CSE 4020 - MACHINE LEARNING

Lab 29+30

Lab Assignment-4

Submitted by: Alokam Nikhitha(19BCE2555)

MLP

Question:

To train a Multi-Layer Perceptron (MLP) model to classify the network traffic record whether it is a normal or attack…

1. Read and parse the dataset.

2. Create Multi-Layer Perceptron Model (MLP)

3. Train and evaluate a Multi-Layer Perceptron (MLP) model

Dataset Used:

NSL KDD – Intrusion Detection Dataset
https://www.unb.ca/cic/datasets/nsl.html

Procedure:

-Using pandas, we first import the dataset into our workspace.

-Assign the column names to our dataset as it doesn't have one.

- **Pick out** and encode our **specific** variable.

- After encoding **the specific** variable, we **want** to dummy encode them **on the way to keep away from** ordinality **among** nominal **information**.

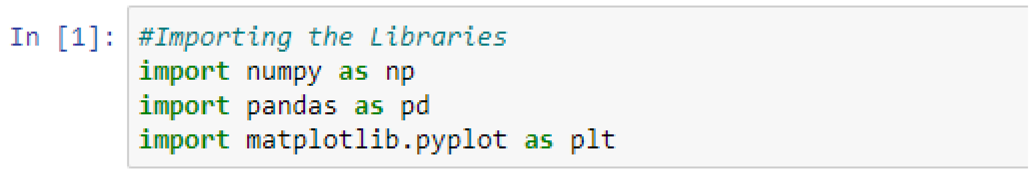
- We then **want** to re-assign our label **information**. All labels **different** than **ordinary** are assigned as attacks.

- We then **want** to divide the **schooling** set and **check** set **information** into set of **structured** attributes and **impartial** attributes.

- Next, we lay down the Multi-Layer Perceptron and **byskip** our **enter records** into **enter** layer of our neural network.

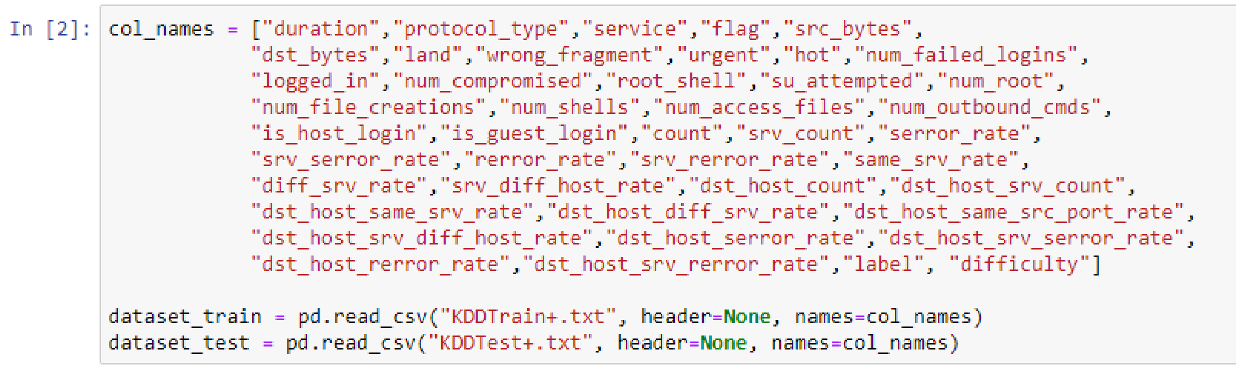
- Finally, we generate our **check** set **consequences** and evaluation metrices.

Code Snippets and Explanation:



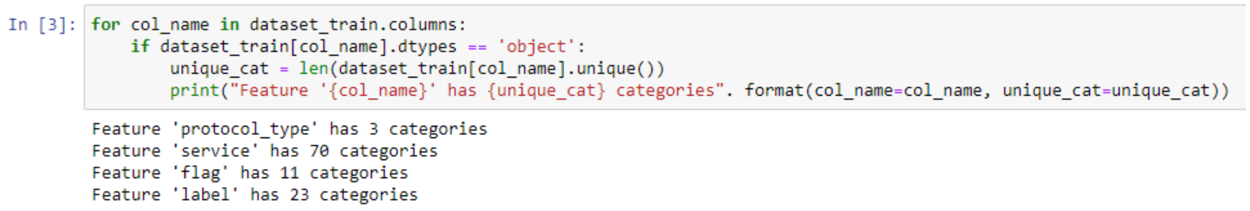
```
In [1]: #Importing the Libraries
        import numpy as np
        import pandas as pd
        import matplotlib.pyplot as plt
```

Here we are importing the necessary libraries in our workspace.

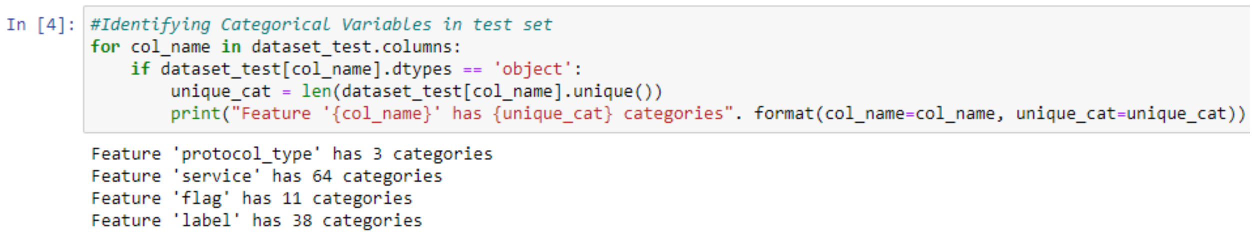


```
In [2]: col_names = ["duration","protocol_type","service","flag","src_bytes",
                     "dst_bytes","land","wrong_fragment","urgent","hot","num_failed_logins",
                     "logged_in","num_compromised","root_shell","su_attempted","num_root",
                     "num_file_creations","num_shells","num_access_files","num_outbound_cmds",
                     "is_host_login","is_guest_login","count","srv_count","serror_rate",
                     "srv_serror_rate","rerror_rate","srv_rerror_rate","same_srv_rate",
                     "diff_srv_rate","srv_diff_host_rate","dst_host_count","dst_host_srv_count",
                     "dst_host_same_srv_rate","dst_host_diff_srv_rate","dst_host_same_src_port_rate",
                     "dst_host_srv_diff_host_rate","dst_host_serror_rate","dst_host_srv_serror_rate",
                     "dst_host_rerror_rate","dst_host_srv_rerror_rate","label", "difficulty"]

        dataset_train = pd.read_csv("KDDTrain+.txt", header=None, names=col_names)
        dataset_test = pd.read_csv("KDDTest+.txt", header=None, names=col_names)
```

Here we're uploading the dataset into our workspace and are assigning them with the column names because it isn't always pre-blanketed in the given dataset.



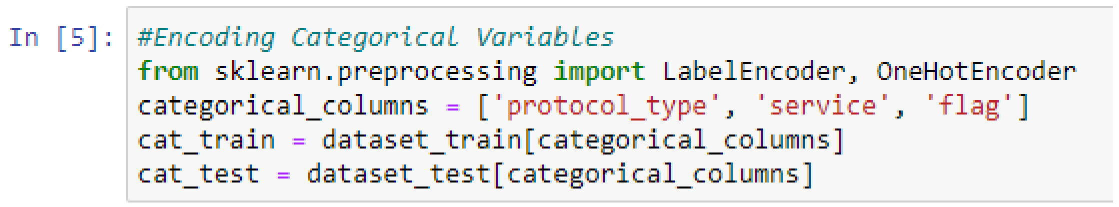
```
In [3]: for col_name in dataset_train.columns:
            if dataset_train[col_name].dtypes == 'object':
                unique_cat = len(dataset_train[col_name].unique())
                print("Feature '{col_name}' has {unique_cat} categories". format(col_name=col_name, unique_cat=unique_cat))

        Feature 'protocol_type' has 3 categories
        Feature 'service' has 70 categories
        Feature 'flag' has 11 categories
        Feature 'label' has 23 categories
```



```
In [4]: #Identifying Categorical Variables in test set
        for col_name in dataset_test.columns:
            if dataset_test[col_name].dtypes == 'object':
                unique_cat = len(dataset_test[col_name].unique())
                print("Feature '{col_name}' has {unique_cat} categories". format(col_name=col_name, unique_cat=unique_cat))

        Feature 'protocol_type' has 3 categories
        Feature 'service' has 64 categories
        Feature 'flag' has 11 categories
        Feature 'label' has 38 categories
```

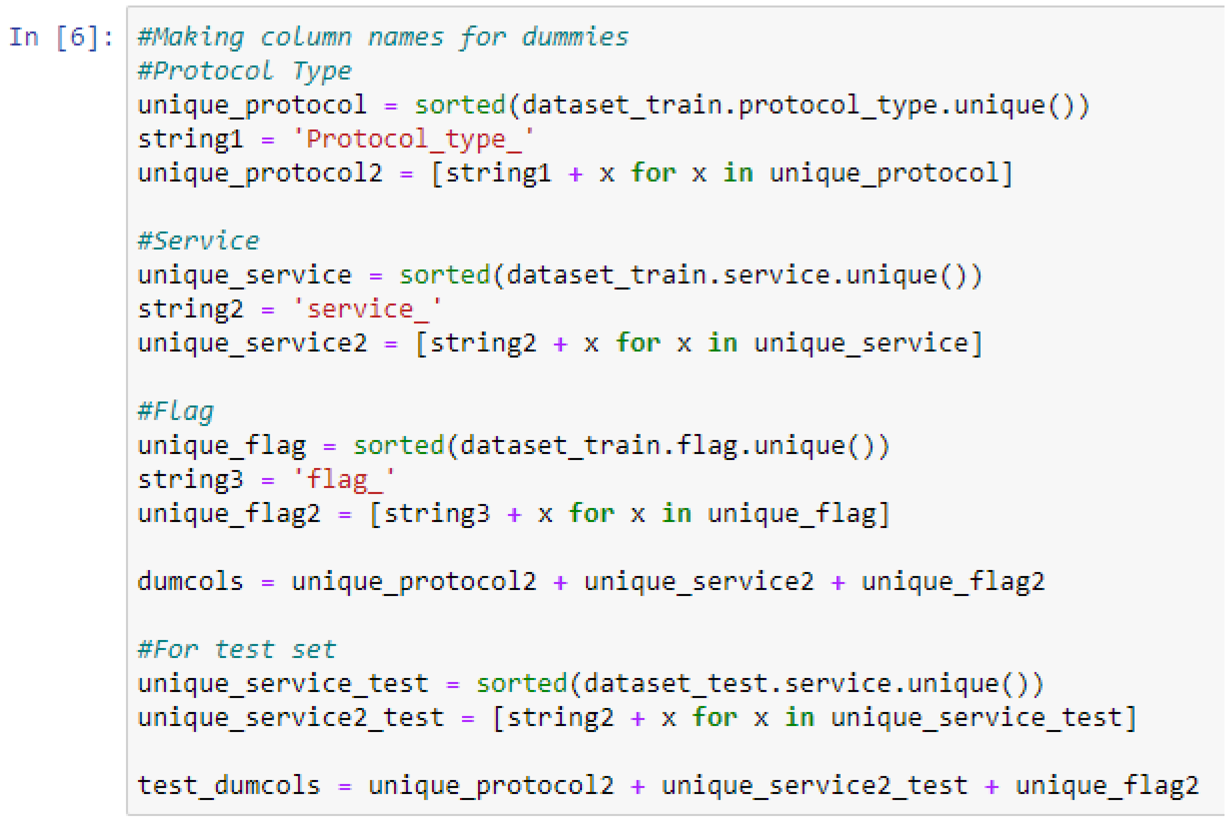
Here we've identified all of the express attributes in our training set and take a look at set. We have additionally identified the range of

classes inculcating inside every attribute



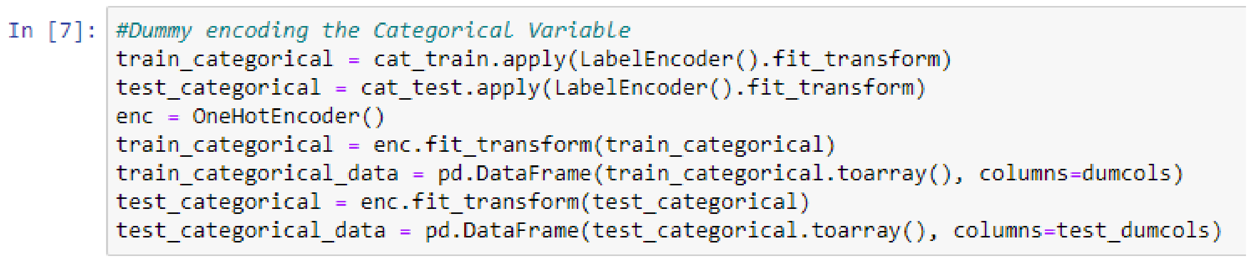
```
In [5]: #Encoding Categorical Variables
        from sklearn.preprocessing import LabelEncoder, OneHotEncoder
        categorical_columns = ['protocol_type', 'service', 'flag']
        cat_train = dataset_train[categorical_columns]
        cat_test = dataset_test[categorical_columns]
```

Here we have created 2 dummy data frames to include the categorical attributes in them



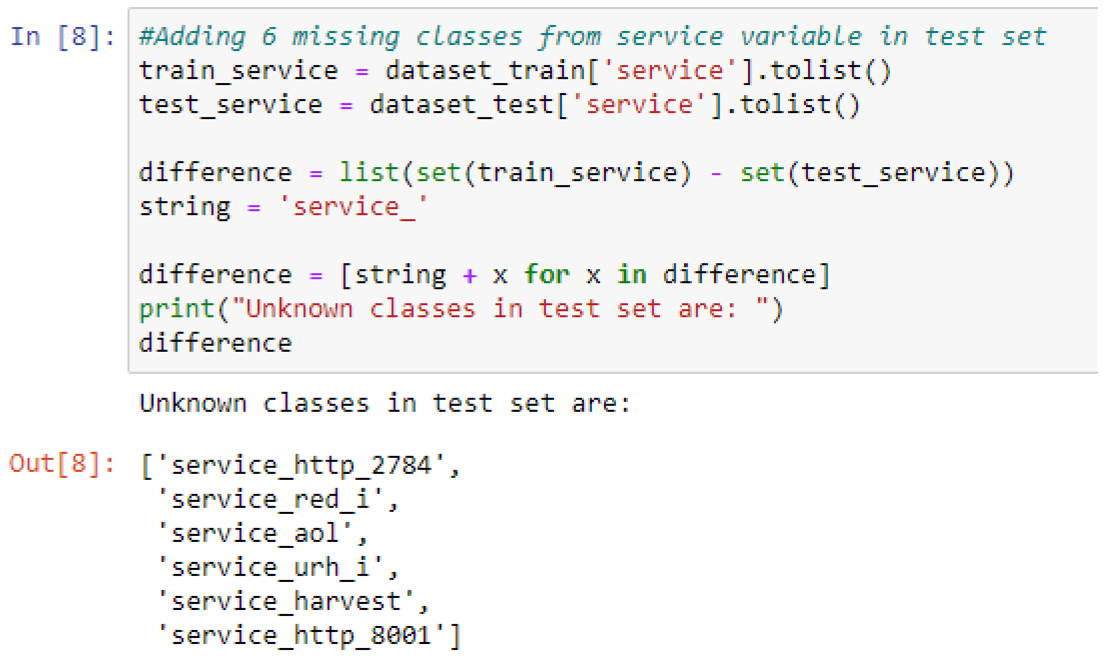
```
In [6]: #Making column names for dummies
        #Protocol Type
        unique_protocol = sorted(dataset_train.protocol_type.unique())
        string1 = 'Protocol_type_'
        unique_protocol2 = [string1 + x for x in unique_protocol]

        #Service
        unique_service = sorted(dataset_train.service.unique())
        string2 = 'service_'
        unique_service2 = [string2 + x for x in unique_service]

        #Flag
        unique_flag = sorted(dataset_train.flag.unique())
        string3 = 'flag_'
        unique_flag2 = [string3 + x for x in unique_flag]

        dumcols = unique_protocol2 + unique_service2 + unique_flag2

        #For test set
        unique_service_test = sorted(dataset_test.service.unique())
        unique_service2_test = [string2 + x for x in unique_service_test]

        test_dumcols = unique_protocol2 + unique_service2_test + unique_flag2
```

Here **we've** created the dummy attributes to **keep away from** the ordinal **introduction among those** nominal **specific** attributes.



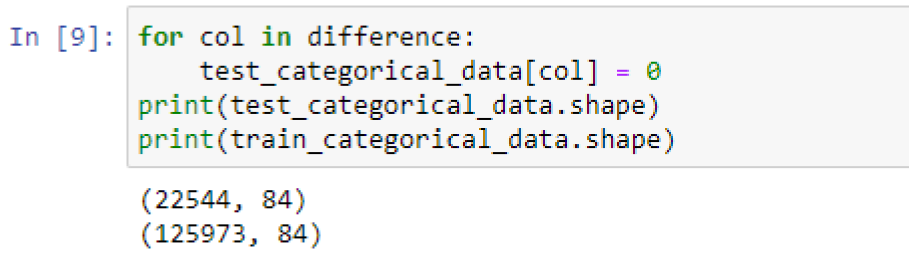
```
In [7]: #Dummy encoding the Categorical Variable
        train_categorical = cat_train.apply(LabelEncoder().fit_transform)
        test_categorical = cat_test.apply(LabelEncoder().fit_transform)
        enc = OneHotEncoder()
        train_categorical = enc.fit_transform(train_categorical)
        train_categorical_data = pd.DataFrame(train_categorical.toarray(), columns=dumcols)
        test_categorical = enc.fit_transform(test_categorical)
        test_categorical_data = pd.DataFrame(test_categorical.toarray(), columns=test_dumcols)
```

Here **we've** used the label encoder to fill **withinside the** dummy attributes in **every** of **the specific** attributes.



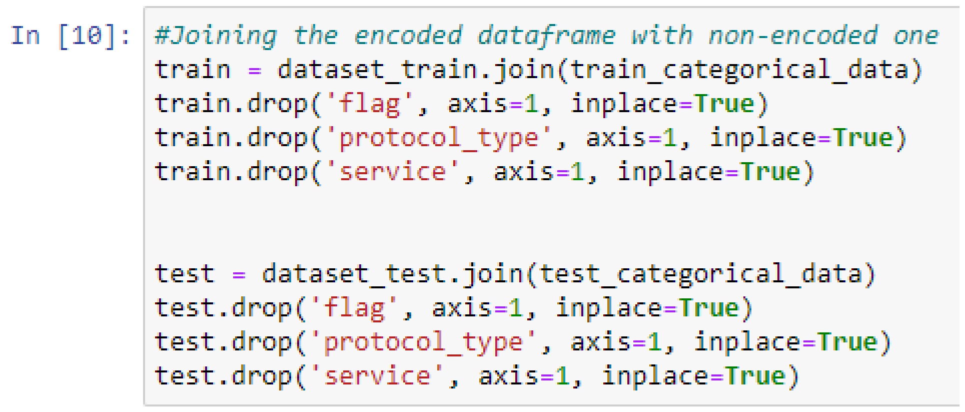
```
In [8]: #Adding 6 missing classes from service variable in test set
        train_service = dataset_train['service'].tolist()
        test_service = dataset_test['service'].tolist()

        difference = list(set(train_service) - set(test_service))
        string = 'service_'

        difference = [string + x for x in difference]
        print("Unknown classes in test set are: ")
        difference

        Unknown classes in test set are:

Out[8]: ['service_http_2784',
         'service_red_i',
         'service_aol',
         'service_urh_i',
         'service_harvest',
         'service_http_8001']
```

While checking **the specific** variables we **noticed** that service **characteristic** in **check** set has 70 **training whilst schooling** set has **sixty four training**. Hence, we **want** to **encompass the ones** 6 dummy attributes with **zero fee** in **every** our **schooling** set. This is what **we've got diagnosed** and **achieved** here.



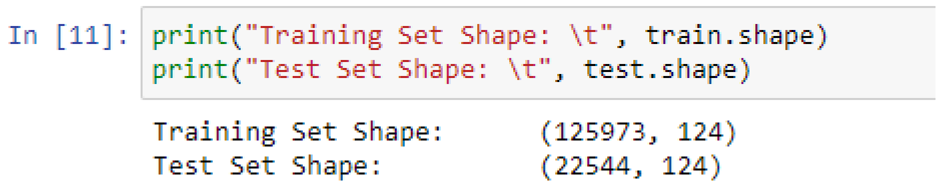
```
In [9]: for col in difference:
            test_categorical_data[col] = 0
        print(test_categorical_data.shape)
        print(train_categorical_data.shape)

        (22544, 84)
        (125973, 84)
```

Here we have finalised our data frames with the dummy values of categorical attributes.



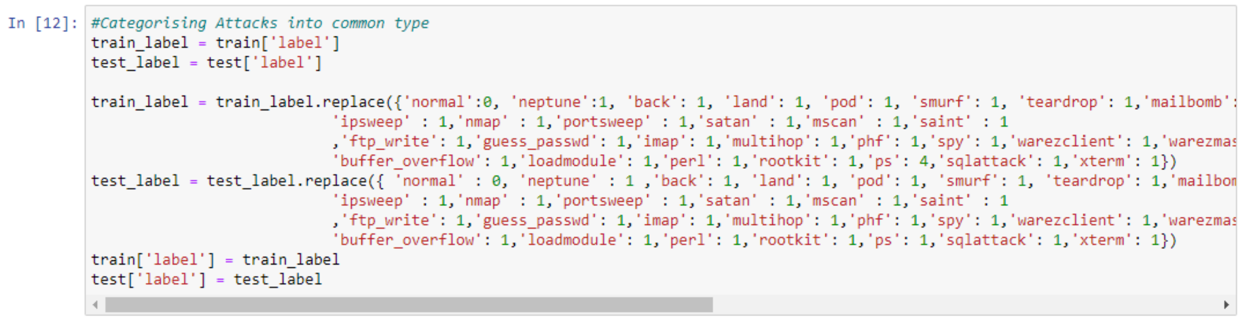
```
In [10]: #Joining the encoded dataframe with non-encoded one
         train = dataset_train.join(train_categorical_data)
         train.drop('flag', axis=1, inplace=True)
         train.drop('protocol_type', axis=1, inplace=True)
         train.drop('service', axis=1, inplace=True)


         test = dataset_test.join(test_categorical_data)
         test.drop('flag', axis=1, inplace=True)
         test.drop('protocol_type', axis=1, inplace=True)
         test.drop('service', axis=1, inplace=True)
```

Next, we've got combined our **original** dataset with dummy attributes that we **acquired** in our **specific** assignment. Also **right here we've got** dropped the **original specific** attributes **for you to** inculcate only the non-**specific** attributes.



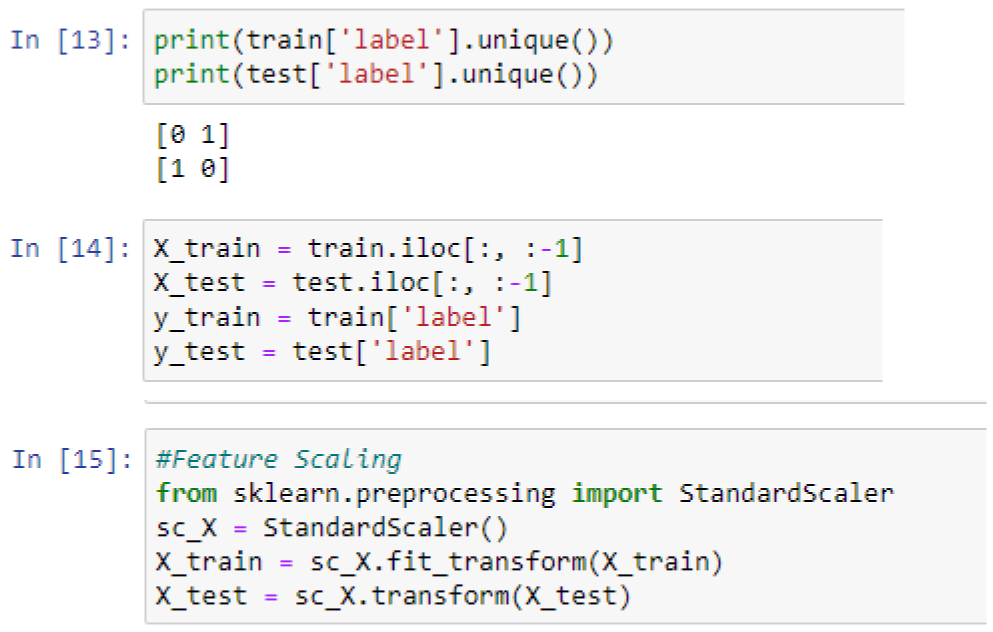
```
In [11]: print("Training Set Shape: \t", train.shape)
         print("Test Set Shape: \t", test.shape)

         Training Set Shape:      (125973, 124)
         Test Set Shape:          (22544, 124)
```

Here we have checked the number of attributes in both the training set and test set to see if they are equal… we can see that they have 124 attributes each and hence are compatible.



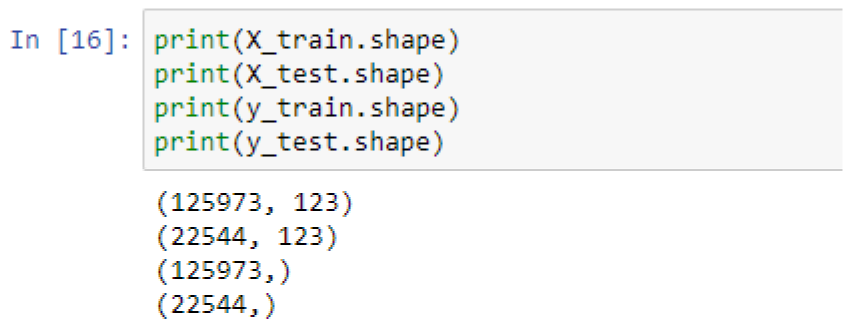
```
In [12]: #Categorising Attacks into common type
         train_label = train['label']
         test_label = test['label']

         train_label = train_label.replace({'normal':0, 'neptune':1, 'back': 1, 'land': 1, 'pod': 1, 'smurf': 1, 'teardrop': 1,'mailbomb':
                                   'ipsweep' : 1,'nmap' : 1,'portsweep' : 1,'satan' : 1,'mscan' : 1,'saint' : 1
                                   ,'ftp_write': 1,'guess_passwd': 1,'imap': 1,'multihop': 1,'phf': 1,'spy': 1,'warezclient': 1,'warezmas
                                   'buffer_overflow': 1,'loadmodule': 1,'perl': 1,'rootkit': 1,'ps': 4,'sqlattack': 1,'xterm': 1})
         test_label = test_label.replace({ 'normal' : 0, 'neptune' : 1 ,'back': 1, 'land': 1, 'pod': 1, 'smurf': 1, 'teardrop': 1,'mailbom
                                   'ipsweep' : 1,'nmap' : 1,'portsweep' : 1,'satan' : 1,'mscan' : 1,'saint' : 1
                                   ,'ftp_write': 1,'guess_passwd': 1,'imap': 1,'multihop': 1,'phf': 1,'spy': 1,'warezclient': 1,'warezmas
                                   'buffer_overflow': 1,'loadmodule': 1,'perl': 1,'rootkit': 1,'ps': 1,'sqlattack': 1,'xterm': 1})
         train['label'] = train_label
         test['label'] = test_label
```

Here we have categorised each of the label attribute either as “normal” or an “attack”. All the labels which are normal are given a label of 0 and all those that indicate an attack are labelled as 1.



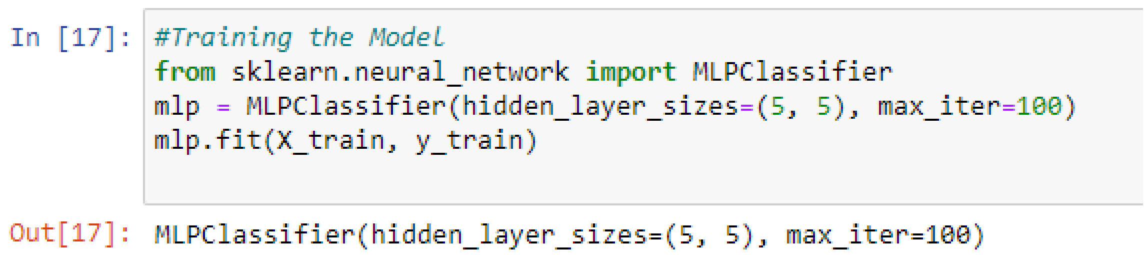
```
In [13]: print(train['label'].unique())
         print(test['label'].unique())

         [0 1]
         [1 0]

In [14]: X_train = train.iloc[:, :-1]
         X_test = test.iloc[:, :-1]
         y_train = train['label']
         y_test = test['label']

In [15]: #Feature Scaling
         from sklearn.preprocessing import StandardScaler
         sc_X = StandardScaler()
         X_train = sc_X.fit_transform(X_train)
         X_test = sc_X.transform(X_test)
```

Since all the attributes in our dataset don't follow a common scale, we need to feature scale the dataset in order to avoid any preassumed weight amongst them. We have used standard scalar to do this and it scales down each attribute to a range in -1 to 1.



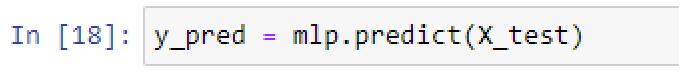
```
In [16]: print(X_train.shape)
         print(X_test.shape)
         print(y_train.shape)
         print(y_test.shape)

         (125973, 123)
         (22544, 123)
         (125973,)
         (22544,)
```

Here we have assigned the set of dependent and independent attributes. Also, we have printed the shape of each category that we have in order to check if they are compatible with each other.

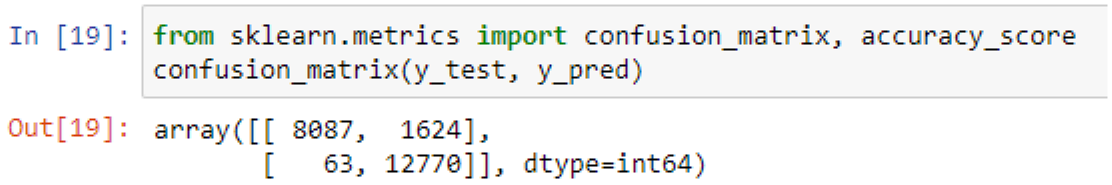


```
In [17]: #Training the Model
         from sklearn.neural_network import MLPClassifier
         mlp = MLPClassifier(hidden_layer_sizes=(5, 5), max_iter=100)
         mlp.fit(X_train, y_train)

Out[17]: MLPClassifier(hidden_layer_sizes=(5, 5), max_iter=100)
```

Here we have laid our neural network and then passed our input and output set to it in-order for it to adjust the weight biases.

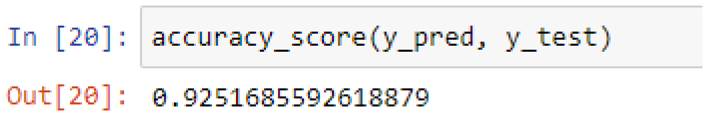


```
In [18]: y_pred = mlp.predict(X_test)
```



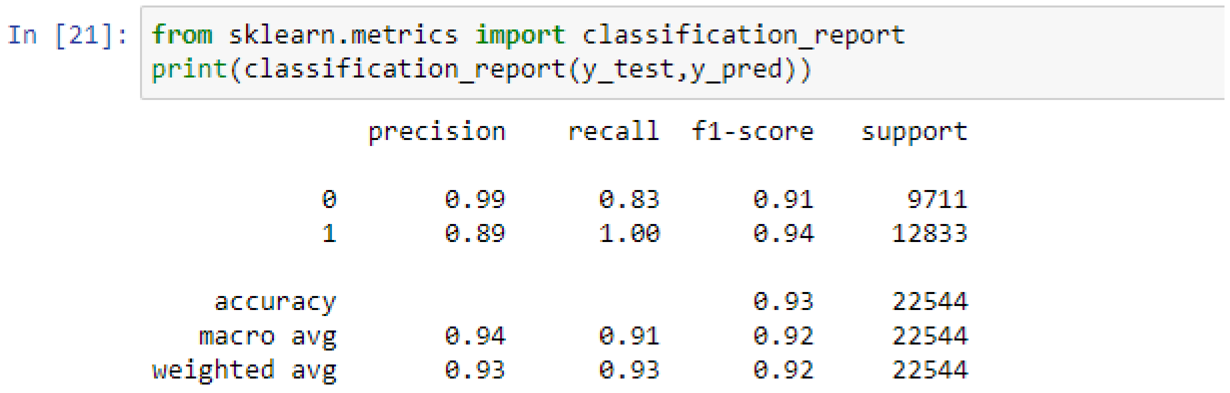
```
In [19]: from sklearn.metrics import confusion_matrix, accuracy_score
         confusion_matrix(y_test, y_pred)

Out[19]: array([[ 8087,  1624],
                [   63, 12770]], dtype=int64)
```

Here we have generated a vector y_pred that stores the result as predicted by our mlp classifier on test set. We have also generated the confusion matrix to check the performance of our classifier.



```
In [20]: accuracy_score(y_pred, y_test)

Out[20]: 0.9251685592618879
```

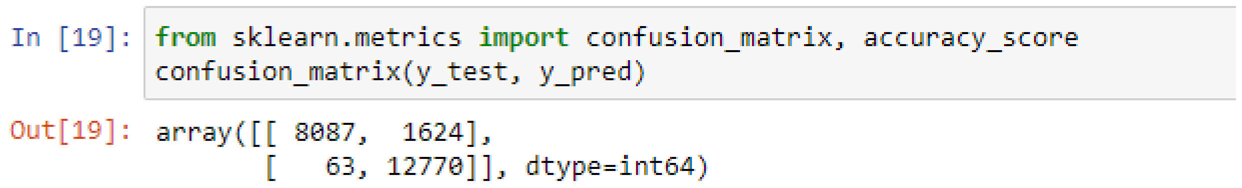
we have printed the accuracy of our model and printed the classification reported to finally check the performance of our model. We can see that the accuracy of the model is 92.51%.



```
In [21]: from sklearn.metrics import classification_report
         print(classification_report(y_test,y_pred))

              precision    recall  f1-score   support

           0       0.99      0.83      0.91      9711
           1       0.89      1.00      0.94     12833

    accuracy                           0.93     22544
   macro avg       0.94      0.91      0.92     22544
weighted avg       0.93      0.93      0.92     22544
```

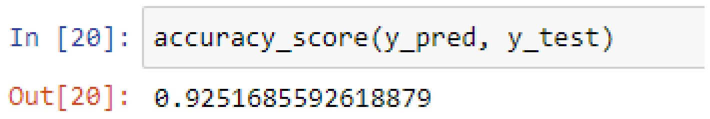
Result and Conclusion:

Confusion Matrix:



```
In [19]: from sklearn.metrics import confusion_matrix, accuracy_score
         confusion_matrix(y_test, y_pred)

Out[19]: array([[ 8087,  1624],
                [   63, 12770]], dtype=int64)
```

Accuracy:



```
In [20]: accuracy_score(y_pred, y_test)

Out[20]: 0.9251685592618879
```

Accuracy:92.51%.

- ✓ **Identified Normal = 9711**
- ✓ **Actual Normal = 9362**
- ✓ **Identified Attack = 12833**
- ✓ **Actual Attack = 13182**
- ✓ **True Normal = 8087**
- ✓ **True Attack = 12770**
- ✓ **False Normal = 1624**
- ✓ **False Attack = 63**
- ✓ **Precision Normal = 0.99**
- ✓ **Precision Attack = 0.89**
- ✓ **Recall Normal = 0.83**
- ✓ **Recall Attack = 1.00**

K Means

Question:

Illustrate the k-means clustering to cluster the data points for at least five epoch properly.

How to Implementing K-Means Clustering?

- **Using the elbow method to determine the optimal number of clusters for kmeans clustering**
- **Visualising the clusters**
- **Plotting the centroids of the clusters**

Dataset Used:

- Shopping-data (Uploaded in MS Team)
- https://archive.ics.uci.edu/ml/machine-learning-databases/

Procedure:

- Import necessary libraries - sklearn, numpy, pandas, etc.

-Using pandas, we first import the dataset into our workspace.

- Select the number of clusters for the dataset (K)

- Select K number of centroids

- By calculating the Euclidean distance or Manhattan distance assign the points to the nearest centroid, thus creating K groups

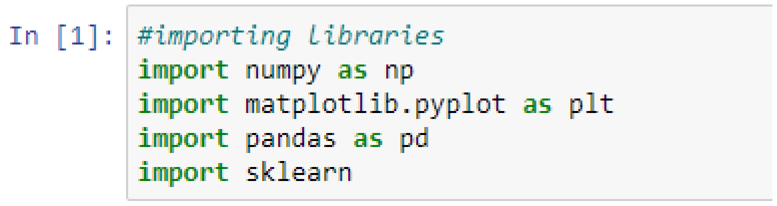
- Now find the original centroid in each group

- Again reassign the whole data point based on this new centroid, then repeat step 4 until the position of the centroid doesn't change.

- Using the elbow method to determine the optimal number of clusters for kmeans clustering

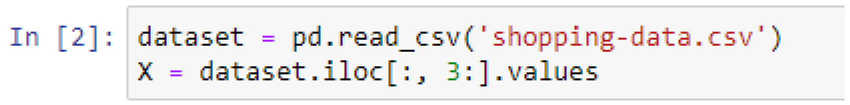
- Visualising the clusters and Plotting the centroids of the clusters

Code Snippets and Explanation:



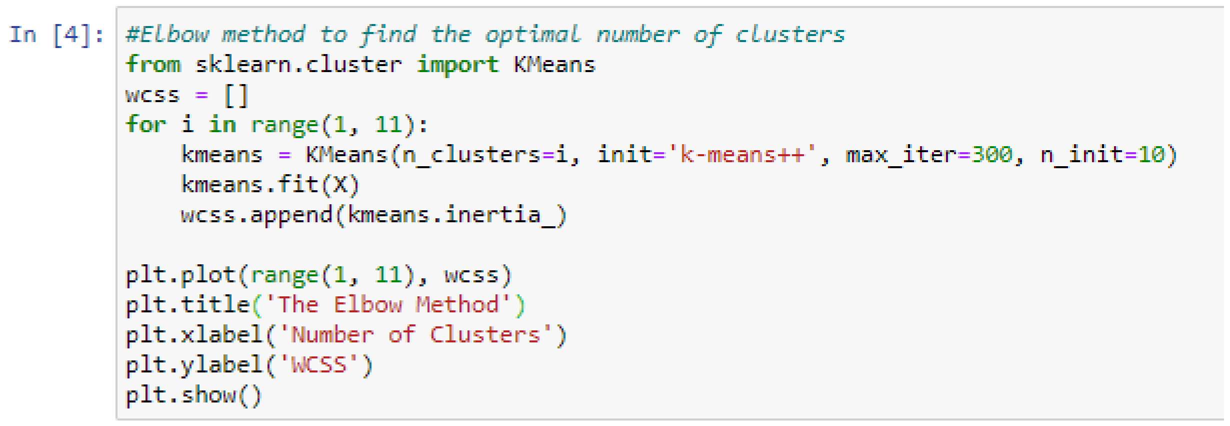
```
In [1]: #importing libraries
        import numpy as np
        import matplotlib.pyplot as plt
        import pandas as pd
        import sklearn
```

Import necessary libraries - sklearn, numpy, pandas, etc.

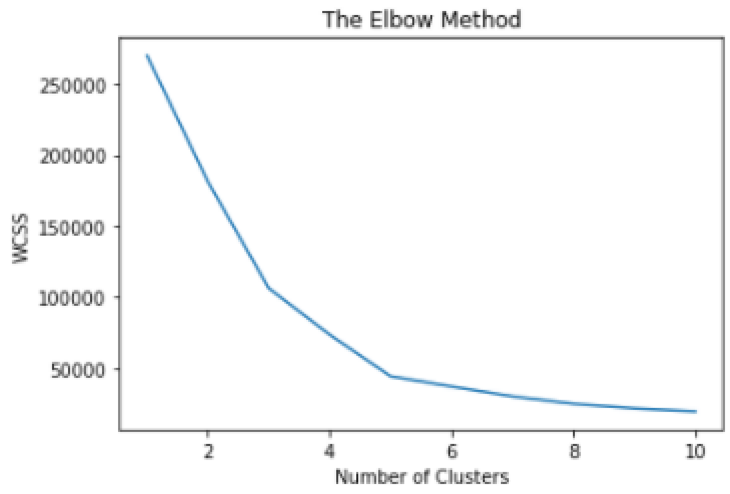


```
In [2]: dataset = pd.read_csv('shopping-data.csv')
        X = dataset.iloc[:, 3:].values
```

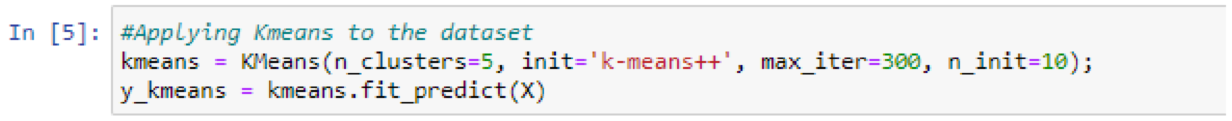
Using pandas, we first import the dataset into our workspace and are assigning the income attribute along with shopping score as independent variables.



```
In [4]: #Elbow method to find the optimal number of clusters
        from sklearn.cluster import KMeans
        wcss = []
        for i in range(1, 11):
            kmeans = KMeans(n_clusters=i, init='k-means++', max_iter=300, n_init=10)
            kmeans.fit(X)
            wcss.append(kmeans.inertia_)

        plt.plot(range(1, 11), wcss)
        plt.title('The Elbow Method')
        plt.xlabel('Number of Clusters')
        plt.ylabel('WCSS')
        plt.show()
```



Here we are plotting a graph that marks Within Cluster Sum of Squares (WCSS) with the increase in number of clusters. We can see an elbow formation when the number of clusters is 5 and hence, we assume that optimal number of clusters in our dataset is 5



```
In [5]: #Applying Kmeans to the dataset
        kmeans = KMeans(n_clusters=5, init='k-means++', max_iter=300, n_init=10);
        y_kmeans = kmeans.fit_predict(X)
```

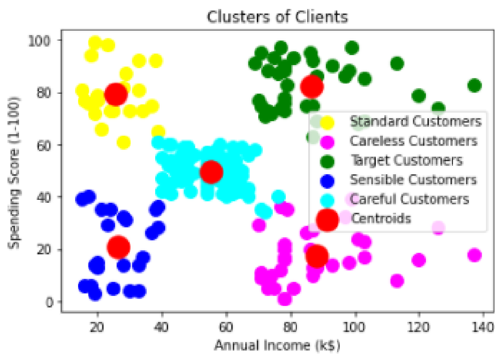
Here we are training our k-means model with 5 clusters. We are also generating the y_kmeans array that stores the cluster index of each input attribute from 0 to 4

```
In [9]: y_kmeans
```

```
Out[9]: array([3, 0, 3, 0, 3, 0, 3, 0, 3, 0, 3, 0, 3, 0, 3, 0, 3, 0, 3, 0, 3, 0,
               3, 0, 3, 0, 3, 0, 3, 0, 3, 0, 3, 0, 3, 0, 3, 0, 3, 0, 3, 0, 3, 4,
               3, 0, 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, 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, 2, 1, 2, 4, 2, 1, 2, 1, 2,
               4, 2, 1, 2, 1, 2, 1, 2, 1, 2, 4, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2,
               1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2,
               1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2,
               1, 2])
```

Here we are printing our y_kmeans array and we can see that each input cell is assigned a value between 0 and 4, both inclusive. This corresponds to the cluster index of each input.

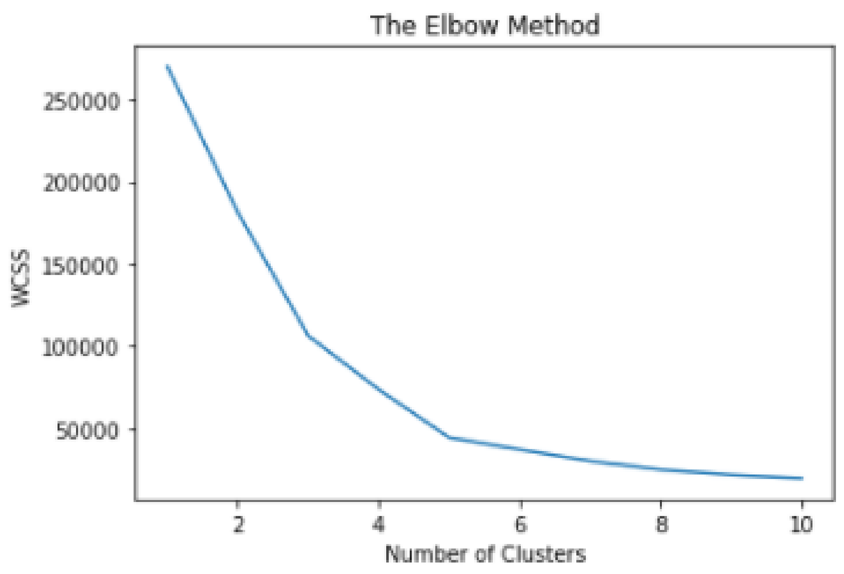
```
In [6]: # Visualising the clusters
        plt.scatter(X[y_kmeans == 0, 0], X[y_kmeans == 0, 1], s = 100, c = 'yellow', label = 'Standard Customers')
        plt.scatter(X[y_kmeans == 1, 0], X[y_kmeans == 1, 1], s = 100, c = 'magenta', label = 'Careless Customers')
        plt.scatter(X[y_kmeans == 2, 0], X[y_kmeans == 2, 1], s = 100, c = 'green', label = 'Target Customers')
        plt.scatter(X[y_kmeans == 3, 0], X[y_kmeans == 3, 1], s = 100, c = 'blue', label = 'Sensible Customers')
        plt.scatter(X[y_kmeans == 4, 0], X[y_kmeans == 4, 1], s = 100, c = 'cyan', label = 'Careful Customers')
        plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s = 300, c = 'red', label = 'Centroids')
        plt.title ('Clusters of Clients')
        plt.xlabel ('Annual Income (k$)')
        plt.ylabel ('Spending Score (1-100)')
        plt.legend()
        plt.show()
```



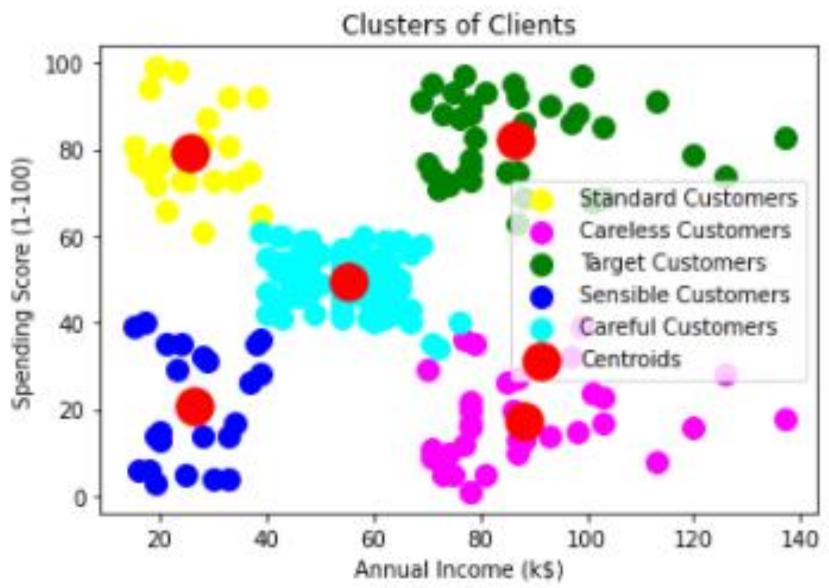
Here we have visualized our results. We have labelled different clusters as blue, green, pink, yellow and cyan. Each cluster correspond to different category of target audience. We have also marked centroids of each cluster which are red in colour.

Result and Conclusion:

Elbow Method Graph



Clustering Graph:



Here different clusters are marked as blue, yellow, magenta, cyan and green. The red dot over each cluster represents its centroid.

We can categorise these clusters as: -

- Yellow Cluster corresponds to careless customers as they have low income but high spending.
- Blue Cluster as Sensible customers, becoz they have low income and low spending.
- Cyan Clusters are standard cluster that suggest they have median income and median spending.
- The pink coloured cluster correspond to Target Customers, as they have high income but low spending, the shopping company can give them offers and attractions as they are capable of spending more but they aren't doing it currently.
- Finally, the Green coloured clusters are Careful customers. They have high income and thus high spending as well.