Name: Aveeral Prakash

Subject: Statistics Lab

Roll No.:20BCAB60

<u>Lab Practicals</u>:-

Practical 1(Linear Algebra):

```
import numpy as np
x=np.array([[1,2,3],[3,2,1]])
y=np.array([[1,2,3],[3,2,1]]).T
print(x)
print(y)
#7 tuple 1d array
a= np.array((1,2,3,4,5,6,7))
print(a)
#2,4,7th array element
print(a[1])
print(a[3])
print(a[6])

#array shape
print(a.shape)

#array transpose
t=np.array([[1,2,3,4,5,6,7]]).T
print(t)
print(t.shape)

#4x5 matrix
m=np.array([[1,0,0,0,0],[0,1,0,0,0],[0,0,1,0,0],[0,0,0,1,0]])
print(m)

#matrix shape
```

```
print(np.shape(m))

#matrix transpose
mt=np.array([[1,0,0,0,0],[0,1,0,0,0],[0,0,1,0,0],[0,0,0,1,0]])
.T
print(mt)

#matrix first col
print(m[:,0])

#matrix first row
print(m[0,:])

#7D array with 0s, 3d array with 1s
print(np.zeros(7))
print(np.ones(3))
#print(np.array([[0,0,0,0,0,0,0]]*7))
```

Practical 2(Linear Algebra):

```
import numpy as np
from numpy import linalg

#6D vector
v = np.array([[1,2,3,4,5,6]])
print(v)
print("\n")

#transpose
print(v.T)
print("\n")

#2 non-square matrix that can be multiplied
m1=np.array([[1,0,2],[0,1,3]])
m2=np.array([[1,2],[2,1],[3,4]])

#matrix shape
print(m1.shape)
print(m2.shape)
print("\n")

# matrix product
print(np.matmul(m1,m2))
z=np.array([np.zeros(2)]*2)
for i in range(len(m1)):
    for j in range(len(m2[1])):
        for k in range(len(m2)):
            z[i][j] += m1[i][j] * m2[i][j]
print(z)
```

```
print("\n")

# sum: two non square matrices of same order
x=np.array([[1,2,1],[2,1,2]])
y=np.array([[1,1,1],[2,2,3]])
print(x+y)
print("\n")

# Define a square matrix A.
a=np.matrix([[1,2],[3,4]])
print("\n")

# Print the identity matrix of the above order I.
i=np.array([[1,0],[0,1]])
print(i)
print("\n")

# Verify A.I = I.A for matrix multiplication.
print("A.I= ",a@i)
print("I.A= ",i@a)
print("A.I = I.A")

# Define another square matrix of the same order as A
a1=np.array([[4,5],[6,7]])
print("\n")

# Print the product of the matrices as matrix multiplication
print(a@a1)
print("\n")
```

```
# Print the product of the matrices by element wise
multiplication

print(np.multiply(a,a1))

print("\n")


# Calculate and print the inverse of A. (Use linalg)Check if
determinant is 0 Use if else statement to calculate inverse
only when determinant is non zero

d=np.linalg.det(a)

print("Determinant: ",d)

if d!=0:

    print("Inverse: ", np.linalg.inv(a))

else:

    print("Inverse does not exist")
```

Practical 3(Basic EDA,plots):

```
#basic EDA, plots, using inbuilt iris dataset
import pandas as pd
import os
import seaborn as sns
import matplotlib.pyplot as plt
os.chdir("C:/Users/aveer/Documents/Dataset")
iris = pd.read_csv('Iris.csv')
print(iris.head())
print(iris.describe())
sns.countplot(x='Species', data=iris)
sns.scatterplot('SepalLengthCm','SepalWidthCm',
hue='Species',data=iris)
sns.pairplot(iris.drop(['Id'], axis
=1),hue='Species',height=2)
#sns.boxenplot()
x=iris.corr(method='pearson')
print(x)
#sns.heatmap(iris.corr_matrix,method='pearson'.drop(['Id'],axi
s=1).drop(['Id'],axis=0))
sns.heatmap(iris.corr(method='pearson').drop(['Id'],axis=1).dr
op(['Id'],axis=0))
sns.heatmap(iris.corr(), data = iris)
plt.boxplot('SepalWidthCm', data=iris)
plt.show()
```

Practical 4(EDA And Linear Regression):

```
import pandas as pd
import os
import seaborn as sns
#import matplotlib.pyplot as plt
from sklearn import linear_model
#from sklearn.linear_model import LinearRegression

os.chdir("C:/Users/aveer/Documents/Dataset")
mtcars = pd.read_csv('CarPrice_Assignment.csv')
print(mtcars.head())
print(mtcars.describe())
sns.pairplot(mtcars)
sns.scatterplot(x='horsepower', y='price', data=mtcars)
sns.scatterplot(x='compressionratio', y='price', data=mtcars)
sns.scatterplot(x='enginesize', y='price', data=mtcars)
sns.scatterplot(x='cylindernumber', y='price', data=mtcars)
sns.countplot(x='enginetype', data=mtcars)
sns.scatterplot(x='enginetype', y='price', data=mtcars)
sns.scatterplot(x='carheight', y='price', data=mtcars)
sns.scatterplot(x='carwidth', y='price', data=mtcars)
sns.scatterplot(x='carlength', y='price', data=mtcars)
sns.scatterplot(x='wheelbase', y='price', data=mtcars)
sns.scatterplot(x='fueltype', y='price', data=mtcars)
#sns.pairplot(mtcars.drop(['car_ID'],axis=1),height=3)
sns.boxplot(y='price', data=mtcars)
sns.boxplot(x='enginetype', y='price', data=mtcars)
sns.boxplot(x='fueltype', y='compressionratio',data=mtcars)
#plt.show()

#one variable regression
```

```
x=mtcars[['price']]
y=mtcars[['highwaympg']]
reg=linear_model.LinearRegression()
#reg=LinearRegression()
#reg.fit([[0,0],[1,1],[2,2],[0,1,2]])
reg.fit(x,y)
print(reg.coef_)
sns.regplot(x,y)

#multiple  regression
X=mtcars[['horsepower','curbweight']]
Y=mtcars[['price']]
reg=linear_model.LinearRegression()
reg.fit(X,Y)
print(reg.coef_)
```

Practical 5(R Code)(Hypothesis Testing):

```
#hypothesis testing, CarPrice Assignment dataset

setwd("C:/Users/aveer/Documents/Dataset")

data=read.csv("CarPrice_Assignment.csv")

View(data)

# if p value is less than alpha(significance value)(alpha = 1-
confidence level), we reject null hypothesis

#Ho: mean of enginesize = 120

#H1: mean of enginesize is not equal to 120

mean(data$enginesize) #mean = 126.9073

t.test(data$enginesize,mu=120,alternative="less",conf.level=0.
95) #p=0.9908
```

Practical 6(Factor Analysis):

```
import os

import pandas as pd

from factor_analyzer import FactorAnalyzer

import seaborn as sns

import matplotlib.pyplot as plt

#from factor_analyzer.factor_analyzer import
calculate_bartlett_sphericity

from factor_analyzer.factor_analyzer import calculate_kmo


os.chdir("C:/Users/aveer/Documents/Dataset")

df=pd.read_csv('FIFA 2018 Statistics.csv')

#dropping the non-numeric columns

df.drop(['Date','Team','Opponent','Man of the
Match','Round'],axis=1,inplace=True)


#drop missing values rows

df.dropna(inplace=True)

#df.fillna(0) #df.replace(np.nan,0)


df.info()


# Checking the correlation

x= df.corr(method= 'pearson')

print(x)

sns.heatmap(df.corr(method='pearson'),data=df)

plt.show()


#adequacy test

# Bartlett’s test

#chi_square_value,p_value=calculate_bartlett_sphericity(df)
```

```
#print(chi_square_value, p_value)


# Kaiser-Meyer-Olkin (KMO) Test
kmo_all,kmo_model=calculate_kmo(df)
print(kmo_model)
# KMO values range between 0 and 1. Value of KMO less than 0.5
is considered inadequate.
# The overall KMO for our data is 0.76, which is pretty good
# This value indicates that we can proceed with our planned
factor analysis.



# Choosing the Number of Factors
# Create factor analysis object and perform factor analysis
fa = FactorAnalyzer()
fa.fit(df)


#Check Eigenvalues
ev, v = fa.get_eigenvalues() #eigen_values, vectors =
fa.get_eigenvalues()
print(ev) #print(vectors) #print(eigen_values)
# 3-factors eigen values are greater than 1
# we choose only 3 factors/unobserved variables



# Create scree plot
plt.scatter(range(1,df.shape[1]+1),ev)
plt.plot(range(1,df.shape[1]+1),ev)
plt.title('Scree Plot')
plt.xlabel('Factors')
plt.ylabel('Eigenvalue')
plt.grid()
```

```
plt.show()

# From the scree plot we can see that the number of factors=3
or 4.


# Create factor analysis object and perform factor analysis

fa = FactorAnalyzer()

fa.set_params(n_factors=3, rotation='varimax')

fa.fit(df)

loadings = fa.loadings_

print(loadings)



# Get variance of each factors

print(fa.get_factor_variance())

# Output is in the format:

#                   Factor 1    Factor2    Factor3

# SS Loadings

# Proportion Var

# Cumulative Var


# Total 52% cumulative Variance is explained by the 3 factors.
```

Practical 7(Logistic Regression):

```
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score

os.chdir("C:/Users/aveer/Documents/Dataset")
dataset = pd.read_csv('User_Data.csv')
#dataset.drop(['User_ID','Gender'])

#to predict whether a user will purchase the product or not,
#we have to find out the relationship between Age and
Estimated Salary.

# input
x = dataset.iloc[:, [2, 3]].values

# output
y = dataset.iloc[:, 4].values

#split data
xtrain, xtest, ytrain, ytest = train_test_split(x, y,
test_size = 0.25, random_state = 0)

#feature scaling age and salary as they lie in different
ranges
```

```
#If not done, salary will dominate age when the model finds
the nearest neighbor to a data point in data space.
sc_x = StandardScaler()
xtrain = sc_x.fit_transform(xtrain)
xtest = sc_x.transform(xtest)
print (xtrain[0:10, :])
#o/p ranges from -1 to 1, equal contribution in finalizing
hypothesis


#train our model
classifier = LogisticRegression(random_state = 0)
classifier.fit(xtrain, ytrain)


#use model on test data
y_pred = classifier.predict(xtest)


#test performance of model on confusion matrix
cm = confusion_matrix(ytest, y_pred)
print ("Confusion Matrix : \n", cm)
# [[65  3]   TP FP
#  [ 8 24]]  FN TN


# accuracy
print ("Accuracy : ", accuracy_score(ytest, y_pred))
# 0.89
# which is pretty good
```

Practical 8(Clustering Analysis):

```
from sklearn.datasets import load_iris
from sklearn.cluster import AgglomerativeClustering
import matplotlib.pyplot as plt
from scipy.cluster.hierarchy import dendrogram, linkage

data = load_iris()
df = data.data
df = df[:,:]
z = linkage(df, method= "ward")
dendro=dendrogram(z)
plt.title('Dendrogram')
plt.ylabel('Euclidean distance')
plt.show()
ac =
AgglomerativeClustering(n_clusters=3,affinity="euclidean",
linkage="ward")

labels= ac.fit_predict(df)
plt.figure(figsize = (8,5))
plt.scatter(df[labels == 0, 0], df[labels == 0,1],c="red")
plt.scatter(df[labels == 1, 0], df[labels==1, 1],c="blue")
plt.scatter(df[labels == 2, 0], df[labels== 2, 1],c="green")
plt.scatter(df[labels == 3, 0], df[labels== 3, 1],c="black")
plt.scatter(df[labels == 4, 0], df[labels== 4, 1],c="orange")
plt.show()
```

Practical 9 (Hierarchical Clustering) :

```
import os
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import normalize
import scipy.cluster.hierarchy as shc
from sklearn.cluster import AgglomerativeClustering

os.chdir("C:/Users/aveer/Documents/Dataset")
data = pd.read_csv('Wholesale customers data.csv')
print(data.head())

#normalize data so the scale of each variable is same
#if not done, model might become biased towards variables with
higher magnitude (in this case fresh or milk)
data_scaled = normalize(data)
data_scaled = pd.DataFrame(data_scaled, columns=data.columns)
print(data_scaled.head())
#similar scales

#Dendrogram to decide the number of clusters
plt.figure(figsize=(10, 7))
plt.title("Dendrograms")
d = shc.dendrogram(shc.linkage(data_scaled, method='ward'))
#x=samples, y=distance between samples. threshold=6

plt.figure(figsize=(10, 7))
plt.title("Dendrograms")
d = shc.dendrogram(shc.linkage(data_scaled, method='ward'))
plt.axhline(y=6, color='r', linestyle='--')
#line divides forming 2 clusters
```

```
#apply hierarchical clustering for 2 clusters

cluster = AgglomerativeClustering(n_clusters=2,
affinity='euclidean', linkage='ward')

print(cluster.fit_predict(data_scaled))

#0=cluster 1, 1=cluster 2


#visualize clusters

plt.figure(figsize=(10, 7))

plt.scatter(data_scaled['Milk'], data_scaled['Grocery'],
c=cluster.labels_)
```