ASSIGNMENT - 1

Practical No. - 1

Aim:

Predict the price of the Uber ride from a given pickup point to the agreed drop-off location.

Perform following tasks:

1. Pre-process the dataset.

2. Identify outliers.

3. Check the correlation.

4. Implement linear regression and random forest regression models.

5. Evaluate the models and compare their respective scores like R2, RMSE, etc.

Software Requirements:

- Anaconda Installer
- Windows 10 OS
- Jupyter Notebook

Theory:

Machine Learning:

Machine learning is a growing technology which enables computer to learn automatically from past data. It is a said as a subset of Artificial intelligence that is mainly concerned with the development of algorithms which allow a computer to learn from the data and past experiences on their own.

Data Pre-processing:

It is a process of preparing the raw data and making it suitable for a machine learning model. It is the first and crucial step while creating a machine learning model.

Data pre-processing is required tasks for cleaning the data and making it suitable for a machine learning, model which also increases the accuracy and efficiency of a machine learning model.

Steps of Data Pre-processing:

1. **Getting the dataset :**

To create a machine learning model, the first thing we required is a dataset as a machine learning model completely works on data. The collected data for a particular problem in a proper format is known as the data sets.

To use dataset in our code, we usually put it into a CSV file(Comma Separated Values->file format which allows us to save the tabular data, such as spreadsheets).

2. Importing libraries:

In order to perform data preprocessing using Python, we need to import some predefined Python libraries. These libraries are used to perform some specific jobs. There are three specific libraries that we will use for data preprocessing, which are:

1. Numpy: Numpy Python library is used for including any type of mathematical operation in the code. It is the fundamental package for scientific calculation in Python. It also supports to add large, multidimensional arrays and matrices.

2. Matplotlib: The second library is **matplotlib**, which is a Python 2D plotting library, and with this library, we need to import a sub-library **pyplot**. This library is used to plot any type of charts in Python for the code.

3. Pandas: The last library is the Pandas library, which is one of the most famous Python libraries and used for importing and managing the datasets. It is an open-source data manipulation and analysis library.

3. Importing datasets:

Now we need to import the datasets which we have collected for our machine learning project. But before importing a dataset, we need to set the current directory as a working directory. To set a working directory in Spyder IDE, we need to follow the below steps:

1. Save your Python file in the directory which contains dataset.
2. Go to File explorer option in Spyder IDE, and select the required directory.
3. Click on F5 button or run option to execute the file.
4. The next step of data preprocessing is to handle missing data in the datasets. If our dataset contains some missing data, then it may create a huge problem for our machine learning model. Hence it is necessary to handle missing values present in the dataset.
5. **Ways to handle missing data:**
6. There are mainly two ways to handle missing data, which are:
7. **By deleting the particular row:** The first way is used to commonly deal with null values. In this way, we just delete the specific row or column which consists of null values. But this way is not so efficient and removing data may lead to loss of information which will not give the accurate output.
8. **By calculating the mean:** In this way, we will calculate the mean of that column or row which contains any missing value and will put it on the place of missing value. This strategy is useful for the features which have numeric data such as age, salary, year, etc. Here, we will use this approach.
9. To handle missing values, we will use **Scikit-learn** library in our code, which contains various libraries for building machine learning models. Here we will use **Imputer** class of **sklearn.preprocessing** library.

4. Finding Missing Data-

Categorical data is data which has some categories such as, in our dataset; there are two categorical variable, **Country**, and **Purchased**.

Since machine learning model completely works on mathematics and numbers, but if our dataset would have a categorical variable, then it may create trouble while building the model. So it is necessary to encode these categorical variables into numbers.

For Country variable:

Firstly, we will convert the country variables into categorical data. So to do this, we will use **LabelEncoder()** class from **preprocessing** library.

5. Encoding Categorical Data-

Categorical data is data which has some categories such as, in our dataset; there are two categorical variable, **Country**, and **Purchased**.

Since machine learning model completely works on mathematics and numbers, but if our dataset would have a categorical variable, then it may create trouble while building the model. So it is necessary to encode these categorical variables into numbers.

For Country variable:

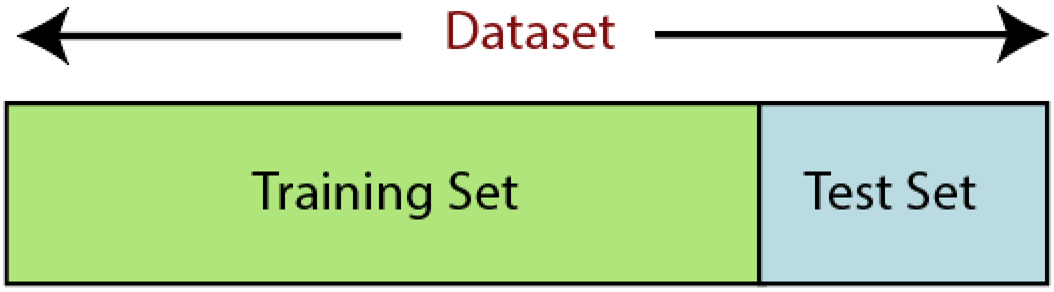
Firstly, we will convert the country variables into categorical data. So to do this, we will use **LabelEncoder()** class from **preprocessing** library.

6. Spliting dataset into training and test set-

In machine learning data preprocessing, we divide our dataset into a training set and test set. This is one of the crucial steps of data preprocessing as by doing this, we can enhance the performance of our machine learning model.

Suppose, if we have given training to our machine learning model by a dataset and we test it by a completely different dataset. Then, it will create difficulties for our model to understand the correlations between the models.

If we train our model very well and its training accuracy is also very high, but we provide a new dataset to it, then it will decrease the performance. So we always try to make a machine learning model which performs well with the training set and also with the test dataset. Here, we can define these datasets as:



Training Set: A subset of dataset to train the machine learning model, and we already know the output.

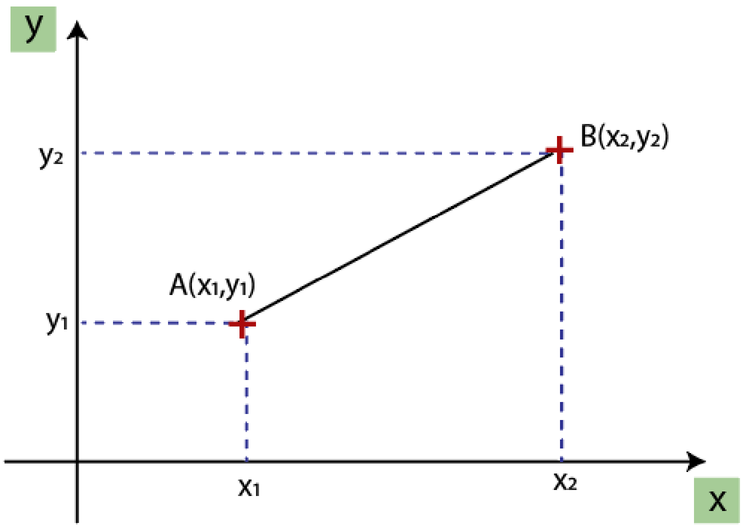
Test set: A subset of dataset to test the machine learning model, and by using the test set, model predicts the output.

7. Feature Scaling-

Feature scaling is the final step of data preprocessing in machine learning. It is a technique to standardize the independent variables of the dataset in a specific range. In feature scaling, we put our variables in the same range and in the same scale so that no any variable dominate the other variable.

As we can see, the age and salary column values are not on the same scale. A machine learning model is based on **Euclidean distance**, and if we do not scale the variable, then it will cause some issue in our machine learning model.

Euclidean distance is given as:

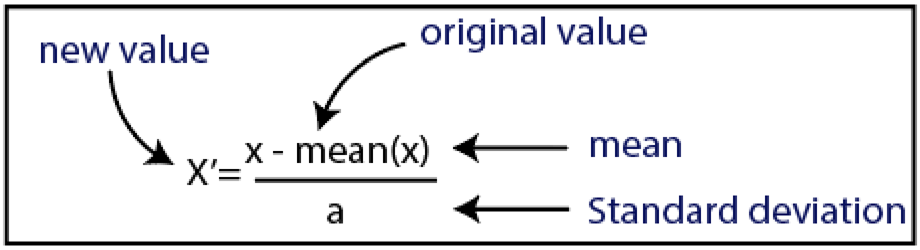


Euclidean Distance Between A and B $=\sqrt{(x_2-x_1)^2+(y_2-y_1)^2}$

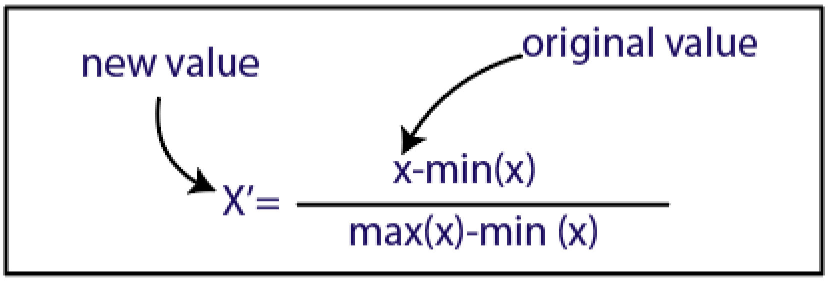
If we compute any two values from age and salary, then salary values will dominate the age values, and it will produce an incorrect result. So to remove this issue, we need to perform feature scaling for machine learning.

There are two ways to perform feature scaling in machine learning:

Standardization



Normalization



Here, we will use the standardization method for our dataset.

Program:

```
#Importing the required libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt


#importing the dataset
df  = pd.read_csv("uber.csv")


df.head()
```

	Unnamed: 0	key	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count
0	24238194	2015-05-07 19:52:06.0000003	7.5	2015-05-07 19:52:06 UTC	-73.999817	40.738354	-73.999512	40.723217	1
1	27835199	2009-07-17 20:04:56.0000002	7.7	2009-07-17 20:04:56 UTC	-73.994355	40.728225	-73.994710	40.750325	1
2	44984355	2009-08-24 21:45:00.00000061	12.9	2009-08-24 21:45:00 UTC	-74.005043	40.740770	-73.962565	40.772647	1
3	25894730	2009-06-26 08:22:21.0000001	5.3	2009-06-26 08:22:21 UTC	-73.976124	40.790844	-73.965316	40.803349	3
4	17610152	2014-08-28 17:47:00.000000188	16.0	2014-08-28 17:47:00 UTC	-73.925023	40.744085	-73.973082	40.761247	5

```
df.info() #To get the required information of the dataset

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200000 entries, 0 to 199999
Data columns (total 9 columns):
 #   Column             Non-Null Count   Dtype
---  ------             --------------   -----
 0   Unnamed: 0         200000 non-null  int64
 1   key                200000 non-null  object
 2   fare_amount        200000 non-null  float64
 3   pickup_datetime    200000 non-null  object
 4   pickup_longitude   200000 non-null  float64
 5   pickup_latitude    200000 non-null  float64
 6   dropoff_longitude  199999 non-null  float64
 7   dropoff_latitude   199999 non-null  float64
 8   passenger_count    200000 non-null  int64
dtypes: float64(5), int64(2), object(2)
memory usage: 13.7+ MB
```

```
df.columns #TO get number of columns in the dataset

Index(['Unnamed: 0', 'key', 'fare_amount', 'pickup_datetime',
'pickup_longitude', 'pickup_latitude', 'dropoff_longitude',
'dropoff_latitude', 'passenger_count'], dtype='object')

df = df.drop(['Unnamed: 0', 'key'], axis= 1) #To drop unnamed column as
 it isn't required

df.head()
```

	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count
0	7.5	2015-05-07 19:52:06 UTC	-73.999817	40.738354	-73.999512	40.723217	1
1	7.7	2009-07-17 20:04:56 UTC	-73.994355	40.728225	-73.994710	40.750325	1
2	12.9	2009-08-24 21:45:00 UTC	-74.005043	40.740770	-73.962565	40.772647	1
3	5.3	2009-06-26 08:22:21 UTC	-73.976124	40.790844	-73.965316	40.803349	3
4	16.0	2014-08-28 17:47:00 UTC	-73.925023	40.744085	-73.973082	40.761247	5

```
df.shape #To get the total (Rows,Columns)
(200000, 7)

df.dtypes #To get the type of each column
fare_amount float64

pickup_datetime object

pickup_longitude float64

pickup_latitude float64

dropoff_longitude float64
```

```
dropoff_latitude float64

passenger_count int64

dtype: object
```

```
df.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200000 entries, 0 to 199999
Data columns (total 7 columns):
 #   Column             Non-Null Count   Dtype
---  ------             --------------   -----
 0   fare_amount        200000 non-null  float64
 1   pickup_datetime    200000 non-null  object
 2   pickup_longitude   200000 non-null  float64
 3   pickup_latitude    200000 non-null  float64
 4   dropoff_longitude  199999 non-null  float64
 5   dropoff_latitude   199999 non-null  float64
 6   passenger_count    200000 non-null  int64
dtypes: float64(5), int64(1), object(1)
memory usage: 10.7+ MB
```

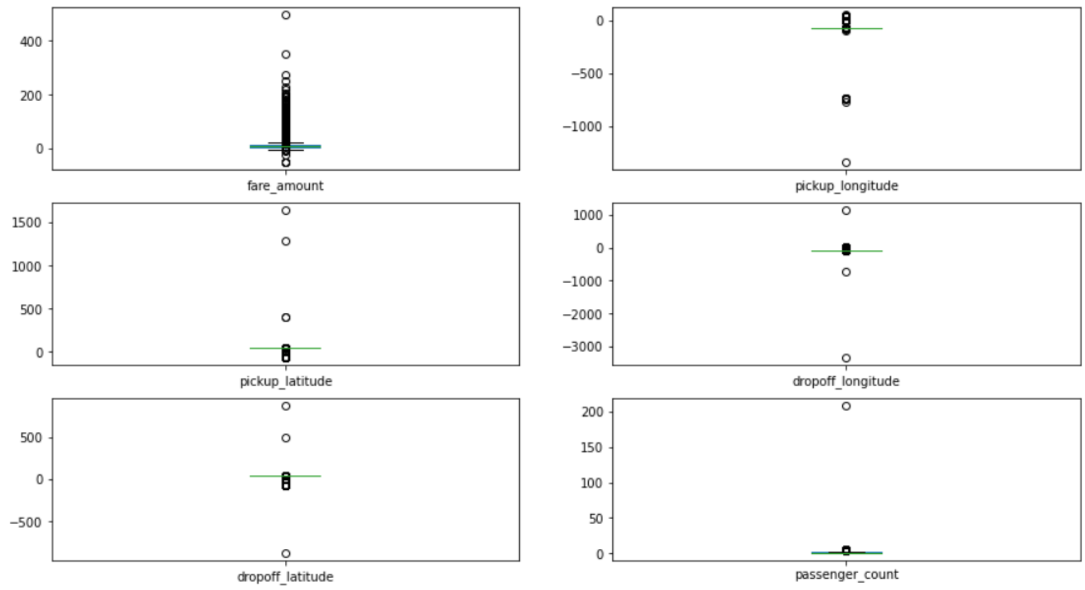
```
df.describe() #To get statistics of each columns
```

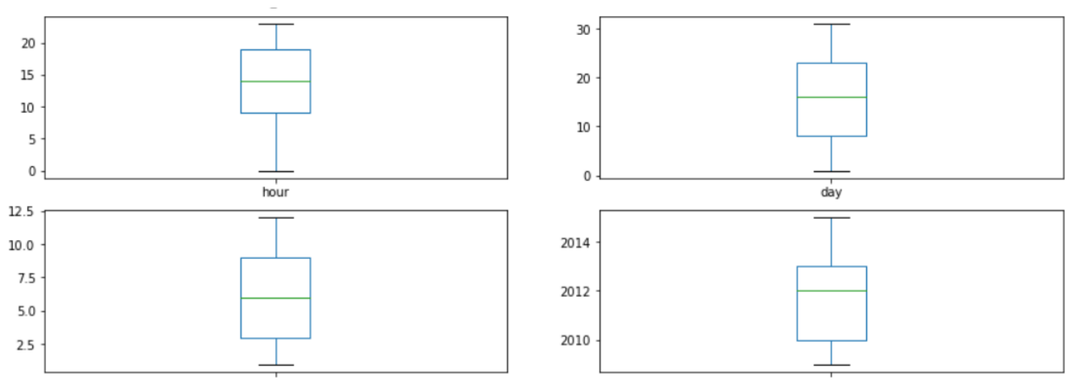
	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count
count	200000.000000	200000.000000	200000.000000	199999.000000	199999.000000	200000.000000
mean	11.359955	-72.527638	39.935885	-72.525292	39.923890	1.684535
std	9.901776	11.437787	7.720539	13.117408	6.794829	1.385997
min	-52.000000	-1340.648410	-74.015515	-3356.666300	-881.985513	0.000000
25%	6.000000	-73.992065	40.734796	-73.991407	40.733823	1.000000
50%	8.500000	-73.981823	40.752592	-73.980093	40.753042	1.000000
75%	12.500000	-73.967154	40.767158	-73.963658	40.768001	2.000000
max	499.000000	57.418457	1644.421482	1153.572603	872.697628	208.000000

```
df.isnull().sum()
fare_amount 0

pickup_datetime 0

pickup_longitude 0

pickup_latitude 0

dropoff_longitude 1

dropoff_latitude 1

passenger_count 0

dtype: int64
```

```
df['dropoff_latitude'].fillna(value=df['dropoff_latitude'].mean(),inpla
ce = True)
df['dropoff_longitude'].fillna(value=df['dropoff_longitude'].median(),i
nplace = True)


df.isnull().sum()
fare_amount 0

pickup_datetime 0

pickup_longitude 0

pickup_latitude 0

dropoff_longitude 0

dropoff_latitude 0

passenger_count 0

dtype: int64



df.dtypes
fare_amount float64

pickup_datetime object

pickup_longitude float64

pickup_latitude float64

dropoff_longitude float64

dropoff_latitude float64

passenger_count int64

dtype: object


df.plot(kind = "box",subplots = True,layout = (7,2),figsize=(15,20)) #B
oxplot to check the outliers
fare_amount AxesSubplot(0.125,0.787927;0.352273x0.0920732)
pickup_longitude AxesSubplot(0.547727,0.787927;0.352273x0.0920732)
pickup_latitude AxesSubplot(0.125,0.677439;0.352273x0.0920732)
dropoff_longitude AxesSubplot(0.547727,0.677439;0.352273x0.0920732)
dropoff_latitude AxesSubplot(0.125,0.566951;0.352273x0.0920732)
passenger_count AxesSubplot(0.547727,0.566951;0.352273x0.0920732)

hour AxesSubplot(0.125,0.456463;0.352273x0.0920732)

day AxesSubplot(0.547727,0.456463;0.352273x0.0920732)

month AxesSubplot(0.125,0.345976;0.352273x0.0920732)

year AxesSubplot(0.547727,0.345976;0.352273x0.0920732)
```

```
dayofweek AxesSubplot(0.125,0.235488;0.352273x0.0920732)

dtype: object
```



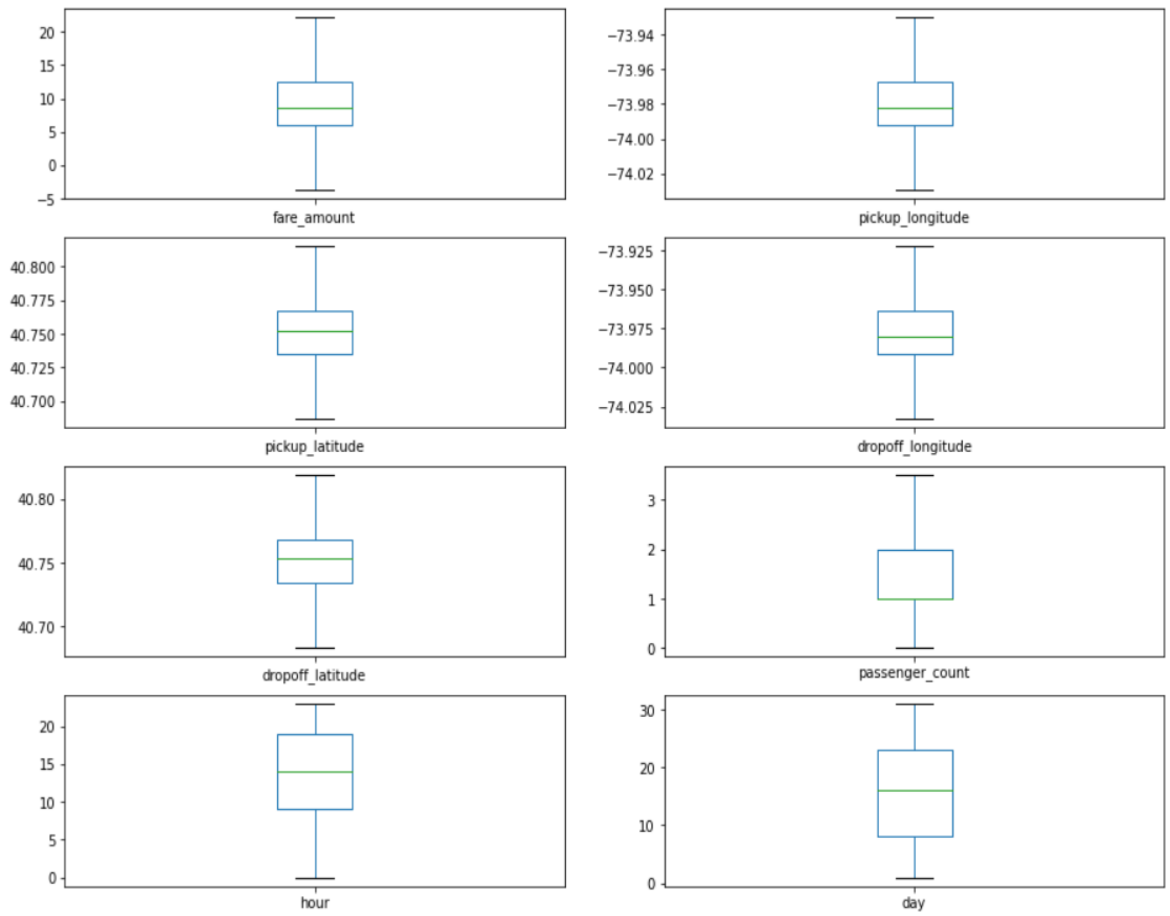


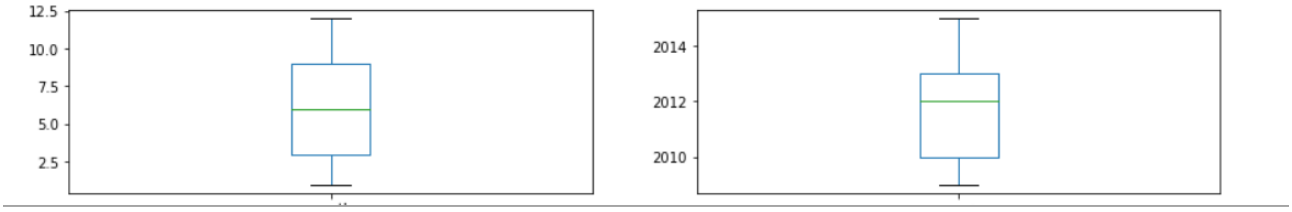

```
#Using the InterQuartile Range to fill the values
def remove_outlier(df1 , col):
    Q1 = df1[col].quantile(0.25)
    Q3 = df1[col].quantile(0.75)
    IQR = Q3 - Q1
    lower_whisker = Q1-1.5*IQR
    upper_whisker = Q3+1.5*IQR
    df[col] = np.clip(df1[col] , lower_whisker , upper_whisker)
    return df1

def treat_outliers_all(df1 , col_list):
    for c in col_list:
        df1 = remove_outlier(df , c)
    return df1
```

```
df = treat_outliers_all(df , df.iloc[: , 0::])
```

```
df.plot(kind = "box",subplots = True,layout = (7,2),figsize=(15,20)) #B
oxplot shows that dataset is free from outliers
```

```
fare_amount AxesSubplot(0.125,0.787927;0.352273x0.0920732)
pickup_longitude AxesSubplot(0.547727,0.787927;0.352273x0.0920732)
pickup_latitude AxesSubplot(0.125,0.677439;0.352273x0.0920732)
dropoff_longitude AxesSubplot(0.547727,0.677439;0.352273x0.0920732)
dropoff_latitude AxesSubplot(0.125,0.566951;0.352273x0.0920732)
passenger_count AxesSubplot(0.547727,0.566951;0.352273x0.0920732)
hour AxesSubplot(0.125,0.456463;0.352273x0.0920732)
day AxesSubplot(0.547727,0.456463;0.352273x0.0920732)
month AxesSubplot(0.125,0.345976;0.352273x0.0920732)
year AxesSubplot(0.547727,0.345976;0.352273x0.0920732)
dayofweek AxesSubplot(0.125,0.235488;0.352273x0.0920732)
dtype: object
```






```
#pip install haversine
import haversine as hs  #Calculate the distance using Haversine to calc
ulate the distance between to points. Can't use Eucladian as it is for
flat surface.
travel_dist = []
for pos in range(len(df['pickup_longitude'])):
        long1,lati1,long2,lati2 = [df['pickup_longitude'][pos],df['pick
up_latitude'][pos],df['dropoff_longitude'][pos],df['dropoff_latitude'][
pos]]
        loc1=(lati1,long1)
        loc2=(lati2,long2)
        c = hs.haversine(loc1,loc2)
        travel_dist.append(c)

print(travel_dist)
df['dist_travel_km'] = travel_dist
df.head()
```

```
IOPub data rate exceeded.
The notebook server will temporarily stop sending output
to the client in order to avoid crashing it.
To change this limit, set the config variable
`--NotebookApp.iopub_data_rate_limit`.

Current values:
NotebookApp.iopub_data_rate_limit=1000000.0 (bytes/sec)
NotebookApp.rate_limit_window=3.0 (secs)
```

	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count	hour	day	month	year	dayofweek	dist_travel_km
0	7.5	-73.999817	40.738354	-73.999512	40.723217	1.0	19	7	5	2015	3	1.683325
1	7.7	-73.994355	40.728225	-73.994710	40.750325	1.0	20	17	7	2009	4	2.457593
2	12.9	-74.005043	40.740770	-73.962565	40.772647	1.0	21	24	8	2009	0	5.036384
3	5.3	-73.976124	40.790844	-73.965316	40.803349	3.0	8	26	6	2009	4	1.661686
4	16.0	-73.929786	40.744085	-73.973082	40.761247	3.5	17	28	8	2014	3	4.116088

```
#Uber doesn't travel over 130 kms so minimize the distance
df= df.loc[(df.dist_travel_km >= 1) | (df.dist_travel_km <= 130)]
print("Remaining observastions in the dataset:", df.shape)
Remaining observastions in the dataset: (200000, 12)
```

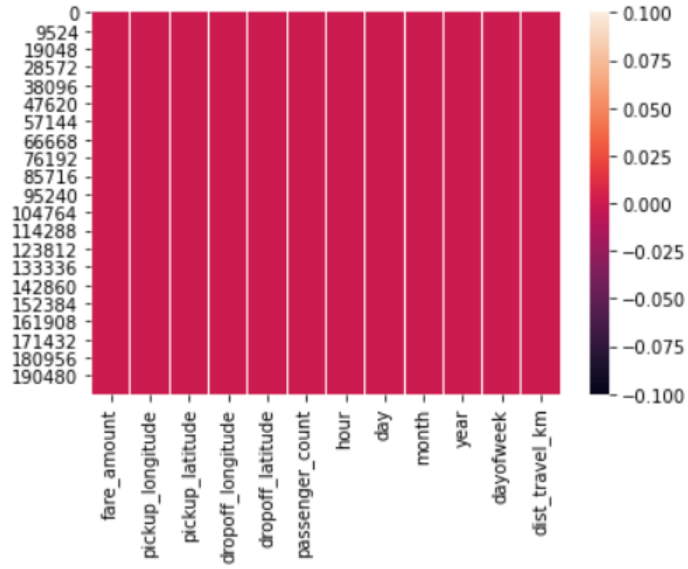
```
#Finding inccorect latitude (Less than or greater than 90) and longitud
e (greater than or less than 180)
incorrect_coordinates = df.loc[(df.pickup_latitude > 90) |(df.pickup_la
titude < -90) |
                                   (df.dropoff_latitude > 90) |(df.drop
off_latitude < -90) |
                                   (df.pickup_longitude > 180) |(df.pic
kup_longitude < -180) |
                                   (df.dropoff_longitude > 90) |(df.dro
poff_longitude < -90)
                                    ]

df.drop(incorrect_coordinates, inplace = True, errors = 'ignore')

df.head()
```

	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count	hour	day	month	year	dayofweek	dist_travel_km
0	7.5	-73.999817	40.738354	-73.999512	40.723217	1.0	19	7	5	2015	3	1.683325
1	7.7	-73.994355	40.728225	-73.994710	40.750325	1.0	20	17	7	2009	4	2.457593
2	12.9	-74.005043	40.740770	-73.962565	40.772647	1.0	21	24	8	2009	0	5.036384
3	5.3	-73.976124	40.790844	-73.965316	40.803349	3.0	8	26	6	2009	4	1.661686
4	16.0	-73.929786	40.744085	-73.973082	40.761247	3.5	17	28	8	2014	3	4.116088

```
df.isnull().sum()
fare_amount 0

pickup_longitude 0

pickup_latitude 0

dropoff_longitude 0

dropoff_latitude 0

passenger_count 0

hour 0

day 0

month 0

year 0

dayofweek 0

dist_travel_km 0

dtype: int64
```

```
sns.heatmap(df.isnull()) #Free for null values
```

<AxesSubplot:>

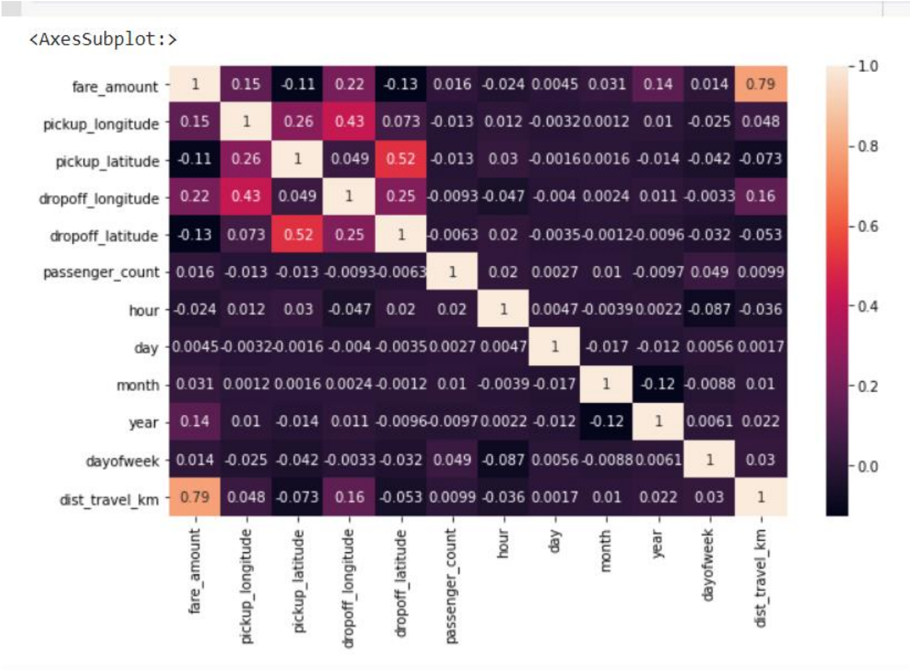



```
corr = df.corr() #Function to find the correlation
```

```
corr
```

	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count	hour	day	month	year	dayofweek	dist_travel_km
fare_amount	1.000000	0.154069	-0.110842	0.218675	-0.125898	0.015778	-0.023623	0.004534	0.030817	0.141277	0.013652	0.786385
pickup_longitude	0.154069	1.000000	0.259497	0.425619	0.073290	-0.013213	0.011579	-0.003204	0.001169	0.010198	-0.024652	0.048446
pickup_latitude	-0.110842	0.259497	1.000000	0.048889	0.515714	-0.012889	0.029681	-0.001553	0.001562	-0.014243	-0.042310	-0.073362
dropoff_longitude	0.218675	0.425619	0.048889	1.000000	0.245667	-0.009303	-0.046558	-0.004007	0.002391	0.011346	-0.003336	0.155191
dropoff_latitude	-0.125898	0.073290	0.515714	0.245667	1.000000	-0.006308	0.019783	-0.003479	-0.001193	-0.009603	-0.031919	-0.052701
passenger_count	0.015778	-0.013213	-0.012889	-0.009303	-0.006308	1.000000	0.020274	0.002712	0.010351	-0.009749	0.048550	0.009884
hour	-0.023623	0.011579	0.029681	-0.046558	0.019783	0.020274	1.000000	0.004677	-0.003926	0.002156	-0.086947	-0.035708
day	0.004534	-0.003204	-0.001553	-0.004007	-0.003479	0.002712	0.004677	1.000000	-0.017360	-0.012170	0.005617	0.001709
month	0.030817	0.001169	0.001562	0.002391	-0.001193	0.010351	-0.003926	-0.017360	1.000000	-0.115859	-0.008786	0.010050
year	0.141277	0.010198	-0.014243	0.011346	-0.009603	-0.009749	0.002156	-0.012170	-0.115859	1.000000	0.006113	0.022294
dayofweek	0.013652	-0.024652	-0.042310	-0.003336	-0.031919	0.048550	-0.086947	0.005617	-0.008786	0.006113	1.000000	0.030382
dist_travel_km	0.786385	0.048446	-0.073362	0.155191	-0.052701	0.009884	-0.035708	0.001709	0.010050	0.022294	0.030382	1.000000

```
fig,axis = plt.subplots(figsize = (10,6))
sns.heatmap(df.corr(),annot = True) #Correlation Heatmap (Light values
means highly correlated)
```



Conclusion:

Exploratory Data Analysis is **not a trivial task**! It requires lots of work and patience, however, it is surely a **powerful tool if correctly applied to your business context**.

This post briefly demonstrated some tips and steps to make analysis easier and undoubtedly highlighted the **crucial importance of a well-defined business problem**, guiding all coding efforts to a specific objective, and also highlighting important insights. This business case also tried to reflect a **practical application of python in daily business activities**, showing how fun, valuable, and interesting it could become.

ASSIGNMENT – 2

Practical No. – 2

Aim –

Given a bank customer, build a neural network-based classifier that can determine whether

they will leave or not in the next 6 months.

Dataset Description: The case study is from an open-source dataset from Kaggle.

The dataset contains 10,000 sample points with 14 distinct features such as

CustomerId, CreditScore, Geography, Gender, Age, Tenure, Balance, etc.

Perform following steps:

1. Read the dataset.

2. Distinguish the feature and target set and divide the data set into training and test sets.

3. Normalize the train and test data.

4. Initialize and build the model. Identify the points of improvement and implement the same.

5. Print the accuracy score and confusion matrix (5 points).

Software Requirements-

- Anaconda Installer
- Windows 10 OS
- Jupyter Notebook

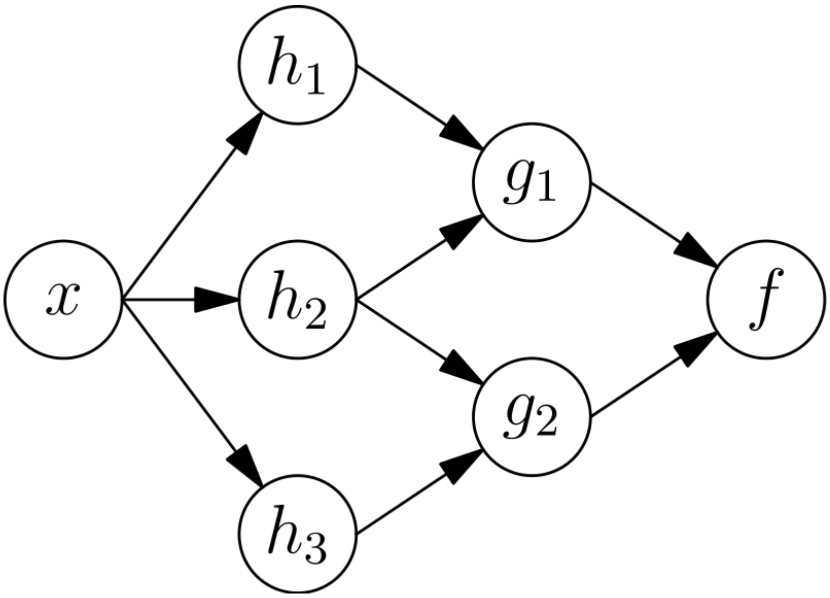
Theory-

Neural Networks:

Neural nets take inspiration from the learning process occurring in human brains. They consists of an artificial network of functions, called parameters, which allows the computer to learn, and to fine tune itself, by analyzing new data. Each parameter, sometimes also referred to as neurons, is a function which produces an output, after receiving one or multiple inputs. Those outputs are then passed to the next layer of neurons, which use them as inputs of their own function, and produce further outputs. Those outputs are then passed on to the next layer of neurons, and so it continues until every layer of neurons have been considered, and the terminal neurons have received their input. Those terminal neurons then output the final result for the model.

Figure 1 shows a visual representation of such a network. The initial input is x, which is then passed to the first layer of neurons (the h bubbles in Figure 1), where three functions consider the input that they receive, and generate an output. That output is then passed to the second layer (the g bubbles in Figure 1). There further output is calculated, based on the output from the first layer. That secondary output is then combined to yield a final output of the model.

Figure 1: A Visual Representation of a Simple Neural Net



An alternative way of thinking about a neural net is to think of it as one massive function which takes inputs and arrives at a final output. The intermediary functions, which are done by the neurons in their many layers, are usually unobserved, and thankfully automated. The mathematics behind them is as interesting as it is complex, and deserves a further look.

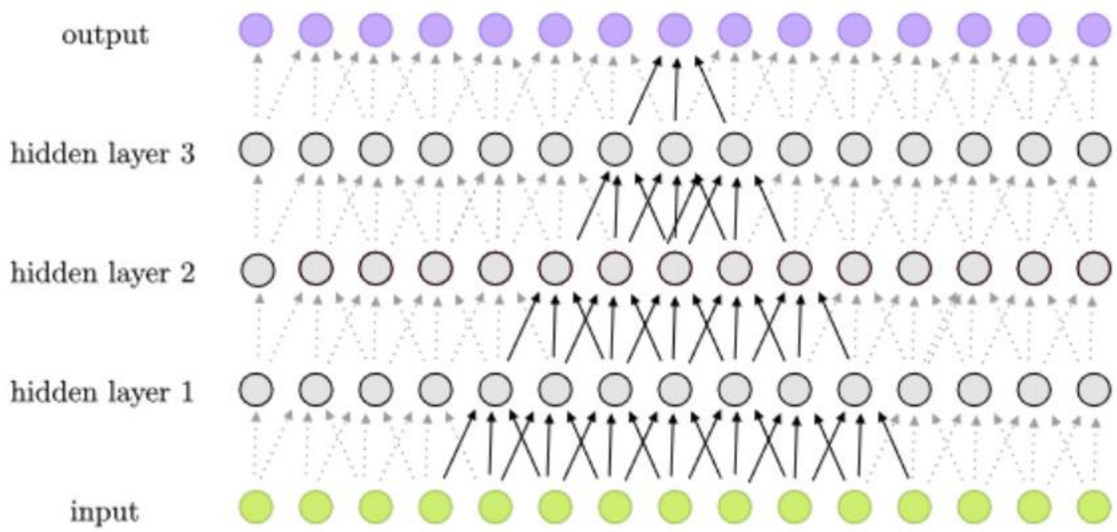
The difficulty lies in determining the optimal value for each bias term, as well as finding the best weighted value for each pass in the neural network. To accomplish this, one must choose a cost function. A cost function is a way of calculating how far a particular solution is from the best possible solution. There are many different possible cost functions, each with advantages and drawbacks, each best suited under certain conditions. Thus, the cost function should be tailored and selected based on individual research needs. Once a cost function has been determined, the neural net can be altered in a way to minimize that cost function.

There are three methods of learning: supervised, unsupervised, and reinforcement learning. The simplest of these learning paradigms is supervised learning, where the neural net is given labelled inputs. The labelled examples, are then used to infer generalizable rules which can be applied to unlabeled cases. It is the simplest learning method, since it can be thought of operating with a ‘teacher’, in the form of a function that allows the net to compare its predictions to the true, and desired results. Unsupervised methods do not require labelled initial inputs, but rather infers the rules and functions, based not only on the given data, but also on the output of the net. This hampers the type of predictions which can be made. Instead of being able to classify, such a model is limited to clustering.

Several types of neural networks exist today. These neural networks are classified based on their density, layers, structure, data flow, depth activation filters among other features. We are going to focus on three types of neural networks.

- Convolutional neural network (CNN)
- Recurrent neural network (RNN)
- Deep Neural Network (DNN)

1. Convolutional Neural Network (CNN)



- For a long time, convolution neural networks were limited in their use due to scalability issues. These neural networks needed a lot of training data for efficiency, and they were only applicable for low-resolution images. Since 2012, however, AlexNet reintroduced multi-layered neural networks and used large data sets from the ImageNet data set, allowing for the creation of complex convolutional neural networks.

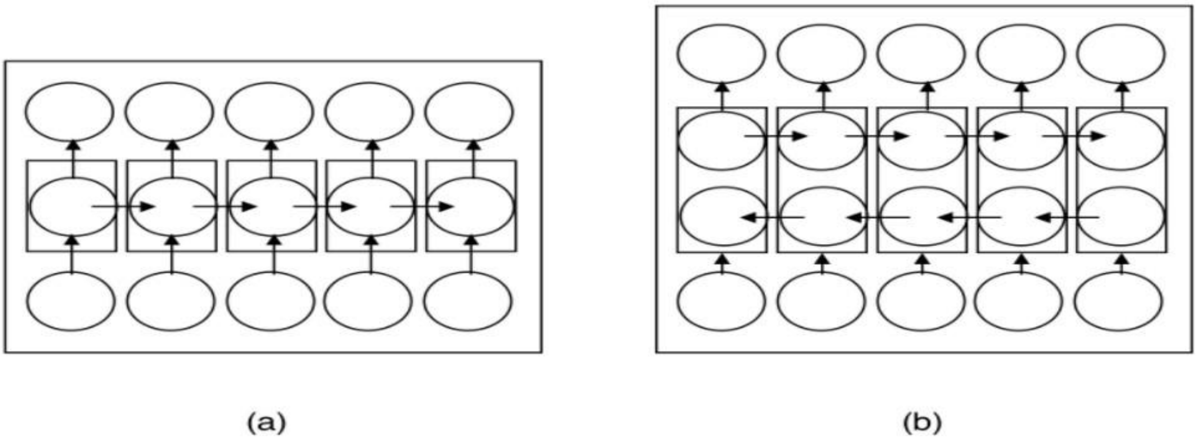
- A convolutional neural network (CNN) is a deep learning algorithm specifically designed to process image data. Convoluted neural networks are used in image recognition and processing.
- The neural networks in a CNN are arranged similarly to the frontal lobe of the human brain, a part of the brain responsible for processing visual stimuli.

2. Recurrent Neural Network (RNN)

What is an RNN?

A recurrent neural network (RNN) is an artificial neural network that uses sequential or time-series data to solve problems in speech recognition and language translation. RNNs have been used in:

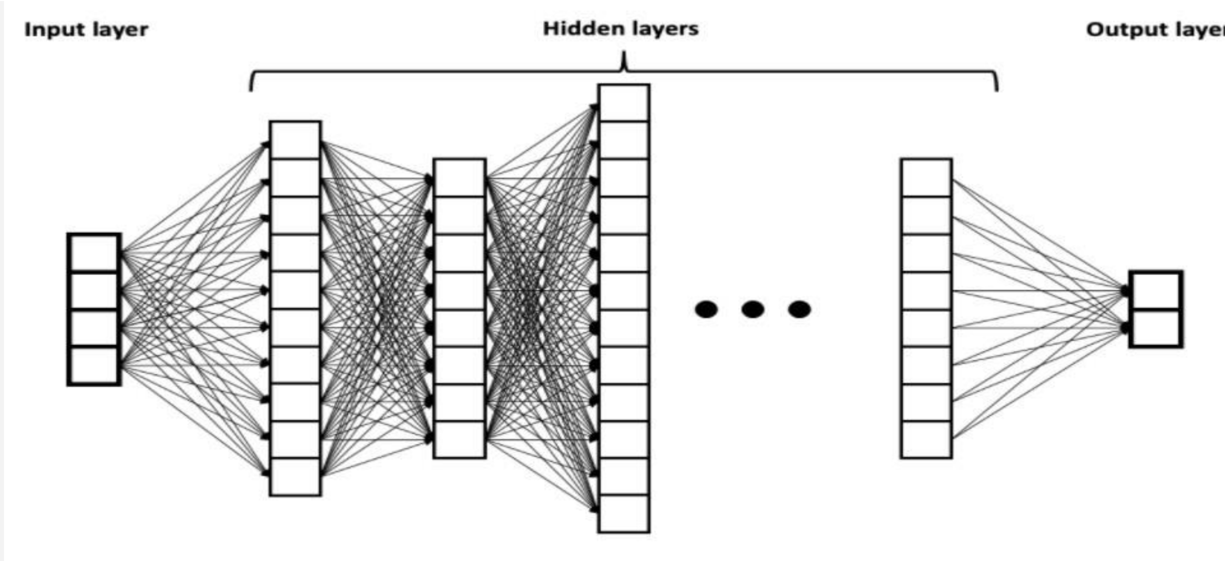
- Language translation
- Natural language processing
- Speech recognition
- Image captioning



Structure overview
(a) unidirectional RNN
(b) bidirectional RNN

4. Deep Neural Network (DNN)

What is a DNN?



Source: Wikimedia Commons

A deep neural network (DNN) is an artificial neural network consisting of multiple layers between the input and output layers. These layers could be recurrent neural network layers or convolutional layers making DNN's a more sophisticated machine learning algorithm. DNNs are capable of recognizing sound, creative thinking, recognizing voice commands, and analysis.

Program –

import pandas as pd

```
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt #Importing the libraries

df = pd.read_csv("Churn_Modelling.csv")
df.head()
```

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Exited
0	1	15634602	Hargrave	619	France	Female	42	2	0.00	1	1	1	101348.88	1
1	2	15647311	Hill	608	Spain	Female	41	1	83807.86	1	0	1	112542.58	0
2	3	15619304	Onio	502	France	Female	42	8	159660.80	3	1	0	113931.57	1
3	4	15701354	Boni	699	France	Female	39	1	0.00	2	0	0	93826.63	0
4	5	15737888	Mitchell	850	Spain	Female	43	2	125510.82	1	1	1	79084.10	0

```
df.shape
(10000, 14)

df.describe()
```

	RowNumber	CustomerId	CreditScore	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Exited
count	10000.00000	1.000000e+04	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.00000	10000.000000	10000.000000	10000.000000
mean	5000.50000	1.569094e+07	650.528800	38.921800	5.012800	76485.889288	1.530200	0.70550	0.515100	100090.239881	0.203700
std	2886.89568	7.193619e+04	96.653299	10.487806	2.892174	62397.405202	0.581654	0.45584	0.499797	57510.492818	0.402769
min	1.00000	1.556570e+07	350.000000	18.000000	0.000000	0.000000	1.000000	0.00000	0.000000	11.580000	0.000000
25%	2500.75000	1.562853e+07	584.000000	32.000000	3.000000	0.000000	1.000000	0.00000	0.000000	51002.110000	0.000000
50%	5000.50000	1.569074e+07	652.000000	37.000000	5.000000	97198.540000	1.000000	1.00000	1.000000	100193.915000	0.000000
75%	7500.25000	1.575323e+07	718.000000	44.000000	7.000000	127644.240000	2.000000	1.00000	1.000000	149388.247500	0.000000
max	10000.00000	1.581569e+07	850.000000	92.000000	10.000000	250898.090000	4.000000	1.00000	1.000000	199992.480000	1.000000

```
df.isnull()
```

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Exited
0	False	False	False	False	False	False	False	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
9995	False	False	False	False	False	False	False	False	False	False	False	False	False	False
9996	False	False	False	False	False	False	False	False	False	False	False	False	False	False
9997	False	False	False	False	False	False	False	False	False	False	False	False	False	False
9998	False	False	False	False	False	False	False	False	False	False	False	False	False	False
9999	False	False	False	False	False	False	False	False	False	False	False	False	False	False

10000 rows × 14 columns

df.isnull().sum()

RowNumber 0

CustomerId 0

Surname 0

CreditScore 0

Geography 0

Gender 0

Age 0

Tenure 0

Balance 0

NumOfProducts 0

HasCrCard 0

IsActiveMember 0

EstimatedSalary 0

Exited 0

dtype: int64

```
df.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 14 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   RowNumber        10000 non-null  int64
 1   CustomerId       10000 non-null  int64
 2   Surname          10000 non-null  object
 3   CreditScore      10000 non-null  int64
 4   Geography        10000 non-null  object
 5   Gender           10000 non-null  object
 6   Age              10000 non-null  int64
 7   Tenure           10000 non-null  int64
 8   Balance          10000 non-null  float64
 9   NumOfProducts    10000 non-null  int64
 10  HasCrCard        10000 non-null  int64
 11  IsActiveMember   10000 non-null  int64
 12  EstimatedSalary  10000 non-null  float64
 13  Exited           10000 non-null  int64
dtypes: float64(2), int64(9), object(3)
memory usage: 1.1+ MB

df.dtypes
RowNumber int64
CustomerId int64
Surname object
CreditScore int64
Geography object
Gender object
Age int64
Tenure int64
Balance float64
NumOfProducts int64
```

HasCrCard int64

IsActiveMember int64

EstimatedSalary float64

Exited int64

dtype: object

df.columns

Index(['RowNumber', 'CustomerId', 'Surname', 'CreditScore', 'Geography', 'Gender', 'Age', 'Tenure', 'Balance', 'NumOfProducts', 'HasCrCard', 'IsActiveMember', 'EstimatedSalary', 'Exited'], dtype='object')

df = df.drop(['RowNumber', 'Surname', 'CustomerId'], axis= 1) #Dropping the unnecessary co lumns

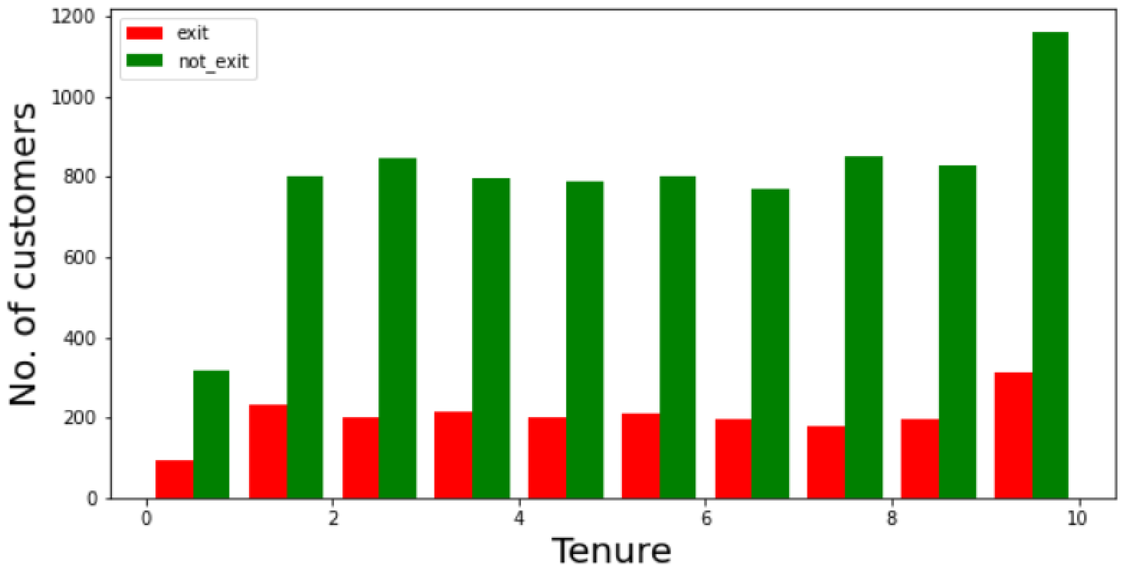
df.head()

	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Exited
0	619	France	Female	42	2	0.00	1	1	1	101348.88	1
1	608	Spain	Female	41	1	83807.86	1	0	1	112542.58	0
2	502	France	Female	42	8	159660.80	3	1	0	113931.57	1
3	699	France	Female	39	1	0.00	2	0	0	93826.63	0
4	850	Spain	Female	43	2	125510.82	1	1	1	79084.10	0

```
def visualization(x, y, xlabel):
    plt.figure(figsize=(10,5))
    plt.hist([x, y], color=['red', 'green'], label = ['exit', 'not_exit'])
    plt.xlabel(xlabel,fontsize=20)
    plt.ylabel("No. of customers", fontsize=20)
    plt.legend()
```

df_churn_exited = df[df['Exited']==1]['Tenure']

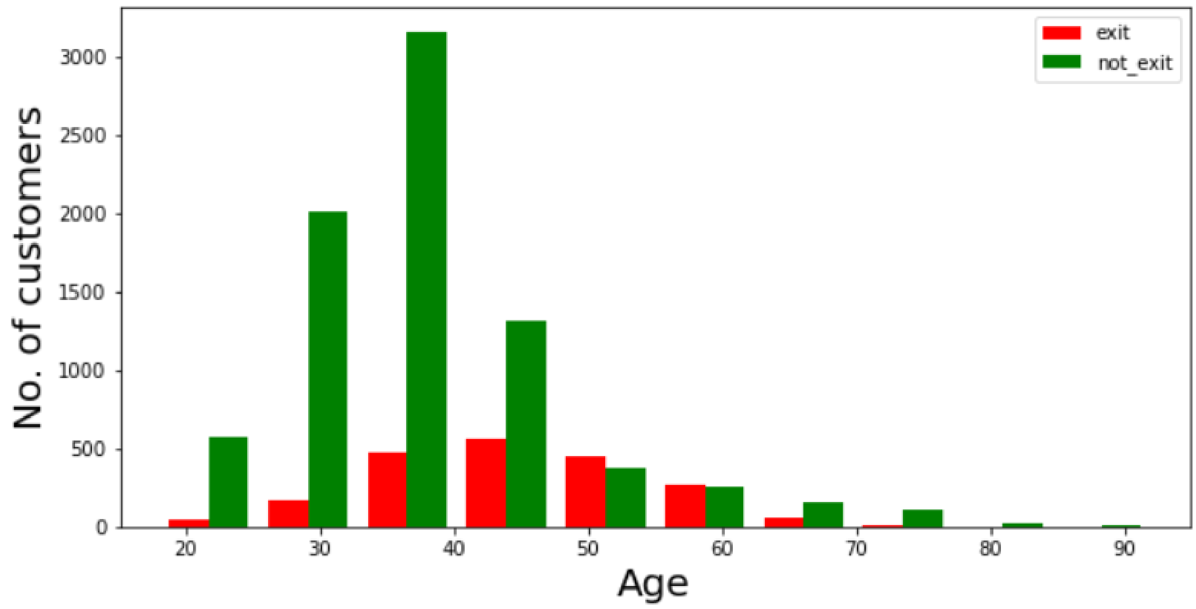
df_churn_not_exited = df[df['Exited']==0]['Tenure'] visualization(df_churn_exited, df_churn_not_exited, "Tenure")



df_churn_exited2 = df[df['Exited']==1]['Age']

df_churn_not_exited2 = df[df['Exited']==0]['Age']

visualization(df_churn_exited2, df_churn_not_exited2, "Age")



Conclusion-

Till now we have trained a deep neural network using TensorFlow to perform basic classification tasks using tabular data. By using the above method, we can train classifier models on any tabular dataset with any number of input features. By leveraging the different types of layers available in Keras, we can optimize and have more control over the model training, thus improving the metric performance. It is recommended to try replicating the above procedure on other datasets and experiment by changing different hyperparameters like learning rate, the number of layers, optimizers etc until we get desirable model performance.

ASSIGNMENT – 3

Practical No. -3

Aim-

Implement K-Nearest Neighbors algorithm on diabetes.csv dataset. Compute confusion matrix, accuracy, error rate, precision and recall on the given dataset.

Software Requirements:

- Anaconda Installer
- Windows 10 OS
- Jupyter Notebook

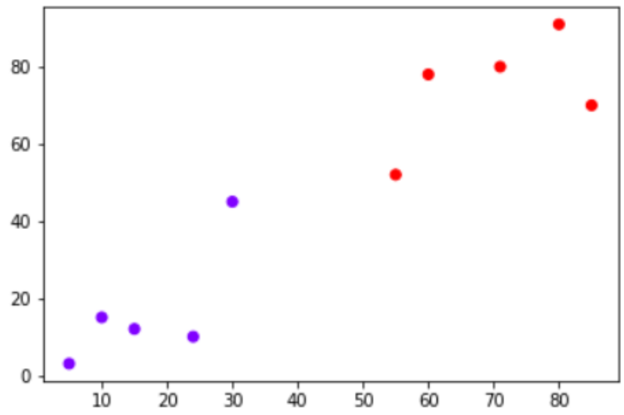
Theory-3

The K-nearest neighbors (KNN) algorithm is a type of supervised machine learning algorithms. KNN is extremely easy to implement in its most basic form, and yet performs quite complex classification tasks. It is a lazy learning algorithm since it doesn't have a specialized training phase. Rather, it uses all of the data for training while classifying a new data point or instance. KNN is a non-parametric learning algorithm, which means that it doesn't assume anything about the underlying data. This is an extremely useful feature since most of the real world data doesn't really follow any theoretical assumption e.g. linear-separability, uniform distribution, etc.

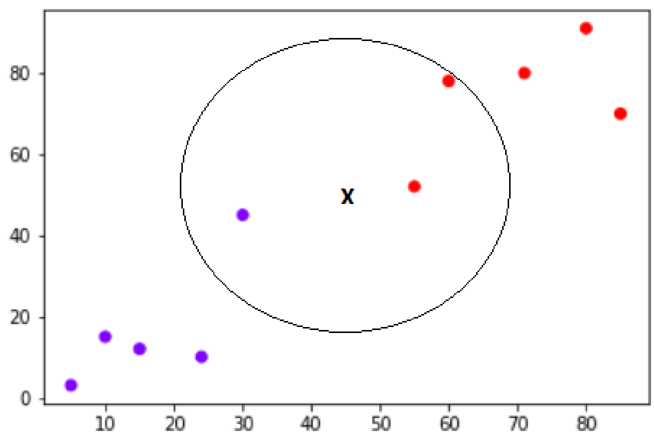
The intuition behind the KNN algorithm is one of the simplest of all the supervised machine learning algorithms. It simply calculates the distance of a new data point to all other training data points. The distance can be of any type e.g Euclidean or Manhattan etc. It then selects the K-nearest data points, where K can be any integer. Finally it assigns the data point to the class to which the majority of the K data points belong.

Let's see this algorithm in action with the help of a simple example.

Suppose you have a dataset with two variables, which when plotted, looks like the one in the following figure.



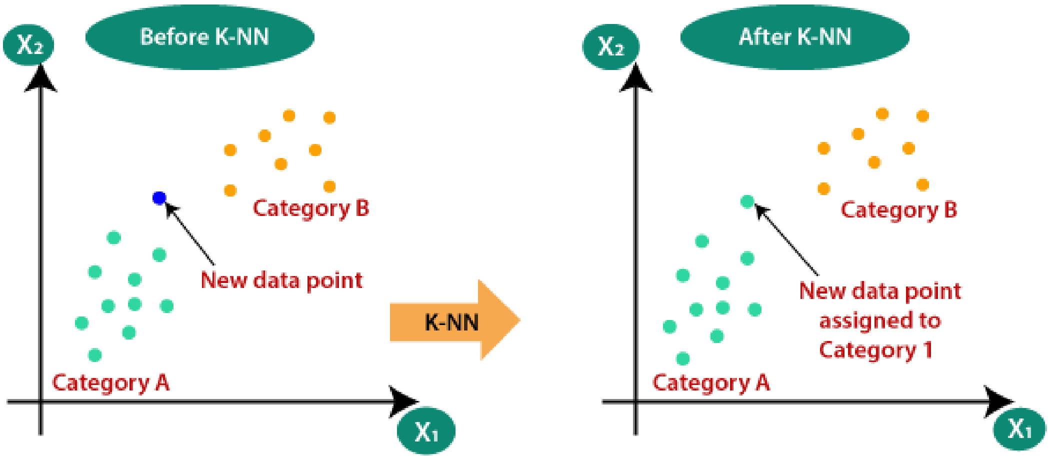
Your task is to classify a new data point with 'X' into "Blue" class or "Red" class. The coordinate values of the data point are x=45 and y=50. Suppose the value of K is 3. The KNN algorithm starts by calculating the distance of point X from all the points. It then finds the 3 nearest points with least distance to point X. This is shown in the figure below. The three nearest points have been encircled.



The final step of the KNN algorithm is to assign new point to the class to which majority of the three nearest points belong. From the figure above we can see that the two of the three nearest points belong to the class "Red" while one belongs to the class "Blue". Therefore the new data point will be classified as "Red".

Why do we need a K-NN Algorithm?

Suppose there are two categories, i.e., Category A and Category B, and we have a new data point x1, so this data point will lie in which of these categories. To solve this type of problem, we need a K-NN algorithm. With the help of K-NN, we can easily identify the category or class of a particular dataset. Consider the below diagram:

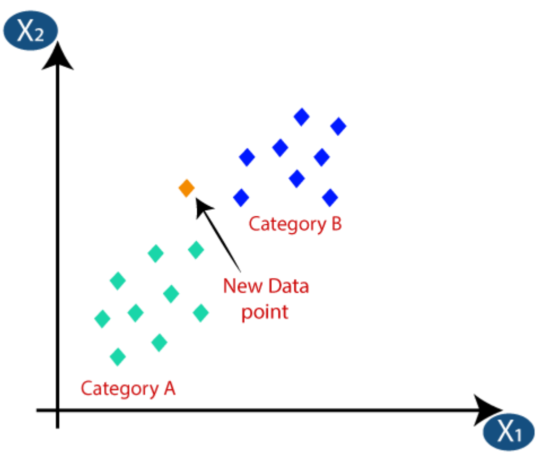


How does K-NN work?

The K-NN working can be explained on the basis of the below algorithm:

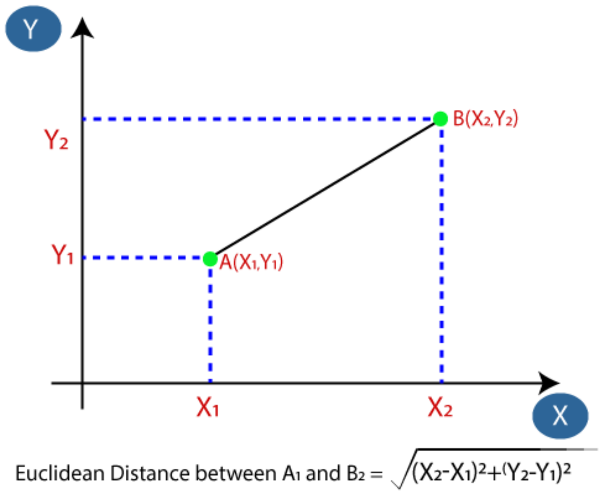
- **Step-1:** Select the number K of the neighbors
- **Step-2:** Calculate the Euclidean distance of **K number of neighbors**
- **Step-3:** Take the K nearest neighbors as per the calculated Euclidean distance.
- **Step-4:** Among these k neighbors, count the number of the data points in each category.
- **Step-5:** Assign the new data points to that category for which the number of the neighbor is maximum.
- **Step-6:** Our model is ready.

Suppose we have a new data point and we need to put it in the required category. Consider the below image:

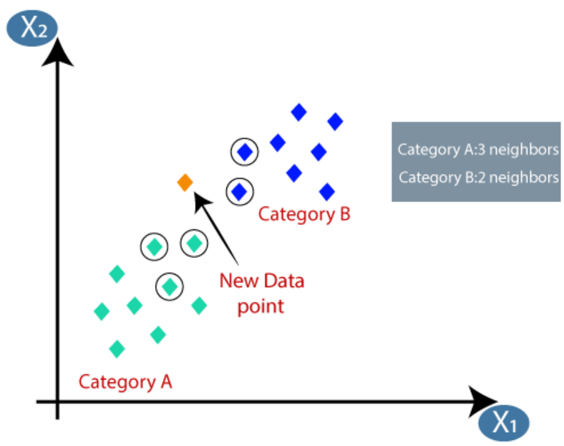


- Firstly, we will choose the number of neighbors, so we will choose the k=5.

- Next, we will calculate the **Euclidean distance** between the data points. The Euclidean distance is the distance between two points, which we have already studied in geometry. It can be calculated as:



- By calculating the Euclidean distance we got the nearest neighbors, as three nearest neighbors in category A and two nearest neighbors in category B. Consider the below image:



- As we can see the 3 nearest neighbors are from category A, hence this new data point must belong to category A.

<u>How to select the value of K in the K-NN Algorithm?</u>

Below are some points to remember while selecting the value of K in the K-NN algorithm:

Pause
Unmute
Loaded: 100.00%

Remaining TimeÂ -4:57

Fullscreen
x

- There is no particular way to determine the best value for "K", so we need to try some values to find the best out of them. The most preferred value for K is 5.
- A very low value for K such as K=1 or K=2, can be noisy and lead to the effects of outliers in the model.
- Large values for K are good, but it may find some difficulties.

Advantages of KNN Algorithm:

- It is simple to implement.
- It is robust to the noisy training data
- It can be more effective if the training data is large.

Disadvantages of KNN Algorithm:

- Always needs to determine the value of K which may be complex some time.
- The computation cost is high because of calculating the distance between the data points for all the training samples.

Program-

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
import warnings
warnings.filterwarnings('ignore')
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn import metrics
df=pd.read_csv('diabetes.csv')
df.columns
Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI', 'Pedigree', 'Age', 'Outcome'], dtype='object')
```

//Check for null values. If present remove null values from the dataset

df.isnull().sum()

Pregnancies 0

Glucose 0

BloodPressure 0

SkinThickness 0

Insulin 0

BMI 0

Pedigree 0

Age 0

Outcome 0

dtype: int64

Output-

//Outcome is the label/target, other columns are features.

X = df.drop('Outcome',axis = 1)

y = df['Outcome']

from sklearn.preprocessing import scale

X = scale(X)

split into train and test

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state = 42)

print("Confusion matrix: ")

cs = metrics.confusion_matrix(y_test,y_pred)

print(cs)

Output-

Confusion matrix:

[[123 28]

 [37 43]]

```
print("Acccuracy ",metrics.accuracy_score(y_test,y_pred))
```

Output-

Acccuracy 0.7186147186147186

/*Classification error rate: proportion of instances misclassified over the whole set of instances. Error rate is calculated as the total number of two incorrect predictions (FN + FP) divided by the total number of a dataset (examples in the dataset.

Also error_rate = 1- accuracy*/

```
total_misclassified = cs[0,1] + cs[1,0]
print(total_misclassified)
total_examples = cs[0,0]+cs[0,1]+cs[1,0]+cs[1,1]
print(total_examples)
print("Error rate",total_misclassified/total_examples)
print("Error rate ",1-metrics.accuracy_score(y_test,y_pred))
print("Precision score",metrics.precision_score(y_test,y_pred))
```

Output-

Precision score 0.6056338028169014

```
print("Recall score ",metrics.recall_score(y_test,y_pred))
```

Output-

Recall score 0.537

```
print("Classification report ",metrics.classification_report(y_test,y_pred))
```

Output-

```
Classification report               precision    recall  f1-score   support

           0       0.77      0.81      0.79       151
           1       0.61      0.54      0.57        80

    accuracy                           0.72       231
```

macro avg	0.69	0.68	0.68	231
weighted avg	0.71	0.72	0.71	231

Conclusion-

KNN is a simple yet powerful classification algorithm. It requires no training for making predictions, which is typically one of the most difficult parts of a machine learning algorithm. The KNN algorithm have been widely used to find document similarity and pattern recognition. It has also been employed for developing recommender systems and for dimensionality reduction and pre-processing steps for computer vision, particularly face recognition tasks.

ASSIGNMENT- 4

Practical No. – 4

Aim-

Implement K-Means clustering/ hierarchical clustering on sales_data_sample.csv dataset. Determine the number of clusters using the elbow method.

Software Requirements-

- Anaconda Installer
- Windows 10 OS
- Jupyter Notebook

Theory-

Clustering is an unsupervised machine learning technique. It is the process of division of the dataset into groups in which the members in the same group possess similarities in features. The commonly used clustering algorithms are K-Means clustering, Hierarchical clustering, Density-based clustering, Model-based clustering, etc. In this article, we are going to discuss K-Means clustering in detail.

K-Means Clustering

It is the simplest and commonly used iterative type unsupervised learning algorithm. In this, we randomly initialize the **K** number of centroids in the data (the number of k is found using the **Elbow** method which will be discussed later in this article) and iterates these centroids until no change happens to the position of the centroid. Let’s go through the steps involved in K means clustering for a better understanding.

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

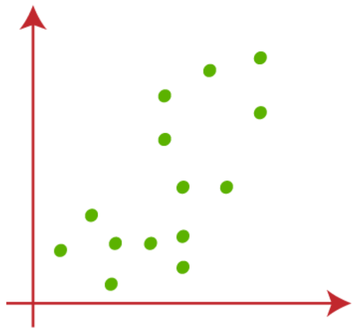
2) Select K number of centroids

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

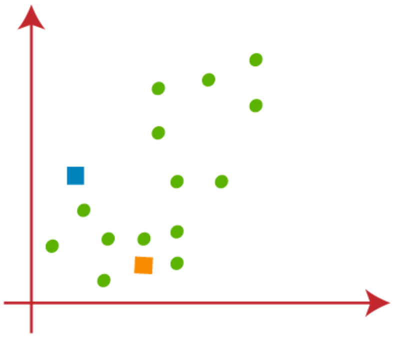
4) Now find the original centroid in each group

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

Suppose we have two variables M1 and M2. The x-y axis scatter plot of these two variables is given below:

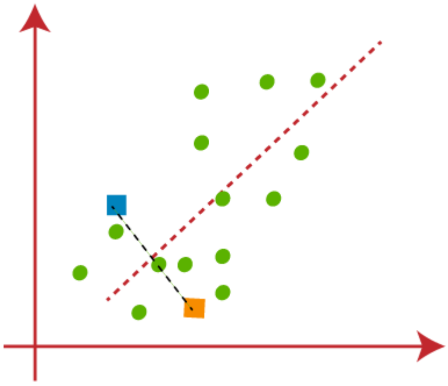


- Let's take number k of clusters, i.e., K=2, to identify the dataset and to put them into different clusters. It means here we will try to group these datasets into two different clusters.
- We need to choose some random k points or centroid to form the cluster. These points can be either the points from the dataset or any other point. So, here we are selecting the below two points as k points, which are not the part of our dataset. Consider the below image:
  -

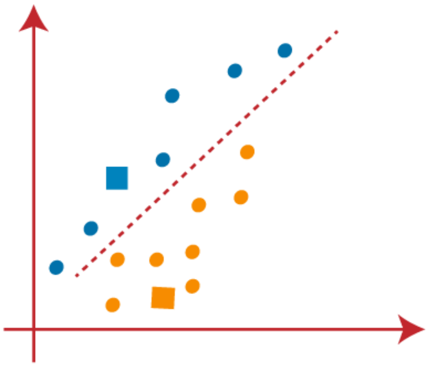


- Now we will assign each data point of the scatter plot to its closest K-point or centroid. We will compute it by applying some mathematics that we have studied to calculate the distance between two points. So, we will draw a median between both the centroids. Consider the below image:

○

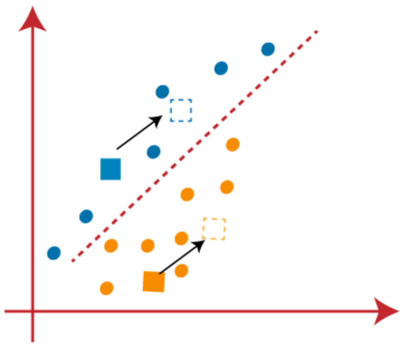


From the above image, it is clear that points left side of the line is near to the K1 or blue centroid, and points to the right of the line are close to the yellow centroid. Let's color them as blue and yellow for clear visualization.



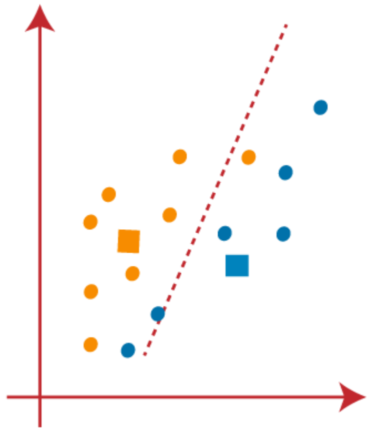
- As we need to find the closest cluster, so we will repeat the process by choosing **a new centroid**. To choose the new centroids, we will compute the center of gravity of these centroids, and will find new centroids as below:

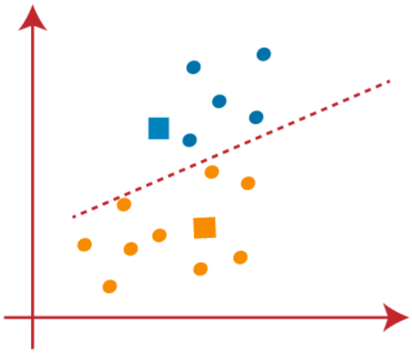
○



- Next, we will reassign each datapoint to the new centroid. For this, we will repeat the same process of finding a median line. The median will be like below image:

   ○



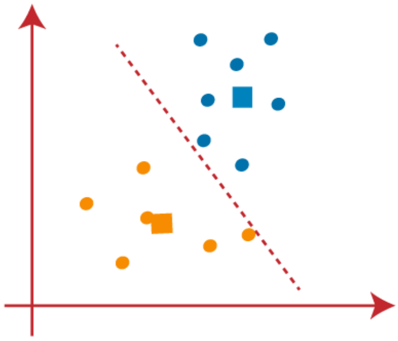


As reassignment has taken place, so we will again go to the step-4, which is finding new centroids or K-points.

- We will repeat the process by finding the center of gravity of centroids, so the new centroids will be as shown in the below image:

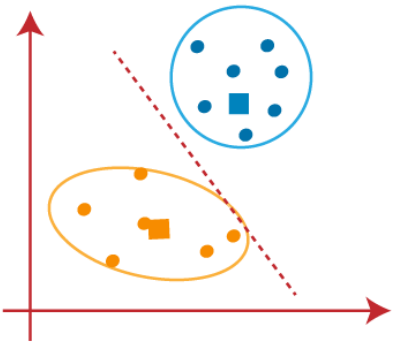
   ○

- As we got the new centroids so again will draw the median line and reassign the data points. So, the image will be:

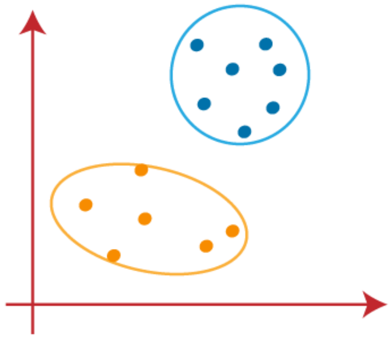


- We can see in the above image; there are no dissimilar data points on either side of the line, which means our model is formed. Consider the below image:

○



As our model is ready, so we can now remove the assumed centroids, and the two final clusters will be as shown in the below image:

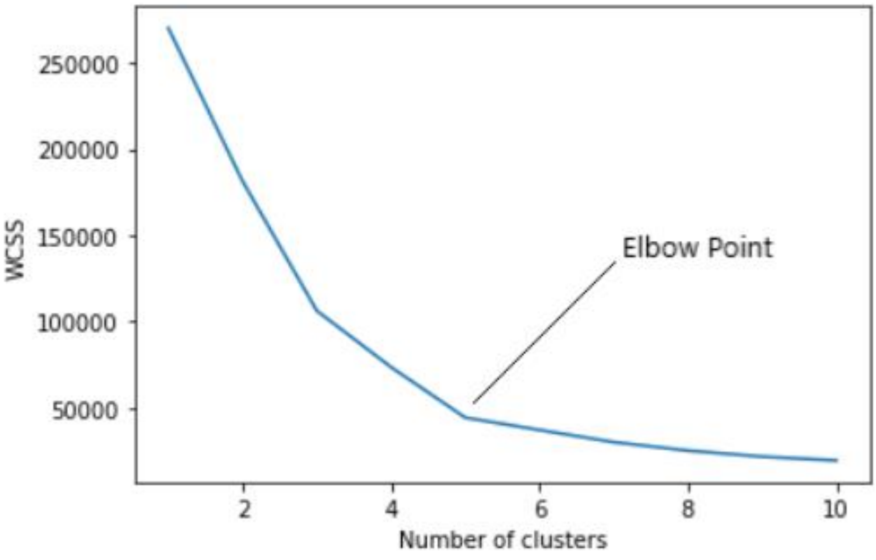


How to choose the value of "K number of clusters" in K-means Clustering?

The performance of the K-means clustering algorithm depends upon highly efficient clusters that it forms. But choosing the optimal number of clusters is a big task. There are some different ways to find the optimal number of clusters, but here we are discussing the most appropriate method to find the number of clusters or value of K. The method is given below:

Elbow Method

In the Elbow method, we are actually varying the number of clusters (K) from 1 – 10. For each value of K, we are calculating WCSS (Within-Cluster Sum of Square). WCSS is the sum of squared distance between each point and the centroid in a cluster. When we plot the WCSS with the K value, the plot looks like an Elbow. As the number of clusters increases, the WCSS value will start to decrease. WCSS value is largest when K = 1. When we analyze the graph we can see that the graph will rapidly change at a point and thus creating an elbow shape. From this point, the graph starts to move almost parallel to the X-axis. The K value corresponding to this point is the optimal K value or an optimal number of clusters.



Program-

import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

df = pd.read_csv("sales_data_sample.csv")

df.head()

```
df.head()
```

	ORDERNUMBER	QUANTITYORDERED	PRICEEACH	ORDERLINENUMBER	SALES	ORDERDATE	STATUS	QTR_ID	MONTH_ID	YEAR_ID	...	ADDRESSLINE1	ADDRESSLINE2	CITY	STATE	POSTALCODE	COUNTRY
0	10107	30	95.70	2	2871.00	2/24/2003 0:00	Shipped	1	2	2003	...	897 Long Airport Avenue	NaN	NYC	NY	10022	USA
1	10121	34	81.35	5	2765.90	5/7/2003 0:00	Shipped	2	5	2003	...	59 rue de l'Abbaye	NaN	Reims	NaN	51100	France
2	10134	41	94.74	2	3884.34	7/1/2003 0:00	Shipped	3	7	2003	...	27 rue du Colonel Pierre Avia	NaN	Paris	NaN	75508	France
3	10145	45	83.26	6	3746.70	8/25/2003 0:00	Shipped	3	8	2003	...	78934 Hillside Dr.	NaN	Pasadena	CA	90003	USA
4	10159	49	100.00	14	5205.27	10/10/2003 0:00	Shipped	4	10	2003	...	7734 Strong St.	NaN	San Francisco	CA	NaN	USA

df.dtypes

ORDERNUMBER int64

QUANTITYORDERED int64

PRICEEACH float64

ORDERLINENUMBER int64

SALES float64

ORDERDATE object

STATUS object

QTR_ID int64

MONTH_ID int64

YEAR_ID int64

PRODUCTLINE object MSRP int64

PRODUCTCODE object

CUSTOMERNAME object

PHONE object

ADDRESSLINE1 object

ADDRESSLINE2 object

CITY object

STATE object

POSTALCODE object

COUNTRY object

TERRITORY object

CONTACTLASTNAME object

CONTACTFIRSTNAME object

DEALSIZE object

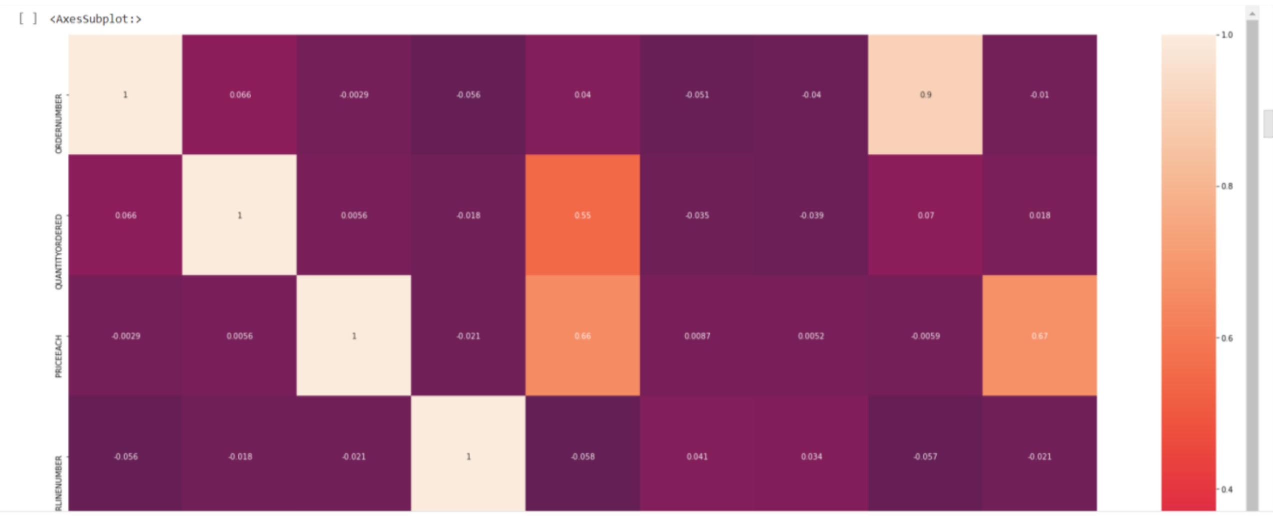
dtype: object

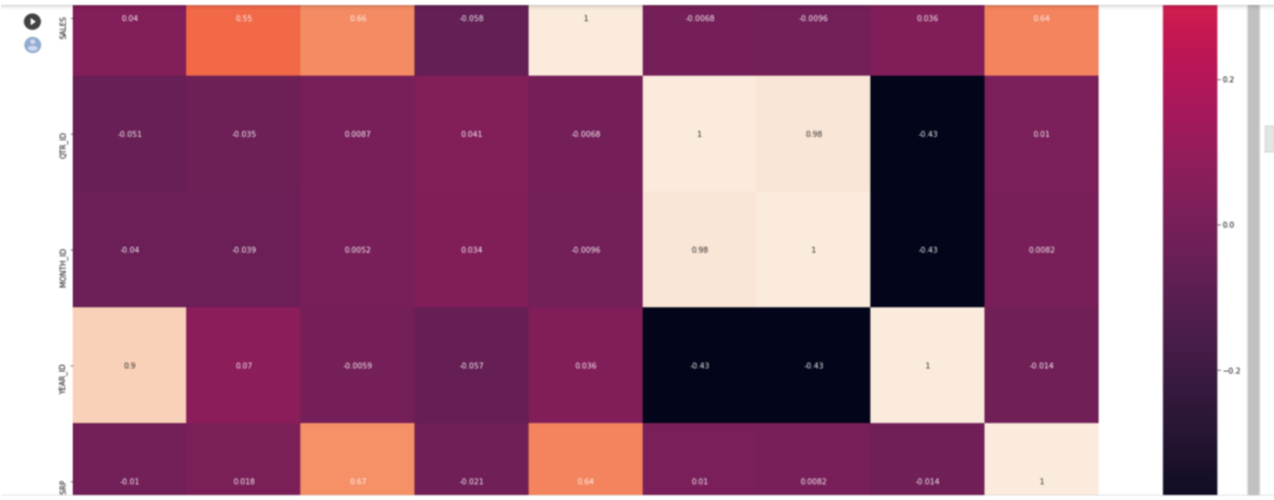
df.info()

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2823 entries, 0 to 2822
Data columns (total 25 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   ORDERNUMBER       2823 non-null   int64
```

```
 1   QUANTITYORDERED   2823 non-null   int64
 2   PRICEEACH         2823 non-null   float64
 3   ORDERLINENUMBER   2823 non-null   int64
 4   SALES             2823 non-null   float64
 5   ORDERDATE         2823 non-null   object
 6   STATUS            2823 non-null   object
 7   QTR_ID            2823 non-null   int64
 8   MONTH_ID          2823 non-null   int64
 9   YEAR_ID           2823 non-null   int64
 10  PRODUCTLINE       2823 non-null   object
 11  MSRP              2823 non-null   int64
 12  PRODUCTCODE       2823 non-null   object
 13  CUSTOMERNAME      2823 non-null   object
 14  PHONE             2823 non-null   object
 15  ADDRESSLINE1      2823 non-null   object
 16  ADDRESSLINE2      302 non-null    object
 17  CITY              2823 non-null   object
 18  STATE             1337 non-null   object
 19  POSTALCODE        2747 non-null   object
 20  COUNTRY           2823 non-null   object
 21  TERRITORY         1749 non-null   object
 22  CONTACTLASTNAME   2823 non-null   object
 23  CONTACTFIRSTNAME  2823 non-null   object
 24  DEALSIZE          2823 non-null   object
dtypes: float64(2), int64(7), object(16)
memory usage: 551.5+ KB
```

```
plt.figure(figsize = (30,26))
sns.heatmap(df.corr(),annot = True)
```





```
df_drop  = ['ADDRESSLINE1', 'ADDRESSLINE2', 'STATUS','POSTALCODE', 'CITY', 'TE
RRITORY', 'PHONE', 'STATE', 'CONTACTFIRSTNAME', 'CONTACTLASTNAME', 'CU
STOMERNAME', 'ORDERNUMBER']
df = df.drop(df_drop, axis=1)
```

```
df.head()
```

	QUANTITYORDERED	PRICEEACH	ORDERLINENUMBER	SALES	ORDERDATE	QTR_ID	MONTH_ID	YEAR_ID	PRODUCTLINE	MSRP	PRODUCTCODE	COUNTRY	DEALSIZE
0	30	95.70	2	2871.00	2/24/2003 0:00	1	2	2003	Motorcycles	95	S10_1678	USA	Small
1	34	81.35	5	2765.90	5/7/2003 0:00	2	5	2003	Motorcycles	95	S10_1678	France	Small
2	41	94.74	2	3884.34	7/1/2003 0:00	3	7	2003	Motorcycles	95	S10_1678	France	Medium
3	45	83.26	6	3746.70	8/25/2003 0:00	3	8	2003	Motorcycles	95	S10_1678	USA	Medium
4	49	100.00	14	5205.27	10/10/2003 0:00	4	10	2003	Motorcycles	95	S10_1678	USA	Medium

df.shape
(2823, 13)

df.isnull().sum()
QUANTITYORDERED 0

PRICEEACH 0

ORDERLINENUMBER 0

SALES 0

ORDERDATE 0

QTR_ID 0

MONTH_ID 0

YEAR_ID 0

PRODUCTLINE 0

MSRP 0

PRODUCTCODE 0

COUNTRY 0

DEALSIZE 0

dtype: int64

```
df.dtypes
```

QUANTITYORDERED int64

PRICEEACH float64

ORDERLINENUMBER int64

SALES float64

ORDERDATE object

QTR_ID int64

MONTH_ID int64

YEAR_ID int64

PRODUCTLINE object

MSRP int64

PRODUCTCODE object

COUNTRY object

DEALSIZE object

dtype: object

```
country = pd.get_dummies(df['COUNTRY'])
productline = pd.get_dummies(df['PRODUCTLINE'])
Dealsize = pd.get_dummies(df['DEALSIZE'])

df = pd.concat([df,country,productline,Dealsize], axis = 1)

df.head()
```

	QUANTITYORDERED	PRICEEACH	ORDERLINENUMBER	SALES	ORDERDATE	QTR_ID	MONTH_ID	YEAR_ID	PRODUCTLINE	MSRP	...	Classic Cars	Motorcycles	Planes	Ships	Trains	Trucks and Buses	Vintage Cars	Large
0	30	95.70	2	2871.00	2/24/2003 0:00	1	2	2003	Motorcycles	95	...	0	1	0	0	0	0	0	0
1	34	81.35	5	2765.90	5/7/2003 0:00	2	5	2003	Motorcycles	95	...	0	1	0	0	0	0	0	0
2	41	94.74	2	3884.34	7/1/2003 0:00	3	7	2003	Motorcycles	95	...	0	1	0	0	0	0	0	0
3	45	83.26	6	3746.70	8/25/2003 0:00	3	8	2003	Motorcycles	95	...	0	1	0	0	0	0	0	0
4	49	100.00	14	5205.27	10/10/2003 0:00	4	10	2003	Motorcycles	95	...	0	1	0	0	0	0	0	0

```
df_drop  = ['COUNTRY','PRODUCTLINE','DEALSIZE']
df = df.drop(df_drop, axis=1)
```

```
df.dtypes
QUANTITYORDERED int64

PRICEEACH float64

ORDERLINENUMBER int64

SALES float64

ORDERDATE object

QTR_ID int64

MONTH_ID int64

YEAR_ID int64

MSRP int64

PRODUCTCODE object

Australia uint8

Austria uint8

Belgium uint8

Canada uint8

Denmark uint8

Finland uint8

France uint8

Germany uint8

Ireland uint8

Italy uint8
```

Japan uint8

Norway uint8

Philippines uint8

Singapore uint8

Spain uint8

Sweden uint8

Switzerland uint8

UK uint8

USA uint8

Classic Cars uint8

Motorcycles uint8

Planes uint8

Ships uint8

Trains uint8

Trucks and Buses uint8

Vintage Cars uint8

Large uint8 Medium uint8

Small uint8

dtype: object

df['PRODUCTCODE'] = pd.Categorical(df['PRODUCTCODE']).codes

df.dtypes
QUANTITYORDERED int64

PRICEEACH float64

ORDERLINENUMBER int64

SALES float64

ORDERDATE object

QTR_ID int64

MONTH_ID int64

YEAR_ID int64

MSRP int64

PRODUCTCODE int8

Australia uint8

Austria uint8

Belgium uint8

Canada uint8

Denmark uint8

Finland uint8

France uint8

Germany uint8

Ireland uint8

Italy uint8

Japan uint8

Norway uint8

Philippines uint8

Singapore uint8

Spain uint8

Sweden uint8

Switzerland uint8

UK uint8

USA uint8

Classic Cars uint8

Motorcycles uint8

Planes uint8

Ships uint8

Trains uint8

Trucks and Buses uint8

Vintage Cars uint8

Large uint8

Medium uint8

Small uint8

dtype: object

df.drop('ORDERDATE', axis=1, inplace=True)

df.dtypes
QUANTITYORDERED int64

PRICEEACH float64

ORDERLINENUMBER int64

SALES float64

QTR_ID int64

MONTH_ID int64

YEAR_ID int64

MSRP int64

PRODUCTCODE int8

Australia uint8

Austria uint8

Belgium uint8

Canada uint8

Denmark uint8

Finland uint8

France uint8

Germany uint8

Ireland uint8

Italy uint8

Japan uint8

Norway uint8

Philippines uint8

Singapore uint8

Spain uint8

Sweden uint8

Switzerland uint8

UK uint8

USA uint8

Classic Cars uint8

Motorcycles uint8

Planes uint8

Ships uint8

Trains uint8

Trucks and Buses uint8

Vintage Cars uint8

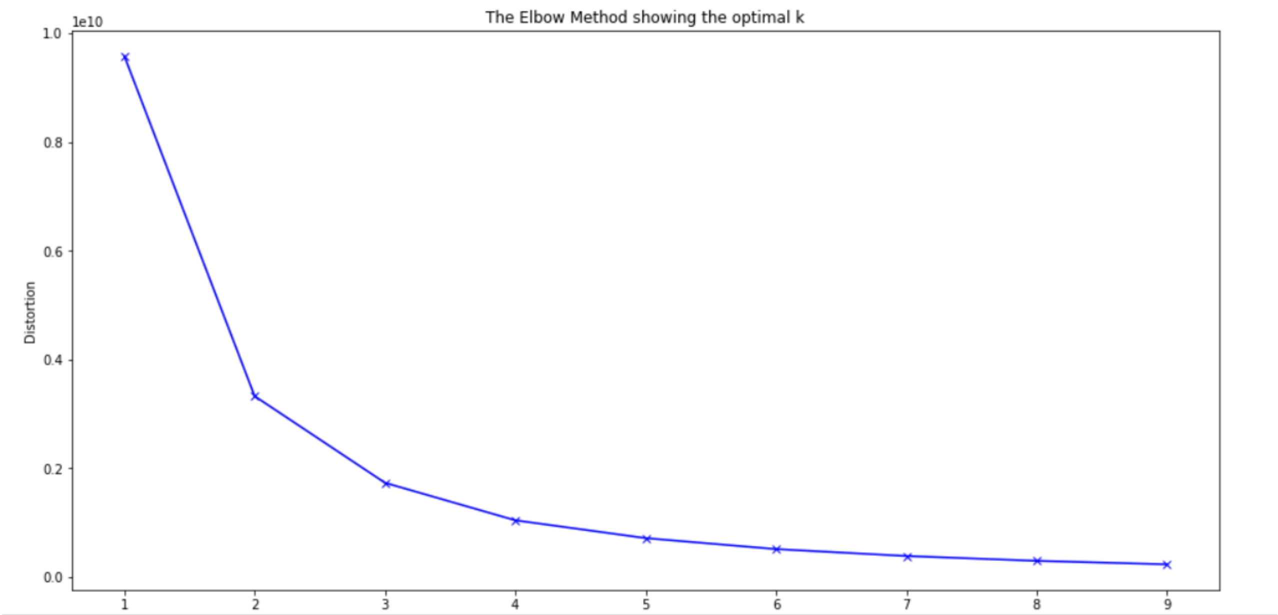
Large uint8

Medium uint8

Small uint8

dtype: object

```
from sklearn.cluster import KMeans

WCSS = [] # Withhin Cluster Sum of Squares from the centroid

distortions = []
K = range(1,10)
for k in K:
    kmeanModel = KMeans(n_clusters=k)
    kmeanModel.fit(df)
    distortions.append(kmeanModel.inertia_)

plt.figure(figsize=(16,8))
plt.plot(K, distortions, 'bx-')
plt.xlabel('k')
plt.ylabel('Distortion')
plt.title('The Elbow Method showing the optimal k')
plt.show()
```



kmeanModel = KMeans(n_clusters=3)
y_kmeans = kmeanModel.fit_predict

print(y_kmeans)

plt.figure(figsize = (30,26))
sns.heatmap(df.corr(),annot = True)

pip install yellowbrick

from yellowbrick.cluster import KElbowVisualizer
model = KMeans()
visualizer = KElbowVisualizer(model,k=(1,0),timings = False)
visualizer.fit(df)
visualizer.show()

df.head()

	QUANTITYORDERED	PRICEEACH	ORDERLINENUMBER	SALES	QTR_ID	MONTH_ID	YEAR_ID	MSRP	PRODUCTCODE	Australia	...	Classic Cars	Motorcycles	Planes	Ships	Trains	Trucks and Buses	Vintage Cars	Large
0	30	95.70	2	2871.00	1	2	2003	95	0	0	...	0	1	0	0	0	0	0	0
1	34	81.35	5	2765.90	2	5	2003	95	0	0	...	0	1	0	0	0	0	0	0
2	41	94.74	2	3884.34	3	7	2003	95	0	0	...	0	1	0	0	0	0	0	0
3	45	83.26	6	3746.70	3	8	2003	95	0	0	...	0	1	0	0	0	0	0	0
4	49	100.00	14	5205.27	4	10	2003	95	0	0	...	0	1	0	0	0	0	0	0

5 rows × 38 columns

```
from sklearn.preprocessing import Normalizer

df_scaled = Normalizer(df)

df_x = pd.DataFrame(df_scaled,columns = df.columns )
```

Conclusion-

KNN is a simple yet powerful classification algorithm. It requires no training for making predictions, which is typically one of the most difficult parts of a machine learning algorithm. The KNN algorithm have been widely used to find document similarity and pattern recognition. It has also been employed for developing recommender systems and for dimensionality reduction and pre-processing steps for computer vision, particularly face recognition tasks.

Mini Project-

Build a machine learning model that predicts the type of people who survived the Titanic shipwreck using passenger data (i.e. name, age, gender, socio-economic class, etc.).

Importing the Libraries

```
# linear algebra
import numpy as np

# data processing
import pandas as pd

# data visualization
import seaborn as sns
%matplotlib inline
from matplotlib import pyplot as plt
from matplotlib import style

# Algorithms
from sklearn import linear_model
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import Perceptron
from sklearn.linear_model import SGDClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC, LinearSVC
from sklearn.naive_bayes import GaussianNB
```

Getting the Data

```
test_df = pd.read_csv("test.csv")
train_df = pd.read_csv("train.csv")
```

Data Exploration/Analysis

```
train_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
PassengerId    891 non-null int64
Survived       891 non-null int64
Pclass         891 non-null int64
Name           891 non-null object
Sex            891 non-null object
Age            714 non-null float64
SibSp          891 non-null int64
Parch          891 non-null int64
Ticket         891 non-null object
Fare           891 non-null float64
Cabin          204 non-null object
Embarked       889 non-null object
dtypes: float64(2), int64(5), object(5)
memory usage: 83.6+ KB
```

The training-set has 891 examples and 11 features + the target variable (survived). 2 of the features are floats, 5 are integers and 5 are objects. Below I have listed the features with a short description:

```
survival:    Survival
PassengerId: Unique Id of a passenger.
pclass:    Ticket class
sex:    Sex
Age:    Age in years
sibsp:    # of siblings / spouses aboard the Titanic
parch:    # of parents / children aboard the Titanic
ticket:    Ticket number
fare:    Passenger fare
cabin:    Cabin number
embarked:    Port of Embarkationtrain_df.describe()
```

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
count	891.000000	891.000000	891.000000	714.000000	891.000000	891.000000	891.000000
mean	446.000000	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208
std	257.353842	0.486592	0.836071	14.526497	1.102743	0.806057	49.693429
min	1.000000	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	223.500000	0.000000	2.000000	20.125000	0.000000	0.000000	7.910400
50%	446.000000	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200
75%	668.500000	1.000000	3.000000	38.000000	1.000000	0.000000	31.000000
max	891.000000	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

Above we can see that **38% out of the training-set survived the Titanic**. We can also see that the passenger ages range from 0.4 to 80. On top of that we can already detect some features, that contain missing values, like the 'Age' feature.

```
train_df.head(8)
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S
5	6	0	3	Moran, Mr. James	male	NaN	0	0	330877	8.4583	NaN	Q
6	7	0	1	McCarthy, Mr. Timothy J	male	54.0	0	0	17463	51.8625	E46	S
7	8	0	3	Palsson, Master. Gosta Leonard	male	2.0	3	1	349909	21.0750	NaN	S
8	9	1	3	Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)	female	27.0	0	2	347742	11.1333	NaN	S

From the table above, we can note a few things. First of all, that we **need to convert a lot of features into numeric** ones later on, so that the machine learning algorithms can process them. Furthermore, we can see that the **features have widely different ranges**, that we will need to convert into roughly the same scale. We can also spot some more features, that contain missing values (NaN = not a number), that wee need to deal with.

Let's take a more detailed look at what data is actually missing:

```
total = train_df.isnull().sum().sort_values(ascending=False)
percent_1 = train_df.isnull().sum()/train_df.isnull().count()*100
percent_2 = (round(percent_1, 1)).sort_values(ascending=False)
missing_data = pd.concat([total, percent_2], axis=1, keys=['Total', '%'])
missing_data.head(5)
```

	Total	%
Cabin	687	77.1
Age	177	19.9
Embarked	2	0.2
Fare	0	0.0
Ticket	0	0.0

The Embarked feature has only 2 missing values, which can easily be filled. It will be much more tricky, to deal with the 'Age' feature, which has 177 missing values. The 'Cabin' feature needs further investigation, but it looks like that we might want to drop it from the dataset, since 77 % of it are missing.

```
train_df.columns.values
```

```
array(['PassengerId', 'Survived', 'Pclass', 'Name', 'Sex', 'Age', 'SibSp',
       'Parch', 'Ticket', 'Fare', 'Cabin', 'Embarked'], dtype=object)
```

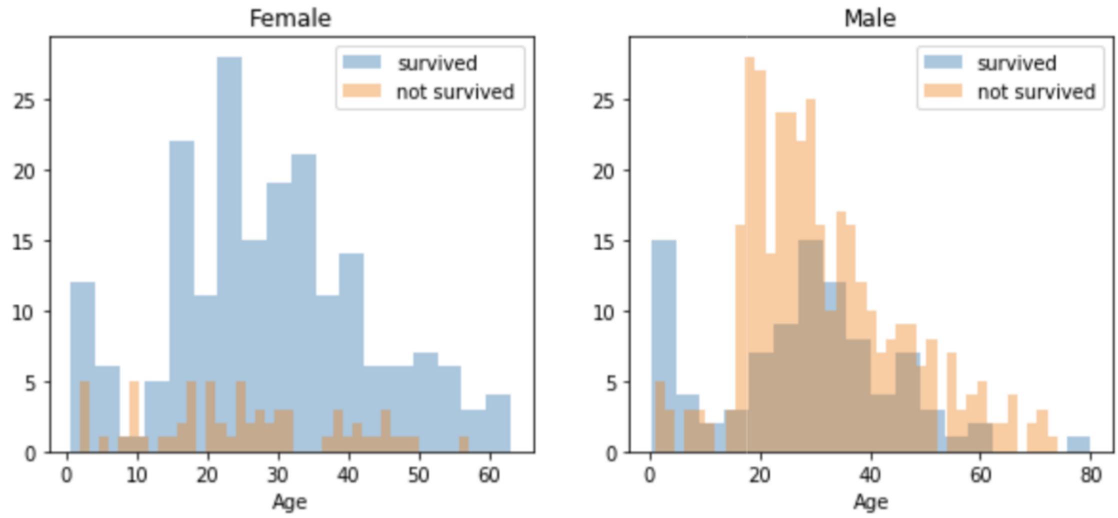
Above you can see the 11 features + the target variable (survived). **What features could contribute to a high survival rate ?**

To me it would make sense if everything except 'PassengerId', 'Ticket' and 'Name' would be correlated with a high survival rate.

1. Age and Sex:

```
survived = 'survived'
not_survived = 'not survived'
fig, axes = plt.subplots(nrows=1, ncols=2,figsize=(10, 4))
women = train_df[train_df['Sex']=='female']
men = train_df[train_df['Sex']=='male']
ax = sns.distplot(women[women['Survived']==1].Age.dropna(), bins=18, label = survived, ax = axes[0], kde =False)
ax = sns.distplot(women[women['Survived']==0].Age.dropna(), bins=40, label = not_survived, ax = axes[0], kde =False)
ax.legend()
```

```
ax.set_title('Female')
ax = sns.distplot(men[men['Survived']==1].Age.dropna(), bins=18, label = survived, ax = axes[1], kde = False)
ax = sns.distplot(men[men['Survived']==0].Age.dropna(), bins=40, label = not_survived, ax = axes[1], kde = False)
ax.legend()
_ = ax.set_title('Male')
```



You can see that men have a high probability of survival when they are between 18 and 30 years old, which is also a little bit true for women but not fully. For women the survival chances are higher between 14 and 40.

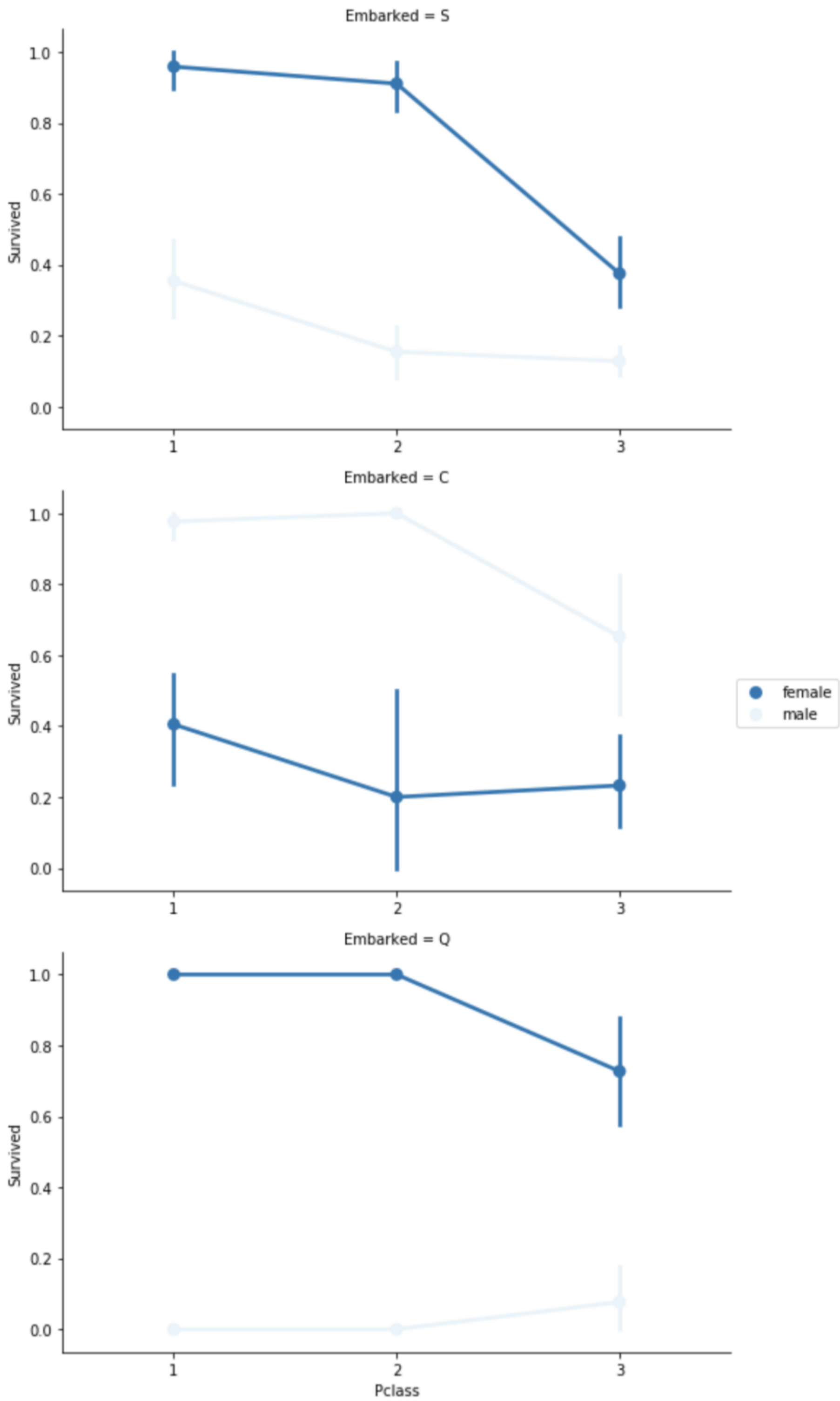
For men the probability of survival is very low between the age of 5 and 18, but that isn't true for women. Another thing to note is that infants also have a little bit higher probability of survival.

Since there seem to be **certain ages, which have increased odds of survival** and because I want every feature to be roughly on the same scale, I will create age groups later on.

3. Embarked, Pclass and Sex:

```
FacetGrid = sns.FacetGrid(train_df, row='Embarked', size=4.5, aspect=1.6)
FacetGrid.map(sns.pointplot, 'Pclass', 'Survived', 'Sex', palette=None,  order=None, hue_order=None )
FacetGrid.add_legend()
```

```
<seaborn.axisgrid.FacetGrid at 0x10ba485c0>
```



Embarked seems to be correlated with survival, depending on the gender.

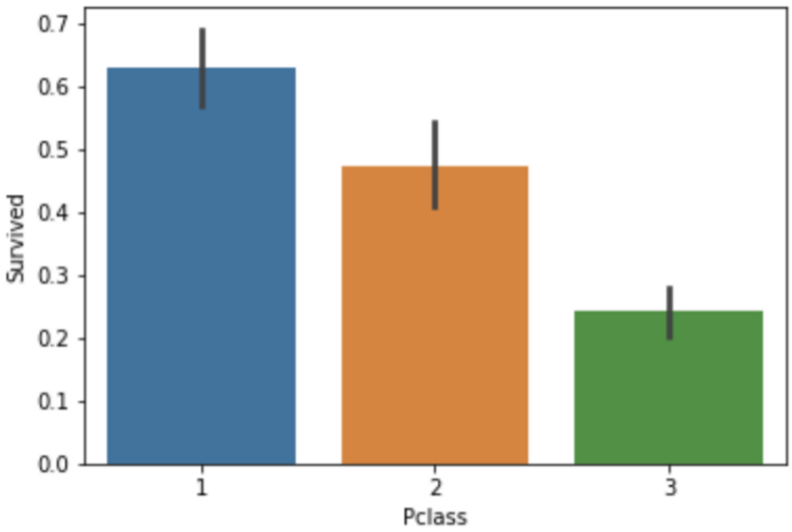
Women on port Q and on port S have a higher chance of survival. The inverse is true, if they are at port C. Men have a high survival probability if they are on port C, but a low probability if they are on port Q or S.

Pclass also seems to be correlated with survival. We will generate another plot of it below.

4. Pclass:

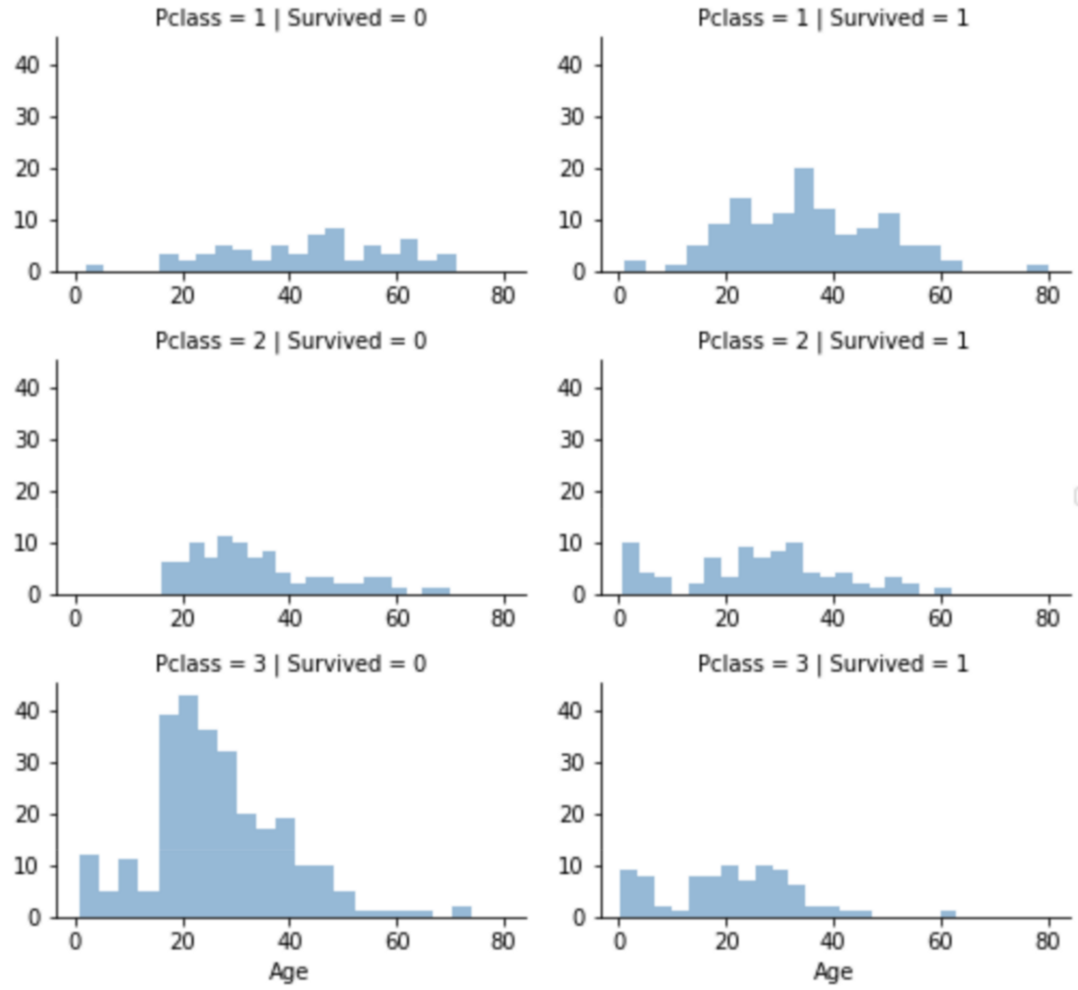
```
sns.barplot(x='Pclass', y='Survived', data=train_df)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x10d1dc7b8>
```



Here we see clearly, that Pclass is contributing to a persons chance of survival, especially if this person is in class 1. We will create another pclass plot below.

```
grid = sns.FacetGrid(train_df, col='Survived', row='Pclass', size=2.2, aspect=1.6)
grid.map(plt.hist, 'Age', alpha=.5, bins=20)
grid.add_legend();
```



The plot above confirms our assumption about pclass 1, but we can also spot a high probability that a person in pclass 3 will not survive.

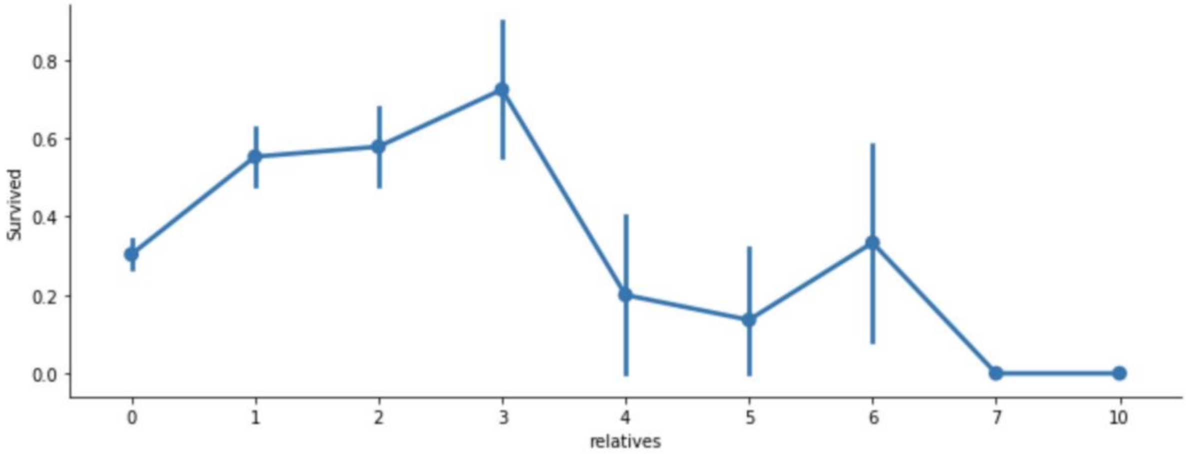
5. SibSp and Parch:

SibSp and Parch would make more sense as a combined feature, that shows the total number of relatives, a person has on the Titanic. I will create it below and also a feature that sows if someone is not alone.

```
data = [train_df, test_df]
for dataset in data:
    dataset['relatives'] = dataset['SibSp'] + dataset['Parch']
    dataset.loc[dataset['relatives'] > 0, 'not_alone'] = 0
    dataset.loc[dataset['relatives'] == 0, 'not_alone'] = 1
    dataset['not_alone'] = dataset['not_alone'].astype(int)train_df['not_alone'].value_counts()
```

```
1    537
0    354
Name: not_alone, dtype: int64
```

```
axes = sns.factorplot('relatives','Survived',
                      data=train_df, aspect = 2.5, )
```



Here we can see that you had a high probabilty of survival with 1 to 3 realitves, but a lower one if you had less than 1 or more than 3 (except for some cases with 6 relatives).

Data Preprocessing

First, I will drop 'PassengerId' from the train set, because it does not contribute to a persons survival probability. I will not drop it from the test set, since it is required there for the submission.

```
train_df = train_df.drop(['PassengerId'], axis=1)
```

Missing Data:

Cabin:

As a reminder, we have to deal with Cabin (687), Embarked (2) and Age (177). First I thought, we have to delete the 'Cabin' variable but then I found something interesting. A cabin number looks like 'C123' and the **letter refers to the deck**. Therefore we're going to extract these and create a new feature, that contains a persons deck. Afterwords we will convert the feature into a numeric variable. The missing values will be converted to zero. In the picture below you can see the actual decks of the titanic, ranging from A to G.

```
import re
deck = {"A": 1, "B": 2, "C": 3, "D": 4, "E": 5, "F": 6, "G": 7, "U": 8}
data = [train_df, test_df]

for dataset in data:
    dataset['Cabin'] = dataset['Cabin'].fillna("U0")
    dataset['Deck'] = dataset['Cabin'].map(lambda x: re.compile("([a-zA-Z]+)").search(x).group())
    dataset['Deck'] = dataset['Deck'].map(deck)
    dataset['Deck'] = dataset['Deck'].fillna(0)
    dataset['Deck'] = dataset['Deck'].astype(int)# we can now drop the cabin feature
train_df = train_df.drop(['Cabin'], axis=1)
test_df = test_df.drop(['Cabin'], axis=1)
```

Age:

Now we can tackle the issue with the age features missing values. I will create an array that contains random numbers, which are computed based on the mean age value in regards to the standard deviation and is_null.

```
data = [train_df, test_df]

for dataset in data:
    mean = train_df["Age"].mean()
    std = test_df["Age"].std()
    is_null = dataset["Age"].isnull().sum()
    # compute random numbers between the mean, std and is_null
    rand_age = np.random.randint(mean - std, mean + std, size = is_null)
    # fill NaN values in Age column with random values generated
    age_slice = dataset["Age"].copy()
    age_slice[np.isnan(age_slice)] = rand_age
    dataset["Age"] = age_slice
    dataset["Age"] = train_df["Age"].astype(int)train_df["Age"].isnull().sum()
```

```
0
```

Embarked:

Since the Embarked feature has only 2 missing values, we will just fill these with the most common one.

```
train_df['Embarked'].describe()
```

```
count      889
unique       3
top          S
freq       644
Name: Embarked, dtype: object
```

```
common_value = 'S'
data = [train_df, test_df]

for dataset in data:
    dataset['Embarked'] = dataset['Embarked'].fillna(common_value)
```

Converting Features:

```
train_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 13 columns):
Survived     891 non-null int64
Pclass       891 non-null int64
Name         891 non-null object
Sex          891 non-null object
Age          891 non-null int64
SibSp        891 non-null int64
Parch        891 non-null int64
Ticket       891 non-null object
Fare         891 non-null float64
Embarked     891 non-null object
relatives    891 non-null int64
not_alone    891 non-null int64
Deck         891 non-null int64
dtypes: float64(1), int64(8), object(4)
memory usage: 90.6+ KB
```

Above you can see that ‘Fare’ is a float and we have to deal with 4 categorical features: Name, Sex, Ticket and Embarked. Lets investigate and transfrom one after another.

Fare:

Converting “Fare” from float to int64, using the “astype()” function pandas provides:

```
data = [train_df, test_df]

for dataset in data:
    dataset['Fare'] = dataset['Fare'].fillna(0)
    dataset['Fare'] = dataset['Fare'].astype(int)
```

Name:

We will use the Name feature to extract the Titles from the Name, so that we can build a new feature out of that.

```
data = [train_df, test_df]
titles = {"Mr": 1, "Miss": 2, "Mrs": 3, "Master": 4, "Rare": 5}

for dataset in data:
    # extract titles
    dataset['Title'] = dataset.Name.str.extract(' ([A-Za-z]+)\.', expand=False)
    # replace titles with a more common title or as Rare
    dataset['Title'] = dataset['Title'].replace(['Lady', 'Countess','Capt', 'Col','Don', 'Dr',\
                                            'Major', 'Rev', 'Sir', 'Jonkheer', 'Dona'], 'Rare')
    dataset['Title'] = dataset['Title'].replace('Mlle', 'Miss')
    dataset['Title'] = dataset['Title'].replace('Ms', 'Miss')
    dataset['Title'] = dataset['Title'].replace('Mme', 'Mrs')
    # convert titles into numbers
    dataset['Title'] = dataset['Title'].map(titles)
    # filling NaN with 0, to get safe
    dataset['Title'] = dataset['Title'].fillna(0)train_df = train_df.drop(['Name'], axis=1)
test_df = test_df.drop(['Name'], axis=1)
```

Sex:

Convert ‘Sex’ feature into numeric.

```
genders = {"male": 0, "female": 1}
data = [train_df, test_df]

for dataset in data:
    dataset['Sex'] = dataset['Sex'].map(genders)
```

Ticket:

```
train_df['Ticket'].describe()
```

```
count       891
unique      681
top        1601
freq          7
Name: Ticket, dtype: object
```

Since the Ticket attribute has 681 unique tickets, it will be a bit tricky to convert them into useful categories. So we will drop it from the dataset.

```
train_df = train_df.drop(['Ticket'], axis=1)
test_df = test_df.drop(['Ticket'], axis=1)
```

Embarked:

Convert ‘Embarked’ feature into numeric.

```
ports = {"S": 0, "C": 1, "Q": 2}
data = [train_df, test_df]

for dataset in data:
    dataset['Embarked'] = dataset['Embarked'].map(ports)
```

Creating Categories:

We will now create categories within the following features:

Age:

Now we need to convert the ‘age’ feature. First we will convert it from float into integer. Then we will create the new ‘AgeGroup” variable, by categorizing every age into a group. Note that it is important to place attention on how you form these groups, since you don’t want for example that 80% of your data falls into group 1.

```
data = [train_df, test_df]
for dataset in data:
    dataset['Age'] = dataset['Age'].astype(int)
    dataset.loc[ dataset['Age'] <= 11, 'Age'] = 0
    dataset.loc[(dataset['Age'] > 11) & (dataset['Age'] <= 18), 'Age'] = 1
    dataset.loc[(dataset['Age'] > 18) & (dataset['Age'] <= 22), 'Age'] = 2
    dataset.loc[(dataset['Age'] > 22) & (dataset['Age'] <= 27), 'Age'] = 3
    dataset.loc[(dataset['Age'] > 27) & (dataset['Age'] <= 33), 'Age'] = 4
    dataset.loc[(dataset['Age'] > 33) & (dataset['Age'] <= 40), 'Age'] = 5
    dataset.loc[(dataset['Age'] > 40) & (dataset['Age'] <= 66), 'Age'] = 6
    dataset.loc[ dataset['Age'] > 66, 'Age'] = 6

# let's see how it's distributed train_df['Age'].value_counts()
```

```
4    165
6    158
5    147
3    129
2    124
1    100
0     68
Name: Age, dtype: int64
```

Fare:

For the ‘Fare’ feature, we need to do the same as with the ‘Age’ feature. But it isn’t that easy, because if we cut the range of the fare values into a few equally big categories, 80% of the values would fall into the first category. Fortunately, we can use sklearn “qcut()” function, that we can use to see, how we can form the categories.

```
train_df.head(10)
```

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked	relatives	not_alone	Deck	Title
0	0	3	0	2	1	0	7	0	1	0	8	1
1	1	1	1	5	1	0	71	1	1	0	3	3
2	1	3	1	3	0	0	7	0	0	1	8	2
3	1	1	1	5	1	0	53	0	1	0	3	3
4	0	3	0	5	0	0	8	0	0	1	8	1
5	0	3	0	4	0	0	8	2	0	1	8	1
6	0	1	0	6	0	0	51	0	0	1	5	1
7	0	3	0	0	3	1	21	0	4	0	8	4
8	1	3	1	3	0	2	11	0	2	0	8	3
9	1	2	1	1	1	0	30	1	1	0	8	3

```
data = [train_df, test_df]

for dataset in data:
    dataset.loc[ dataset['Fare'] <= 7.91, 'Fare'] = 0
    dataset.loc[(dataset['Fare'] > 7.91) & (dataset['Fare'] <= 14.454), 'Fare'] = 1
    dataset.loc[(dataset['Fare'] > 14.454) & (dataset['Fare'] <= 31), 'Fare']   = 2
    dataset.loc[(dataset['Fare'] > 31) & (dataset['Fare'] <= 99), 'Fare']   = 3
    dataset.loc[(dataset['Fare'] > 99) & (dataset['Fare'] <= 250), 'Fare']   = 4
    dataset.loc[ dataset['Fare'] > 250, 'Fare'] = 5
    dataset['Fare'] = dataset['Fare'].astype(int)
```

Creating new Features

I will add two new features to the dataset, that I compute out of other features.

1. Age times Class

```
data = [train_df, test_df]
for dataset in data:
    dataset['Age_Class']= dataset['Age']* dataset['Pclass']
```

2. Fare per Person

```
for dataset in data:
    dataset['Fare_Per_Person'] = dataset['Fare']/(dataset['relatives']+1)
    dataset['Fare_Per_Person'] = dataset['Fare_Per_Person'].astype(int)# Let's take a last look at the training set, before we start training the models.
train_df.head(10)
```

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked	relatives	not_alone	Deck	Title	Age_Class	Fare_Per_Pers
0	0	3	0	2	1	0	0	0	1	0	8	1	6	0
1	1	1	1	5	1	0	3	1	1	0	3	3	5	1
2	1	3	1	3	0	0	0	0	0	1	8	2	9	0
3	1	1	1	5	1	0	3	0	1	0	3	3	5	1
4	0	3	0	5	0	0	1	0	0	1	8	1	15	1
5	0	3	0	4	0	0	1	2	0	1	8	1	12	1
6	0	1	0	6	0	0	3	0	0	1	5	1	6	3
7	0	3	0	0	3	1	2	0	4	0	8	4	0	0
8	1	3	1	3	0	2	1	0	2	0	8	3	9	0
9	1	2	1	1	1	0	2	1	1	0	8	3	2	1
10	1	3	1	0	1	1	2	0	2	0	7	2	0	0

Building Machine Learning Models

Now we will train several Machine Learning models and compare their results. Note that because the dataset does not provide labels for their testing-set, we need to use the predictions on the training set to compare the algorithms with each other. Later on, we will use cross validation.

```
X_train = train_df.drop("Survived", axis=1)
Y_train = train_df["Survived"]
X_test  = test_df.drop("PassengerId", axis=1).copy()
```

Stochastic Gradient Descent (SGD):

```
sgd = linear_model.SGDClassifier(max_iter=5, tol=None)
sgd.fit(X_train, Y_train)
Y_pred = sgd.predict(X_test)
```

```
sgd.score(X_train, Y_train)

acc_sgd = round(sgd.score(X_train, Y_train) * 100, 2)
```

Random Forest:

```
random_forest = RandomForestClassifier(n_estimators=100)
random_forest.fit(X_train, Y_train)

Y_prediction = random_forest.predict(X_test)

random_forest.score(X_train, Y_train)
acc_random_forest = round(random_forest.score(X_train, Y_train) * 100, 2)
```

Logistic Regression:

```
logreg = LogisticRegression()
logreg.fit(X_train, Y_train)

Y_pred = logreg.predict(X_test)

acc_log = round(logreg.score(X_train, Y_train) * 100, 2)
```

K Nearest Neighbor:

```
# KNN knn = KNeighborsClassifier(n_neighbors = 3) knn.fit(X_train, Y_train)  Y_pred =
knn.predict(X_test)  acc_knn = round(knn.score(X_train, Y_train) * 100, 2)
```