MRIDUL HARISH, CED18I034, END SEMESTER EXAMINATION

In [ ]:

```
conda install -c conda-forge/label/gcc7 missingno
```

```
Collecting package metadata (current_repodata.json): ...working... done
Solving environment: ...working... done

# All requested packages already installed.


Note: you may need to restart the kernel to use updated packages.
```

In [ ]:

```
pip install plotly
```

```
Requirement already satisfied: plotly in c:\users\hp\anaconda3\lib\site-packages (5.
7.0)
Requirement already satisfied: six in c:\users\hp\anaconda3\lib\site-packages (from p
lotly) (1.15.0)
Requirement already satisfied: tenacity>=6.2.0 in c:\users\hp\anaconda3\lib\site-pack
ages (from plotly) (8.0.1)
Note: you may need to restart the kernel to use updated packages.
```

In [ ]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import missingno as msno
import plotly.express as px
import plotly.graph_objects as go


%matplotlib inline

import warnings
warnings.filterwarnings("ignore")
```

In [ ]:

```
main_df=pd.read_csv("dermatologyCSV.csv")
main_df.head()
```

Out[ ]:

	Index	erythema	scaling	definite borders	itching	koebner phenomenon	polygonal papules	follicular papules	oral mucosal involvement	k invo
0	1	2	2	0	3	0	0	0	0	
1	2	3	3	3	2	1	0	0	0	
2	3	2	1	2	3	1	3	0	3	
3	4	2	2	2	0	0	0	0	0	
4	5	2	3	2	2	2	2	0	2	

5 rows × 36 columns

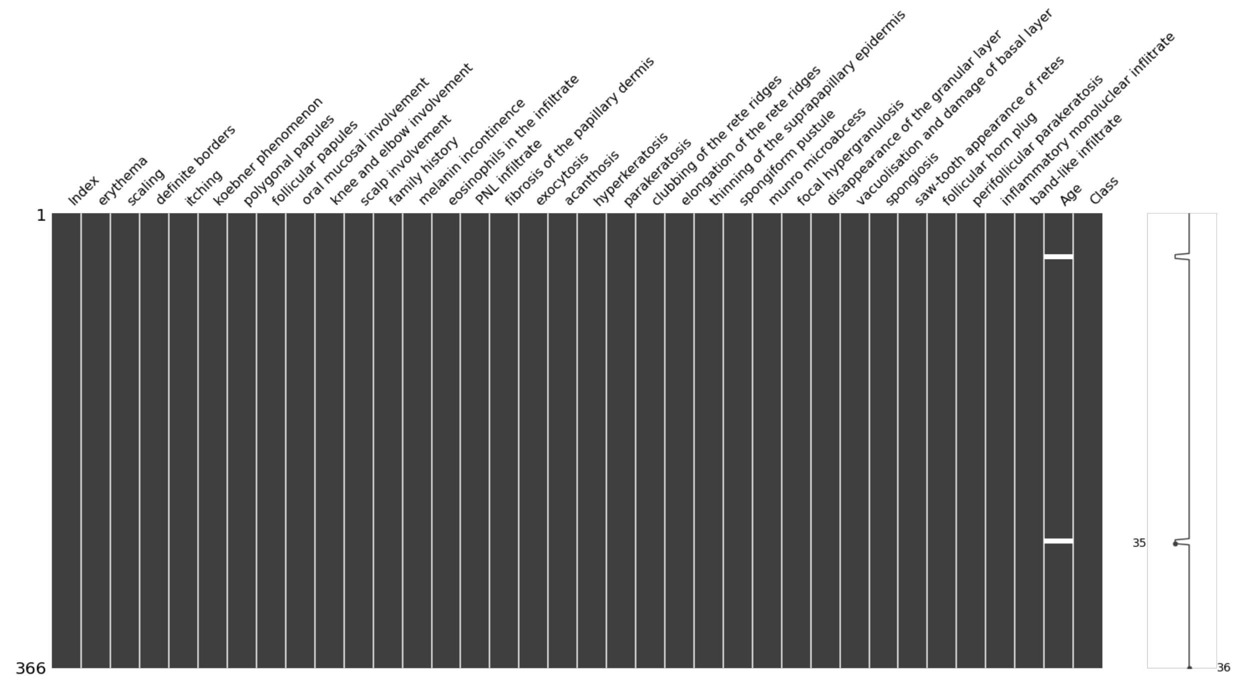
In [ ]:
```
main_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 366 entries, 0 to 365
Data columns (total 36 columns):
 #   Column                                    Non-Null Count  Dtype  
---  ------                                    --------------  -----  
 0   Index                                     366 non-null    int64  
 1   erythema                                  366 non-null    int64  
 2   scaling                                   366 non-null    int64  
 3   definite borders                          366 non-null    int64  
 4   itching                                   366 non-null    int64  
 5   koebner phenomenon                        366 non-null    int64  
 6   polygonal papules                         366 non-null    int64  
 7   follicular papules                        366 non-null    int64  
 8   oral mucosal involvement                  366 non-null    int64  
 9   knee and elbow involvement                366 non-null    int64  
 10  scalp involvement                         366 non-null    int64  
 11  family history                            366 non-null    int64  
 12  melanin incontinence                      366 non-null    int64  
 13  eosinophils in the infiltrate             366 non-null    int64  
 14  PNL infiltrate                            366 non-null    int64  
 15  fibrosis of the papillary dermis          366 non-null    int64  
 16  exocytosis                                366 non-null    int64  
 17  acanthosis                                366 non-null    int64  
 18  hyperkeratosis                            366 non-null    int64  
 19  parakeratosis                             366 non-null    int64  
 20  clubbing of the rete ridges               366 non-null    int64  
 21  elongation of the rete ridges             366 non-null    int64  
 22  thinning of the suprapapillary epidermis  366 non-null    int64  
 23  spongiform pustule                        366 non-null    int64  
 24  munro microabcess                         366 non-null    int64  
 25  focal hypergranulosis                     366 non-null    int64  
 26  disappearance of the granular layer       366 non-null    int64  
 27  vacuolisation and damage of basal layer   366 non-null    int64  
 28  spongiosis                                366 non-null    int64  
 29  saw-tooth appearance of retes             366 non-null    int64  
 30  follicular horn plug                      366 non-null    int64  
 31  perifollicular parakeratosis              366 non-null    int64  
 32  inflammatory monoluclear inflitrate       366 non-null    int64  
 33  band-like infiltrate                      366 non-null    int64  
 34  Age                                       358 non-null    float64
 35  Class                                     366 non-null    int64  
dtypes: float64(1), int64(35)
memory usage: 103.1 KB
```

CLEANING OF DATA

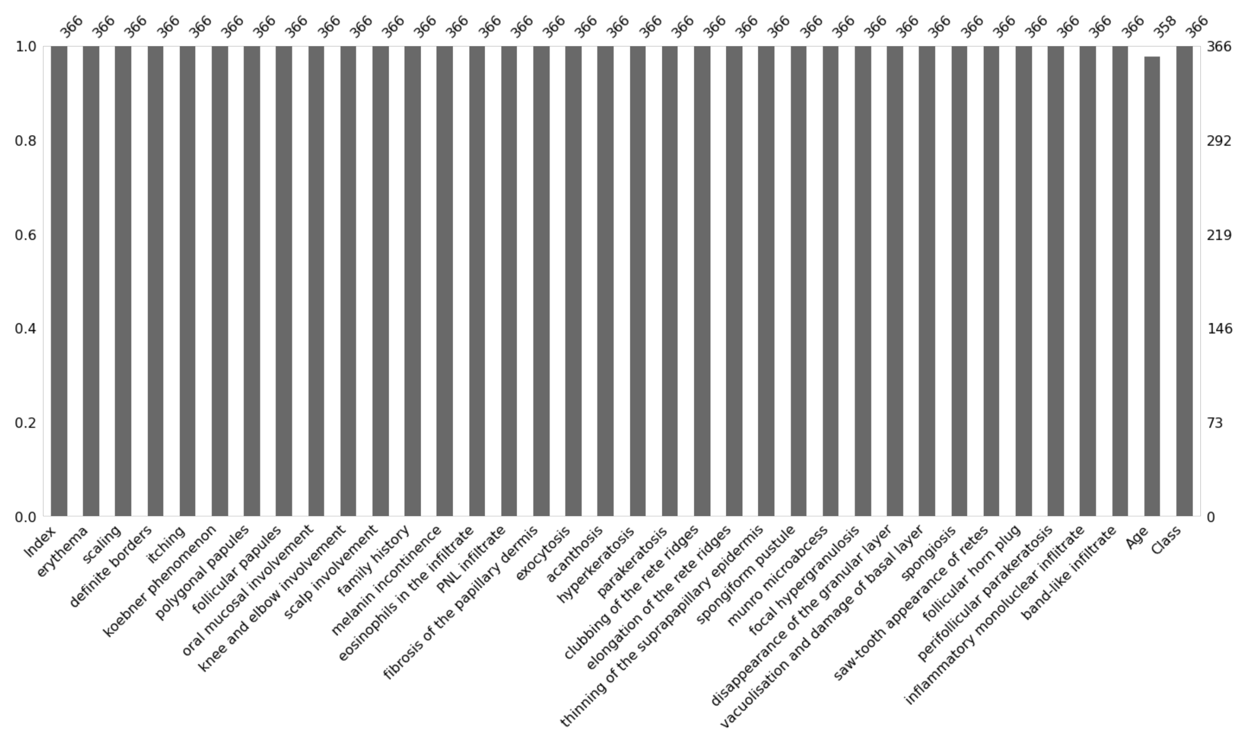
Finding the distribution of missing values in the data set

In [ ]:
```
msno.matrix(main_df);
```



Finding the number of data values in each coloumn and comparing them using a bar graph

In [ ]:
```
msno.bar(main_df);
```



Finding the number of unique values for each attribute

In [ ]:
```
main_df.nunique()
```

Out[ ]:
```
Index                    366
erythema                   4
scaling                    4
definite borders           4
```

```
itching                                    4
koebner phenomenon                         4
polygonal papules                          4
follicular papules                         4
oral mucosal involvement                   4
knee and elbow involvement                 4
scalp involvement                          4
family history                             2
melanin incontinence                       4
eosinophils in the infiltrate              3
PNL infiltrate                             4
fibrosis of the papillary dermis           4
exocytosis                                 4
acanthosis                                 4
hyperkeratosis                             4
parakeratosis                              4
clubbing of the rete ridges                4
elongation of the rete ridges              4
thinning of the suprapapillary epidermis   4
spongiform pustule                         4
munro microabcess                          4
focal hypergranulosis                      4
disappearance of the granular layer        4
vacuolisation and damage of basal layer    4
spongiosis                                 4
saw-tooth appearance of retes              4
follicular horn plug                       4
perifollicular parakeratosis               4
inflammatory monoluclear inflitrate        4
band-like infiltrate                       4
Age                                       60
Class                                      6
dtype: int64
```

Finding out the number of missing values in each attribute

In [ ]:

```
main_df.isna().sum()
```

Out[ ]:

```
Index                                     0
erythema                                  0
scaling                                   0
definite borders                          0
itching                                   0
koebner phenomenon                        0
polygonal papules                         0
follicular papules                        0
oral mucosal involvement                  0
knee and elbow involvement                0
scalp involvement                         0
family history                            0
melanin incontinence                      0
eosinophils in the infiltrate             0
PNL infiltrate                            0
fibrosis of the papillary dermis          0
exocytosis                                0
acanthosis                                0
hyperkeratosis                            0
parakeratosis                             0
clubbing of the rete ridges               0
elongation of the rete ridges             0
thinning of the suprapapillary epidermis  0
```

```
spongiform pustule                          0
munro microabcess                           0
focal hypergranulosis                       0
disappearance of the granular layer         0
vacuolisation and damage of basal layer     0
spongiosis                                  0
saw-tooth appearance of retes               0
follicular horn plug                        0
perifollicular parakeratosis                0
inflammatory monoluclear inflitrate         0
band-like infiltrate                        0
Age                                         8
Class                                       0
```

Filling up the missing values using the mean of that attribute

In [ ]:

```
main_df = pd.read_csv('dermatologyCSV.csv', na_values='?')

main_df.columns = ['index',
    'erythema',
    'scaling',
    'definite borders',
    'itching',
    'koebner phenomenon',
    'polygonal papules',
    'follicular papules',
    'oral mucosal involvement',
    'knee and elbow involvement',
    'scalp involvement',
    'family history',
    'melanin incontinence',
    'eosinophils in the infiltrate',
    'PNL infiltrate',
    'fibrosis of the papillary dermis',
    'exocytosis',
    'acanthosis',
    'hyperkeratosis',
    'parakeratosis',
    'clubbing of the rete ridges',
    'elongation of the rete ridges',
    'thinning of the suprapapillary epidermis',
    'spongiform pustule',
    'munro microabcess',
    'focal hypergranulosis',
    'disappearance of the granular layer',
    'vacuolisation and damage of basal layer',
    'spongiosis',
    'saw-tooth appearance of retes',
    'follicular horn plug',
    'perifollicular parakeratosis',
    'inflammatory monoluclear inflitrate',
    'band-like infiltrate',
    'Age',
    'Class'
]

main_df['Age'] = main_df['Age'].fillna(main_df['Age'].mean())
```

Verifying by rechecking the number of missing values in each attribute(it should be 0 for all)

In [ ]:
```
main_df.isna().sum()
```

Out[ ]:
```
index                                       0
erythema                                    0
scaling                                     0
definite borders                            0
itching                                     0
koebner phenomenon                          0
polygonal papules                           0
follicular papules                          0
oral mucosal involvement                    0
knee and elbow involvement                  0
scalp involvement                           0
family history                              0
melanin incontinence                        0
eosinophils in the infiltrate               0
PNL infiltrate                              0
fibrosis of the papillary dermis            0
exocytosis                                  0
acanthosis                                  0
hyperkeratosis                              0
parakeratosis                               0
clubbing of the rete ridges                 0
elongation of the rete ridges               0
thinning of the suprapapillary epidermis    0
spongiform pustule                          0
munro microabcess                           0
focal hypergranulosis                       0
disappearance of the granular layer         0
vacuolisation and damage of basal layer     0
spongiosis                                  0
saw-tooth appearance of retes               0
follicular horn plug                        0
perifollicular parakeratosis                0
inflammatory monoluclear inflitrate         0
band-like infiltrate                        0
Age                                         0
Class                                       0
dtype: int64
```

EXPLORATORY DATA ANALYTICS

In [ ]:
```
fig, ax = plt.subplots(figsize =(10, 7))
ax.hist(main_df.Age, bins = [0, 15, 30, 45, 60, 75, 90])
```

Out[ ]:
```
(array([ 26.,  99., 126.,  84.,  30.,   1.]),
 array([ 0, 15, 30, 45, 60, 75, 90]),
 <BarContainer object of 6 artists>)
```

In [ ]:

```
AgeP = main_df.Age
AgeP%10
```

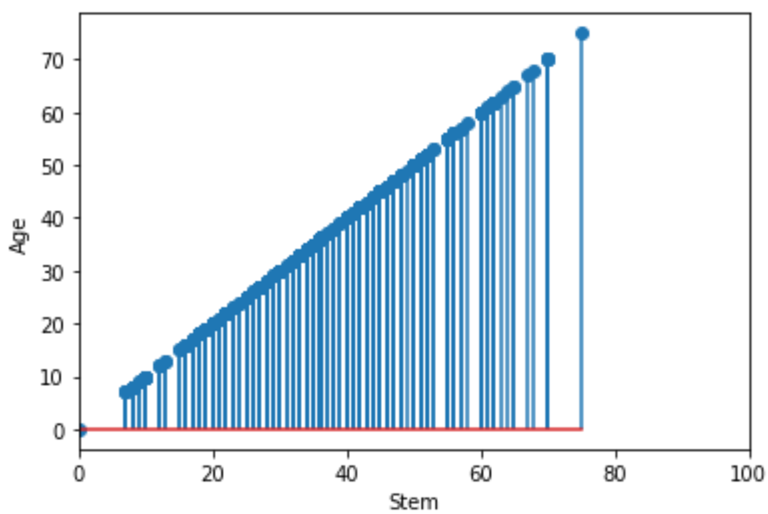
Out[ ]:

```
0      5.0
1      8.0
2      6.0
3      0.0
4      5.0
      ...
361    5.0
362    6.0
363    8.0
364    0.0
365    5.0
Name: Age, Length: 366, dtype: float64
```

In [ ]:

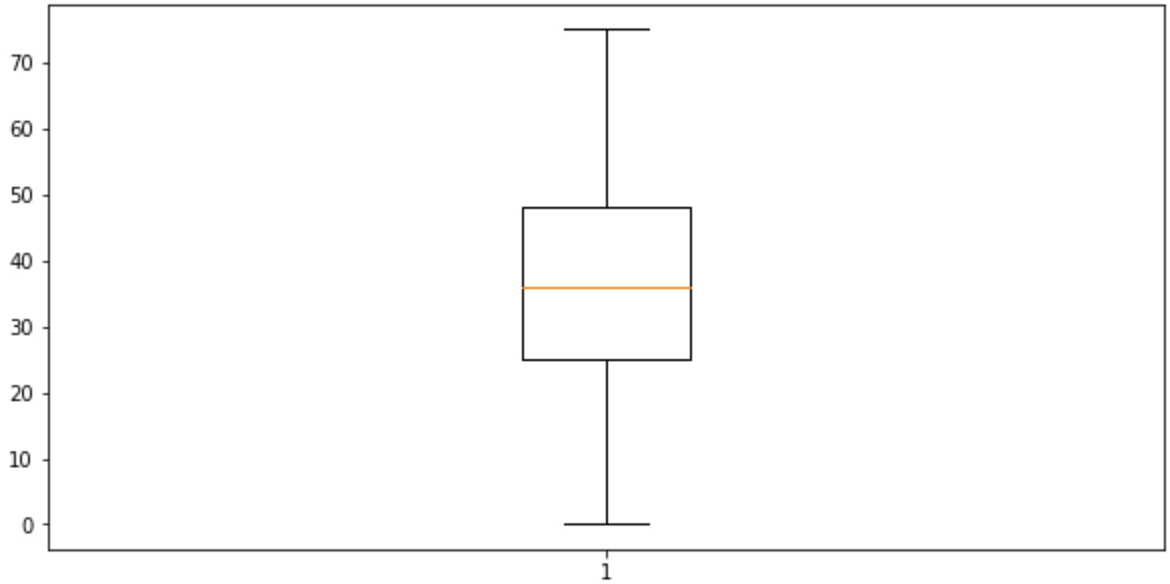
```
plt.ylabel('Age')
plt.xlabel('Stem')
plt.xlim(0, 100)
plt.stem(AgeP, main_df.Age)
```

Out[ ]: <StemContainer object of 3 artists>



In [ ]:

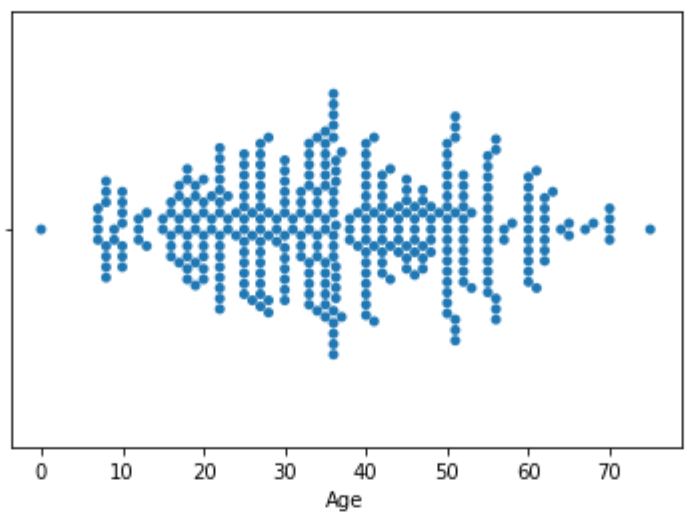
```
fig = plt.figure(figsize=(10, 5))
plt.boxplot(AgeP)
plt.show()
```



In [ ]:

```
sns.swarmplot(x=AgeP)
```

Out[ ]: <AxesSubplot:xlabel='Age'>



ACLOSE ALGORITHM

By definition, An itemset is maximal frequent if none of its immediate supersets is frequent. An itemset is closed if none of its immediate supersets has the same support as the itemset. Let's use an example and diagram representation to better understand the concept.

There are many different approaches trying to efficiently find close and maximal frequent itemsets.

But this approach can be quite time consuming, considering an $O(n^2)$ runtime complexity. To optimize the algorithm when dealing with large databases, we need to take advantage of a

python dictionary. By storing all itemsets with the same support count into a dictionary, using support as the key, we can reduce the complexity to O(n). Because we do not need to compare every item since all supersets have ≤ support from their parents. And we only need to compare items with the same support count when finding closed itemsets. The same thing applies when finding maximal itemsets.
For implementation, I used the MLXtend library and fpgrowth function to compute the frequent itemsets first, and write my own function to mine the closed and maximal frequent itemsets from the result of the first step.

Importing all the basic librarys

Computing the Frequent Item Set using mlxtend.frequent_patterns

In [ ]:

```
from mlxtend.preprocessing import TransactionEncoder
import time
from mlxtend.frequent_patterns import fpgrowth
```

In [ ]:

```
te = TransactionEncoder()
te_ary = te.fit('dermatologyCSV.csv').transform('dermatologyCSV.csv')
df = pd.DataFrame(te_ary, columns=te.columns_)
```

In [ ]:

```
start_time = time.time()
frequent = fpgrowth(df, min_support=0.001, use_colnames=True)
print('Time to find frequent itemset')
print("--- %s seconds ---" % (time.time() - start_time))
```

```
Time to find frequent itemset
--- 0.20922088623046875 seconds ---
```

Finding closed/max frequent itemset using frequent itemset found in task1

In [ ]:

```
su = frequent.support.unique()
```

Dictionary storing the itemset with same support count key

In [ ]:

```
fredic = {}
for i in range(len(su)):
    inset = list(frequent.loc[frequent.support ==su[i]]['itemsets'])
    fredic[su[i]] = inset
```

Dictionary storing the itemset with support count <= key

In [ ]:

```
fredic2 = {}
for i in range(len(su)):
    inset2 = list(frequent.loc[frequent.support<=su[i]]['itemsets'])
    fredic2[su[i]] = inset2
```

Finding Closed frequent itemset

In [ ]:
```
start_time = time.time()
```

In [ ]:
```
cl = []
for index, row in frequent.iterrows():
    isclose = True
    cli = row['itemsets']
    cls = row['support']
    checkset = fredic[cls]
    for i in checkset:
        if (cli!=i):
            if(frozenset.issubset(cli,i)):
                isclose = False
                break

    if(isclose):
        cl.append(row['itemsets'])

print('Time to find Close frequent itemset')
print("--- %s seconds ---" % (time.time() - start_time))
```

```
Time to find Close frequent itemset
--- 111.80388355255127 seconds ---
```

In [ ]:
```
cl
```

Out[ ]:
```
[frozenset({'d'}),
 frozenset({'e'}),
 frozenset({'r'}),
 frozenset({'m'}),
 frozenset({'a'}),
 frozenset({'t'}),
 frozenset({'o'}),
 frozenset({'l'}),
 frozenset({'g'}),
 frozenset({'y'}),
 frozenset({'C'}),
 frozenset({'S'}),
 frozenset({'V'}),
 frozenset({'.'}),
 frozenset({'c'}),
 frozenset({'s'}),
 frozenset({'v'})]
```

Finding Maximal frequent itemset

In [ ]:
```
start_time = time.time()
```

In [ ]:

```
ml = []
for index, row in frequent.iterrows():
    isclose = True
    cli = row['itemsets']
    cls = row['support']
    checkset = fredic2[cls]
    for i in checkset:
        if (cli!=i):
            if(frozenset.issubset(cli,i)):
                isclose = False
                break
    if(isclose):
        ml.append(row['itemsets'])
```

In [ ]:

```
print('Time to find Max frequent itemset')
print("--- %s seconds ---" % (time.time() - start_time))
```

```
Time to find Max frequent itemset
--- 124.77249717712402 seconds ---
```

In [ ]:

```
ml
```

Out[ ]:

```
[frozenset({'d'}),
 frozenset({'e'}),
 frozenset({'r'}),
 frozenset({'m'}),
 frozenset({'a'}),
 frozenset({'t'}),
 frozenset({'o'}),
 frozenset({'l'}),
 frozenset({'g'}),
 frozenset({'y'}),
 frozenset({'C'}),
 frozenset({'S'}),
 frozenset({'V'}),
 frozenset({'.'}),
 frozenset({'c'}),
 frozenset({'s'}),
 frozenset({'v'})]
```

KNN CLASSIFICATION/REGRESSION

The intuition behind the KNN algorithm is one of the simplest of all the supervised machine learning algorithms. It simply calculates the distance of a new data point to all other training data points. The distance can be of any type e.g Euclidean or Manhattan etc. It then selects the K-nearest data points, where K can be any integer. Finally it assigns the data point to the class to which the majority of the K data points belong.

In this section we'll present some of the pros and cons of using the KNN algorithm. Pros

```
It is extremely easy to implement
As said earlier, it is lazy learning algorithm and therefore requires
no training prior to making real time predictions. This makes the KNN
algorithm much faster than other algorithms that require training e.g
```

```
SVM, linear regression, etc.
Since the algorithm requires no training before making predictions,
new data can be added seamlessly.
There are only two parameters required to implement KNN i.e. the value
of K and the distance function (e.g. Euclidean or Manhattan etc.)
```

Cons

```
The KNN algorithm doesn't work well with high dimensional data because
with large number of dimensions, it becomes difficult for the
algorithm to calculate distance in each dimension.
The KNN algorithm has a high prediction cost for large datasets. This
is because in large datasets the cost of calculating distance between
new point and each existing point becomes higher.
Finally, the KNN algorithm doesn't work well with categorical features
since it is difficult to find the distance between dimensions with
categorical features
```

Splitting our dataset into its attributes and labels.

In [ ]:

```
X = main_df.iloc[:, :-1].values
Y = main_df.iloc[:, 35].values
```

In [ ]:

```
X
```

Out[ ]:

```
array([[ 2.,  2.,  0., ...,  1.,  0., 55.],
       [ 3.,  3.,  3., ...,  1.,  0.,  8.],
       [ 2.,  1.,  2., ...,  2.,  3., 26.],
       ...,
       [ 3.,  2.,  2., ...,  2.,  3., 28.],
       [ 2.,  1.,  3., ...,  2.,  3., 50.],
       [ 3.,  2.,  2., ...,  3.,  0., 35.]])
```

In [ ]:

```
Y
```

Out[ ]:

```
array([2, 1, 3, 1, 3, 2, 5, 3, 4, 4, 1, 2, 2, 1, 3, 4, 2, 1, 3, 5, 6, 2,
       5, 3, 5, 1, 6, 5, 2, 3, 1, 2, 1, 1, 4, 2, 3, 2, 3, 1, 2, 4, 1, 2,
       5, 3, 4, 6, 2, 3, 3, 4, 1, 1, 5, 1, 2, 3, 4, 2, 6, 1, 5, 1, 2, 3,
       1, 4, 5, 1, 2, 6, 3, 5, 4, 2, 2, 1, 3, 5, 1, 2, 2, 2, 5, 1, 1, 3,
       1, 4, 2, 2, 5, 1, 3, 4, 2, 5, 1, 6, 2, 5, 1, 2, 2, 1, 4, 1, 3, 1,
       1, 3, 5, 3, 3, 5, 2, 3, 4, 1, 2, 5, 6, 1, 1, 2, 6, 3, 5, 4, 1, 1,
       3, 5, 5, 1, 4, 2, 3, 1, 2, 1, 1, 3, 3, 3, 2, 5, 4, 2, 2, 1, 1, 1,
       5, 3, 2, 3, 2, 2, 4, 2, 3, 6, 2, 1, 1, 3, 4, 3, 3, 1, 1, 1, 3, 1,
       1, 2, 3, 3, 1, 1, 1, 1, 6, 2, 2, 2, 2, 1, 3, 3, 3, 1, 1, 2, 3, 2,
       2, 2, 5, 5, 5, 5, 5, 1, 1, 1, 1, 1, 1, 1, 3, 3, 3, 3, 3, 3, 4, 4,
       4, 4, 5, 5, 5, 5, 5, 5, 5, 2, 2, 2, 2, 1, 1, 1, 1, 1, 1, 6, 6, 1,
       1, 1, 1, 1, 1, 1, 1, 3, 3, 3, 3, 3, 3, 3, 4, 4, 4, 4, 4, 4, 5, 5,
       5, 5, 6, 6, 6, 4, 4, 4, 1, 1, 1, 1, 1, 2, 2, 4, 4, 4, 1, 1, 2, 2,
       2, 3, 3, 3, 3, 1, 1, 1, 1, 5, 5, 5, 5, 5, 3, 3, 3, 4, 1, 1, 4, 4,
       4, 1, 1, 1, 3, 3, 3, 3, 3, 1, 1, 1, 1, 4, 4, 1, 1, 4, 3, 3, 4, 1,
       1, 4, 4, 5, 5, 1, 1, 5, 5, 3, 1, 5, 5, 6, 6, 4, 4, 6, 6, 6, 1, 1,
       1, 5, 5, 1, 1, 1, 1, 2, 2, 4, 4, 3, 3, 1], dtype=int64)
```

To avoid over-fitting, we will divide our dataset into training and test splits, which gives us a better idea as to how our algorithm performed during the testing phase. This way our algorithm is tested on un-seen data, as it would be in a production application

The below script splits the dataset into 80% train data and 20% test data.

In [ ]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.20)
```

Before making any actual predictions, it is always a good practice to scale the features so that all of them can be uniformly evaluated.

Since the range of values of raw data varies widely, in some machine learning algorithms, objective functions will not work properly without normalization. For example, the majority of classifiers calculate the distance between two points by the Euclidean distance. If one of the features has a broad range of values, the distance will be governed by this particular feature. Therefore, the range of all features should be normalized so that each feature contributes approximately proportionately to the final distance.

In [ ]:

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X_train)

X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
```

The first step is to import the KNeighborsClassifier class from the sklearn.neighbors library. In the second line, this class is initialized with one parameter, i.e. n_neigbours. This is basically the value for the K.

In [ ]:

```
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors=5)
classifier.fit(X_train, Y_train)
```

Out[ ]:

```
KNeighborsClassifier()
```

In [ ]:

```
Y_pred = classifier.predict(X_test)
```

For evaluating an algorithm, confusion matrix, precision, recall and f1 score are the most commonly used metrics. The confusion_matrix and classification_report methods of the sklearn.metrics can be used to calculate these metrics.

In [ ]:

```
from sklearn.metrics import classification_report, confusion_matrix
print(confusion_matrix(Y_test, Y_pred))
print(classification_report(Y_test, Y_pred))
```

```
[[17  0  0  0  0  0]
 [ 0 15  0  4  0  0]
 [ 0  0 13  0  0  0]
 [ 0  0  0 11  0  0]
 [ 0  0  0  0  8  0]
```

```
 [ 0  0  0  0  0  6]]
              precision    recall  f1-score   support

           1       1.00      1.00      1.00        17
           2       1.00      0.79      0.88        19
           3       1.00      1.00      1.00        13
           4       0.73      1.00      0.85        11
           5       1.00      1.00      1.00         8
           6       1.00      1.00      1.00         6

    accuracy                           0.95        74
   macro avg       0.96      0.96      0.95        74
weighted avg       0.96      0.95      0.95        74
```

The results show that our KNN algorithm was able to classify all the 30 records in the test set with 95% accuracy, which is excellent. Although the algorithm performed very well with this dataset, don't expect the same results with all applications. As noted earlier, KNN doesn't always perform as well with high-dimensionality or categorical features.

In the training and prediction section we said that there is no way to know beforehand which value of K that yields the best results in the first go. We randomly chose 5 as the K value and it just happen to result in 95% accuracy.

One way to help you find the best value of K is to plot the graph of K value and the corresponding error rate for the dataset.

In this section, we will plot the mean error for the predicted values of test set for all the K values between 1 and 40.

To do so, let's first calculate the mean of error for all the predicted values where K ranges from 1 and 40.

In [ ]:

```
error = []

for i in range(1, 40):
    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(X_train, Y_train)
    pred_i = knn.predict(X_test)
    error.append(np.mean(pred_i != Y_test))
```

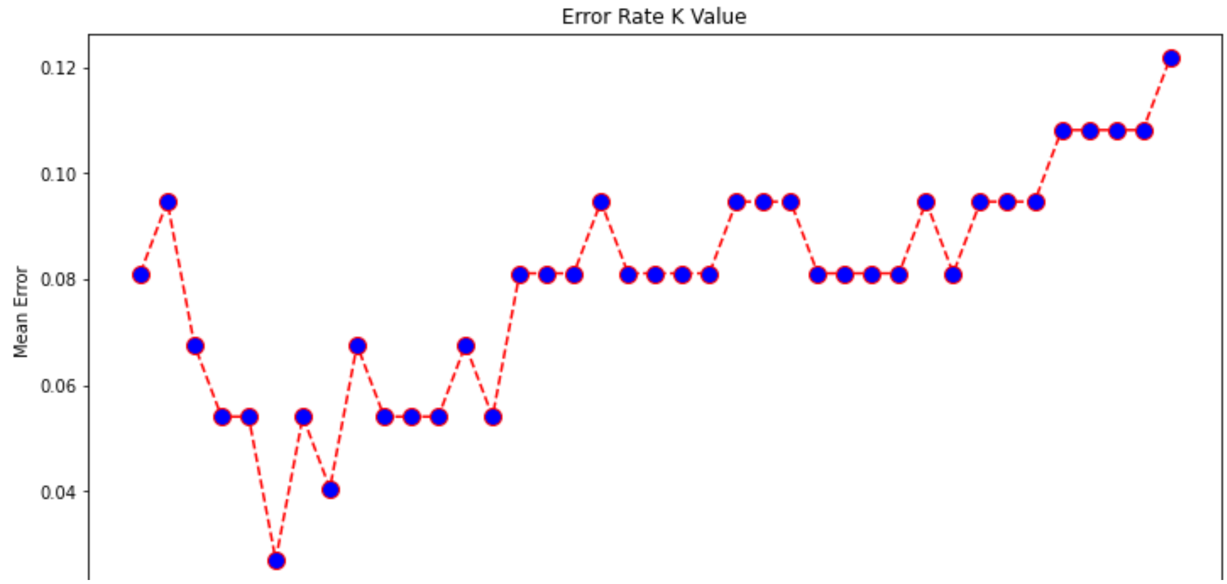
In each iteration the mean error for predicted values of test set is calculated and the result is appended to the error list

In [ ]:

```
plt.figure(figsize=(12, 6))
plt.plot(range(1, 40), error, color='red', linestyle='dashed', marker='o',
         markerfacecolor='blue', markersize=10)
plt.title('Error Rate K Value')
plt.xlabel('K Value')
plt.ylabel('Mean Error')
```

Out[ ]:

```
Text(0, 0.5, 'Mean Error')
```



AGNES Clustering

Steps to Perform AGNES Clustering

Following are the steps involved in agglomerative clustering:

```
At the start, treat each data point as one cluster. Therefore, the
number of clusters at the start will be K, while K is an integer
representing the number of data points.
Form a cluster by joining the two closest data points resulting in K-1
clusters.
Form more clusters by joining the two closest clusters resulting in
K-2 clusters.
Repeat the above three steps until one big cluster is formed.
Once single cluster is formed, dendrograms are used to divide into
multiple clusters depending upon the problem. We will study the
concept of dendrogram in detail in an upcoming section.
```

There are different ways to find distance between the clusters. The distance itself can be Euclidean or Manhattan distance. Following are some of the options to measure distance between two clusters:

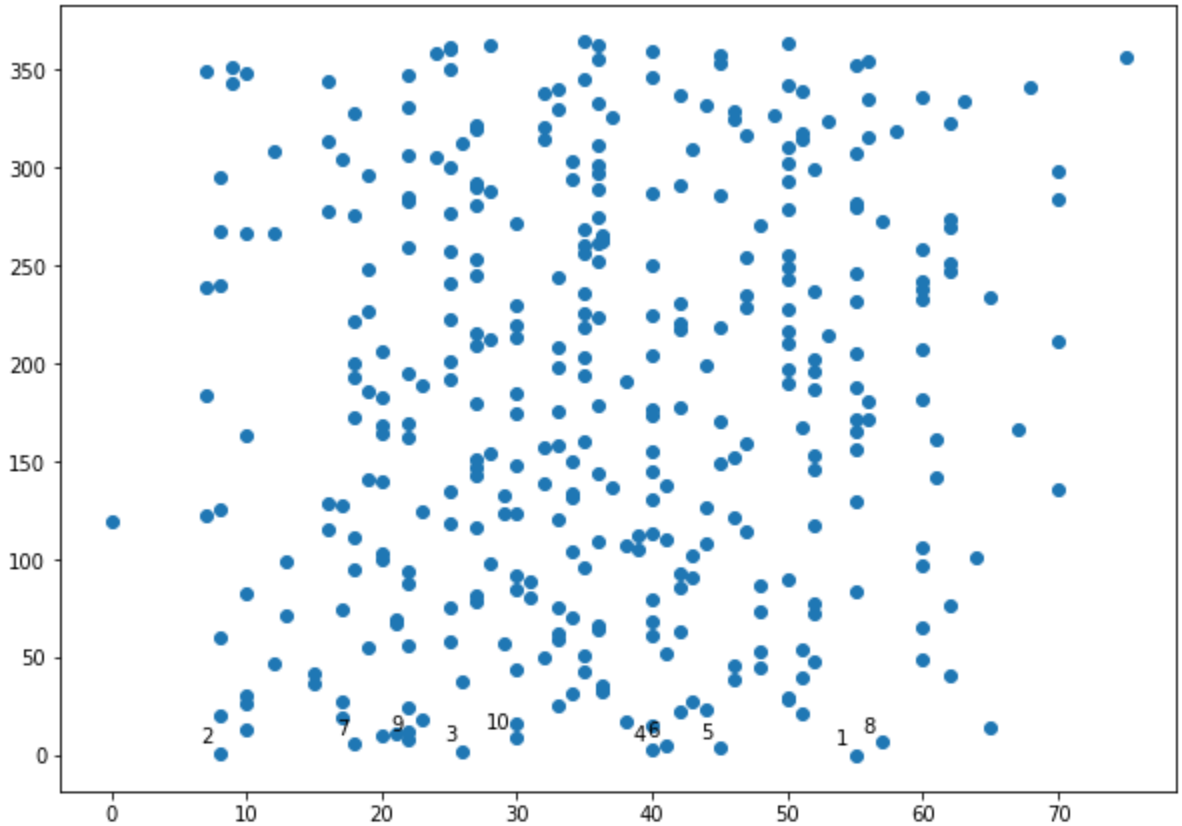
```
Measure the distance between the closes points of two clusters.
Measure the distance between the farthest points of two clusters.
Measure the distance between the centroids of two clusters.
Measure the distance between all possible combination of points
between the two clusters and take the mean.
```

In [ ]:

```
AgePlot = main_df.Age
IndexPlot = main_df.index
```

Plotting the scatter plot of the Age attribute

In [ ]:
```
import matplotlib.pyplot as plt

labels = range(1, 11)
plt.figure(figsize=(10, 7))
plt.subplots_adjust(bottom=0.1)
plt.scatter(AgePlot, IndexPlot , label='True Position')

for label, x, y in zip(labels, AgePlot, IndexPlot):
    plt.annotate(
        label,
        xy=(x, y), xytext=(-3, 3),
        textcoords='offset points', ha='right', va='bottom')
plt.show()
```



In [ ]:
```
X = []

for i in range(0, 366):
    X.append([AgePlot[i], IndexPlot[i]])
```

In [ ]:
```
X
```

Out[ ]:
```
[[55.0, 0],
 [8.0, 1],
 [26.0, 2],
 [40.0, 3],
 [45.0, 4],
 [41.0, 5],
 [18.0, 6],
```

```
[57.0, 7],
[22.0, 8],
[30.0, 9],
[20.0, 10],
[21.0, 11],
[22.0, 12],
[10.0, 13],
[65.0, 14],
[40.0, 15],
[30.0, 16],
[38.0, 17],
[23.0, 18],
[17.0, 19],
[8.0, 20],
[51.0, 21],
[42.0, 22],
[44.0, 23],
[22.0, 24],
[33.0, 25],
[10.0, 26],
[17.0, 27],
[43.0, 28],
[50.0, 29],
[50.0, 30],
[10.0, 31],
[34.0, 32],
[36.29608938547486, 33],
[36.29608938547486, 34],
[36.29608938547486, 35],
[36.29608938547486, 36],
[15.0, 37],
[26.0, 38],
[46.0, 39],
[51.0, 40],
[62.0, 41],
[15.0, 42],
[35.0, 43],
[30.0, 44],
[48.0, 45],
[46.0, 46],
[12.0, 47],
[52.0, 48],
[60.0, 49],
[32.0, 50],
[35.0, 51],
[41.0, 52],
[48.0, 53],
[51.0, 54],
[19.0, 55],
[22.0, 56],
[29.0, 57],
[25.0, 58],
[33.0, 59],
[8.0, 60],
[40.0, 61],
[33.0, 62],
[42.0, 63],
[36.0, 64],
[60.0, 65],
[36.0, 66],
[21.0, 67],
[40.0, 68],
```

```
[21.0, 69],
[34.0, 70],
[13.0, 71],
[52.0, 72],
[48.0, 73],
[17.0, 74],
[25.0, 75],
[33.0, 76],
[62.0, 77],
[52.0, 78],
[27.0, 79],
[40.0, 80],
[31.0, 81],
[27.0, 82],
[10.0, 83],
[55.0, 84],
[30.0, 85],
[42.0, 86],
[48.0, 87],
[22.0, 88],
[31.0, 89],
[50.0, 90],
[43.0, 91],
[30.0, 92],
[42.0, 93],
[22.0, 94],
[18.0, 95],
[35.0, 96],
[60.0, 97],
[28.0, 98],
[13.0, 99],
[20.0, 100],
[64.0, 101],
[43.0, 102],
[20.0, 103],
[34.0, 104],
[39.0, 105],
[60.0, 106],
[38.0, 107],
[44.0, 108],
[36.0, 109],
[41.0, 110],
[18.0, 111],
[39.0, 112],
[40.0, 113],
[47.0, 114],
[16.0, 115],
[27.0, 116],
[52.0, 117],
[25.0, 118],
[0.0, 119],
[33.0, 120],
[46.0, 121],
[7.0, 122],
[30.0, 123],
[29.0, 124],
[23.0, 125],
[8.0, 126],
[44.0, 127],
[17.0, 128],
[16.0, 129],
[55.0, 130],
```

```
[40.0, 131],
[34.0, 132],
[29.0, 133],
[34.0, 134],
[25.0, 135],
[70.0, 136],
[37.0, 137],
[41.0, 138],
[32.0, 139],
[20.0, 140],
[19.0, 141],
[61.0, 142],
[27.0, 143],
[36.0, 144],
[40.0, 145],
[52.0, 146],
[27.0, 147],
[30.0, 148],
[45.0, 149],
[34.0, 150],
[27.0, 151],
[46.0, 152],
[52.0, 153],
[28.0, 154],
[40.0, 155],
[55.0, 156],
[32.0, 157],
[33.0, 158],
[47.0, 159],
[35.0, 160],
[61.0, 161],
[22.0, 162],
[10.0, 163],
[20.0, 164],
[55.0, 165],
[67.0, 166],
[51.0, 167],
[20.0, 168],
[22.0, 169],
[45.0, 170],
[55.0, 171],
[56.0, 172],
[18.0, 173],
[40.0, 174],
[30.0, 175],
[33.0, 176],
[40.0, 177],
[42.0, 178],
[36.0, 179],
[27.0, 180],
[56.0, 181],
[60.0, 182],
[20.0, 183],
[7.0, 184],
[30.0, 185],
[19.0, 186],
[52.0, 187],
[55.0, 188],
[23.0, 189],
[50.0, 190],
[38.0, 191],
[25.0, 192],
```

```
[18.0, 193],
[35.0, 194],
[22.0, 195],
[52.0, 196],
[50.0, 197],
[33.0, 198],
[44.0, 199],
[18.0, 200],
[25.0, 201],
[52.0, 202],
[35.0, 203],
[40.0, 204],
[55.0, 205],
[20.0, 206],
[60.0, 207],
[33.0, 208],
[27.0, 209],
[50.0, 210],
[70.0, 211],
[28.0, 212],
[30.0, 213],
[53.0, 214],
[27.0, 215],
[50.0, 216],
[42.0, 217],
[45.0, 218],
[35.0, 219],
[30.0, 220],
[42.0, 221],
[18.0, 222],
[25.0, 223],
[36.0, 224],
[40.0, 225],
[35.0, 226],
[19.0, 227],
[50.0, 228],
[47.0, 229],
[30.0, 230],
[42.0, 231],
[55.0, 232],
[60.0, 233],
[65.0, 234],
[47.0, 235],
[35.0, 236],
[52.0, 237],
[60.0, 238],
[7.0, 239],
[8.0, 240],
[25.0, 241],
[60.0, 242],
[50.0, 243],
[33.0, 244],
[27.0, 245],
[55.0, 246],
[62.0, 247],
[19.0, 248],
[50.0, 249],
[40.0, 250],
[62.0, 251],
[36.0, 252],
[27.0, 253],
[47.0, 254],
```

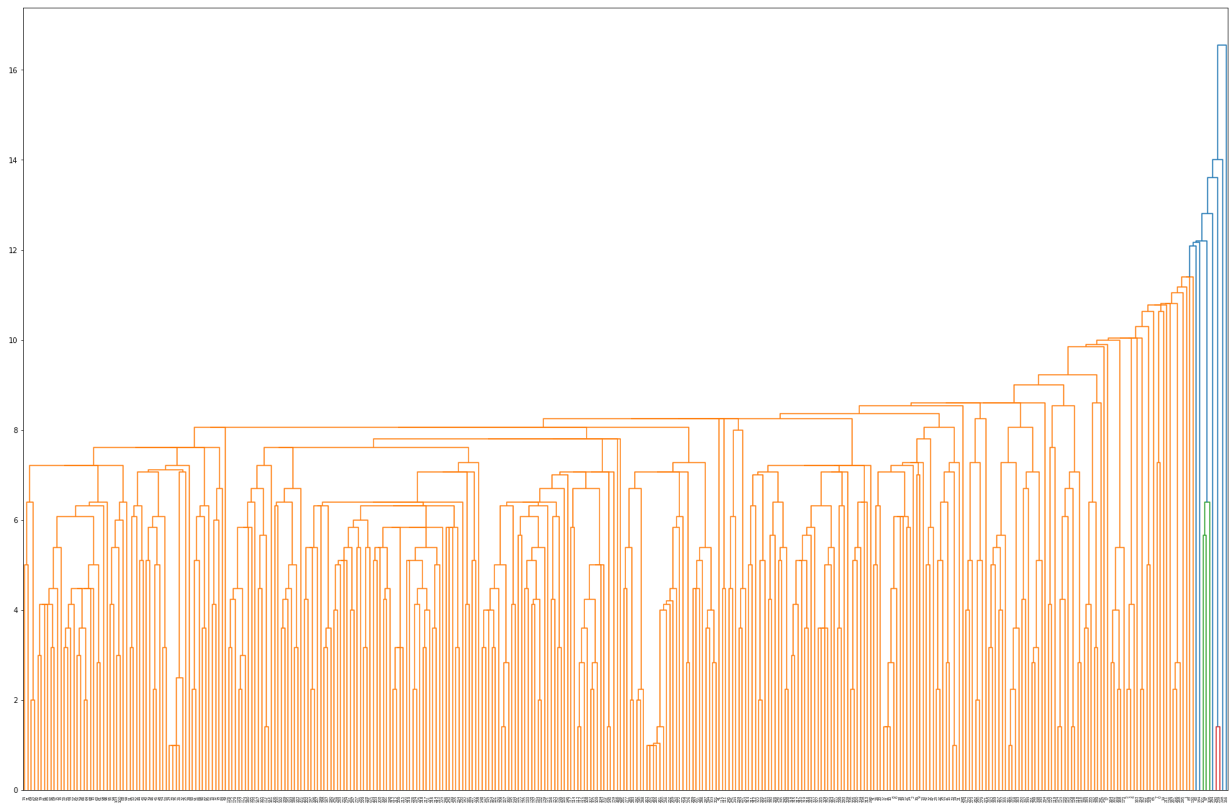
```
[50.0, 255],
[35.0, 256],
[25.0, 257],
[60.0, 258],
[22.0, 259],
[35.0, 260],
[36.0, 261],
[36.29608938547486, 262],
[36.29608938547486, 263],
[36.29608938547486, 264],
[36.29608938547486, 265],
[10.0, 266],
[12.0, 267],
[8.0, 268],
[35.0, 269],
[62.0, 270],
[48.0, 271],
[30.0, 272],
[57.0, 273],
[62.0, 274],
[36.0, 275],
[18.0, 276],
[25.0, 277],
[16.0, 278],
[50.0, 279],
[55.0, 280],
[27.0, 281],
[55.0, 282],
[22.0, 283],
[70.0, 284],
[22.0, 285],
[45.0, 286],
[40.0, 287],
[28.0, 288],
[36.0, 289],
[27.0, 290],
[42.0, 291],
[27.0, 292],
[50.0, 293],
[34.0, 294],
[8.0, 295],
[19.0, 296],
[36.0, 297],
[70.0, 298],
[52.0, 299],
[25.0, 300],
[36.0, 301],
[50.0, 302],
[34.0, 303],
[17.0, 304],
[24.0, 305],
[22.0, 306],
[55.0, 307],
[12.0, 308],
[43.0, 309],
[50.0, 310],
[36.0, 311],
[26.0, 312],
[16.0, 313],
[32.0, 314],
[51.0, 315],
[56.0, 316],
```

```
 [47.0, 317],
 [51.0, 318],
 [58.0, 319],
 [27.0, 320],
 [32.0, 321],
 [27.0, 322],
 [62.0, 323],
 [53.0, 324],
 [46.0, 325],
 [37.0, 326],
 [49.0, 327],
 [18.0, 328],
 [46.0, 329],
 [33.0, 330],
 [22.0, 331],
 [44.0, 332],
 [36.0, 333],
 [63.0, 334],
 [56.0, 335],
 [60.0, 336],
 [42.0, 337],
 [32.0, 338],
 [51.0, 339],
 [33.0, 340],
 [68.0, 341],
 [50.0, 342],
 [9.0, 343],
 [16.0, 344],
 [35.0, 345],
 [40.0, 346],
 [22.0, 347],
 [10.0, 348],
 [7.0, 349],
 [25.0, 350],
 [9.0, 351],
 [55.0, 352],
 [45.0, 353],
 [56.0, 354],
 [36.0, 355],
 [75.0, 356],
 [45.0, 357],
 [24.0, 358],
 [40.0, 359],
 [25.0, 360],
 [25.0, 361],
```

Forming a Dendogram

In [ ]:

```
from scipy.cluster.hierarchy import dendrogram, linkage
from matplotlib import pyplot as plt

linked = linkage(X, 'single')

labelList = range(0, 366)

plt.figure(figsize=(30, 20))
dendrogram(linked,
            orientation='top',
            labels=labelList,
            distance_sort='descending',
            show_leaf_counts=True)
plt.show()
```

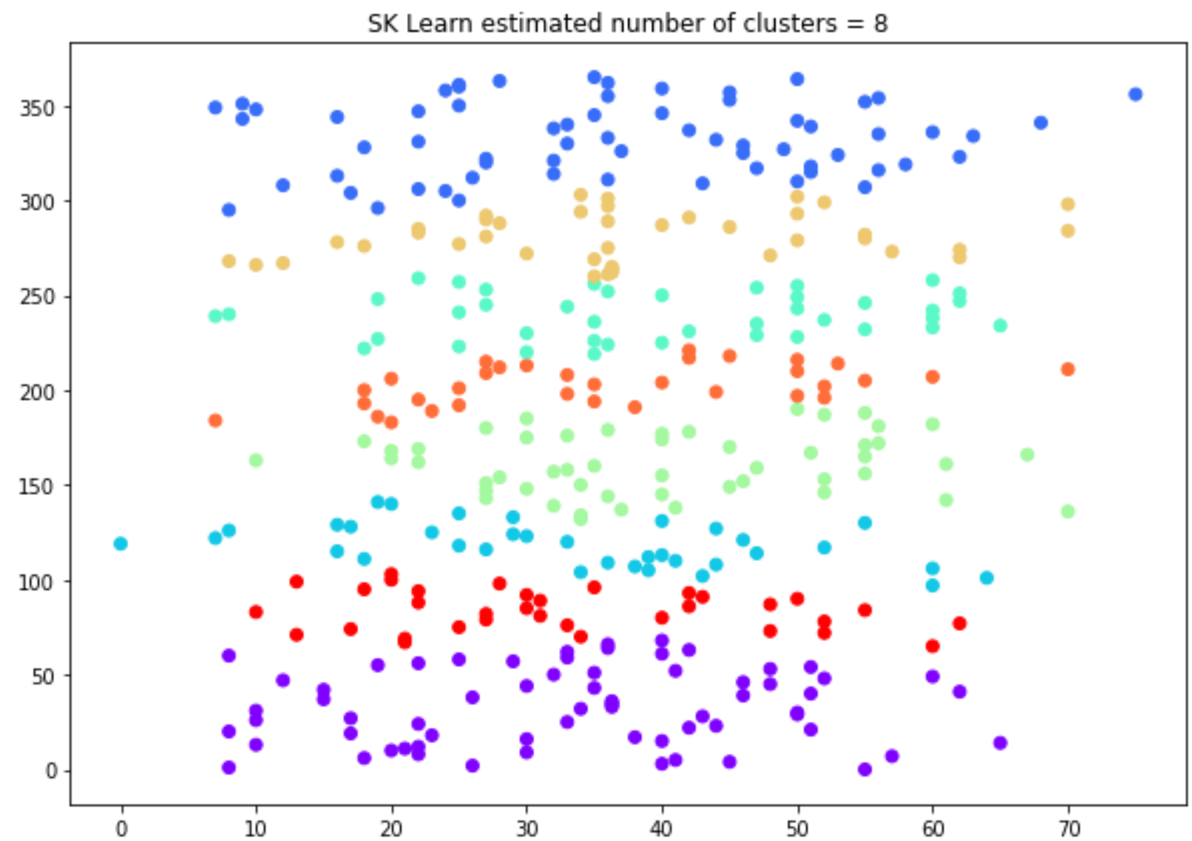


You can see the cluster labels from all of your data points. Since we had eight clusters, we have five labels in the output i.e. 0 to 7.

In [ ]:

```
cluster = AgglomerativeClustering(n_clusters=None, affinity='euclidean', linkage='war

cluster.fit_predict(X)

print(f"Number of clusters = {1+np.amax(cluster.labels_)}")

print("Classifying the points into clusters:")
print(cluster.labels_)

plt.figure(figsize=(10, 7))
plt.scatter(AgePlot ,IndexPlot, c=cluster.labels_, cmap='rainbow')
plt.title(f"SK Learn estimated number of clusters = {1+np.amax(cluster.labels_)}")
plt.show()
```

```
Number of clusters = 8
Classifying the points into clusters:
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 7 0 7 0 7 7 7 7 7
 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 2 7 7 7 2 2 7 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 4 2 4 2 4 4 4 4 2 2 4 4 4 4 4 4
 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 6 6
 4 6 4 4 6 4 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 3 3 6
 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
 3 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 1
 1 5 5 5 1 5 5 5 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
```



Plot of the clusters