Name: Sneha Mishra

UIN: 733000826

Assignment No. 7: STAT-650

Problem 1

This question uses the california_housing dataset from the package. To load the dataset, use the following code to load the dataset and set random seed as SEED=10

```
from sklearn.datasets import fetch_california_housing;
california_housing = fetch_california_housing(as_frame=True);
california_housing.frame;
SEED = 10
```

In [ ]:
```
# Importing the various required libraries

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

In [ ]:
```
#Loading california_housing data from sklearn datasets

from sklearn.datasets import fetch_california_housing
california_housing = fetch_california_housing(as_frame=True)
df = california_housing.frame
SEED = 10
```

In [ ]:
```
#Gleaning at the dataset for the first few rows and the contained attributes
df.head()
```

Out[ ]:

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude	MedHouseVal
0	8.3252	41.0	6.984127	1.023810	322.0	2.555556	37.88	-122.23	4.526
1	8.3014	21.0	6.238137	0.971880	2401.0	2.109842	37.86	-122.22	3.585
2	7.2574	52.0	8.288136	1.073446	496.0	2.802260	37.85	-122.24	3.521
3	5.6431	52.0	5.817352	1.073059	558.0	2.547945	37.85	-122.25	3.413
4	3.8462	52.0	6.281853	1.081081	565.0	2.181467	37.85	-122.25	3.422

1.1) Create a new feature called MedHouseValCat by discretizing the atribute MedHouseVal into four classes labeled as 1,2,3, and 4 and adding it to your dataframe. Show that the frequency of these four class labels is approximately 5160 (Hint pd.qcut(feature_to_be _discritized, number_of_classes,retbins = False, labels)).

In [ ]:
```
#Creating dataframe MedHouseValCat and adding it to df.
df['MedHouseValCat'] = pd.qcut(df['MedHouseVal'], 4, retbins=False, labels=[1,2,3,4])
```

In [ ]:
```
#Finding frequency of four class labels
df['MedHouseValCat'].value_counts()
```

Out[ ]:
```
1    5162
2    5161
4    5160
3    5157
Name: MedHouseValCat, dtype: int64
```

In [ ]:
```
#Dataframe after addition of 'MedHouseValCat' to its attributes
df.head(15)
```

Out[ ]:

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude	MedHouseVal	MedHouseValCat
0	8.3252	41.0	6.984127	1.023810	322.0	2.555556	37.88	-122.23	4.526	4
1	8.3014	21.0	6.238137	0.971880	2401.0	2.109842	37.86	-122.22	3.585	4
2	7.2574	52.0	8.288136	1.073446	496.0	2.802260	37.85	-122.24	3.521	4
3	5.6431	52.0	5.817352	1.073059	558.0	2.547945	37.85	-122.25	3.413	4
4	3.8462	52.0	6.281853	1.081081	565.0	2.181467	37.85	-122.25	3.422	4
5	4.0368	52.0	4.761658	1.103627	413.0	2.139896	37.85	-122.25	2.697	4
6	3.6591	52.0	4.931907	0.951362	1094.0	2.128405	37.84	-122.25	2.992	4
7	3.1200	52.0	4.797527	1.061824	1157.0	1.788253	37.84	-122.25	2.414	3
8	2.0804	42.0	4.294118	1.117647	1206.0	2.026891	37.84	-122.26	2.267	3
9	3.6912	52.0	4.970588	0.990196	1551.0	2.172269	37.84	-122.25	2.611	3
10	3.2031	52.0	5.477612	1.079602	910.0	2.263682	37.85	-122.26	2.815	4
11	3.2705	52.0	4.772480	1.024523	1504.0	2.049046	37.85	-122.26	2.418	3
12	3.0750	52.0	5.322650	1.012821	1098.0	2.346154	37.85	-122.26	2.135	3
13	2.6736	52.0	4.000000	1.097701	345.0	1.982759	37.84	-122.26	1.913	3
14	1.9167	52.0	4.262903	1.009677	1212.0	1.954839	37.85	-122.26	1.592	2

From the above outputs, we can conclude that:

1) Attribute 'MedHouseValCat' has been added to the dataframe df. 2) The approximate frequency of these four class labels is 5160.

1.2) Define the feature matrix X by excluding MedHouseVal and MedHouseValCat and attributes and dependent variable matrix Y including MedHouseValCat.

```
In [ ]: #Defining feature matrix X and dependent variable matrix y.
        X,y = df.iloc[:,0:8], df.iloc[:,-1]
```

1.3) Divide the data into training and testing by allocating 25 percent of the samples to testing.

```
In [ ]: #Dividing the data into training and testing by allocating 25% of the samples to testing.
        from sklearn.model_selection import train_test_split
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 10)
```

1.4) Scale the data using the standard scaler and then perform a KNN classification using the five nearest neighbors. Show that the classification accuracies at training and testing are 74.47 and 63.48, respectively.

```
In [ ]: #Scaling the data using standard scaler
        from sklearn.preprocessing import StandardScaler
        scaler = StandardScaler()
        scaler.fit(X_train)
        X_train_scaled = scaler.transform(X_train)
        X_test_scaled = scaler.transform(X_test)
```

```
In [ ]: #Performing KNN_classification using nearest neighbors=5
        from sklearn.neighbors import KNeighborsClassifier
        KNNmodel_ = KNeighborsClassifier(n_neighbors = 5)
        KNNmodel_.fit(X_train_scaled, y_train.values.ravel())

Out[ ]: KNeighborsClassifier()
```

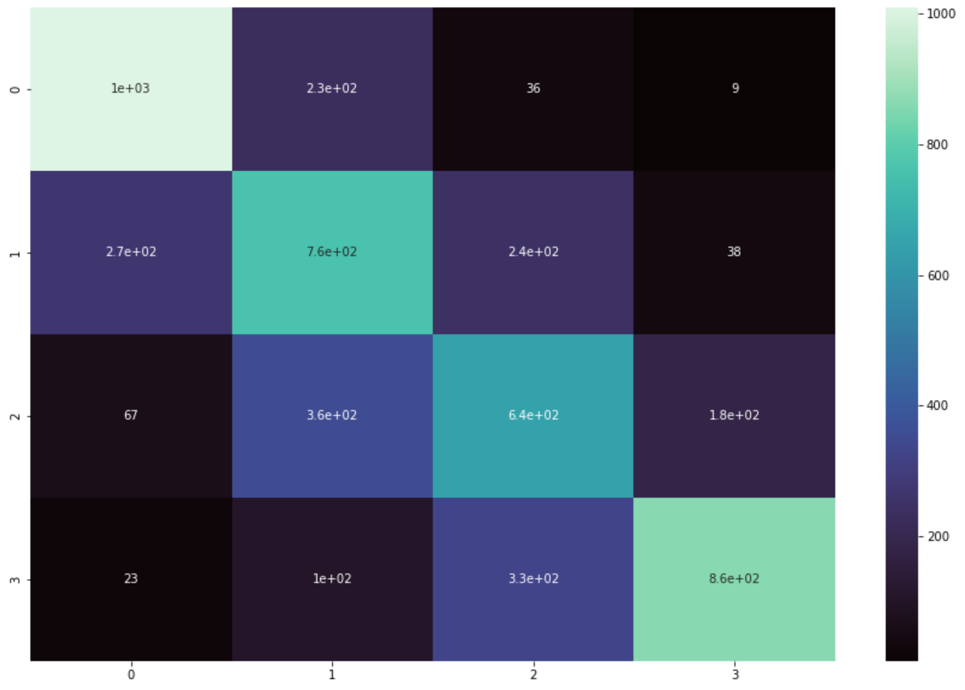
```
In [ ]: y_pred_train = KNNmodel_.predict(X_train_scaled)
        y_pred_test = KNNmodel_.predict(X_test_scaled)
```

```
In [ ]: #Displaying classification accuracies at training and testing data.
        from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

        print("Accuracy of Training data:", accuracy_score(y_train,y_pred_train))
        print("Accuracy of Testing data:", accuracy_score(y_test,y_pred_test))

        Accuracy of Training data: 0.744702842377261
        Accuracy of Testing data: 0.6348837209302326
```

1.5) Visualize the confusion matrix as a heatmap and print the classification report for the testing dataset.

```
In [ ]: #Displaying confusion matrix
        c_matrix = confusion_matrix(y_test, y_pred_test)
        print("Confusion Matrix:\n", c_matrix)

        Confusion Matrix:
         [[1009  231   36    9]
         [ 270  761  243   38]
         [  67  355  644  183]
         [  23  103  326  862]]
```

```
In [ ]: #Visualization of confusion matrix as a heatmap
        plt.figure(figsize=(15, 10))
        sns.heatmap(c_matrix, annot=True, cmap='mako')

Out[ ]: <AxesSubplot:>
```



In [ ]:
```python
#Displaying classification report for the testing dataset
c_report = classification_report(y_test, y_pred_test)
print("Classification Report:\n", c_report)
```

```
Classification Report:
               precision    recall  f1-score   support

           1       0.74      0.79      0.76      1285
           2       0.52      0.58      0.55      1312
           3       0.52      0.52      0.52      1249
           4       0.79      0.66      0.72      1314

    accuracy                           0.63      5160
   macro avg       0.64      0.63      0.64      5160
weighted avg       0.64      0.63      0.64      5160
```

1.6) Tune the hyperparameters, the number of near neighbours, and the weight function using GridSearchCV. Show that 13 is the optimal number of nearest neighbors and distance is the best function for computing weight. Calculate the mean squared errors for the training and testing of the classifier using the tuned hyperparameters.

In [ ]:
```python
#Optimal no. of nearest neighbors using gridsearch and best function for computing weight.
from sklearn.model_selection import GridSearchCV

parameters = { "n_neighbors": range(1, 25),"weights": ["uniform", "distance"]}
KNNgridsearch_ = GridSearchCV(KNeighborsClassifier(), parameters)
KNNgridsearch_.fit(X_train_scaled, y_train.values.ravel())

KNNgridsearch_.best_params_
```

Out[ ]:
```
{'n_neighbors': 13, 'weights': 'distance'}
```

In [ ]:
```python
#Displaying Optimal no. of nearest neighbors and best function for computing weight
print(f"The optimal number of n_neighbors for KNN are: {KNNgridsearch_.best_params_['n_neighbors']}")
print(f"The Best function to compute distance: {KNNgridsearch_.best_params_['weights']}")
```

```
The optimal number of n_neighbors for KNN are: 13
The Best function to compute distance: distance
```

In [ ]:
```python
#Calculating mean squared errors for training and testing of the classifier using the tuned hyperparameters
from sklearn.metrics import mean_squared_error
from math import sqrt

grid_y_pred_train = KNNgridsearch_.predict(X_train_scaled)
print("Train_mse:", mean_squared_error(y_train, grid_y_pred_train))
print('Train_rmse:', sqrt(mean_squared_error(y_train, grid_y_pred_train)))

grid_y_pred_test = KNNgridsearch_.predict(X_test_scaled)
print("Test_mse:", mean_squared_error(y_test, grid_y_pred_test))
print('Test_rmse:', sqrt(mean_squared_error(y_test, grid_y_pred_test)))
```

```
Train_mse: 0.0
Train_rmse: 0.0
Test_mse: 0.5341085271317829
Test_rmse: 0.7308272895368528
```

1.7) Answer this question based on the data in step 3. Create a pipeline that includes a standard scaler and a KNN classifier with 13 nearest neighbors. Use your Pipeline to perform 10-fold cross-validation. Display the results of the classification showing an accuracy of 0.688 +/- 0.018.

In [ ]:
```python
#Creating required pipeline
from sklearn.pipeline import Pipeline
from sklearn.model_selection import cross_val_score
```

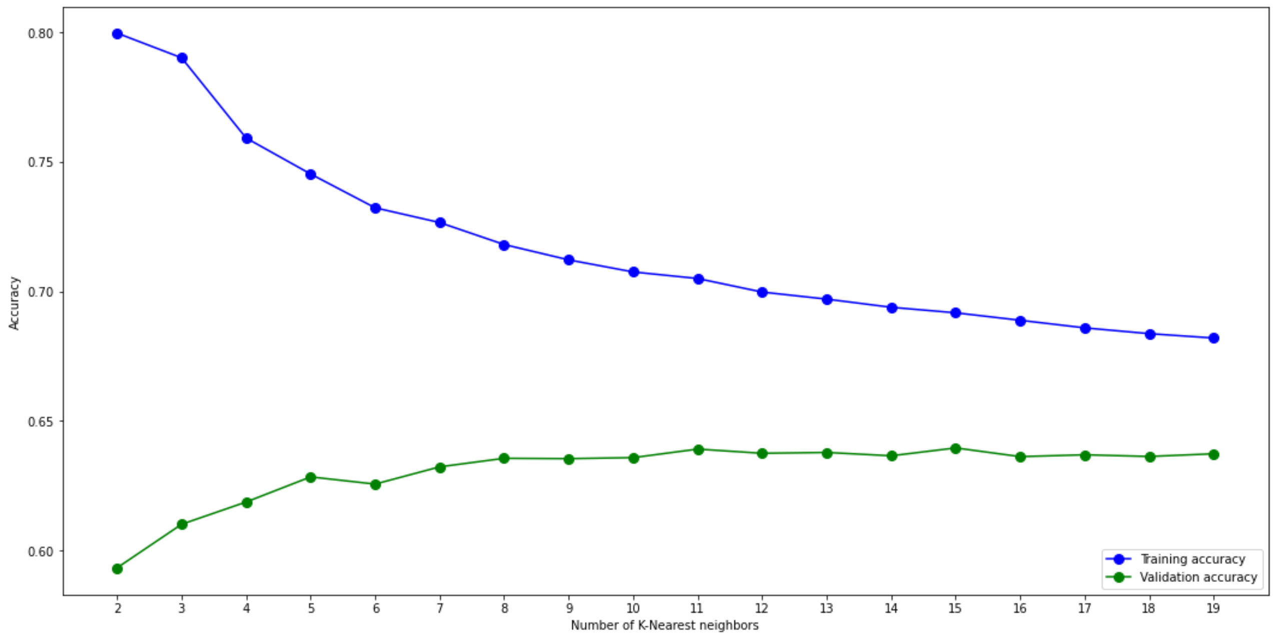
```
pipe = Pipeline([('scaler', StandardScaler()), ('knn', KNeighborsClassifier(n_neighbors = 13))])

#performing 10-fold cross validation
scores = cross_val_score(pipe, X=X_train, y=y_train.values.ravel(), cv=10)

#Displaying classification accuracy results
print(scores)
print("Mean score:", scores.mean())
print("Std score:", scores.std())
print("Accuracy: %0.3f +/- %0.3f "%(scores.mean(), scores.std()))
```

```
[0.63630491 0.6369509  0.67312661 0.62919897 0.64405685 0.63049096
 0.64341085 0.62403101 0.62144703 0.63953488]
Mean score: 0.6378552971576228
Std score: 0.013816438641131638
Accuracy: 0.638 +/- 0.014
```

1.8) Answer this question using the data in step 3. Perform 10-fold cross-validation while changing the number of nearest neighbors from 2 to 20 and then plotting the validation curve.

In [ ]:
```
#10-fold cross validation with no. of nearest neighbors from [2-20]
from sklearn.model_selection import validation_curve
param_range = range(2,20)
#pipeline_ = Pipeline([('scaler', StandardScaler()), ('knn', KNeighborsClassifier(n_neighbors=13))])
train_scores, test_scores = validation_curve(estimator=pipe, X=X_train, y=y_train.values.ravel(), cv=10, param_name='knn__n_neighbors', param
```

In [ ]:
```
#Plotting validation curve
train_mean = np.mean(train_scores, axis=1)
test_mean = np.mean(test_scores, axis=1)
plt.figure(figsize=(18, 9))
plt.plot(param_range, train_mean, color='blue', marker='o', markersize=8,
label='Training accuracy')
plt.plot(param_range, test_mean, color='green', marker='o', markersize=8,
label='Validation accuracy')
plt.xlabel('Number of K-Nearest neighbors')
plt.ylabel('Accuracy')
plt.xticks(range(2,20))
plt.legend(loc='lower right')
plt.show()
```



Problem 2

Using the Pima Indians Database to predict Diabetes Outcome

The Pima are Native Americans based in Arizona. As a result of changes in diet and physical activity, they have developed a very high incidence of Type 2 diabetes. The anonymous medical data used in this notebook was obtained from 768 Pima women. It comprises 8 attributes that might be used to predict diabetes status (the 9th column in the dataset, which is the class to be predicted). You can download this data as a csv (comma-separated variable) file using the python code given below

```
dataDir = "../data"
dataFile = dataDir + '/pima-indians-diabetes.csv'
import os.path
if not os.path.isfile(dataFile):
import requests # Remember: you may need to install the requests module: conda install -c anaconda requests \
url='https://gist.github.com/chaityacshah/899a95deaf8b1930003ae93944fd17d7/raw/3d35de839da708595a444187e9f13237b51a2cbe/pima-
indians-diabetes.csv'
r = requests.get(url)
with open(dataFile, 'wb') as f:
f.write(r.content)
seed = 42
```

2.1) Load the dataFile as a data frame and name it as pimaDf.

In [ ]:
```
import requests # Remember: you may need to install the requests module: conda i nstall -c anaconda requests \
import os

dataDir = "./data"
dataFile = dataDir + '/pima-indians-diabetes.csv'
# os.mkdir(dataDir)
if not os.path.isfile(dataFile):

    url='https://gist.githubusercontent.com/chaityacshah/899a95deaf8b1930003ae93944fd17d7/raw/3d35de839da708595a444187e9f13237b51a2cbe/pima-i
    r = requests.get(url)
    with open(dataFile, 'wb') as f:
        f.write(r.content)
# seed = 42
```

In [ ]:
```
#Loading the dataFile as a data frame and naming it as pimaDf
pimaDf = pd.read_csv("pima-indians-diabetes.csv")

#Gleaning at the dataset for the first few rows and the contained attributes
pimaDf.head()
```

Out[ ]:

	1. Number of times pregnant	2. Plasma glucose concentration a 2 hours in an oral glucose tolerance test	3. Diastolic blood pressure (mm Hg)	4. Triceps skin fold thickness (mm)	5. 2-Hour serum insulin (mu U/ml)	6. Body mass index (weight in kg/(height in m)^2)	7. Diabetes pedigree function	8. Age (years)	9. Class variable (0 or 1)
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

2.2) While the existing predictor names are descriptive, they are cumbersome when used in models. Use the names given below to replace them with simpler predictor names predNames and className .

predNames = NumPreg, PlasmaGlucose, DiastolicBP, SkinFold, SerumInsulin, BMI, PedFn, AgeYrs and

className = DiabetesClass

In [ ]:
```
#Changing attribute headings as per the question
pimaDf.columns = ['NumPreg', 'PlasmaGlucose', 'DiastolicBP', 'SkinFold', 'SerumInsulin', 'BMI', 'PedFn', 'AgeYrs', 'DiabetesClass']
pimaDf.head()
```

Out[ ]:

	NumPreg	PlasmaGlucose	DiastolicBP	SkinFold	SerumInsulin	BMI	PedFn	AgeYrs	DiabetesClass
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

2.3) Given the fact that there are so many predictors (8, though often the number of predictors), it is often advisable to look for:

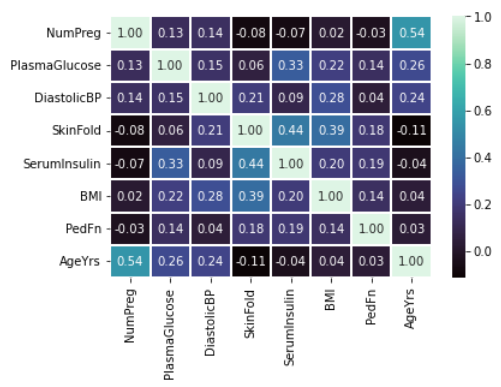
a) correlations between predictors and

b) correlations between predictors and the assigned class labels.

Compute the Pearson correlation coefficient between the predictors as well as the predictors and the lables. Briefly describe the correlation matrix.

In [ ]:
```
#Pearson correlation coefficient matrix between the predictors
corrpredictors_ = pimaDf.iloc[:,0:8].corr(method = "pearson")
sns.heatmap(corrpredictors_, annot=True, fmt=".2f",linewidth=1.5,cmap="mako")
```

Out[ ]:
```
<AxesSubplot:>
```



From the above pearson's correlation heatmap, we can conclude that is no strong linear correlation between the predictor attributes except for No. of pregnancies and age.

Other than these parameters,we can say that the following attribute pairs (to name a few) have slight linear-correlation:

1) PlasmaGlucose & SerumInsulin

2) PlasmaGlucose & BMI

3) PlasmaGlucose & Age

4) DiastolicBP & Age

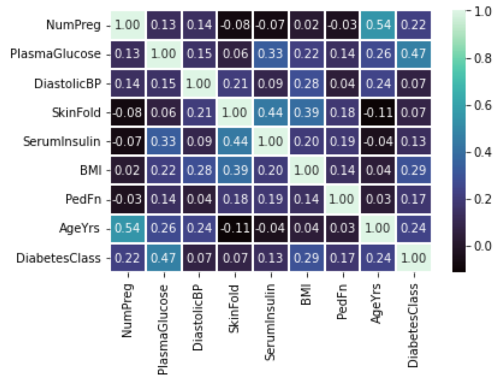
5) Skinfold & BMI

6) BMI & DiastolicBP

7) Skinfold & SerumInsulin

```
In [ ]: #Pearson correlation coefficien matrix between the predictors and the labels
        corr_labels = pimaDf.corr(method = "pearson")
        sns.heatmap(corr_labels, annot=True, fmt=".2f",linewidth=1.5,cmap="mako")
```

```
Out[ ]: <AxesSubplot:>
```

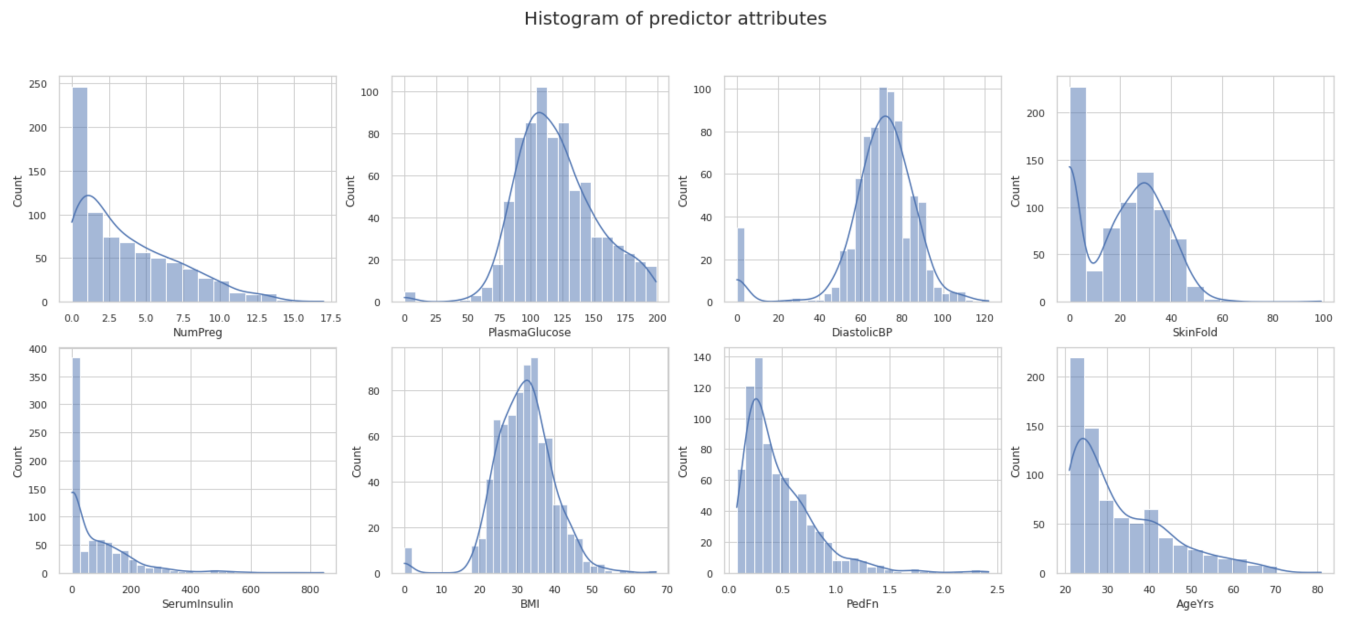


From the above pearson's correlation heatmap, we can conclude that is no strong linear correlation between the predictor attributes and DiabetesClass except for PlasmaGlucose whereas NumPreg, BMI and age have slight linear correlation with the Diabetes class lebel.

2.4) In order to see the general distribution for each column and also the effects of the missing values, generate histograms of the predictor values. What are the predictors that may contain missing?

```
In [ ]: #Plotting histograms for the predictor attributes
        fig, axes = plt.subplots(2,4, figsize=(25,10))
        sns.set(style="whitegrid")
        sns.histplot(x = pimaDf['NumPreg'], kde=True, ax=axes[0][0])
        sns.histplot(x = pimaDf['PlasmaGlucose'], kde=True, ax=axes[0][1])
        sns.histplot(x = pimaDf['DiastolicBP'], kde=True, ax=axes[0][2])
        sns.histplot(x = pimaDf['SkinFold'], kde=True, ax=axes[0][3])
        sns.histplot(x = pimaDf['SerumInsulin'], kde=True, ax=axes[1][0])
        sns.histplot(x = pimaDf['BMI'], kde=True, ax=axes[1][1])
        sns.histplot(x = pimaDf['PedFn'], kde=True, ax=axes[1][2])
        sns.histplot(x = pimaDf['AgeYrs'], kde=True, ax=axes[1][3])
        fig.suptitle('Histogram of predictor attributes', size=20)
```

```
Out[ ]: Text(0.5, 0.98, 'Histogram of predictor attributes')
```



We can see from the plots that missing values cause distortion to the shape of the graph and increase the skewness of the distribution. Further, we have performed some analysis to check for the exact null or missing values present in the attributes.

In [ ]:
```
# Counting the number of nulls in each column
pimaDf.isna().sum()
```

Out[ ]:
```
NumPreg          0
PlasmaGlucose    0
DiastolicBP      0
SkinFold         0
SerumInsulin     0
BMI              0
PedFn            0
AgeYrs           0
DiabetesClass    0
dtype: int64
```

We can see that there are no null values present in the attributes, so we will further check for the no. of zeroes present in the attributes.

In [ ]:
```
# Counting the number of zeros in each column
for clm_name in pimaDf.columns:
    column = pimaDf[clm_name]
    # Get the count of Zeros in column
    count = (column == 0).sum()
    print('No. of zeros in column', clm_name, ' is : ', count)
```

```
No. of zeros in column NumPreg  is :  111
No. of zeros in column PlasmaGlucose  is :  5
No. of zeros in column DiastolicBP  is :  35
No. of zeros in column SkinFold  is :  227
No. of zeros in column SerumInsulin  is :  374
No. of zeros in column BMI  is :  11
No. of zeros in column PedFn  is :  0
No. of zeros in column AgeYrs  is :  0
No. of zeros in column DiabetesClass  is :  500
```

From the above outputs we can observe that, except for prediction function for diabetes and age, all other predictor attributes have certain no. of zeroes which can be considered as missing values.

However, the no. of zeroes for the attribute "No. of pregnancies" cannot be considered as a missing value as it is plausible for a person to have no pregnancies, i.e having zero values in the NumPreg attributes.

So, we can conclude that the following attributes have the possible missing values: PlasmaGlucose, DiastolicBP, SkinFold, SerumInsulin, BMI.

2.5) A way of accommodating missing data is to impute values are statistically "neutral". Do this by assigning, when the predictor value is zero, the median of the remaining values of a predictor. Name the updated database as filteredPimaDf

In [ ]:
```
#Replacing attribute values with the median of those columns where value = 0.
filteredPimaDf = pimaDf.copy()
for col in pimaDf.columns:
    filteredPimaDf[col] = filteredPimaDf[col].replace(0,filteredPimaDf[col].median())
```

In [ ]:
```
filteredPimaDf.DiabetesClass.value_counts()
```

Out[ ]:
```
0    500
1    268
Name: DiabetesClass, dtype: int64
```

2.6) Using the filteredPimaDf, create training and testing datasets by spliting 768 rows into train=514 and test=254 rows. Then, apply _MiniMax_scaler on the predictor metrices

In [ ]:
```
#Creating training and testing datasets by splitting 768 rows into train=514, test=254 using filteredPimaDf
X,y = filteredPimaDf.iloc[:,0:8], filteredPimaDf.iloc[:,8:]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 254, random_state = 45)
```

In [ ]:
```
#Applying MinMaxScaler on the predictor metrices
from sklearn.preprocessing import MinMaxScaler

X_train_scaler = MinMaxScaler()
X_train_scaler.fit(X_train)
X_train_scaled = X_train_scaler.transform(X_train)

X_test_scaler = MinMaxScaler()
X_test_scaler.fit(X_test)
X_test_scaled = X_test_scaler.transform(X_test)
```

In [ ]:
```
df_t_train = pd.DataFrame(X_train_scaled,columns = filteredPimaDf.columns[:-1])
df_t_test = pd.DataFrame(X_test_scaled,columns = filteredPimaDf.columns[:-1])
```

2.7) Create a Naive Bayes Classifier (GaussianNB) and show the classifcation diagnostics (e.g., accuracy (=

0.75984), Confusion Matrix, Classification Report).

In [ ]:
```
#Creating Naive Bayes Classifier (GaussianNB)
from sklearn.naive_bayes import GaussianNB
gaussNB_ = GaussianNB()
gaussNBmodel_ = gaussNB_.fit(df_t_train, y_train.values.ravel())
```

In [ ]:
```
#Printing accuracies of training and test set
print("training set score: %f" % gaussNB_.score(df_t_train, y_train))
print("test set score: %f" % gaussNB_.score(df_t_test, y_test))
```

```
training set score: 0.754864
test set score: 0.720472
```

In [ ]:
```
#Printing confusion Matrix
y_pred = gaussNBmodel_.predict(df_t_test);
cmatrix_ = confusion_matrix(y_test, y_pred)
```

```
print("Confusion Matrix:")
print(cmatrix_)
```

```
Confusion Matrix:
[[130  36]
 [ 35  53]]
```

In [ ]:
```
#Printing classification report and accuracy of the classifier
creport_ = classification_report(y_test, y_pred)
print("Classification Report:",)
print (creport_)
from sklearn import metrics
print("Accuracy of the classifier:",metrics.accuracy_score(y_test,y_pred))
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.79      0.78      0.79       166
           1       0.60      0.60      0.60        88

    accuracy                           0.72       254
   macro avg       0.69      0.69      0.69       254
weighted avg       0.72      0.72      0.72       254

Accuracy of the classifier: 0.7204724409448819
```

2.8) Create a support vector machine (SVM) classifier (set the hyperparameter gamma to auto) and show the

classifcation diagnostics (e.g., accuracy (= 0.7677), Confusion Matrix, Classification Report).

In [ ]:
```
#Creating SVM with gamma='auto'
from sklearn import svm
clf = svm.SVC(gamma="auto")
svm_model = clf.fit(df_t_train,y_train.values.ravel())
```

In [ ]:
```
y_pred = svm_model.predict(df_t_test);
```

In [ ]:
```
#Printing accuracies of training and test set
print("training set score: %.2f" % (svm_model.score(df_t_train, y_train)))
print("test set score: %.2f" % (svm_model.score(df_t_test, y_test)))
```

```
training set score: 0.78
test set score: 0.75
```

In [ ]:
```
#Printing confusion matrix
y_pred = svm_model.predict(df_t_test);
cmatrix__ = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:\n", cmatrix__)
```

```
Confusion Matrix:
 [[148  18]
 [ 45  43]]
```

In [ ]:
```
#Printing classification report and accuracy of the classifier
creport__ = classification_report(y_test, y_pred)
print("Classification Report:\n",creport__)
from sklearn import metrics
print("Accuracy:",metrics.accuracy_score(y_test,y_pred))
```

```
Classification Report:
               precision    recall  f1-score   support

           0       0.77      0.89      0.82       166
           1       0.70      0.49      0.58        88

    accuracy                           0.75       254
   macro avg       0.74      0.69      0.70       254
weighted avg       0.75      0.75      0.74       254

Accuracy: 0.7519685039370079
```

Problem-3

This question is intended to practice Hierarchical clustering and K-Means by using the clusterdata.csv file. The dataset should be loaded as a data frame and named df. Before clustering can be performed, the dataset must be processed. That is because it appears that there are two variables, but they are stored under a single variable name Var1 separated by a comma. As an example, df.head(3) outputs

3.1) Separate two values in df into two columns and name them Var1 and Var2 in a new data frame named df_new. If you run df_new.head(3) , you should see the following output:

In [ ]:
```
import pandas as pd
df = pd.read_csv("clusterdata.csv")
df.head(3)
```

Out[ ]:

	Var1
0	-2.9191,0.32036
1	-5.4637,-0.87935
2	-3.0553,0.42799

In [ ]:
```
# Creating a new dataset with two columns
df_new = pd.DataFrame()
df_new['Var1'] = pd.to_numeric(df['Var1'].apply(lambda x: x.split(',')[0]))
```

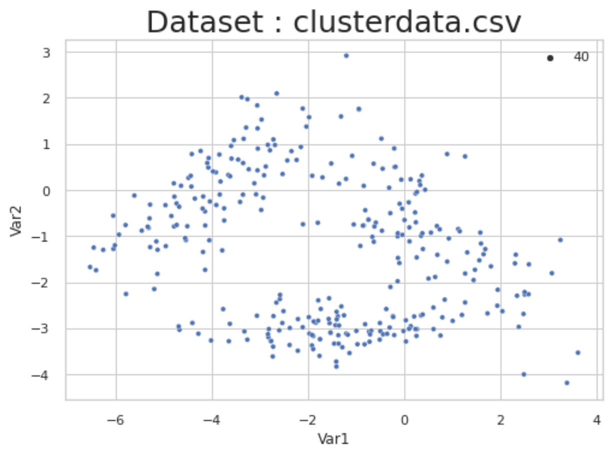
```
df_new['Var2'] = pd.to_numeric(df['Var1'].apply(lambda x: x.split(',')[1]))
df_new.head(3)
```

Out[ ]:

	Var1	Var2
0	-2.9191	0.32036
1	-5.4637	-0.87935
2	-3.0553	0.42799

3.2) Create a scatter plot Var1 vs Var2.

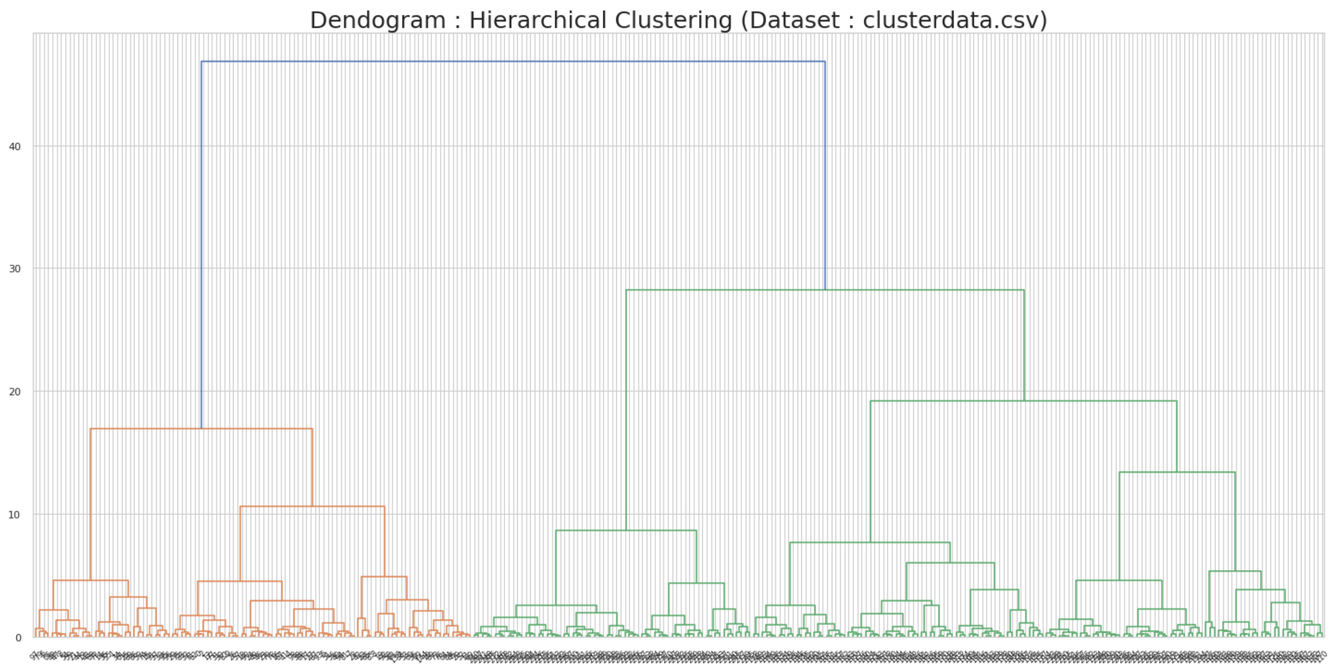
In [ ]:
```
# Scatter plot Var1 vs Var2
txt = sns.scatterplot(data = df_new, x = 'Var1', y = 'Var2')
txt = plt.title('Dataset : clusterdata.csv', fontsize = 25)
```



3.3) Using the df_new, perform hierarchical clustering using the linkage method with the options

method='ward' and metric='euclidean' . To represent the clustering process, create a dendogram.

In [ ]:
```
# Hierarchical clustering and representation with dendogram
import scipy.cluster.hierarchy as shc

plt.figure(figsize = (25,12))
clusters = shc.linkage(df_new, method='ward', metric="euclidean")
shc.dendrogram(Z=clusters)
plt.xticks(rotation=45, fontsize=8)
plt.title("Dendogram : Hierarchical Clustering (Dataset : clusterdata.csv)", fontsize = 25)
plt.show()
```



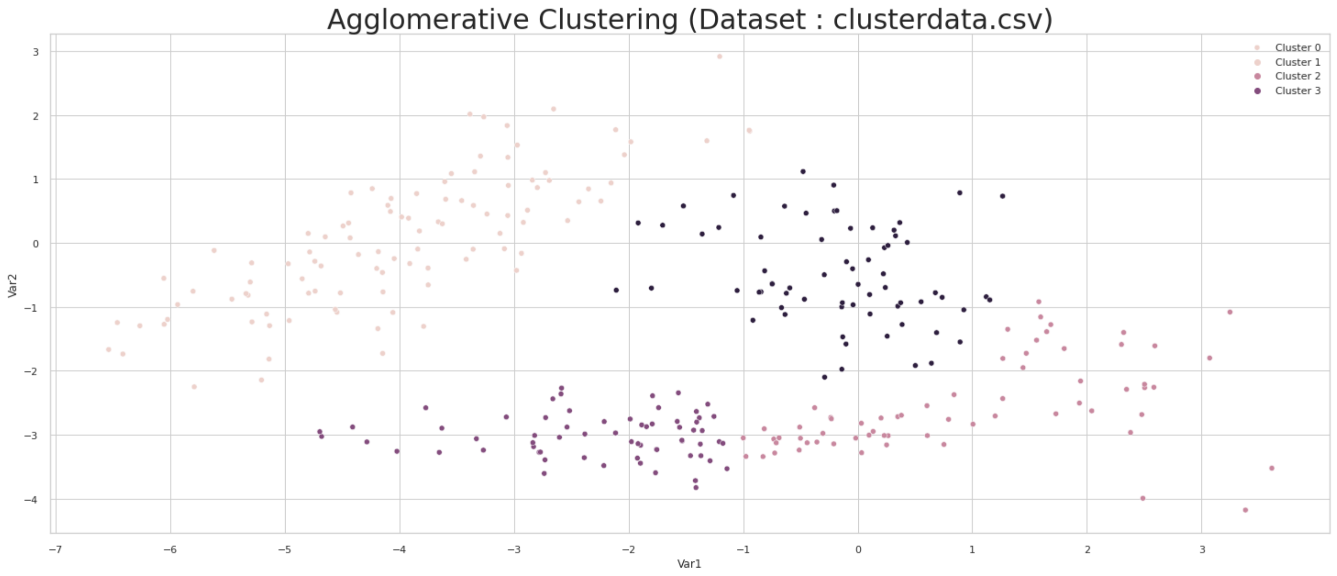
3.4) Perform Agglomerative Clustering using the following settings: n_clusters = 4, affinity =

'euclidean', linkage = 'ward' . Make a scatter plot of the four clusters you created (you should use

different colors).

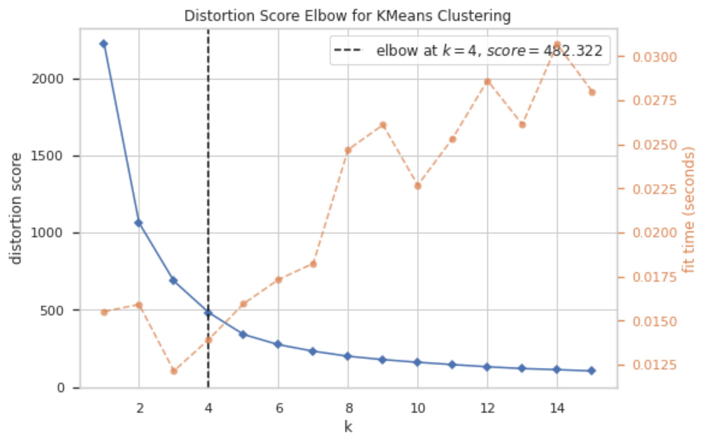
In [ ]:
```
# Agglomerative Clustering
```

```
from sklearn.cluster import AgglomerativeClustering

clustering_model = AgglomerativeClustering(n_clusters=4, affinity='euclidean', linkage='ward')
clustering_model.fit(df_new)
txt = plt.figure(figsize = (25,10))
txt = sns.scatterplot(data = df_new, x = 'Var1', y = 'Var2', hue = clustering_model.labels_)
txt = plt.title("Agglomerative Clustering (Dataset : clusterdata.csv)", fontsize = 30)
plt.xticks(np.arange(-8, 4, step=1))
plt.legend(["Cluster 0","Cluster 1", "Cluster 2", "Cluster 3" ])
txt = plt.plot()

# plt.scatter(clustering_model.cluster_centers_[:,0], clustering_model.cluster_centers_[:,1],
#             marker="X", c="r", s=80, label="centroids")
```



3.5) Use the df_new to perform K-Means clustering (set random_state = 0). Identify the optimal number of clusters as four. Based on the optimal number of clusters, calculate the clustering parameters cluster centroids, the sum of the squares within each cluster, Cluster labels, and Cluster size.

```
In [ ]: # K Means clustering
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
from yellowbrick.cluster import KElbowVisualizer
from yellowbrick.cluster import SilhouetteVisualizer


scores = []

for n_cluster in range(2,15):
    kmeans = KMeans(n_clusters=n_cluster, random_state=0).fit(df_new)
    scores.append(kmeans.inertia_)

visualizer = KElbowVisualizer(kmeans, k=(1,16)).fit(df_new)
txt = visualizer.show()
```



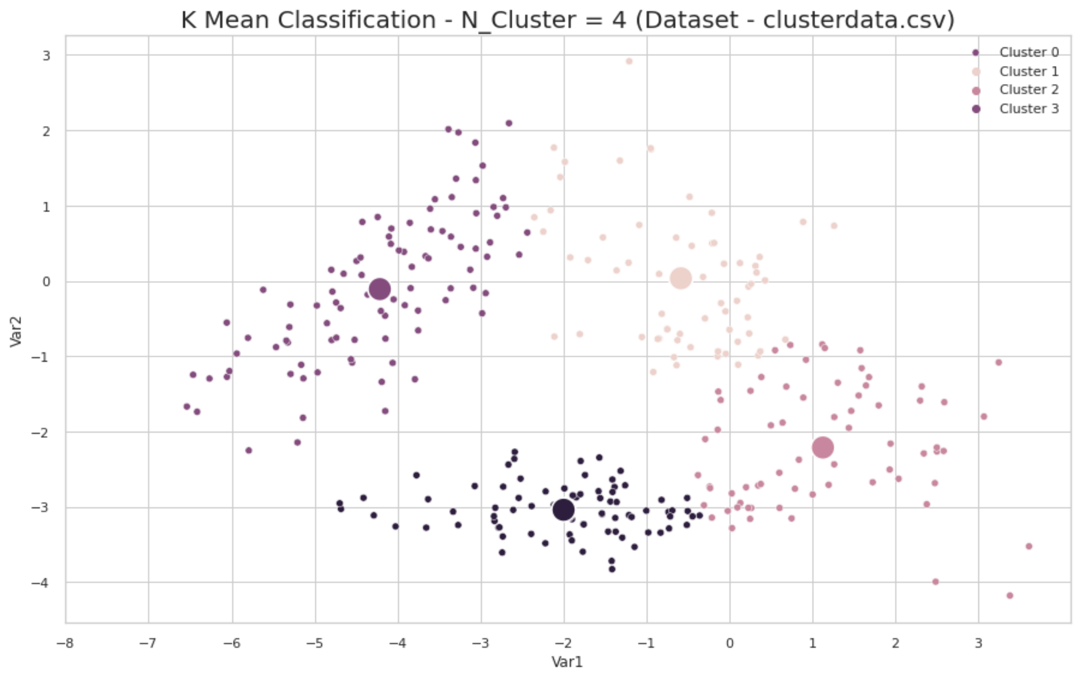
From the above plot, the elbow can be identied as at 4. That is, the optimal number of clusters is 4.

```
In [ ]: # Optimal K Mean Classification and calculating cluster parameters
optimal_n_cluster = 4
kmeans = KMeans(n_clusters=optimal_n_cluster, random_state=0).fit(df_new)

print("K-Means Clustering Results : \n")
print("Sum of squares within each Cluster (Inertia) : ",kmeans.inertia_)
print("Cluster Centroid : \n",kmeans.cluster_centers_)
print("Cluster Labels : \n",kmeans.labels_)
print("Cluster Size : ")
vals, count = np.unique(kmeans.labels_, return_counts = True)
for i in range(len(vals)):
    print("Cluster ",vals[i], "\t Size :", count[i])
```

```
K-Means Clustering Results :

Sum of squares within each Cluster (Inertia) :  482.3223358872427
Cluster Centroid :
 [[-0.5823526   0.03485511]
 [ 1.13031173 -2.21168224]
 [-4.21242174 -0.10889622]
 [-1.99659718 -3.03798974]]
Cluster Labels :
 [2 2 2 0 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 0 2 2 0 2 2 2 2 2 2 2 2 2 2 0 2 2 2
 2 2 2 2 2 0 2 2 2 2 2 2 2 2 2 2 2 0 2 2 2 2 2 2 2 2 2 2 2 2 0 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 0 2 2 2 2 2 2 0 2 2 2 2 2 2 2 1 0 1 1 0 0 0 1 0 0 1
 0 0 1 0 1 0 0 0 1 0 0 1 0 1 1 1 0 0 0 0 0 1 1 1 1 1 1 0 0 1 0 0 0 2 1 0 0
 0 0 0 1 1 0 0 1 1 0 1 0 1 1 1 1 0 1 0 1 1 0 0 1 1 0 0 0 0 0 0 1 0 0 1 1 0
 0 0 1 1 0 1 0 1 0 0 0 1 1 1 1 3 3 1 3 3 3 3 1 3 3 3 3 3 3 3 3 3 1 3 1 3 3
 3 3 3 1 3 1 3 3 3 3 3 3 1 3 1 3 3 1 3 3 1 3 3 3 3 3 3 1 3 3 1 3 3 3 3 3 3
 1 3 3 3 3 3 1 1 3 3 3 3 1 3 3 3 3 3 3 3 3 1 1 1 1 1 3 3 3 3 3 3 3 3 3 1 3
 3 3 3 3]
Cluster Size :
Cluster  0       Size : 63
Cluster  1       Size : 67
Cluster  2       Size : 92
Cluster  3       Size : 78
```

3.6) Using the information in step 5, create a scatter plot that shows clusters and their centroids.

In [ ]:
```
# Scatter plot post K Means Classification
txt = plt.figure(figsize = (15,9))
txt = sns.scatterplot(data = df_new, x = 'Var1', y = 'Var2', hue = kmeans.labels_)
txt = sns.scatterplot(x = kmeans.cluster_centers_[:,0] , y = kmeans.cluster_centers_[:,1], marker = 'o',  s=400, hue = [0,1,2,3])
plt.xticks(np.arange(-8, 4, step=1))
plt.legend(["Cluster 0","Cluster 1", "Cluster 2", "Cluster 3" ])
txt = plt.title("K Mean Classification - N_Cluster = 4 (Dataset - clusterdata.csv)", fontsize = 20)
```



3.7) You should identify a suitable technique that will reduce the inertia value observed in step 5 to

approximately 7 (prove this by performing clustering).

In [ ]:
```
# Technique to reduce inertia
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler(feature_range=(0, 1))
scaler.fit(df_new)
df_scaled = scaler.transform(df_new)
```

In [ ]:
```
optimal_n_cluster = 4
kmeans = KMeans(n_clusters=optimal_n_cluster, random_state=0).fit(df_scaled)
print("K-Means Clustering Result : ")
print("Inertia  : ",kmeans.inertia_)

print("Hence proved that by using appropriate scaling, we can reduce inertia score to less than 7")
```

```
K-Means Clustering Result :
Inertia  :  6.4750666075581575
Hence proved that by using appropriate scaling, we can reduce inertia score to less than 7
```

Problem 4

The intention of this question is to practice K-Mean clustering with a higher dimensional data. The dataset,

Live.csv, contains information about a set of different reactions (e.g., number of comments, likes, shares, etc)

on a public posts made by a group of students.

4.1) Load the dataset as a data frame df and inspect the dataset to identify unncecessery data and remove them.

In [ ]:
```
#Loading dataset
df = pd.read_csv('Live.csv')
df.head()
```

Out[ ]:

	status_id	status_type	status_published	num_reactions	num_comments	num_shares	num_likes	num_loves	num_wows	num_hahas	num_sads	nu
0	246675545449582_1649696485147474	video	4/22/2018 6:00	529	512	262	432	92	3	1	1	
1	246675545449582_1649426988507757	photo	4/21/2018 22:45	150	0	0	150	0	0	0	0	
2	246675545449582_1648730588577397	video	4/21/2018 6:17	227	236	57	204	21	1	1	0	
3	246675545449582_1648576705259452	photo	4/21/2018 2:29	111	0	0	111	0	0	0	0	
4	246675545449582_1645700502213739	photo	4/18/2018 3:22	213	0	0	204	9	0	0	0	

In [ ]:
```
#Removing unnecessary data.
print(df.info())
print("Column1, Column2, Column3 and Column are completely blank. These should be removed")
# Remove Blank Columns
df.drop(['Column1','Column2','Column3','Column4'], axis = 1, inplace = True)
df.head()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7050 entries, 0 to 7049
Data columns (total 16 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   status_id         7050 non-null   object 
 1   status_type       7050 non-null   object 
 2   status_published  7050 non-null   object 
 3   num_reactions     7050 non-null   int64  
 4   num_comments      7050 non-null   int64  
 5   num_shares        7050 non-null   int64  
 6   num_likes         7050 non-null   int64  
 7   num_loves         7050 non-null   int64  
 8   num_wows          7050 non-null   int64  
 9   num_hahas         7050 non-null   int64  
 10  num_sads          7050 non-null   int64  
 11  num_angrys        7050 non-null   int64  
 12  Column1           0 non-null      float64
 13  Column2           0 non-null      float64
 14  Column3           0 non-null      float64
 15  Column4           0 non-null      float64
dtypes: float64(4), int64(9), object(3)
memory usage: 881.4+ KB
None
Column1, Column2, Column3 and Column are completely blank. These should be removed
```

Out[ ]:

	status_id	status_type	status_published	num_reactions	num_comments	num_shares	num_likes	num_loves	num_wows	num_hahas	num_sads	nu
0	246675545449582_1649696485147474	video	4/22/2018 6:00	529	512	262	432	92	3	1	1	
1	246675545449582_1649426988507757	photo	4/21/2018 22:45	150	0	0	150	0	0	0	0	
2	246675545449582_1648730588577397	video	4/21/2018 6:17	227	236	57	204	21	1	1	0	
3	246675545449582_1648576705259452	photo	4/21/2018 2:29	111	0	0	111	0	0	0	0	
4	246675545449582_1645700502213739	photo	4/18/2018 3:22	213	0	0	204	9	0	0	0	

- From the above data observation, we can infer that 'Column 1' to 'Column4' named data has Nan values and they need to dropped as they do not contribute to further data processing or analysis, even Nan in those particular columns cannot be replaced with the influence or combinations of other columns. So it is better to drop such columns to reduce the complexity and dimensionality of the data.

4.2) Consider the three features, status_id, status_published, and status_type. Among these three features, identify the features that do not share certain type of common property (support your answer with sufficient evidence) with the remaining features of the dataset.

Answer: All three are different from other features based on their datatype. While the remaining features are numeric,

- Status_ID is a unique identifier with String data type - which has very low probability of being a useful feature.
- Status_type is a categorical variable which says if the status was a Photo or Video
- Status_published is a timestamp feature

However the main difference is that : Status_ID and Status_published are features that are mostly used for administrative purpose and unique identification of data points. While Status_type is a feature that is a characteristic/behaviour of a paritcular status - which is similar to the remaining features of the dataset.

4.3) Based on your inspection in step 2, declare the feature vector X by removing the features that do share commom property. Also, define a target variable Y as the featre that shares the common property with the remaining data.

In [ ]:
```
from sklearn import preprocessing
le = preprocessing.LabelEncoder()
```

```
# Create feature vector X
X = df.drop(['status_id','status_published'], axis = 1)
X['status_type'] = le.fit_transform(X['status_type'])
# Target Variable
Y = df['status_type'].astype('category')
```

The Status type can be considered as the target variable because using all other features - reactions, likes, comments etc., we can try to predict if the status published is a photo or a comment.

4.4) Apply MiniMaxScaler on the feature vector X and then perform K-Mean clustering with 2 clusters (set n_clusters = 2, random_state = 0)

In [ ]:
```
#Applying MinMaxScaler on the feature vector X
from sklearn.preprocessing import MinMaxScaler
from sklearn.cluster import KMeans

# Scaling using MinMax
scaler = MinMaxScaler(feature_range=(0, 1))
scaler.fit(X)
X = scaler.transform(X)

# K Means classification
kmeans = KMeans(n_clusters=2, random_state=0).fit(X)
```

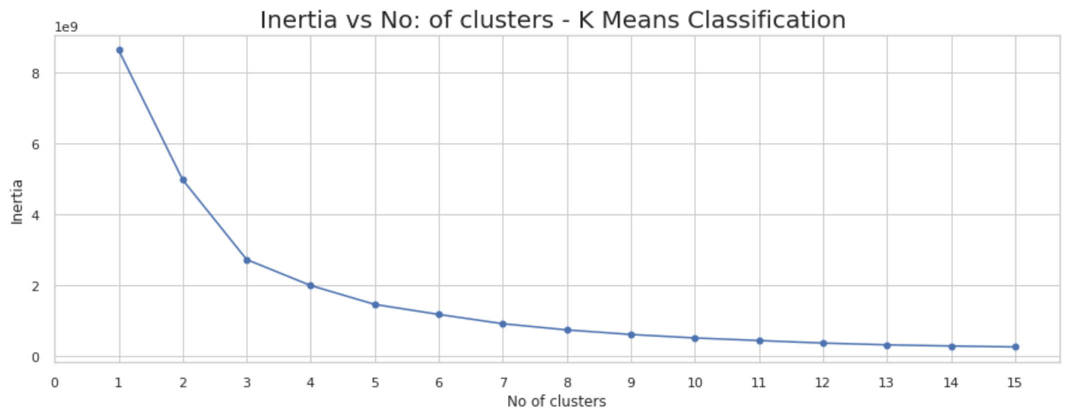
4.5) Show that the Inertia = 237.757. Also, print Cluster centroids, cluster labels, and cluster size.

In [ ]:
```
#Computing internia value. Printing cluster centroids, cluster labels, and cluster size.
print("K-Means Clustering Result : \n")
print("Inertia : ",kmeans.inertia_)
print("Centroid for Cluster 0 : ", kmeans.cluster_centers_[0])
print("Centroid for Cluster 1 : ", kmeans.cluster_centers_[1])
print("Cluster Labels : \n",kmeans.labels_)
vals, count = np.unique(kmeans.labels_, return_counts = True)
for i in range(len(vals)):
    print("Cluster ",vals[i], "\t Size :", count[i])
```

```
K-Means Clustering Result :

Inertia :  237.7572640441955
Centroid for Cluster 0 :  [0.32850686 0.03907109 0.00075485 0.00075367 0.03854389 0.00217449
 0.00243721 0.0012004  0.00275348 0.00145313]
Centroid for Cluster 1 :  [0.95492158 0.06463304 0.02670287 0.02931717 0.05712315 0.04710071
 0.00818582 0.00965208 0.00804219 0.00719502]
Cluster Labels :
 [1 0 1 ... 0 0 0]
Cluster  0       Size : 4351
Cluster  1       Size : 2699
```

4.6) Compute inertia for different number of clusters (i.e., $k = 1, 2, 3, \cdots, 15$) and then plot inertia vs number of clusters. Use your plot show that the optimal number of clusters is approximately 3.

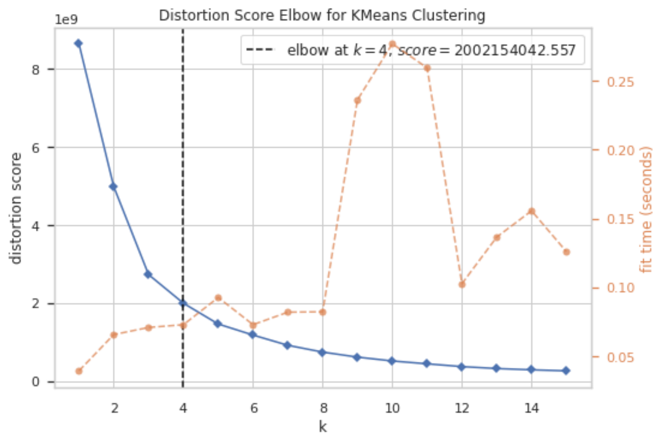
In [ ]:
```
#Computing inertia for different number of clusters
inertias = []

for n_cluster in range(1,16):

    kmeans = KMeans(n_clusters=n_cluster, random_state=0).fit(X)
    inertias.append(kmeans.inertia_)

#Plotting inertia vs no. of clusters
plt.figure(figsize = (15,5))
plt.plot(range(1,16), inertias, marker = 'o')
plt.xlabel('No of clusters')
plt.ylabel('Inertia')
plt.title("Inertia vs No: of clusters - K Means Classification ", fontsize = 20)
plt.xticks(np.arange(0, 16, step=1))
plt.show()
```



From the above above graph, we can observe that as the number of clusters cross beyond 3, there is only a slight decrease in inertia values. Hence, we can conclude that the elbow is present at 3. That is, the optimal no of clusters=3.

4.7) Use the KElbowVisualizer function to find the optimal number of clusters.

```
In [ ]: from sklearn.cluster import KMeans
        from sklearn.datasets import make_blobs

        from yellowbrick.cluster import KElbowVisualizer

        model = KMeans(random_state = 0)
        visualizer = KElbowVisualizer(model, k=(1,16))

        txt = visualizer.fit(X)        # Fit the data to the visualizer
        txt = visualizer.show()        # Finalize and render the figure
```



From the above plot, we can see that the optimal no. of clusters is 3.

4.8) Fit K-Mean clustering method with the optimal number of clusters you selected in step 6 or 7 and show that the interia is reduced to 161.596.

```
In [ ]: from sklearn.cluster import KMeans

        kmeans = KMeans(n_clusters=3, random_state=0).fit(X)
        print("K-Means Clustering Result : ")
        print("Inertia at N_cluster=3 : ",kmeans.inertia_)
```

```
K-Means Clustering Result : 
Inertia at N_cluster=3 :  161.59633400033613
```

From the above, we can observe that the Inertia has been reduced to 161.596