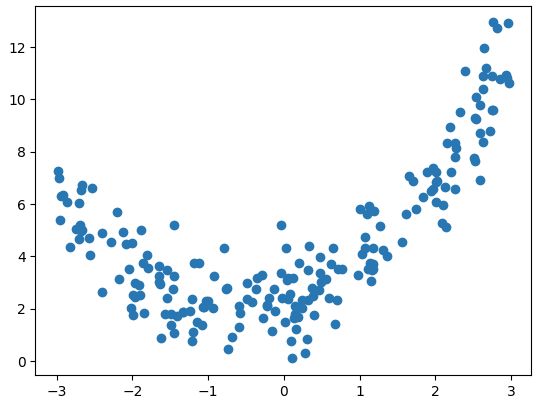
import sklearn.metrics as metrics

df=pd.read\_csv('/content/drive/MyDrive/supervisedlearning/datasets /PolyData.csv')

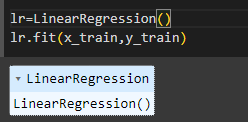


df.drop('Unnamed: 0',axis=1,inplace=True)

plt.scatter(df['x'],df['y'])

****

x\_train,x\_test,y\_train,y\_test=train\_test\_split(df.drop('y',axis=1),df['y'],test\_size=0.3,random\_state=42)



lr.coef\_y\_pred=lr.predict(x\_test)

plt.scatter(df['x'],df['y'],color='blue',label='Actual data')

plt.plot(x\_test,y\_pred, color='red',linewidth=2,label='reggresion liine ')

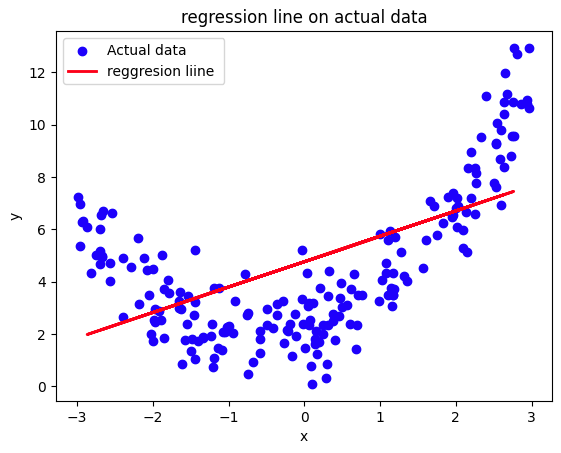
plt.xlabel('x')

plt.ylabel('y')

plt.title("regression line on actual data")

plt.legend()

plt.show()



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

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

mae=mean\_absolute\_error(y\_test,y\_pred)

rmse=np.sqrt(mse)

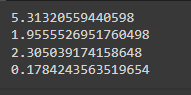
r2=metrics.r2\_score(y\_test,y\_pred)

print(mse)

print(mae)

print(rmse)

print(r2)



* **creating polynomial feature**

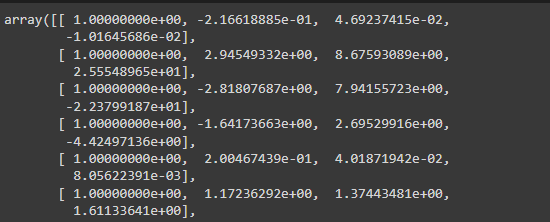
x=df['x'].values.reshape(-1,1)

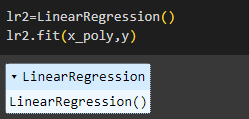
y=df['y'].values.reshape(-1,1)

poly=PolynomialFeatures(degree=3)

x\_poly=poly.fit\_transform(x)

x\_poly

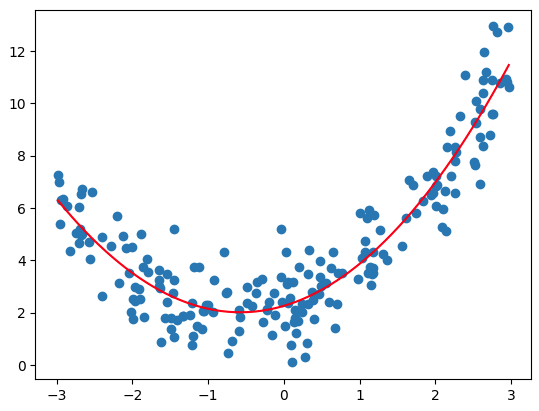




x\_range=np.linspace(min(x),max(x),len(x))

plt.scatter(x,y)

plt.plot(x\_range,lr2.predict(poly.fit\_transform(x\_range.reshape(-1,1))),color='red')



**Practical 3**

**Regularization**

import pandas as pd

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.linear\_model import LinearRegression

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

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

import sklearn.metrics as metrics

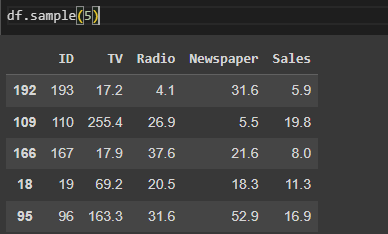
from sklearn.linear\_model import Ridge,Lasso

import warnings

warnings.filterwarnings('ignore')

import plotly.express as px

df=pd.read\_csv('/content/drive/MyDrive/supervisedlearning/datasets /Advertising.csv')

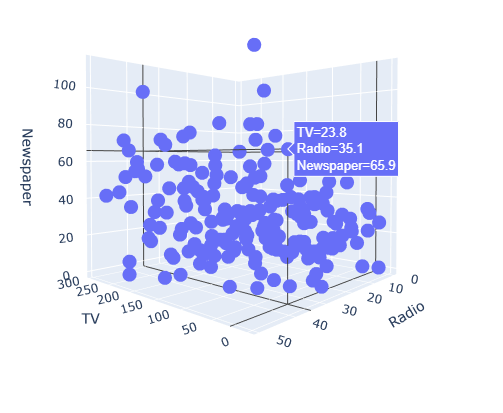


df.drop('ID',axis=1,inplace=True)



x\_train,x\_test,y\_train,y\_test=train\_test\_split(df.drop('Sales',axis=1),df['Sales'],test\_size=0.3,random\_state=42)

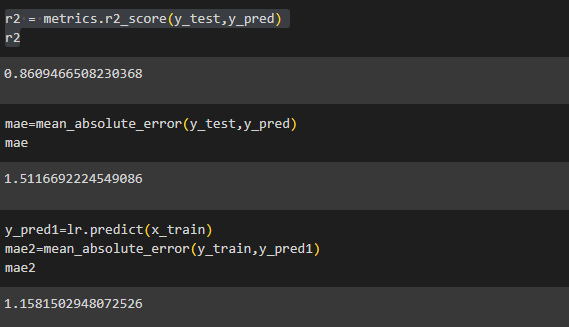
px.scatter\_3d(df,x='TV',y='Radio',z='Newspaper')



lr=LinearRegression()

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

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



**3.1 L2 regularization ridge regression**

rg=Ridge(alpha=3)

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



rg\_pred=rg.predict(x\_test)

r2=metrics.r2\_score(y\_test,rg\_pred)

r2



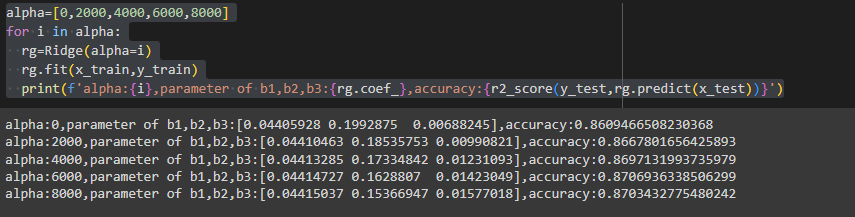
alpha=[0,2000,4000,6000,8000]

for i in alpha:

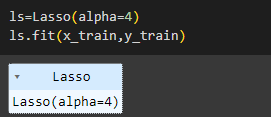
rg=Ridge(alpha=i)

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

print(f'alpha:{i},parameter of b1,b2,b3:{rg.coef\_},accuracy:{r2\_score(y\_test,rg.predict(x\_test))}')



**3.2 L1 regularization Lasso regression**



ls\_pred=ls.predict(x\_test)

r2 = metrics.r2\_score(y\_test,ls\_pred)

r2



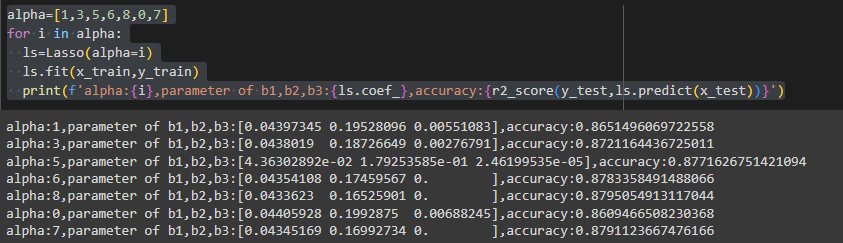
alpha=[1,3,5,6,8,0,7]

for i in alpha:

ls=Lasso(alpha=i)

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

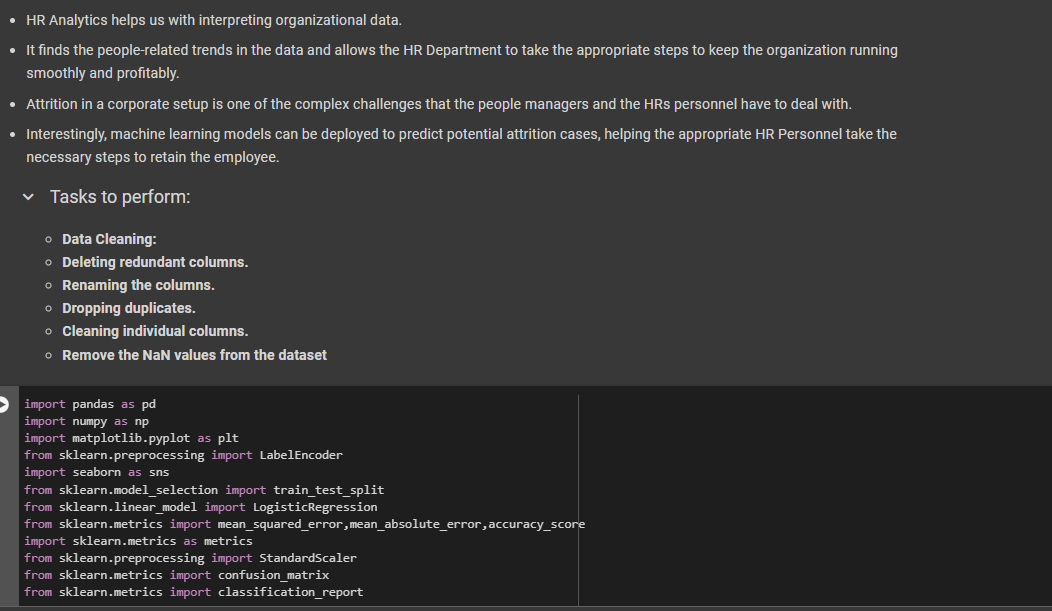
print(f'alpha:{i},parameter of b1,b2,b3:{ls.coef\_},accuracy:{r2\_score(y\_test,ls.predict(x\_test))}')

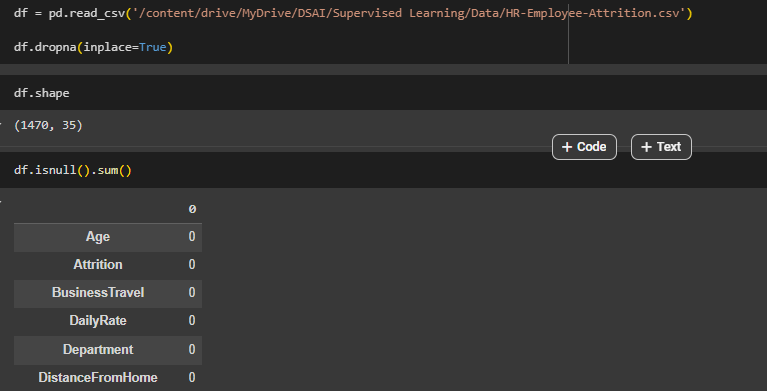


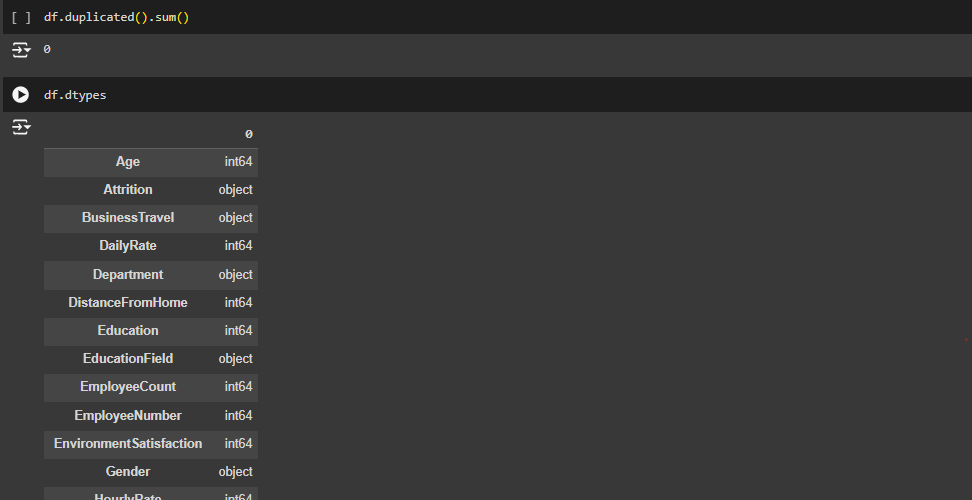
**Practical 4**

**Classification**

**4.1 Logistic Regression**







Converting all the Object columns to Numeric (int) as for correlation we need numeric values.

* **Performing Label Encoding**

categorical\_columns = [col for col in df.columns if

df[col].dtype == 'object']

print(categorical\_columns)

for i in categorical\_columns:

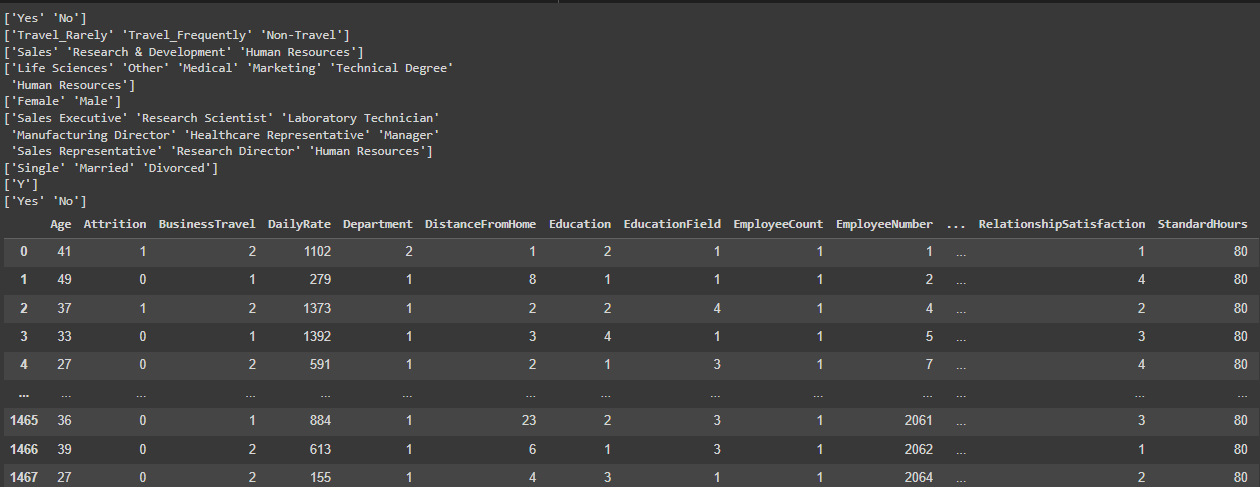
print(df[i].unique())

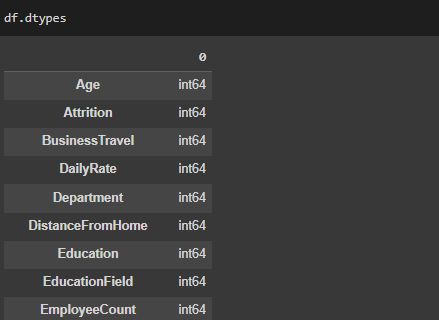
le = LabelEncoder()

for i in categorical\_columns:

df[i]=le.fit\_transform(df[i])

df





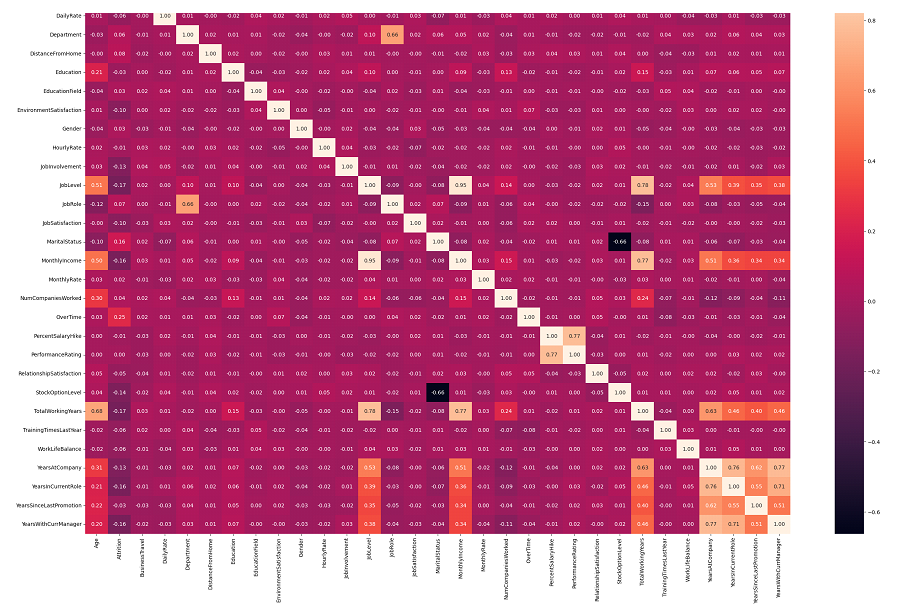
df.corr()

plt.figure(figsize=(30, 20))

sns.heatmap(df.corr(),fmt='.2f',annot=True)

plt.title('Correlation Heatmap')

plt.show()



df.drop(['EmployeeCount', 'EmployeeNumber', 'Over18', 'StandardHours'], axis=1, inplace=True)

To address multicollinearity, features with a correlation coefficient greater than 0.75 will be identified. These highly correlated features will then be compared to determine which ones exhibit stronger correlations with each other rather than with the target variable, Attrition.

cm = df.corr()

# Create a mask to identify the features with a correlation coefficient greater than or equal to 0.75

high\_correlation\_columns = cm >= 0.75

highly\_correlated\_features = []

for feature in high\_correlation\_columns.columns:

correlated\_with =

high\_correlation\_columns.index[high\_correlation\_columns[feature]].tolist()

for correlated\_feature in correlated\_with:

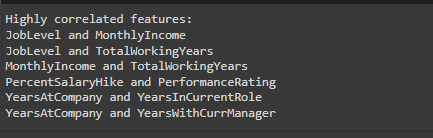
if feature != correlated\_feature and (correlated\_feature, feature) not in highly\_correlated\_features:

highly\_correlated\_features.append((feature, correlated\_feature))

print("Highly correlated features:")

for feature1, feature2 in highly\_correlated\_features:

print(f"{feature1} and {feature2}")



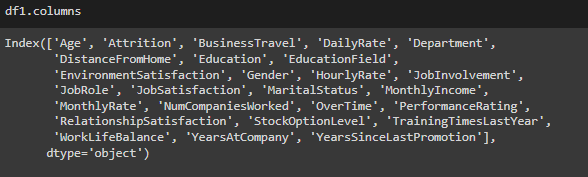
By observation, we will drop those columns that are not likely to contribute more in outcome.

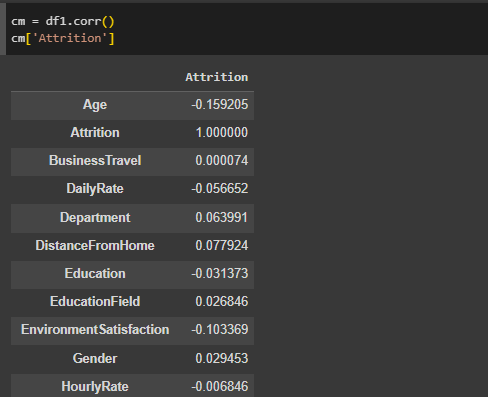
# droping columns which are highly correlated

df1=df

cols = ["JobLevel", "TotalWorkingYears", "PercentSalaryHike", "YearsInCurrentRole", "YearsWithCurrManager"]

df1.drop(columns=cols,axis=1, inplace=True)





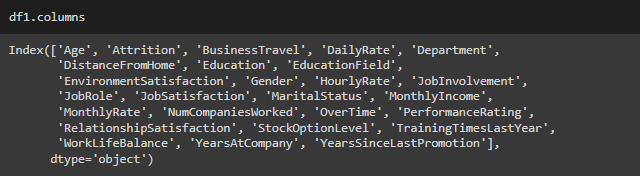
negative\_corr=[]

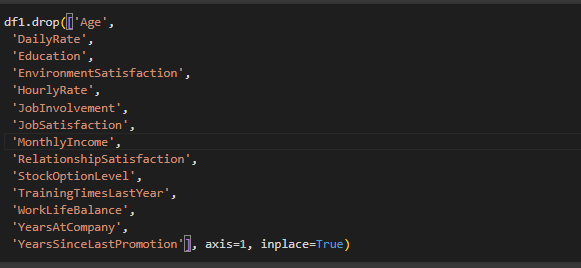
negative\_corr= cm['Attrition'][cm['Attrition'] < 0]

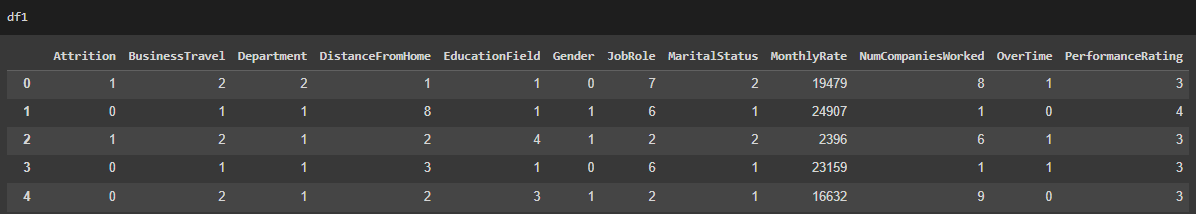
#df=df.drop(negative\_corr, axis=1, inplace=True)

deletecolumns= negative\_corr.index.tolist()

deletecolumns







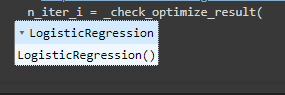
X = df1.drop('Attrition', axis=1)

y = df1['Attrition']

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

model = LogisticRegression()

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



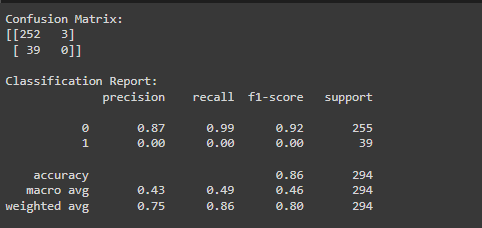
y\_pred = model.predict(X\_test)

print("Confusion Matrix:")

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

print("\nClassification Report:")

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



* **Performing SMOTE Analysis**

import imblearn

from imblearn.over\_sampling import SMOTE

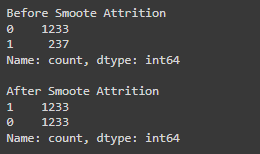
smote = SMOTE()

x\_smote, y\_smote = smote.fit\_resample(X, y)

print("Before Smoote" , y.value\_counts())

print()

print("After Smoote" , y\_smote.value\_counts())



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

scaler = StandardScaler()

X\_train\_scaled = scaler.fit\_transform(X\_train)

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

LR = LogisticRegression()

LR.fit(X\_train\_scaled, y\_train)

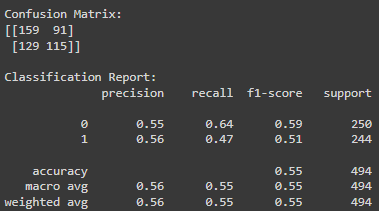
y\_pred = model.predict(X\_test\_scaled)

print("Confusion Matrix:")

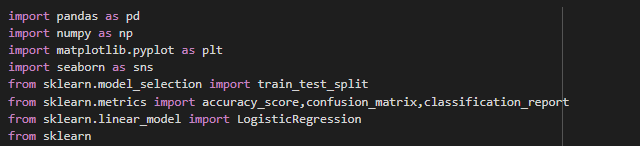
print(confusion\_matrix(y\_test, y\_pred))

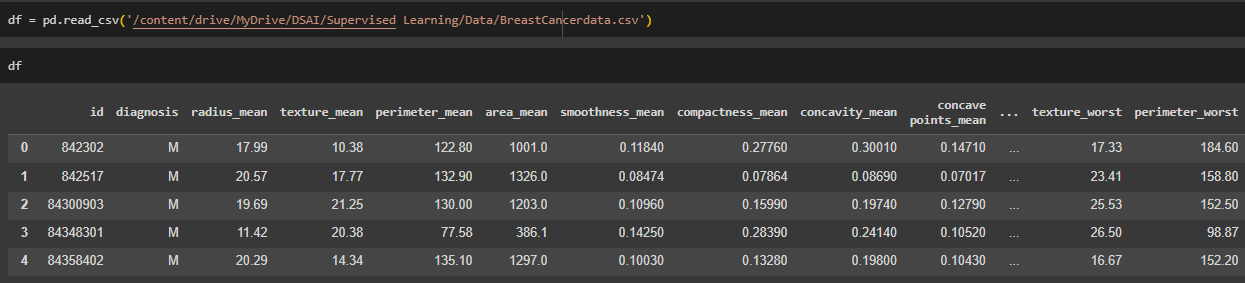
print("\nClassification Report:")

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



**4.2 Decision Tree**





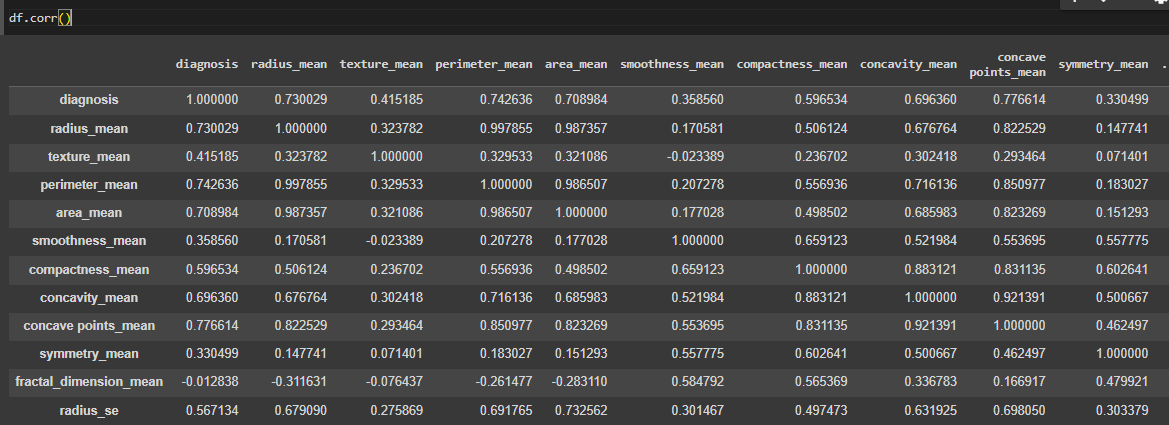
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()

df['diagnosis'] = le.fit\_transform(df['diagnosis'])

df.drop('id','Unnamed: 32',axis=1,inplace=True)

df.corr()



Xtrain,Xtest,Ytrain,Ytest = train\_test\_split(df.drop('diagnosis',axis=1),df['diagnosis'],test\_size=0.2,random\_state=42)

from sklearn.tree import DecisionTreeClassifier

dt = DecisionTreeClassifier()

dt.fit(Xtrain,Ytrain)



y\_pred=dt.predict(Xtest)

accuracy = accuracy\_score(Ytest,y\_pred)

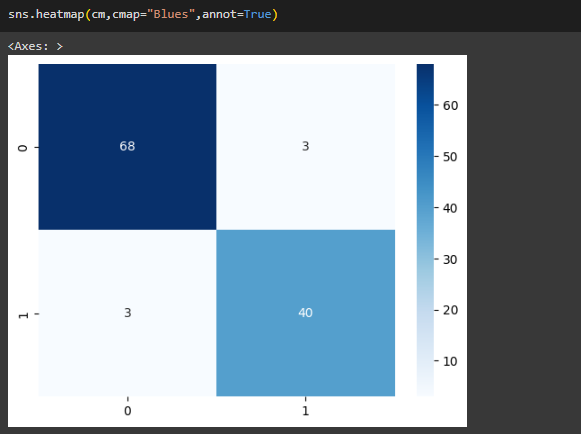
accuracy



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

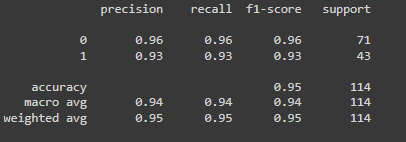
cm





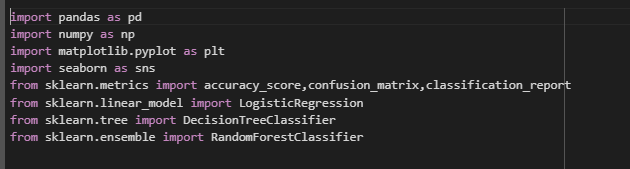
cf= classification\_report(Ytest,y\_pred)

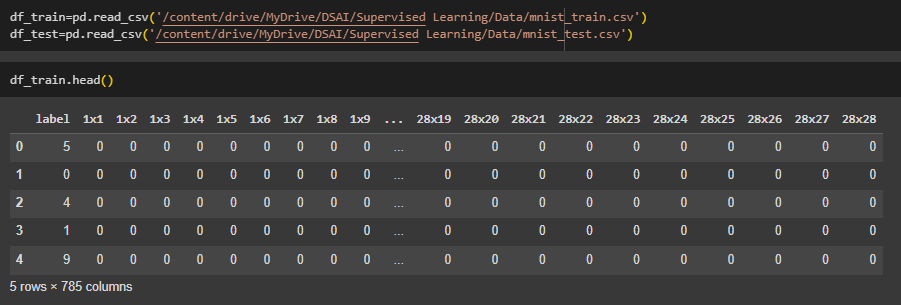
print(cf)





**4.3 Random Forest**





rf=RandomForestClassifier(n\_estimators=150)

x\_train=df\_train.drop('label',axis=1)

y\_train=df\_train['label']

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

****

x\_test=df\_test.drop('label',axis=1)

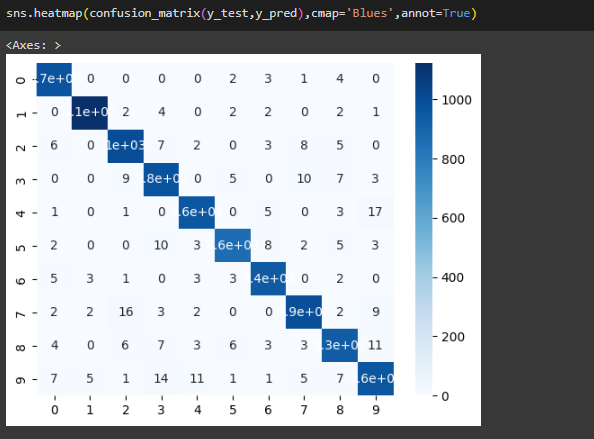
y\_test=df\_test['label']

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

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

accuarcy





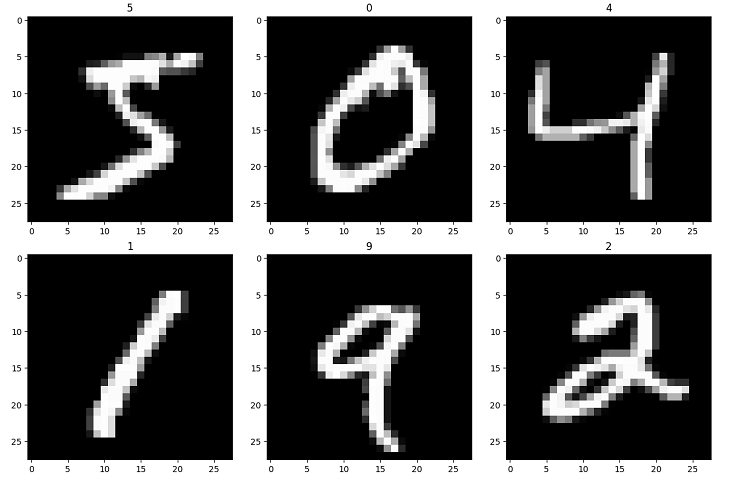
for i in range(0,5):

pixel = df\_train.iloc[i,1:].values.reshape(28,28)

plt.imshow(pixel,cmap='gray')

plt.title(df\_train.iloc[i,0])

plt.show()



**4.4 Naive Bayes**

import pandas as pd

import re

import seaborn as sns

import matplotlib.pyplot as plt

from nltk.corpus import stopwords

from nltk.tokenize import word\_tokenize

from nltk.stem import PorterStemmer

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

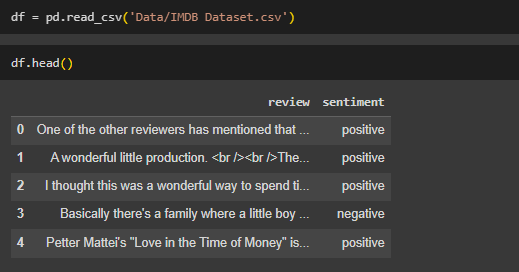
from sklearn.naive\_bayes import MultinomialNB

from sklearn.feature\_extraction.text import CountVectorizer

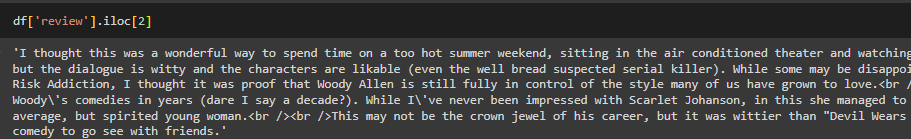
from sklearn.metrics import accuracy\_score,confusion\_matrix,classification\_report

import nltk

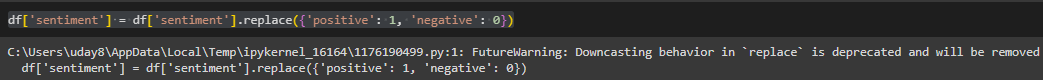
nltk.download('stopwords')

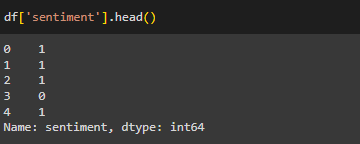


df['review'].iloc[2]



df['sentiment'] = df['sentiment'].replace({'positive': 1, 'negative': 0})





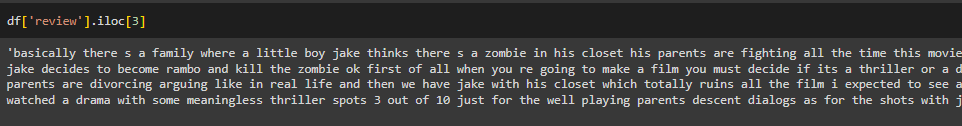
clean = re.compile('<.\*?>')

df['review'] = df['review'].apply(lambda x: re.sub(clean, '', x))

df['review'] = df['review'].str.lower()

df['review'] = df['review'].str.replace(r'[^\w\s]', ' ', regex=True) # Removing Punctuation and Non-Alphanumeric Characters

df['review'] = df['review'].str.replace(r'\s+', ' ', regex=True).str.strip() # Removing Extra Whitespace



* **Removing stop words**

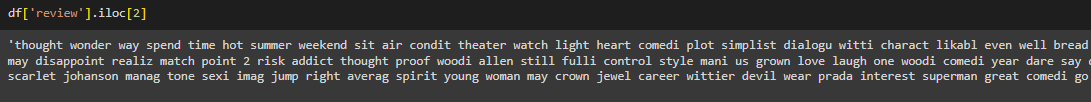
st = set(stopwords.words('english'))

df['review'] = df['review'].apply(lambda x: ' '.join([word for word in x.split() if word not in st]))

* **Stemming**

porter = PorterStemmer()

df['review'] = df['review'].apply(lambda x: ' '.join([porter.stem(word) for word in word\_tokenize(x)]))



* **Vectorization**

vect = CountVectorizer()

X = vect.fit\_transform(df['review'])

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, df['sentiment'], test\_size=0.2, random\_state=42)

* **Naive Bayes model**

nb = MultinomialNB()

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



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

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

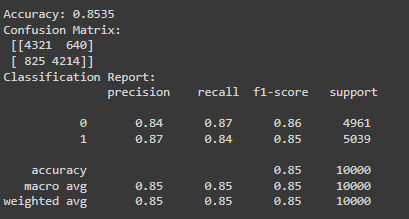
conf\_matrix = confusion\_matrix(y\_test, y\_pred)

class\_report = classification\_report(y\_test, y\_pred)

print(f'Accuracy: {accuracy}')

print('Confusion Matrix:\n', conf\_matrix)

print('Classification Report:\n', class\_report)



sns.heatmap(conf\_matrix, annot=True, fmt='d', cmap='Blues',

xticklabels=['Negative', 'Positive'],

yticklabels=['Negative', 'Positive'])

plt.xlabel('Predicted Label')

plt.ylabel('True Label')

plt.title('Confusion Matrix')

