Trip details - Clustering

- We have details of 91 trips taken by different drivers from a cab service company
- The variables shared by the company are- TripID, TripLength, MaxSpeed, MostFreqSpeed, TripDuration, Brakes, IdlingTime, Honking
- Analyze the dataset and see whether the data can be separated into different clusters
- Can you identify from the trip details, whether the drive was taken inside the city or on the highway?
- If it is a city drive, can you identify whether it was taken during peak hours or non-peak hours?
- File – tripDetails.xlsx

Importing necessary packages

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

%matplotlib inline
```

In [2]:
```
data = pd.read_excel('tripDetails.xlsx')
data.head()
```

Out[2]:

	TripID	TripLength	MaxSpeed	MostFreqSpeed	TripDuration	Brakes	IdlingTime	Honking
0	1	21	51	14	93	307	27	112
1	2	148	130	106	156	226	5	114
2	3	18	38	16	100	351	26	107
3	4	22	43	48	36	17	4	5
4	5	183	108	90	171	88	5	29

In [3]:
```
data.drop(['TripID'],axis = 1,inplace = True)
data.head()
```

Out[3]:

	TripLength	MaxSpeed	MostFreqSpeed	TripDuration	Brakes	IdlingTime	Honking
0	21	51	14	93	307	27	112
1	148	130	106	156	226	5	114
2	18	38	16	100	351	26	107
3	22	43	48	36	17	4	5
4	183	108	90	171	88	5	29

In [4]:
```
features = list(data.columns)
print(features)
```

```
['TripLength', 'MaxSpeed', 'MostFreqSpeed', 'TripDuration', 'Brakes', 'IdlingTime', 'H
onking']
```

In [5]:
```
units = ['kms','kmph','kmph','mins','counts','mins','counts']
feature_units = dict(zip(features,units))
feature_units
```

Out[5]:
```
{'TripLength': 'kms',
 'MaxSpeed': 'kmph',
 'MostFreqSpeed': 'kmph',
 'TripDuration': 'mins',
 'Brakes': 'counts',
 'IdlingTime': 'mins',
 'Honking': 'counts'}
```

Datatype of variables

In [6]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 91 entries, 0 to 90
Data columns (total 7 columns):
TripLength       91 non-null int64
MaxSpeed         91 non-null int64
MostFreqSpeed    91 non-null int64
TripDuration     91 non-null int64
Brakes           91 non-null int64
IdlingTime       91 non-null int64
Honking          91 non-null int64
dtypes: int64(7)
memory usage: 5.1 KB
```

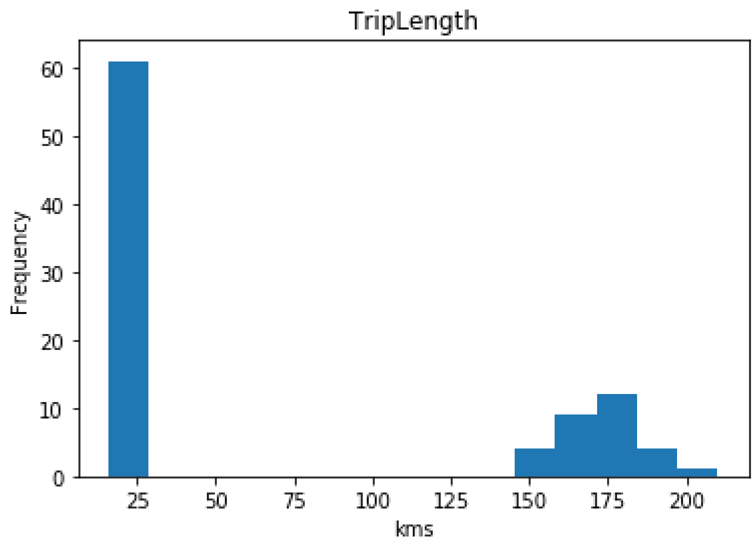
In [7]:
```
data.describe()
```

Out[7]:

	TripLength	MaxSpeed	MostFreqSpeed	TripDuration	Brakes	IdlingTime	Honking
count	91.000000	91.000000	91.000000	91.000000	91.000000	91.000000	91.000000
mean	70.769231	70.362637	50.648352	87.373626	135.439560	11.593407	49.923077
std	73.302126	34.509424	34.349632	47.123160	114.758607	9.796800	46.371023
min	16.000000	35.000000	12.000000	22.000000	14.000000	4.000000	4.000000
25%	20.000000	42.000000	15.500000	34.500000	36.500000	5.000000	20.000000
50%	21.000000	54.000000	42.000000	88.000000	100.000000	5.000000	25.000000
75%	163.000000	105.500000	89.000000	133.000000	198.000000	24.000000	97.500000
max	210.000000	138.000000	118.000000	171.000000	429.000000	32.000000	155.000000

A look at histogram of each feature

In [8]:
```python
for item in features:
    data[item].plot(kind='hist', bins = 15)
    plt.title(item)
    plt.xlabel(feature_units[item])
    plt.show()
```



From histograms, we observe that the data points are clearly segregated into different groups, with differing number of segregations for each feature.

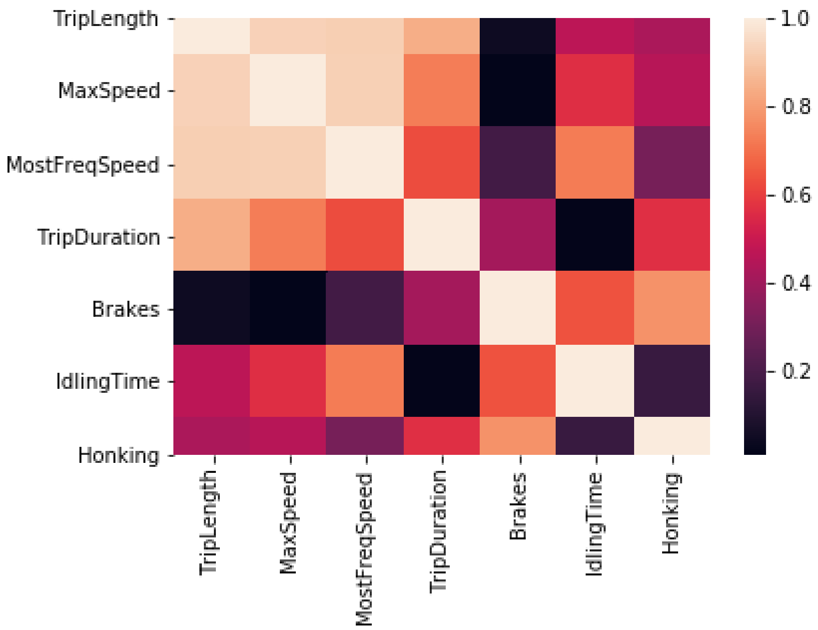
A look at relationship between different features - correlation

In [9]:
```python
correlation = data.corr()
print(correlation)
```

```
              TripLength  MaxSpeed  MostFreqSpeed  TripDuration    Brakes  \
TripLength      1.000000  0.933549       0.922928      0.842934  0.047158
MaxSpeed        0.933549  1.000000       0.928592      0.730388  0.011993
MostFreqSpeed   0.922928  0.928592       1.000000      0.632675 -0.182159
TripDuration    0.842934  0.730388       0.632675      1.000000  0.416028
Brakes          0.047158  0.011993      -0.182159      0.416028  1.000000
IdlingTime     -0.471204 -0.564379      -0.726001      0.018913  0.641201
Honking         0.429318  0.458151       0.309691      0.571365  0.778774

              IdlingTime   Honking
TripLength     -0.471204  0.429318
MaxSpeed       -0.564379  0.458151
MostFreqSpeed  -0.726001  0.309691
TripDuration    0.018913  0.571365
Brakes          0.641201  0.778774
IdlingTime      1.000000  0.160450
Honking         0.160450  1.000000
```

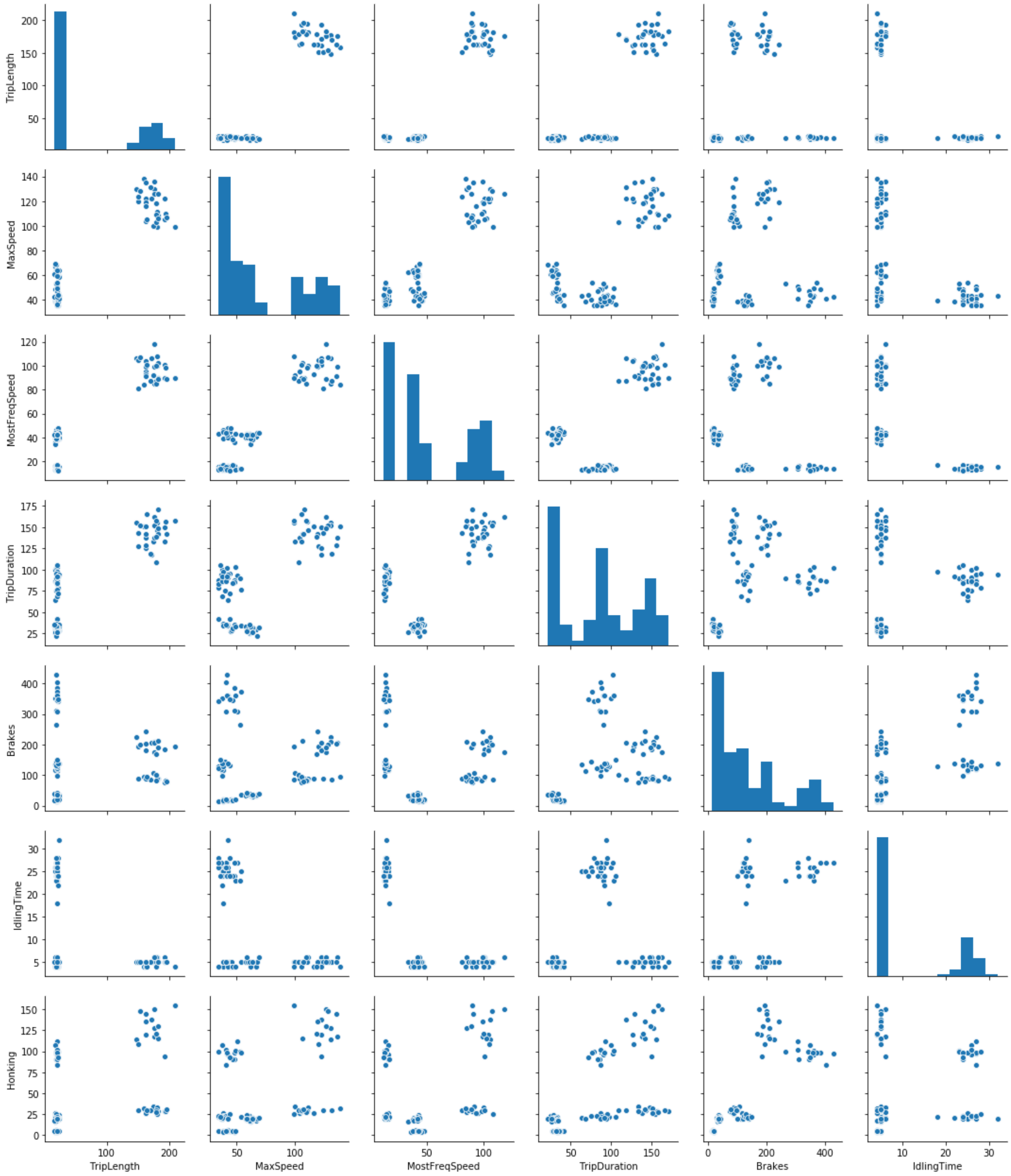
In [10]:
```
sns.heatmap(np.abs(correlation), xticklabels = correlation.columns, yticklabels = corre
plt.show()
```



From correlation table and correlation heatmap, we see that TripLength, MaxSpeed, MostFreqSpeed are highly correlated.

Visualizing scatter of the data

In [11]:
```
sns.pairplot(data)
plt.show()
```



Observation about scatter

- We see that few clusters are spherically distributed and few are elliptically distributed
- Also there exist different number of clusters (2,3,4,5) for different pair combination of features
- Few clusters are compact while others are not
- In most of the scatter plots (subplots) above, we see that there are 3 candidate clusters (based on compactness and isolation)

Scaling : Important step in every Machine Learning problem -

- To avoid giving undue advantage to some features which are expressed in some particular units, whose magnitude might be higer than some other feature variable (due to choice of units), scaling all features, so that they are numerically of same order of magnitude, is essential.
- We will use standard scaling (xi-u)/sigma.

In [12]:

```
from sklearn.preprocessing import StandardScaler
import copy as cp
```

In [13]:

```
data2 = data.copy()
data2 = StandardScaler().fit_transform(data2.values)
data2 = pd.DataFrame(data2,columns = features)
```

Let us help them discover the patterns in the data they have gathered using K-Means clustering

K-Means Clsutering: ¶

- A technique to partition N observations into K clusters (K≤N) in which each observation belongs to cluster with nearest mean
- One of the simplest unsupervised algorithms
- Given N observations (x1,x2,...,xN) , K-means clustering will partition n observations into K (K≤N) sets S={s1,...,sk} so as to minimize the within cluster sum of squares (WCSS)

$$\underset{\mathbf{S}}{\arg\min} \sum_{i=1}^{k} \sum_{\mathbf{x} \in S_i} \left\| \mathbf{x} - \boldsymbol{\mu}_i \right\|^2$$

K-Means Algorithm

Input: D , k

Algorithm:

- Step 1: Randomly choose two points as the cluster centers
- Step 2: Compute the distances and group the closest ones
- Step 3: Compute the new mean and repeat step 2
- Step 4: If change in mean is negligible or no reassignment then stop the process

Output: Ci - Centroids of k clusters, cluster assignment labels for each datapoint

One Convergence Criteria: No significant decrease in the sum squared error .i.e sum of square of distance between each datapoint to its assigned centroid. This is also called inertia

K-Means using sklearn in python

In [14]:

```
from sklearn import cluster
```

Determining number of clusters(K):

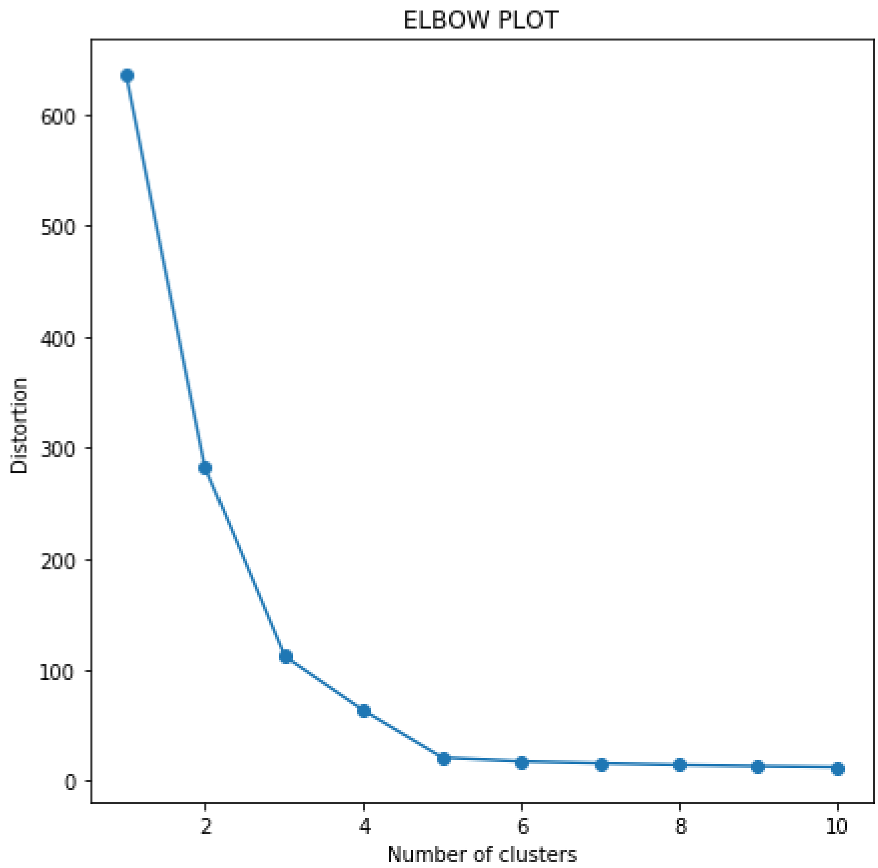
- Let us try clustering the data with K-Means for different values of K
- Elbow method – looks at percentage of variance explained as a function of number of clusters
- The point where marginal decrease plateaus is an indicator of the optimal number of clusters

  We will summarize K-Means for different k in an elbow plot below

In [15]:
```
distortions = []  # Empty list to store wss

for i in range(1, 11):
    km = cluster.KMeans(n_clusters=i,
                        init='k-means++',
                        n_init = 10,
                        max_iter = 300,
                        random_state = 100)
    km.fit(data2.values)
    distortions.append(km.inertia_)

#Plotting the K-means Elbow plot
plt.figure(figsize = (7,7))
plt.plot(range(1,11), distortions, marker='o')
plt.title('ELBOW PLOT')
plt.xlabel('Number of clusters')
plt.ylabel('Distortion')
plt.show()
```



Though from elbow plot, we see that k=5 is best number of clusters, we will choose k=3 , because that is the point where marginal decrease plateaus.

We will cluster the data into 3 groups and label the datapoints with their assignment to the clusters.

In [16]:
```python
k = 3
km3 = cluster.KMeans(n_clusters=k,
                init='k-means++',
                n_init = 10,
                max_iter = 300,
                random_state = 100)
km3.fit(data2.values)
```

Out[16]:
```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
       n_clusters=3, n_init=10, n_jobs=None, precompute_distances='auto',
       random_state=100, tol=0.0001, verbose=0)
```

In [17]:
```python
labels = km3.labels_
Ccenters = km3.cluster_centers_
data2['labels'] = labels
data2['labels'] = data2['labels'].astype('str')
print(data2['labels'])
```

```
0     0
1     1
2     0
3     2
4     1
     ..
86    0
87    1
88    0
89    1
90    2
Name: labels, Length: 91, dtype: object
```

A look at pair plots after clustering

In [*]:
```python
sns.pairplot(data2, x_vars = features, y_vars = features, hue='labels', diag_kind='kde'
plt.show()
```

We see from pair plot that, for every pair of features, the points have been well clustered into different groups. Though isolation and compactness are not observed together in all possible pairs of features.

In [*]:
```python
c_df = pd.concat([data[data2['labels']=='0'].mean(),
                  data[data2['labels']=='1'].mean(),
                  data[data2['labels']=='2'].mean()],
                  axis=1)
c_df.columns = ['cluster1','cluster2','cluster3']
c_df
```

Observations:

- **Cluster1** is distinguised by comparatively very high values for *Brakes, IdlingTime, Honking, low MaxSpeed and TripLength*
- This is indicative of *intercity* travel during *peak hours*
- *MaxSpeed, MostFreqSpeed* and *TripDuration* is higher for cluster2 than cluster 1 and 3
- **Cluster2** is is indicative of *highway trips*
- **Cluster3** is indicative of city trips during *non-peak hours* }

Let us assign these names to the clusters -

In [*]:
```
triptype       = ['Intercity-Peak hours','Highway','Intercity-Non-peak hours']
data['labels'] = labels
data['labels'] = data['labels'].map({0:triptype[0],1:triptype[1],2:triptype[2]})
```

In [*]:
```
print(data.head())
```

End of script