ML LAB PROGRAMS

Task 1: Write a python program to import and export data using Pandas library functions.
Task 2: Demonstrate various data pre-processing techniques for a given dataset.
Task 3: Implement Dimensionality reduction using Principle Component Analysis (PCA) method.
Task 4: Write a Python program to demonstrate various Data Visualization Techniques.
Task 5: Implement Simple and Multiple Linear Regression Models.
Task 6: Develop Logistic Regression Model for a given dataset.
Task 7: Develop Decision Tree Classification model for a given dataset and use it to classify a new sample.
Task 8: Implement Naïve Bayes Classification in Python
Task 9: Build KNN Classification model for a given dataset.
Task 10: Build Artificial Neural Network model with back propagation on a given dataset.
Task 11
a) Implement Random forest ensemble method on a given dataset.
b) Implement Boosting ensemble method on a given dataset.
Task 12 : Write a python program to implement K-Means clustering Algorithm.

Task 1: Write a python program to import and export data using Pandas library functions.

Go to Google Page and find www.kaggle.com .Select Datasets and find Titanic dataset , then download train.csv file ans save it to desktop.

1. Read a CSV file

```
import pandas as pd
url='C:/Users/MRCET1/Desktop/train.csv'
dataframe=pd.read_csv(url)
dataframe.head(5)
```

2. Write a CSV file

```
import pandas as pd
marks_data=pd. DataFrame({'ID':{0:23,1:43,2:12,3:13,4:67,5:89},
'NAME':{0:'Ram',1:'Deep',2:'Yash',3:'Arjun',4:'Aditya',5:'Divya'},
'Marks':{0:89,1:92,2:45,3:78,4:56,5:76},
'Grade':{0:'b',1:'a',2:'f',3:' c',4:'e',5:'c'}})
filename='C:/Users/MRCET1/Desktop/Marksdata.xlsx'
marks_data.to_excel(filename)
print('Data frame written to Excel')
```

3. Read an Excel File

```
import pandas as pd
url='C:/Users/MRCET1/Desktop/train.csv.xls'
dataframe=pd.read_excel(url)
dataframe.head(5)
```

4.Write an Excel file

```
import pandas as pd
marks_data=pd.DataFrame({'ID':{0:23,1:43,2:12,3:13,4:67,5:89},'NAME':{0:'Ram',1:'Deep',2:'
Yash',3:'Arjun',4:'Aditya',5:'Divya'},'Marks':{0:89,1:92,2:45,3:78,4:56,5:76},'Grade':{0:'b',1:'a',2
:'f', 3:'c',4:'e',5:'c'}})
filename='C:/Users/MRCET1/Desktop/Marksdata.csv'
marks_data.to_csv(filename)
print('Data frame written to CSV');
```

5.Student Marks sheet

```
import pandas as pd
import numpy as np
marks  = { "English" :[67,89,90,55],
"Maths":[55,67,45,56],
"IP":[66,78,89,90],
"Chemistry" :[45,56,67,65],
"Biology":[54,65,76,87]}
result = pd.DataFrame(marks,index=["Athang","Sujata","Sushil","Sumedh"])
print("OUTPUT")
print("*****************Marksheet****************")
print(result)
result.to_csv("result.csv")
df = pd.read_csv("result.csv")
print(df)
```

```
OUTPUT
*****************Marksheet****************

          English  Maths  IP  Chemistry  Biology
Athang     67     55      66      45         54
Sujata     89     67      78      56         65
Sushil     90     45      89      67         76
Sumedh     55     56      90      65         87

Unnamed: 0  English  Maths  IP  Chemistry  Biology
0   Athang      67          55   66       45         54
1   Sujata      89          67   78       56         65
2   Sushil      90          45    89       67         76
3   Sumedh      55          56    90       65         87
```

Task 2: Demonstrate various data pre-processing techniques for a given dataset.

Program:

a. Rescaling Data

For data with attributes of varying scales, we can rescale attributes to possess the same scale. We rescale attributes into the range 0 to 1 and call it normalization. We use the MinMaxScaler class from scikit-learn.

Let's take an example.

```
import pandas, scipy, numpy
from sklearn.preprocessing import MinMaxScaler
df=pandas.read_csv( 'http://archive.ics.uci.edu/ml/machine-learning-
databases/winequality/winequality-red.csv ',sep=';')
array=df.values
#Separating data into input and OUTPUT components
x=array[:,0:8]
y=array[:,8]
scaler=MinMaxScaler(feature_range=(0,1))
rescaledX=scaler.fit_transform(x)
numpy.set_printoptions(precision=3) #Setting precision for the OUTPUT
print("OUTPUT")
rescaledX[0:5,:]
```

OUTPUT

```
array([[0.248, 0.397, 0.   , 0.068, 0.107, 0.141, 0.099, 0.568],
[0.283, 0.521, 0.   , 0.116, 0.144, 0.338, 0.216, 0.494],
[0.283, 0.438, 0.04 , 0.096, 0.134, 0.197, 0.17 , 0.509],
[0.584, 0.11 , 0.56 , 0.068, 0.105, 0.225, 0.191, 0.582],
[0.248, 0.397, 0.   , 0.068, 0.107, 0.141, 0.099, 0.568]])
```

b. Standardizing Data

With standardizing, we can take attributes with a Gaussian distribution and different means and standard deviations and transform them into a standard Gaussian distribution with a mean of 0 and a standard deviation of 1. For this, we use the StandardScaler class.

```
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler().fit(x)
rescaledX=scaler.transform(x)
print("OUTPUT")
rescaledX[0:5,:]
```

OUTPUT

```
array([[-0.528,  0.962, -1.391, -0.453, -0.244, -0.466, -0.379,  0.558],
[-0.299,  1.967, -1.391,  0.043,  0.224,  0.873,  0.624,  0.028],
[-0.299,  1.297, -1.186, -0.169,  0.096, -0.084,  0.229,  0.134],
```

```
[ 1.655, -1.384,  1.484, -0.453, -0.265,  0.108,  0.412,  0.664],
[-0.528,  0.962, -1.391, -0.453, -0.244, -0.466, -0.379,  0.558]])
```

c. Normalizing Data

In this task, we rescale each observation to a length of 1 (a unit norm). For this, we use the Normalizer class. Let's take an example.

```
from sklearn.preprocessing import Normalizer
scaler=Normalizer().fit(x)
normalizedX=scaler.transform(x)
print("OUTPUT")
normalizedX[0:5,:]
```

```
OUTPUT
array([[2.024e-01, 1.914e-02, 0.000e+00, 5.196e-02, 2.079e-03, 3.008e-01,
9.299e-01, 2.729e-02],
[1.083e-01, 1.222e-02, 0.000e+00, 3.611e-02, 1.361e-03, 3.472e-01,
9.306e-01, 1.385e-02],
[1.377e-01, 1.342e-02, 7.061e-04, 4.060e-02, 1.624e-03, 2.648e-01,
9.533e-01, 1.760e-02],
1.767e-01, 4.416e-03, 8.833e-03, 2.997e-02, 1.183e-03, 2.681e-01,
9.464e-01, 1.574e-02],
[2.024e-01, 1.914e-02, 0.000e+00, 5.196e-02, 2.079e-03, 3.008e-01,
9.299e-01, 2.729e-02]])
```

d. Binarizing Data

Using a binary threshold, it is possible to transform our data by marking the values above it 1 and those equal to or below it, 0. For this purpose, we use the Binarizer class.

```
from sklearn.preprocessing import Binarizer
binarizer=Binarizer(threshold=0.0).fit(x)
binaryX=binarizer.transform(x)
print("OUPUT")
binaryX[0:5,:]
```

```
OUPUT
array([[1., 1., 0., 1., 1., 1., 1., 1.],
[1., 1., 0., 1., 1., 1., 1., 1.],
[1., 1., 1., 1., 1., 1., 1., 1.],
[1., 1., 1., 1., 1., 1., 1., 1.],
[1., 1., 0., 1., 1., 1., 1., 1.]]
```

e. Mean Removal

We can remove the mean from each feature to center it on zero.

```
from sklearn.preprocessing import scale
data_standardized=scale(df)
data_standardized.mean(axis=0)
```

```
print("OUTPUT")
data_standardized.std(axis=0)
```

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

f. One Hot Encoding

When dealing with few and scattered numerical values, we may not need to store these. Then, we can perform One Hot Encoding. For k distinct values, we can transform the feature into a k-dimensional vector with one value of 1 and 0 as the rest values.

```
from numpy import array
from numpy import argmax
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import OneHotEncoder
# define example
data = ['cold', 'cold', 'warm', 'cold', 'hot', 'hot', 'warm', 'cold', 'warm', 'hot']
values = array(data)
print(values)
# integer encode
label_encoder = LabelEncoder()
integer_encoded = label_encoder.fit_transform(values)
print(integer_encoded)

# binary encode
onehot_encoder = OneHotEncoder(sparse=False)
integer_encoded = integer_encoded.reshape(len(integer_encoded), 1)
onehot_encoded = onehot_encoder.fit_transform(integer_encoded)
print(onehot_encoded)


# invert first example
inverted = label_encoder.inverse_transform([argmax(onehot_encoded[0, :])])
print("OUTPUT")
print(inverted)
```

```
OUTPUT :
['cold' 'cold' 'warm' 'cold' 'hot' 'hot' 'warm' 'cold' 'warm' 'hot']
[0 0 2 0 1 1 2 0 2 1]
[[1. 0. 0.]
[1. 0. 0.]
[0. 0. 1.]
[1. 0. 0.]
[0. 1. 0.]
[0. 1. 0.]
0. 0. 1.]
[1. 0. 0.]
[0. 0. 1.]
```

```
[0. 1. 0.]]
['cold']
```

g. Label Encoding Some labels can be words or numbers. Usually, training data is labelled with words to make it readable. Label encoding converts word labels into numbers to let algorithms work on them. Let’s take an example.

```
from sklearn.preprocessing import LabelEncoder
label_encoder=LabelEncoder()
input_classes=['Havells','Philips','Syska','Eveready','Lloyd']
_encoder.fit(input_classes)
LabelEncoder()
for i,item in enumerate(label_encoder.classes_):
print(item,'-->',i)
labels=['Lloyd','Syska','Philips']
label_encoder.transform(labels)
array([2, 4, 3], dtype='int32')
label_encoder.inverse_transform(label_encoder.transform(labels))
```

OUTPUT

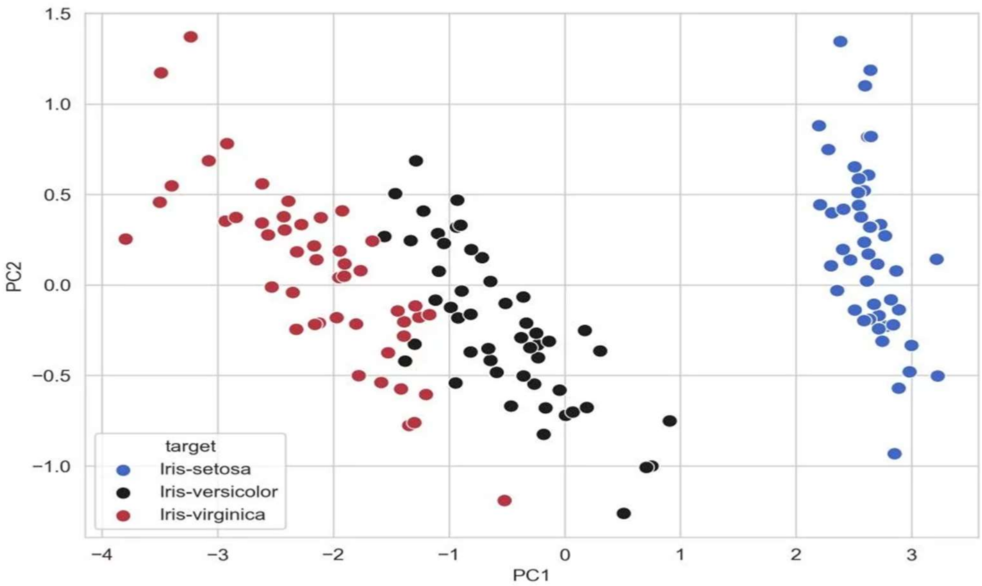
```
Eveready --> 0
Havells --> 1
Lloyd --> 2
Philips --> 3
Syska --> 4
array(['Lloyd', 'Syska', 'Philips'], dtype='<U8')
```

Task 3: Implement Dimensionality reduction using Principle Component Analysis (PCA) method.
Aim: To implement using python about using Matplotlib packages in Python

Program:

```
import numpy as np
import pandas as pd
import seaborn as sb
import matplotlib.pyplot as plt
def PCA(X , num_components):

#Step-1
X_meaned = X - np.mean(X , axis = 0)

#Step-2
cov_mat = np.cov(X_meaned , rowvar = False)

#Step-3
eigen_values , eigen_vectors = np.linalg.eigh(cov_mat)

#Step-4
sorted_index = np.argsort(eigen_values)[::-1]
sorted_eigenvalue = eigen_values[sorted_index]
sorted_eigenvectors = eigen_vectors[:,sorted_index]

#Step-5
eigenvector_subset = sorted_eigenvectors[:,0:num_components]

#Step-6
X_reduced =
np.dot(eigenvector_subset.transpose() , X_meaned.transpose() ).transpose()

return X_reduced
 #Get the IRIS dataset
 url = "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"
 data = pd.read_csv(url, names=
 ['sepal length','sepal width','petal length','petal width','target'])

 #prepare the data
 x = data.iloc[:,0:4]
 #prepare the target
 target = data.iloc[:,4]
 #Applying it to PCA function
 mat_reduced = PCA(x , 2)

 #Creating a Pandas DataFrame of reduced Dataset
 principal_df = pd.DataFrame(mat_reduced , columns = ['PC1','PC2'])

 #Concat it with target variable to create a complete Dataset
 principal_df = pd.concat([principal_df , pd.DataFrame(target)] , axis = 1)
```

```
plt.figure(figsize = (6,6))
sb.scatterplot(data = principal_df , x = 'PC1',y = 'PC2' , hue = 'target' , s = 60 , palette= 'icefire')
```



Task 4: Write a Python program to demonstrate various Data Visualization Techniques.

```
import pandas as pd
import matplotlib.pyplot as plt
 import seaborn as sns

 # Downdload dataset and read it
 csv_url = 'https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data'
# using the attribute information as the column names

col_names = ['Sepal_Length','Sepal_Width','Petal_Length','Petal_Width ','Class']
iris = pd.read_csv(csv_url, names = col_names)
iris.head()
iris["Class"].value_counts()
```

 # Line plots

```
import numpy as np
 x = np.linspace(0,20,30)
y= x**2
plt.plot(x, y)
 plt.show()
```

 # Line plot with grid

```
x = np.linspace(0,20,30)
 y= x**2 plt.plot(x, y)
plt.xlabel('x-values')
plt.ylabel('x^2-values')
```

```
plt.title('line plot')
 plt.grid(True)
 plt.show()
```

 # Scatter Plot

```
iris.plot(kind="scatter", x="Sepal_Length", y="Sepal_Width")
 colours = {'Iris-setosa':'orange', 'Irisversicolor':'lightgreen', 'Iris-virginica':'lightblue'}
 for i in range(len(iris['Sepal_Length']))
     plt.scatter(iris['Petal_Length'][i],iris['Petal_Width'][i], color = colours[iris['Class'][i]])
plt.title('Iris')
 plt.xlabel('petal length')
 plt.ylabel('petal width')
 plt.grid(True)
 plt.show()


# We can also use the seaborn library to make a similar plot
sns.jointplot(x="Sepal_Length", y="Sepal_Width", data=iris, size=5)
```

 # Bar Graph

```
 a= iris['Class'].value_counts()
species = a.index
count = a.values
 plt.bar(species,count,color = 'lightgreen')
plt.xlabel('species')
 plt.ylabel('count')
 plt.show()
```

 # Box Plot

```
 length_width = iris[['Petal_Length','Petal_Width','Sepal_Length','Sepal_Wi dth']]
 #excluding species column
 length_width.boxplot()
 plt.xlabel('Flower measurements')
plt.ylabel('values')
plt.title("Iris dataset analysis")
 # We can look at an individual feature in Seaborn through mnay different kinds of plots.
# Here's a boxplot
sns.boxplot(x="Class", y="Petal_Length", palette="husl", data=iris)
```

#Histogram

```
 import numpy as np
 data_ = np.random.randn(1000)
 plt.hist(data_,bins = 40,color='gold')
plt.grid(True)
 plt.xlabel('points')
plt.title("Histogram")
plt.show()
```

Task 5: Implement Simple and Multiple Linear Regression Models.

Program:

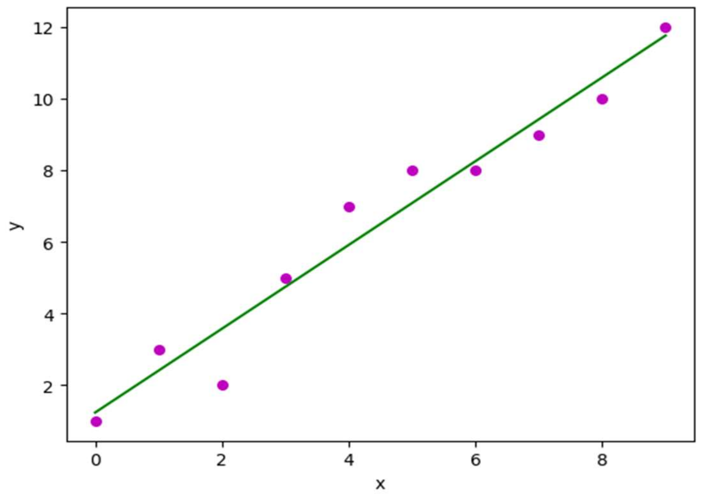
Simple Linear Regression:

```
import numpy as np
import matplotlib.pyplot as plt
def estimate_coef(x, y):
        # number of observations/points
        n = np.size(x)

        # mean of x and y vector
        m_x = np.mean(x)
        m_y = np.mean(y)

        # calculating cross-deviation and deviation about x
        SS_xy = np.sum(y*x) - n*m_y*m_x
        SS_xx = np.sum(x*x) - n*m_x*m_x

        # calculating regression coefficients
        b_1 = SS_xy / SS_xx
        b_0 = m_y - b_1*m_x

        return (b_0, b_1)

def plot_regression_line(x, y, b):
        # plotting the actual points as scatter plot
        plt.scatter(x, y, color = "m",
                        marker = "o", s = 30)

        # predicted response vector
        y_pred = b[0] + b[1]*x

        # plotting the regression line
```

```
    plt.plot(x, y_pred, color = "g")

    # putting labels
    plt.xlabel('x')
    plt.ylabel('y')

    # function to show plot
    plt.show()

def main():
    # observations / data
    x = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
    y = np.array([1, 3, 2, 5, 7, 8, 8, 9, 10, 12])

    # estimating coefficients
    b = estimate_coef(x, y)
    print("Estimated coefficients:\nb_0 = {} \
        \nb_1 = {}".format(b[0], b[1]))

    # plotting regression line
    plot_regression_line(x, y, b)

if __name__ == "__main__":
    main()
```

OUTPUT

```
Estimated coefficients:
b_0 = 1.2363636363636363
b_1 = 1.1696969696969697
```



Multiple linear regression:

```
import numpy as np
import matplotlib.pyplot as plt

def estimate_coef(x, y):
                        # number of observations/points
                        n = np.size(x)

                        # mean of x and y vector
                        m_x = np.mean(x)
                        m_y = np.mean(y)

                        # calculating cross-deviation and deviation about x
                        SS_xy = np.sum(y*x) - n*m_y*m_x
                        SS_xx = np.sum(x*x) - n*m_x*m_x

                        # calculating regression coefficients
                        b_1 = SS_xy / SS_xx
                        b_0 = m_y - b_1*m_x

                        return (b_0, b_1)

def plot_regression_line(x, y, b):
```

```
        # plotting the actual points as scatter plot
        plt.scatter(x, y, color = "m",
                    marker = "o", s = 30)

        # predicted response vector
        y_pred = b[0] + b[1]*x

        # plotting the regression line
        plt.plot(x, y_pred, color = "g")

        # putting labels
        plt.xlabel('x')
        plt.ylabel('y')

        # function to show plot
        plt.show()

def main():
        # observations / data
        x = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
        y = np.array([1, 3, 2, 5, 7, 8, 8, 9, 10, 12])

        # estimating coefficients
        b = estimate_coef(x, y)
        print("Estimated coefficients:\nb_0 = {} \
              \nb_1 = {}".format(b[0], b[1]))

        # plotting regression line
        plot_regression_line(x, y, b)

if __name__ == "__main__":
        main()
```

OUTPUT:

```
Estimated coefficients:
b_0 = 1.2363636363636363
b_1 = 1.1696969696969697
```