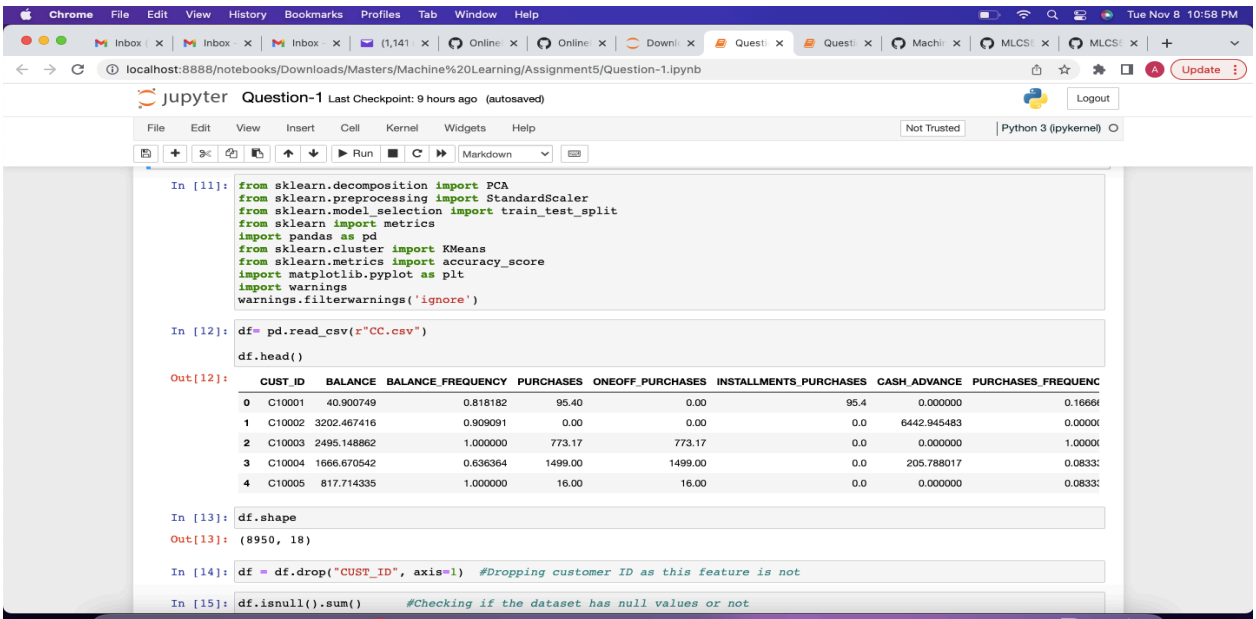
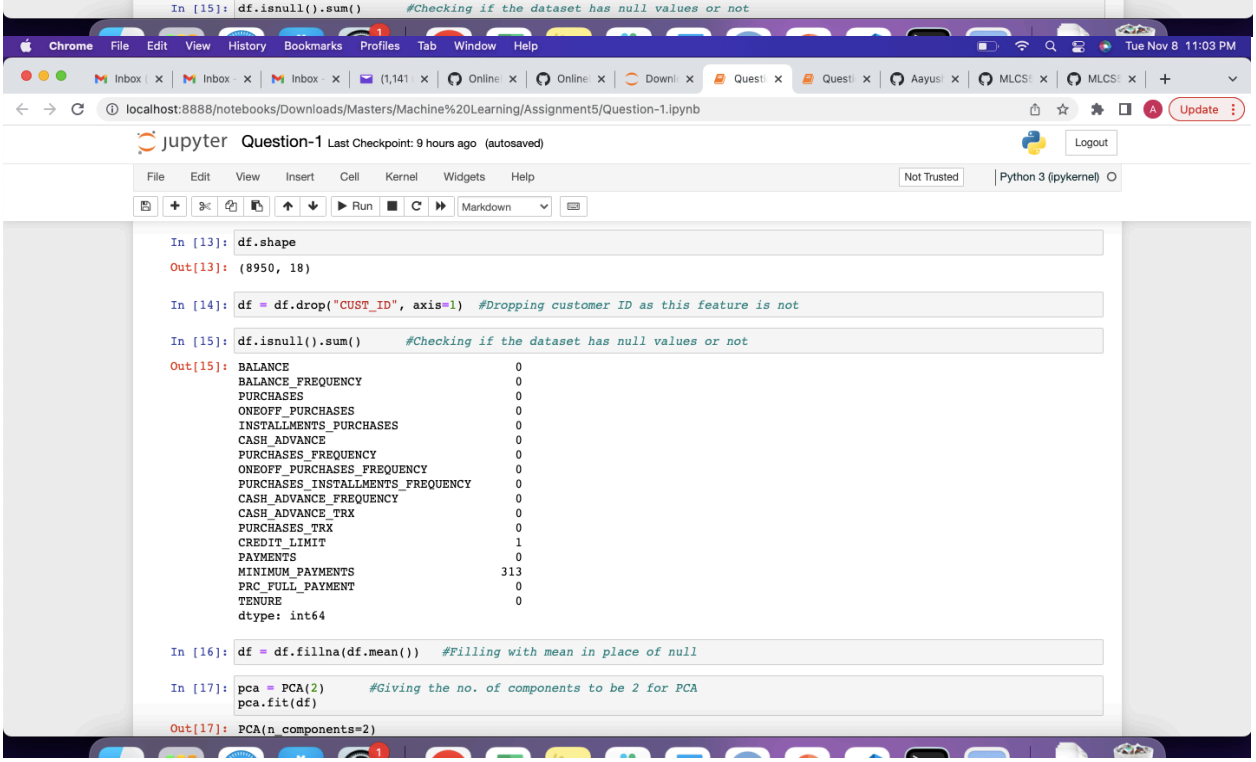
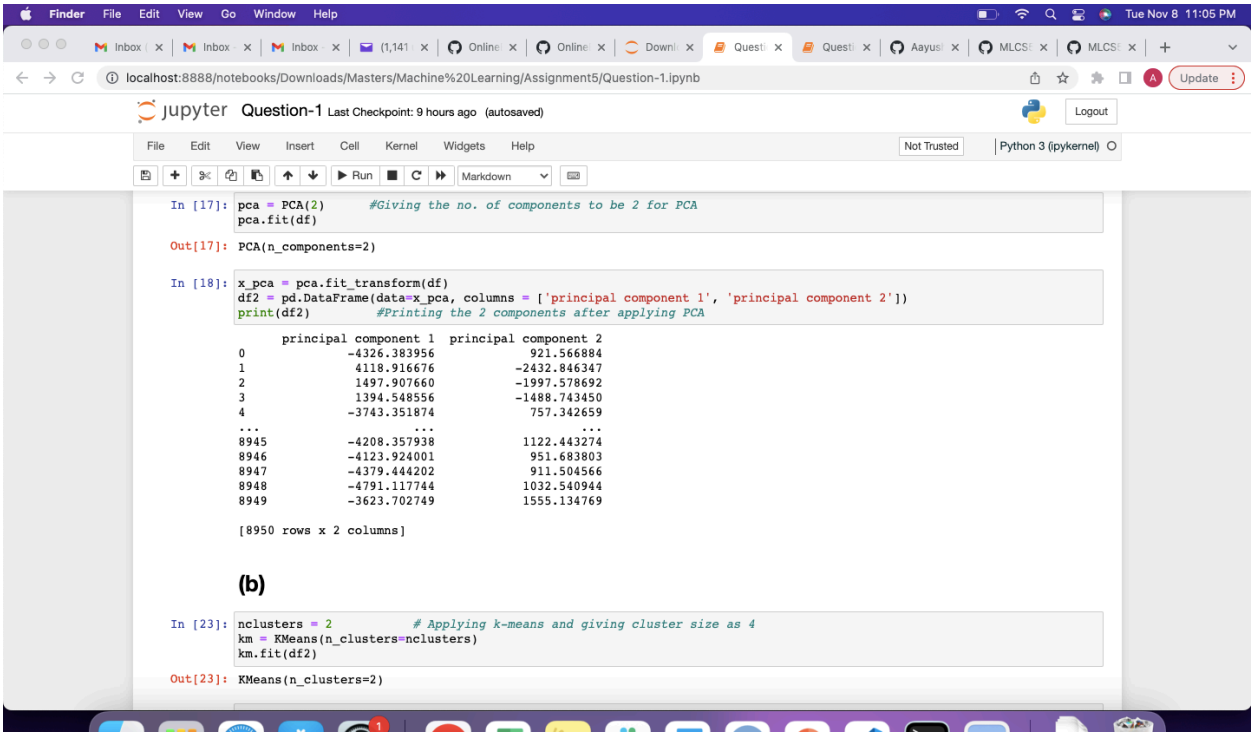
Question-1

For Question 1, I have imported the CC dataset and then imported all the necessary libraries to carry out various functions. Firstly, I checked the total no. of records and the total no. of attributes present in my dataset. Then I checked for null values and since there were null values present, I filled them with the mean of the column and also dropped the column which was irrelevant.





As you can see there are 755 attributes, hence I applied PCA to preprocess the data and reduce the no. of attributes.

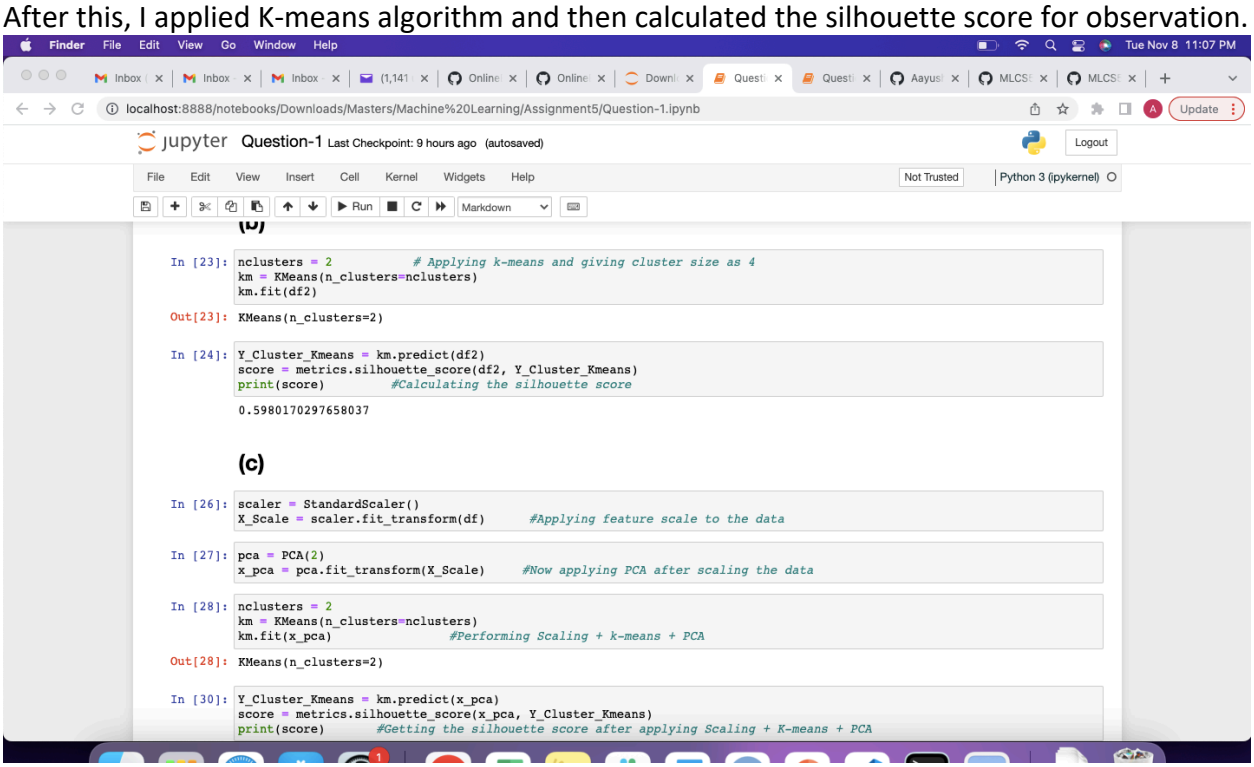


```
In [17]: pca = PCA(2)      #Giving the no. of components to be 2 for PCA
         pca.fit(df)

Out[17]: PCA(n_components=2)

In [18]: x_pca = pca.fit_transform(df)
         df2 = pd.DataFrame(data=x_pca, columns = ['principal component 1', 'principal component 2'])
         print(df2)        #Printing the 2 components after applying PCA

         principal component 1  principal component 2
0               -4326.383956             921.566884
1                4118.916676           -2432.846347
2                1497.907660           -1997.578692
3                1394.548556           -1488.743450
4               -3743.351874             757.342659
...                      ...                    ...
8945            -4208.357938            1122.443274
8946            -4123.924001             951.683803
8947            -4379.444202             911.504566
8948            -4791.117744            1032.540944
8949            -3623.702749            1555.134769

[8950 rows x 2 columns]
```

(b)

```
In [23]: nclusters = 2           # Applying k-means and giving cluster size as 4
         km = KMeans(n_clusters=nclusters)
         km.fit(df2)

Out[23]: KMeans(n_clusters=2)
```

After this, I applied K-means algorithm and then calculated the silhouette score for observation.



```
In [23]: nclusters = 2           # Applying k-means and giving cluster size as 4
         km = KMeans(n_clusters=nclusters)
         km.fit(df2)

Out[23]: KMeans(n_clusters=2)

In [24]: Y_Cluster_Kmeans = km.predict(df2)
         score = metrics.silhouette_score(df2, Y_Cluster_Kmeans)
         print(score)        #Calculating the silhouette score

         0.5980170297658037
```

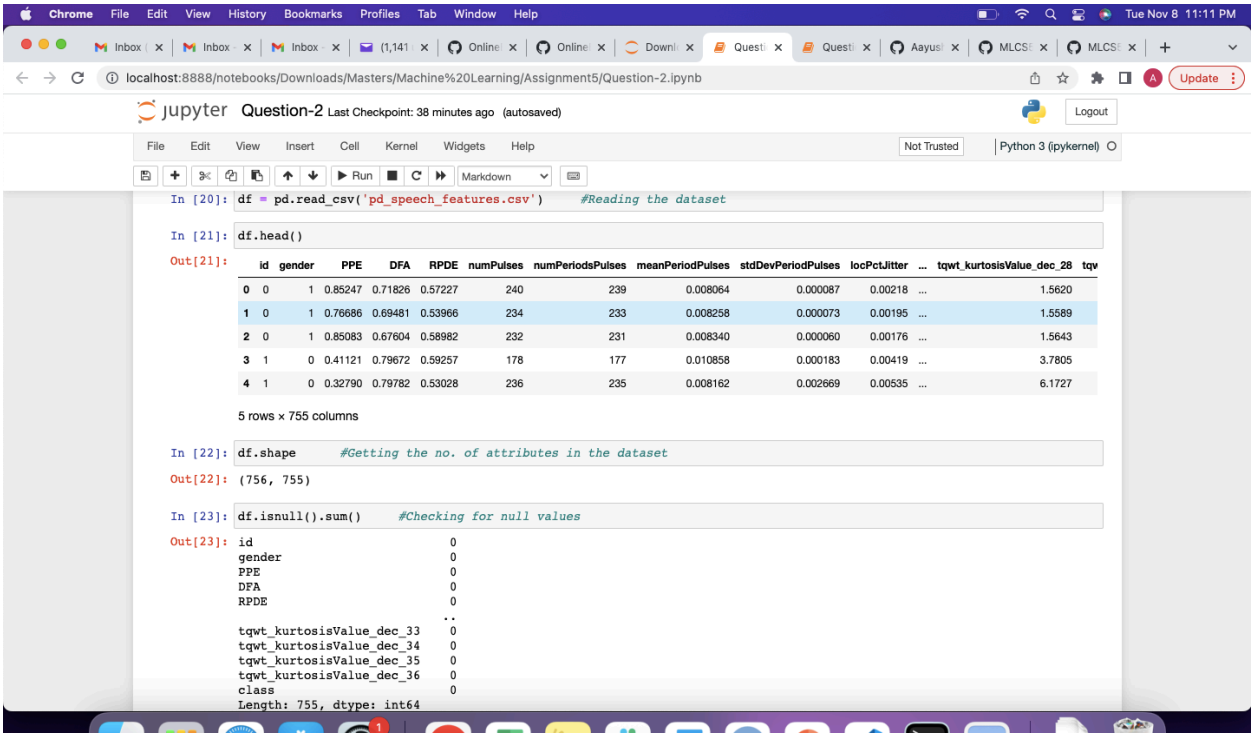
(c)

```
In [26]: scaler = StandardScaler()
         X_Scale = scaler.fit_transform(df)      #Applying feature scale to the data

In [27]: pca = PCA(2)
         x_pca = pca.fit_transform(X_Scale)      #Now applying PCA after scaling the data

In [28]: nclusters = 2
         km = KMeans(n_clusters=nclusters)
         km.fit(x_pca)                  #Performing Scaling + k-means + PCA

Out[28]: KMeans(n_clusters=2)

In [30]: Y_Cluster_Kmeans = km.predict(x_pca)
         score = metrics.silhouette_score(x_pca, Y_Cluster_Kmeans)
         print(score)       #Getting the silhouette score after applying Scaling + K-means + PCA
```

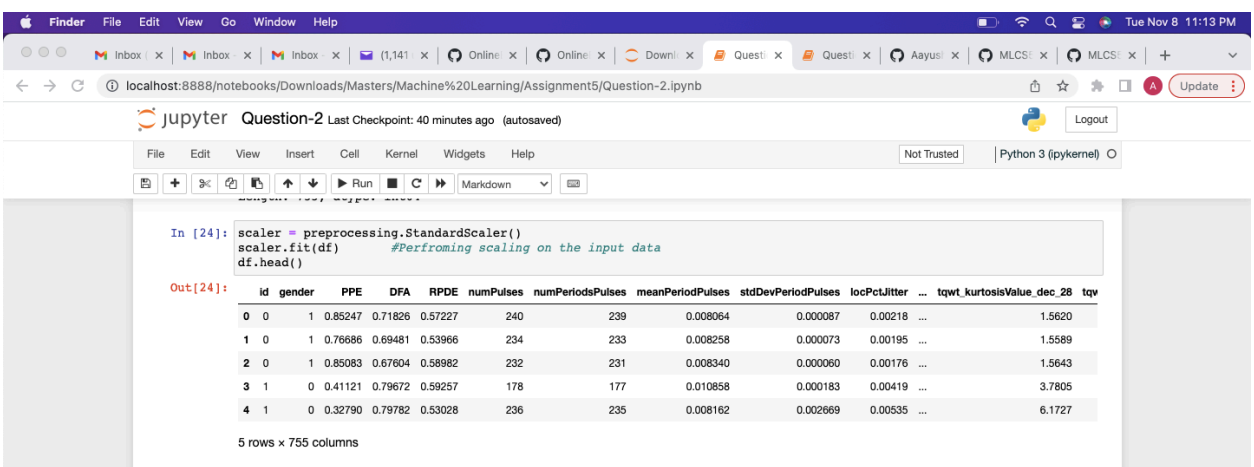
Finally I scaled the data and then performed PCA and then K-means to get the observation. The scaling didn't work better for this particular instance as I received a low silhouette score.

Question-2

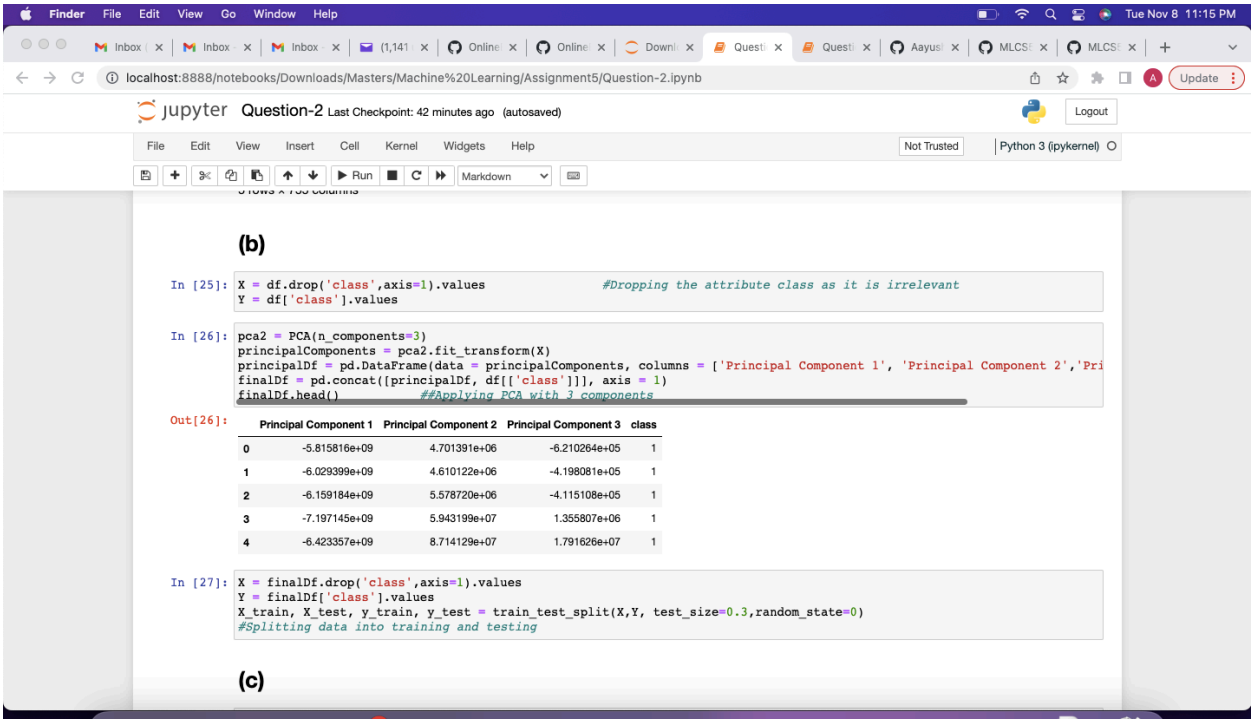
For this question, I worked with the speech features data set. As always, I looked at the total no. of records and attributes and also checked if there are any null values in the data set, which wasn't the case at this instance.



After loading the dataset and checking for null, then I used scaling.



After scaling, I preprocessed the data using PCA and used 3 components and displayed.



(b)

In [25]:
```
X = df.drop('class',axis=1).values                #Dropping the attribute class as it is irrelevant
Y = df['class'].values
```

In [26]:
```
pca2 = PCA(n_components=3)
principalComponents = pca2.fit_transform(X)
principalDf = pd.DataFrame(data = principalComponents, columns = ['Principal Component 1', 'Principal Component 2','Pri
finalDf = pd.concat([principalDf, df[['class']]], axis = 1)
finalDf.head()            ##Applying PCA with 3 components
```

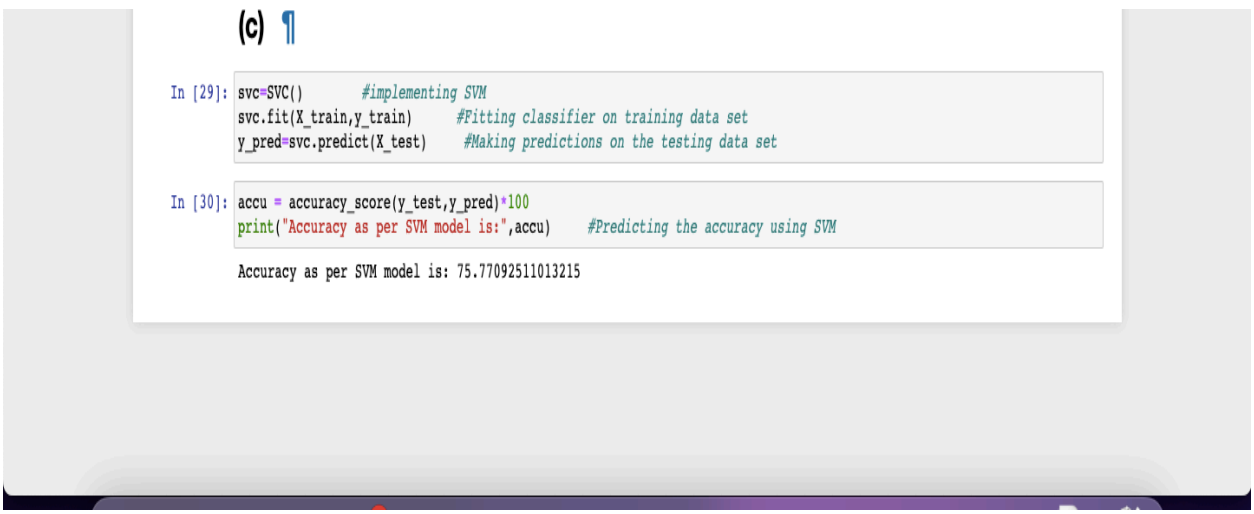
Out[26]:

	Principal Component 1	Principal Component 2	Principal Component 3	class
0	-5.815816e+09	4.701391e+06	-6.210264e+05	1
1	-6.029399e+09	4.610122e+06	-4.198081e+05	1
2	-6.159184e+09	5.578720e+06	-4.115108e+05	1
3	-7.197145e+09	5.943199e+07	1.355807e+06	1
4	-6.423357e+09	8.714129e+07	1.791626e+07	1

In [27]:
```
X = finalDf.drop('class',axis=1).values
Y = finalDf['class'].values
X_train, X_test, y_train, y_test = train_test_split(X,Y, test_size=0.3,random_state=0)
#Splitting data into training and testing
```

(c)

Finally, I split the data into training and testing sets and implemented SVM and calculated the accuracy of the model which came out to be 75.77%



(c) ¶

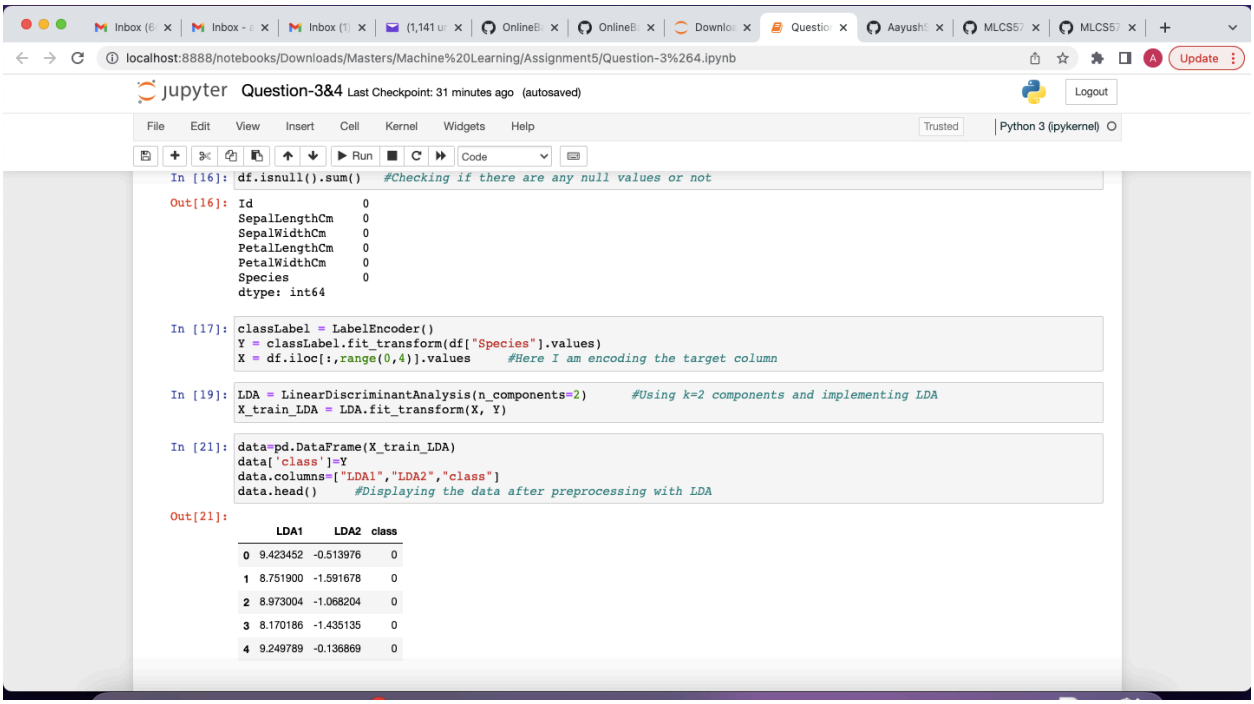
In [29]:
```
svc=SVC()        #implementing SVM
svc.fit(X_train,y_train)      #Fitting classifier on training data set
y_pred=svc.predict(X_test)      #Making predictions on the testing data set
```

In [30]:
```
accu = accuracy_score(y_test,y_pred)*100
print("Accuracy as per SVM model is:",accu)      #Predicting the accuracy using SVM
```

```
Accuracy as per SVM model is: 75.77092511013215
```

Question-3

For Question 3, I worked with the Iris data set. I again loaded the data, checked for total no. of attributes and records and the also checked for null values. Once all basic steps were done, I used LDA to pre-process the data as it had a decision attribute which hinted on the use of LDA for pre processing the data.



Finally I also applied PCA to the same data set for comparison purposes which didn't perform well as for data set having decision attribute, LDA works better.