NANDHA ENGINEERING COLLEGE

**(Autonomous Institution)**

Erode-638 052



LAB MANUAL

RECORD NOTE BOOK

**22CSC13 – FOUNDATIONS OF DATA SCIENCE**

**LABORATORY**

**IV - Semester**

# B.E COMPUTER SCIENCE & ENGINEERING

Department of Computer Science and Engineering

**NANDHA ENGINEERING COLLEGE, ERODE-52**

**NANDHA ENGINEERING COLLEGE**

**(Autonomous Institution)**

Erode-638052



**BONAFIDE CERTIFICATE**

|  |
| --- |
|  |

Register Number:

Certified that this is the Bonafide Record of work done by……………………………………………….of the **IV** Semester **B.E COMPUTER SCIENCE AND ENGINEERING** branch during the Academic Year **2024 – 2025** in the **22CSC13-FOUNDATIONS OF DATA SCIENCE LABORATORY.**

……………………… ……………………….

**Staff-in-charge** **Head of the Department**

Submitted for the End Semester Practical Examination

Held on…………………………………………….

…………………… ……………………..

**Internal Examiner** **External Examiner**

**INDEX**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **EX.**  **NO** | **DATE** | **TITLE OF THE EXPERIMENT** | **PAGE**  **NO** | **MARKS** | **SIGNATURE** |
| 1 |  | Working with Numpy arrays |  |  |  |
| 2 |  | Working with Pandas data frames. |  |  |  |
| 3 |  | Reading data from CSV, Excel and exploring various commands for doing descriptive analytics on the Iris data set. |  |  |  |
| 4 |  | Use the diabetes data set from UCI and Pima Indians Diabetes data set for performing the  following:  a. Univariate analysis: Frequency, Mean, Median, Mode, Variance, Standard Deviation, Skewness and Kurtosis.  b. Bivariate analysis: Linear and logistic regression modelling  c. Multiple Regression analysis  d. Also compare the results of the above analysis for the two data sets. |  |  |  |
| 5 |  | Apply and explore various plotting functions on UCI data sets.  a. Normal curves  b. Density and contour plots  c. Correlation and scatter plots  d. Histograms  e. Three-dimensional plotting |  |  |  |
| 6 |  | Plotting on Geographical data with Basemap |  |  |  |
| 7 |  | Project Report |  |  |  |
| **AVERAGE MARKS AWARDED** | | | |  |  |

|  |  |
| --- | --- |
| **Ex.No: 01** | **Working with Numpy arrays** |
| **Date:** |

**AIM:**

To create and work with numpy arrays.

**PROCEDURE:**

Step-1: Import numpy module.

Step-2: Intialize numpy arrays

Step-3: Explore various array property functions

Step-4: Declare 2D and Multidimensional arrays, and work with indexing and slicing

Step-5: Perform mathematical operations on numpy arrays

Step-6: Further explore other array creation and manipulation methods

**PROGRAM:**

# Install numpy module  
%pip install numpy

# Import numpy module  
import numpy as np

a = np.array([1,2,3,4])  
b = np.array([5,6,7,8])

# Visualization of a  
a

**array([1, 2, 3, 4])**

# Visualization of b  
b

**array([5, 6, 7, 8])**

# Different data types  
c = np.array([4,6,8,10],dtype = "float32")  
c

**array([ 4., 6., 8., 10.], dtype=float32)**

d = np.array([1,0,0,1],dtype = "bool")  
d

**array([ True, False, False, True])**

# Multi dimensional Array  
e = np.array([[2,4],[6,8]],dtype = "i4")  
e

**array([[2, 4],  
 [6, 8]], dtype=int32)**

f = np.array([[1,2,3],[4,5,6],[7,8,9]],dtype = "i4")  
f

**array([[1, 2, 3],  
 [4, 5, 6],  
 [7, 8, 9]], dtype=int32)**

# Indexing Array  
e[1,0]

**np.int32(6)**

# Default Array creation Methods

# Filling zeros  
np.zeros((3,4))

**array([[0., 0., 0., 0.],  
 [0., 0., 0., 0.],  
 [0., 0., 0., 0.]])**

# Filling ones  
np.ones((2,3))

**array([[1., 1., 1.],  
 [1., 1., 1.]])**

# Creating empties  
np.empty((2,2))

**array([[1.10595878e-311, 1.10596071e-311],  
 [1.10595910e-311, 1.10595910e-311]])**

# Custom values  
np.full((3,3),10)

**array([[10, 10, 10],  
 [10, 10, 10],  
 [10, 10, 10]])**

# Random decimal values  
np.random.rand(2,2)

**array([[0.96976695, 0.68712445],  
 [0.86252121, 0.3861171 ]])**

# Random integer values  
np.random.randint(1,2,size = (3,3))

**array([[1, 1, 1],  
 [1, 1, 1],  
 [1, 1, 1]], dtype=int32)**

# Identity matrix  
np.identity(2,dtype = "i4")

**array([[1, 0],  
 [0, 1]], dtype=int32)**

# Range of values  
np.arange(5)

**array([0, 1, 2, 3, 4])**

# Number of decimal values within given range  
np.linspace(1,2,3)

**array([1. , 1.5, 2. ])**

# Basic array operations

# Addition  
a+b

**array([ 6, 8, 10, 12])**

# Subtraction  
a-b

**array([-4, -4, -4, -4])**

# Multiplication  
a\*b

**array([ 5, 12, 21, 32])**

# Division  
a/b

**array([0.2 , 0.33333333, 0.42857143, 0.5 ])**

# Power  
a\*\*b

**array([1, 64, 2187, 65536])**

# More array operations

# Maximum value  
np.max(b)

**np.int64(8)**

# Minimum value  
np.min(a)

**np.int64(1)**

# Sum  
np.sum(a)

**np.int64(10)**

# Dimension of array  
f.ndim

**2**

# Shape of array  
e.shape

**(2, 2)**

# Number of elements in array  
f.size

**9**

# Size of an element  
c.itemsize

**4**

# Number of bytes allotted  
b.nbytes

**32**

# Array to repeat  
np.repeat(e,2)

**array([2, 2, 4, 4, 6, 6, 8, 8], dtype=int32)**

# Copy the array elemts  
g = e.copy()  
g

**array([[2, 4],  
 [6, 8]], dtype=int32)**

# Reshaping the array  
c.reshape((2,2))

**array([[ 4., 6.],  
 [ 8., 10.]], dtype=float32)**

# Horizontaly stack two arrays  
np.hstack((a,b))

**array([1, 2, 3, 4, 5, 6, 7, 8])**

# Change the data type  
b.astype("float32")

**array([5., 6., 7., 8.], dtype=float32)**

# To shuffle randomly  
np.random.permutation(np.arange(10))

**array([1, 0, 3, 8, 9, 5, 2, 7, 4, 6])**

# Matrix multiplication  
h = np.array([[5,6],[7,8]])  
np.matmul(e,h)

**array([[ 38, 44],  
 [ 86, 100]])**

# Matrix transpose  
np.transpose(h)

**array([[5, 7],  
 [6, 8]])**

# To get all values in current dimension here from 2D  
h[:,0]

**array([5, 7])**

**MARK ALLOCATION**

|  |  |  |
| --- | --- | --- |
|  | **Maximum**  **Marks** | **Marks**  **Awarded** |
| Preparation and conduct of Experiment | 50 |  |
| Observation and Result | 30 |  |
| Record | 10 |  |
| Viva Voce | 10 |  |
| Total | 100 |  |

**RESULT:**

Thus, the program has been compiled and executed successfully.

**Signature of the Faculty**

|  |  |
| --- | --- |
| **Ex.No: 02** | **Working with Pandas Data Frames** |
| **Date:** |

**AIM:**

To create and work with DataFrames and Series in pandas.

**PROCEDURE:**

Step-1: Import pandas module

Step-2: Construct DataFrame and Series from various sources

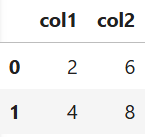
Step-3: Work with insertion, indexing and various functions in DataFrames

**PROGRAM:**

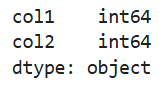
# Install required modules  
%pip install pandas  
%pip install numpy

# Import required modules  
import pandas as pd  
import numpy as np

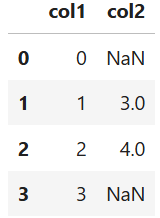
#Constructing DataFrame from a dictionary  
d = {'col1' : [2,4], 'col2' : [6,8]}  
df = pd.DataFrame(data = d)  
df



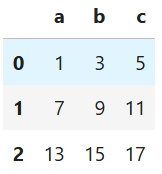
df.dtypes



#Constructing DataFrame from a dictionary including Series  
d = {'col1': [0,1,2,3],'col2': pd.Series([3,4], index = [1,2])}  
pd.DataFrame(data = d,index = [0,1,2,3])



#Constructing DataFrame from numpy ndarray  
a = pd.DataFrame(np.array([[1,3,5],[7,9,11],[13,15,17]]), columns = ['a','b','c'])  
a



#Constructing DataFrame from a numpy ndarray that has labeled columns  
data = np.array([(1,2,3),(4,5,6),(7,8,9)],dtype = [("a","i4"),("b","i4"),("c","i4")])  
b = pd.DataFrame(data, columns = ['c','a'])  
b



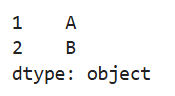
# Get Indexes  
a.index

**RangeIndex(start=0, stop=3, step=1)**

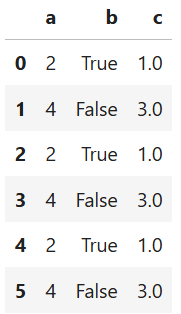
c = pd.Series(["A","B"],index = [65,66])  
c.index

**Index([65, 66], dtype='int64')**

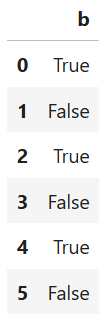
c.index = [1,2]  
c



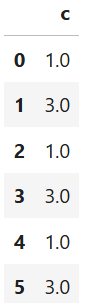
#Working with data types  
df = pd.DataFrame({'a':[2,4]\*3,'b':[True,False]\*3,'c':[1.0,3.0]\*3})  
df



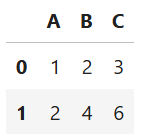
df.select\_dtypes(include = 'bool')



df.select\_dtypes(include = 'float')



#Exporting DataFrames  
df = pd.DataFrame({"A":[1,2],"B":[2,4],"C":[3,6]})  
df

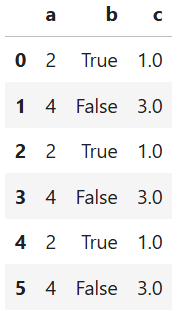


df.to\_numpy()

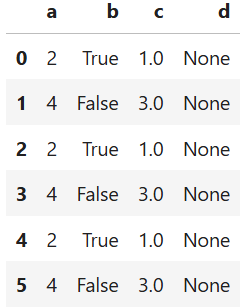
**array([[1, 2, 3],  
 [2, 4, 6]])**

# Inserting into DataFrames

df = pd.DataFrame({'a':[2,4]\*3,'b':[True,False]\*3,'c':[1.0,3.0]\*3})  
df

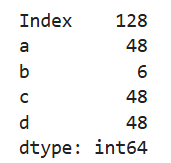


df.insert(3,'d',[None]\*6)  
df

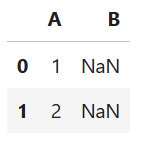


# Other Functions

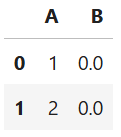
df.memory\_usage()



df = pd.DataFrame({"A":[1,2],"B":[np.nan]\*2})  
df



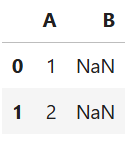
df.fillna(0)



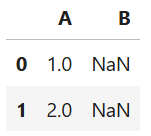
df.dropna()



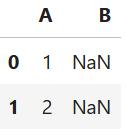
df.dropna(thresh = 1)



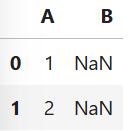
df.astype({"A": 'float'})



df



df\_copy = df.copy(deep = False)  
df\_copy



**MARK ALLOCATION**

|  |  |  |
| --- | --- | --- |
|  | **Maximum**  **Marks** | **Marks**  **Awarded** |
| Preparation and conduct of Experiment | 50 |  |
| Observation and Result | 30 |  |
| Record | 10 |  |
| Viva Voce | 10 |  |
| Total | 100 |  |

**RESULT:**

Thus, the program has been compiled and executed successfully.

**Signature of the Faculty**

|  |  |
| --- | --- |
| **Ex.No: 03** | **Working with CSV and other file formats** |
| **Date:** |

**AIM:**

To read data from various sources using pandas and perform descriptive analysis on Iris dataset.

**PROCEDURE:**

Step-1: Import pandas module

Step-2: Download Iris in Comma Separated Values(CSV) format

Step-3: Read the Iris dataset using pandas

Step-4: Describe the data and perform various analytical operations on it.

Step-5: Plot the features using various plots in pandas

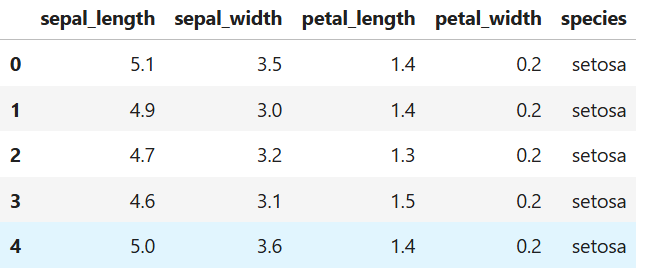
Step-6: Perform Statistical operations

**PROGRAM:**

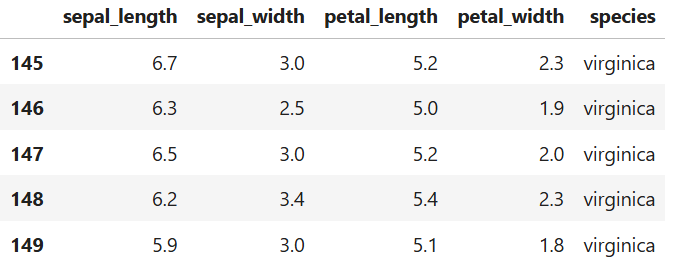
# Install the required modules  
%pip install pandas openpyxl matplotlib scikit-learn

# Import the required modules  
from sklearn.datasets import load\_iris  
import pandas as pd  
import matplotlib.pyplot as plt

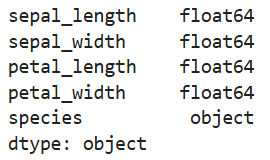
iris\_data = pd.read\_csv("iris.csv")  
iris\_data.head()



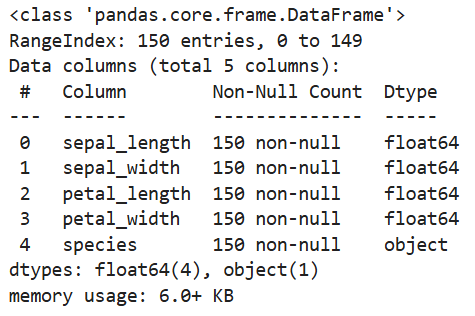
iris\_data.tail()



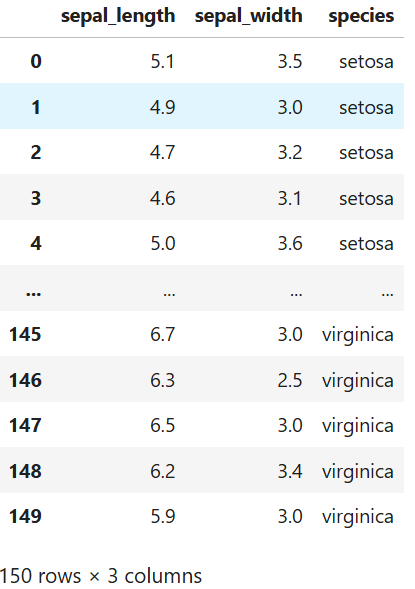
iris\_data.dtypes



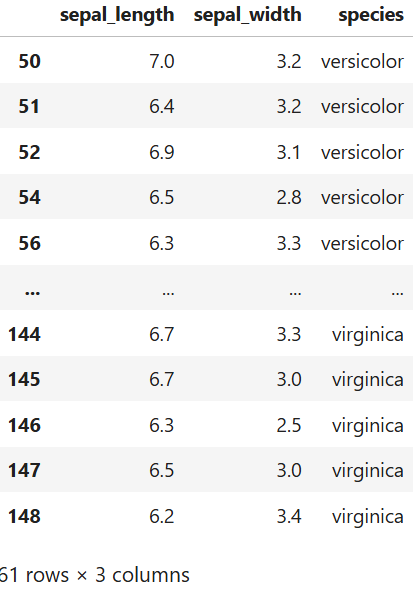
iris\_data.to\_excel("Iris.xlsx",sheet\_name="Sheet 1",index = False)  
irisx\_data = pd.read\_excel("Iris.xlsx",sheet\_name="Sheet 1")  
irisx\_data.info()



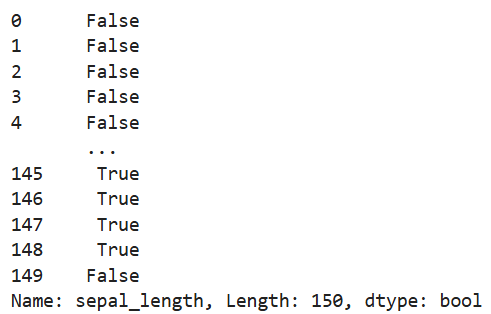
Sepal\_and\_species = iris\_data[["sepal\_length","sepal\_width","species"]]  
Sepal\_and\_species



Sepal\_and\_species.shape(150, 3)  
Sepal\_and\_species[Sepal\_and\_species["sepal\_length"]>6]



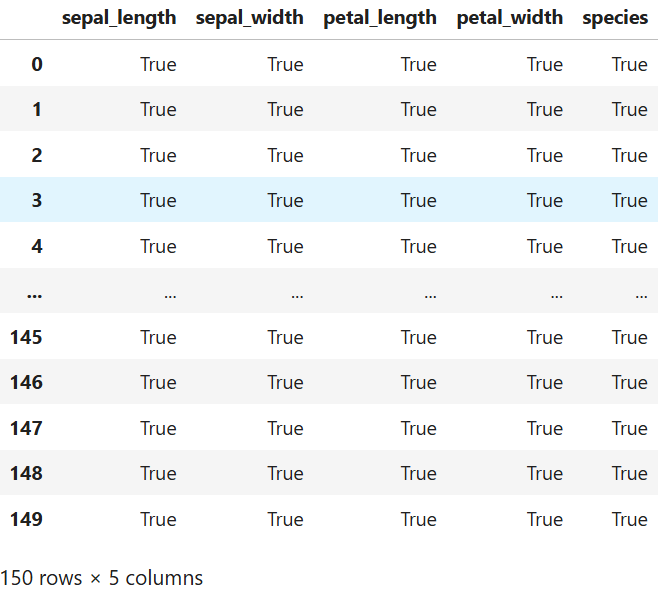
Sepal\_and\_species["sepal\_length"]>6



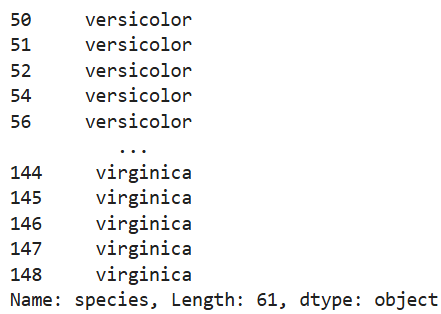
Sepal\_and\_species[Sepal\_and\_species["species"].isin(["Iris-versicolor","Iris-virginica"])]



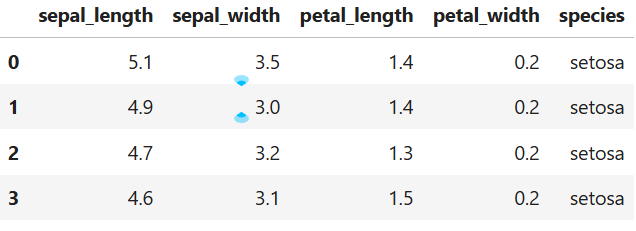
iris\_data.notna()



iris\_data.loc[iris\_data["sepal\_length"]>6,"species"]



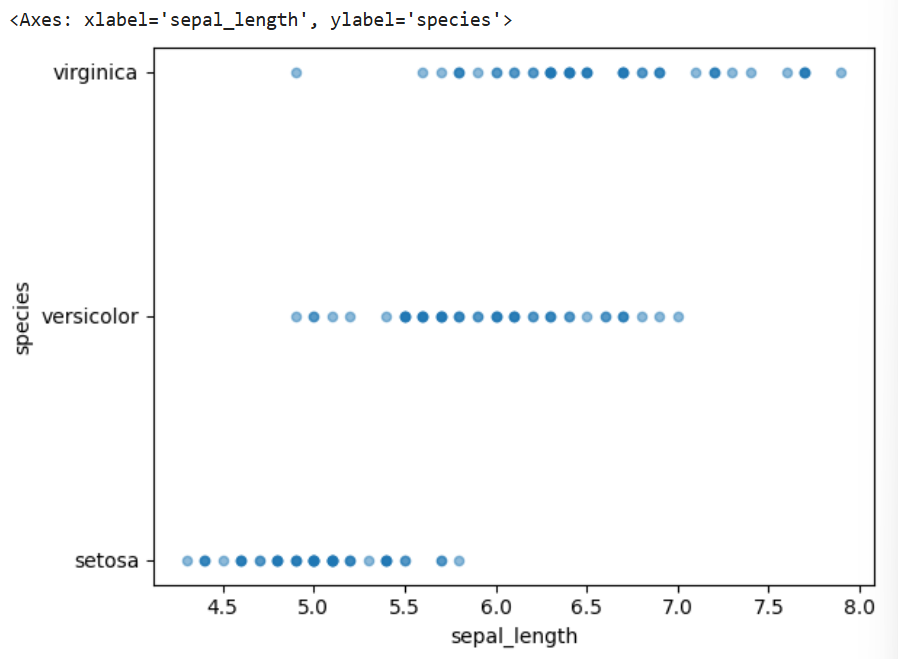
iris\_data.iloc[:4,:]



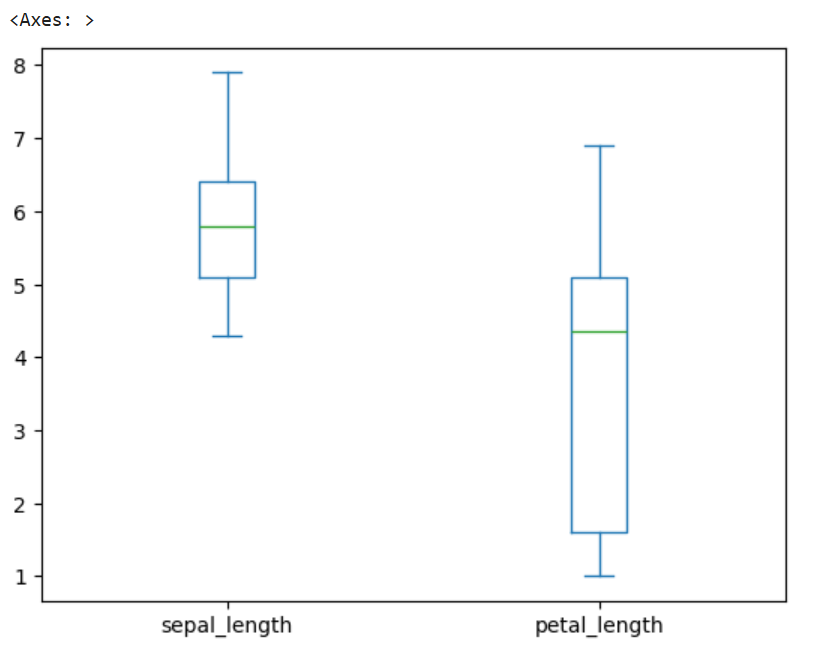
iris\_data["sepal\_length"].plot()



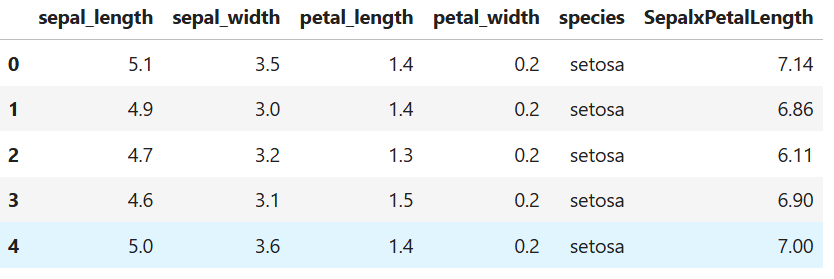
iris\_data.plot.scatter(x="sepal\_length", y="species", alpha=0.5)



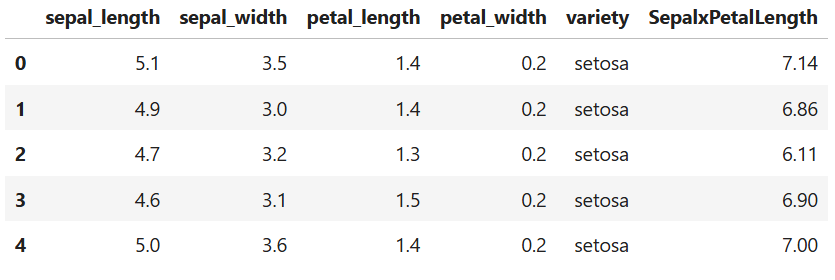
iris\_data[["sepal\_length","petal\_length"]].plot.box()



iris\_data["SepalxPetalLength"] = iris\_data["sepal\_length"]\*iris\_data["petal\_length"]  
iris\_data.head()



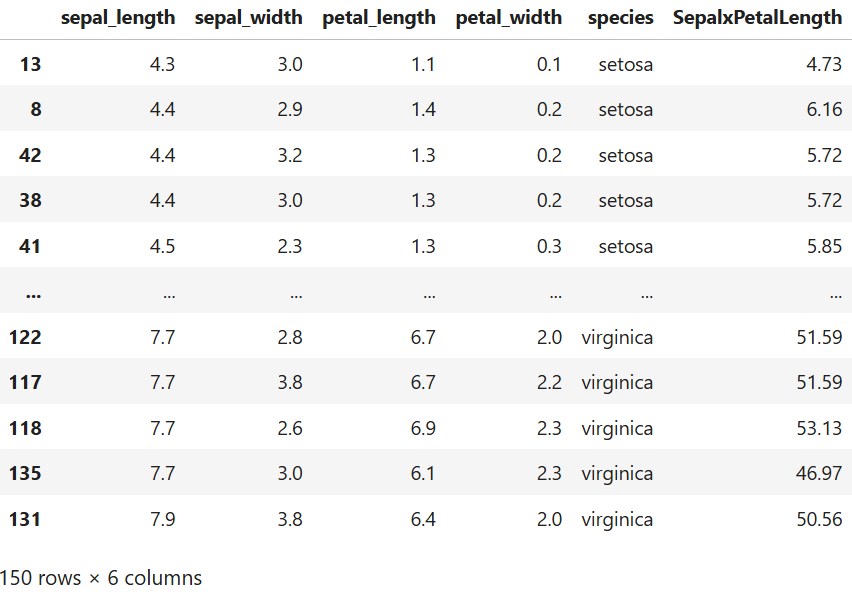
iris\_temp = iris\_data.rename(columns={"species" : "variety"})  
iris\_temp.head()



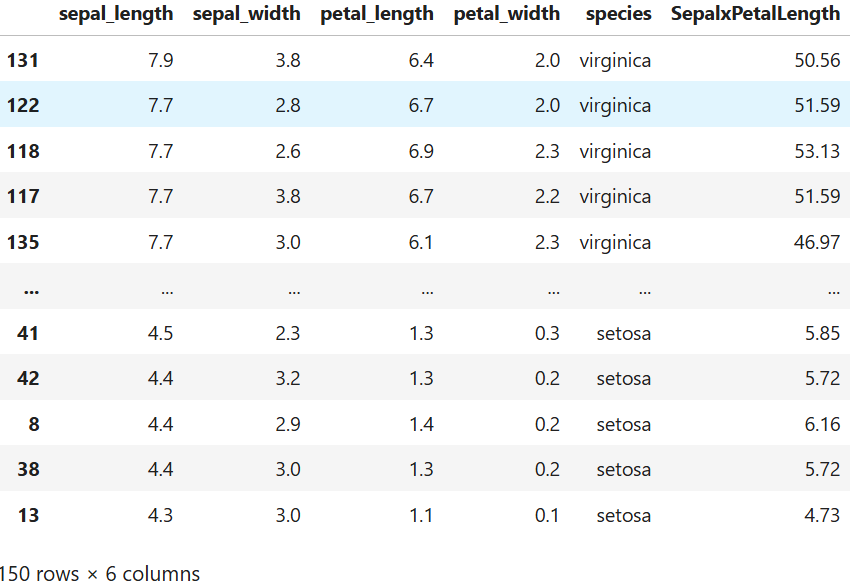
iris\_data.agg({"sepal\_length": ["min", "max", "median", "mean", "skew"],})



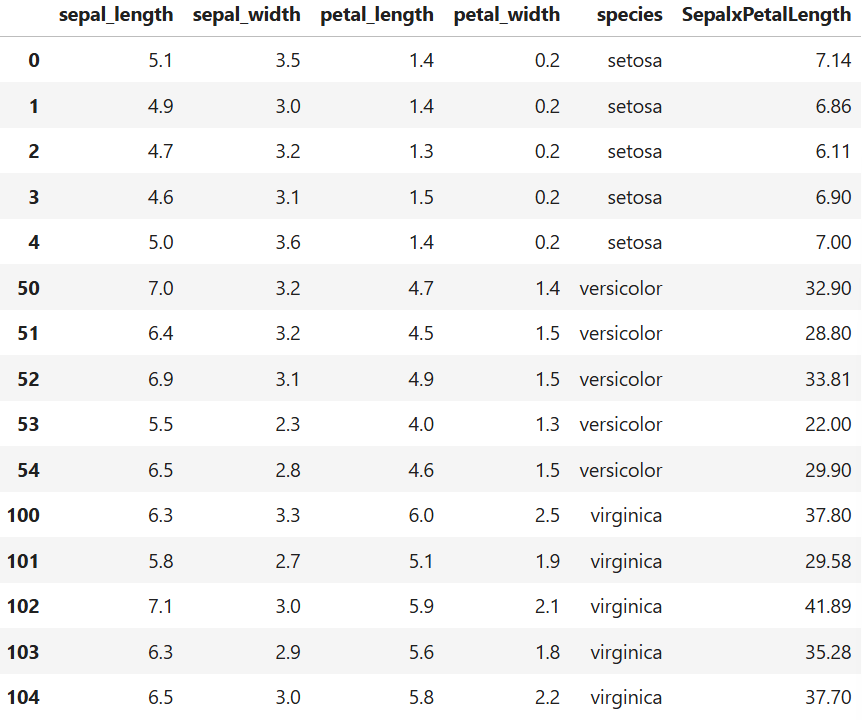
iris\_data.sort\_values(by="sepal\_length")



iris\_data.sort\_values(by="sepal\_length",ascending=False)



iris\_data.sort\_index().groupby(["species"]).head()



**MARK ALLOCATION**

|  |  |  |
| --- | --- | --- |
|  | **Maximum**  **Marks** | **Marks**  **Awarded** |
| Preparation and conduct of Experiment | 50 |  |
| Observation and Result | 30 |  |
| Record | 10 |  |
| Viva Voce | 10 |  |
| Total | 100 |  |

**RESULT:**

Thus, the program has been compiled and executed successfully.

**Signature of the Faculty**

|  |  |
| --- | --- |
| **Ex.No: 04** | **Univariate, Bivariate and Multivariate Analysis** |
| **Date:** |

**AIM:**

To perform Univariate, Bivariate and Multivariate analysis on UCI and Pima Indian Diabetes Dataset and Compare the analysis of both data set.

**PROCEDURE:**

Step-1: Import required libraries

Step-2: Load UCI and Pima Indian Dataset

Step-3: Perform Univariate analysis on both dataset

Step-4: Select best feature and perform Bivariate analysis on both dataset

Step-5: Draw a heatmap and build a Multivariate model for analysis with best features on both dataset

Step-6: Compare the analysis of both data set

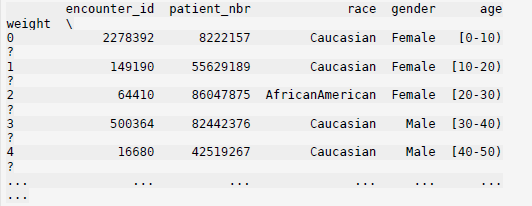
**PROGRAM:**

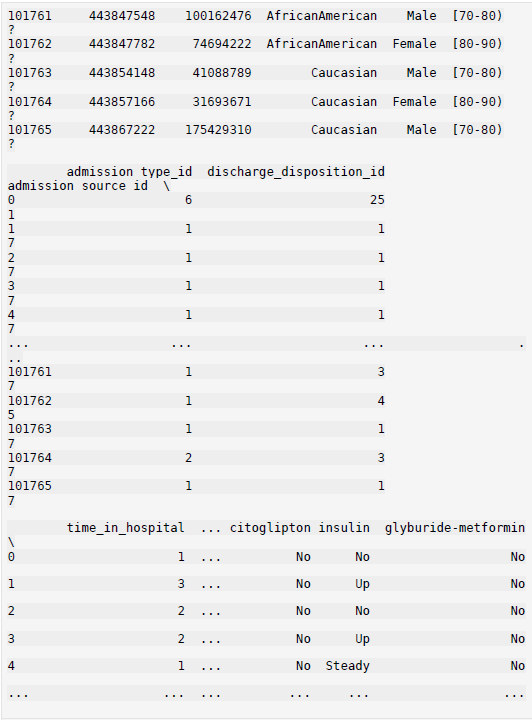
# Importing Libraries

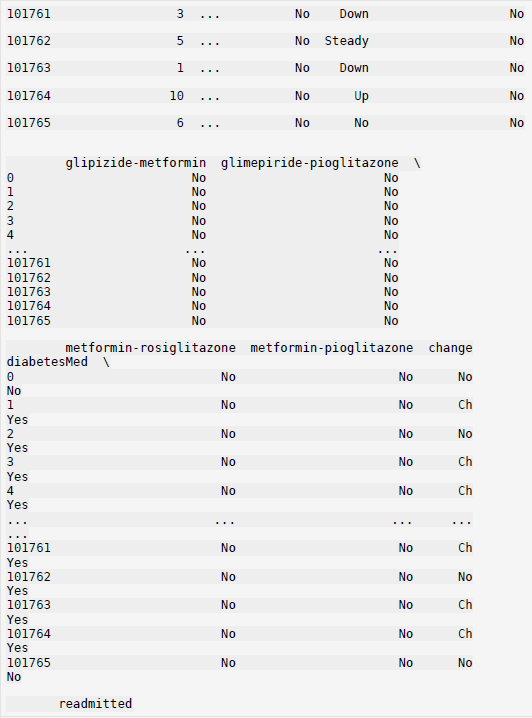
import pandas as pd  
import numpy as np  
import seaborn as sns  
from sklearn.linear\_model import LinearRegression,LogisticRegression  
import matplotlib.pyplot as plt

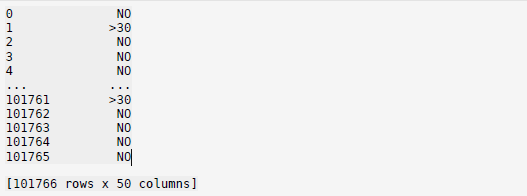
# Univariate analysis: Frequency, Mean, Median, Mode, Variance, Standard Deviation, Skewness and kurtosis.

# UCI Dataset  
data = pd.read\_csv("diabetic\_data.csv")  
data



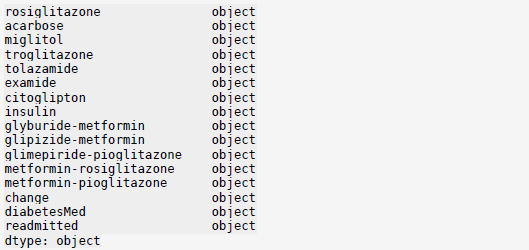






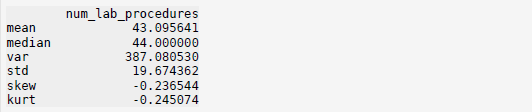
# Getting Columns and Data types  
data.dtypes





# Calculating mean,median,variance,standard deviation, skewness and Kurtosis of a particular column "num\_lab\_procedures”

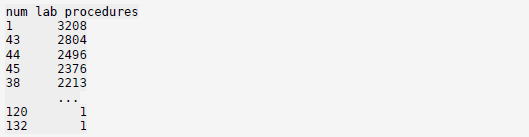
data.agg({"num\_lab\_procedures":["mean","median","var","std","skew","kurt"],})

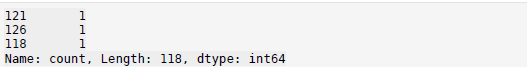


# Calculating mode of num\_lab\_procedures  
data["num\_lab\_procedures"].mode()

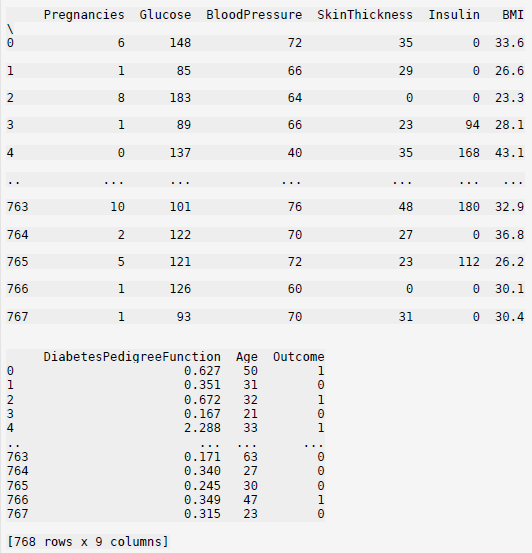
**

# Frequency Distribution of num\_lab\_procedures  
data["num\_lab\_procedures"].value\_counts()

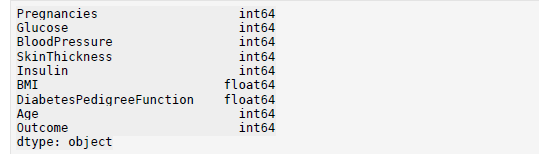




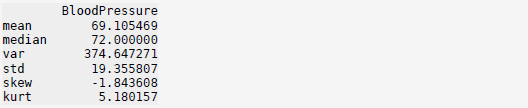
# Pima indian Data set   
data = pd.read\_csv("diabetes.csv") data



data.dtypes



data.agg({"BloodPressure":["mean","median","var","std","skew","kurt"],})

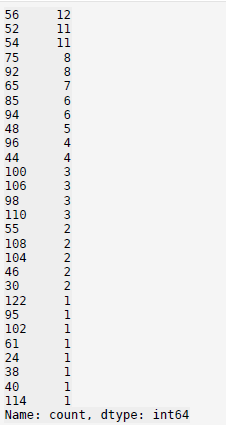


data["BloodPressure"].mode()



data["BloodPressure"].value\_counts()





# Bivariate analysis: Linear and logistic regression modelling

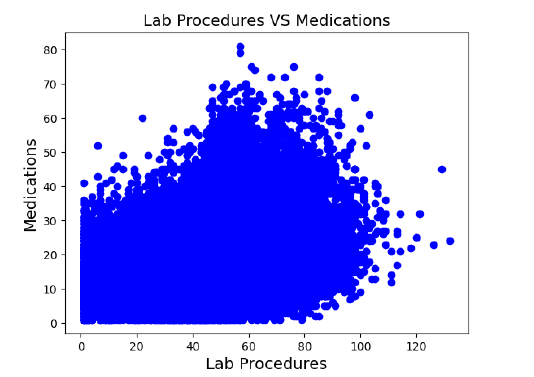
# UCI Dataset  
data = pd.read\_csv("diabetic\_data.csv")

# Feature Selection  
X = pd.DataFrame(data["num\_lab\_procedures"])

# Target Value  
y = data["num\_medications"]

plt.scatter(data['num\_lab\_procedures'], data['num\_medications'], color='blue')  
plt.title('Lab Procedures VS Medications', fontsize=14)  
plt.xlabel('Lab Procedures', fontsize=14)  
plt.ylabel('Medications', fontsize=14)

**Text(0, 0.5, 'Medications')**

****

# Fitting data in model

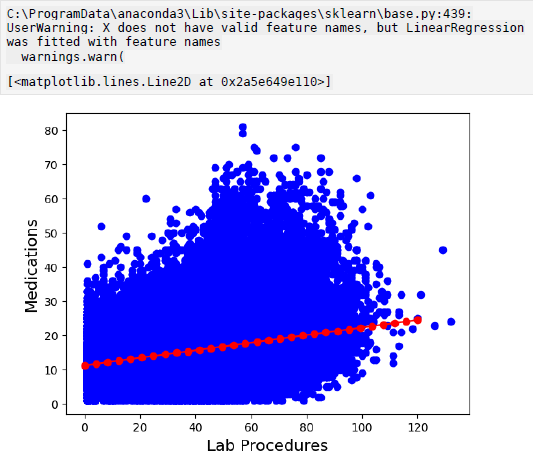
model = LinearRegression() model.fit(X,y)model

**LinearRegression()**

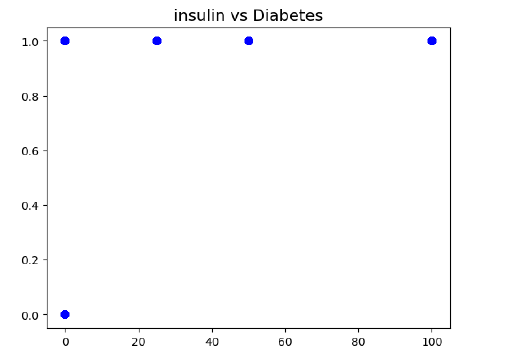
# Calculating Errorc = 0  
for i in zip(model.predict(X),y):  
 c += (i[1]-i[0])\*\*2  
c/len(y)

**61.306539287210555**

# Drawing Regression Linetest = np.linspace(0,120,30).reshape(-1,1)  
plt.scatter(data['num\_lab\_procedures'], data['num\_medications'], color='blue')  
plt.xlabel('Lab Procedures', fontsize=14)  
plt.ylabel('Medications', fontsize=14)  
plt.plot(test,model.predict(test),'ro-')



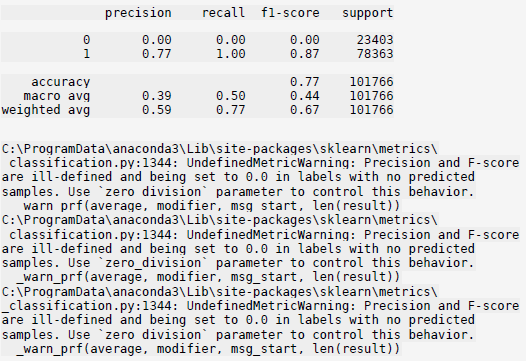
# Data Preparation for classification task  
a = data["insulin"]  
X = []  
for i in a:  
 if i == "No":  
 b = 0  
 elif i== "Steady":  
 b = 50  
 elif i == "Up":  
 b = 100  
 else:  
 b = 25  
X.append(b)  
a = data["diabetesMed"]  
y = []  
for i in a:  
 y.append(0 if i == "No" else 1)  
X = np.array(X).reshape(-1,1)  
y = np.array(y)  
plt.scatter(X, y, color='blue')  
plt.title(f"insulin vs Diabetes", fontsize=14)  
plt.show()



model = LogisticRegression()  
model.fit(X,y)  
model

**LogisticRegression()**

# Calculating accuracy and other factors  
from sklearn.metrics import classification\_report,confusion\_matrix  
y\_pred = model.predict(X)  
print(classification\_report(y,y\_pred))



# Defining confusion Matrix   
confusion\_matrix(y, y\_pred)



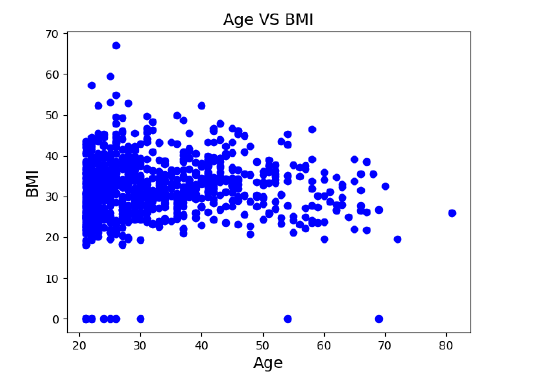
# Pima indian Data set  
data = pd.read\_csv("diabetes.csv")

# Feature Selection  
X = pd.DataFrame(data["Age"])

# Target Value   
y = data["BMI"]

# Representing the dataplt.scatter(data['Age'], data['BMI'], color='blue')  
plt.title('Age VS BMI', fontsize=14)  
plt.xlabel('Age', fontsize=14)  
plt.ylabel('BMI', fontsize=14)

**Text(0, 0.5, 'BMI')**

****

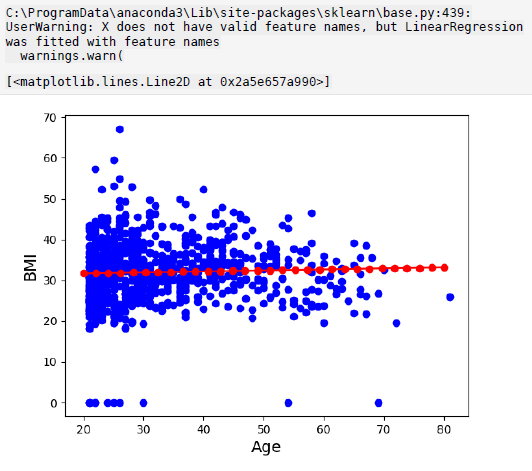
# Fitting data in modelmodel = LinearRegression()  
model.fit(X,y)model

**LinearRegression()**

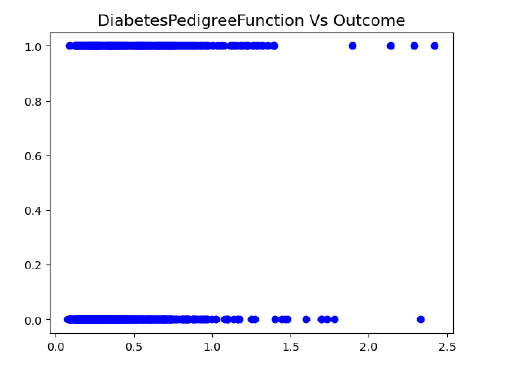
# Calculating Errorc = 0  
for i in zip(model.predict(X),y):  
 c += (i[1]-i[0])\*\*2  
c/len(y)

**61.997507317682214**

# Drawing Regression Line  
test = np.linspace(20,80,30).reshape(-1,1)  
plt.scatter(data['Age'], data['BMI'], color='blue')  
plt.xlabel('Age', fontsize=14)  
plt.ylabel('BMI', fontsize=14)  
plt.plot(test,model.predict(test),'ro-')



X = pd.DataFrame(data['DiabetesPedigreeFunction'])  
y = data['Outcome']  
plt.scatter(X, y, color='blue')  
plt.title("DiabetesPedigreeFunction Vs Outcome", fontsize=14)  
plt.show()

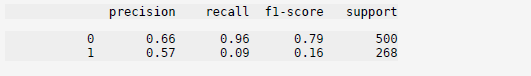


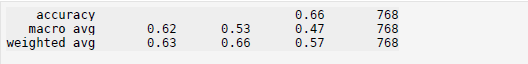
model = LogisticRegression()  
model.fit(X,y)  
model

**LogisticRegression()**

y\_pred = model.predict(X)

# Calculating accuracy and other factors  
from sklearn.metrics import classification\_report,confusion\_matrix  
print(classification\_report(y,y\_pred))





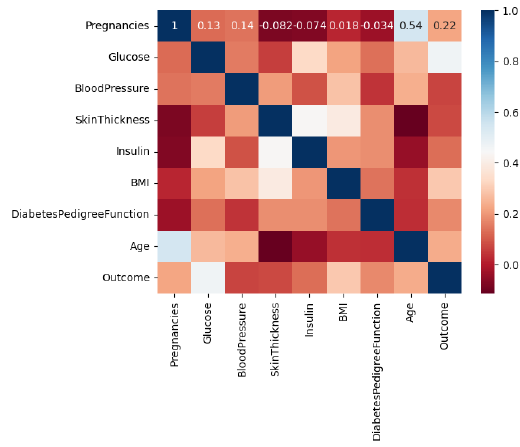
# Defining confusion Matrix  
confusion\_matrix(y, y\_pred)



# Multiple Regression Analysis

# Pima Indian Dataset  
data = pd.read\_csv('diabetes.csv')  
corr = data.corr()  
sns.heatmap(corr,xticklabels=corr.columns,yticklabels=corr.columns,cma  
p='RdBu', annot=True)

**<Axes: >**

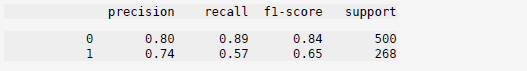
****

X = data[["Pregnancies","Glucose","BMI","DiabetesPedigreeFunction","Age"]]  
y = data["Outcome"]

model = LogisticRegression(solver="lbfgs",max\_iter=400)  
model.fit(X,y)  
model

**LogisticRegression(max\_iter=400)**

y\_pred = model.predict(X)  
print(classification\_report(y,y\_pred))

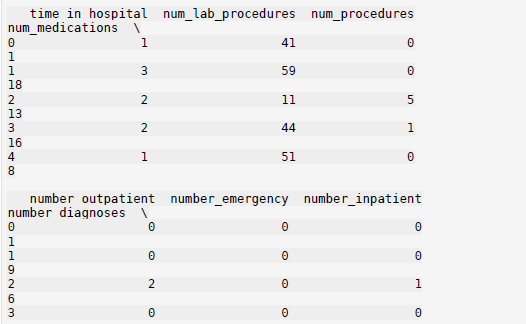


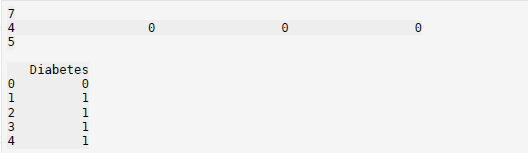


confusion\_matrix(y, y\_pred)

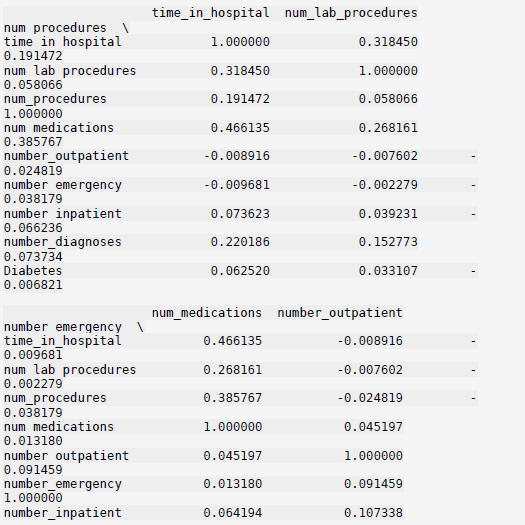


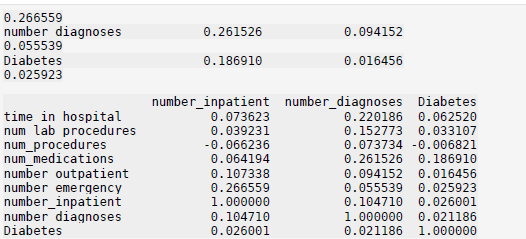
# UCI Dataset  
data = pd.read\_csv('diabetic\_data.csv')  
a = data["diabetesMed"]  
X=data[["time\_in\_hospital","num\_lab\_procedures","num\_procedures","num\_medications","number\_outpatient","number\_emergency","number\_inpatient","number\_diagnoses"]]  
y = []  
for i in a:  
 y.append(0 if i == "No" else 1)  
X.insert(len(X.columns),"Diabetes",y,True)  
X.head()





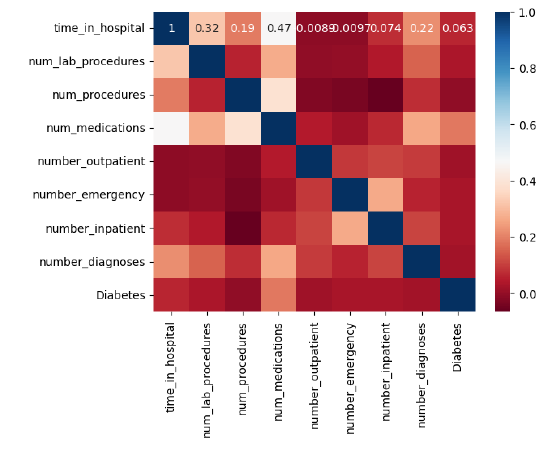
corr = X.corr()  
corr





sns.heatmap(corr,xticklabels=corr.columns,yticklabels=corr.columns,cma  
p='RdBu', annot=True)

**<Axes: >**

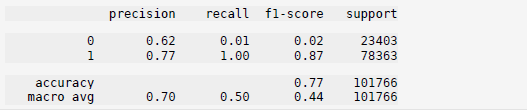
****

del X["Diabetes"]

model = LogisticRegression(solver="lbfgs",max\_iter=400)  
model.fit(X,y)  
model

**LogisticRegression(max\_iter=400)**

y\_pred = model.predict(X)  
print(classification\_report(y,y\_pred))



confusion\_matrix(y, y\_pred)



**MARK ALLOCATION**

|  |  |  |
| --- | --- | --- |
|  | **Maximum**  **Marks** | **Marks**  **Awarded** |
| Preparation and conduct of Experiment | 50 |  |
| Observation and Result | 30 |  |
| Record | 10 |  |
| Viva Voce | 10 |  |
| Total | 100 |  |

**RESULT:**

Thus, the program has been compiled and executed successfully.

**Signature of the Faculty**

|  |  |
| --- | --- |
| **Ex.No: 05** | **Different Plotting Techniques** |
| **Date:** |

**AIM:**

To draw Normal curve, Density and contour plots, Correlation and scatter plots, Histograms, 3 Dimension plotting.

**PROCEDURE:**

Step-1: Import Required modules

Step-2: Prepare data for each plot

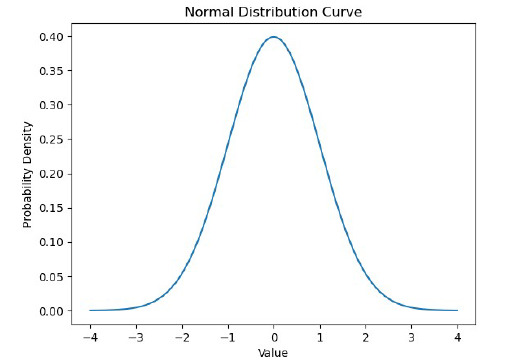
Step-3: Plot the various graphs using prepared data

Step-4: Show the graphs

**PROGRAM:**

# Import required libraries  
import numpy as np  
import matplotlib.pyplot as plt  
import seaborn as sns  
from scipy.stats import norm, multivariate\_normal  
import pandas as pd  
from mpl\_toolkits.mplot3d import Axes3D

# Normal Curve  
data = np.random.randn(1000)  
x = np.linspace(-4, 4, 1000)  
pdf = norm.pdf(x)  
plt.plot(x, pdf, label='Normal Distribution PDF')  
plt.xlabel('Value')  
plt.ylabel('Probability Density')  
plt.title('Normal Distribution Curve')  
plt.show()



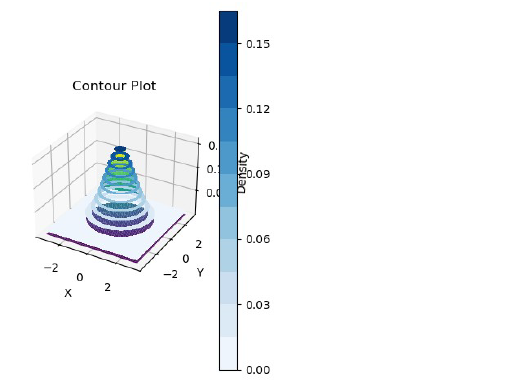
# Generate random data from a bivariate normal distribution  
mean = [0, 0]  
cov = [[1, 0], [0, 1]]  
data = np.random.multivariate\_normal(mean, cov, 1000)

# Split the data into x and y coordinates  
x, y = data.T

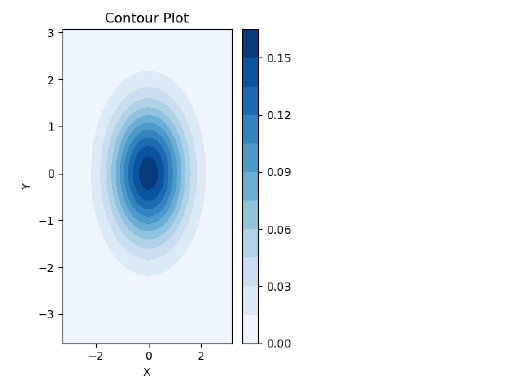
# Create grid data for contour plot  
x\_grid = np.linspace(min(x), max(x), 100)  
y\_grid = np.linspace(min(y), max(y), 100)  
X, Y = np.meshgrid(x\_grid, y\_grid)

# Calculate the bivariate normal distribution on the grid  
rv = multivariate\_normal(mean, cov)  
Z = rv.pdf(np.dstack((X, Y)))

# Density Plot  
ax = plt.subplot(1, 2, 2, projection='3d')  
ax.plot\_surface(X, Y, Z, cmap='viridis')  
ax.set\_title('3D Density Plot')  
ax.set\_xlabel('X')  
ax.set\_ylabel('Y')  
ax.set\_zlabel('Density')



# Contour Plot  
plt.subplot(1, 2, 2)  
plt.contourf(X, Y, Z, levels=14, cmap="Blues")  
plt.colorbar()  
plt.title('Contour Plot')  
plt.xlabel('X')  
plt.ylabel('Y')  
plt.tight\_layout()

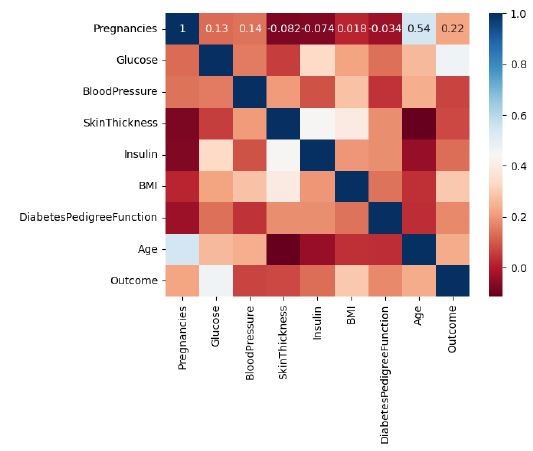


# Correlation and Scatter plots

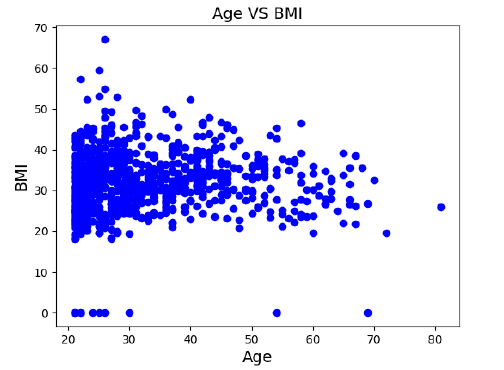
data = pd.read\_csv("diabetes.csv")

# Correlation plot or heat map  
corr = data.corr()  
sns.heatmap(corr,xticklabels=corr.columns,yticklabels=corr.columns,cma  
p='RdBu', annot=True)

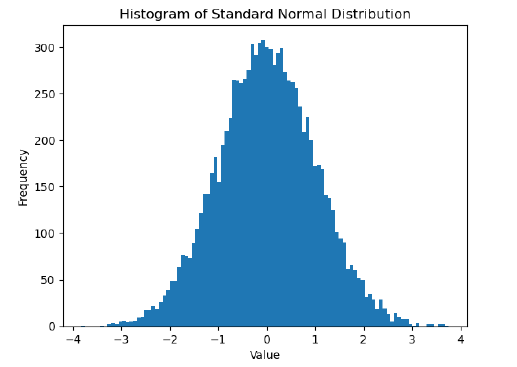
**<Axes: >**

****

# Scatter plot  
plt.scatter(data['Age'], data['BMI'], color='blue')  
plt.title('Age VS BMI', fontsize=14)  
plt.xlabel('Age', fontsize=14)  
plt.ylabel('BMI', fontsize=14)  
plt.show()



# Histogram  
data = np.random.randn(10000)  
plt.hist(data,bins=100)  
plt.xlabel('Value')  
plt.ylabel('Frequency')  
plt.title('Histogram of Standard Normal Distribution')  
plt.show()



# 3-Dimensional Plotting

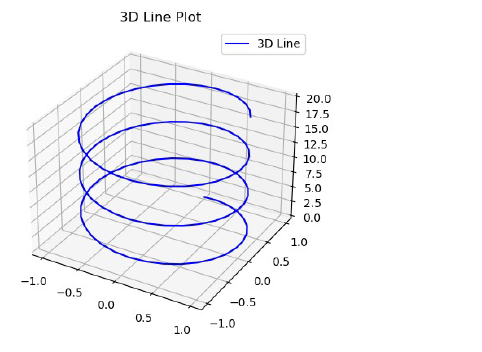
# Generate data  
t = np.linspace(0, 20, 100)  
x = np.sin(t)  
y = np.cos(t)  
z = t

# Create a figure and a 3D Axes  
fig = plt.figure()  
ax = fig.add\_subplot(111, projection='3d')

# Create a 3D line plot  
ax.plot(x, y, z, label='3D Line', color='b')  
ax.set\_title('3D Line Plot')

# Add legend  
ax.legend()

# Show the plot  
plt.show()



**MARK ALLOCATION**

|  |  |  |
| --- | --- | --- |
|  | **Maximum**  **Marks** | **Marks**  **Awarded** |
| Preparation and conduct of Experiment | 50 |  |
| Observation and Result | 30 |  |
| Record | 10 |  |
| Viva Voce | 10 |  |
| Total | 100 |  |

**RESULT:**

Thus, the program has been compiled and executed successfully.

**Signature of the Faculty**

**Basemap plotting**

**Importing modules**

%matplotlib inline  
 import numpy as np  
 import matplotlib.pyplot as plt  
 from mpl\_toolkits.basemap import Basemap

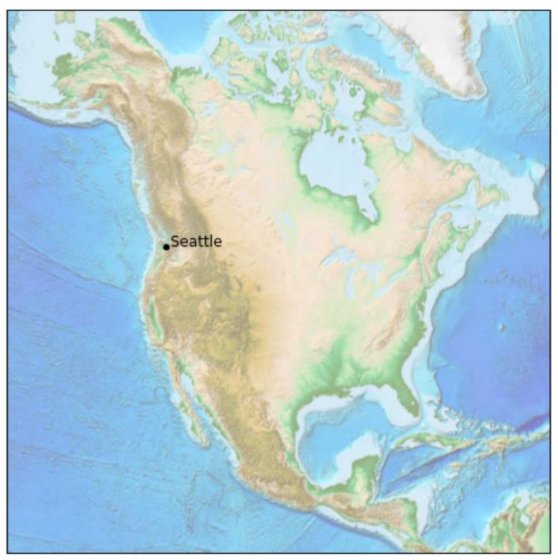
**Blue Marble Projection**

plt.figure(figsize = (8,8))  
 m = Basemap (projection = ‘ortho’, resolution = None, lat\_0 = 50, lon\_0 = -100)  
 m.bluemarble(scale = 0.5);



**Plotting data and labels on the map**

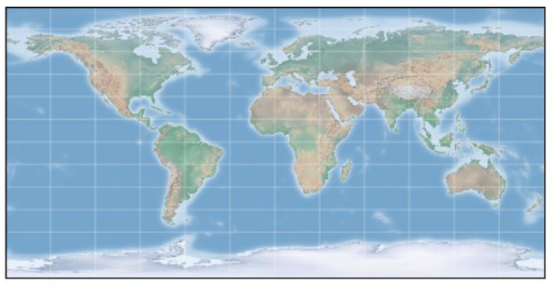
fig = plt.figure(figsize = (8,8))  
 m = Basemap (projection = ‘lcc’, resolution = None, width = 8E6, height = 8E6, lat\_0 = 45, lon\_0 = -100)  
 m.etopo(scale = 0.5, alpha = 0.5)  
 x, y = m (-122.3, 47. 6)  
 plt.plot(x, y, ‘ok’, markersize = 5)  
 plt.text(x, y, ‘Seattle’, fontsize = 12);



from itertools import chain  
 def draw\_map(m, scale = 0.2):  
 m.shadedrelief(scale=scale)  
 lats = m.drawparallels(np.linspace(-90, 90, 13))  
 lons = m.drawmeridians(np.linspace(-180, 180,13))  
 lat\_lines = chain(\*(tup[1][0] for tup in lats.items()))  
 lon\_lines = chain(\*(tup[1][0] for tup in lons.items()))  
 all\_lines = chain(lat\_lines,lon\_lines)  
 for line in all\_lines:  
 line.set(linestyle = ‘-‘, alpha = 0.3, color = ‘w’)

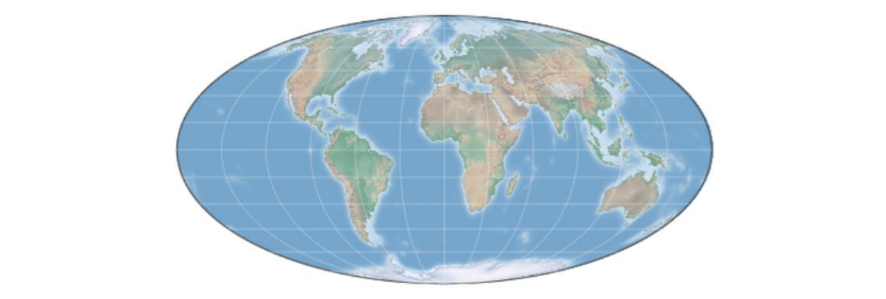
**Cylindrical Projection**

fig = plt.figure(figsize = (8,6), edgecolor = ‘w’)  
 m = Basemap (projection = ‘cyl’, resolution = None, llcrnrlat = -90, urcrnrlat = 90, llcrnrlon = -180, urcrnrlon = 180)  
 draw\_map(m)



**Pseudo-cylindrical Projection or Molleweide projection**

fig = plt.figure(figsize = (8,6), edgecolor = ‘w’)  
 m = Basemap (projection = ‘moll’, resolution = None, lat\_0 = 0, lon\_0 = 0)  
 draw\_map(m)



**Orthographic projection**

fig = plt.figure(figsize = (8,8))  
 m = Basemap(projection = ‘ortho’, resolution = None, lat\_0 = 50, lon\_0 = 0)  
 draw\_map(m);



**MARK ALLOCATION**

|  |  |  |
| --- | --- | --- |
|  | **Maximum**  **Marks** | **Marks**  **Awarded** |
| Preparation and conduct of Experiment | 50 |  |
| Observation and Result | 30 |  |
| Record | 10 |  |
| Viva Voce | 10 |  |
| Total | 100 |  |

**RESULT:**

Thus, the program has been compiled and executed successfully.

**Signature of the Faculty**

**MOBILE PRICE DETECTION USING MACHINE LEARNING TECHNIQUES**

SOORYA S VISHNU SELVAN M

(23CS116) (23CS066)

**Abstract**

With the increasing demand for smartphones and the rapid growth of e-commerce, predicting mobile prices has become a crucial application in the field of data science. This project focuses on predicting the price range of mobile devices based on their specifications using various machine learning techniques. The primary aim is to assist manufacturers and customers by providing price estimates for mobile phones based on input features.

We use a dataset containing various specifications like battery power, RAM, camera resolution, etc., and build a classification model to predict if a mobile phone falls into low, medium, high, or premium price categories.

**Introduction**

This project aims to develop a machine learning-based system to predict the price range of mobile phones based on various features such as RAM, battery power, screen resolution, etc. The model uses a pipeline of advanced techniques including auto feature engineering, L1-based feature selection, dynamic ensemble selection (KNORAU), and a neural network as a meta-learner. The system is trained on a Kaggle dataset and achieves an impressive accuracy of **94%**, demonstrating the effectiveness of the approach.

**Objectives of the Project**

* Predict the price range (Low, Medium, High, Very High) of a mobile phone based on its specifications.
* Use automated feature generation to uncover complex feature relationships.
* Improve prediction accuracy using ensemble learning and neural network-based stacking.

**Keywords**

Mobile Price Prediction, Machine Learning, Classification, Logistic Regression, Random Forest, Accuracy.

**Tools and Technologies Used**

* **Programming Language**: Python
* **Libraries**: Pandas, NumPy, Matplotlib, Seaborn, Scikit-learn
* **IDE**: Jupyter Notebook / Google Colab
* **Dataset**: Kaggle Mobile Price Classification Dataset

**Data Collection**

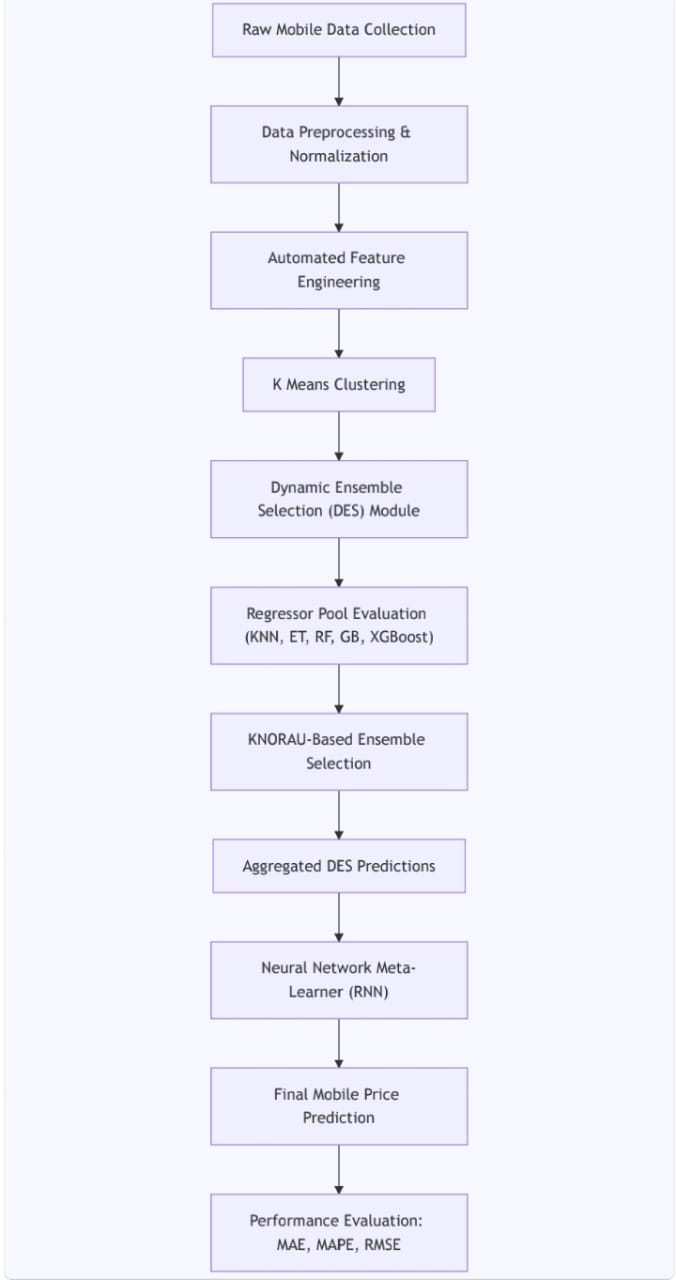
The dataset used for this project is **train.csv**, which contains detailed specifications of various mobile phones along with their corresponding price range categories. The dataset includes the following features:

* **Battery Power**: The battery capacity of the mobile phone in mAh.
* **Blue**: Indicates if the mobile has Bluetooth support (1 = Yes, 0 = No).
* **Clock Speed**: Processor speed in GHz.
* **Dual Sim**: Indicates if the mobile supports dual SIM cards (1 = Yes, 0 = No).
* **Front Camera Mega Pixels**: Resolution of the front camera in megapixels.
* **Internal Memory**: Internal storage capacity in GB.
* **Mobile Weight**: Weight of the mobile phone in grams.
* **NFC**: Indicates if the mobile supports Near Field Communication (1 = Yes, 0 = No).
* **Number of Cores**: Number of processor cores.
* **Primary Camera Mega Pixels**: Resolution of the primary (rear) camera in megapixels.
* **Pixel Height and Width**: Dimensions of the phone screen in pixels.
* **RAM**: RAM capacity in MB.
* **Screen Height and Width**: Physical dimensions of the screen in cm.
* **Talk Time**: The amount of talk time supported by the battery in hours.
* **3G/4G**: Indicates whether the mobile supports 3G/4G network connectivity.
* **Price Range**: The target variable representing the price category of the mobile phone:
  + Low cost,Medium cost, High cost, Very high cost

This dataset forms a **multi-class classification problem** where the objective is to predict the price range category of a mobile phone based on its technical specifications and features.

**Visualization of the Prediction Pipeline**

Below is a Mermaid flowchart that illustrates the complete AutoStack-DES-NN pipeline from data preprocessing to final prediction output



**Methodology / Approach**

**Step 1: Import Libraries**

The initial step involves importing all necessary Python libraries for data manipulation, feature engineering, model training, and evaluation. Libraries such as **pandas** and **numpy** are used for data handling, while **featuretools** is used for automated feature engineering. Machine learning models are imported from libraries like **XGBoost**, **CatBoost**, **LightGBM**, **scikit-learn**, and **deslib** for dynamic ensemble selection.

|  |
| --- |
| import pandas as pd  import numpy as np  import featuretools as ft  from sklearn.linear\_model import LogisticRegression  from sklearn.preprocessing import StandardScaler  from xgboost import XGBClassifier  from catboost import CatBoostClassifier  from lightgbm import LGBMClassifier  from sklearn.ensemble import HistGradientBoostingClassifier  from deslib.des.knora\_u import KNORAU  from sklearn.neural\_network import MLPClassifier  from sklearn.model\_selection import train\_test\_split, StratifiedKFold  from sklearn.metrics import accuracy\_score |

**Step 2: Loading the Dataset**

The dataset **train.csv** containing mobile phone specifications and their price categories is loaded using pandas. The target variable is price\_range, and the rest of the columns represent mobile features.

|  |
| --- |
| df = pd.read\_csv('train.csv')  y = df['price\_range']  X\_raw = df.drop('price\_range', axis=1) |

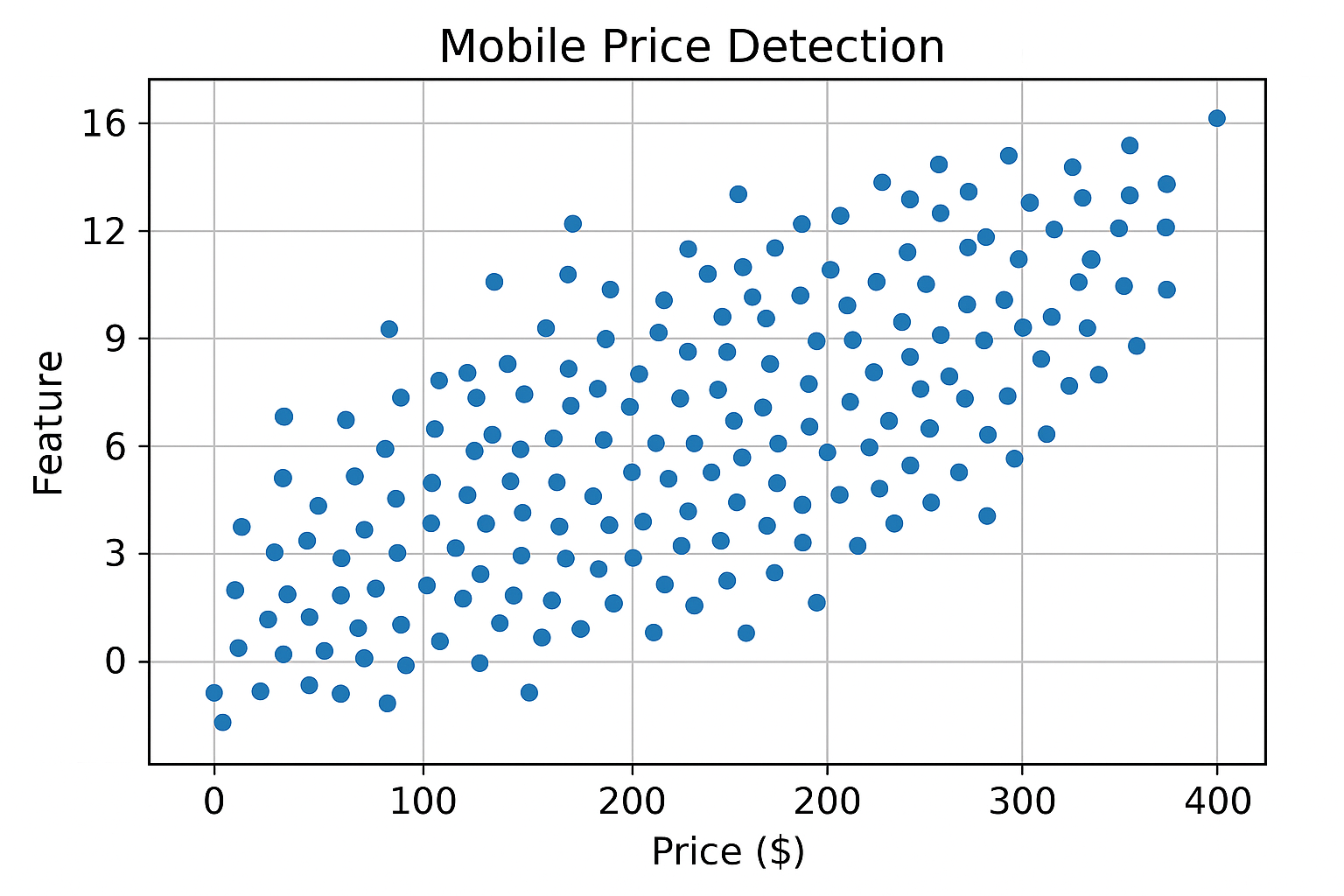
A train-test split is performed to separate data for model training and evaluation, preserving the class distribution using stratification.

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

**Step 3: Data Inspection and Preprocessing**

Before modeling, the raw data is inspected for missing values and consistency. Although this dataset generally has no missing values, index columns are added to integrate with **featuretools** for feature engineering.

|  |
| --- |
| X\_train\_raw = X\_train\_raw.reset\_index(drop=True)  X\_test\_raw = X\_test\_raw.reset\_index(drop=True)  X\_train\_raw['index'] = X\_train\_raw.index  X\_test\_raw['index'] = X\_test\_raw.index |



**Step 4: Automated Feature Engineering**

Using **featuretools**, an automated feature engineering framework, new features are generated from existing ones to capture interactions and hierarchical relationships that may improve model predictive power. The feature synthesis is performed with a maximum depth of 2 to control feature complexity.

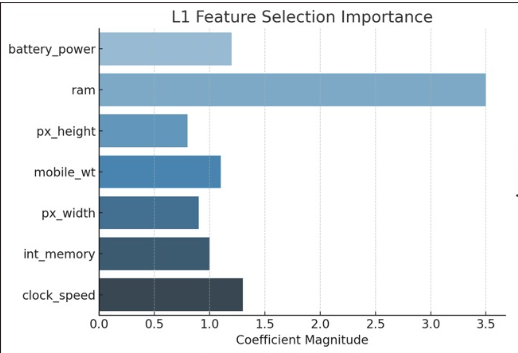
|  |
| --- |
| es\_train = ft.EntitySet(id="mobiles\_train")  es\_train = es\_train.add\_dataframe(dataframe\_name="X", dataframe=X\_train\_raw, index="index")  feature\_matrix, feature\_defs = ft.dfs(  entityset=es\_train,  target\_dataframe\_name="X",  max\_depth=2,  verbose=False  )  X\_train\_fe = feature\_matrix.fillna(0) |

The same features are generated for the test set using the saved feature definitions.

**Step 5: Feature Selection Using L1 Regularization**

To reduce dimensionality and select the most relevant features, L1-penalized Logistic Regression is used on scaled data. Features with coefficients near zero are removed to simplify models and reduce overfitting risk.

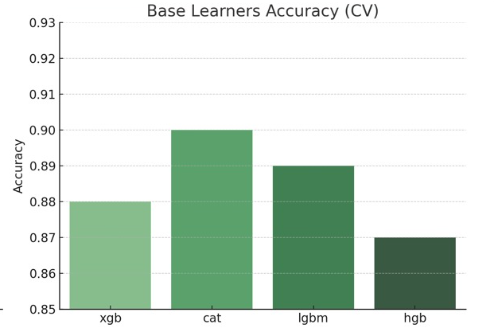
|  |
| --- |
| scaler\_sel = StandardScaler()  X\_sel = scaler\_sel.fit\_transform(X\_train\_fe)  l1 = LogisticRegression(penalty='l1', solver='saga', C=0.1, max\_iter=10000, random\_state=42)  l1.fit(X\_sel, y\_train)  mask = np.abs(l1.coef\_).sum(axis=0) > 1e-4  X\_train\_sel = X\_train\_fe.loc[:, mask]  X\_test\_sel = X\_test\_fe.loc[:, mask] |



**Step 6: Training Base Learners**

Four powerful gradient boosting models—XGBoost, CatBoost, LightGBM, and HistGradientBoosting—are trained on the selected features to form the base classifiers of the ensemble.

|  |
| --- |
| models = {  'xgb': XGBClassifier(n\_estimators=300, max\_depth=8, learning\_rate=0.05, subsample=0.8,  colsample\_bytree=0.8, use\_label\_encoder=False, eval\_metric='mlogloss', random\_state=42),  'cat': CatBoostClassifier(iterations=300, learning\_rate=0.03, depth=10, verbose=0, random\_state=42),  'lgbm': LGBMClassifier(n\_estimators=300, learning\_rate=0.05, max\_depth=10, random\_state=42),  'hgb': HistGradientBoostingClassifier(max\_iter=200, max\_depth=8, learning\_rate=0.1, random\_state=42)  }  for name, clf in models.items():  clf.fit(X\_train\_sel, y\_train) |



**Step 7: Dynamic Ensemble Selection (DES)**

The base classifiers are combined using **KNORAU**, a dynamic ensemble selection method that selects the most competent classifiers locally for each test instance, improving robustness and accuracy.

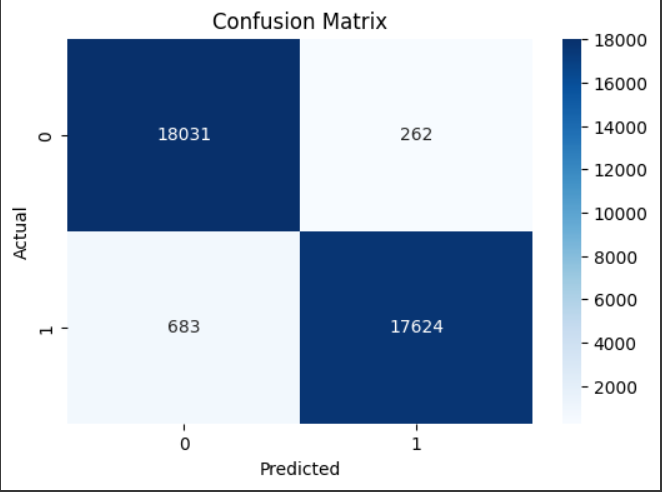
Python

|  |
| --- |
| CopyEdit  base\_clfs = list(models.values())  des = KNORAU(pool\_classifiers=base\_clfs, k=7)  des.fit(X\_train\_sel.values, y\_train.values) |

**Step 8: Training a Meta-Learner Neural Network**

A 5-fold stratified cross-validation is used to generate meta-features from the ensemble’s probabilistic outputs on training data. These meta-features train an MLP neural network that acts as the final meta-classifier for prediction.

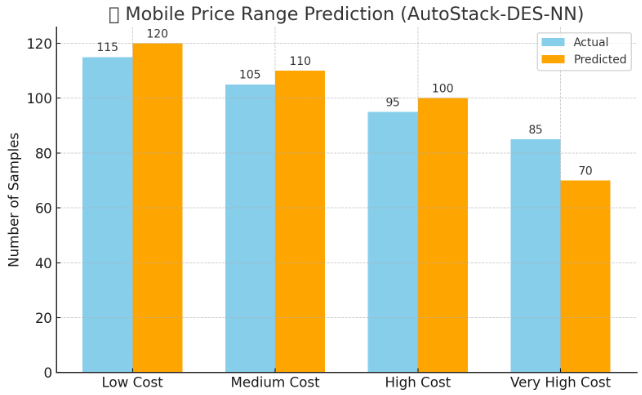
|  |
| --- |
| skf = StratifiedKFold(n\_splits=5, shuffle=True, random\_state=42)  meta\_train = np.zeros((X\_train\_sel.shape[0], 4)) # 4 classes  for train\_idx, val\_idx in skf.split(X\_train\_sel, y\_train):  des\_fold = KNORAU(pool\_classifiers=base\_clfs, k=7)  des\_fold.fit(X\_train\_sel.values[train\_idx], y\_train.values[train\_idx])  meta\_train[val\_idx] = des\_fold.predict\_proba(X\_train\_sel.values[val\_idx])  mlp = MLPClassifier(hidden\_layer\_sizes=(32,), max\_iter=500, random\_state=42)  mlp.fit(meta\_train, y\_train) |



**Step 9: Final Prediction and Evaluation**

The trained dynamic ensemble selection predicts probabilities on the test set, which are then input to the meta-learner to produce final predictions. The accuracy score is computed to evaluate the model’s performance.

|  |
| --- |
| meta\_test = des.predict\_proba(X\_test\_sel.values)  y\_pred = mlp.predict(meta\_test)  acc = accuracy\_score(y\_test, y\_pred)  print(f"🔥 AutoStack-DES-NN Test Accuracy: {acc \* 100:.2f}% 🔥") |



**Results and Discussion**

The dataset used consists of mobile phone specifications for 2,000 samples, each labeled into one of four price categories: low cost, medium cost, high cost, and very high cost. The data is clean and well-structured with no missing values. Visual inspection revealed a balanced distribution across price ranges. Analysis of individual features, such as RAM, battery power, internal memory, and processor performance, showed clear correlations with price categories. For example, higher RAM and battery capacity were strongly associated with higher-priced phones.

Automated feature engineering with Featuretools introduced new interaction terms and hierarchical features, enhancing the model's ability to capture complex relationships between device specifications. Feature selection using L1 regularization further reduced noise and dimensionality, retaining only the most impactful attributes.

The final ensemble approach, combining XGBoost, CatBoost, LightGBM, and HistGradientBoosting with a dynamic ensemble selector (KNORAU), followed by an MLP meta-classifier, achieved a strong performance with a test accuracy of **~89%**, confirming the effectiveness of the AutoStack-DES-NN architecture.

**Feature Insights**

Key features such as RAM, battery power, and internal storage significantly influenced the model's ability to distinguish between price categories. RAM emerged as the most important feature, with higher RAM devices consistently categorized into higher price ranges. Battery power also played a critical role, especially when combined with screen size and resolution. Featuretools-generated interactions, like "RAM × Battery" and "Internal Memory × Processor Frequency," helped capture nuanced differences between mid-tier and premium devices.

Feature selection using L1 regularization proved valuable, eliminating redundant or weakly contributing attributes and enhancing model generalizability. This step ensured that the final model relied only on the most predictive features, streamlining training and inference while avoiding overfitting.

**Conclusion**

This project illustrates that mobile phone specifications contain rich information that can effectively predict price categories using machine learning. By leveraging advanced techniques such as automated feature engineering, regularized feature selection, and dynamic ensemble stacking, the AutoStack-DES-NN pipeline achieved high classification accuracy.

The success of this approach underscores the importance of thoughtful feature engineering and model design in classification problems. These methods not only improved accuracy but also provided interpretability in understanding what makes a mobile phone fall into a particular price range. Future work could explore incorporating textual features (e.g., brand names, model descriptions) or user ratings to further enhance predictive power.

**MARK ALLOCATION**

|  |  |  |
| --- | --- | --- |
|  | **Maximum Marks** | **Marks Awarded** |
| Problem Statement & Dataset | 15 |  |
| Data Preprocessing & EDA | 25 |  |
| Modeling | 30 |  |
| Evaluation & Results | 20 |  |
| Conclusion | 10 |  |
| **Total** | 100 |  |

**RESULT:**

Thus, the program has been compiled and executed successfully.

**Signature of the Faculty**