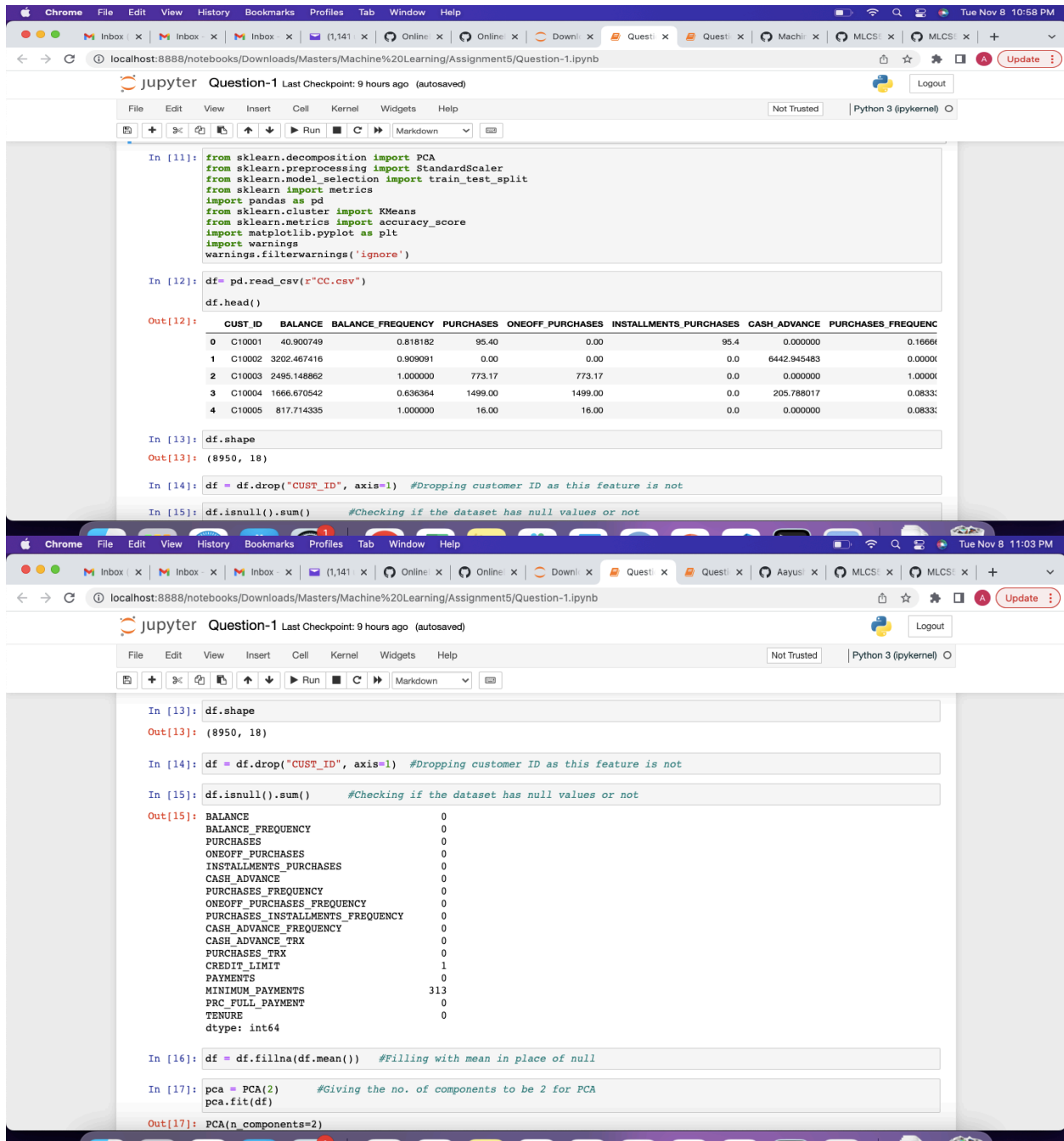


Question-1

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.



The image displays two screenshots of a Jupyter Notebook interface, showing the initial steps of data loading and preprocessing for a machine learning task.

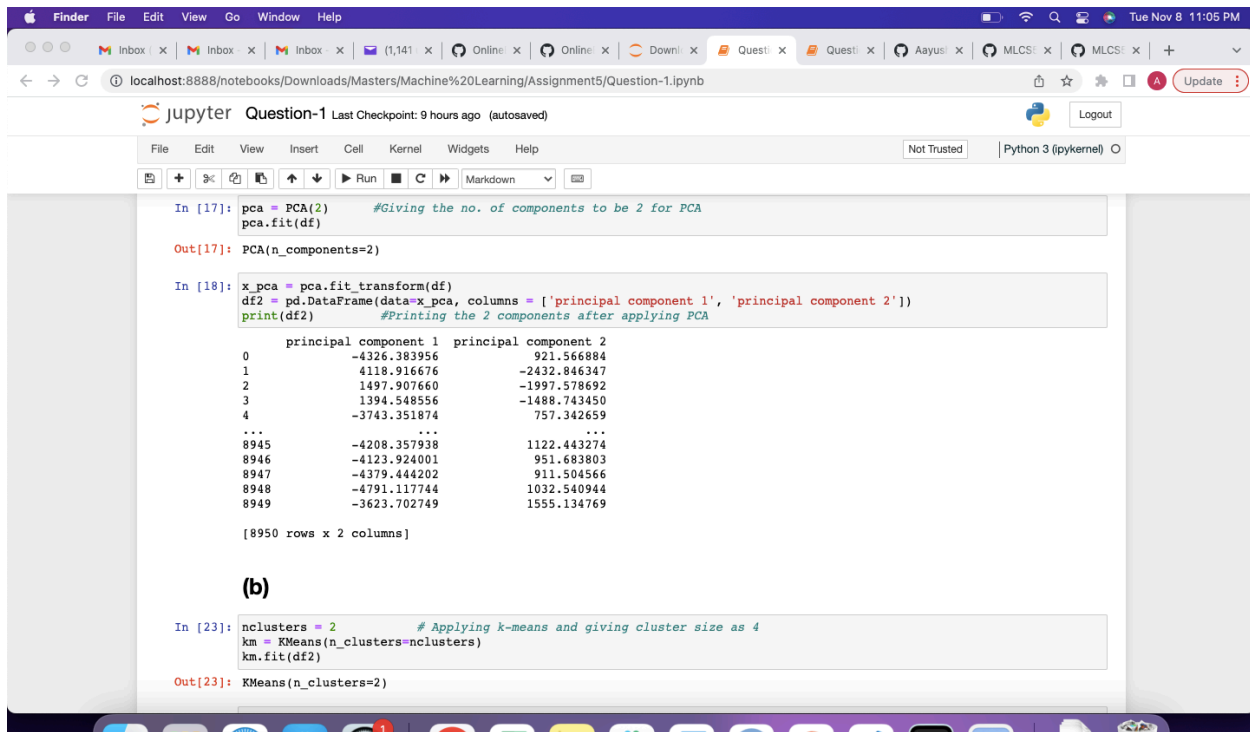
Top Screenshot: The notebook is titled "Question-1" and shows the first five code cells. The first cell imports necessary libraries: `from sklearn.decomposition import PCA`, `from sklearn.preprocessing import StandardScaler`, `from sklearn.model_selection import train_test_split`, `import pandas as pd`, `from sklearn.cluster import KMeans`, `from sklearn.metrics import accuracy_score`, `import matplotlib.pyplot as plt`, and `import warnings`. The second cell reads the dataset: `df = pd.read_csv("CC.csv")` and displays the first five rows of the dataset. The third cell shows the shape of the dataset: `df.shape` returns `(8950, 18)`. The fourth cell drops the `CUST_ID` column: `df = df.drop("CUST_ID", axis=1)`. The fifth cell checks for null values: `df.isnull().sum()`.

	CUST_ID	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	CASH_ADVANCE	PURCHASES_FREQUENCY
0	C10001	40.900749	0.818182	95.40	0.00	95.4	0.000000	0.166667
1	C10002	3202.467416	0.909091	0.00	0.00	0.0	6442.945483	0.000000
2	C10003	2495.148662	1.000000	773.17	773.17	0.0	0.000000	1.000000
3	C10004	1666.670542	0.636364	1499.00	1499.00	0.0	205.788017	0.083333
4	C10005	817.714335	1.000000	16.00	16.00	0.0	0.000000	0.083333

Bottom Screenshot: The notebook continues with the next three code cells. The sixth cell shows the shape of the dataset after dropping `CUST_ID`: `df.shape` returns `(8950, 18)`. The seventh cell checks for null values: `df.isnull().sum()`. The eighth cell fills the null values with the mean of the column: `df = df.fillna(df.mean())`. The ninth cell performs PCA: `pca = PCA(2)` and `pca.fit(df)`. The final output shows the PCA results: `Out[17]: PCA(n_components=2)`.

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	CASH_ADVANCE	PURCHASES_FREQUENCY	ONEOFF_PURCHASES_FREQUENCY	PURCHASES_INSTALLMENTS_FREQUENCY	CASH_ADVANCE_FREQUENCY	CASH_ADVANCE_TRX	PURCHASES_TRX	CREDIT_LIMIT	PAYMENTS	MINIMUM_PAYMENTS	PRC_FULL_PAYMENT	TENURE
0	40.900749	0.818182	95.40	0.00	95.4	0.000000	0.166667	0.000000	0.000000	0.000000	0.000000	0.000000	1	0	313	0	0
1	3202.467416	0.909091	0.00	0.00	0.0	6442.945483	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	1	0	313	0	0
2	2495.148662	1.000000	773.17	773.17	0.0	0.000000	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	1	0	313	0	0
3	1666.670542	0.636364	1499.00	1499.00	0.0	205.788017	0.083333	0.000000	0.000000	0.000000	0.000000	0.000000	1	0	313	0	0
4	817.714335	1.000000	16.00	16.00	0.0	0.000000	0.083333	0.000000	0.000000	0.000000	0.000000	0.000000	1	0	313	0	0

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



The screenshot shows a Jupyter Notebook interface with the following code and output:

```
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

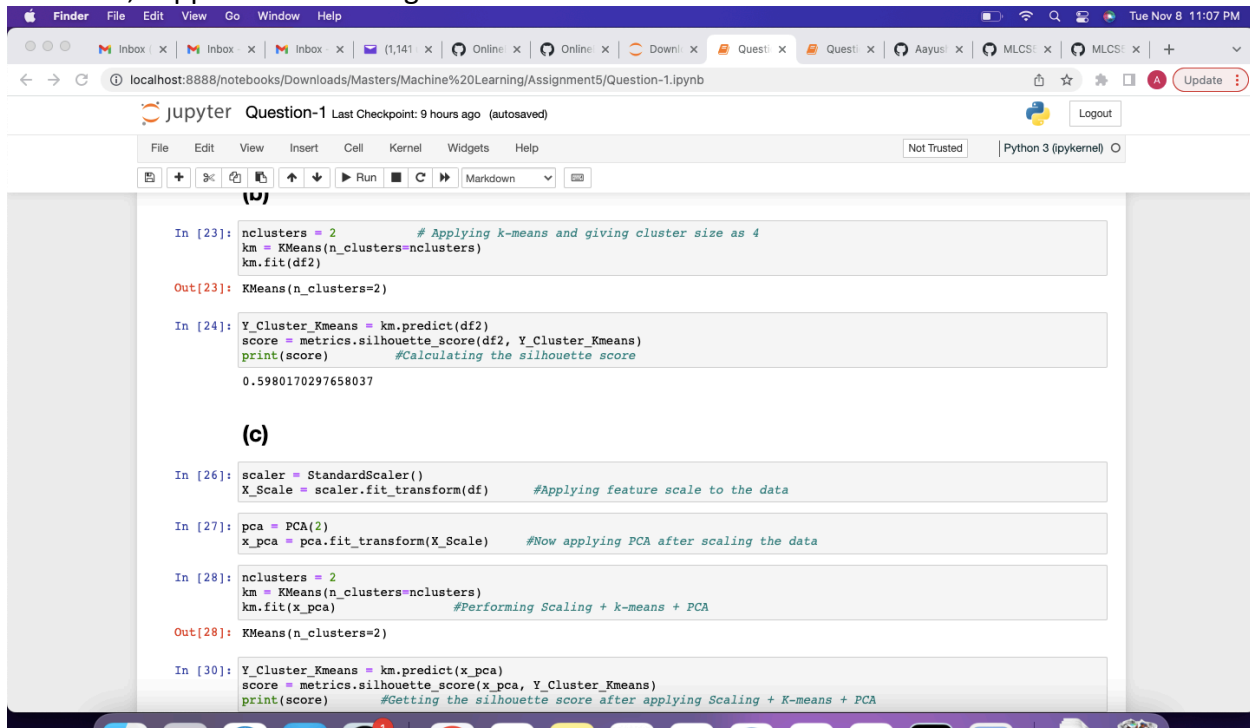
[8950 rows x 2 columns]

(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.



The screenshot shows a Jupyter Notebook interface with the following code and output:

```
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

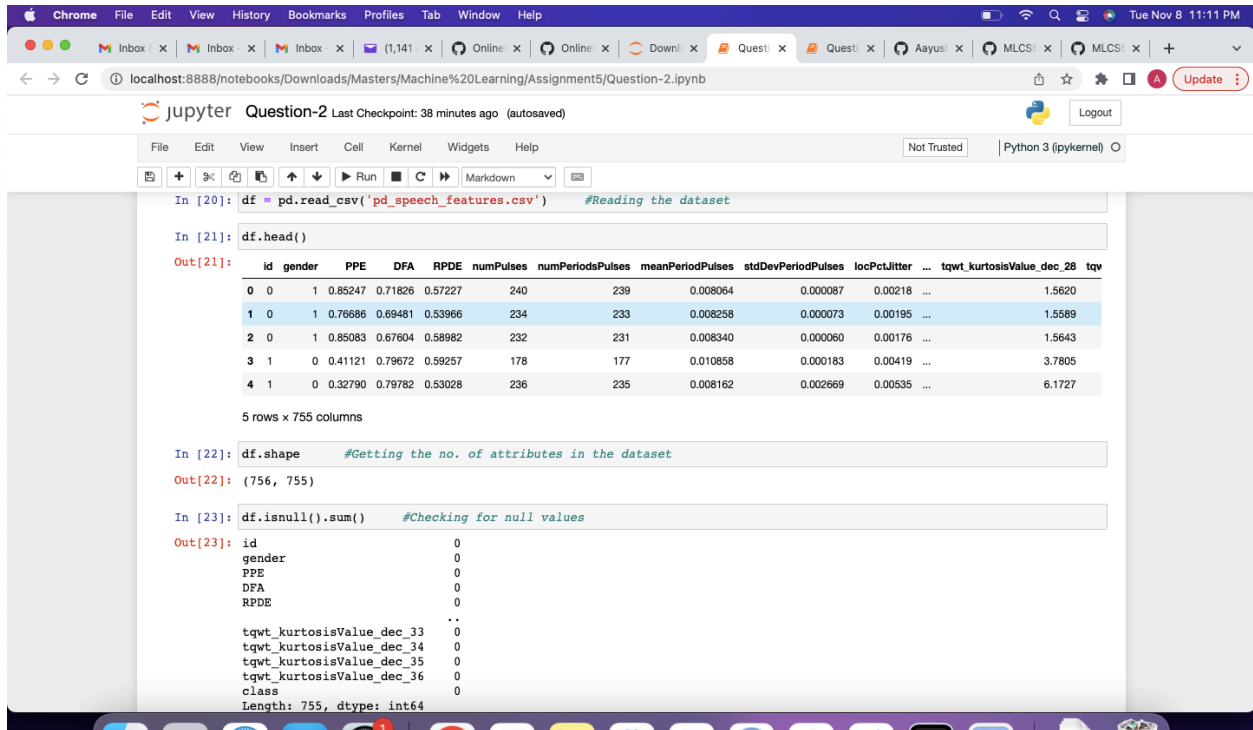
Out[28]: KMeans(n_clusters=2)

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.



The screenshot shows a Jupyter Notebook interface with the following code and output:

```
In [20]: df = pd.read_csv('pd_speech_features.csv') #Reading the dataset
```

```
In [21]: df.head()
```

```
Out[21]:
```

	id	gender	PPE	DFA	RPDE	numPulses	numPeriodsPulses	meanPeriodPulses	stdDevPeriodPulses	locPctJitter	...	twqt_kurtosisValue_dec_28	twqt_kurtosisValue_dec_33
0	0	1	0.85247	0.71826	0.57227	240	239	0.008064	0.000087	0.00218	...	1.5620	1.5589
1	0	1	0.76686	0.69481	0.53966	234	233	0.008258	0.000073	0.00195	...	1.5643	1.5643
2	0	1	0.85083	0.67604	0.58982	232	231	0.008340	0.000060	0.00176	...	3.7805	6.1727
3	1	0	0.41121	0.79672	0.59257	178	177	0.010858	0.000183	0.00419	...		
4	1	0	0.32790	0.79782	0.53028	236	235	0.008162	0.002669	0.00535	...		

5 rows x 755 columns

```
In [22]: df.shape #Getting the no. of attributes in the dataset
```

```
Out[22]: (756, 755)
```

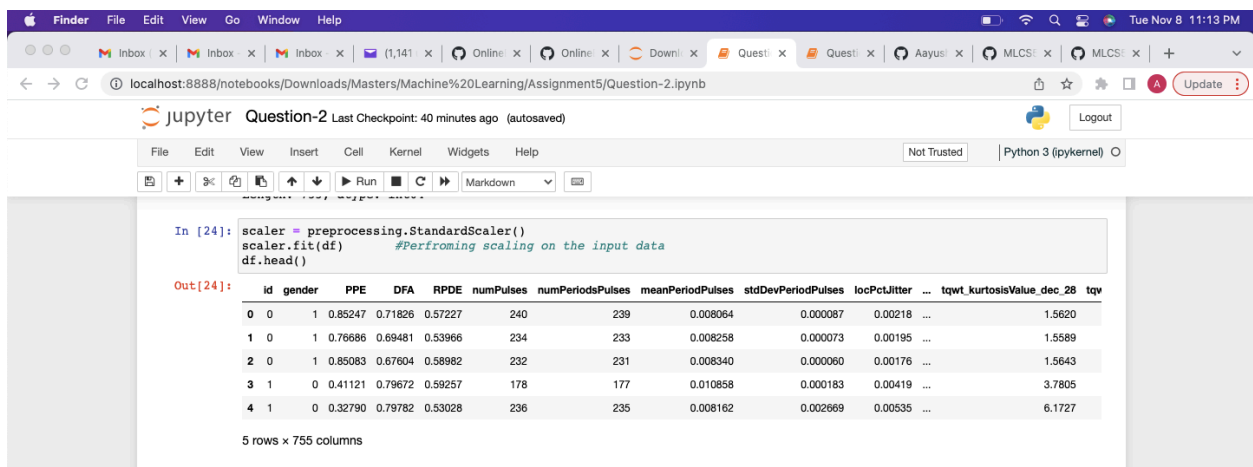
```
In [23]: df.isnull().sum() #Checking for null values
```

```
Out[23]:
```

	id	gender	PPE	DFA	RPDE	...	twqt_kurtosisValue_dec_33	twqt_kurtosisValue_dec_34	twqt_kurtosisValue_dec_35	twqt_kurtosisValue_dec_36	class
	0	0	0	0	0	...	0	0	0	0	0

Length: 755, dtype: int64

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



The screenshot shows the same Jupyter Notebook interface with the following code and output:

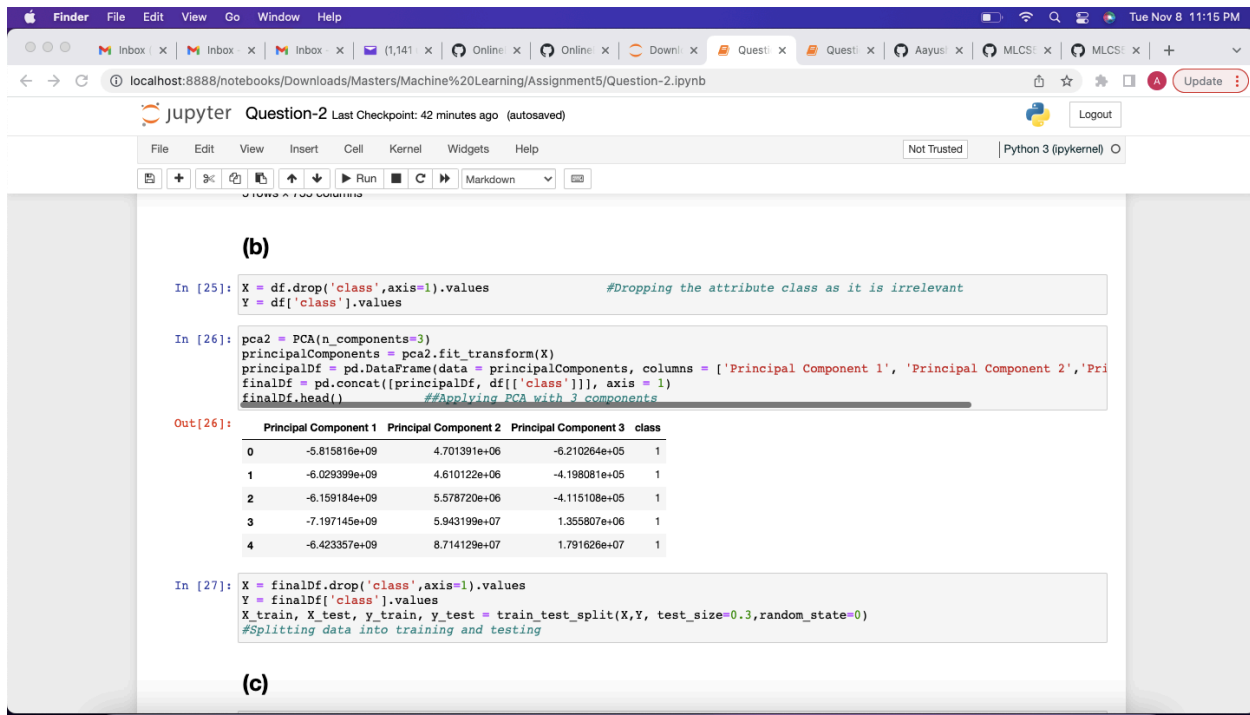
```
In [24]: scaler = preprocessing.StandardScaler()
scaler.fit(df) #Performing scaling on the input data
df.head()
```

```
Out[24]:
```

	id	gender	PPE	DFA	RPDE	numPulses	numPeriodsPulses	meanPeriodPulses	stdDevPeriodPulses	locPctJitter	...	twqt_kurtosisValue_dec_28	twqt_kurtosisValue_dec_33
0	0	1	0.85247	0.71826	0.57227	240	239	0.008064	0.000087	0.00218	...	1.5620	1.5589
1	0	1	0.76686	0.69481	0.53966	234	233	0.008258	0.000073	0.00195	...	1.5643	1.5643
2	0	1	0.85083	0.67604	0.58982	232	231	0.008340	0.000060	0.00176	...	3.7805	6.1727
3	1	0	0.41121	0.79672	0.59257	178	177	0.010858	0.000183	0.00419	...		
4	1	0	0.32790	0.79782	0.53028	236	235	0.008162	0.002669	0.00535	...		

5 rows x 755 columns

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



The screenshot shows a Jupyter Notebook window titled "Question-2" with a last checkpoint 42 minutes ago. The interface includes a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help) and a toolbar with icons for file operations, running cells, and markdown. The notebook content is as follows:

(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', 'Principal Component 3'], index = range(1,5))
         finalDf = pd.concat([principalDf, df[['class']]], axis = 1)
         finalDf.head()
```

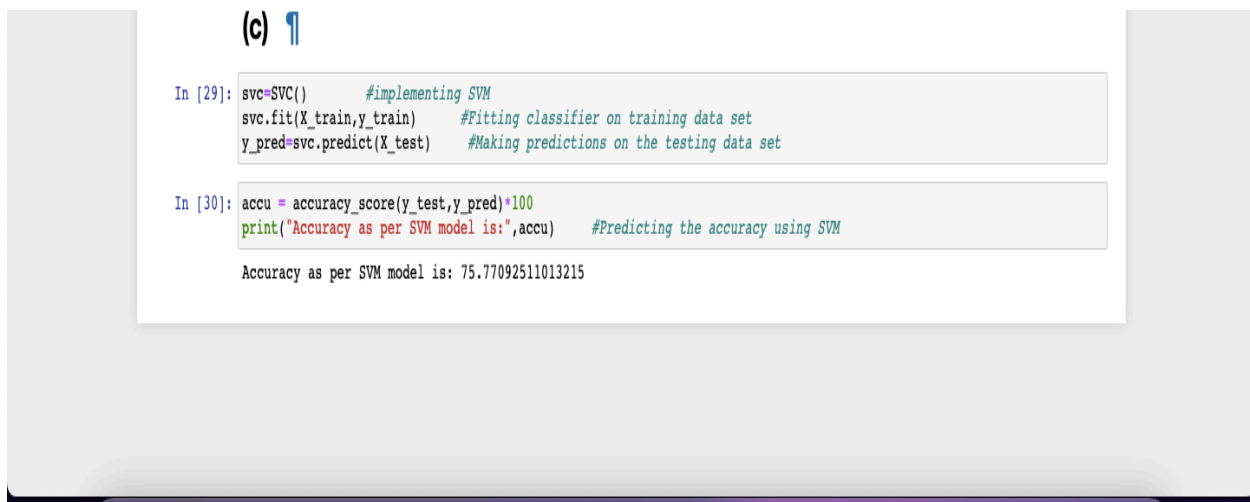
Out[26]:

	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%



The screenshot shows the continuation of the Jupyter Notebook. The code for implementing SVM and calculating accuracy is as follows:

(c)

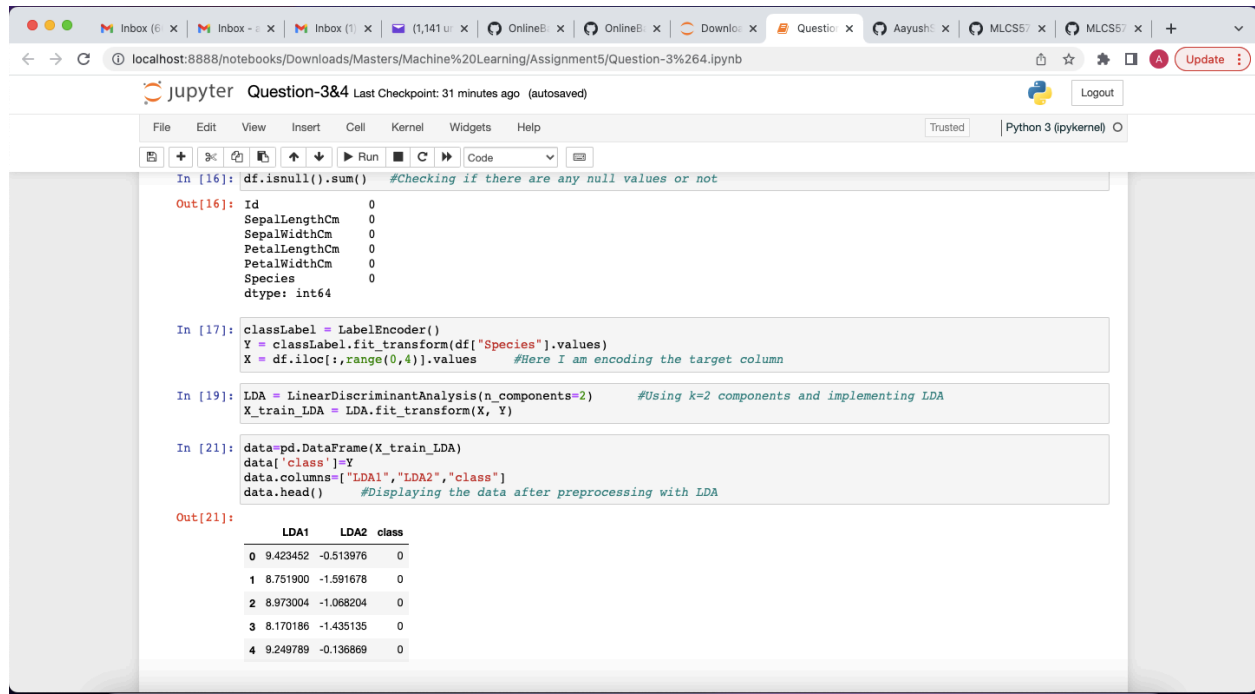
```
In [29]: svc=SVC()           #implementing SVM
         svc.fit(X_train,y_train)   #Fitting classifier on training data set
         y_pred=svc.predict(X_test) #Making predictions on the testing data set
```

```
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.



```
In [16]: df.isnull().sum() #Checking if there are any null values or not

Out[16]: Id 0
SepalLengthCm 0
SepalWidthCm 0
PetalLengthCm 0
PetalWidthCm 0
Species 0
dtype: int64

In [17]: classLabel = LabelEncoder()
Y = classLabel.fit_transform(df["Species"].values)
X = df.iloc[:,range(0,4)].values #Here I am encoding the target column

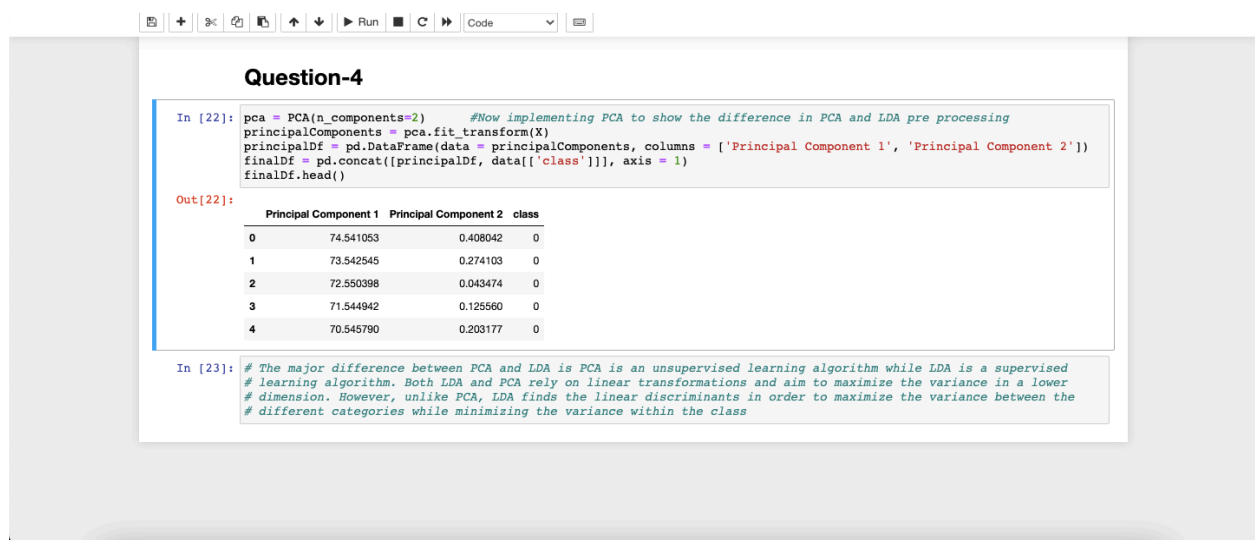
In [19]: LDA = LinearDiscriminantAnalysis(n_components=2) #Using k=2 components and implementing LDA
X_train_LDA = LDA.fit_transform(X, Y)

In [21]: data=pd.DataFrame(X_train_LDA)
data['class']=Y
data.columns=["LDA1","LDA2","class"]
data.head() #Displaying the data after preprocessing with LDA

Out[21]:
```

	LDA1	LDA2	class
0	9.423452	-0.513976	0
1	8.751900	-1.591678	0
2	8.973004	-1.068204	0
3	8.170186	-1.435135	0
4	9.249789	-0.136869	0

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.



```
In [22]: pca = PCA(n_components=2) #Now implementing PCA to show the difference in PCA and LDA pre processing
principalComponents = pca.fit_transform(X)
principalDf = pd.DataFrame(data = principalComponents, columns = ['Principal Component 1', 'Principal Component 2'])
finalDf = pd.concat([principalDf, data[['class']]], axis = 1)
finalDf.head()

Out[22]:
```

	Principal Component 1	Principal Component 2	class
0	74.541053	0.408042	0
1	73.542545	0.274103	0
2	72.550398	0.043474	0
3	71.544942	0.125560	0
4	70.545790	0.203177	0

```
In [23]: # The major difference between PCA and LDA is PCA is an unsupervised learning algorithm while LDA is a supervised
# learning algorithm. Both LDA and PCA rely on linear transformations and aim to maximize the variance in a lower
# dimension. However, unlike PCA, LDA finds the linear discriminants in order to maximize the variance between the
# different categories while minimizing the variance within the class
```