- Dealing with Missing values and noise in the selected features

- Comparing the changes before and after Data Cleaning

- Buidling the Model

DATA CLEANING:

Importing the Libraries

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

Loading the dataset

In [2]:
```
data = pd.read_csv('iris_dataset.csv')
```

Displaying properties of dataset for further processing

In [3]:
```
data.info()
data.head()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 105 entries, 0 to 104
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   sepal_length  105 non-null    float64
 1   sepal_width   101 non-null    float64
 2   petal_length  97 non-null     float64
 3   petal_width   105 non-null    float64
 4   species       105 non-null    object 
dtypes: float64(4), object(1)
memory usage: 4.2+ KB
```

Out[3]:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.045070	2.508203	3.018024	1.164924	Iris-versicolor
1	6.325517	2.115481	4.542052	1.413651	Iris-versicolor
2	5.257497	3.814303	1.470660	0.395348	Iris-setosa
3	6.675168	3.201700	5.785461	2.362764	Iris-virginica
4	5.595237	2.678166	4.077750	1.369266	Iris-versicolor

In [4]:
```
data.tail()
```

Out[4]:

	sepal_length	sepal_width	petal_length	petal_width	species
100	4.874848	3.217348	1.592887	0.123588	Iris-setosa
101	5.564197	2.771731	3.483588	1.074754	Iris-versicolor
102	5.548047	4.249211	1.453466	0.214527	Iris-setosa
103	5.510482	2.652867	4.276817	1.298032	Iris-versicolor
104	4.538713	3.056142	1.545136	0.241424	Iris-setosa

In [5]:
```
data.describe()
```

Out[5]:

	sepal_length	sepal_width	petal_length	petal_width
count	105.000000	101.000000	97.000000	105.000000
mean	5.858909	3.059083	3.812370	1.199708
std	0.861638	0.455116	1.793489	0.787193
min	4.344007	1.946010	1.033031	-0.072203
25%	5.159145	2.768688	1.545136	0.333494
50%	5.736104	3.049459	4.276817	1.331797
75%	6.435413	3.290318	5.094427	1.817211
max	7.795561	4.409565	6.768611	2.603123

In [6]:
```
data.shape
```

Out[6]:
```
(105, 5)
```

FINDING IF THERE ARE ANY MISSING VALUES IN THE GIVEN DATASET AND VARIOUS WAYS OF HANDLING THEM :

In [7]:
```
data.isnull().sum()
```

Out[7]:
```
sepal_length    0
sepal_width     4
petal_length    8
petal_width     0
species         0
dtype: int64
```

From the above, we could observe that there are **4** missing values in the feature **"sepal_width"** and **8** missing values in the feature **"petal_length".**

sepal_width :

The rows with missing data in the column "sepal_width" are:

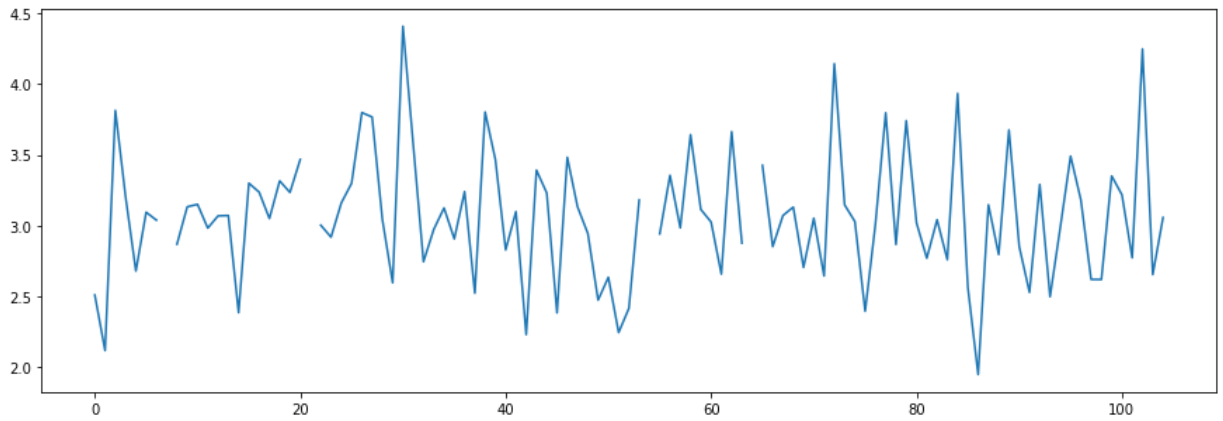
In [8]:
```
data[data.sepal_width.isnull()]
```

Out[8]:

	sepal_length	sepal_width	petal_length	petal_width	species
7	5.205868	NaN	1.675654	0.112269	Iris-setosa
21	6.365979	NaN	4.964905	1.817211	Iris-virginica
54	6.265590	NaN	4.701306	1.290187	Iris-versicolor
64	6.340344	NaN	4.302989	1.331797	Iris-versicolor

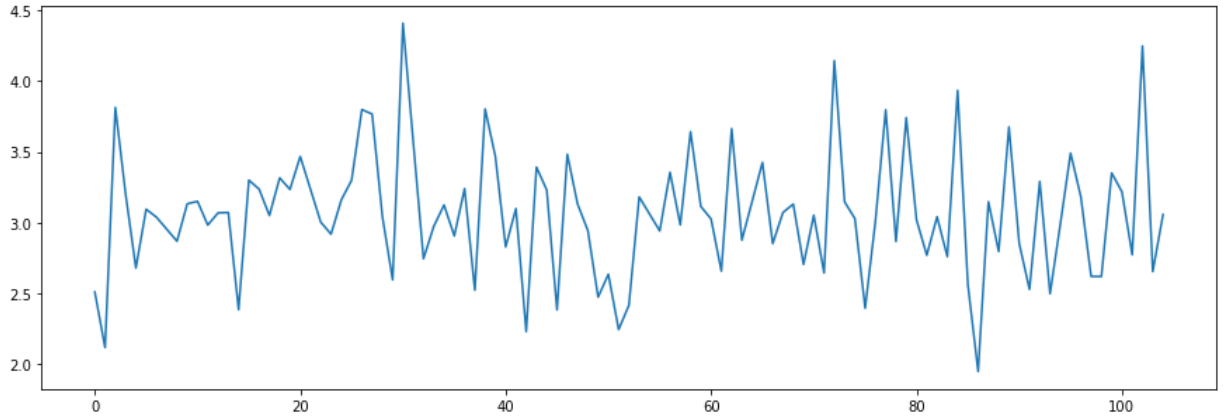
Let us visualize these missing values in a plot for better understanding as shown below.

In [9]:
```
plt.figure(figsize=(15,5))
sepalbfplot = plt.plot(data.sepal_width)
```



As there are very less missing values and the datavalues in the feature **"sepal_width"** are numerical,these missing values can be handled by either dropping them or replacing them with the mean of the variable **"sepal_width"** into a new variable **"new_sepal_width".**

In [10]:
```
#dropping the missing values from the variable sepal_width
plt.figure(figsize=(15,5))
sepaldrpplot = plt.plot(data.sepal_width.dropna())
```



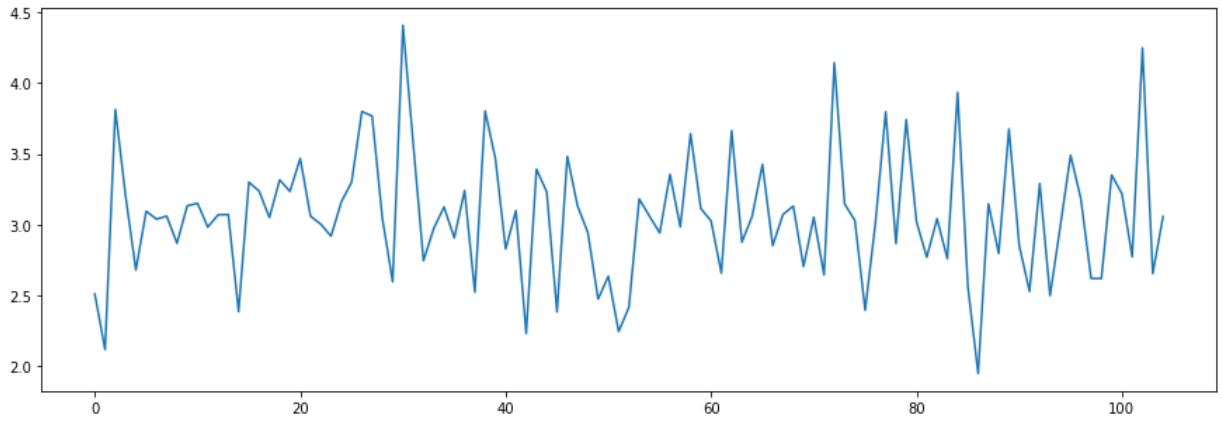
It can be visualised from the above plot that the missing values of the feature **"sepal_width"** have been handled by dropping them in this case.

In [11]:
```
#calculating the mean of the variable "sepal_width"
sepmean = data.sepal_width.mean()
sepmean
```

Out[11]:
```
3.0590829538769277
```

In [12]:
```
#replacing the missing values with mean of the variable
data['new_sepal_width'] = data.sepal_width.fillna(sepmean)
```

In [13]:
```
plt.figure(figsize=(15,5))
sepalaftplot = plt.plot(data.new_sepal_width)
```



It can be visualized from the above plot that all the missing values have been handled by replacing them with the mean.

petal_length :

The rows with missing data in the column "petal_length" are:

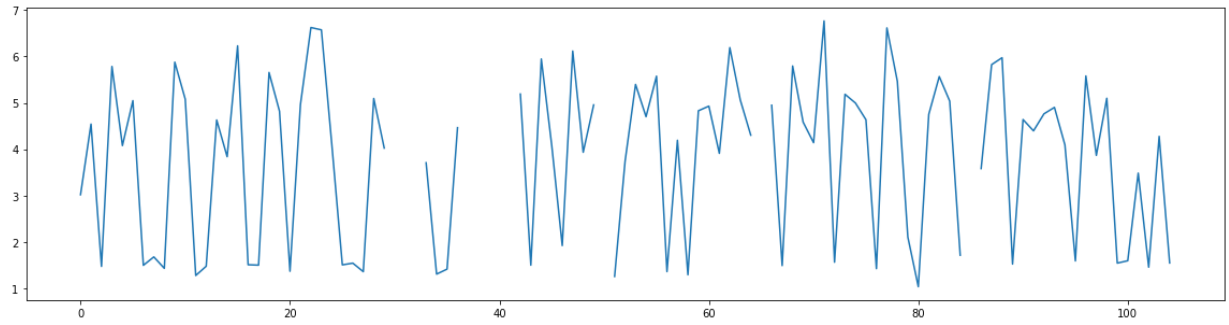
In [14]:
```
data[data.petal_length.isnull()]
```

Out[14]:

	sepal_length	sepal_width	petal_length	petal_width	species	new_sepal_width
30	5.673096	4.409565	NaN	0.370518	Iris-setosa	4.409565
32	5.847160	2.743619	NaN	0.748681	Iris-versicolor	2.743619
37	6.271780	2.521065	NaN	1.896626	Iris-virginica	2.521065
39	5.040516	3.466344	NaN	0.314548	Iris-setosa	3.466344
41	4.496342	3.098270	NaN	0.242853	Iris-setosa	3.098270
50	5.817283	2.633800	NaN	1.141347	Iris-versicolor	2.633800
65	6.235536	3.425253	NaN	2.423053	Iris-virginica	3.425253
85	5.911822	2.560512	NaN	1.766513	Iris-virginica	2.560512

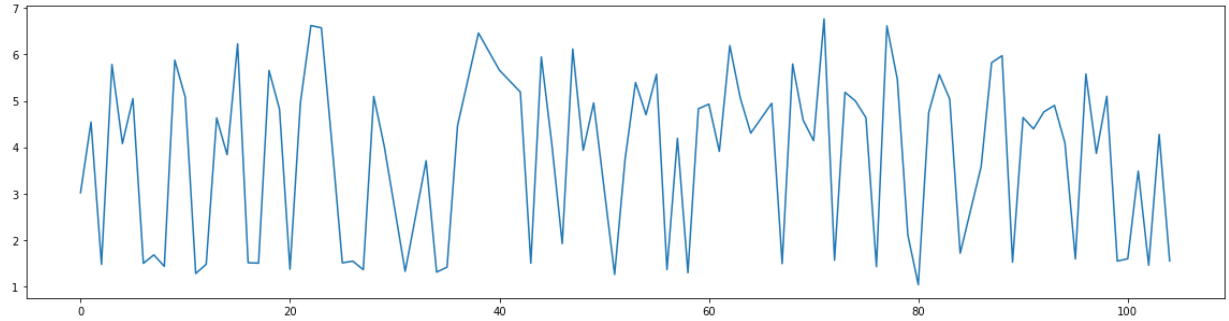
Let us visualize the above missing values in a plot for better understanding as shown.

In [15]:
```
plt.figure(figsize=(20,5))
petalbfplot = plt.plot(data.petal_length)
```



As the missing values are very less in number and the data values in the feature **"petal_length"** are numerical, these missing values can be handled by either dropping them or replacing them with the mean of the variable **"petal_length"** into a new variable **"new_petal_length".**

In [16]:
```
#dropping the missing values from the variable petal_length
plt.figure(figsize=(20,5))
petaldrpplot = plt.plot(data.petal_length.dropna())
```



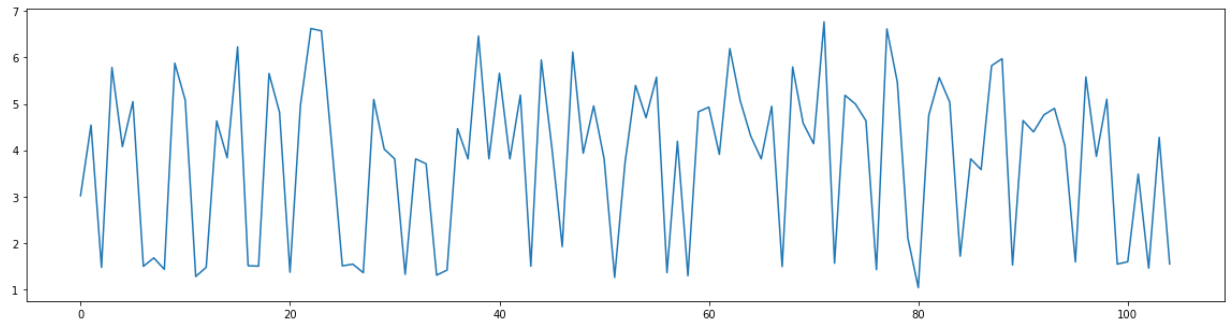
It can be visualised from the above plot that the missing values of the variable **"petal_length"** have been handled by dropping them in this case.

In [17]:
```
#calculating the mean of the variable "petal_length"
ptlmean = data.petal_length.mean()
ptlmean
```

Out[17]:
```
3.812369746505283
```

In [18]:
```
#replacing the missing values with mean of the variable
data['new_petal_length'] = data.petal_length.fillna(ptlmean)
```

In [19]:
```
plt.figure(figsize=(20,5))
petalaftplot = plt.plot(data.new_petal_length)
```



It can be visualized from the above plot that all the missing values of the variable **"petal_length"** have been handled by replacing them with the mean.

petal_width:

It can be observerd from the statistical details that there exists some negative values for the data in the feature **"petal_width"** which is due to some noise. This must be handled and we are replacing these negatives values with mean of the feature **"petal_width"** here.

In [20]:
```
#getting the index numbers of the negative data values in petal_width feature
ne_val=[]
ne_val=data[data.petal_width<0].index.values
print("The rows with negative data in the column petal_width are "+str(ne_val))
```

```
The rows with negative data in the column petal_width are [ 6 67]
```

In [21]:
```
#calculating mean to replace the negative values
mnpetal = data['petal_width'].mean()
mnpetal
```

Out[21]:
```
1.199708380963351
```

In [22]:
```
# replacing the negative values with the mean of the feature
for x in ne_val:
    data.iloc[x,data.columns.get_loc('petal_width')]=mnpetal
```

In [23]:
```
# cross-checking if the values are replaced by mean
for x in ne_val:
    print(data.iloc[x,data.columns.get_loc('petal_width')])
```

```
1.199708380963351
1.199708380963351
```

In [24]:
```
data.describe()
```

Out[24]:

	sepal_length	sepal_width	petal_length	petal_width	new_sepal_width	new_petal_length
count	105.000000	101.000000	97.000000	105.000000	105.000000	105.000000
mean	5.858909	3.059083	3.812370	1.223652	3.059083	3.812370
std	0.861638	0.455116	1.793489	0.767269	0.446278	1.723128
min	4.344007	1.946010	1.033031	0.020731	1.946010	1.033031
25%	5.159145	2.768688	1.545136	0.343669	2.771731	1.588587
50%	5.736104	3.049459	4.276817	1.331797	3.056142	4.089166
75%	6.435413	3.290318	5.094427	1.817211	3.239682	5.062244
max	7.795561	4.409565	6.768611	2.603123	4.409565	6.768611

From the above statistical details of the dataset, we can now say that the negative values in the feature **"petal_width"** have been handled successfully by replacing them with mean as a part of data cleaning.

COMPARISON OF THE CLASSIFICATION MODEL IN THE CASES WHEN MISSING VALUES ARE REPLACED BY APPROXIMATIONS AND WHEN MISSING VALUES ARE DROPPED:

Finding accuracy of the model when missing values are replaced with mean

In [25]:
```
data.isnull().sum()
```

Out[25]:
```
sepal_length        0
sepal_width         4
petal_length        8
petal_width         0
species             0
new_sepal_width     0
new_petal_length    0
dtype: int64
```

In [26]:
```
features_columns = ['sepal_length','new_sepal_width','new_petal_length','petal_wi
X = data[features_columns].values
Y = data['species'].values
```

In [27]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.2, random
```

In [28]:
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
```

In [29]:
```
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, Y_train)
Y_pred = knn.predict(X_test)
accuracy_rate = accuracy_score(Y_test, Y_pred)*100
print('The accuracy rate is '+ str(round(accuracy_rate))+' %.')
```

```
The accuracy rate is 86 %.
```

Finding accuracy of the model when missing values are dropped

In [30]:
```
data_dropped = data.dropna()
```

In [31]:
```
data_dropped.isnull().sum()
```

Out[31]:
```
sepal_length        0
sepal_width         0
petal_length        0
petal_width         0
species             0
new_sepal_width     0
new_petal_length    0
dtype: int64
```

In [32]:
```
features_columns1 = ['sepal_length','sepal_width','petal_length','petal_width']
X1 = data_dropped[features_columns1].values
Y1 = data_dropped['species'].values
```

In [33]:
```
from sklearn.model_selection import train_test_split
X1_train, X1_test, Y1_train, Y1_test = train_test_split(X1, Y1, test_size = 0.2,
```

In [34]:
```
knn1 = KNeighborsClassifier(n_neighbors=5)
knn1.fit(X1_train, Y1_train)
Y1_pred = knn.predict(X1_test)
accuracy_rate1 = accuracy_score(Y1_test, Y1_pred)*100
print('The accuracy rate is '+ str(round(accuracy_rate1))+' %.')
```

```
The accuracy rate is 100 %.
```

In [35]:
```
print("Accuracy of the classification model when dropping the missing values is '
print("Accuracy of the classification model when using replacement with approxima
```

```
Accuracy of the classification model when dropping the missing values is  100
%.
Accuracy of the classification model when using replacement with approximations
is  86 %.
```

HENCE WE CHOSE THE METHOD OF ELIMINATING THE MISSING VALUES FROM THE DATASET FOR A BETTER ACCURACY OF THE CLASSIFICATION MODEL !!

In [36]:
```
data_dropped=data_dropped.drop(['new_sepal_width','new_petal_length'],axis=1)
```

CLASSIFYING THE DATA USING KNN CLASSIFIER

In [37]:
```
data_dropped.to_csv('iris_dataset_cleaned.csv',index=False)
```

At this stage, the data is now free from missing values and is stored in a new CSV file named **"iris_dataset_cleaned.csv"** and the data in this new CSV file is used for the questions **CM2 and CM3**.

HERE WE ARE CONSIDERING THE DATA FROM THE CSV FILE 'cleaned_iris_dataset.csv' THAT IS FREE FROM OUTLIERS WHICH WAS PERFORMED IN THE QUESTION CM3.

Loading the cleaned dataset which is free from outliers :

In [38]:
```
dataknn = pd.read_csv('cleaned_iris_dataset.csv')
```

Displaying properties of the dataset for further processing

In [39]:
```
dataknn.info()
dataknn.head()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 93 entries, 0 to 92
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   sepal_length  93 non-null     float64
 1   sepal_width   93 non-null     float64
 2   petal_length  93 non-null     float64
 3   petal_width   93 non-null     float64
 4   species       93 non-null     object 
dtypes: float64(4), object(1)
memory usage: 3.8+ KB
```

Out[39]:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.045070	2.508203	3.018024	1.164924	Iris-versicolor
1	6.325517	2.498496	4.542052	1.413651	Iris-versicolor
2	5.257497	3.673501	1.470660	0.395348	Iris-setosa
3	6.675168	3.201700	5.785461	2.362764	Iris-virginica
4	5.595237	2.678166	4.077750	1.369266	Iris-versicolor

In [40]:
```
dataknn.tail()
```

Out[40]:

	sepal_length	sepal_width	petal_length	petal_width	species
88	4.874848	3.217348	1.592887	0.123588	Iris-setosa
89	5.564197	2.771731	3.483588	1.074754	Iris-versicolor
90	5.548047	3.673501	1.453466	0.214527	Iris-setosa
91	5.510482	2.652867	4.276817	1.298032	Iris-versicolor
92	4.538713	3.056142	1.545136	0.241424	Iris-setosa

In [41]:
```
dataknn.describe()
```

Out[41]:

	sepal_length	sepal_width	petal_length	petal_width
count	93.000000	93.000000	93.000000	93.000000
mean	5.867894	3.054063	3.808118	1.236858
std	0.892271	0.358692	1.811399	0.770872
min	4.344007	2.498496	1.033031	0.020731
25%	5.152435	2.794790	1.541564	0.343669
50%	5.636744	3.049459	4.192791	1.369266
75%	6.478961	3.239682	5.098860	1.837925
max	7.795561	3.673501	6.768611	2.603123

In [42]:
```
dataknn.shape
```

Out[42]:
```
(93, 5)
```

In [43]:
```
features_columns2 = ['sepal_length','sepal_width','petal_length','petal_width']
X2 = dataknn[features_columns2].values
Y2 = dataknn['species'].values
```

In [44]:
```
from sklearn.model_selection import train_test_split
X2_train, X2_test, Y2_train, Y2_test = train_test_split(X2, Y2, test_size = 0.2,
```

Training the learning model with classifier's default parameters:

In [45]:
```
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier()

# Fitting the model
classifier.fit(X2_train, Y2_train)

# Predicting the Test set results
Y2_pred = classifier.predict(X2_test)
```

Testing for accuracy of the trained model on the test data:

In [46]:
```
accuracy_rate2 = accuracy_score(Y2_test, Y2_pred)*100
print('The accuracy rate of our KNN model with classifier\'s default parameters i
```

```
The accuracy rate of our KNN model with classifier's default parameters is 100
%.
```

Testing the model with the validation set for k= {1, 5, 10, 15, 20, 25, 30, 35}

In our next step, we are performing cross validation test on our trained classification model, with the validation set which is 20% of the data.

This validation data is obtained by taking the value of "n_splits" as 5 in the "KFold" function and using that for cross validation with given set of K values.

This can be used to find the best fit K value, that gives a good accuracy rate for our data classification model.

In [47]:
```
from sklearn.model_selection import cross_val_score,KFold

kf=KFold(n_splits=5,random_state=275,shuffle=True)

listk=[1, 5, 10, 15, 20, 25, 30, 35]

for i in listk:

    knn2 = KNeighborsClassifier(n_neighbors=i)
    score2 = cross_val_score(knn2, X2_train, Y2_train, cv=kf, scoring='accuracy')
    #print(score2)
    accuracy_rate3 = (score2.mean())*100
    print('Accuracy of our model at k='+str(i)+' is equal to '+ str(round(accurac
```

```
Accuracy of our model at k=1 is equal to 89 %.
Accuracy of our model at k=5 is equal to 92 %.
Accuracy of our model at k=10 is equal to 92 %.
Accuracy of our model at k=15 is equal to 90 %.
Accuracy of our model at k=20 is equal to 86 %.
Accuracy of our model at k=25 is equal to 89 %.
Accuracy of our model at k=30 is equal to 86 %.
Accuracy of our model at k=35 is equal to 70 %.
```