PART A

1. MEAN,MODE,MEDIAN

```
x=[115.3, 195.5, 120.5, 110.2, 90.4, 105.6, 110.9, 116.3, 122.3, 125.4,90.4]
-
mean = sum(x) / len(x)
-
mean
-
import statistics
-
print(statistics.mean(x))
-
x.sort()
-
x
-
n=len(x)
-
if  n% 2 == 0:
   median1 = x[n//2]
   median2 = x[n//2 - 1]
   median = (median1 + median2)/2
else:
   median = x[n//2]
```

```
print("Median is: " + str(median))

-

print(statistics.median(x))

-

#mode
X=(115.3, 195.5, 120.5, 110.2, 90.4, 105.6, 110.9, 116.3, 122.3, 125.4, 90.4)
d={}
for i in X:
    if i in d:
        d[i]=d[i]+1
    else:
        d[i]=1
max=0
for i in d:
    if(d[i]>max):
        max=d[i]
        ans=i
print("MODE",ans)

-

variance = sum((i - mean) ** 2 for i in x) / (n-1)

-

variance

-

print(statistics.variance(x))

-

standarddev= variance ** 0.5
```

-

standarddev

-

```
print(statistics.stdev(x))
```

-

```
#normalization
def nor(l):
    min=l[0]
    max=l[0]
    for i in l:
        if(i>max):
            max=i
        elif(i<min):
            min=i

    for i in l:
        print((i-min)/(max-min))

l=(115.3, 195.5, 120.5, 110.2, 90.4, 105.6, 110.9, 116.3, 122.3, 125.4,90.4)
nor(l)
```

-

```
#STANDARDIZATION
import numpy as np

def std(lst):
    mean_val = np.mean(lst)
    sd_val = np.std(lst)

    standardized_list = [(x - mean_val) / sd_val for x in lst]

    return standardized_list

# Example usage
data_list = [115.3, 195.5, 120.5, 110.2, 90.4, 105.6, 110.9, 116.3, 122.3, 125.4, 90.4]
result = std(data_list)
print(result)
```

2. PCA

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler
-
df=pd.read_csv('/home/exam/Downloads/iris(For PCA Program) (2).csv')
-
df.head()
-
X=df.drop(['species'],axis=1)
X
-
```

```
Y=df['species']
Y
-
X_scaled = StandardScaler().fit_transform(X)
X_scaled[:5]
-
features = X_scaled.T
cov_matrix = np.cov(features)
cov_matrix[:5]
values, vectors = np.linalg.eig(cov_matrix)
values[:5]
vectors[:5]
-
explained_variances = []
for i in range(len(values)):
 explained_variances.append((values[i] / np.sum(values))*100)
print("variances of each feature",explained_variances)
plt.figure(figsize=(8,4))
plt.bar(range(4),explained_variances, alpha=0.6)
plt.ylabel('Percentage of explained variance')
plt.xlabel('Dimensions')
projected_1 = X_scaled.dot(vectors.T[0])
projected_2 = X_scaled.dot(vectors.T[1])
res = pd.DataFrame(projected_1, columns=['PC1'])
res['PC2'] = projected_2
res['Y'] = Y
```

```
res.head()

sns.FacetGrid(res, hue="Y", height=6).map(plt.scatter, 'PC1', 'PC2').add_legend()

plt.show()
```

3. K-MEANS

```
import numpy as np

from sklearn.cluster import KMeans


X = np.array([[5.9, 3.2],[4.6, 2.9],[6.2, 2.8],[4.7, 3.2],[5.5, 4.2],[5.0, 3.0],[4.9, 3.1],[6.7, 3.1],

    [5.1, 3.8],

    [6.0, 3.0],

    # Continue with your 2D data points

])


k = 3


inital_clusters = np.array([[6.2,3.2],[6.6,3.7],[6.5,3.0]])


# First subquestion - run different kmeans with different variables for each question
```

```
kmeans = KMeans(n_clusters=k  , init=inital_clusters, n_init=1, max_iter=1 ) # runs only one iteration

kmeans.fit(X)

first_cluster_center = kmeans.cluster_centers_

a = first_cluster_center[0] # the index 0 is red - 1 is green , 2 is blue CONFIRM BASED ON QUESTION

print("after one iteration , red is: ", first_cluster_center )

iters = kmeans.n_iter_

print("Number of iterations: ", iters)


# Second subquestion



km = KMeans(n_clusters=k  , init=inital_clusters, n_init=1, max_iter=2 ) # runs only two iterations

km.fit(X)

first_cluster_center = km.cluster_centers_

b = first_cluster_center[1] # green COLOR

print("after two iterations , green is: ", b )

iters = km.n_iter_

print("Number of iterations: ", iters) # prints number of iterations


# Third Subquestion


kmx = KMeans(n_clusters=k  , init=inital_clusters, n_init=1) # runs iterations till convergence

kmx.fit(X)
```

```
first_cluster_center = kmx.cluster_centers_

b = first_cluster_center[2] # blue color

print("after cluster converges , blue is: ", b )

iters = kmx.n_iter_

print("Number of iterations: ", iters)



# Fourth subquestion


boo = KMeans(n_clusters=k  , init=inital_clusters, n_init=1) # runs only iterations till convergence

boo.fit(X)

first_cluster_center = boo.cluster_centers_


iters = boo.n_iter_

print("Number of iterations for all clusters to converge: ", iters)
```

4. DECISION TREE

```
import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.metrics import confusion_matrix

from sklearn.tree import DecisionTreeClassifier


zoo_data = pd.read_csv("zoo_data(For Decision Tree Program).csv")


X = zoo_data.iloc[: , :-1]
```

```
y = zoo_data.iloc[: , -1]

X_train , X_test , y_train , y_test = train_test_split(X,y, test_size=0.3,train_size=0.7)

dt_model = DecisionTreeClassifier()

dt_model.fit(X_train,y_train)
y_pred = dt_model.predict(X_test)

conf = confusion_matrix(y_test , y_pred)
print("Confusion Matrix is\n" , conf)
-
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred,zero_division=0))
-
from sklearn import tree
from matplotlib import pyplot as plt
fig=plt.figure(figsize=(25,20))
_=tree.plot_tree(dt_model)
```

5.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_squared_error, r2_score
import matplotlib.pyplot as plt

# Load the dataset
df = pd.read_csv('Food-Truck(For Linear Regression Program).csv')
X = df.iloc[: , :-1]
y = df.iloc[: , -1]

# Scatter plot
plt.scatter(X, y)
plt.title('Scatter Plot of X vs Y')
plt.xlabel('X')
plt.ylabel('Y')
plt.show()

# Build a correlation matrix
correlation_matrix = df.corr()

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3,train_size=0.7, random_state=42)
model = LinearRegression()
model.fit(X_train, y_train)

# Predict on the test set
y_pred = model.predict(X_test)

# Compute regression parameters
```

```
intercept = model.intercept_
slope = model.coef_[0]

# Compute metrics
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

# Calculate SSE, SSR, SST
SSE = np.sum((y_pred - y_test) ** 2)
SSR = np.sum((y_pred - np.mean(y_test)) ** 2)
SST = np.sum((y_test - np.mean(y_test)) ** 2)

# Print results
#print(f'Intercept: {intercept}')
#print(f'Slope: {slope}')
print("Cost:",mse)
print("R-squared (R2):",r2)
print("Sum of Squared Errors (SSE):",SSE)
print("Sum of Squared Residuals (SSR):",SSR)
print("Total Sum of Squares (SST):",SST)

# Display correlation matrix
print('\nCorrelation Matrix:')
print(correlation_matrix)
```

6. LOGISTIC REGRESSION

```
import pandas as pd

import numpy as np

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import StandardScaler

from sklearn.model_selection import KFold

-

col_names=['Univ_Mrks','School_Mrks','Admission']

df=pd.read_csv("Student-University(For Logistic Regression Program).csv")

-

print(df)

df.head()

-

x=df.iloc[:,[0,1]].values

y=df.iloc[:,2].values

-

for train_index,test_index in kf.split(p):

    xtrain,xtest,ytrain,ytest=train_test_split(p,y,test_size=0.20,random_state=0)

    x1=xtrain[:,0]

    x2=xtrain[:,1]

    b0=0.0

    b1=0.0

    b2=0.0

    epoch=10000

    alpha=0.001
    while(epoch>0):
        for i in range(len(xtrain)):
            prediction=1/(1+np.exp(-(b0+b1*x1[i]+b2*x2[i])))
            b0=b0+alpha*(ytrain[i]-prediction)*prediction*(1-prediction)*1.0
            b1=b1+alpha*(ytrain[i]-prediction)*prediction*(1-prediction)*x1[i]
            b2=b2+alpha*(ytrain[i]-prediction)*prediction*(1-prediction)*x2[i]
            epoch=epoch-1
-
print(b0)
print(b1)
print(b2)
-
final_prediction=[]
x3=xtest[:,0]
x4=xtest[:,1]

print(ytest)
-
y_pred=[0]*len(xtest)

for i in range(len(xtest)):
    y_pred[i]=np.round(1/(1+np.exp(-(b0+b1*x3[i]+b2*x4[i]))))
    final_prediction.append(np.ceil(y_pred[i]))
-
print(final_prediction)
-
```

```
from sklearn.metrics import accuracy_score

print("Accuracy",accuracy_score(ytest,y_pred))
```