Assignment 3 Report

Automation of model

INFO 7390

Advance Data Science & Architecture

Professor: Srikanth Krishnamurthy

By: Team 5

Akash Jagtap

Jerin Rajan

Nitin Prince Reuben

Contents

Abstract

Data in the social networking services is increasing day by day. So, there is heavy requirement to study the highly dynamic behavior of the users towards these services. This work is a preliminary work to study and model the user activity patterns. We had targeted the most active social networking service 'Facebook' importantly the 'Facebook Pages' for analysis. The task here is to estimate the comment count that a post is expected to receive in next few hours. The analysis is done by modeling the comment patterns using variety of regressive modeling techniques.

We would create a web page for hosting our application using Amazon S3 (ec2 instance) through which we will make provisions for our clients to provide us data. Also, we can show them the output in the UI itself. In this process, we will be automating the models for our dataset making things easier and faster.

Introduction

The increasing use of social networking services had drawn the public attention explosively from last 15 years. The merging up of physical things with the social networking services had enabled the conversion of routine objects into information appliances. These services are acting like a multi-tool with daily applications like: advertisement, news, communication, banking, commenting, marketing etc. These services are revolutionizing day by day and many more on the way. These all services have one thing in common that is daily huge content generation, that is more likely to be stored on Hadoop cluster. As in Facebook, 500+ terabytes of new data ingested into the databases every day, 100+ petabytes of disk space in one of FB's largest Hadoop (HDFS) clusters and there are 2.5 billion content items shared per day (status updates + wall posts + photos + videos + comments). The Twitter went from 5,000 tweets per day in 2007 to 500,000,000 tweets per day in 2013. Flickr features 5.5 billion images as that of January 31,2011 and around 3k-5k images are adding up per minute.

In this research, we targeted the most active social networking service 'Facebook' importantly the 'Facebook Pages' for analysis. Our research is oriented towards the estimation of comment volume that a post is expected to receive in next few hours. Before continuing to the problem of comment volume prediction, some domain specific concepts are discussed below:

- ***Public Group/Facebook Page:*** It is a public profile specifically created for businesses, brands, celebrities etc.
- ***Post/Feed:*** These are basically the individual stories published on page by administrators of page.
- ***Comment:*** It is an important activity in social sites, that gives potential to become a discussion forum and it is only one measure of popularity/interest towards post is to which extent readers are inspired to leave comments on document/post.

Dataset

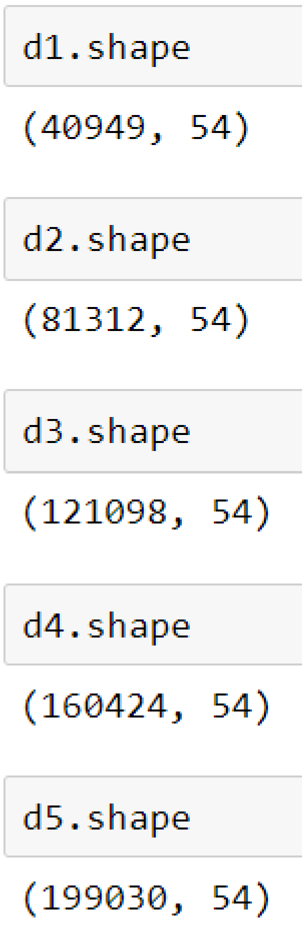
Our dataset is available in the following link

https://archive.ics.uci.edu/ml/datasets/Facebook+Comment+Volume+Dataset

Exploratory Data Analysis

We Have 5 variation of training dataset and 10 variation of testing dataset.

Following is the details of training dataset.



```
d1.shape
```

```
(40949, 54)
```

```
d2.shape
```

```
(81312, 54)
```

```
d3.shape
```

```
(121098, 54)
```

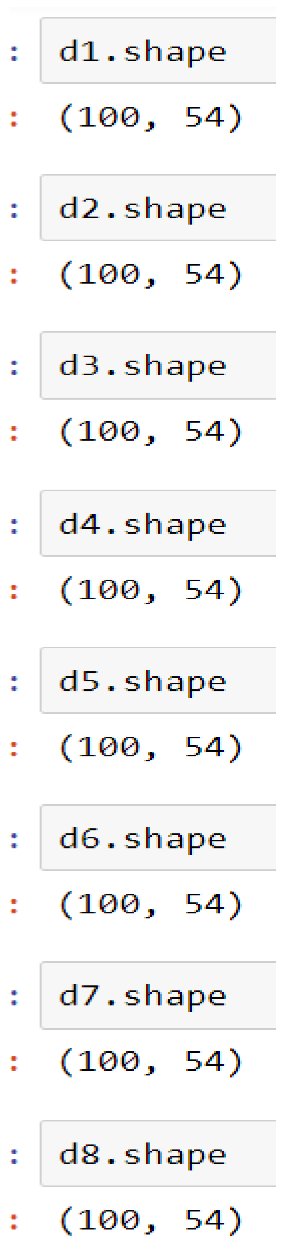
```
d4.shape
```

```
(160424, 54)
```

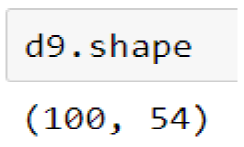
```
d5.shape
```

```
(199030, 54)
```

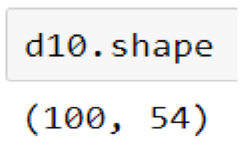
Following are the details of testing dataset.



```
: d1.shape
: (100, 54)

: d2.shape
: (100, 54)

: d3.shape
: (100, 54)

: d4.shape
: (100, 54)

: d5.shape
: (100, 54)

: d6.shape
: (100, 54)

: d7.shape
: (100, 54)

: d8.shape
: (100, 54)
```



```
d9.shape
(100, 54)
```



```
d10.shape
(100, 54)
```

We can see that all 10 variants of dataset have 100 rows.

Our dataset has 54 columns. It is important for us to understand what they represent and what is their importance.

Column No	Column Name	Data Description
1	Page Popularity/Likes	Defines the popularity or support for the source of the document.
2	Page Checking	Describes how many individuals so far visited this place. This feature is only associated with the places e.g.: some institution, place, theater etc.
3	Page Talking About	Defines the daily interest of individuals towards source of the document/ Post. The people who come back to the page, after liking the page. This include activities such as comments, likes to a post, shares, etc. by visitors to the page.
4	Page Category	Defines the category of the source of the document e.g.: place, institution, brand etc.
5-29	Derived	These features are aggregated by page, by calculating min, max, average, median and standard deviation of essential features.
30	CC1	The total number of comments before selected base date/time.
31	CC2	The number of comments in last 24 hours, relative to base date/time.
32	CC3	The number of comments in last 48 to last 24 hours relative to base date/time.
33	CC4	The number of comments in the first 24 hours after the publication of post but before base date/time.
34	CC5	The difference between CC2 and CC3.
35	Base Time	Selected time in order to simulate the scenario.
36	Post Length	Character count in the post
37	Post Share Count	This feature counts the no of shares of the post, that how many peoples had shared this post on to their timeline.
38	Post Promotion Count	To reach more people with posts in News Feed, individual promote their post and this feature tells that whether the post is promoted (1) or not (0).
39	H Local	This describes the H hrs., for which we have the target variable/ comments received.
40-46	Post Published weekday	This represents the day (Sunday...Saturday) on which the post was published. (One hot encoding)

47-53	Base Date Time weekday	This represents the day (Sunday...Saturday) on selected base Date/Time. (One hot encoding)
54	Target Variable	The no of comments in next H hrs. (H is given in Feature no 39).

We are merging the 5-training dataset into one to make the calculations easier and to proceed in a streamline way.

```
frames = [d1 , d2 , d3 , d4 , d5]
```

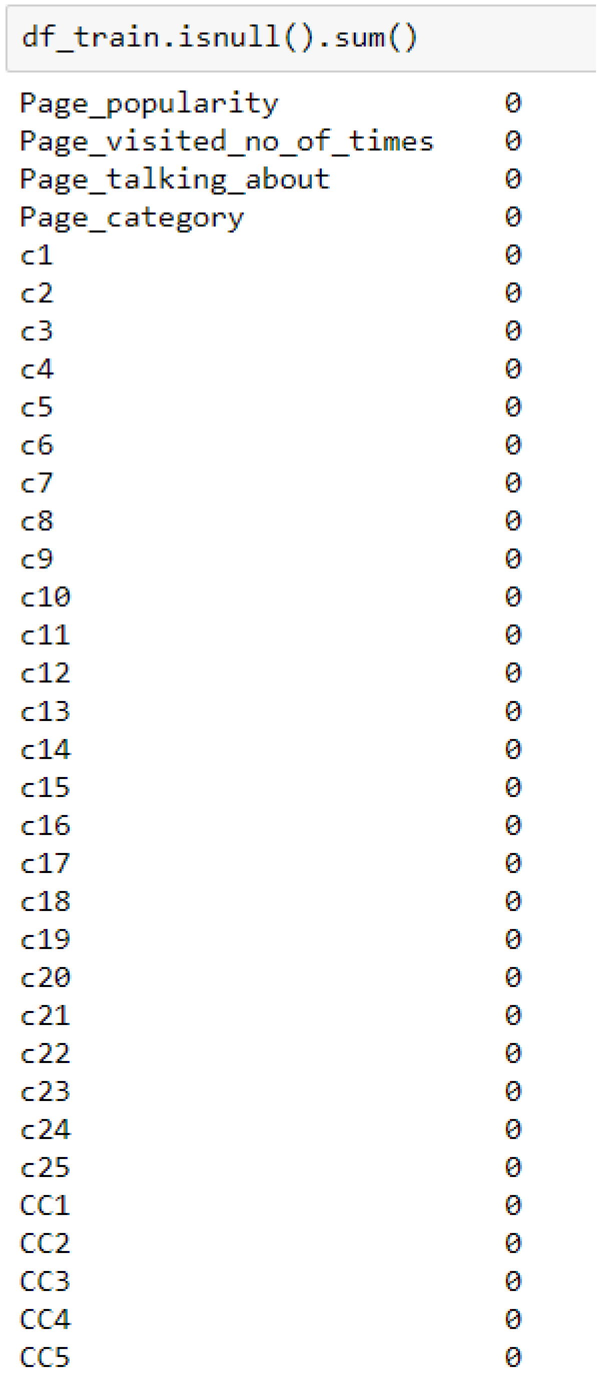
```
df_train = pd.concat(frames)
```

After the merge is done, we can check that the columns have just been added one after another.

```
df_train.shape
```

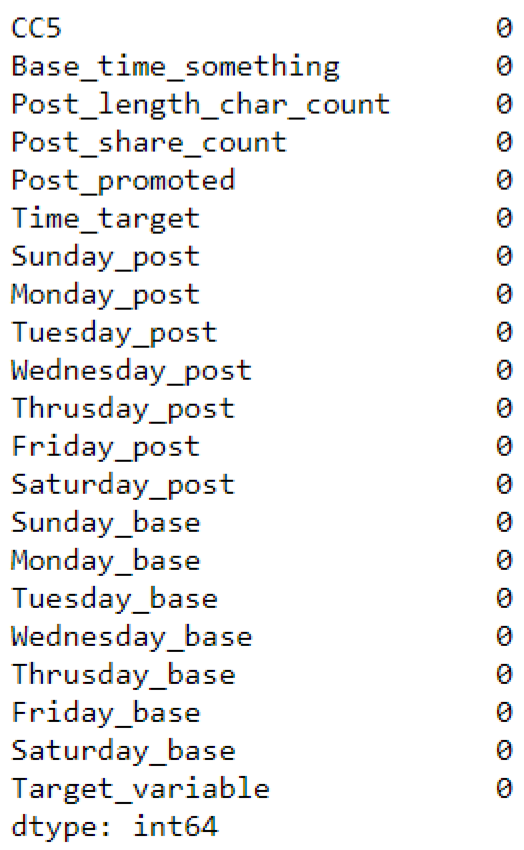
```
(602813, 54)
```

Now we must check whether our dataset contains any '*null*' values or not.



```
df_train.isnull().sum()
```

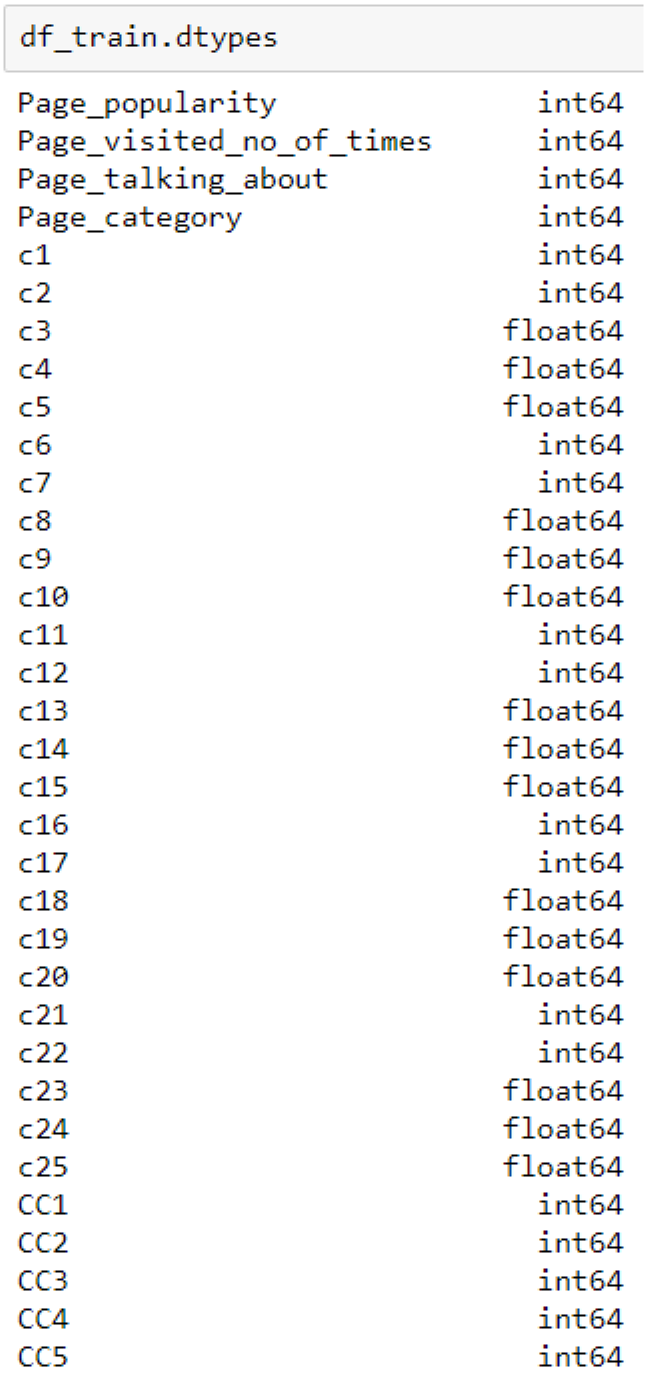
```
Page_popularity            0
Page_visited_no_of_times   0
Page_talking_about         0
Page_category              0
c1                         0
c2                         0
c3                         0
c4                         0
c5                         0
c6                         0
c7                         0
c8                         0
c9                         0
c10                        0
c11                        0
c12                        0
c13                        0
c14                        0
c15                        0
c16                        0
c17                        0
c18                        0
c19                        0
c20                        0
c21                        0
c22                        0
c23                        0
c24                        0
c25                        0
CC1                        0
CC2                        0
CC3                        0
CC4                        0
CC5                        0
```



```
CC5                          0
Base_time_something          0
Post_length_char_count       0
Post_share_count             0
Post_promoted                0
Time_target                  0
Sunday_post                  0
Monday_post                  0
Tuesday_post                 0
Wednesday_post               0
Thrusday_post                0
Friday_post                  0
Saturday_post                0
Sunday_base                  0
Monday_base                  0
Tuesday_base                 0
Wednesday_base               0
Thrusday_base                0
Friday_base                  0
Saturday_base                0
Target_variable              0
dtype: int64
```

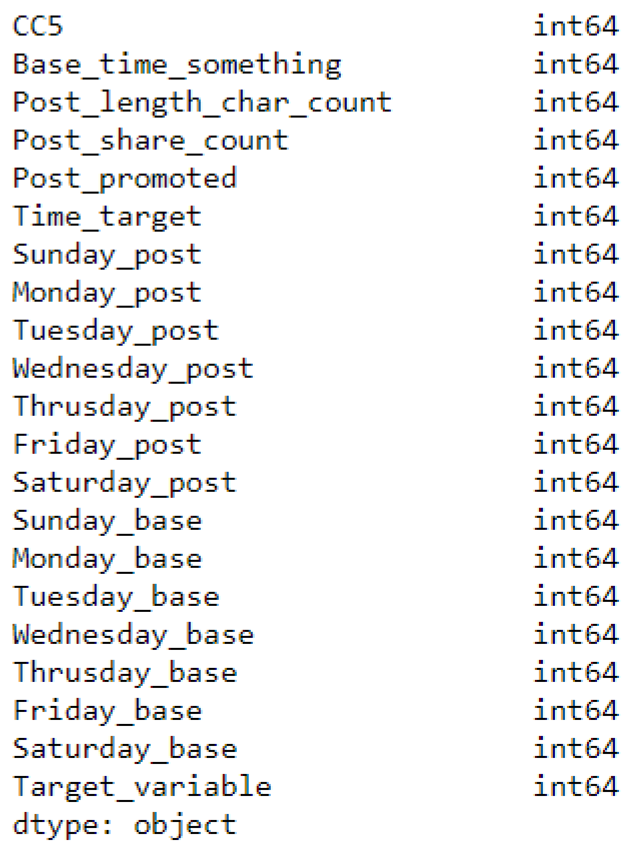
We can see that out dataset is clean and it does not have any null values.

To make calculations and to conduct tests, it is necessary for us to understand the type of data we have in our dataset.



```
df_train.dtypes
```

```
Page_popularity              int64
Page_visited_no_of_times     int64
Page_talking_about           int64
Page_category                int64
c1                           int64
c2                           int64
c3                         float64
c4                         float64
c5                         float64
c6                           int64
c7                           int64
c8                         float64
c9                         float64
c10                        float64
c11                          int64
c12                          int64
c13                        float64
c14                        float64
c15                        float64
c16                          int64
c17                          int64
c18                        float64
c19                        float64
c20                        float64
c21                          int64
c22                          int64
c23                        float64
c24                        float64
c25                        float64
CC1                          int64
CC2                          int64
CC3                          int64
CC4                          int64
CC5                          int64
```

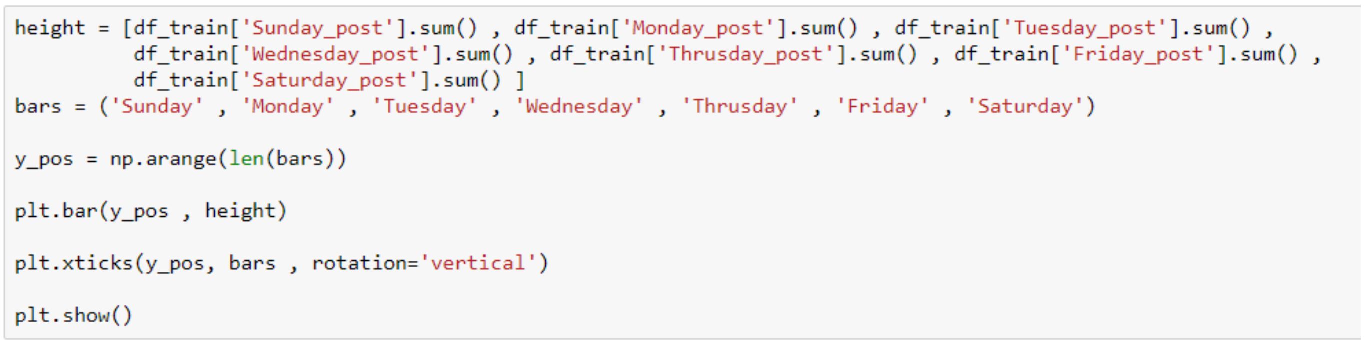


```
CC5                          int64
Base_time_something          int64
Post_length_char_count       int64
Post_share_count             int64
Post_promoted                int64
Time_target                  int64
Sunday_post                  int64
Monday_post                  int64
Tuesday_post                 int64
Wednesday_post               int64
Thrusday_post                int64
Friday_post                  int64
Saturday_post                int64
Sunday_base                  int64
Monday_base                  int64
Tuesday_base                 int64
Wednesday_base               int64
Thrusday_base                int64
Friday_base                  int64
Saturday_base                int64
Target_variable              int64
dtype: object
```

We can see from the above screenshot that our data is only in '*integer*' or '*float*' format. So, this make our mathematical calculations easier.

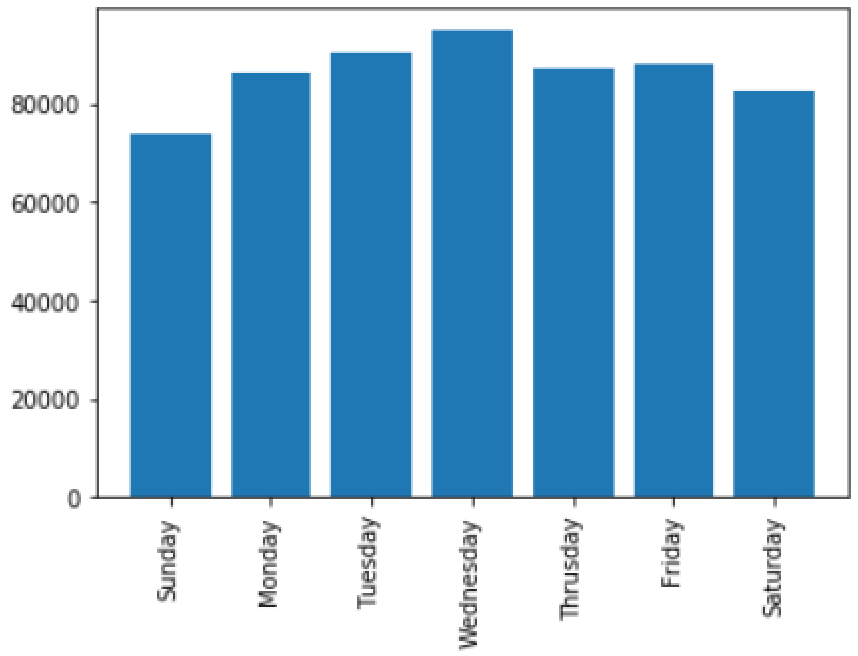
Now, we go through our dataset to understand how the data is distributed.

To understand what day of the week is the post been posted, we plot a graph.



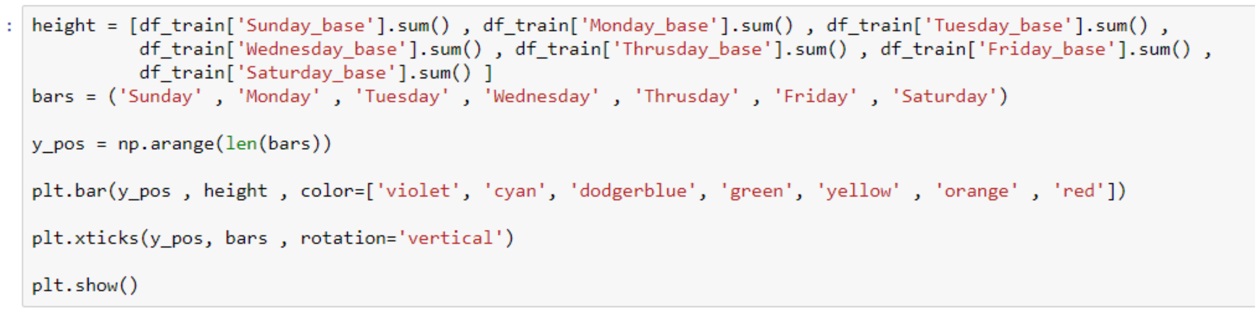
```
height = [df_train['Sunday_post'].sum() , df_train['Monday_post'].sum() , df_train['Tuesday_post'].sum() ,
          df_train['Wednesday_post'].sum() , df_train['Thrusday_post'].sum() , df_train['Friday_post'].sum() ,
          df_train['Saturday_post'].sum() ]
bars = ('Sunday' , 'Monday' , 'Tuesday' , 'Wednesday' , 'Thrusday' , 'Friday' , 'Saturday')

y_pos = np.arange(len(bars))

plt.bar(y_pos , height)

plt.xticks(y_pos, bars , rotation='vertical')

plt.show()
```

If we run the above code, we receive the below graph.



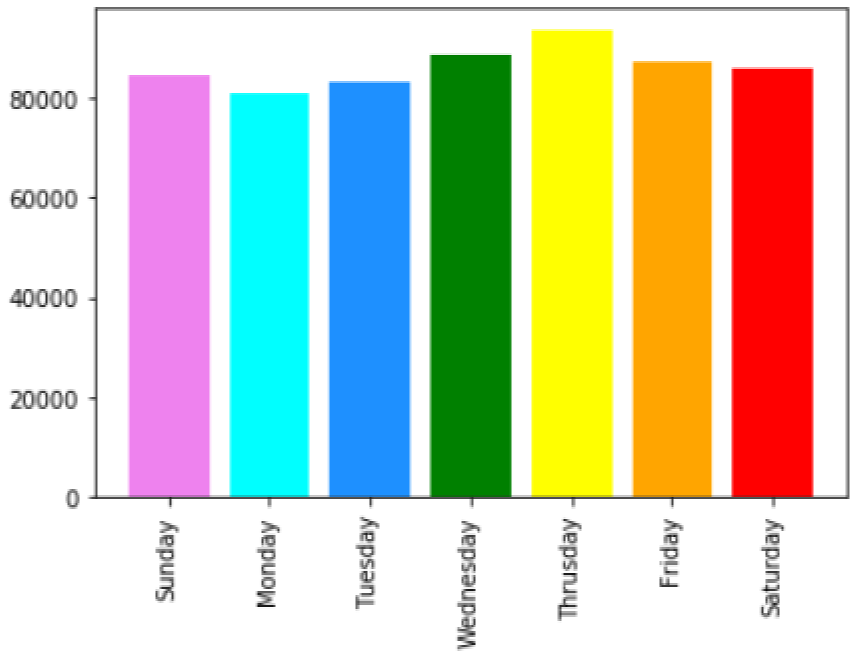
From the above graph, we can understand that frequency of post increases on daily basis and it reaches its maximum point at Wednesday and then it declines gradually.

Now we must understand how the comments are coming for these posts when compared to base time.



```
height = [df_train['Sunday_base'].sum() , df_train['Monday_base'].sum() , df_train['Tuesday_base'].sum() ,
          df_train['Wednesday_base'].sum() , df_train['Thrusday_base'].sum() , df_train['Friday_base'].sum() ,
          df_train['Saturday_base'].sum() ]
bars = ('Sunday' , 'Monday' , 'Tuesday' , 'Wednesday' , 'Thrusday' , 'Friday' , 'Saturday')

y_pos = np.arange(len(bars))

plt.bar(y_pos , height , color=['violet', 'cyan', 'dodgerblue', 'green', 'yellow' , 'orange' , 'red'])

plt.xticks(y_pos, bars , rotation='vertical')

plt.show()
```

When we run the above code, we get the following output.

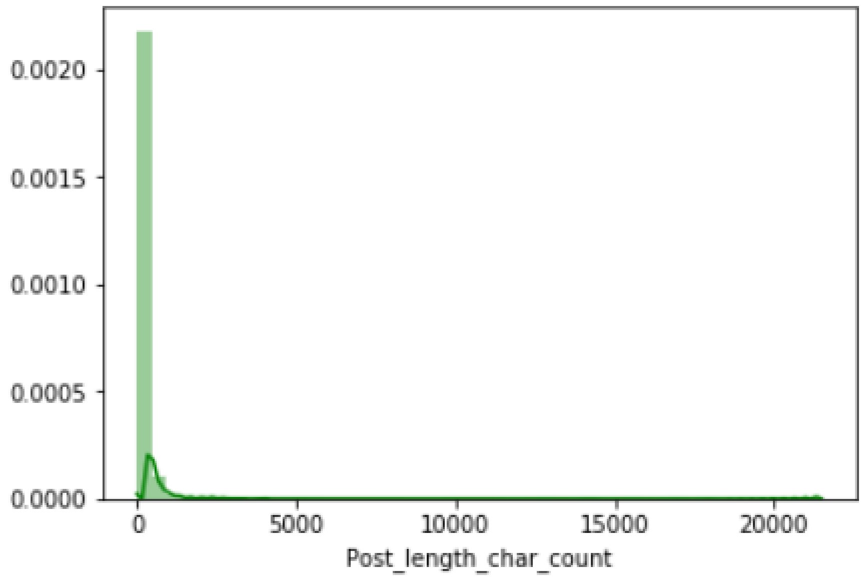


Now, we must understand the characteristics of length of the post.

```
print(df_train['Post_length_char_count'].describe())
sns.distplot(df_train['Post_length_char_count'], color='g',  hist_kws={'alpha': 0.4});
```

When we run the above code, we get the following graph.

```
count     602813.000000
mean         163.653299
std          375.782004
min            0.000000
25%           38.000000
50%           97.000000
75%          172.000000
max        21480.000000
Name: Post_length_char_count, dtype: float64
```



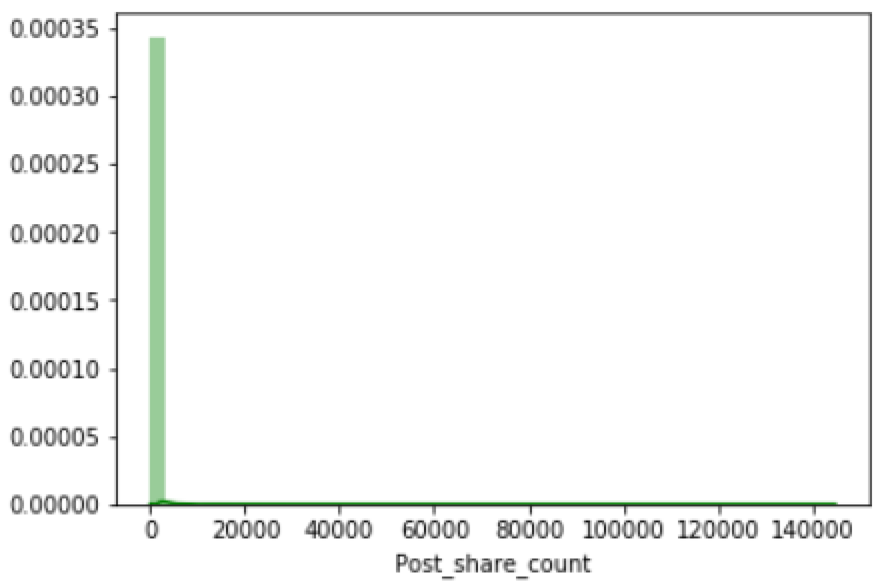
With the count and mean mentioned, we can clearly understand how the data is distributed.

Similarly, we must understand the characteristics of '*Post_share_count*'

```
print(df_train['Post_share_count'].describe())
sns.distplot(df_train['Post_share_count'], color='g', hist_kws={'alpha': 0.4});
```

When we run the above code, we get the following graph as output.

```
count    602813.000000
mean        117.301825
std         951.304620
min           1.000000
25%           2.000000
50%          13.000000
75%          61.000000
max      144860.000000
Name: Post_share_count, dtype: float64
```



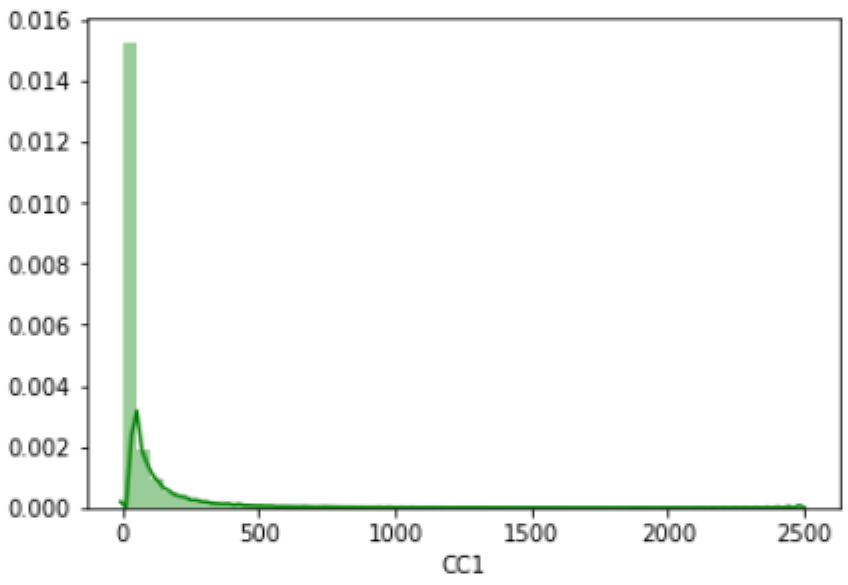
With the count and mean mentioned, we can clearly understand how the data is distributed.

Similarly, we must understand the characteristics of ‘*CC1*’ which represents total number of comments before selected base date/time.

```
print(df_train['CC1'].describe())
sns.distplot(df_train['CC1'], color='g');
```

When we run the above code, we get the following graph as output.

```
count    602813.000000
mean         55.862191
std         137.552591
min           0.000000
25%           2.000000
50%          11.000000
75%          46.000000
max        2495.000000
Name: CC1, dtype: float64
```

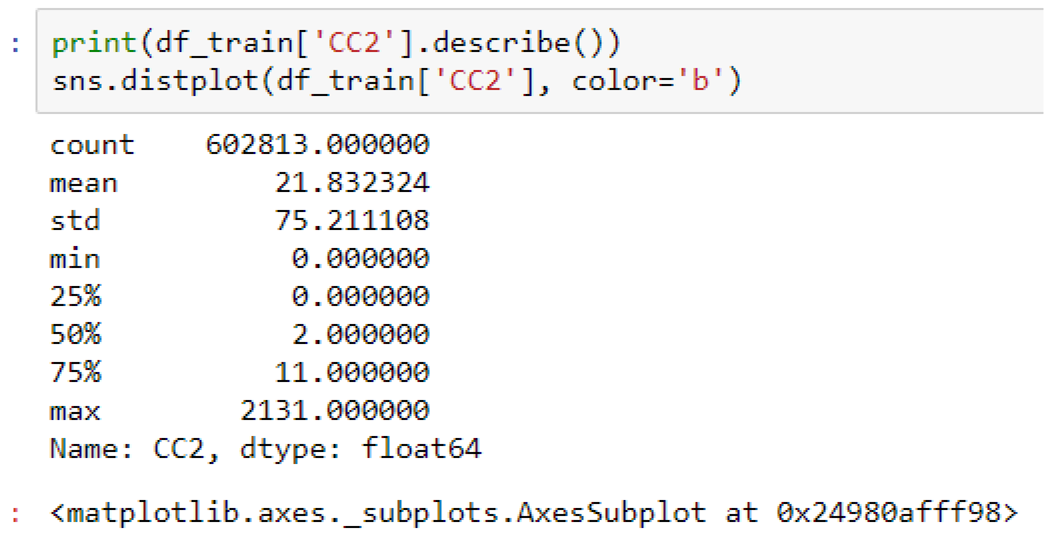


With the count and mean mentioned, we can clearly understand how the data is distributed.

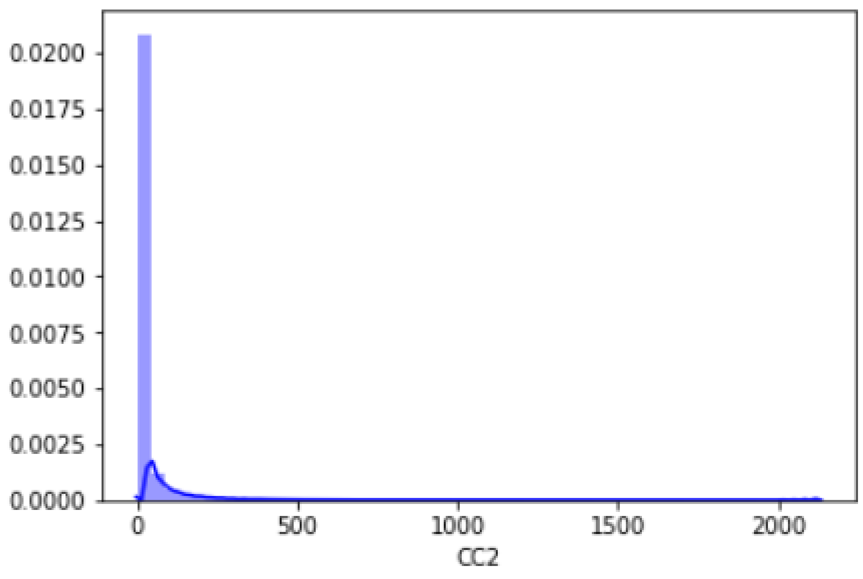
Similarly, we must understand the characteristics of ‘CC2’ which represents number of comments in last 24 hours, relative to base date/time.

```
print(df_train['CC2'].describe())
sns.distplot(df_train['CC2'], color='b')
```

When we run the above code, we get the following graph as output.

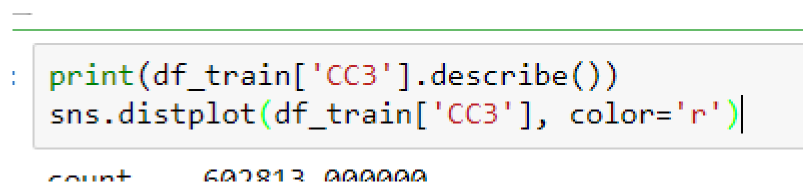


```
: print(df_train['CC2'].describe())
  sns.distplot(df_train['CC2'], color='b')

  count    602813.000000
  mean         21.832324
  std          75.211108
  min           0.000000
  25%           0.000000
  50%           2.000000
  75%          11.000000
  max        2131.000000
  Name: CC2, dtype: float64

: <matplotlib.axes._subplots.AxesSubplot at 0x24980afff98>
```



With the count and mean mentioned, we can clearly understand how the data is distributed.

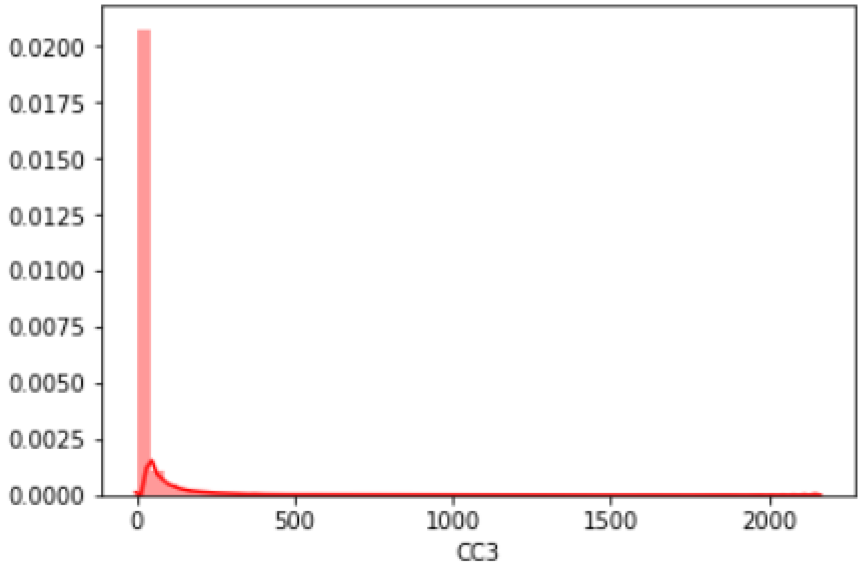
Similarly, we must understand the characteristics of '*CC3*' which represents number of comments in last 48 hours to last 24 hours relative to base date/time.



```
: print(df_train['CC3'].describe())
  sns.distplot(df_train['CC3'], color='r')

  count    602813.000000
```

When we run the above code, we get the following graph as output.

```
count     602813.000000
mean          19.975198
std           73.096146
min            0.000000
25%            0.000000
50%            0.000000
75%            9.000000
max         2162.000000
Name: CC3, dtype: float64

<matplotlib.axes._subplots.AxesSubplot at 0x24980c2f048>
```

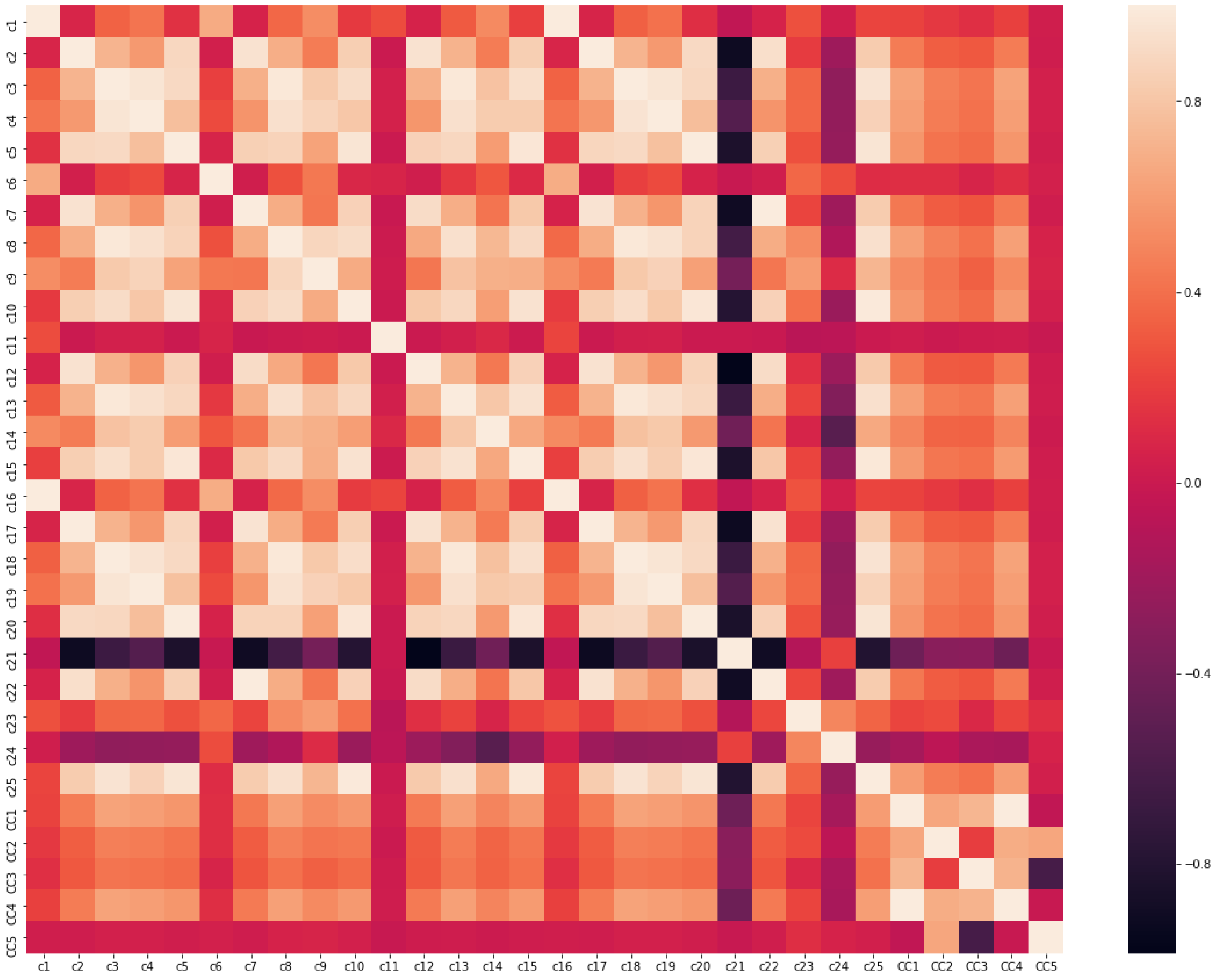


With the count and mean mentioned, we can clearly understand how the data is distributed.

There are 25 other columns in our dataset for which we don't understand the content. In dictionary, generic terms are mentioned as they represent mean, minimum value, maximum value, average, median, standard deviation, etc. So, it's better to understand the correlation between these columns by drawing heat maps.

```
a = df_train[[ 'c1', 'c2', 'c3', 'c4',
       'c5', 'c6', 'c7', 'c8', 'c9', 'c10', 'c11', 'c12', 'c13', 'c14',
       'c15', 'c16', 'c17', 'c18', 'c19', 'c20', 'c21', 'c22', 'c23',
       'c24', 'c25', 'CC1', 'CC2', 'CC3', 'CC4', 'CC5']]

plt.subplots(figsize=(20,15))

df = pd.DataFrame(a)

corr = df.corr()
sns.heatmap(corr,
        xticklabels=df.columns,
        yticklabels=df.columns)

plt.show()
```

If we run the above code, we get the following heat map.

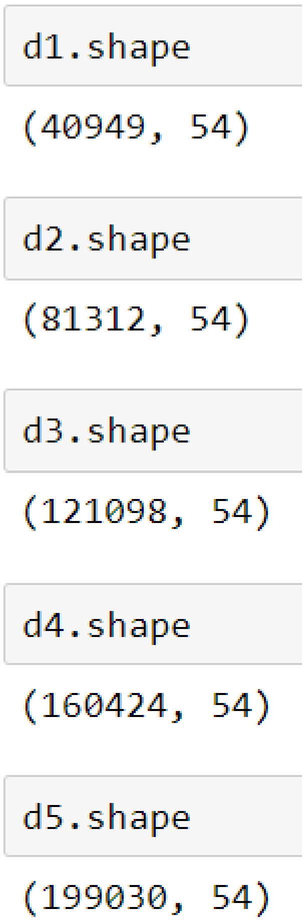


From the above heat map, we can understand that ‘c21’ has least values as compared to all other columns. ‘c24’ also have very low values but slightly higher as compared to ‘c21’. ‘c11’ have highest value as compared to every other column. ‘c6’ also contains higher set of values but not higher than ‘c11’. Data in columns ‘CC1’ , ‘CC2’ , ‘CC3’ and ‘CC4’ are evenly distributed with no column have regular high or low data as compared to other columns.

Feature Engineering

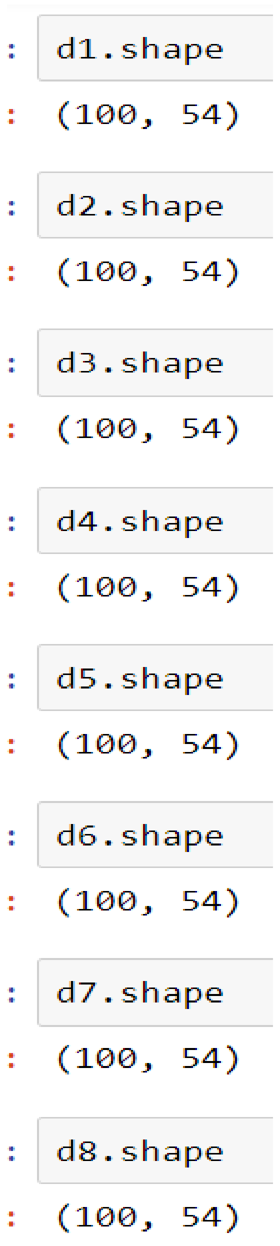
We Have 5 variation of training dataset and 10 variation of testing dataset.

Following is the details of training dataset.

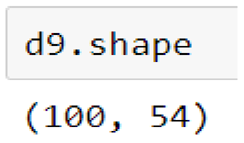


```
d1.shape

(40949, 54)

d2.shape

(81312, 54)

d3.shape

(121098, 54)

d4.shape

(160424, 54)

d5.shape

(199030, 54)
```

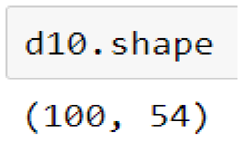
Following are the details of testing dataset.



```
: d1.shape
: (100, 54)

: d2.shape
: (100, 54)

: d3.shape
: (100, 54)

: d4.shape
: (100, 54)

: d5.shape
: (100, 54)

: d6.shape
: (100, 54)

: d7.shape
: (100, 54)

: d8.shape
: (100, 54)
```



```
d9.shape
(100, 54)
```



```
d10.shape
(100, 54)
```

We can see that all 10 variants of dataset have 100 rows.

Our dataset has 54 columns. It is important for us to understand what they represent and what is their importance.

Column No	Column Name	Data Description
1	Page Popularity/Likes	Defines the popularity or support for the source of the document.
2	Page Checking	Describes how many individuals so far visited this place. This feature is only associated with the places e.g.: some institution, place, theater etc.
3	Page Talking About	Defines the daily interest of individuals towards source of the document/ Post. The people who come back to the page, after liking the page. This include activities such as comments, likes to a post, shares, etc. by visitors to the page.
4	Page Category	Defines the category of the source of the document e.g.: place, institution, brand etc.
5-29	Derived	These features are aggregated by page, by calculating min, max, average, median and standard deviation of essential features.
30	CC1	The total number of comments before selected base date/time.
31	CC2	The number of comments in last 24 hours, relative to base date/time.
32	CC3	The number of comments in last 48 to last 24 hours relative to base date/time.
33	CC4	The number of comments in the first 24 hours after the publication of post but before base date/time.
34	CC5	The difference between CC2 and CC3.
35	Base Time	Selected time in order to simulate the scenario.
36	Post Length	Character count in the post
37	Post Share Count	This feature counts the no of shares of the post, that how many peoples had shared this post on to their timeline.
38	Post Promotion Count	To reach more people with posts in News Feed, individual promote their post and this feature tells that whether the post is promoted (1) or not (0).
39	H Local	This describes the H hrs., for which we have the target variable/ comments received.

40-46	Post Published weekday	This represents the day (Sunday...Saturday) on which the post was published. (One hot encoding)
47-53	Base Date Time weekday	This represents the day (Sunday...Saturday) on selected base Date/Time. (One hot encoding)
54	Target Variable	The no of comments in next H hrs. (H is given in Feature no 39).

We are merging the 5-training dataset into one to make the calculations easier and to proceed in a streamline way.

```
frames = [d1 , d2 , d3 , d4 , d5]
```

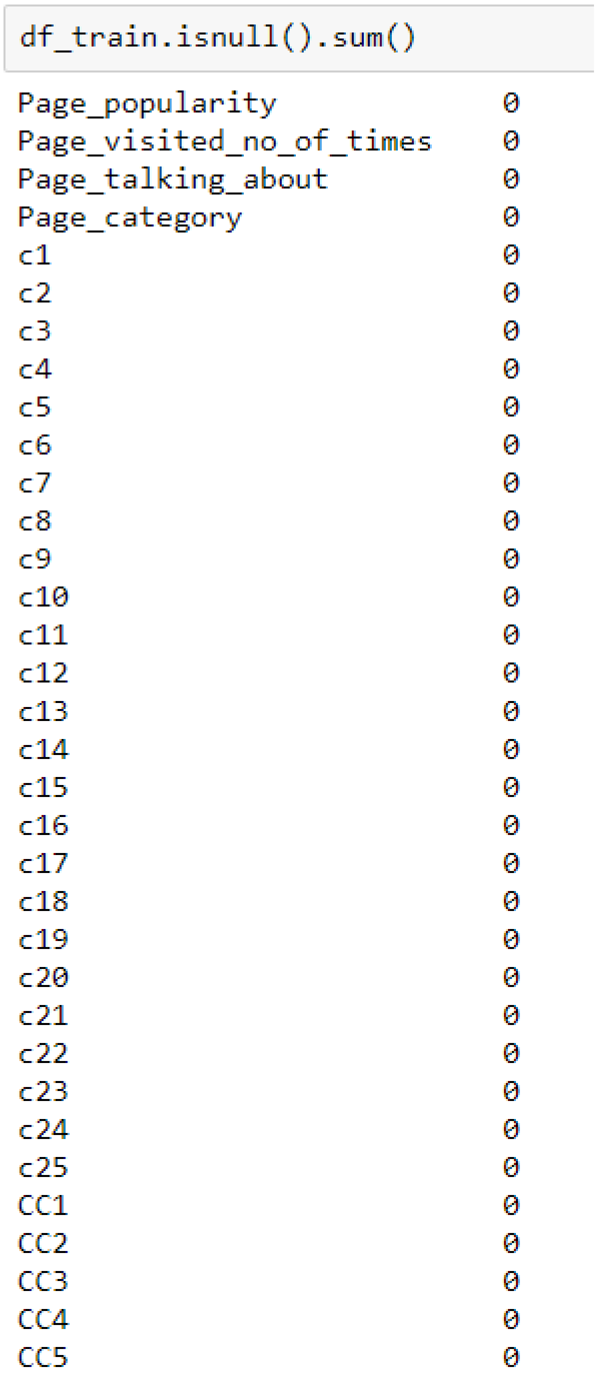
```
df_train = pd.concat(frames)
```

After the merge is done, we can check that the columns have just been added one after another.

```
df_train.shape
```

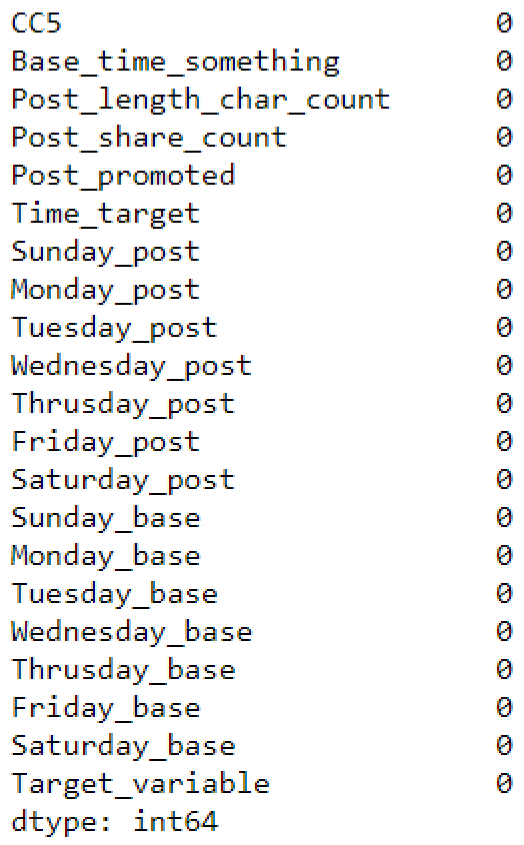
```
(602813, 54)
```

Now we must check whether our dataset contains any ‘*null*’ values or not.



```
df_train.isnull().sum()
```

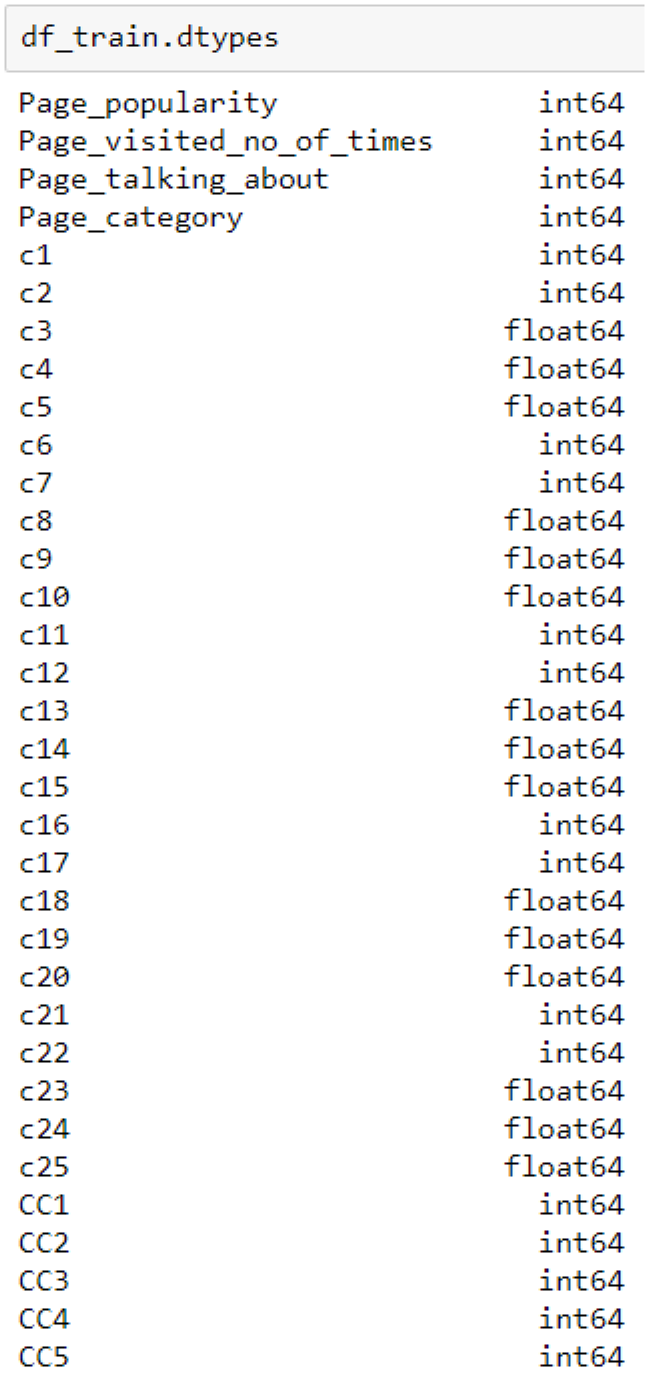
```
Page_popularity            0
Page_visited_no_of_times   0
Page_talking_about         0
Page_category              0
c1                         0
c2                         0
c3                         0
c4                         0
c5                         0
c6                         0
c7                         0
c8                         0
c9                         0
c10                        0
c11                        0
c12                        0
c13                        0
c14                        0
c15                        0
c16                        0
c17                        0
c18                        0
c19                        0
c20                        0
c21                        0
c22                        0
c23                        0
c24                        0
c25                        0
CC1                        0
CC2                        0
CC3                        0
CC4                        0
CC5                        0
```



```
CC5                           0
Base_time_something           0
Post_length_char_count        0
Post_share_count              0
Post_promoted                 0
Time_target                   0
Sunday_post                   0
Monday_post                   0
Tuesday_post                  0
Wednesday_post                0
Thrusday_post                 0
Friday_post                   0
Saturday_post                 0
Sunday_base                   0
Monday_base                   0
Tuesday_base                  0
Wednesday_base                0
Thrusday_base                 0
Friday_base                   0
Saturday_base                 0
Target_variable               0
dtype: int64
```

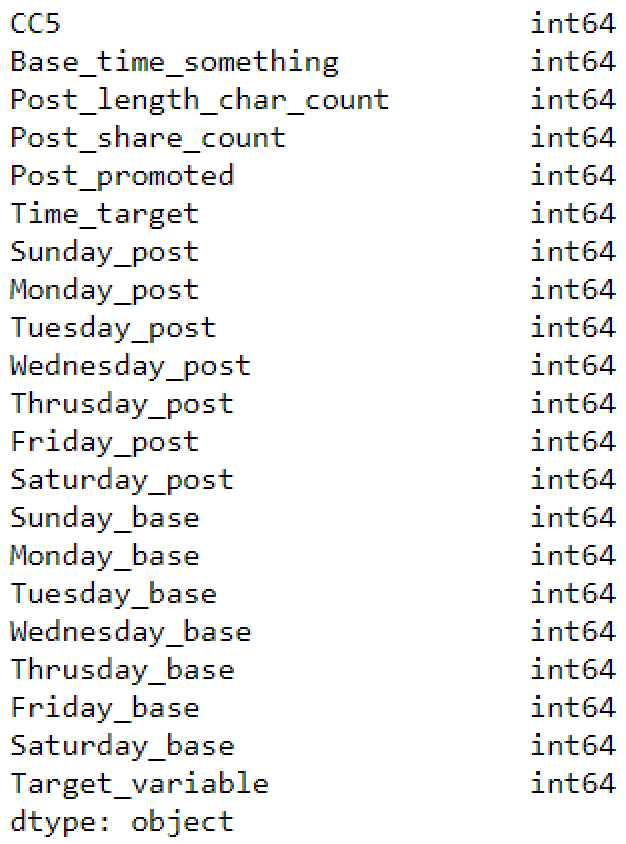
We can see that out dataset is clean and it does not have any null values.

To make calculations and to conduct tests, it is necessary for us to understand the type of data we have in our dataset.



```
df_train.dtypes
```

```
Page_popularity                int64
Page_visited_no_of_times       int64
Page_talking_about             int64
Page_category                  int64
c1                             int64
c2                             int64
c3                           float64
c4                           float64
c5                           float64
c6                             int64
c7                             int64
c8                           float64
c9                           float64
c10                          float64
c11                            int64
c12                            int64
c13                          float64
c14                          float64
c15                          float64
c16                            int64
c17                            int64
c18                          float64
c19                          float64
c20                          float64
c21                            int64
c22                            int64
c23                          float64
c24                          float64
c25                          float64
CC1                            int64
CC2                            int64
CC3                            int64
CC4                            int64
CC5                            int64
```



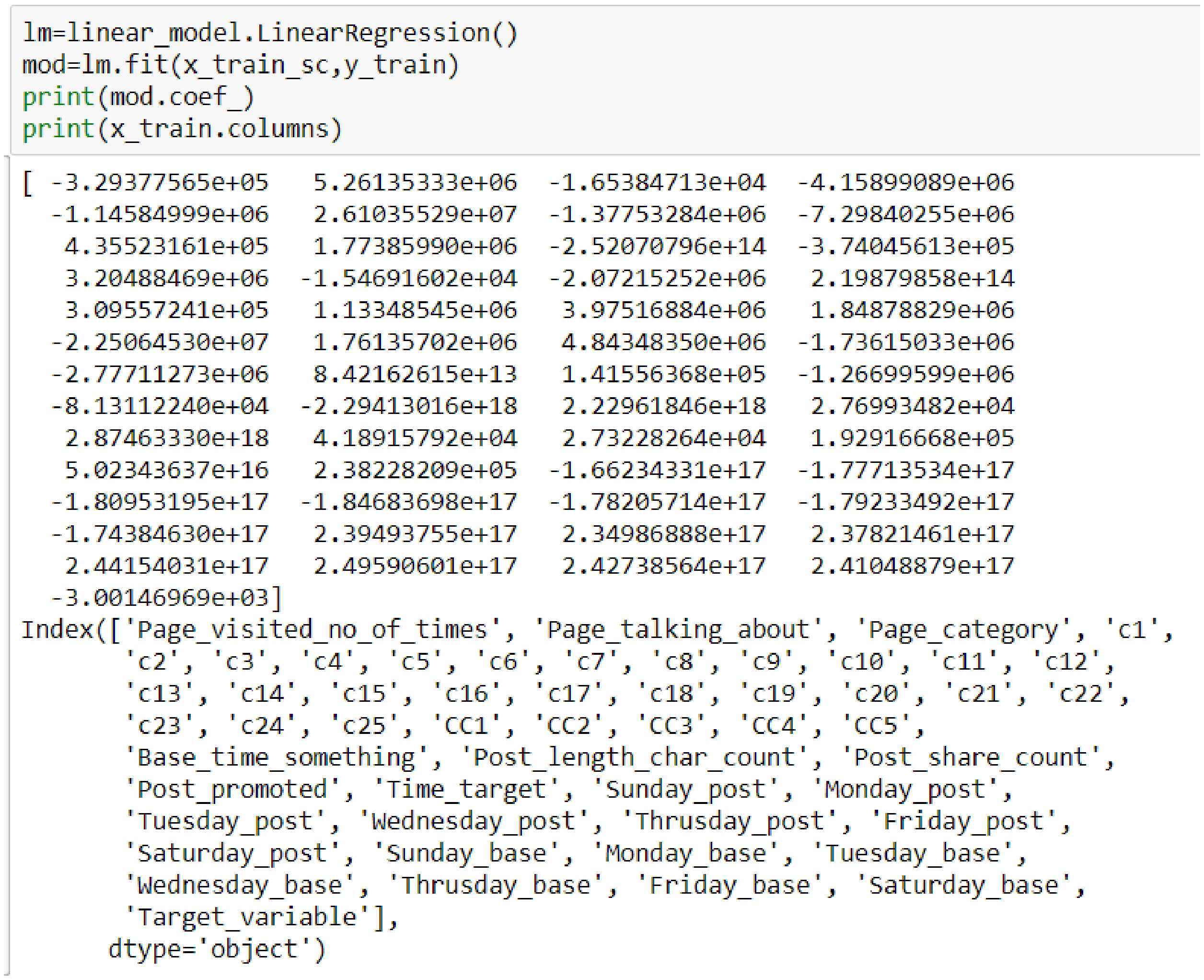
```
CC5                          int64
Base_time_something          int64
Post_length_char_count       int64
Post_share_count             int64
Post_promoted                int64
Time_target                  int64
Sunday_post                  int64
Monday_post                  int64
Tuesday_post                 int64
Wednesday_post               int64
Thrusday_post                int64
Friday_post                  int64
Saturday_post                int64
Sunday_base                  int64
Monday_base                  int64
Tuesday_base                 int64
Wednesday_base               int64
Thrusday_base                int64
Friday_base                  int64
Saturday_base                int64
Target_variable              int64
dtype: object
```

We can see from the above screenshot that our data is only in ‘*integer*’ or ‘*float*’ format. So, this make our mathematical calculations easier.

Now, we have to understand what different features can be used while using different algorithms as their models differ and we have to provide perfect features for every algorithm for us to get the best output.

Linear Regression

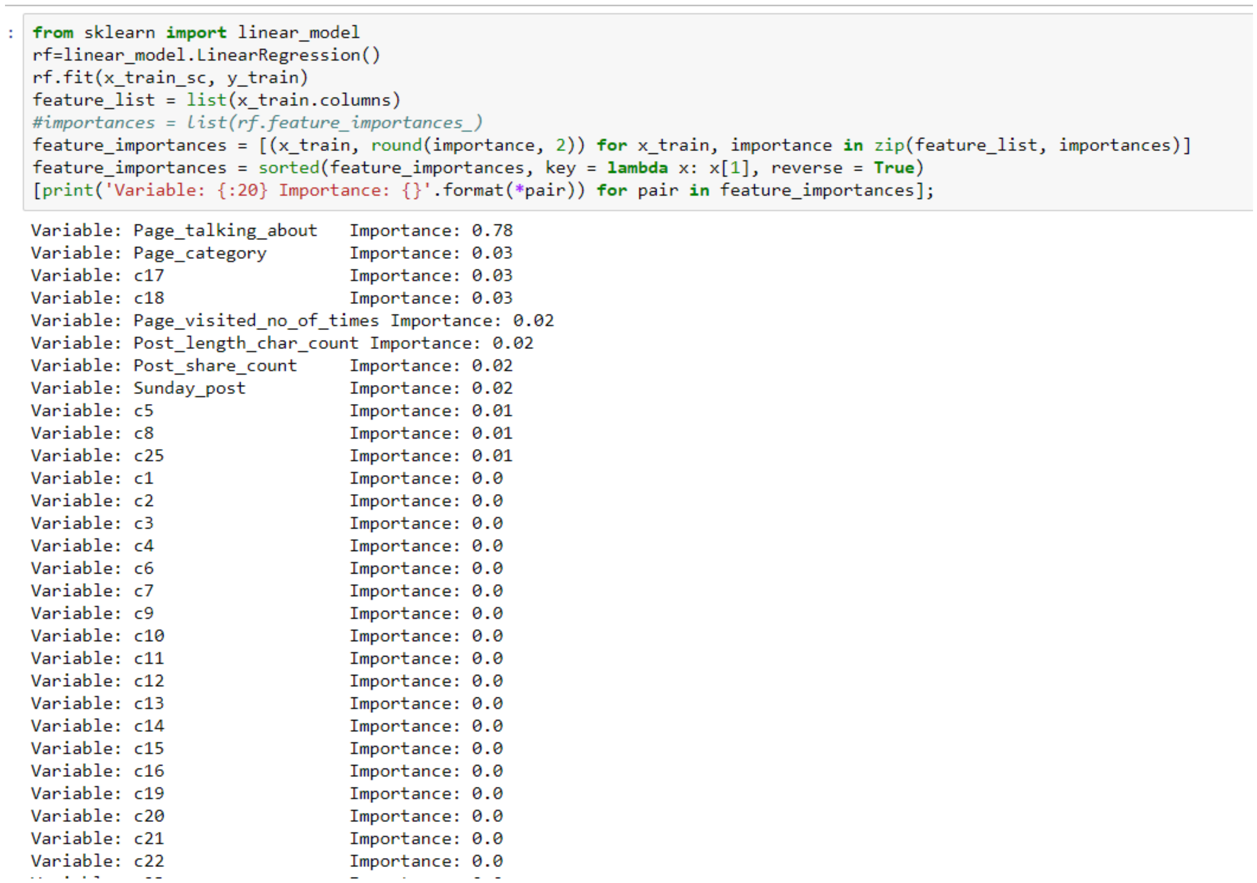
When we run the model for linear regression, we will get the notable features too.



```
lm=linear_model.LinearRegression()
mod=lm.fit(x_train_sc,y_train)
print(mod.coef_)
print(x_train.columns)
```

```
[ -3.29377565e+05   5.26135333e+06  -1.65384713e+04  -4.15899089e+06
  -1.14584999e+06   2.61035529e+07  -1.37753284e+06  -7.29840255e+06
   4.35523161e+05   1.77385990e+06  -2.52070796e+14  -3.74045613e+05
   3.20488469e+06  -1.54691602e+04  -2.07215252e+06   2.19879858e+14
   3.09557241e+05   1.13348545e+06   3.97516884e+06   1.84878829e+06
  -2.25064530e+07   1.76135702e+06   4.84348350e+06  -1.73615033e+06
  -2.77711273e+06   8.42162615e+13   1.41556368e+05  -1.26699599e+06
  -8.13112240e+04  -2.29413016e+18   2.22961846e+18   2.76993482e+04
   2.87463330e+18   4.18915792e+04   2.73228264e+04   1.92916668e+05
   5.02343637e+16   2.38228209e+05  -1.66234331e+17  -1.77713534e+17
  -1.80953195e+17  -1.84683698e+17  -1.78205714e+17  -1.79233492e+17
  -1.74384630e+17   2.39493755e+17   2.34986888e+17   2.37821461e+17
   2.44154031e+17   2.49590601e+17   2.42738564e+17   2.41048879e+17
  -3.00146969e+03]
Index(['Page_visited_no_of_times', 'Page_talking_about', 'Page_category', 'c1',
       'c2', 'c3', 'c4', 'c5', 'c6', 'c7', 'c8', 'c9', 'c10', 'c11', 'c12',
       'c13', 'c14', 'c15', 'c16', 'c17', 'c18', 'c19', 'c20', 'c21', 'c22',
       'c23', 'c24', 'c25', 'CC1', 'CC2', 'CC3', 'CC4', 'CC5',
       'Base_time_something', 'Post_length_char_count', 'Post_share_count',
       'Post_promoted', 'Time_target', 'Sunday_post', 'Monday_post',
       'Tuesday_post', 'Wednesday_post', 'Thrusday_post', 'Friday_post',
       'Saturday_post', 'Sunday_base', 'Monday_base', 'Tuesday_base',
       'Wednesday_base', 'Thrusday_base', 'Friday_base', 'Saturday_base',
       'Target_variable'],
      dtype='object')
```

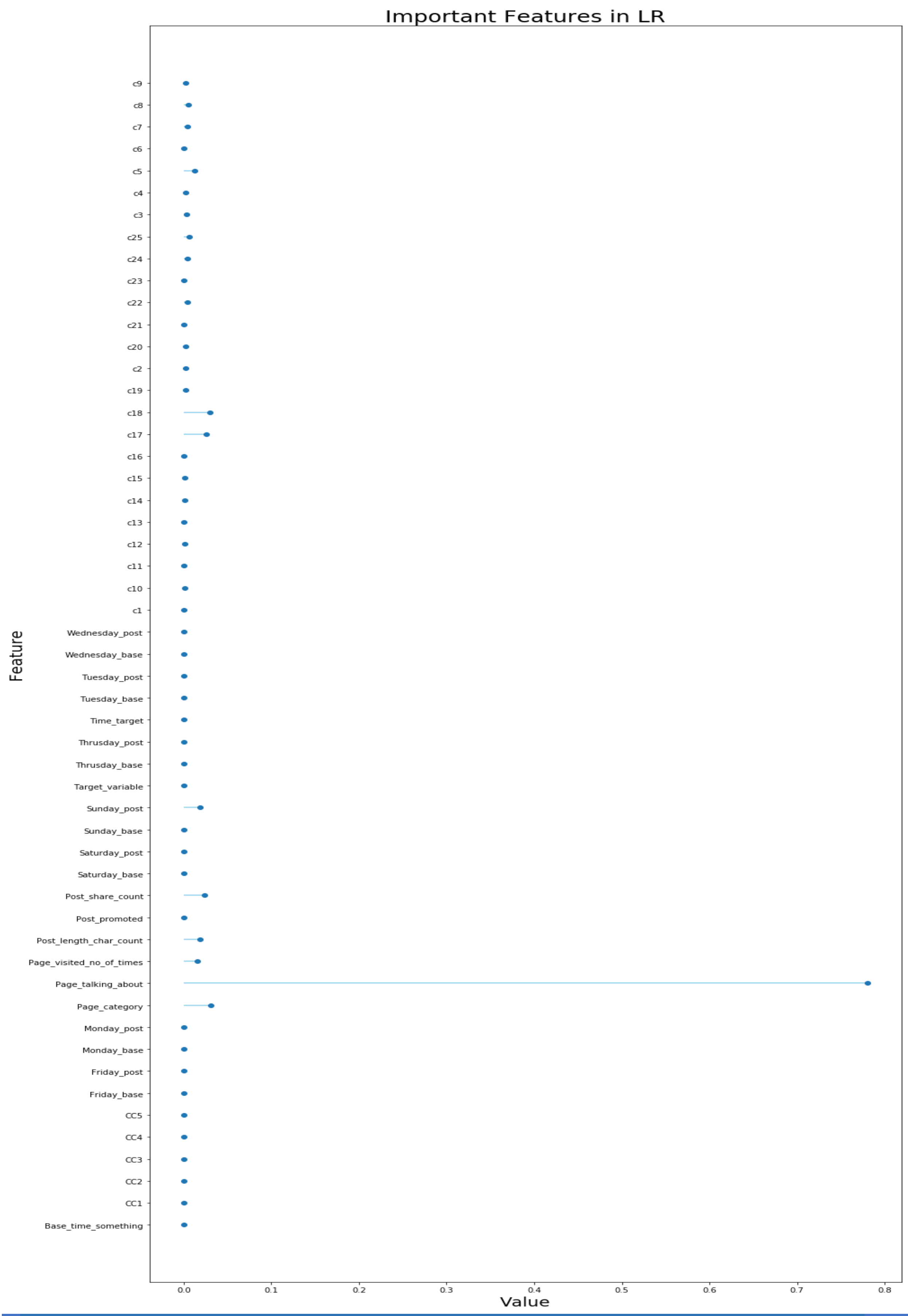
Now when we have important features, we also have to understand the weightage features.



```
from sklearn import linear_model
rf=linear_model.LinearRegression()
rf.fit(x_train_sc, y_train)
feature_list = list(x_train.columns)
#importances = list(rf.feature_importances_)
feature_importances = [(x_train, round(importance, 2)) for x_train, importance in zip(feature_list, importances)]
feature_importances = sorted(feature_importances, key = lambda x: x[1], reverse = True)
[print('Variable: {:20} Importance: {}'.format(*pair)) for pair in feature_importances];
```

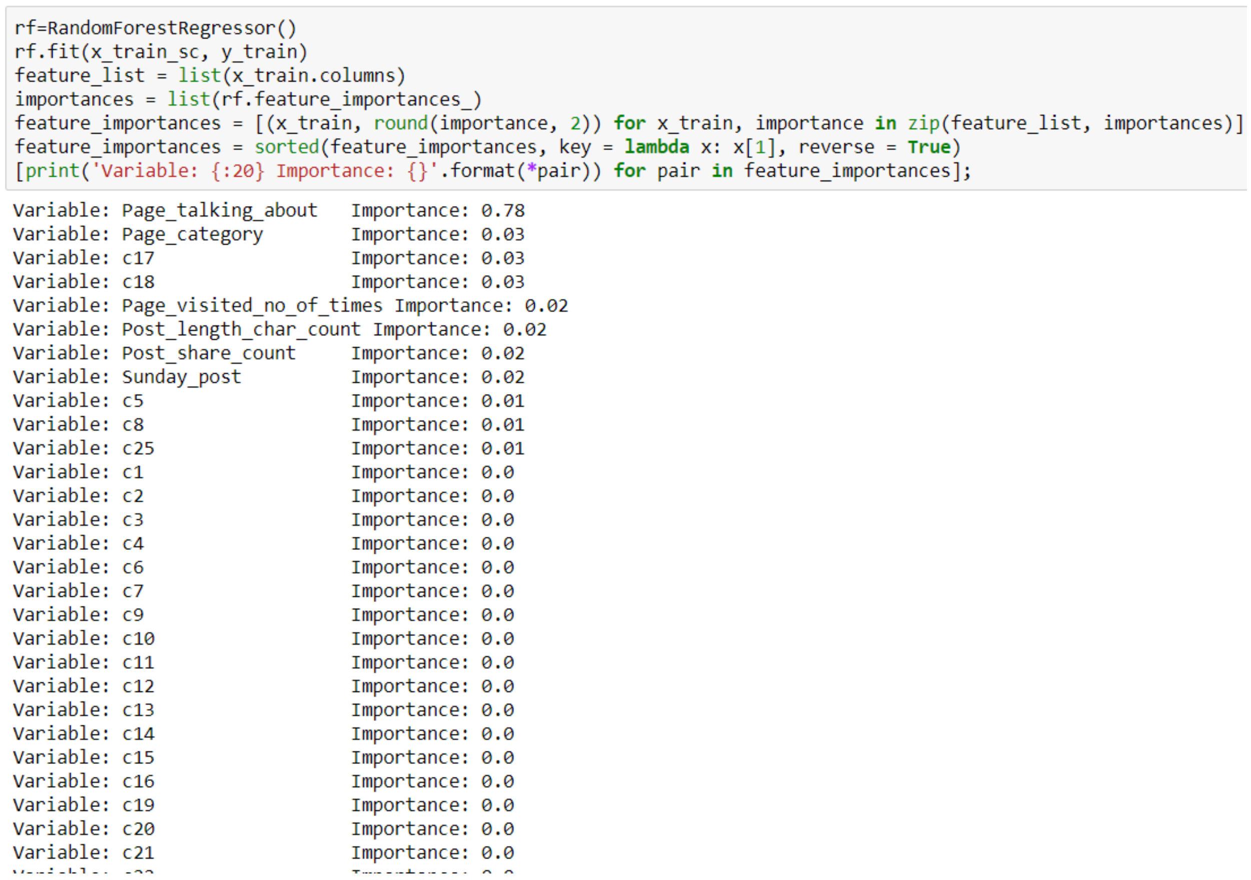
```
Variable: Page_talking_about   Importance: 0.78
Variable: Page_category        Importance: 0.03
Variable: c17                  Importance: 0.03
Variable: c18                  Importance: 0.03
Variable: Page_visited_no_of_times Importance: 0.02
Variable: Post_length_char_count Importance: 0.02
Variable: Post_share_count     Importance: 0.02
Variable: Sunday_post          Importance: 0.02
Variable: c5                   Importance: 0.01
Variable: c8                   Importance: 0.01
Variable: c25                  Importance: 0.01
Variable: c1                   Importance: 0.0
Variable: c2                   Importance: 0.0
Variable: c3                   Importance: 0.0
Variable: c4                   Importance: 0.0
Variable: c6                   Importance: 0.0
Variable: c7                   Importance: 0.0
Variable: c9                   Importance: 0.0
Variable: c10                  Importance: 0.0
Variable: c11                  Importance: 0.0
Variable: c12                  Importance: 0.0
Variable: c13                  Importance: 0.0
Variable: c14                  Importance: 0.0
Variable: c15                  Importance: 0.0
Variable: c16                  Importance: 0.0
Variable: c19                  Importance: 0.0
Variable: c20                  Importance: 0.0
Variable: c21                  Importance: 0.0
Variable: c22                  Importance: 0.0
```

Graphical representation of the same is as follows.


Important Features in LR
Feature
Value
c9
c8
c7
c6
c5
c4
c3
c25
c24
c23
c22
c21
c20
c2
c19
c18
c17
c16
c15
c14
c13
c12
c11
c10
c1
Wednesday_post
Wednesday_base
Tuesday_post
Tuesday_base
Time_target
Thrusday_post
Thrusday_base
Target_variable
Sunday_post
Sunday_base
Saturday_post
Saturday_base
Post_share_count
Post_promoted
Post_length_char_count
Page_visited_no_of_times
Page_talking_about
Page_category
Monday_post
Monday_base
Friday_post
Friday_base
CC5
CC4
CC3
CC2
CC1
Base_time_something
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8

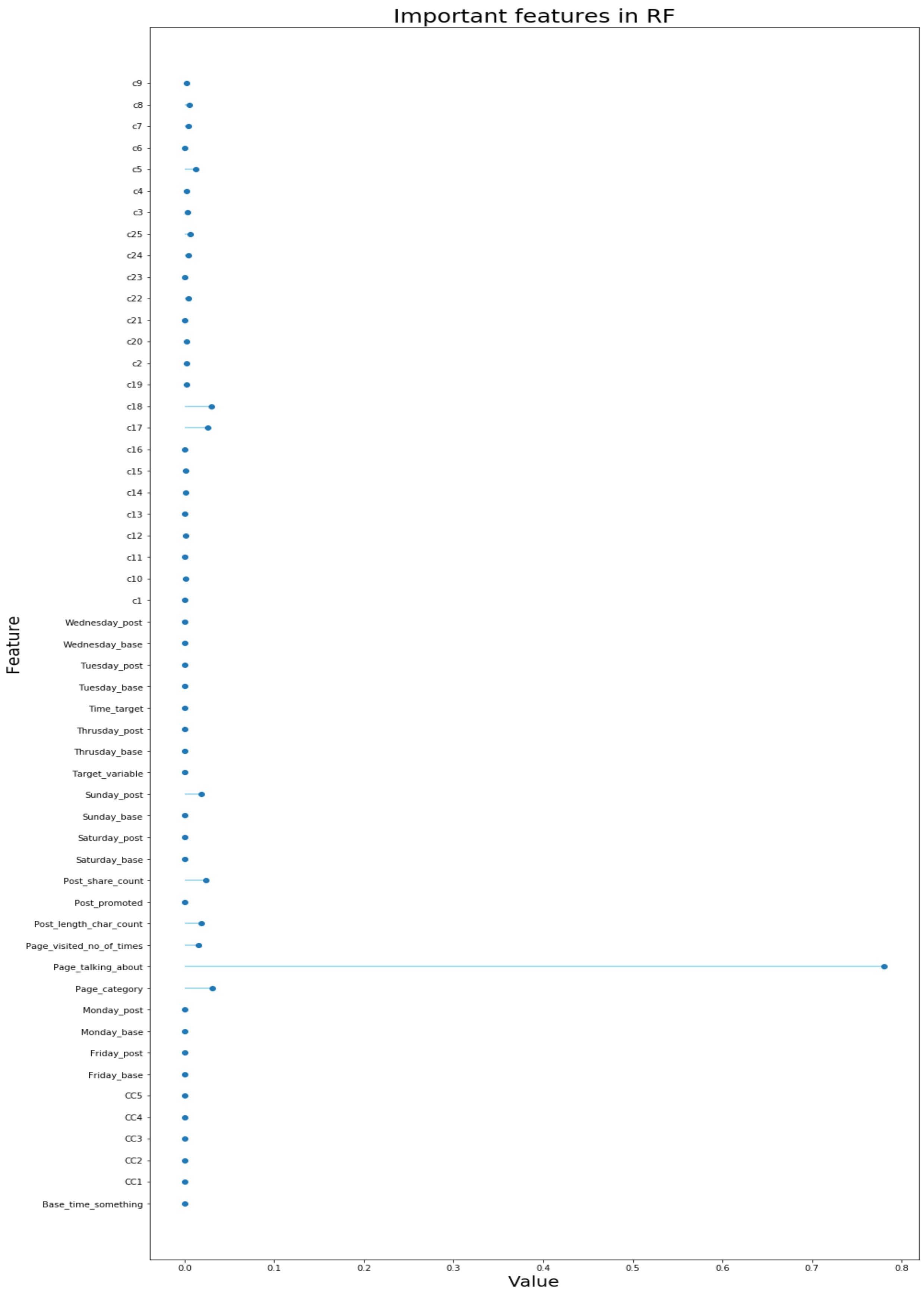
Random Forest

We have to find important features, we also have to understand the weightage features.



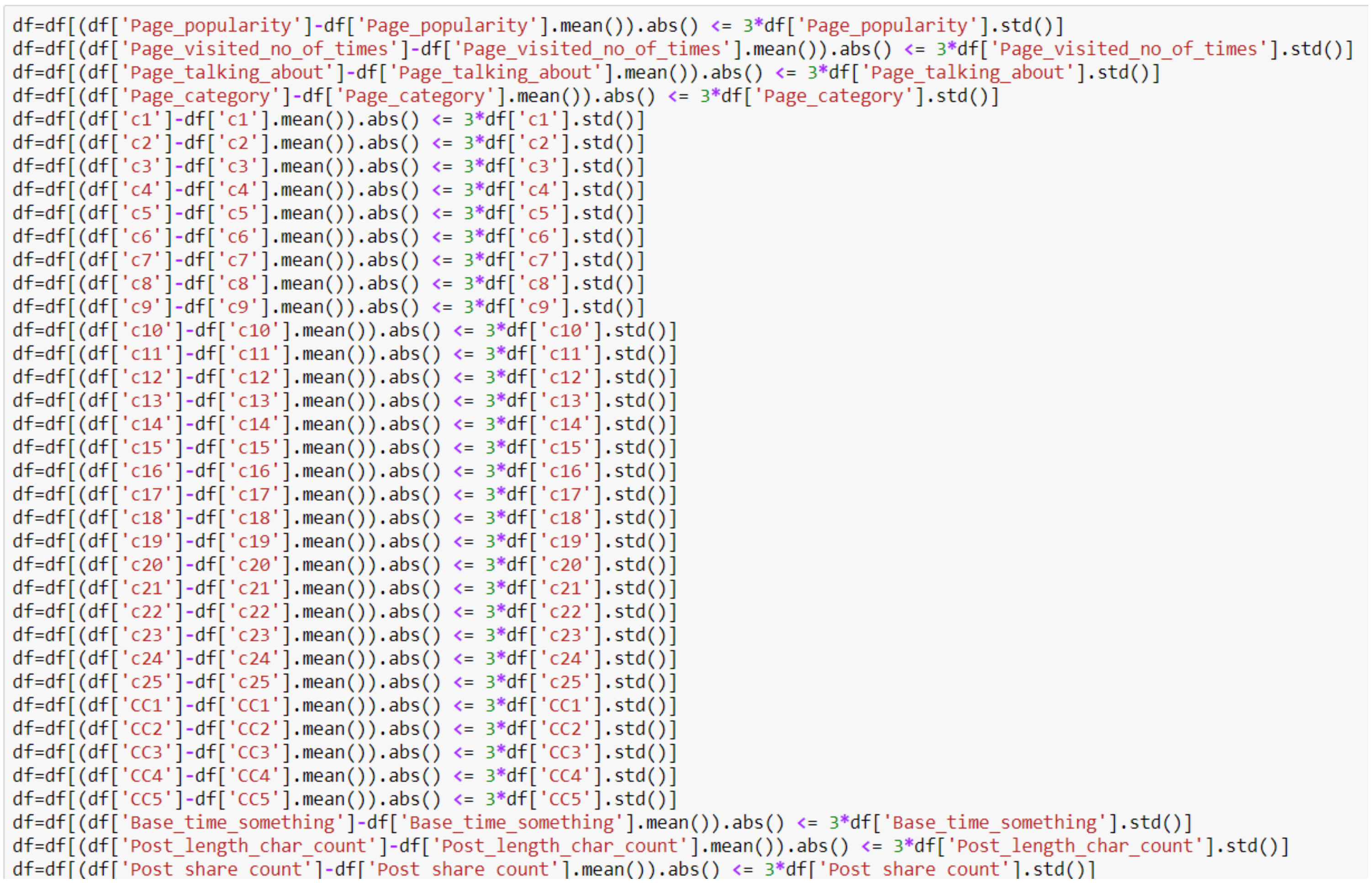
```
rf=RandomForestRegressor()
rf.fit(x_train_sc, y_train)
feature_list = list(x_train.columns)
importances = list(rf.feature_importances_)
feature_importances = [(x_train, round(importance, 2)) for x_train, importance in zip(feature_list, importances)]
feature_importances = sorted(feature_importances, key = lambda x: x[1], reverse = True)
[print('Variable: {:20} Importance: {}'.format(*pair)) for pair in feature_importances];
```

```
Variable: Page_talking_about   Importance: 0.78
Variable: Page_category        Importance: 0.03
Variable: c17                  Importance: 0.03
Variable: c18                  Importance: 0.03
Variable: Page_visited_no_of_times Importance: 0.02
Variable: Post_length_char_count Importance: 0.02
Variable: Post_share_count     Importance: 0.02
Variable: Sunday_post          Importance: 0.02
Variable: c5                   Importance: 0.01
Variable: c8                   Importance: 0.01
Variable: c25                  Importance: 0.01
Variable: c1                   Importance: 0.0
Variable: c2                   Importance: 0.0
Variable: c3                   Importance: 0.0
Variable: c4                   Importance: 0.0
Variable: c6                   Importance: 0.0
Variable: c7                   Importance: 0.0
Variable: c9                   Importance: 0.0
Variable: c10                  Importance: 0.0
Variable: c11                  Importance: 0.0
Variable: c12                  Importance: 0.0
Variable: c13                  Importance: 0.0
Variable: c14                  Importance: 0.0
Variable: c15                  Importance: 0.0
Variable: c16                  Importance: 0.0
Variable: c19                  Importance: 0.0
Variable: c20                  Importance: 0.0
Variable: c21                  Importance: 0.0
```


Important features in RF
Feature
Value
c9
c8
c7
c6
c5
c4
c3
c25
c24
c23
c22
c21
c20
c2
c19
c18
c17
c16
c15
c14
c13
c12
c11
c10
c1
Wednesday_post
Wednesday_base
Tuesday_post
Tuesday_base
Time_target
Thrusday_post
Thrusday_base
Target_variable
Sunday_post
Sunday_base
Saturday_post
Saturday_base
Post_share_count
Post_promoted
Post_length_char_count
Page_visited_no_of_times
Page_talking_about
Page_category
Monday_post
Monday_base
Friday_post
Friday_base
CC5
CC4
CC3
CC2
CC1
Base_time_something
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8

Algorithm

Before working with models and algorithms, it is important for us to make the model perfectly. There are lots of values in our dataset. And if we consider everything, we might not get perfect result and to eliminate this problem we are removing outliers.



```
df=df[(df['Page_popularity']-df['Page_popularity'].mean()).abs() <= 3*df['Page_popularity'].std()]
df=df[(df['Page_visited_no_of_times']-df['Page_visited_no_of_times'].mean()).abs() <= 3*df['Page_visited_no_of_times'].std()]
df=df[(df['Page_talking_about']-df['Page_talking_about'].mean()).abs() <= 3*df['Page_talking_about'].std()]
df=df[(df['Page_category']-df['Page_category'].mean()).abs() <= 3*df['Page_category'].std()]
df=df[(df['c1']-df['c1'].mean()).abs() <= 3*df['c1'].std()]
df=df[(df['c2']-df['c2'].mean()).abs() <= 3*df['c2'].std()]
df=df[(df['c3']-df['c3'].mean()).abs() <= 3*df['c3'].std()]
df=df[(df['c4']-df['c4'].mean()).abs() <= 3*df['c4'].std()]
df=df[(df['c5']-df['c5'].mean()).abs() <= 3*df['c5'].std()]
df=df[(df['c6']-df['c6'].mean()).abs() <= 3*df['c6'].std()]
df=df[(df['c7']-df['c7'].mean()).abs() <= 3*df['c7'].std()]
df=df[(df['c8']-df['c8'].mean()).abs() <= 3*df['c8'].std()]
df=df[(df['c9']-df['c9'].mean()).abs() <= 3*df['c9'].std()]
df=df[(df['c10']-df['c10'].mean()).abs() <= 3*df['c10'].std()]
df=df[(df['c11']-df['c11'].mean()).abs() <= 3*df['c11'].std()]
df=df[(df['c12']-df['c12'].mean()).abs() <= 3*df['c12'].std()]
df=df[(df['c13']-df['c13'].mean()).abs() <= 3*df['c13'].std()]
df=df[(df['c14']-df['c14'].mean()).abs() <= 3*df['c14'].std()]
df=df[(df['c15']-df['c15'].mean()).abs() <= 3*df['c15'].std()]
df=df[(df['c16']-df['c16'].mean()).abs() <= 3*df['c16'].std()]
df=df[(df['c17']-df['c17'].mean()).abs() <= 3*df['c17'].std()]
df=df[(df['c18']-df['c18'].mean()).abs() <= 3*df['c18'].std()]
df=df[(df['c19']-df['c19'].mean()).abs() <= 3*df['c19'].std()]
df=df[(df['c20']-df['c20'].mean()).abs() <= 3*df['c20'].std()]
df=df[(df['c21']-df['c21'].mean()).abs() <= 3*df['c21'].std()]
df=df[(df['c22']-df['c22'].mean()).abs() <= 3*df['c22'].std()]
df=df[(df['c23']-df['c23'].mean()).abs() <= 3*df['c23'].std()]
df=df[(df['c24']-df['c24'].mean()).abs() <= 3*df['c24'].std()]
df=df[(df['c25']-df['c25'].mean()).abs() <= 3*df['c25'].std()]
df=df[(df['CC1']-df['CC1'].mean()).abs() <= 3*df['CC1'].std()]
df=df[(df['CC2']-df['CC2'].mean()).abs() <= 3*df['CC2'].std()]
df=df[(df['CC3']-df['CC3'].mean()).abs() <= 3*df['CC3'].std()]
df=df[(df['CC4']-df['CC4'].mean()).abs() <= 3*df['CC4'].std()]
df=df[(df['CC5']-df['CC5'].mean()).abs() <= 3*df['CC5'].std()]
df=df[(df['Base_time_something']-df['Base_time_something'].mean()).abs() <= 3*df['Base_time_something'].std()]
df=df[(df['Post_length_char_count']-df['Post_length_char_count'].mean()).abs() <= 3*df['Post_length_char_count'].std()]
df=df[(df['Post_share_count']-df['Post_share_count'].mean()).abs() <= 3*df['Post_share_count'].std()]
```

In the above screenshot, we can understand that we are eliminating outlier between where mean is below 25% and above 75%

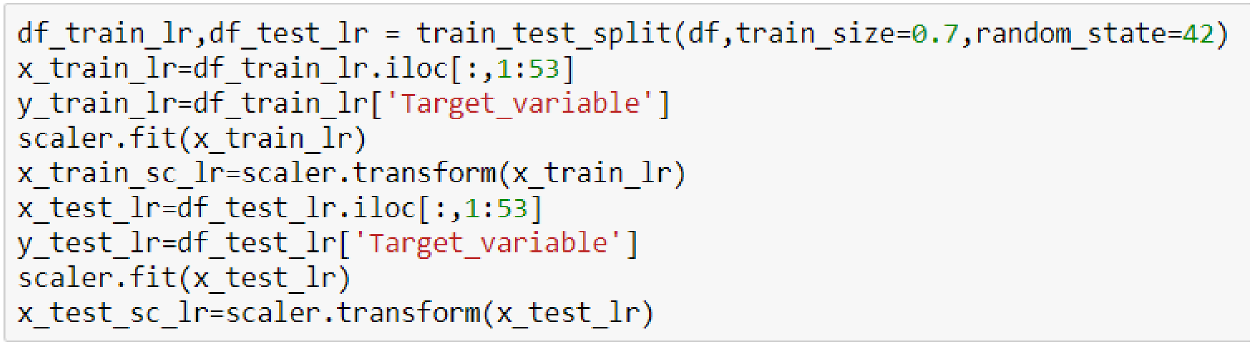
Linear Regression

In statistics, linear regression is a linear approach for modelling the relationship between a scalar dependent variable y and one or more explanatory variables (or independent variables) denoted X. The case of one explanatory variable is called simple linear regression.

In linear regression, the relationships are modeled using linear predictor functions whose unknown model parameters are estimated from the data. Such models are called linear models. Most commonly, the conditional mean of y given the value of X is assumed to be an affine function of X; less commonly, the median or some other quantile of the conditional distribution of y given X is expressed as a linear function of X. Like all forms of regression analysis, linear regression focuses on the conditional probability distribution of y given X, rather than on the joint probability distribution of y and X, which is the domain of multivariate analysis.

Linear regression was the first type of regression analysis to be studied rigorously, and to be used extensively in practical applications. This is because models which depend linearly on their unknown parameters are easier to fit than models which are non-linearly related to their parameters and because the statistical properties of the resulting estimators are easier to determine.

Before running the code for Linear Regression, we must make the model perfect for it.



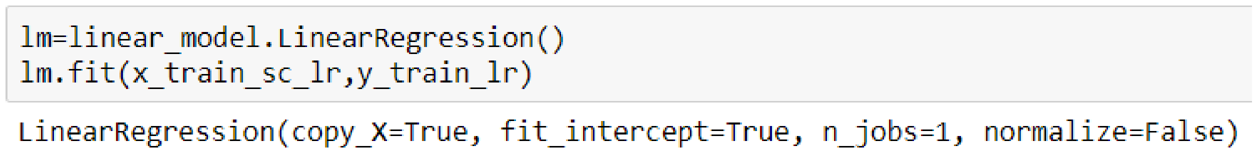
```
df_train_lr,df_test_lr = train_test_split(df,train_size=0.7,random_state=42)
x_train_lr=df_train_lr.iloc[:,1:53]
y_train_lr=df_train_lr['Target_variable']
scaler.fit(x_train_lr)
x_train_sc_lr=scaler.transform(x_train_lr)
x_test_lr=df_test_lr.iloc[:,1:53]
y_test_lr=df_test_lr['Target_variable']
scaler.fit(x_test_lr)
x_test_sc_lr=scaler.transform(x_test_lr)
```

From the above screenshot, we can understand that ‘scaler’ is done to bring all the data in the dataset to one scale as there are certain columns like ‘*Page_popularity*’ which signifies how much ‘*likes*’ that page has and it is in ten thousand and there are columns which signify ‘*time*’ where value is at most 72 as we are predicting for only

3 days. While running algorithm, our model should not understand '*Page_popularity*' has more importance just because the values in that columns is high as compared to any other column in our dataset. So, it is necessary for us to bring all data in every column in single scale.

Once scaling is done, we are splitting the dataset into 70:30 for training and testing.

We are selecting '*traget_variable*' as our '*y*' and rest all features as our '*x*'.

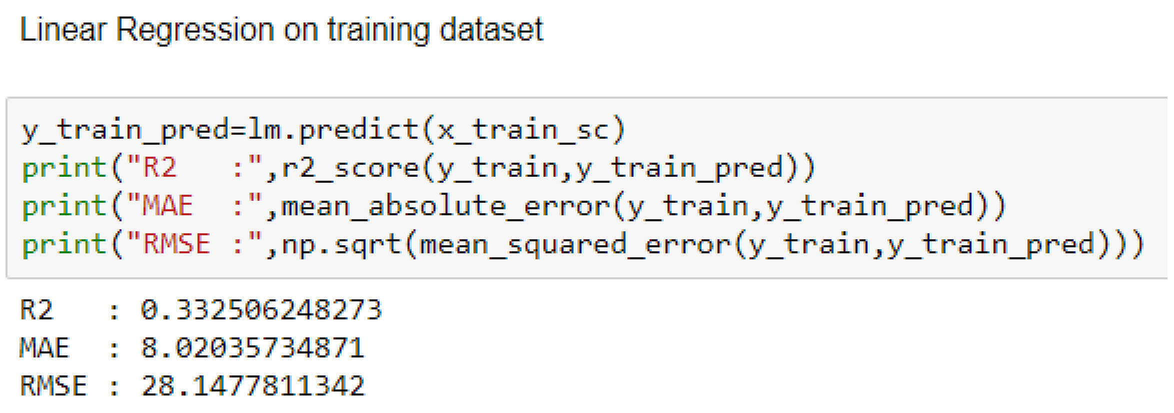


```
lm=linear_model.LinearRegression()
lm.fit(x_train_sc_lr,y_train_lr)
```

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)
```

After creating the model, now we can check how accurate our model is. RMSE helps us to find how accurate or model is and also whether our model is overfitting or not.

Training data.

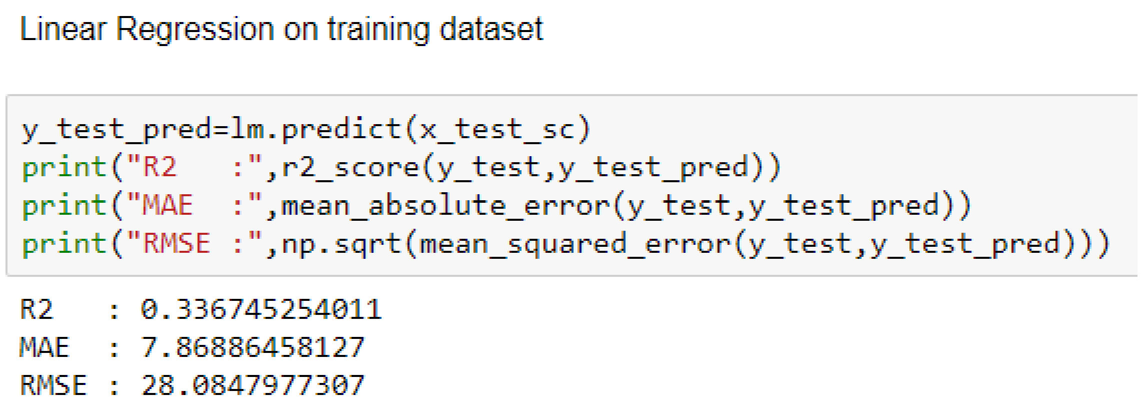


Linear Regression on training dataset

```
y_train_pred=lm.predict(x_train_sc)
print("R2   :",r2_score(y_train,y_train_pred))
print("MAE  :",mean_absolute_error(y_train,y_train_pred))
print("RMSE :",np.sqrt(mean_squared_error(y_train,y_train_pred)))
```

```
R2   : 0.332506248273
MAE  : 8.02035734871
RMSE : 28.1477811342
```

Testing data



Linear Regression on training dataset

```
y_test_pred=lm.predict(x_test_sc)
print("R2   :",r2_score(y_test,y_test_pred))
print("MAE  :",mean_absolute_error(y_test,y_test_pred))
print("RMSE :",np.sqrt(mean_squared_error(y_test,y_test_pred)))
```

```
R2   : 0.336745254011
MAE  : 7.86886458127
RMSE : 28.0847977307
```

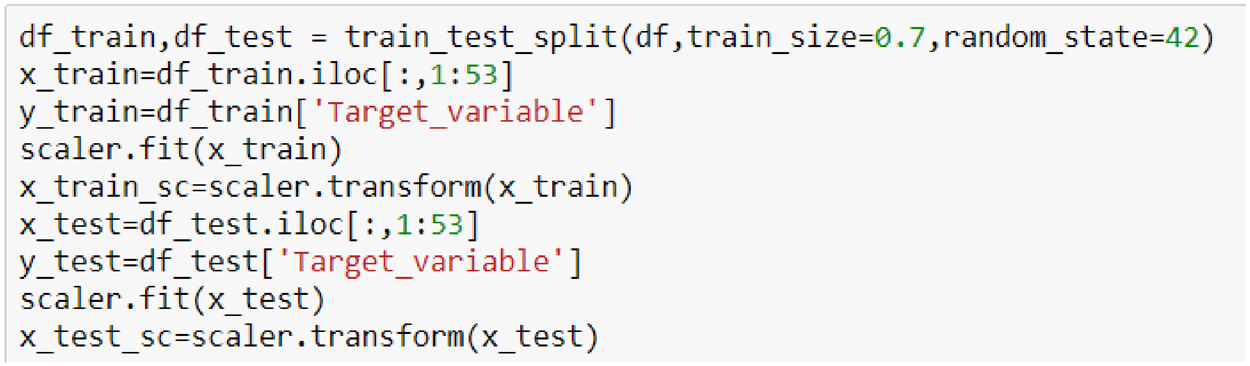
We can observe there is not significant difference between RMSE score of training data and testing data. So, this model is reliable.

Decision Tree Regressor

A decision tree is a decision support tool that uses a tree-like graph or model of decisions and their possible consequences, including chance event outcomes, resource costs, and utility. It is one way to display an algorithm that only contains conditional control statements.

Decision trees are commonly used in operations research, specifically in decision analysis, to help identify a strategy most likely to reach a goal, but are also a popular tool in machine learning.

Before running the code for decision tree regressor, we must make the model perfect for it.



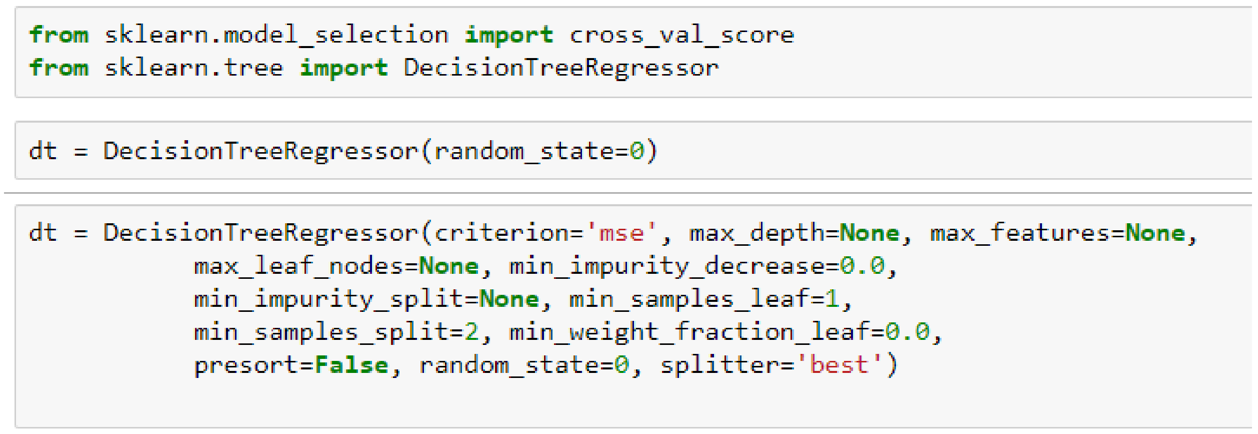
```
df_train,df_test = train_test_split(df,train_size=0.7,random_state=42)
x_train=df_train.iloc[:,1:53]
y_train=df_train['Target_variable']
scaler.fit(x_train)
x_train_sc=scaler.transform(x_train)
x_test=df_test.iloc[:,1:53]
y_test=df_test['Target_variable']
scaler.fit(x_test)
x_test_sc=scaler.transform(x_test)
```

From the above screenshot, we can understand that ‘scaler’ is done to bring all the data in the dataset to one scale as there are certain columns like ‘*Page_popularity*’ which signifies how much ‘*likes*’ that page has and it is in ten thousand and there are columns which signify ‘*time*’ where value is at most 72 as we are predicting for only 3 days. While running algorithm, our model should not understand ‘*Page_popularity*’ has more importance just because the values in that columns is high as compared to any other column in our dataset. So, it is necessary for us to bring all data in every column in single scale.

Once scaling is done, we are splitting the dataset into 70:30 for training and testing.

We are selecting ‘*traget_variable*’ as our ‘*y*’ and rest all features as our ‘*x*’.

Decision Tree



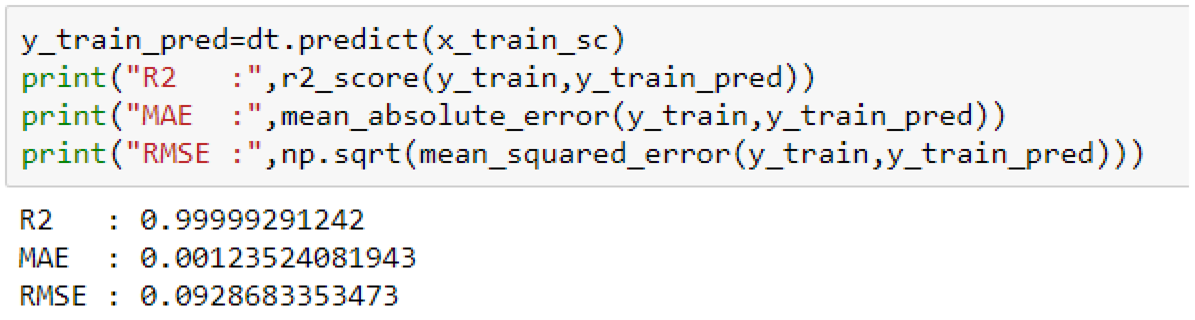
```
from sklearn.model_selection import cross_val_score
from sklearn.tree import DecisionTreeRegressor
```

```
dt = DecisionTreeRegressor(random_state=0)
```

```
dt = DecisionTreeRegressor(criterion='mse', max_depth=None, max_features=None,
           max_leaf_nodes=None, min_impurity_decrease=0.0,
           min_impurity_split=None, min_samples_leaf=1,
           min_samples_split=2, min_weight_fraction_leaf=0.0,
           presort=False, random_state=0, splitter='best')
```

After creating the model, now we can check how accurate our model is. RMSE helps us to find how accurate or model is and whether our model is overfitting or not.

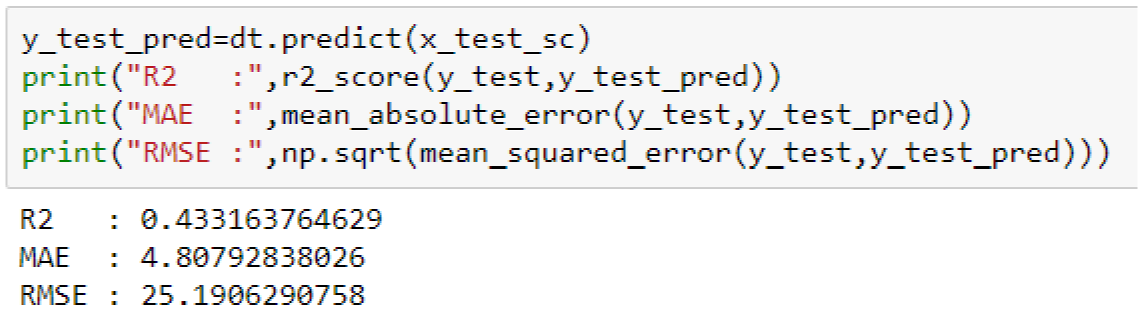
Training data.



```
y_train_pred=dt.predict(x_train_sc)
print("R2   :",r2_score(y_train,y_train_pred))
print("MAE  :",mean_absolute_error(y_train,y_train_pred))
print("RMSE :",np.sqrt(mean_squared_error(y_train,y_train_pred)))
```

```
R2   : 0.99999291242
MAE  : 0.00123524081943
RMSE : 0.0928683353473
```

Testing data



```
y_test_pred=dt.predict(x_test_sc)
print("R2   :",r2_score(y_test,y_test_pred))
print("MAE  :",mean_absolute_error(y_test,y_test_pred))
print("RMSE :",np.sqrt(mean_squared_error(y_test,y_test_pred)))
```

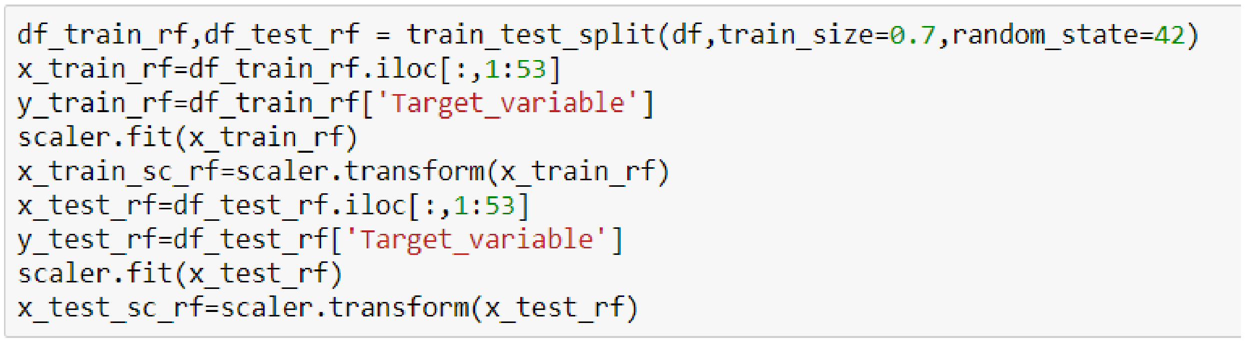
```
R2   : 0.433163764629
MAE  : 4.80792838026
RMSE : 25.1906290758
```

We can observe there is a very less change between RMSE score of training data and testing data. So, this model is reliable is very high scale.

Random Forest

Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks, that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees. Random decision forests correct for decision trees' habit of overfitting to their training set.

Before running the code for Random forest, we must make the model perfect for it.

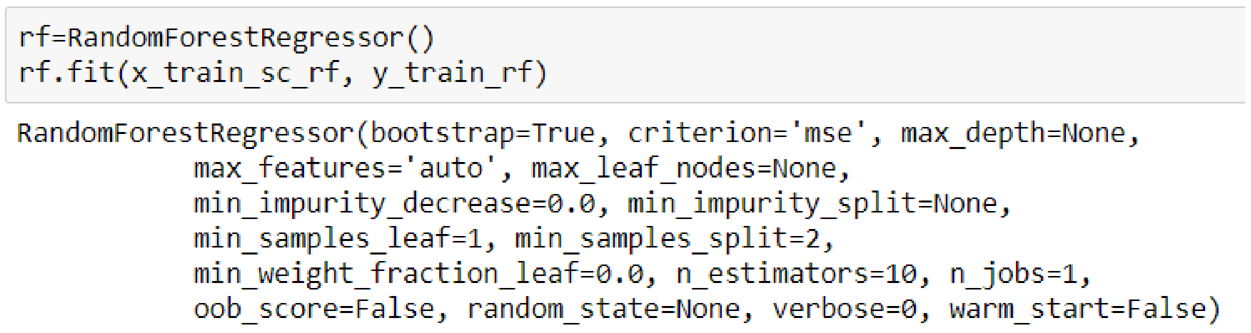


```
df_train_rf,df_test_rf = train_test_split(df,train_size=0.7,random_state=42)
x_train_rf=df_train_rf.iloc[:,1:53]
y_train_rf=df_train_rf['Target_variable']
scaler.fit(x_train_rf)
x_train_sc_rf=scaler.transform(x_train_rf)
x_test_rf=df_test_rf.iloc[:,1:53]
y_test_rf=df_test_rf['Target_variable']
scaler.fit(x_test_rf)
x_test_sc_rf=scaler.transform(x_test_rf)
```

From the above screenshot, we can understand that 'scaler' is done to bring all the data in the dataset to one scale as there are certain columns like '*Page_popularity*' which signifies how much '*likes*' that page has and it is in ten thousand and there are columns which signify '*time*' where value is at most 72 as we are predicting for only 3 days. While running algorithm, our model should not understand '*Page_popularity*' has more importance just because the values in that columns is high as compared to any other column in our dataset. So, it is necessary for us to bring all data in every column in single scale.

Once scaling is done, we are splitting the dataset into 70:30 for training and testing.

We are selecting '*traget_variable*' as our '*y*' and rest all features as our '*x*'.

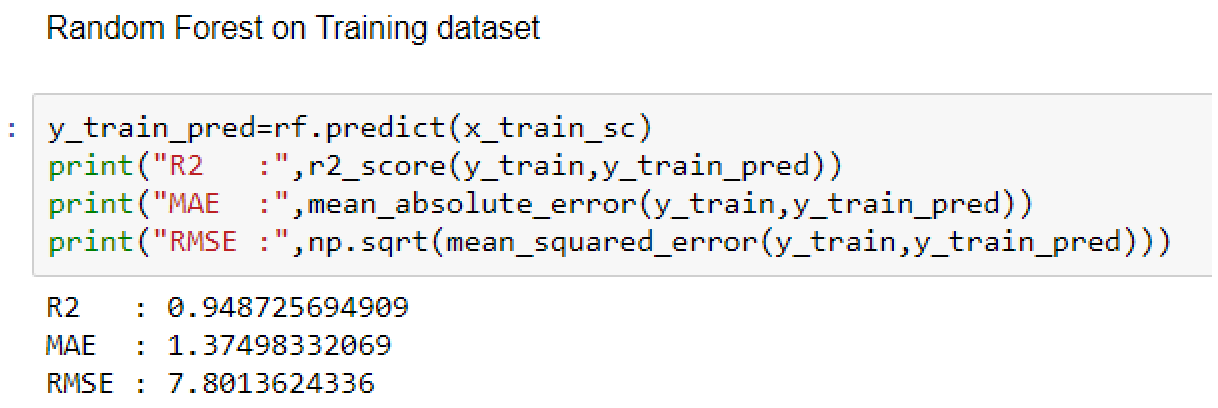


```
rf=RandomForestRegressor()
rf.fit(x_train_sc_rf, y_train_rf)
```

```
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
           max_features='auto', max_leaf_nodes=None,
           min_impurity_decrease=0.0, min_impurity_split=None,
           min_samples_leaf=1, min_samples_split=2,
           min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=1,
           oob_score=False, random_state=None, verbose=0, warm_start=False)
```

After creating the model, now we can check how accurate our model is. RMSE helps us to find how accurate or model is and whether our model is overfitting or not.

Training data.

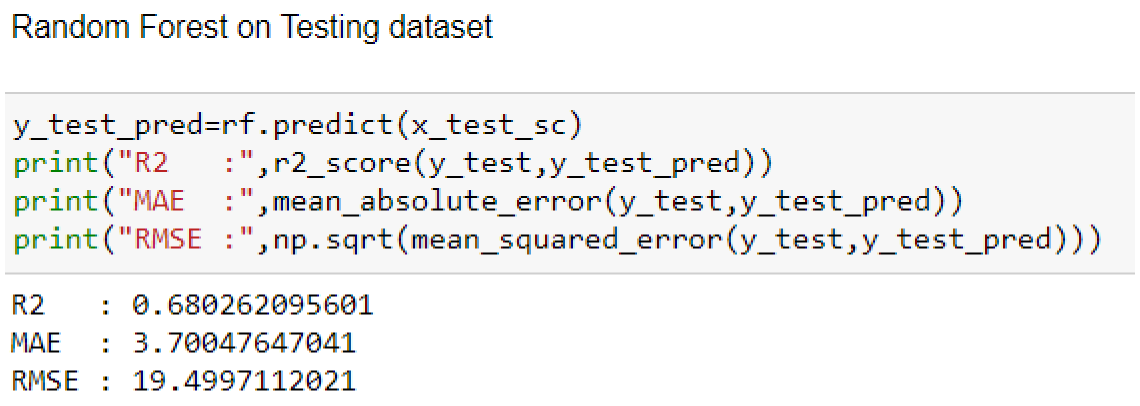


Random Forest on Training dataset

```
y_train_pred=rf.predict(x_train_sc)
print("R2   :",r2_score(y_train,y_train_pred))
print("MAE  :",mean_absolute_error(y_train,y_train_pred))
print("RMSE :",np.sqrt(mean_squared_error(y_train,y_train_pred)))
```

```
R2   : 0.948725694909
MAE  : 1.37498332069
RMSE : 7.8013624336
```

Testing data



Random Forest on Testing dataset

```
y_test_pred=rf.predict(x_test_sc)
print("R2   :",r2_score(y_test,y_test_pred))
print("MAE  :",mean_absolute_error(y_test,y_test_pred))
print("RMSE :",np.sqrt(mean_squared_error(y_test,y_test_pred)))
```

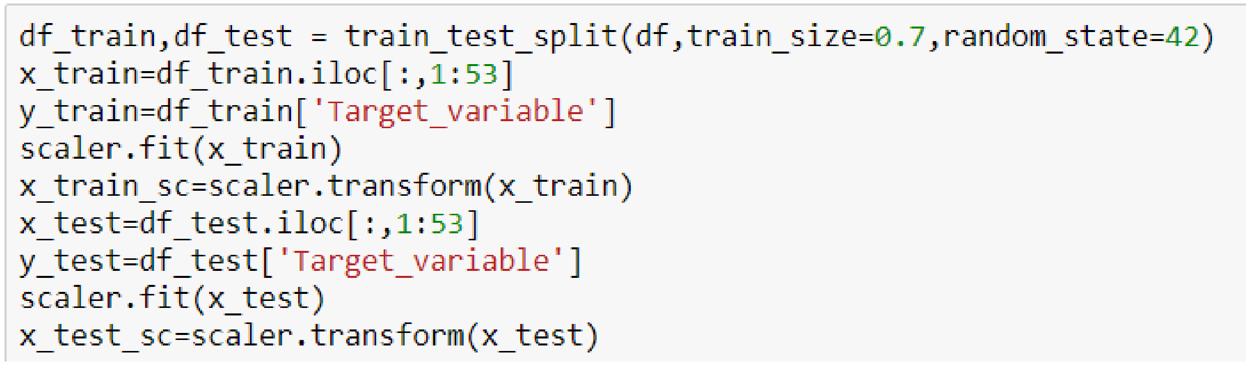
```
R2   : 0.680262095601
MAE  : 3.70047647041
RMSE : 19.4997112021
```

We can observe there is a very less change between RMSE score of training data and testing data. So, this model is reliable is very high scale.

Support Vector Regression

Support Vector Machine can also be used as a regression method, maintaining all the key features that characterize the algorithm (maximal margin). The Support Vector Regression (SVR) uses the same principles as the SVM for classification, with only a few minor differences. First, because output is a real number it becomes very difficult to predict the information at hand, which has infinite possibilities. In the case of regression, a margin of tolerance (epsilon) is set in approximation to the SVM which would have already requested from the problem. But besides this fact, there is also a more complicated reason, the algorithm is more complicated therefore to be taken in consideration. However, the main idea is always the same: to minimize error, individualizing the hyperplane which maximizes the margin, keeping in mind that part of the error is tolerated.

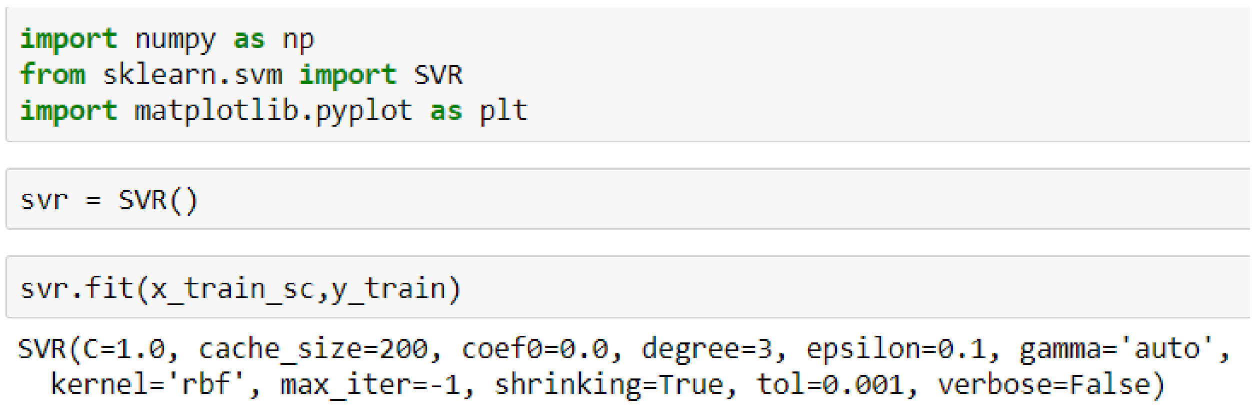
Before running the code for Support Vector Regression, we must make the model perfect for it.



```
df_train,df_test = train_test_split(df,train_size=0.7,random_state=42)
x_train=df_train.iloc[:,1:53]
y_train=df_train['Target_variable']
scaler.fit(x_train)
x_train_sc=scaler.transform(x_train)
x_test=df_test.iloc[:,1:53]
y_test=df_test['Target_variable']
scaler.fit(x_test)
x_test_sc=scaler.transform(x_test)
```

From the above screenshot, we can understand that ‘scaler’ is done to bring all the data in the dataset to one scale as there are certain columns like ‘*Page_popularity*’ which signifies how much ‘*likes*’ that page has and it is in ten thousand and there are columns which signify ‘*time*’ where value is at most 72 as we are predicting for only 3 days. While running algorithm, our model should not understand ‘*Page_popularity*’ has more importance just because the values in that columns is high as compared to any other column in our dataset. So, it is necessary for us to bring all data in every column in single scale.

Once scaling is done, we are splitting the dataset into 70:30 for training and testing. We are selecting ‘*traget_variable*’ as our ‘*y*’ and rest all features as our ‘*x*’.



```
import numpy as np
from sklearn.svm import SVR
import matplotlib.pyplot as plt
```

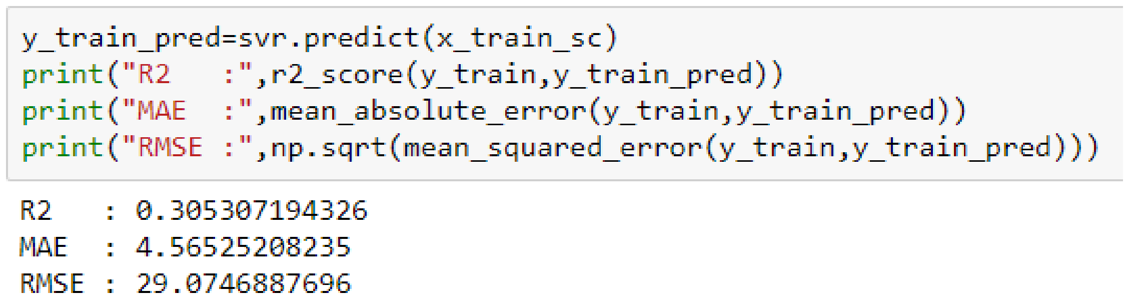
```
svr = SVR()
```

```
svr.fit(x_train_sc,y_train)
```

```
SVR(C=1.0, cache_size=200, coef0=0.0, degree=3, epsilon=0.1, gamma='auto',
  kernel='rbf', max_iter=-1, shrinking=True, tol=0.001, verbose=False)
```

After creating the model, now we can check how accurate our model is. RMSE helps us to find how accurate or model is and whether our model is overfitting or not.

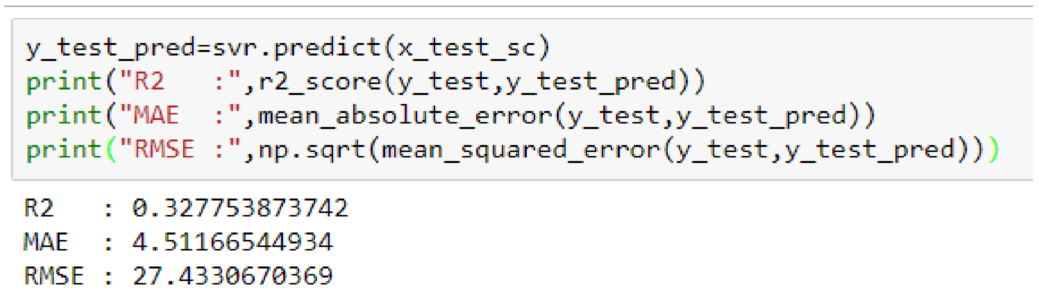
Training data.



```
y_train_pred=svr.predict(x_train_sc)
print("R2   :",r2_score(y_train,y_train_pred))
print("MAE  :",mean_absolute_error(y_train,y_train_pred))
print("RMSE :",np.sqrt(mean_squared_error(y_train,y_train_pred)))
```

```
R2   : 0.305307194326
MAE  : 4.56525208235
RMSE : 29.0746887696
```

Testing data



```
y_test_pred=svr.predict(x_test_sc)
print("R2   :",r2_score(y_test,y_test_pred))
print("MAE  :",mean_absolute_error(y_test,y_test_pred))
print("RMSE :",np.sqrt(mean_squared_error(y_test,y_test_pred)))
```

```
R2   : 0.327753873742
MAE  : 4.51166544934
RMSE : 27.4330670369
```

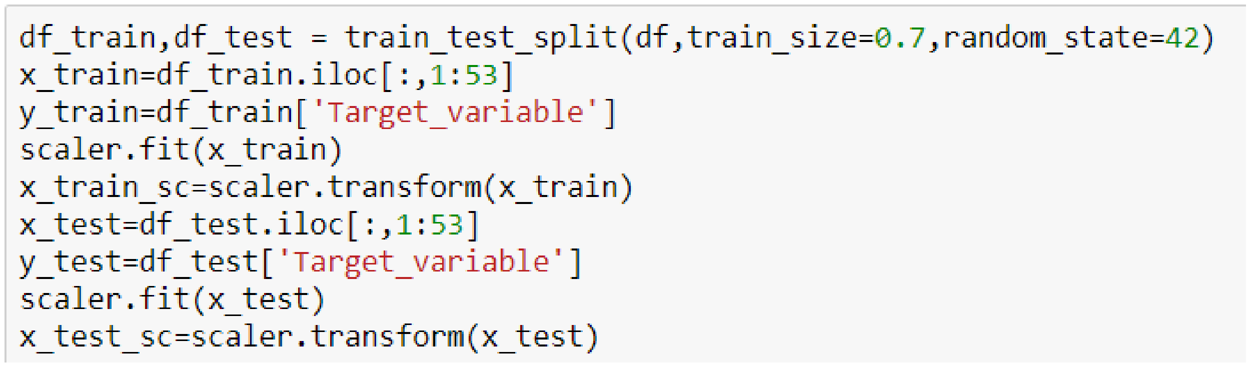
We can observe there is a very less change between RMSE score of training data and testing data. So, this model is reliable is very high scale.

Neural Network

Artificial neural networks (ANNs) or connectionist systems are computing systems vaguely inspired by the biological neural networks that constitute animal brains. Such systems "learn" (i.e. progressively improve performance on) tasks by considering examples, generally without task-specific programming. For example, in image recognition, they might learn to identify images that contain cats by analyzing example images that have been manually labeled as "cat" or "no cat" and using the results to identify cats in other images. They do this without any a priori knowledge about cats, e.g., that they have fur, tails, whiskers and cat-like faces. Instead, they evolve their own set of relevant characteristics from the learning material that they process.

An ANN is based on a collection of connected units or nodes called artificial neurons (a simplified version of biological neurons in an animal brain). Each connection (a simplified version of a synapse) between artificial neurons can transmit a signal from one to another. The artificial neuron that receives the signal can process it and then signal artificial neurons connected to it.

Before running the code for Neural Network, we must make the model perfect for it.



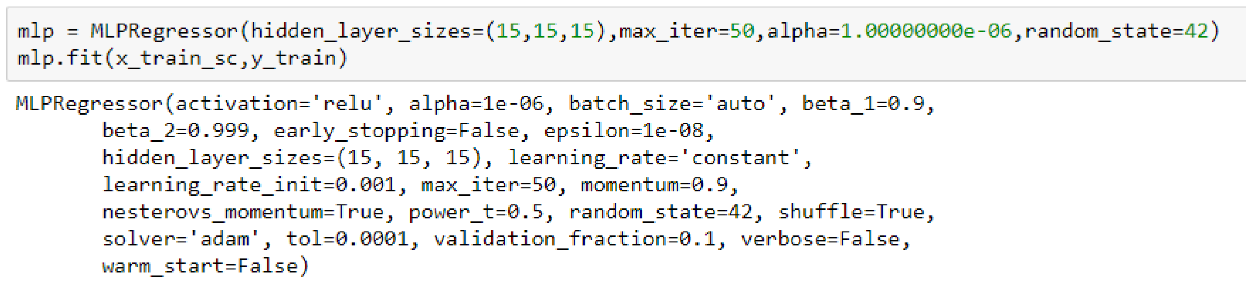
```
df_train,df_test = train_test_split(df,train_size=0.7,random_state=42)
x_train=df_train.iloc[:,1:53]
y_train=df_train['Target_variable']
scaler.fit(x_train)
x_train_sc=scaler.transform(x_train)
x_test=df_test.iloc[:,1:53]
y_test=df_test['Target_variable']
scaler.fit(x_test)
x_test_sc=scaler.transform(x_test)
```

From the above screenshot, we can understand that ‘scaler’ is done to bring all the data in the dataset to one scale as there are certain columns like ‘*Page_popularity*’ which signifies how much ‘*likes*’ that page has and it is in ten thousand and there are columns which signify ‘*time*’ where value is at most 72 as we are predicting for only 3 days. While running algorithm, our model should not understand

‘*Page_popularity*’ has more importance just because the values in that columns is high as compared to any other column in our dataset. So, it is necessary for us to bring all data in every column in single scale.

Once scaling is done, we are splitting the dataset into 70:30 for training and testing.

We are selecting ‘*traget_variable*’ as our ‘*y*’ and rest all features as our ‘*x*’.

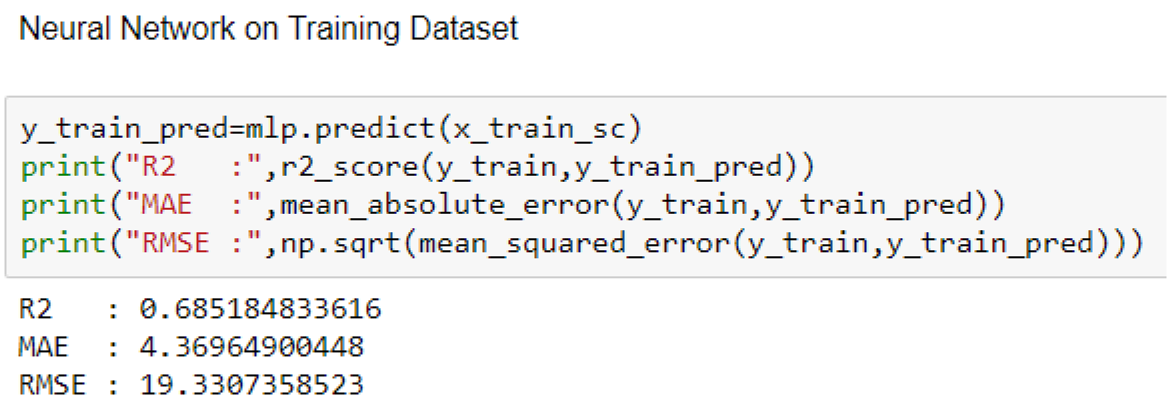


```
mlp = MLPRegressor(hidden_layer_sizes=(15,15,15),max_iter=50,alpha=1.00000000e-06,random_state=42)
mlp.fit(x_train_sc,y_train)
```

```
MLPRegressor(activation='relu', alpha=1e-06, batch_size='auto', beta_1=0.9,
       beta_2=0.999, early_stopping=False, epsilon=1e-08,
       hidden_layer_sizes=(15, 15, 15), learning_rate='constant',
       learning_rate_init=0.001, max_iter=50, momentum=0.9,
       nesterovs_momentum=True, power_t=0.5, random_state=42, shuffle=True,
       solver='adam', tol=0.0001, validation_fraction=0.1, verbose=False,
       warm_start=False)
```

After creating the model, now we can check how accurate our model is. RMSE helps us to find how accurate or model is and whether our model is overfitting or not.

Training data.

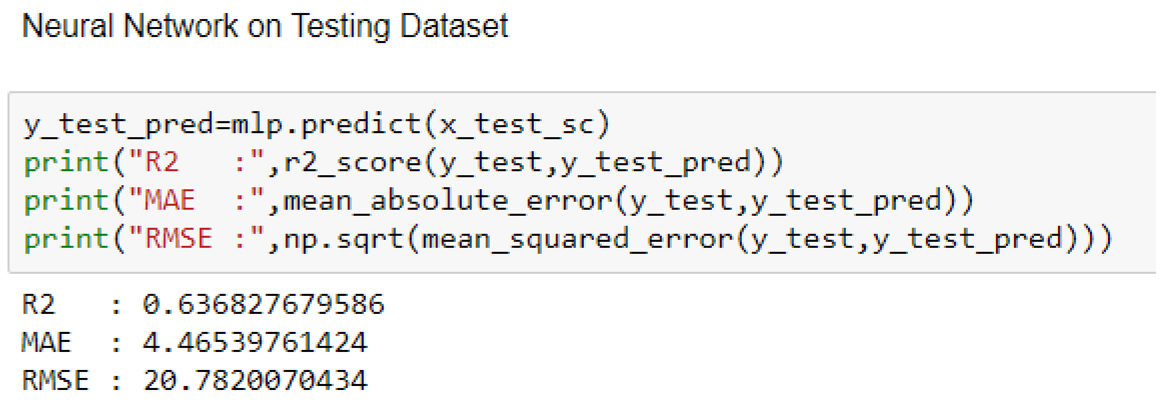


Neural Network on Training Dataset

```
y_train_pred=mlp.predict(x_train_sc)
print("R2   :",r2_score(y_train,y_train_pred))
print("MAE  :",mean_absolute_error(y_train,y_train_pred))
print("RMSE :",np.sqrt(mean_squared_error(y_train,y_train_pred)))
```

```
R2   : 0.685184833616
MAE  : 4.36964900448
RMSE : 19.3307358523
```

Testing data



Neural Network on Testing Dataset

```
y_test_pred=mlp.predict(x_test_sc)
print("R2   :",r2_score(y_test,y_test_pred))
print("MAE  :",mean_absolute_error(y_test,y_test_pred))
print("RMSE :",np.sqrt(mean_squared_error(y_test,y_test_pred)))
```

```
R2   : 0.636827679586
MAE  : 4.46539761424
RMSE : 20.7820070434
```

We can observe there is a very less change between RMSE score of training data and testing data. So, this model is reliable is very high scale.

K Nearest Regressor

In pattern recognition, the k-nearest neighbors algorithm (k-NN) is a non-parametric method used for classification and regression. In both cases, the input consists of the k closest training examples in the feature space. The output depends on whether k-NN is used for classification or regression:

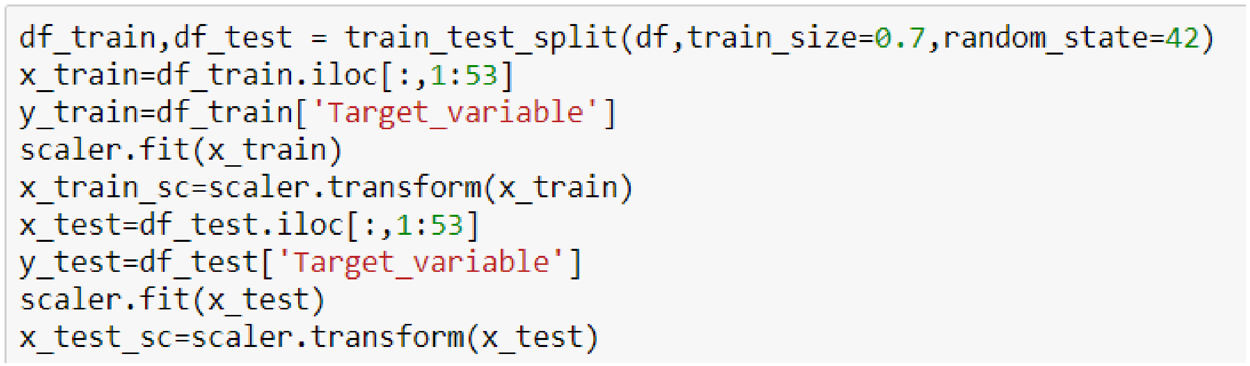
- In k-NN classification, the output is a class membership. An object is classified by a majority vote of its neighbors, with the object being assigned to the class most common among its k nearest neighbors (k is a positive integer, typically small). If k = 1, then the object is simply assigned to the class of that single nearest neighbor.

- In k-NN regression, the output is the property value for the object. This value is the average of the values of its k nearest neighbors.

k-NN is a type of instance-based learning, or lazy learning, where the function is only approximated locally and all computation is deferred until classification. The k-NN algorithm is among the simplest of all machine learning algorithms.

Both for classification and regression, a useful technique can be to assign weight to the contributions of the neighbors, so that the nearer neighbors contribute more to the average than the more distant ones. For example, a common weighting scheme consists in giving each neighbor a weight of 1/d, where d is the distance to the neighbor.

Before running the code for k-nearest regressor, we must make the model perfect for it.

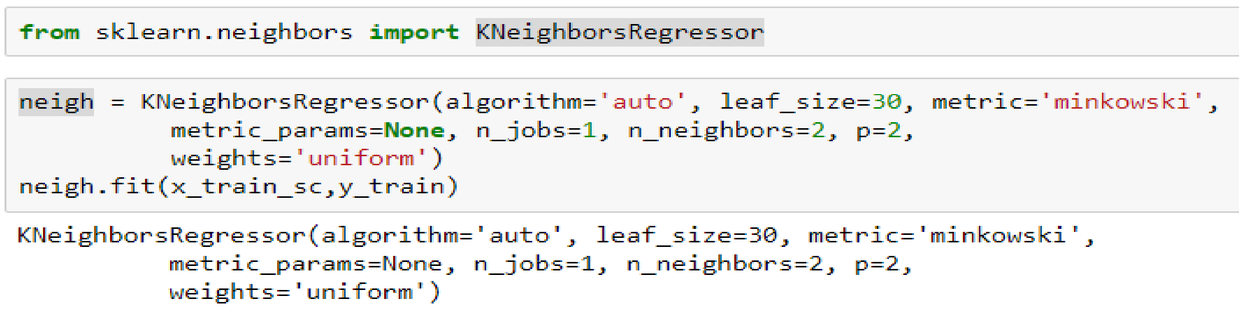


```
df_train,df_test = train_test_split(df,train_size=0.7,random_state=42)
x_train=df_train.iloc[:,1:53]
y_train=df_train['Target_variable']
scaler.fit(x_train)
x_train_sc=scaler.transform(x_train)
x_test=df_test.iloc[:,1:53]
y_test=df_test['Target_variable']
scaler.fit(x_test)
x_test_sc=scaler.transform(x_test)
```

From the above screenshot, we can understand that 'scaler' is done to bring all the data in the dataset to one scale as there are certain columns like '*Page_popularity*' which signifies how much '*likes*' that page has and it is in ten thousand and there are columns which signify '*time*' where value is at most 72 as we are predicting for only 3 days. While running algorithm, our model should not understand '*Page_popularity*' has more importance just because the values in that columns is high as compared to any other column in our dataset. So, it is necessary for us to bring all data in every column in single scale.

Once scaling is done, we are splitting the dataset into 70:30 for training and testing.

We are selecting '*traget_variable*' as our '*y*' and rest all features as our '*x*'.



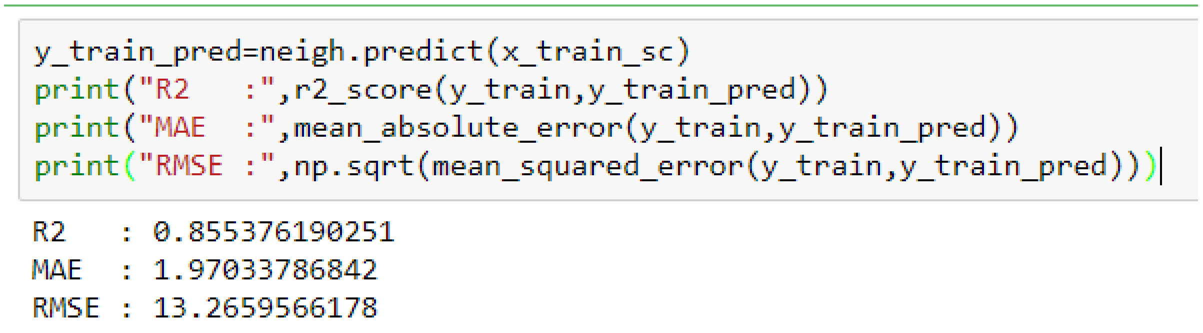
```
from sklearn.neighbors import KNeighborsRegressor
```

```
neigh = KNeighborsRegressor(algorithm='auto', leaf_size=30, metric='minkowski',
          metric_params=None, n_jobs=1, n_neighbors=2, p=2,
          weights='uniform')
neigh.fit(x_train_sc,y_train)
```

```
KNeighborsRegressor(algorithm='auto', leaf_size=30, metric='minkowski',
          metric_params=None, n_jobs=1, n_neighbors=2, p=2,
          weights='uniform')
```

After creating the model, now we can check how accurate our model is. RMSE helps us to find how accurate or model is and whether our model is overfitting or not.

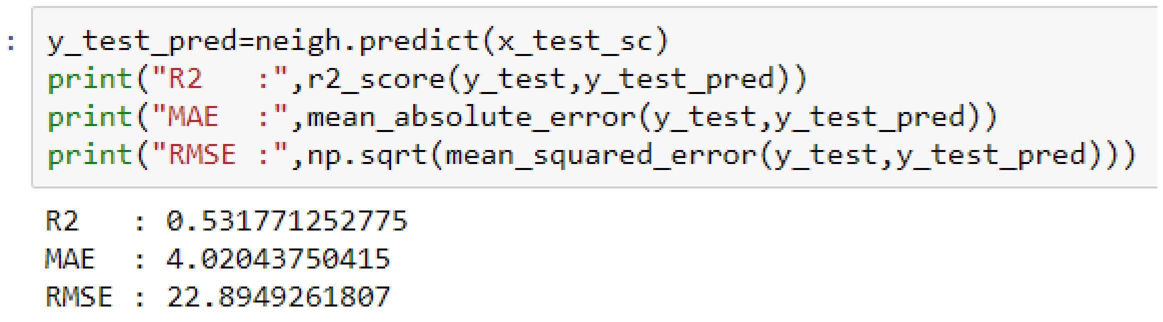
Training data.



```
y_train_pred=neigh.predict(x_train_sc)
print("R2   :",r2_score(y_train,y_train_pred))
print("MAE  :",mean_absolute_error(y_train,y_train_pred))
print("RMSE :",np.sqrt(mean_squared_error(y_train,y_train_pred)))
```

```
R2   : 0.855376190251
MAE  : 1.97033786842
RMSE : 13.2659566178
```

Testing data



```
: y_test_pred=neigh.predict(x_test_sc)
  print("R2   :",r2_score(y_test,y_test_pred))
  print("MAE  :",mean_absolute_error(y_test,y_test_pred))
  print("RMSE :",np.sqrt(mean_squared_error(y_test,y_test_pred)))
```

```
R2   : 0.531771252775
MAE  : 4.02043750415
RMSE : 22.8949261807
```

We can observe there is a very less change between RMSE score of training data and testing data. So, this model is reliable is very high scale.

Conclusion

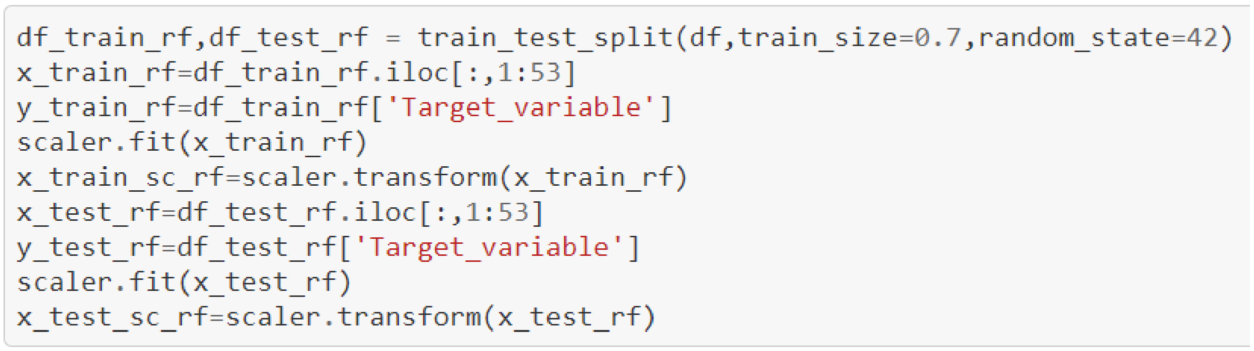
As we didn’t get satisfactory results with decision tree algorithm and we feel the model is overfitting, we have decided to not use it and proceed with the rest algorithms which we have performed.

Feature Selection

Boruta

Boruta is an all relevant feature selection method, while most other are minimal optimal; this means it tries to find all features carrying information usable for prediction, rather than finding a possibly compact subset of features on which some classifier has a minimal error.

We must create the model first.



```
df_train_rf,df_test_rf = train_test_split(df,train_size=0.7,random_state=42)
x_train_rf=df_train_rf.iloc[:,1:53]
y_train_rf=df_train_rf['Target_variable']
scaler.fit(x_train_rf)
x_train_sc_rf=scaler.transform(x_train_rf)
x_test_rf=df_test_rf.iloc[:,1:53]
y_test_rf=df_test_rf['Target_variable']
scaler.fit(x_test_rf)
x_test_sc_rf=scaler.transform(x_test_rf)
```

Once we are done with creating model, we can run the Boruta code.

```python
import pandas as pd
#from sklearn.ensemble import RandomForestClassifier
from boruta import BorutaPy

# load X and y
# NOTE BorutaPy accepts numpy arrays only, hence the .values attribute
X_rf = x_train_sc_rf
y_rf = y_train_rf

# define random forest classifier, with utilising all cores and
# sampling in proportion to y labels
rf = RandomForestRegressor(n_jobs=-1, max_depth=25)

# define Boruta feature selection method
feat_selector_rf = BorutaPy(rf, n_estimators='auto', verbose=2)

# find all relevant features
feat_selector_rf.fit(X_rf, y_rf)
```

When we run the above code, we get the following output.

```
BorutaPy(alpha=0.05,
     estimator=RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=25,
           max_features='auto', max_leaf_nodes=None,
           min_impurity_decrease=0.0, min_impurity_split=None,
           min_samples_leaf=1, min_samples_split=2,
           min_weight_fraction_leaf=0.0, n_estimators=23, n_jobs=-1,
           oob_score=False,
           random_state=<mtrand.RandomState object at 0x000001423415FFC0>,
           verbose=0, warm_start=False),
     max_iter=100, n_estimators='auto', perc=100,
     random_state=<mtrand.RandomState object at 0x000001423415FFC0>,
     two_step=True, verbose=2)
```

To get the names of the important columns, we have to run the following code.

```
x_train_rf.columns[feat_selector_rf.support_]
```

And we get the following output.

```
Index(['Page_talking_about', 'c2', 'c3', 'c4', 'c13', 'c15', 'c18', 'c25',
       'CC1', 'CC2', 'CC4', 'CC5', 'Base_time', 'Post_length_char_count',
       'Post_share_count'],
      dtype='object')
```

Now, we can create a model using these features.

```
column_list = x_train_rf.columns[feat_selector_rf.support_]
x_train_rf=df_train_rf.iloc[:,1:53]
x_train_rf= x_train_rf[column_list]
print(x_train_rf.shape)
y_train_rf=df_train_rf['Target_variable']
scaler.fit(x_train_rf)
x_train_sc_rf=scaler.transform(x_train_rf)
x_test_rf=df_test_rf.iloc[:,1:53]
x_test_rf = x_test_rf[column_list]
print(x_test_rf.shape)
y_test_rf=df_test_rf['Target_variable']
x_test_sc_rf=scaler.transform(x_test_rf)
```

Now we can run ‘Random Forest’ on this.

```
rf = RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=25,
           max_features='sqrt', max_leaf_nodes=None,
           min_impurity_decrease=0.0, min_impurity_split=None,
           min_samples_leaf=1, min_samples_split=2,
           min_weight_fraction_leaf=0.0, n_estimators=23, n_jobs=-1,
           oob_score=False,
           random_state=42,
           verbose=0, warm_start=False)
```

Once the model is done, we can retrieve RMSE and r-square values of the model on training and testing data.

Training Data

```
y_train_pred_rf=rf.predict(x_train_sc_rf)
print("R2   :",r2_score(y_train_rf,y_train_pred_rf))
print("RMSE :",np.sqrt(mean_squared_error(y_train_rf,y_train_pred_rf)))
print("MAE  :",mean_absolute_error(y_train_rf,y_train_pred_rf))
```

```
R2   : 0.95649216056
RMSE : 7.18628309458
MAE  : 1.4252996339
```

Testing Data

```
y_test_pred_rf =rf.predict(x_test_sc_rf)
print("R2   :",r2_score(y_test_rf,y_test_pred_rf))
print("MAE  :",mean_absolute_error(y_test_rf,y_test_pred_rf))
print("RMSE :",np.sqrt(mean_squared_error(y_test_rf,y_test_pred_rf)))
```

```
R2   : 0.731210914096
MAE  : 3.1526667579
RMSE : 17.8787409748
```

RFSEC

We must import the necessary libraries.

```
from sklearn.datasets import make_friedman1
from sklearn.feature_selection import RFECV
from sklearn.svm import SVR
```

We are passing linear regressor and decision tree through RFSEC.

Linear Regressor

```
estimator = linear_model.LinearRegression()
```

```
selector = RFECV(estimator, step=1, cv=5)
selector.fit(x_train_sc_lr,y_train_lr )
```

List of key features are as follows.

```
x_train_lr.columns[selector.support_]
```

```
Index(['Page_visited_no_of_times', 'Page_talking_about', 'Page_category', 'c1',
       'c2', 'c3', 'c4', 'c5', 'c6', 'c7', 'c8', 'c9', 'c10', 'c11', 'c12',
       'c13', 'c14', 'c15', 'c16', 'c17', 'c18', 'c19', 'c20', 'c21', 'c22',
       'c23', 'c24', 'c25', 'CC1', 'CC2', 'CC3', 'CC4', 'CC5',
       'Base_time_something', 'Post_length_char_count', 'Post_share_count',
       'Post_promoted', 'Time_target', 'Sunday_post', 'Monday_post',
       'Tuesday_post', 'Wednesday_post', 'Thrusday_post', 'Friday_post',
       'Saturday_post', 'Sunday_base', 'Monday_base', 'Tuesday_base',
       'Wednesday_base', 'Thrusday_base', 'Friday_base', 'Saturday_base'],
      dtype='object')
```

Decision Tree

```
from sklearn import tree
dt_estimator = tree.DecisionTreeRegressor()
```

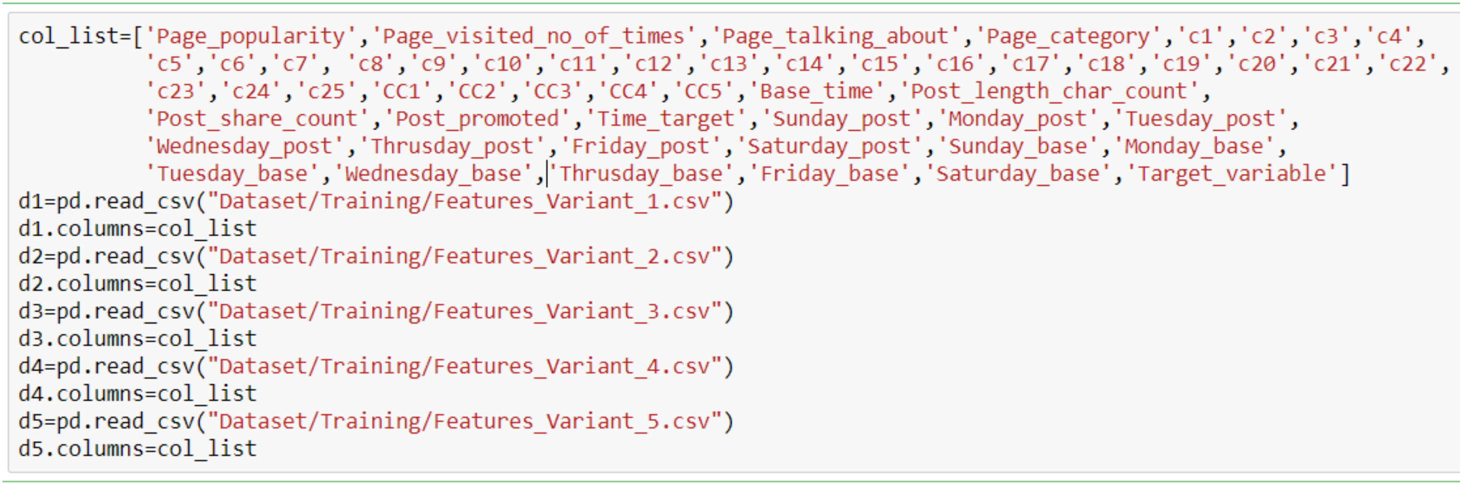
```
selector = RFECV(dt_estimator, step=1, cv=5)
selector.fit(x_train_sc_dt,y_train_dt )
```

List of key features in decision tree.

```
RFECV(cv=5,
   estimator=DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=None,
            splitter='best'),
   n_jobs=1, scoring=None, step=1, verbose=0)
```

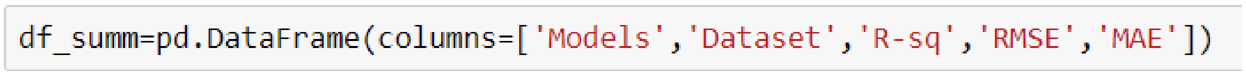
Building Model

Building a model is very important. When we are automating the calculation process, it is important for us to build model.



```
col_list=['Page_popularity','Page_visited_no_of_times','Page_talking_about','Page_category','c1','c2','c3','c4',
          'c5','c6','c7', 'c8','c9','c10','c11','c12','c13','c14','c15','c16','c17','c18','c19','c20','c21','c22',
          'c23','c24','c25','CC1','CC2','CC3','CC4','CC5','Base_time','Post_length_char_count',
          'Post_share_count','Post_promoted','Time_target','Sunday_post','Monday_post','Tuesday_post',
          'Wednesday_post','Thrusday_post','Friday_post','Saturday_post','Sunday_base','Monday_base',
          'Tuesday_base','Wednesday_base','Thrusday_base','Friday_base','Saturday_base','Target_variable']
d1=pd.read_csv("Dataset/Training/Features_Variant_1.csv")
d1.columns=col_list
d2=pd.read_csv("Dataset/Training/Features_Variant_2.csv")
d2.columns=col_list
d3=pd.read_csv("Dataset/Training/Features_Variant_3.csv")
d3.columns=col_list
d4=pd.read_csv("Dataset/Training/Features_Variant_4.csv")
d4.columns=col_list
d5=pd.read_csv("Dataset/Training/Features_Variant_5.csv")
d5.columns=col_list
```

From the above screenshot, we can see that we are importing the files and taking only those columns, which are important. These data are received after performing feature engineering using Boruta and RFSEC.



```
df_summ=pd.DataFrame(columns=['Models','Dataset','R-sq','RMSE','MAE'])
```

We are creating a summary matrix which will have all the data of all the models(algorithms) which we have using and their scores in r-square, rmse and mae in testing and training data.

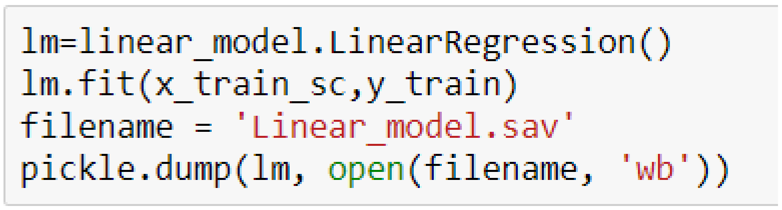
Pickle File

The pickle module implements a fundamental, but powerful algorithm for serializing and de-serializing a Python object structure. “Pickling” is the process whereby a Python object hierarchy is converted into a byte stream, and “unpickling” is the inverse operation, whereby a byte stream is converted back into an object hierarchy. Pickling (and unpickling) is alternatively known as “serialization”, "marshalling," or "flattening", however, to avoid confusion, the terms used here are "pickling" and "unpickling".

The pickle module has an optimized cousin called the cPickle module. As its name implies, cPickle is written in C, so it can be up to 1000 times faster than pickle. However it does not support subclassing of the Pickler() and Unpickler() classes, because in cPickle these are functions, not classes. Most applications have no need for this functionality, and can benefit from the improved performance of cPickle. Other than that, the interfaces of the two modules are nearly identical; the common interface is described in this manual and differences are pointed out where necessary.

Linear Regressor

We must run the below code to run the model and save it in pickle file.



```
lm=linear_model.LinearRegression()
lm.fit(x_train_sc,y_train)
filename = 'Linear_model.sav'
pickle.dump(lm, open(filename, 'wb'))
```

Once it's done, we can get the scores for training data and testing data using the model.

Training Data

Linear Regression on training dataset

```
y_train_pred=lm.predict(x_train_sc)
r2=r2_score(y_train,y_train_pred)
mae=mean_absolute_error(y_train,y_train_pred)
rmse=np.sqrt(mean_squared_error(y_train,y_train_pred))
mod='Linear Regression'
dataset='Training'
print(mod,' on ',dataset,' dataset ',' : ')
print("R2   :",r2_score(y_train,y_train_pred))
print("MAE  :",mean_absolute_error(y_train,y_train_pred))
print("RMSE :",np.sqrt(mean_squared_error(y_train,y_train_pred)))
data={'Models':mod,'Dataset':dataset,'R-sq':r2,'RMSE':rmse,'MAE':mae}
df_summ=df_summ.append(data,ignore_index=True)
```

```
Linear Regression  on  Training  dataset   :
R2   : 0.327121579213
MAE  : 8.01159873155
RMSE : 28.6145535415
```

Testing Data

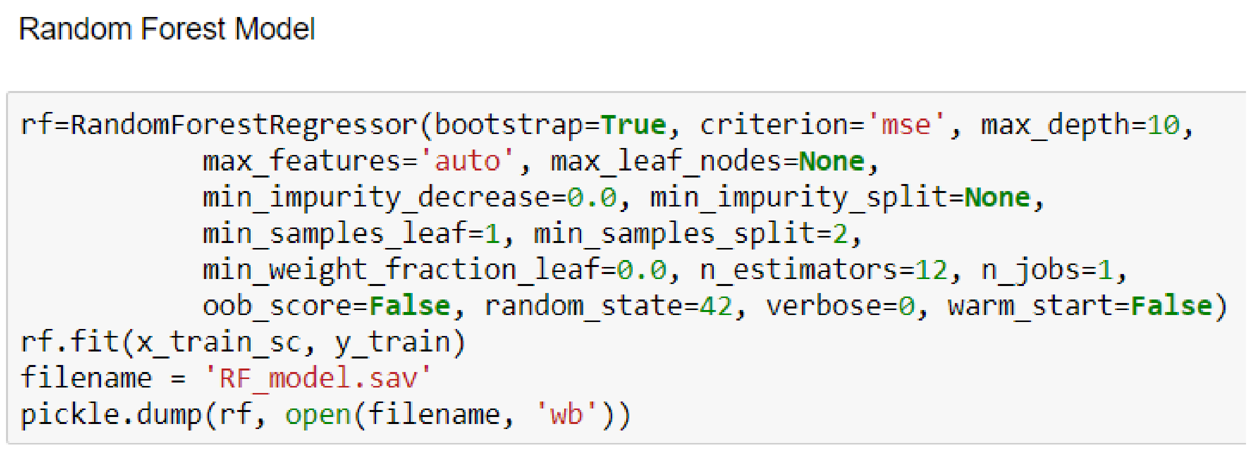
Linear Regression on Testing dataset

```
y_test_pred=lm.predict(x_test_sc)
r2=r2_score(y_test,y_test_pred)
mae=mean_absolute_error(y_test,y_test_pred)
rmse=np.sqrt(mean_squared_error(y_test,y_test_pred))
mod='Linear Regression'
dataset='Testing'
print(mod,' on ',dataset,' dataset ',' : ')
print("R2   :",r2_score(y_test,y_test_pred))
print("MAE  :",mean_absolute_error(y_test,y_test_pred))
print("RMSE :",np.sqrt(mean_squared_error(y_test,y_test_pred)))
data={'Models':mod,'Dataset':dataset,'R-sq':r2,'RMSE':rmse,'MAE':mae}
df_summ=df_summ.append(data,ignore_index=True)
```

```
Linear Regression  on  Testing  dataset   :
R2   : 0.350502039324
MAE  : 7.84511272228
RMSE : 26.9649181582
```

Random Forest

We must run the below code to run the model and save it in pickle file.

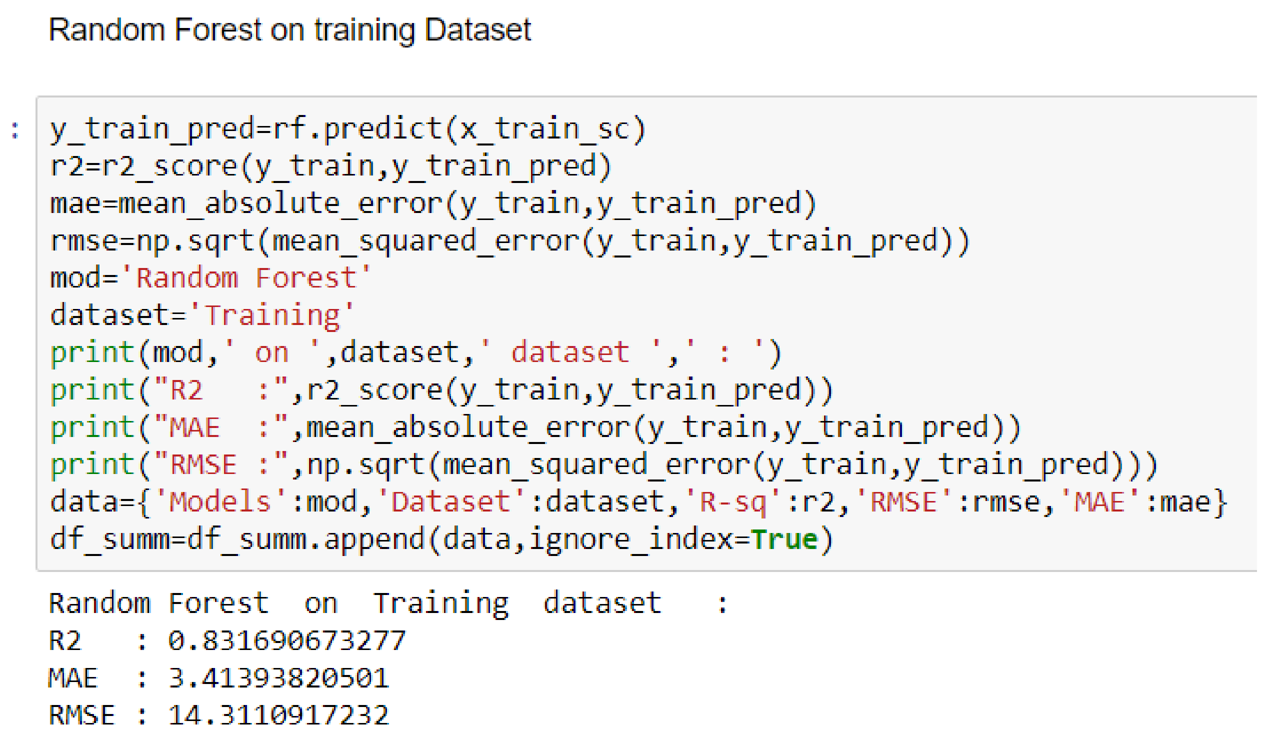


Random Forest Model

```
rf=RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=10,
           max_features='auto', max_leaf_nodes=None,
           min_impurity_decrease=0.0, min_impurity_split=None,
           min_samples_leaf=1, min_samples_split=2,
           min_weight_fraction_leaf=0.0, n_estimators=12, n_jobs=1,
           oob_score=False, random_state=42, verbose=0, warm_start=False)
rf.fit(x_train_sc, y_train)
filename = 'RF_model.sav'
pickle.dump(rf, open(filename, 'wb'))
```

Once it’s done, we can get the scores for training data and testing data using the model.

Training Data

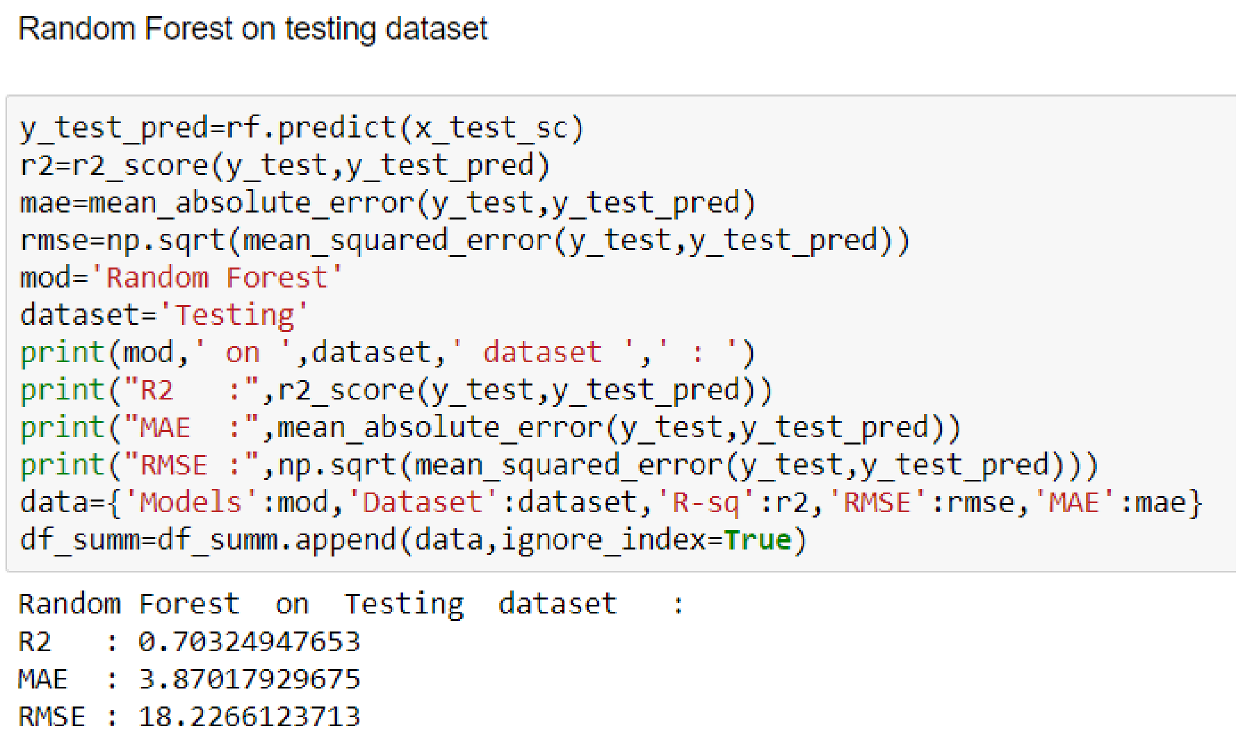


Random Forest on training Dataset

```
: y_train_pred=rf.predict(x_train_sc)
  r2=r2_score(y_train,y_train_pred)
  mae=mean_absolute_error(y_train,y_train_pred)
  rmse=np.sqrt(mean_squared_error(y_train,y_train_pred))
  mod='Random Forest'
  dataset='Training'
  print(mod,' on ',dataset,' dataset ',' : ')
  print("R2   :",r2_score(y_train,y_train_pred))
  print("MAE  :",mean_absolute_error(y_train,y_train_pred))
  print("RMSE :",np.sqrt(mean_squared_error(y_train,y_train_pred)))
  data={'Models':mod,'Dataset':dataset,'R-sq':r2,'RMSE':rmse,'MAE':mae}
  df_summ=df_summ.append(data,ignore_index=True)
```

```
Random Forest  on  Training  dataset   :
R2   : 0.831690673277
MAE  : 3.41393820501
RMSE : 14.3110917232
```

Testing Data



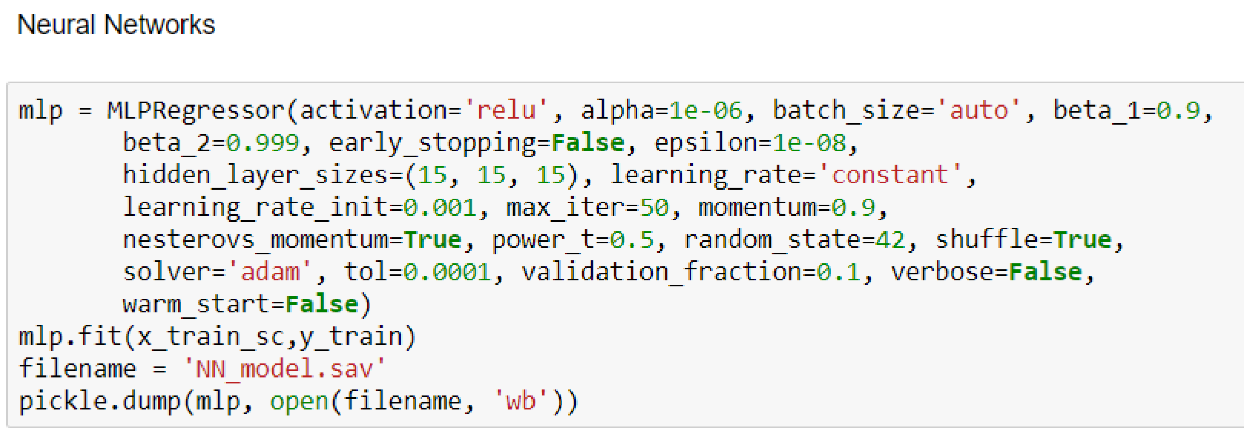
Random Forest on testing dataset

```
y_test_pred=rf.predict(x_test_sc)
r2=r2_score(y_test,y_test_pred)
mae=mean_absolute_error(y_test,y_test_pred)
rmse=np.sqrt(mean_squared_error(y_test,y_test_pred))
mod='Random Forest'
dataset='Testing'
print(mod,' on ',dataset,' dataset ',' : ')
print("R2   :",r2_score(y_test,y_test_pred))
print("MAE  :",mean_absolute_error(y_test,y_test_pred))
print("RMSE :",np.sqrt(mean_squared_error(y_test,y_test_pred)))
data={'Models':mod,'Dataset':dataset,'R-sq':r2,'RMSE':rmse,'MAE':mae}
df_summ=df_summ.append(data,ignore_index=True)
```

```
Random Forest  on  Testing  dataset   :
R2   : 0.70324947653
MAE  : 3.87017929675
RMSE : 18.2266123713
```

Neural Network

We must run the below code to run the model and save it in pickle file.

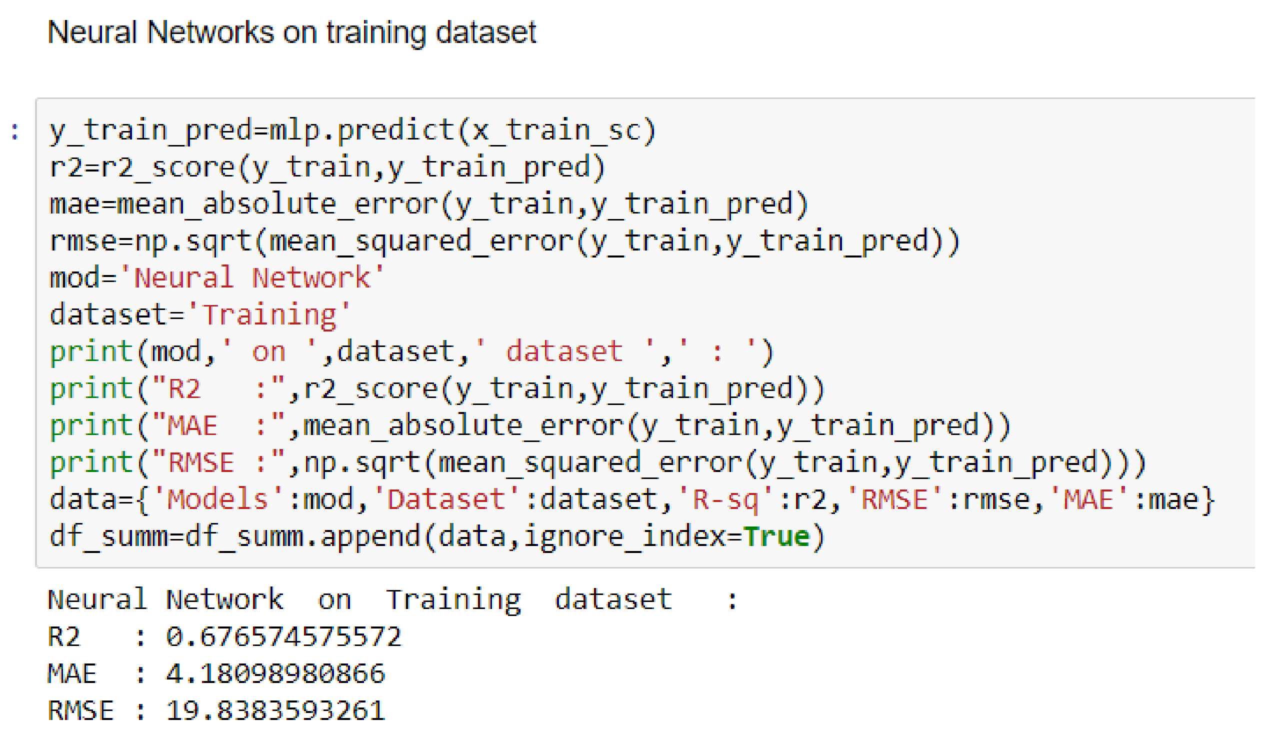


Neural Networks

```
mlp = MLPRegressor(activation='relu', alpha=1e-06, batch_size='auto', beta_1=0.9,
       beta_2=0.999, early_stopping=False, epsilon=1e-08,
       hidden_layer_sizes=(15, 15, 15), learning_rate='constant',
       learning_rate_init=0.001, max_iter=50, momentum=0.9,
       nesterovs_momentum=True, power_t=0.5, random_state=42, shuffle=True,
       solver='adam', tol=0.0001, validation_fraction=0.1, verbose=False,
       warm_start=False)
mlp.fit(x_train_sc,y_train)
filename = 'NN_model.sav'
pickle.dump(mlp, open(filename, 'wb'))
```

Once it’s done, we can get the scores for training data and testing data using the model.

Training Data

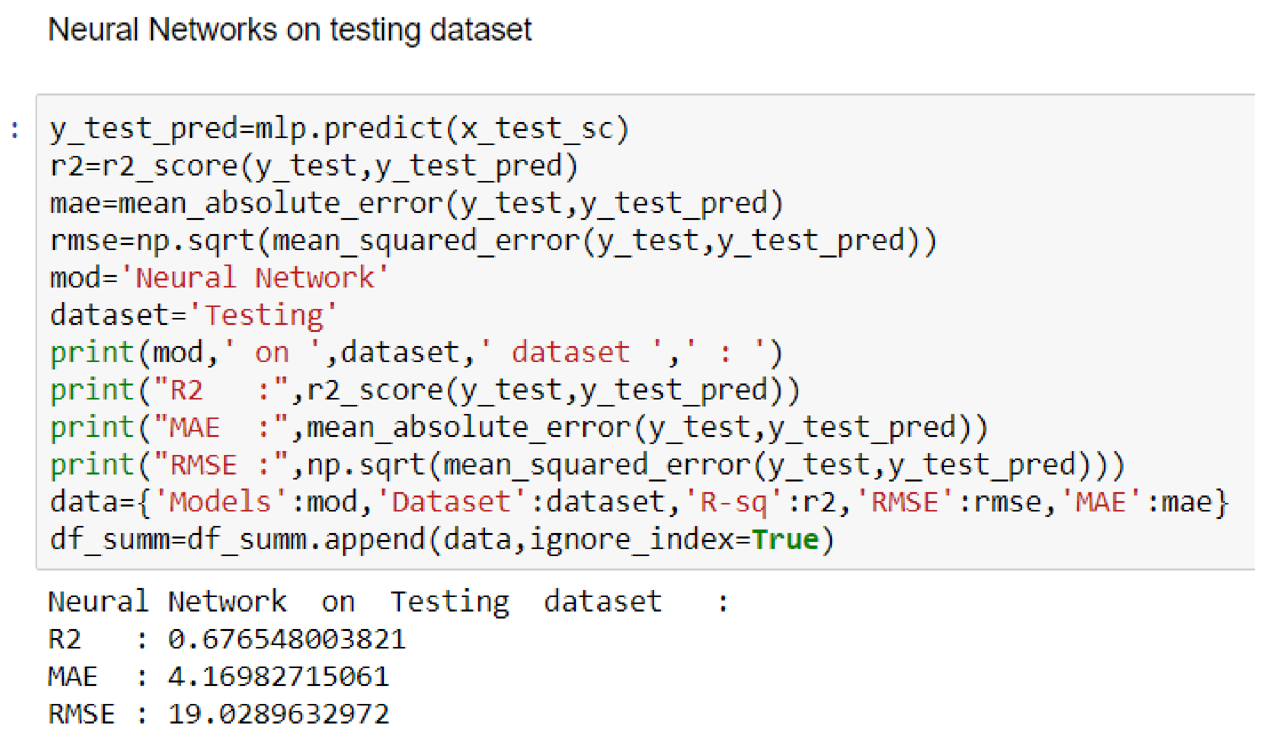


Neural Networks on training dataset

```
y_train_pred=mlp.predict(x_train_sc)
r2=r2_score(y_train,y_train_pred)
mae=mean_absolute_error(y_train,y_train_pred)
rmse=np.sqrt(mean_squared_error(y_train,y_train_pred))
mod='Neural Network'
dataset='Training'
print(mod,' on ',dataset,' dataset ',' : ')
print("R2   :",r2_score(y_train,y_train_pred))
print("MAE  :",mean_absolute_error(y_train,y_train_pred))
print("RMSE :",np.sqrt(mean_squared_error(y_train,y_train_pred)))
data={'Models':mod,'Dataset':dataset,'R-sq':r2,'RMSE':rmse,'MAE':mae}
df_summ=df_summ.append(data,ignore_index=True)
```

```
Neural Network  on  Training  dataset   :
R2   : 0.676574575572
MAE  : 4.18098980866
RMSE : 19.8383593261
```

Testing Data



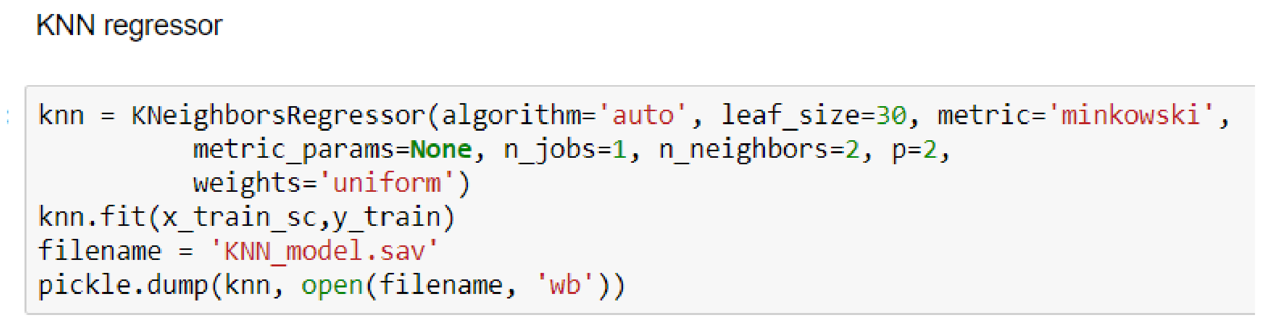
Neural Networks on testing dataset

```
y_test_pred=mlp.predict(x_test_sc)
r2=r2_score(y_test,y_test_pred)
mae=mean_absolute_error(y_test,y_test_pred)
rmse=np.sqrt(mean_squared_error(y_test,y_test_pred))
mod='Neural Network'
dataset='Testing'
print(mod,' on ',dataset,' dataset ',' : ')
print("R2   :",r2_score(y_test,y_test_pred))
print("MAE  :",mean_absolute_error(y_test,y_test_pred))
print("RMSE :",np.sqrt(mean_squared_error(y_test,y_test_pred)))
data={'Models':mod,'Dataset':dataset,'R-sq':r2,'RMSE':rmse,'MAE':mae}
df_summ=df_summ.append(data,ignore_index=True)
```

```
Neural Network  on  Testing  dataset   :
R2   : 0.676548003821
MAE  : 4.16982715061
RMSE : 19.0289632972
```

K-nearest Neighbor

We must run the below code to run the model and save it in pickle file.



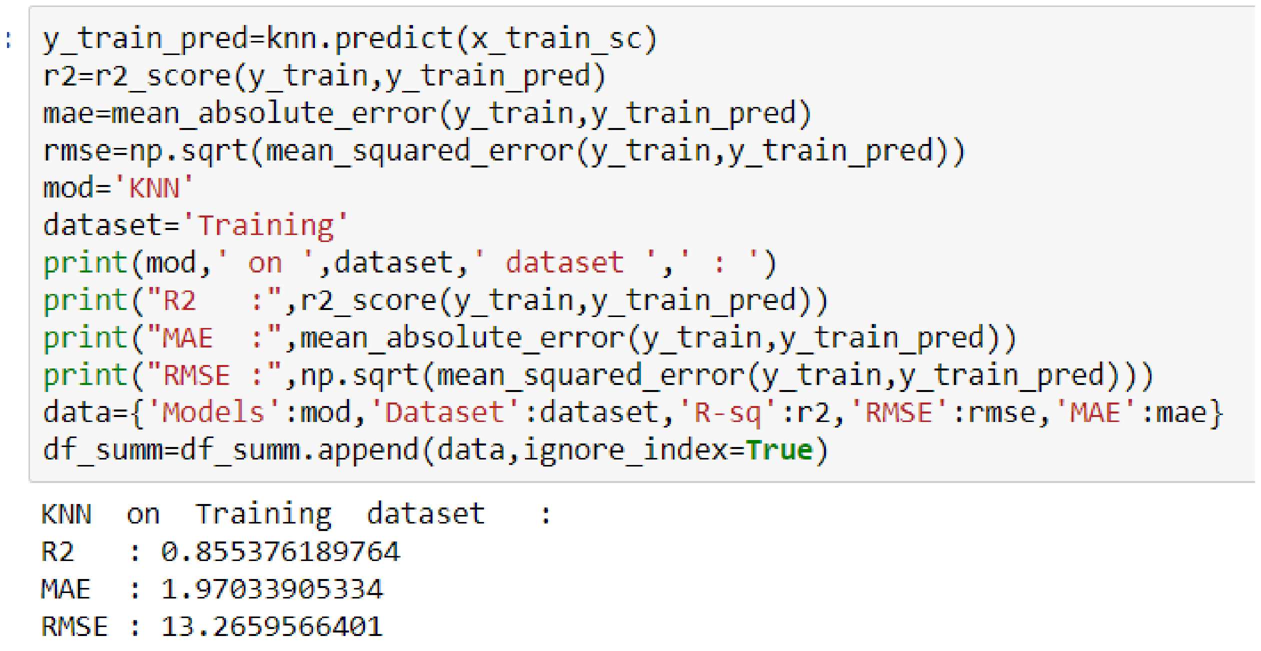
KNN regressor

```
knn = KNeighborsRegressor(algorithm='auto', leaf_size=30, metric='minkowski',
          metric_params=None, n_jobs=1, n_neighbors=2, p=2,
          weights='uniform')
knn.fit(x_train_sc,y_train)
filename = 'KNN_model.sav'
pickle.dump(knn, open(filename, 'wb'))
```

Once it's done, we can get the scores for training data and testing data using the model.

Training Data

KNN on training Dataset



```
y_train_pred=knn.predict(x_train_sc)
r2=r2_score(y_train,y_train_pred)
mae=mean_absolute_error(y_train,y_train_pred)
rmse=np.sqrt(mean_squared_error(y_train,y_train_pred))
mod='KNN'
dataset='Training'
print(mod,' on ',dataset,' dataset ',' : ')
print("R2   :",r2_score(y_train,y_train_pred))
print("MAE  :",mean_absolute_error(y_train,y_train_pred))
print("RMSE :",np.sqrt(mean_squared_error(y_train,y_train_pred)))
data={'Models':mod,'Dataset':dataset,'R-sq':r2,'RMSE':rmse,'MAE':mae}
df_summ=df_summ.append(data,ignore_index=True)
```

```
KNN  on  Training  dataset   :
R2   : 0.855376189764
MAE  : 1.97033905334
RMSE : 13.2659566401
```

Testing Data

KNN on testing dataset

```
y_test_pred=knn.predict(x_test_sc)
r2=r2_score(y_test,y_test_pred)
mae=mean_absolute_error(y_test,y_test_pred)
rmse=np.sqrt(mean_squared_error(y_test,y_test_pred))
mod='KNN'
dataset='Testing'
print(mod,' on ',dataset,' dataset ',' : ')
print("R2   :",r2_score(y_test,y_test_pred))
print("MAE  :",mean_absolute_error(y_test,y_test_pred))
print("RMSE :",np.sqrt(mean_squared_error(y_test,y_test_pred)))
data={'Models':mod,'Dataset':dataset,'R-sq':r2,'RMSE':rmse,'MAE':mae}
df_summ=df_summ.append(data,ignore_index=True)
```

```
KNN  on  Testing  dataset   :
R2   : 0.676548003821
MAE  : 4.16982715061
RMSE : 19.0289632972
```

Support Vector Regressor

We must run the below code to run the model and save it in pickle file.

SVR

```
svr = SVR(C=1.0, cache_size=200, coef0=0.0, degree=3, epsilon=0.1, gamma='auto',
  kernel='rbf', max_iter=-1, shrinking=True, tol=0.001, verbose=False)
svr.fit(x_train_sc,y_train)
filename = 'SVR_model.sav'
pickle.dump(svr, open(filename, 'wb'))
```

Once it’s done, we can get the scores for training data and testing data using the model.

Training Data

SVR on training dataset

```
y_train_pred=svr.predict(x_train_sc)
r2=r2_score(y_train,y_train_pred)
mae=mean_absolute_error(y_train,y_train_pred)
rmse=np.sqrt(mean_squared_error(y_train,y_train_pred))
mod='SVR'
dataset='Training'
print(mod,' on ',dataset,' dataset ',' : ')
print("R2   :",r2_score(y_train,y_train_pred))
print("MAE  :",mean_absolute_error(y_train,y_train_pred))
print("RMSE :",np.sqrt(mean_squared_error(y_train,y_train_pred)))
data={'Models':mod,'Dataset':dataset,'R-sq':r2,'RMSE':rmse,'MAE':mae}
df_summ=df_summ.append(data,ignore_index=True)
```

```
SVR  on  Training  dataset   :
R2   : 0.855376189764
MAE  : 1.97033905334
RMSE : 13.2659566401
```

Testing Data

SVR on testing dataset

```
y_test_pred=svr.predict(x_test_sc)
r2=r2_score(y_test,y_test_pred)
mae=mean_absolute_error(y_test,y_test_pred)
rmse=np.sqrt(mean_squared_error(y_test,y_test_pred))
mod='SVR'
dataset='Testing'
print(mod,' on ',dataset,' dataset ',' : ')
print("R2   :",r2_score(y_test,y_test_pred))
print("MAE  :",mean_absolute_error(y_test,y_test_pred))
print("RMSE :",np.sqrt(mean_squared_error(y_test,y_test_pred)))
data={'Models':mod,'Dataset':dataset,'R-sq':r2,'RMSE':rmse,'MAE':mae}
df_summ=df_summ.append(data,ignore_index=True)
```

```
SVR  on  Testing  dataset   :
R2   : 0.676548003821
MAE  : 4.16982715061
RMSE : 19.0289632972
```

Summary Metrics

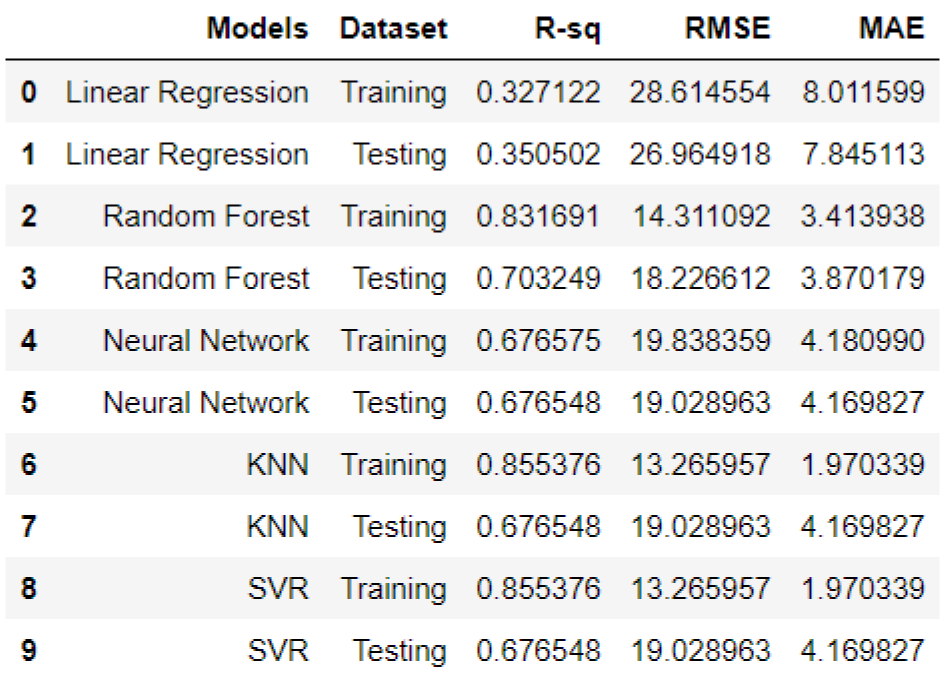
We must save the data received from model above into a file.

```
df_summ.to_csv('Summarry.csv',sep=',',index=False)
```

When we run the above code, we can save the data stored in the dataframe to a csv file.

We must see what the dataframe contains.

```
df_summ
```



	Models	Dataset	R-sq	RMSE	MAE
0	Linear Regression	Training	0.327122	28.614554	8.011599
1	Linear Regression	Testing	0.350502	26.964918	7.845113
2	Random Forest	Training	0.831691	14.311092	3.413938
3	Random Forest	Testing	0.703249	18.226612	3.870179
4	Neural Network	Training	0.676575	19.838359	4.180990
5	Neural Network	Testing	0.676548	19.028963	4.169827
6	KNN	Training	0.855376	13.265957	1.970339
7	KNN	Testing	0.676548	19.028963	4.169827
8	SVR	Training	0.855376	13.265957	1.970339
9	SVR	Testing	0.676548	19.028963	4.169827

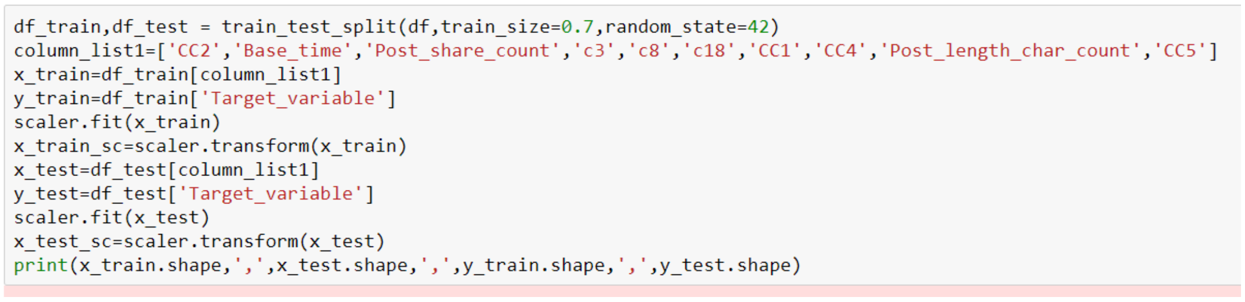
From the above table, we can clearly understand the results and get a clear view of which algorithm is better for our model.

Model Validation

Regularization

In mathematics, statistics, and computer science, particularly in the fields of machine learning and inverse problems, regularization is a process of introducing additional information in order to solve an ill-posed problem or to prevent overfitting.

First, we have make a model.



```
df_train,df_test = train_test_split(df,train_size=0.7,random_state=42)
column_list1=['CC2','Base_time','Post_share_count','c3','c8','c18','CC1','CC4','Post_length_char_count','CC5']
x_train=df_train[column_list1]
y_train=df_train['Target_variable']
scaler.fit(x_train)
x_train_sc=scaler.transform(x_train)
x_test=df_test[column_list1]
y_test=df_test['Target_variable']
scaler.fit(x_test)
x_test_sc=scaler.transform(x_test)
print(x_train.shape,',',x_test.shape,',',y_train.shape,',',y_test.shape)
```

There are three types of regularization.

- Lasso(L1)
- Ridge(L2)
- Elastic Net (L1+L2)

There are 2 major methods of solving regularization.

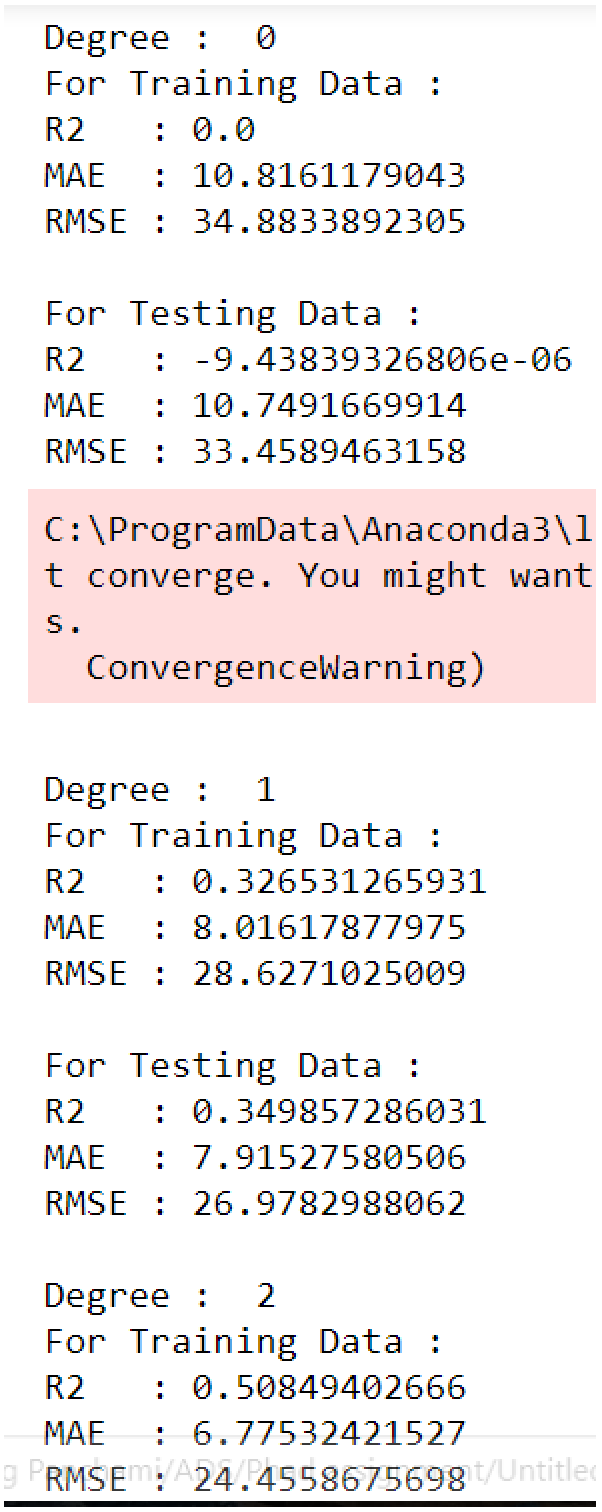
- Without using optimized ‘alpha’ value.
- Using optimized ‘alpha’ value.

Without using optimized ‘alpha’ value

Lasso(L1)

```python
for i in range(0, 4):
    model = PolynomialFeatures(degree=i)
    x_train_ = model.fit_transform(x_train)
    x_test_ = model.fit_transform(x_test)

    l1reg.fit(x_train_, y_train)
    train_pred_l1 = l1reg.predict(x_train_)
    test_pred_l1 = l1reg.predict(x_test_)
    l1reg_test_mse_list.append(mean_squared_error(y_test, test_pred_l1))
    print("\nDegree : ",i)
    print("For Training Data : ")
    print("R2   :",r2_score(y_train,train_pred_l1))
    print("MAE  :",mean_absolute_error(y_train,train_pred_l1))
    print("RMSE :",np.sqrt(mean_squared_error(y_train,train_pred_l1)))

    print("\nFor Testing Data : ")
    print("R2   :",r2_score(y_test,test_pred_l1))
    print("MAE  :",mean_absolute_error(y_test,test_pred_l1))
    print("RMSE :",np.sqrt(mean_squared_error(y_test,test_pred_l1)))


plt.xlabel('degree of polynomial')
plt.ylabel('MSE')
plt.grid(True)
plt.title('Polynomial Regression without L1(Lasso) Regularization')
plt.plot(degree_of_polynomial, test_mse_list, '-b', degree_of_polynomial, l1reg_test_mse_list, '--r')
plt.show()
```

When we run the above code, we get the following output.

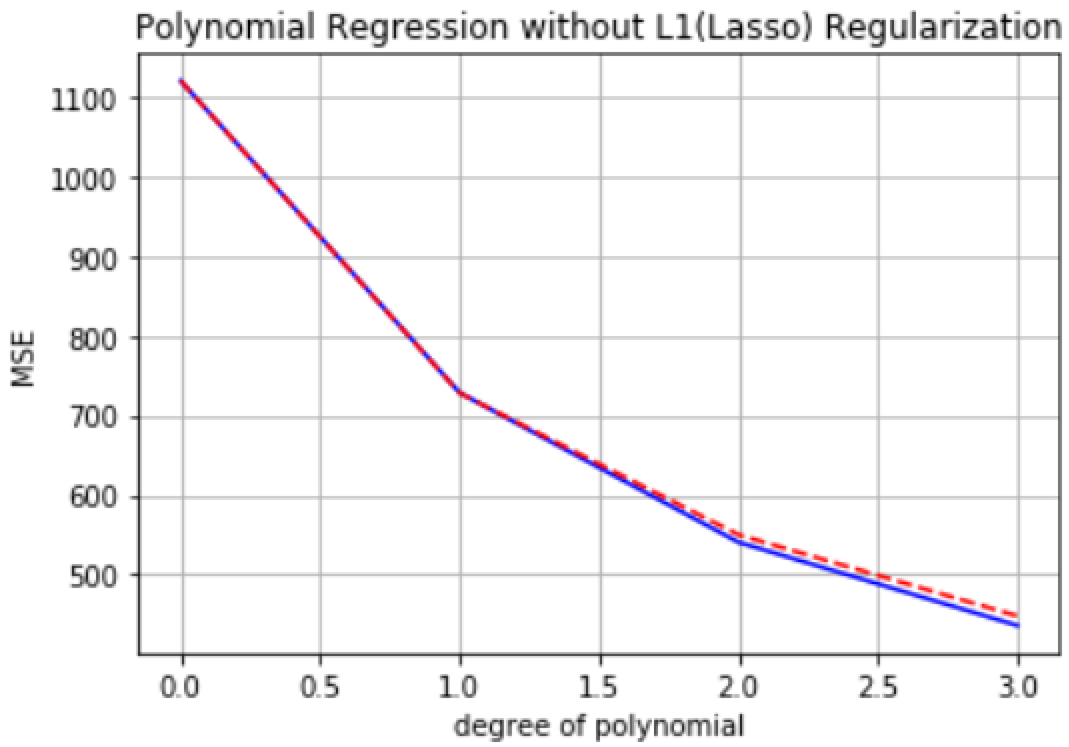


```
Degree :  0
For Training Data :
R2   : 0.0
MAE  : 10.8161179043
RMSE : 34.8833892305

For Testing Data :
R2   : -9.43839326806e-06
MAE  : 10.7491669914
RMSE : 33.4589463158
```

```
C:\ProgramData\Anaconda3\l
t converge. You might want
s.
  ConvergenceWarning)
```

```
Degree :  1
For Training Data :
R2   : 0.326531265931
MAE  : 8.01617877975
RMSE : 28.6271025009

For Testing Data :
R2   : 0.349857286031
MAE  : 7.91527580506
RMSE : 26.9782988062

Degree :  2
For Training Data :
R2   : 0.50849402666
MAE  : 6.77532421527
RMSE : 24.4558675698
```

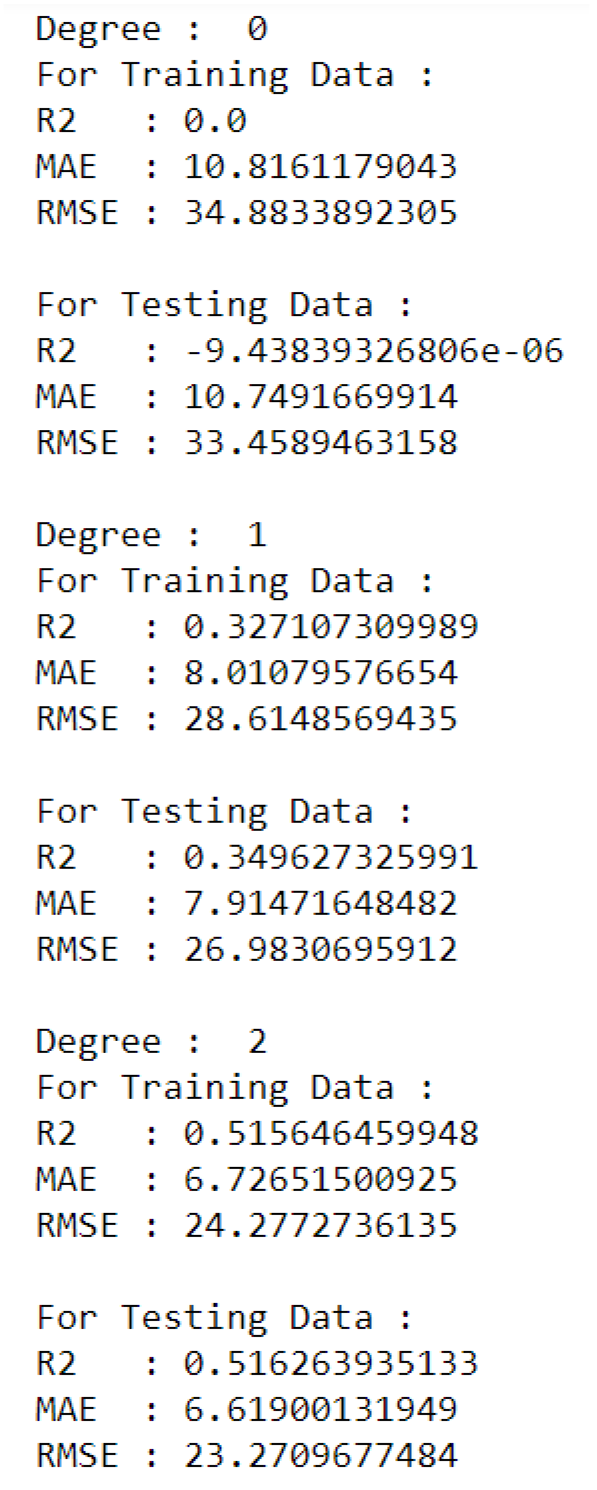
And we get the following graph.


Polynomial Regression without L1(Lasso) Regularization
MSE
1100
1000
900
800
700
600
500
0.0
0.5
1.0
1.5
2.0
2.5
3.0
degree of polynomial

Ridge(L2)

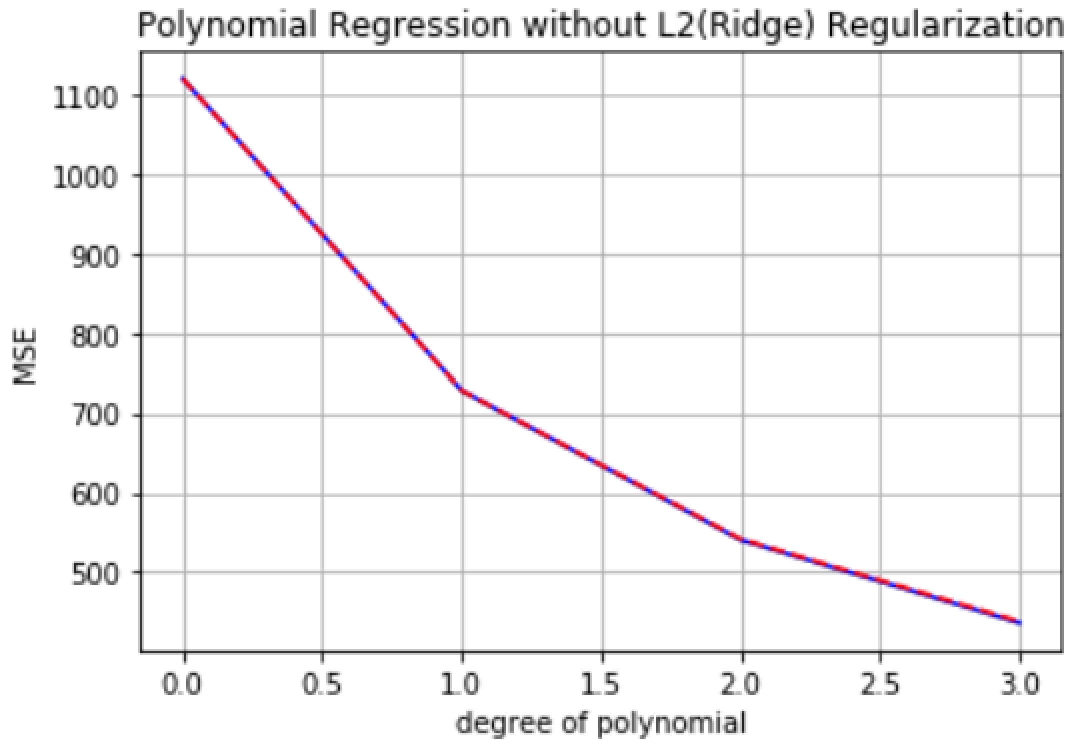
```
for i in range(0, 4):
    model = PolynomialFeatures(degree=i)
    x_train_ = model.fit_transform(x_train)
    x_test_ = model.fit_transform(x_test)

    l2reg.fit(x_train_, y_train)
    train_pred_l2 = l2reg.predict(x_train_)
    test_pred_l2 = l2reg.predict(x_test_)
    l2reg_test_mse_list.append(mean_squared_error(y_test, test_pred_l2))
    print("\nDegree : ",i)
    print("For Training Data : ")
    print("R2   :",r2_score(y_train,train_pred_l2))
    print("MAE  :",mean_absolute_error(y_train,train_pred_l2))
    print("RMSE :",np.sqrt(mean_squared_error(y_train,train_pred_l2)))

    print("\nFor Testing Data : ")
    print("R2   :",r2_score(y_test,test_pred_l2))
    print("MAE  :",mean_absolute_error(y_test,test_pred_l2))
    print("RMSE :",np.sqrt(mean_squared_error(y_test,test_pred_l2)))

plt.xlabel('degree of polynomial')
plt.ylabel('MSE')
plt.grid(True)
plt.title('Polynomial Regression without L2(Ridge) Regularization')
plt.plot(degree_of_polynomial, test_mse_list, '-b', degree_of_polynomial, l2reg_test_mse_list, '--r')
plt.show()
```

When we run the above code, we get the following output.

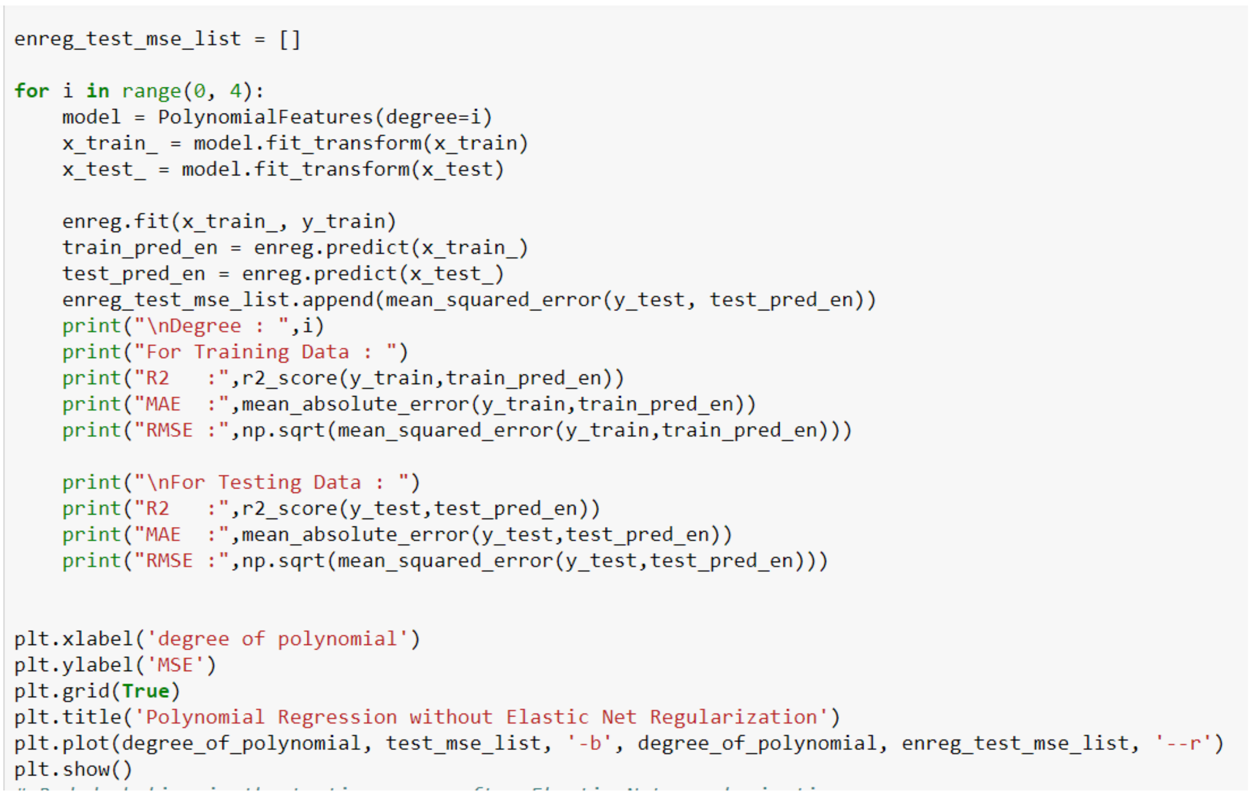


```
Degree :  0
For Training Data :
R2   : 0.0
MAE  : 10.8161179043
RMSE : 34.8833892305

For Testing Data :
R2   : -9.43839326806e-06
MAE  : 10.7491669914
RMSE : 33.4589463158

Degree :  1
For Training Data :
R2   : 0.327107309989
MAE  : 8.01079576654
RMSE : 28.6148569435

For Testing Data :
R2   : 0.349627325991
MAE  : 7.91471648482
RMSE : 26.9830695912

Degree :  2
For Training Data :
R2   : 0.515646459948
MAE  : 6.72651500925
RMSE : 24.2772736135

For Testing Data :
R2   : 0.516263935133
MAE  : 6.61900131949
RMSE : 23.2709677484
```

And we get the following graph.

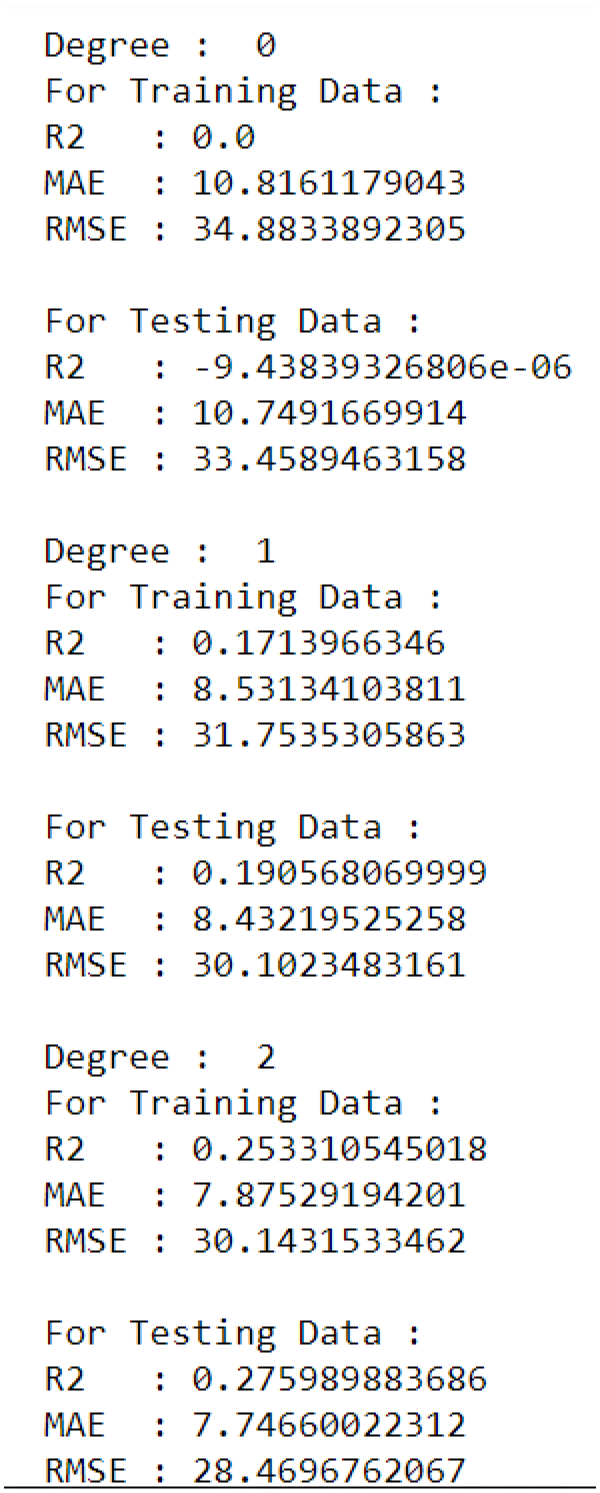


Elastic Net(L1+L2)



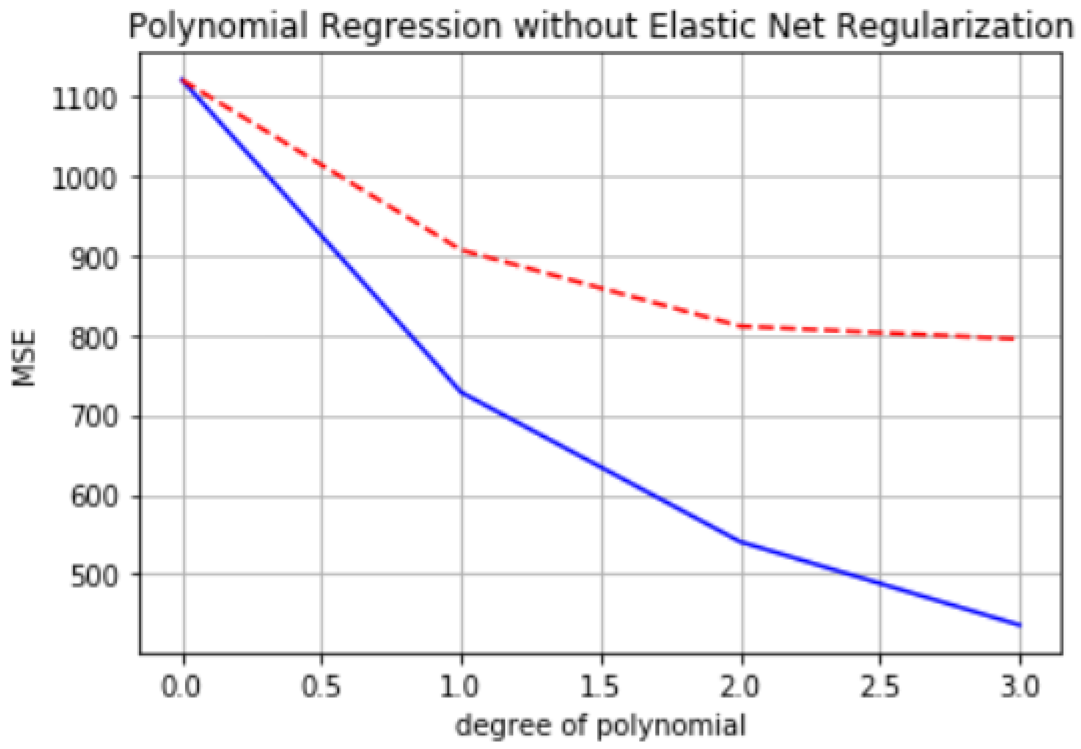
```python
enreg_test_mse_list = []

for i in range(0, 4):
    model = PolynomialFeatures(degree=i)
    x_train_ = model.fit_transform(x_train)
    x_test_ = model.fit_transform(x_test)

    enreg.fit(x_train_, y_train)
    train_pred_en = enreg.predict(x_train_)
    test_pred_en = enreg.predict(x_test_)
    enreg_test_mse_list.append(mean_squared_error(y_test, test_pred_en))
    print("\nDegree : ",i)
    print("For Training Data : ")
    print("R2   :",r2_score(y_train,train_pred_en))
    print("MAE  :",mean_absolute_error(y_train,train_pred_en))
    print("RMSE :",np.sqrt(mean_squared_error(y_train,train_pred_en)))

    print("\nFor Testing Data : ")
    print("R2   :",r2_score(y_test,test_pred_en))
    print("MAE  :",mean_absolute_error(y_test,test_pred_en))
    print("RMSE :",np.sqrt(mean_squared_error(y_test,test_pred_en)))


plt.xlabel('degree of polynomial')
plt.ylabel('MSE')
plt.grid(True)
plt.title('Polynomial Regression without Elastic Net Regularization')
plt.plot(degree_of_polynomial, test_mse_list, '-b', degree_of_polynomial, enreg_test_mse_list, '--r')
plt.show()
```

When we run the above code, we get the following output.



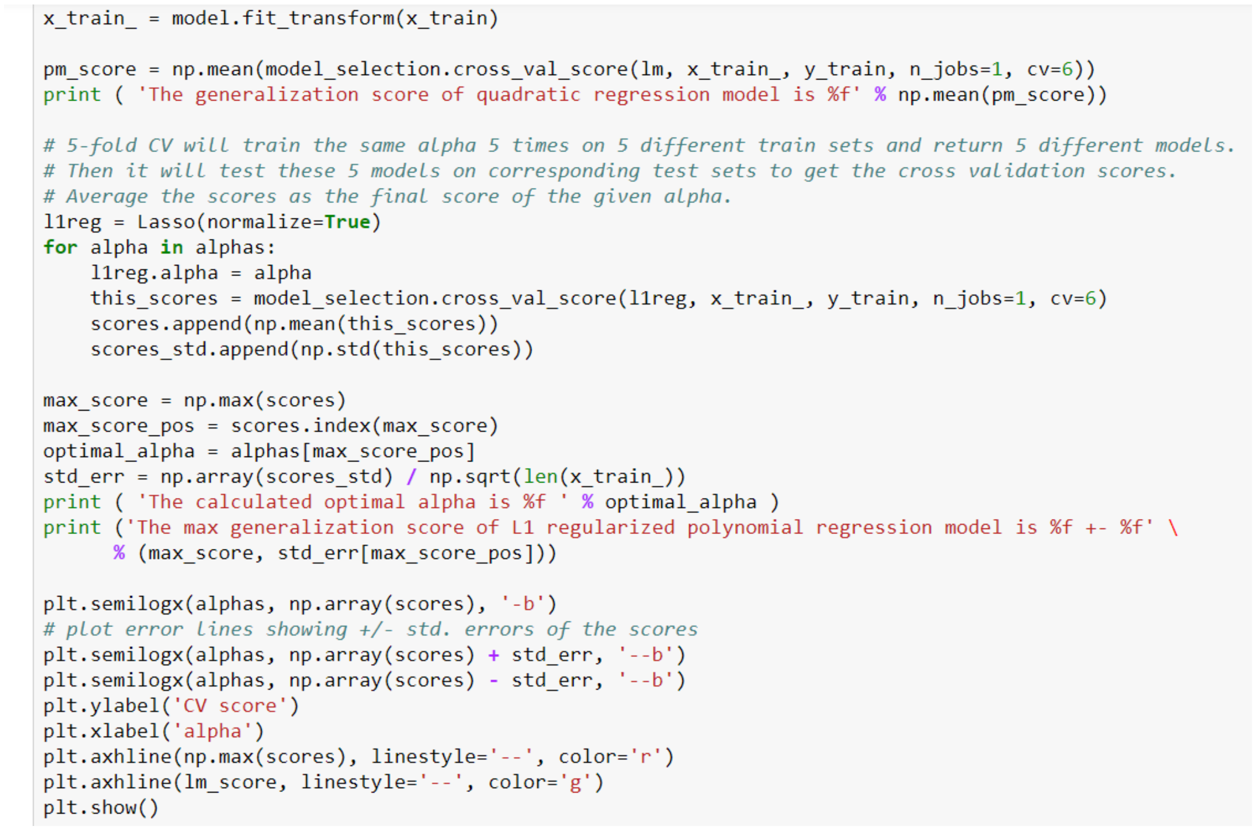
```
Degree :  0
For Training Data :
R2   : 0.0
MAE  : 10.8161179043
RMSE : 34.8833892305

For Testing Data :
R2   : -9.43839326806e-06
MAE  : 10.7491669914
RMSE : 33.4589463158

Degree :  1
For Training Data :
R2   : 0.1713966346
MAE  : 8.53134103811
RMSE : 31.7535305863

For Testing Data :
R2   : 0.190568069999
MAE  : 8.43219525258
RMSE : 30.1023483161

Degree :  2
For Training Data :
R2   : 0.253310545018
MAE  : 7.87529194201
RMSE : 30.1431533462

For Testing Data :
R2   : 0.275989883686
MAE  : 7.74660022312
RMSE : 28.4696762067
```

And we get the following graph.



Using optimized ‘alpha’ value

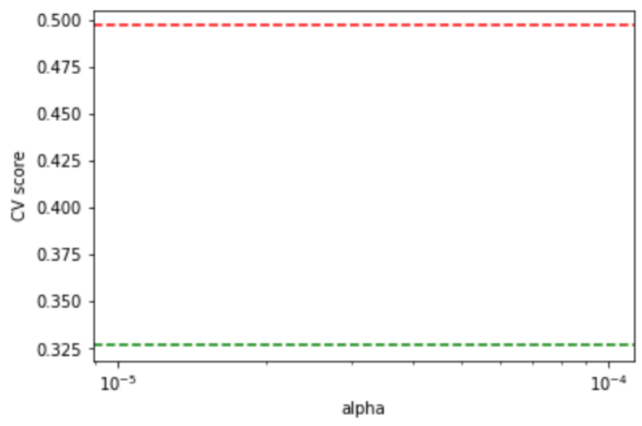
To achieve optimized alpha value, we have to run the code.



```python
x_train_ = model.fit_transform(x_train)

pm_score = np.mean(model_selection.cross_val_score(lm, x_train_, y_train, n_jobs=1, cv=6))
print ( 'The generalization score of quadratic regression model is %f' % np.mean(pm_score))

# 5-fold CV will train the same alpha 5 times on 5 different train sets and return 5 different models.
# Then it will test these 5 models on corresponding test sets to get the cross validation scores.
# Average the scores as the final score of the given alpha.
l1reg = Lasso(normalize=True)
for alpha in alphas:
    l1reg.alpha = alpha
    this_scores = model_selection.cross_val_score(l1reg, x_train_, y_train, n_jobs=1, cv=6)
    scores.append(np.mean(this_scores))
    scores_std.append(np.std(this_scores))

max_score = np.max(scores)
max_score_pos = scores.index(max_score)
optimal_alpha = alphas[max_score_pos]
std_err = np.array(scores_std) / np.sqrt(len(x_train_))
print ( 'The calculated optimal alpha is %f ' % optimal_alpha )
print ('The max generalization score of L1 regularized polynomial regression model is %f +- %f' \
      % (max_score, std_err[max_score_pos]))

plt.semilogx(alphas, np.array(scores), '-b')
# plot error lines showing +/- std. errors of the scores
plt.semilogx(alphas, np.array(scores) + std_err, '--b')
plt.semilogx(alphas, np.array(scores) - std_err, '--b')
plt.ylabel('CV score')
plt.xlabel('alpha')
plt.axhline(np.max(scores), linestyle='--', color='r')
plt.axhline(lm_score, linestyle='--', color='g')
plt.show()
```

And the output we got is below.

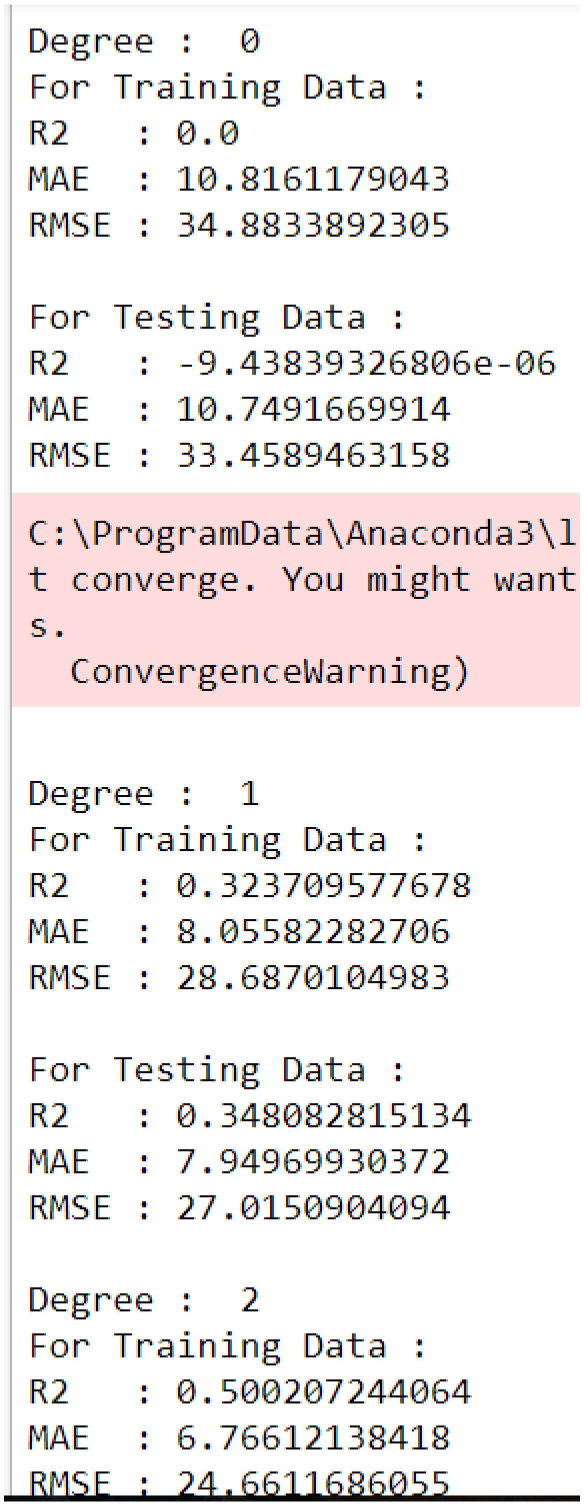
```
The calculated optimal alpha is 0.000050
The max generalization score of L1 regularized polynomial regression model is 0.496639 +- 0.000041
```



Lasso(L1)

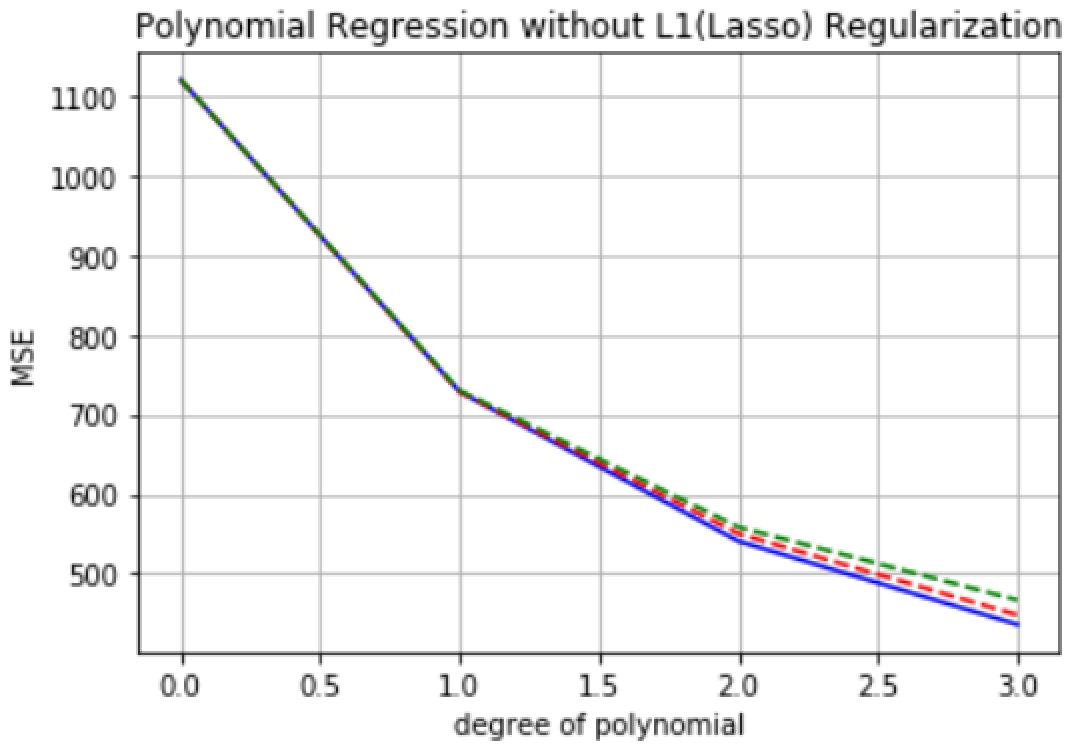
```
optimal_l1reg = Lasso(alpha=0.000050, normalize=True)

opt_l1reg_test_mse_list = []

for i in range(0, 4):
    model = PolynomialFeatures(degree=i)
    x_train_ = model.fit_transform(x_train)
    x_test_ = model.fit_transform(x_test)
    optimal_l1reg.fit(x_train_, y_train)
    train_opt_pred_l1 = optimal_l1reg.predict(x_train_)
    test_opt_pred_l1 = optimal_l1reg.predict(x_test_)
    opt_l1reg_test_mse_list.append(mean_squared_error(y_test, test_opt_pred_l1))
    print("\nDegree : ",i)
    print("For Training Data : ")
    print("R2   :",r2_score(y_train,train_opt_pred_l1))
    print("MAE  :",mean_absolute_error(y_train,train_opt_pred_l1))
    print("RMSE :",np.sqrt(mean_squared_error(y_train,train_opt_pred_l1)))

    print("\nFor Testing Data : ")
    print("R2   :",r2_score(y_test,test_opt_pred_l1))
    print("MAE  :",mean_absolute_error(y_test,test_opt_pred_l1))
    print("RMSE :",np.sqrt(mean_squared_error(y_test,test_opt_pred_l1)))


print ('MSE of linear regression model is %f' % test_mse_list[1])
print ('MSE of quadratic regression model is %f' % test_mse_list[2])
print ('MSE of optimal L1 regularized quadratic regression model is %f' % opt_l1reg_test_mse_list[2])


plt.xlabel('degree of polynomial')
plt.ylabel('MSE')
```

When we run the above code, we get the following output.



```
Degree :  0
For Training Data :
R2   : 0.0
MAE  : 10.8161179043
RMSE : 34.8833892305

For Testing Data :
R2   : -9.43839326806e-06
MAE  : 10.7491669914
RMSE : 33.4589463158
C:\ProgramData\Anaconda3\l
t converge. You might want
s.
  ConvergenceWarning)


Degree :  1
For Training Data :
R2   : 0.323709577678
MAE  : 8.05582282706
RMSE : 28.6870104983

For Testing Data :
R2   : 0.348082815134
MAE  : 7.94969930372
RMSE : 27.0150904094

Degree :  2
For Training Data :
R2   : 0.500207244064
MAE  : 6.76612138418
RMSE : 24.6611686055
```

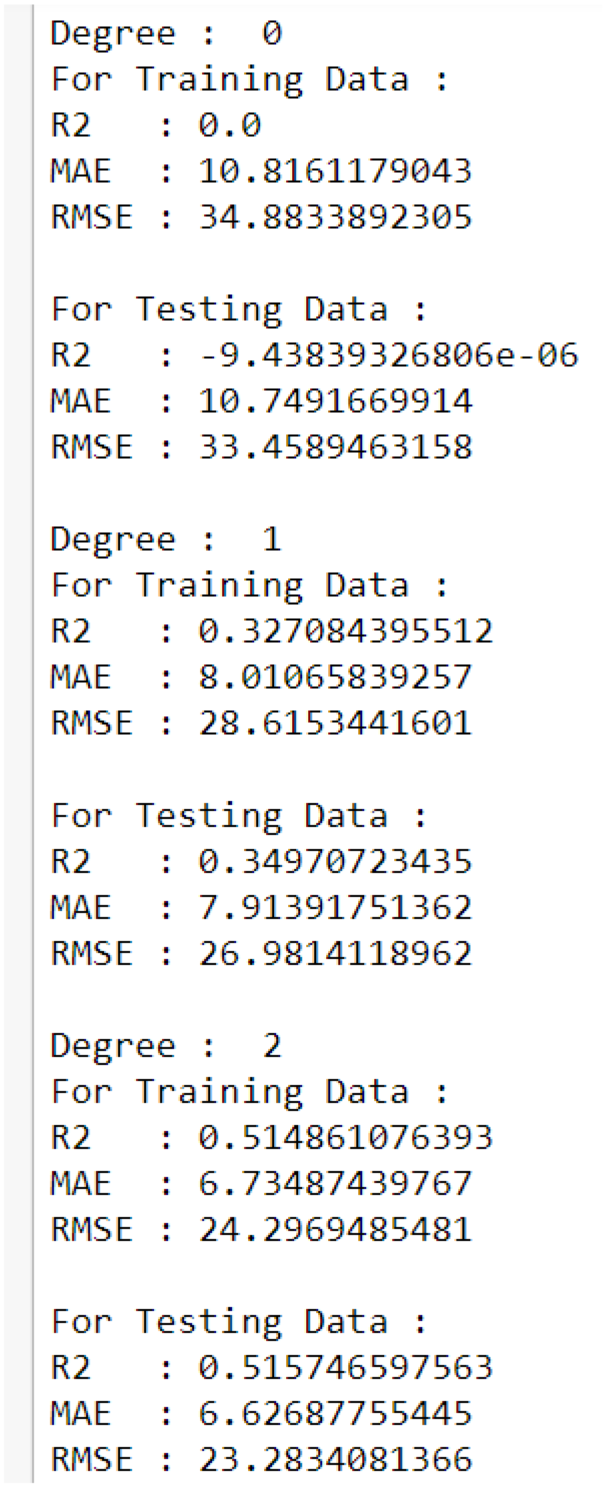
And we get the following graph.



Ridge(L2)

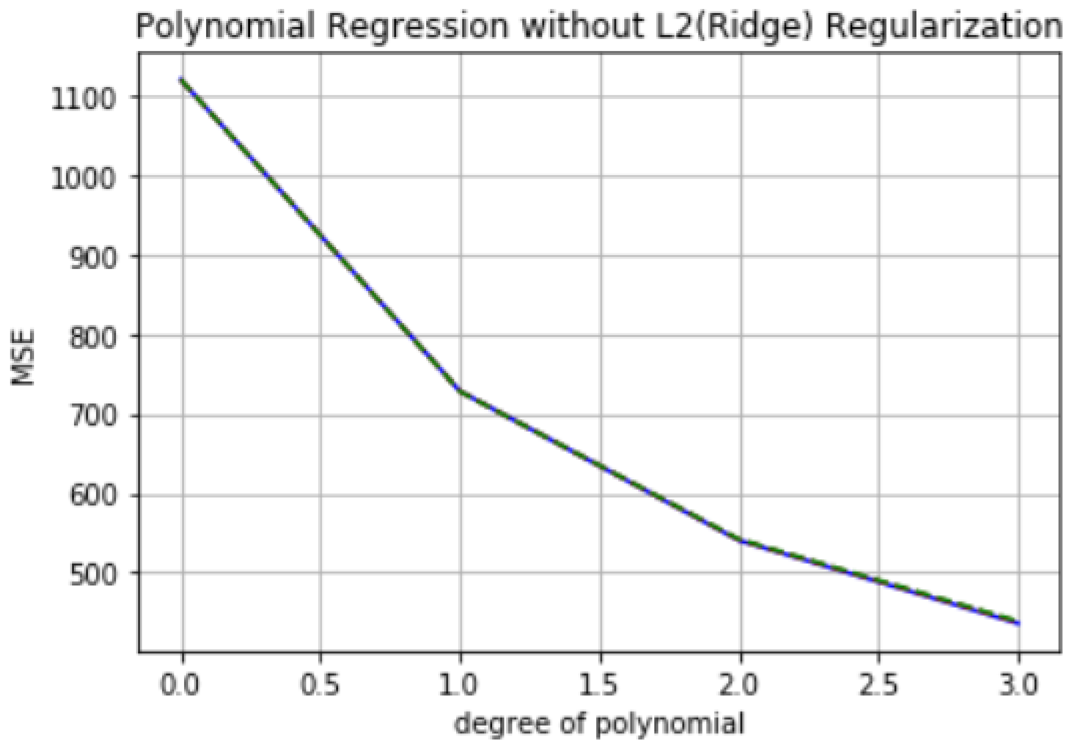
```
optimal_l2reg = Ridge(alpha=0.000050, normalize=True)

opt_l2reg_test_mse_list = []

for i in range(0, 4):
    model = PolynomialFeatures(degree=i)
    x_train_ = model.fit_transform(x_train)
    x_test_ = model.fit_transform(x_test)
    optimal_l2reg.fit(x_train_, y_train)
    train_opt_pred_l2 = optimal_l2reg.predict(x_train_)
    test_opt_pred_l2 = optimal_l2reg.predict(x_test_)
    opt_l2reg_test_mse_list.append(mean_squared_error(y_test, test_opt_pred_l2))
    print("\nDegree : ",i)
    print("For Training Data : ")
    print("R2   :",r2_score(y_train,train_opt_pred_l2))
    print("MAE  :",mean_absolute_error(y_train,train_opt_pred_l2))
    print("RMSE :",np.sqrt(mean_squared_error(y_train,train_opt_pred_l2)))

    print("\nFor Testing Data : ")
    print("R2   :",r2_score(y_test,test_opt_pred_l2))
    print("MAE  :",mean_absolute_error(y_test,test_opt_pred_l2))
    print("RMSE :",np.sqrt(mean_squared_error(y_test,test_opt_pred_l2)))


print ('MSE of linear regression model is %f' % test_mse_list[1])
print ('MSE of quadratic regression model is %f' % test_mse_list[2])
print ('MSE of optimal L2 regularized quadratic regression model is %f' % opt_l2reg_test_mse_list[2])


plt.xlabel('degree of polynomial')
plt.ylabel('MSE')
```

When we run the above code, we get the following output.

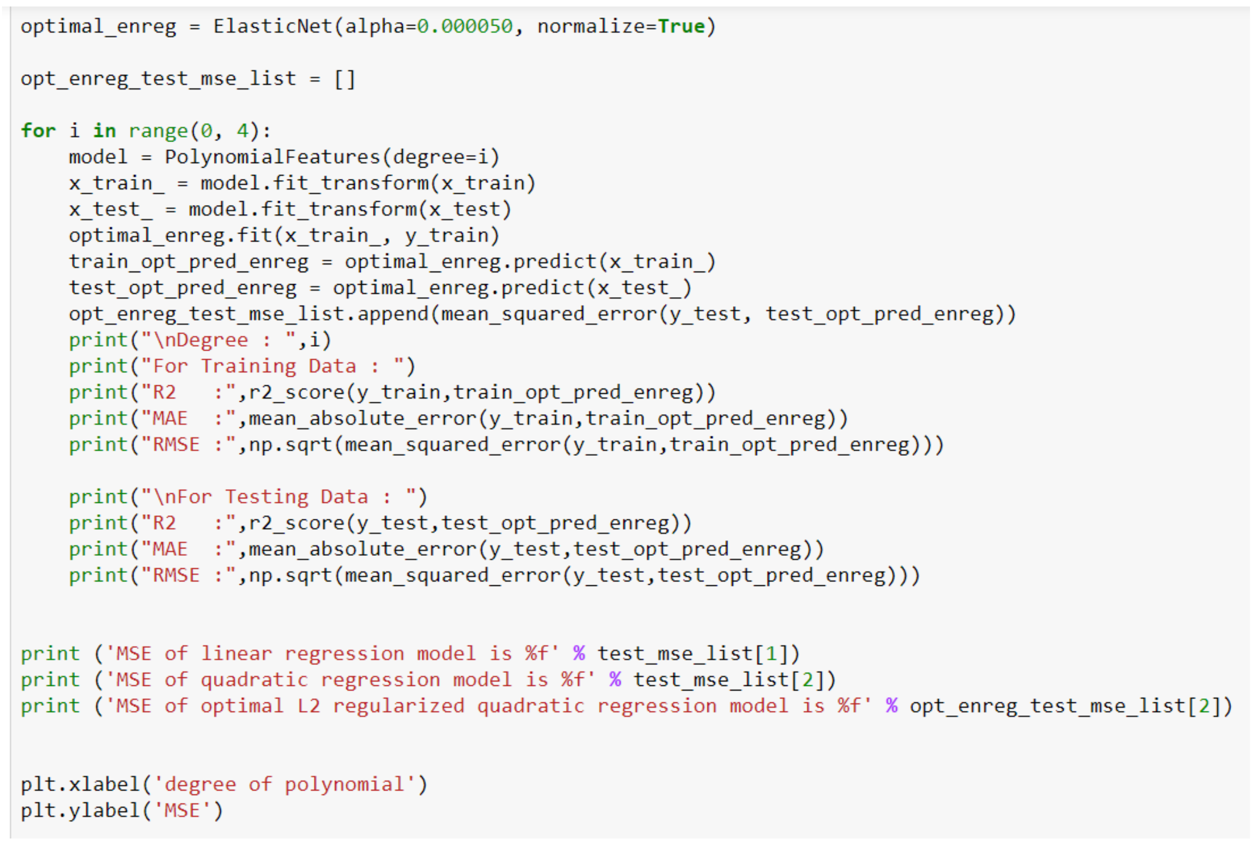


```
Degree :  0
For Training Data :
R2   : 0.0
MAE  : 10.8161179043
RMSE : 34.8833892305

For Testing Data :
R2   : -9.43839326806e-06
MAE  : 10.7491669914
RMSE : 33.4589463158

Degree :  1
For Training Data :
R2   : 0.327084395512
MAE  : 8.01065839257
RMSE : 28.6153441601

For Testing Data :
R2   : 0.34970723435
MAE  : 7.91391751362
RMSE : 26.9814118962

Degree :  2
For Training Data :
R2   : 0.514861076393
MAE  : 6.73487439767
RMSE : 24.2969485481

For Testing Data :
R2   : 0.515746597563
MAE  : 6.62687755445
RMSE : 23.2834081366
```

And we get the following graph.

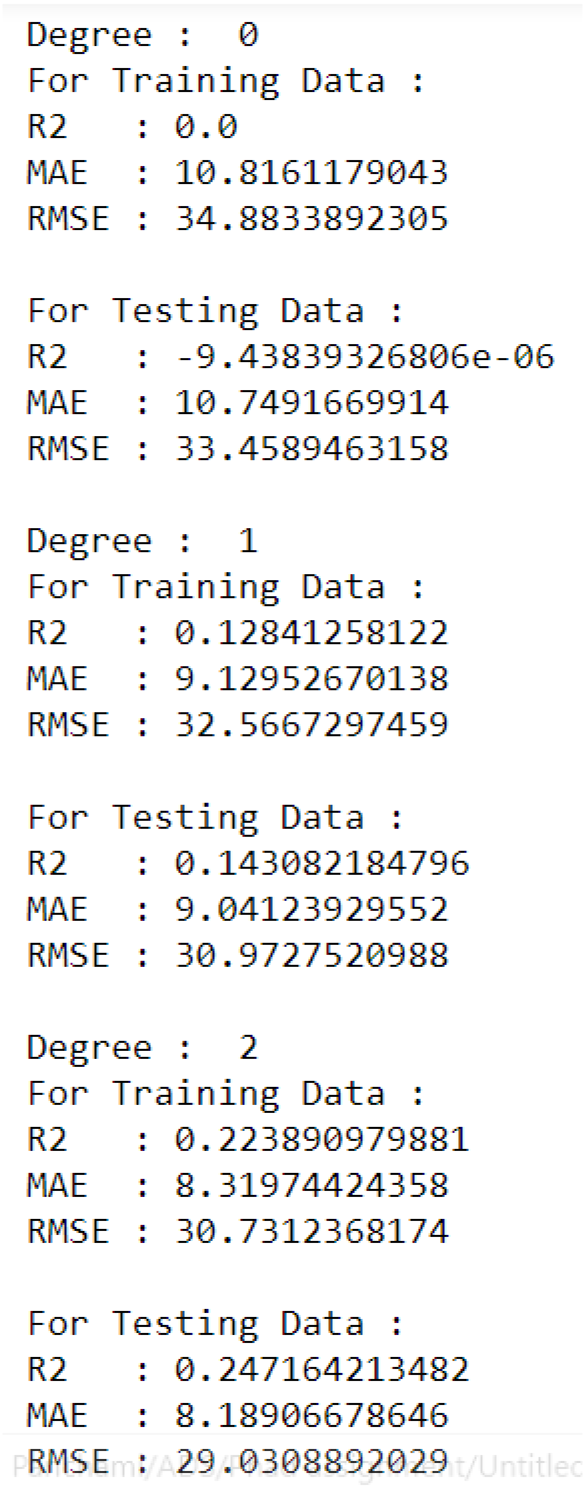


Elastic Net(L1+L2)



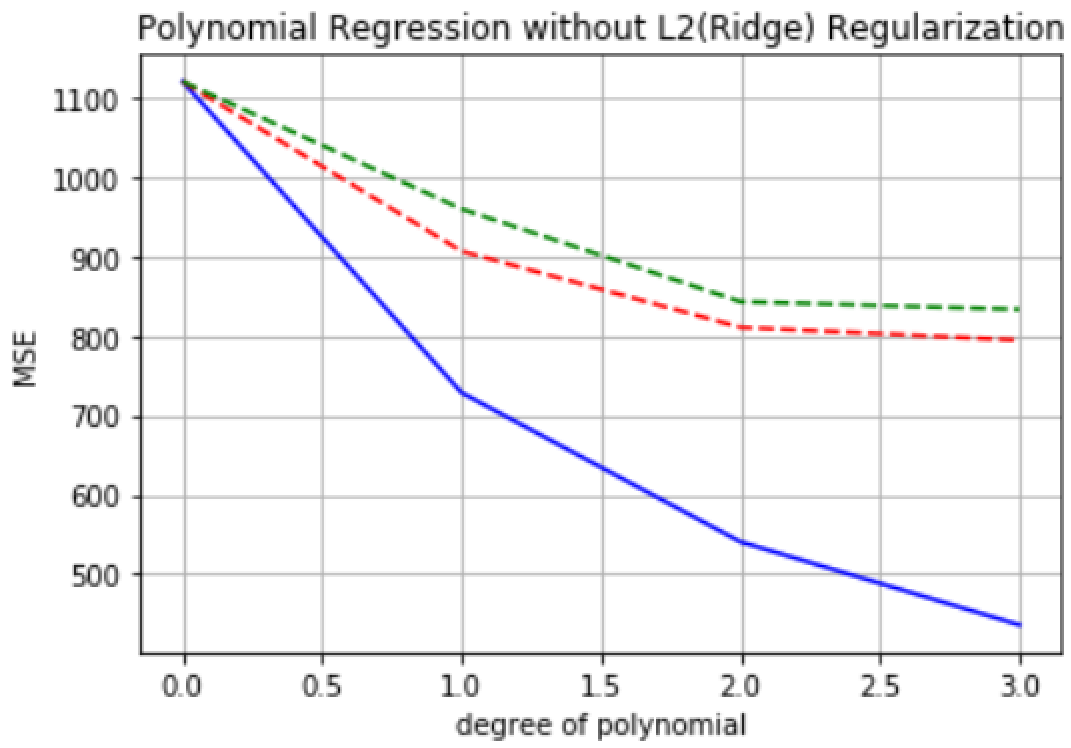
```python
optimal_enreg = ElasticNet(alpha=0.000050, normalize=True)

opt_enreg_test_mse_list = []

for i in range(0, 4):
    model = PolynomialFeatures(degree=i)
    x_train_ = model.fit_transform(x_train)
    x_test_ = model.fit_transform(x_test)
    optimal_enreg.fit(x_train_, y_train)
    train_opt_pred_enreg = optimal_enreg.predict(x_train_)
    test_opt_pred_enreg = optimal_enreg.predict(x_test_)
    opt_enreg_test_mse_list.append(mean_squared_error(y_test, test_opt_pred_enreg))
    print("\nDegree : ",i)
    print("For Training Data : ")
    print("R2   :",r2_score(y_train,train_opt_pred_enreg))
    print("MAE  :",mean_absolute_error(y_train,train_opt_pred_enreg))
    print("RMSE :",np.sqrt(mean_squared_error(y_train,train_opt_pred_enreg)))

    print("\nFor Testing Data : ")
    print("R2   :",r2_score(y_test,test_opt_pred_enreg))
    print("MAE  :",mean_absolute_error(y_test,test_opt_pred_enreg))
    print("RMSE :",np.sqrt(mean_squared_error(y_test,test_opt_pred_enreg)))


print ('MSE of linear regression model is %f' % test_mse_list[1])
print ('MSE of quadratic regression model is %f' % test_mse_list[2])
print ('MSE of optimal L2 regularized quadratic regression model is %f' % opt_enreg_test_mse_list[2])


plt.xlabel('degree of polynomial')
plt.ylabel('MSE')
```

When we run the above code, we get the following output.



```
Degree :  0
For Training Data :
R2   : 0.0
MAE  : 10.8161179043
RMSE : 34.8833892305

For Testing Data :
R2   : -9.43839326806e-06
MAE  : 10.7491669914
RMSE : 33.4589463158

Degree :  1
For Training Data :
R2   : 0.12841258122
MAE  : 9.12952670138
RMSE : 32.5667297459

For Testing Data :
R2   : 0.143082184796
MAE  : 9.04123929552
RMSE : 30.9727520988

Degree :  2
For Training Data :
R2   : 0.223890979881
MAE  : 8.31974424358
RMSE : 30.7312368174

For Testing Data :
R2   : 0.247164213482
MAE  : 8.18906678646
RMSE : 29.0308892029
```

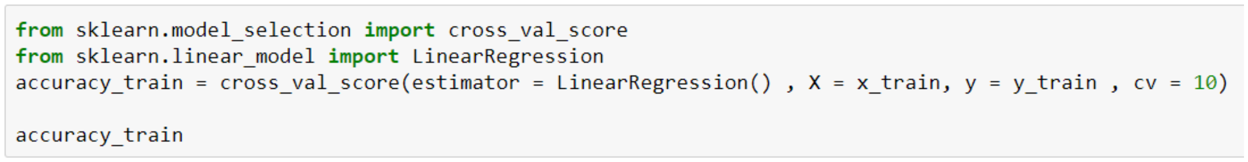
And we get the following graph.



Cross Validation

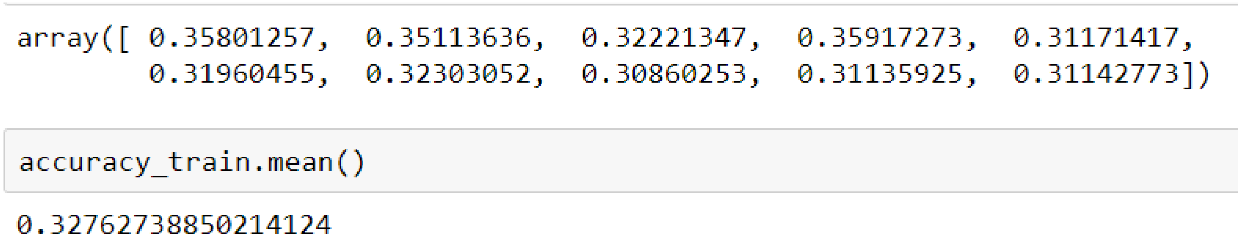
Cross-validation, sometimes called rotation estimation, is a model validation technique for assessing how the results of a statistical analysis will generalize to an independent data set. It is mainly used in settings where the goal is prediction, and one wants to estimate how accurately a predictive model will perform in practice. In a prediction problem, a model is usually given a dataset of known data on which training is run (training dataset), and a dataset of unknown data (or first seen data) against which the model is tested (called the validation dataset or testing set). The goal of cross validation is to define a dataset to "test" the model in the training phase (i.e., the validation set), in order to limit problems like overfitting[citation needed], give an insight on how the model will generalize to an independent dataset (i.e., an unknown dataset, for instance from a real problem), etc.

For Training data



```python
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LinearRegression
accuracy_train = cross_val_score(estimator = LinearRegression() , X = x_train, y = y_train , cv = 10)

accuracy_train
```

If we run the above code, we get the following output.

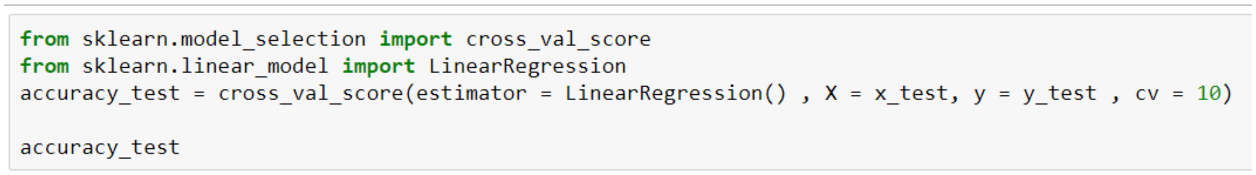


```
array([ 0.35801257,  0.35113636,  0.32221347,  0.35917273,  0.31171417,
        0.31960455,  0.32303052,  0.30860253,  0.31135925,  0.31142773])
```

```python
accuracy_train.mean()
```

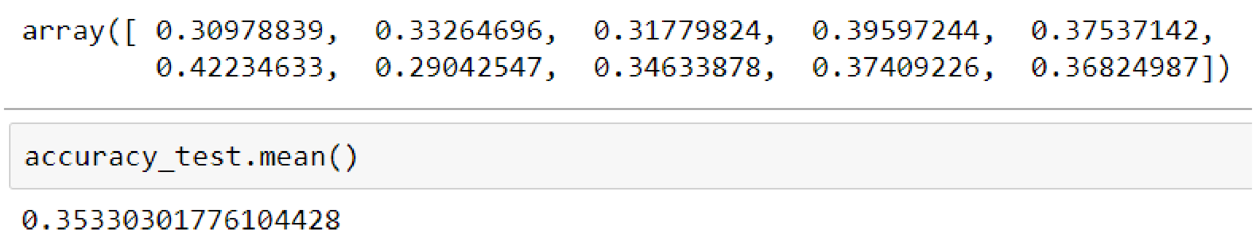
```
0.32762738850214124
```

For Testing data



```python
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LinearRegression
accuracy_test = cross_val_score(estimator = LinearRegression() , X = x_test, y = y_test , cv = 10)

accuracy_test
```

If we run the above code, we get the following output.



```
array([ 0.30978839,  0.33264696,  0.31779824,  0.39597244,  0.37537142,
        0.42234633,  0.29042547,  0.34633878,  0.37409226,  0.36824987])
```

```python
accuracy_test.mean()
```

```
0.35330301776104428
```

Bias Variance Tradeoff

In statistics and machine learning, the bias–variance tradeoff (or dilemma) is the problem of simultaneously minimizing two sources of error that prevent supervised learning algorithms from generalizing beyond their training set

- The bias is an error from erroneous assumptions in the learning algorithm. High bias can cause an algorithm to miss the relevant relations between features and target outputs (underfitting).
- The variance is an error from sensitivity to small fluctuations in the training set. High variance can cause an algorithm to model the random noise in the training data, rather than the intended outputs (overfitting).

The bias–variance decomposition is a way of analyzing a learning algorithm's expected generalization error with respect to a particular problem as a sum of three terms, the bias, variance, and a quantity called the irreducible error, resulting from noise in the problem itself.

This tradeoff applies to all forms of learning: classification, regression (function fitting) and structured output learning. It also has been invoked to explain the effectiveness of heuristics in human learning.

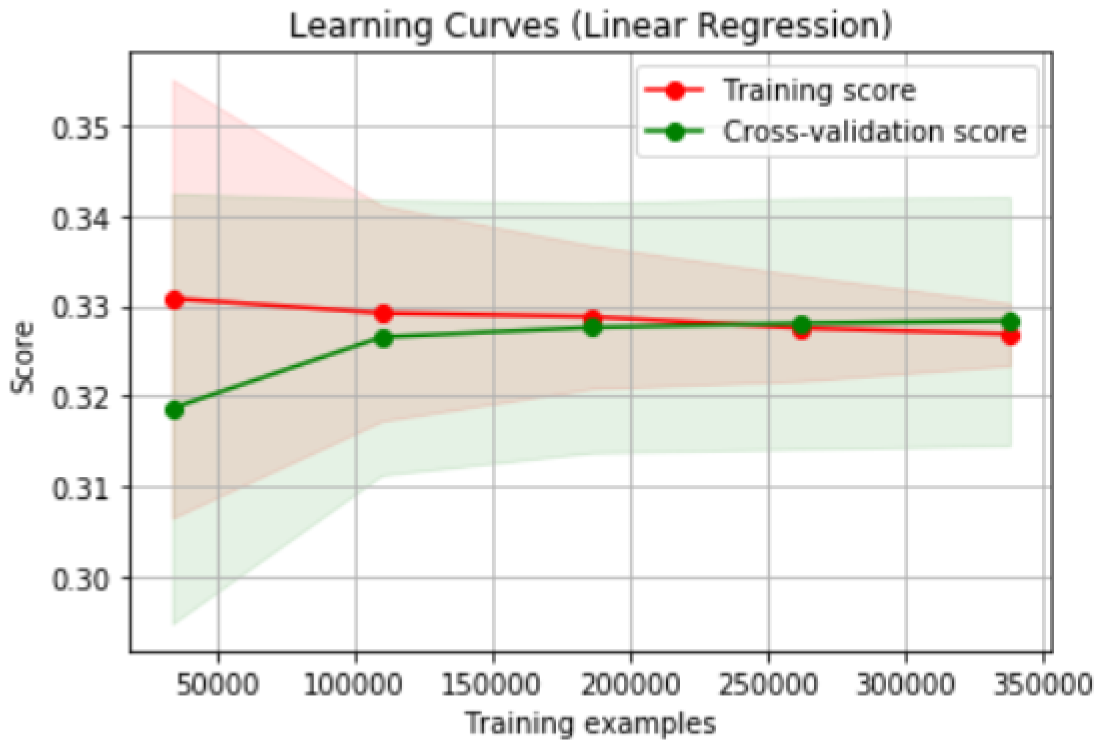
We must run the following code.

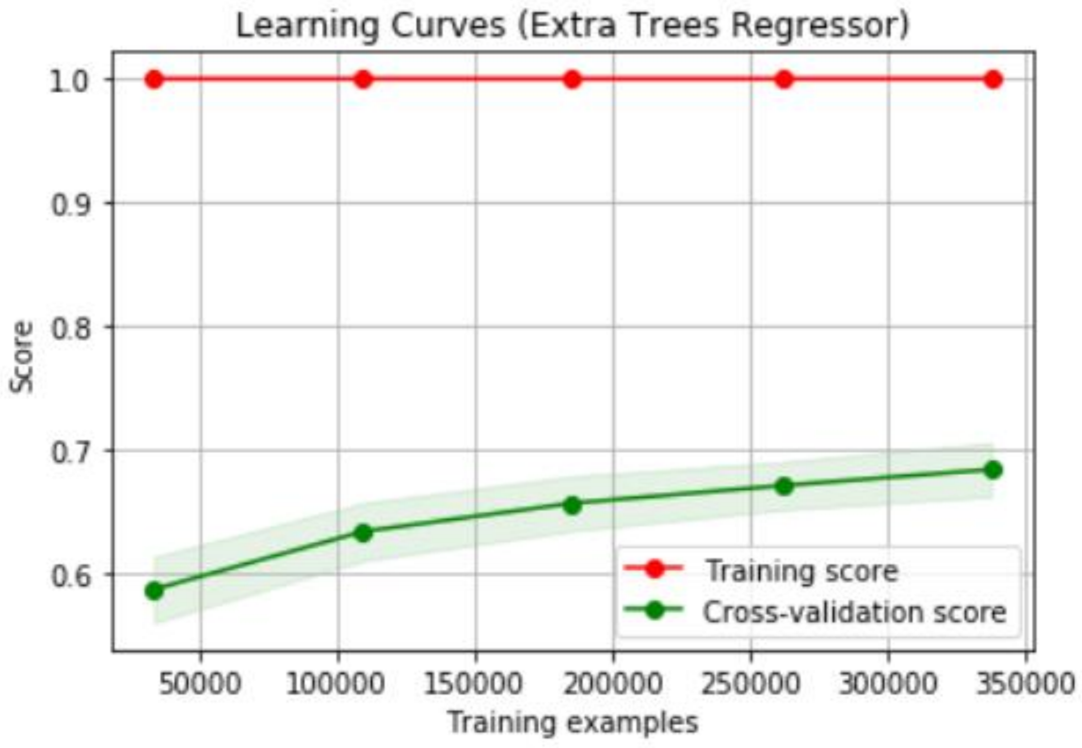
```
%%time
from sklearn.model_selection import learning_curve
from sklearn.model_selection import ShuffleSplit
from sklearn.ensemble import ExtraTreesRegressor,RandomForestRegressor


def plot_learning_curve(estimator, title, X, y, ylim=None, cv=None,
                        n_jobs=1, train_sizes=np.linspace(.1, 1.0, 5)):

    plt.figure()
    plt.title(title)
    if ylim is not None:
        plt.ylim(*ylim)
    plt.xlabel("Training examples")
    plt.ylabel("Score")
    train_sizes, train_scores, test_scores = learning_curve(
        estimator, X, y, cv=cv, n_jobs=n_jobs, train_sizes=train_sizes)
    train_scores_mean = np.mean(train_scores, axis=1)
    train_scores_std = np.std(train_scores, axis=1)
    test_scores_mean = np.mean(test_scores, axis=1)
    test_scores_std = np.std(test_scores, axis=1)
    plt.grid()

    plt.fill_between(train_sizes, train_scores_mean - train_scores_std,
                     train_scores_mean + train_scores_std, alpha=0.1,
                     color="r")
    plt.fill_between(train_sizes, test_scores_mean - test_scores_std,
                     test_scores_mean + test_scores_std, alpha=0.1, color="g")
    plt.plot(train_sizes, train_scores_mean, 'o-', color="r",
             label="Training score")
    plt.plot(train_sizes, test_scores_mean, 'o-', color="g",
             label="Cross-validation score")
```

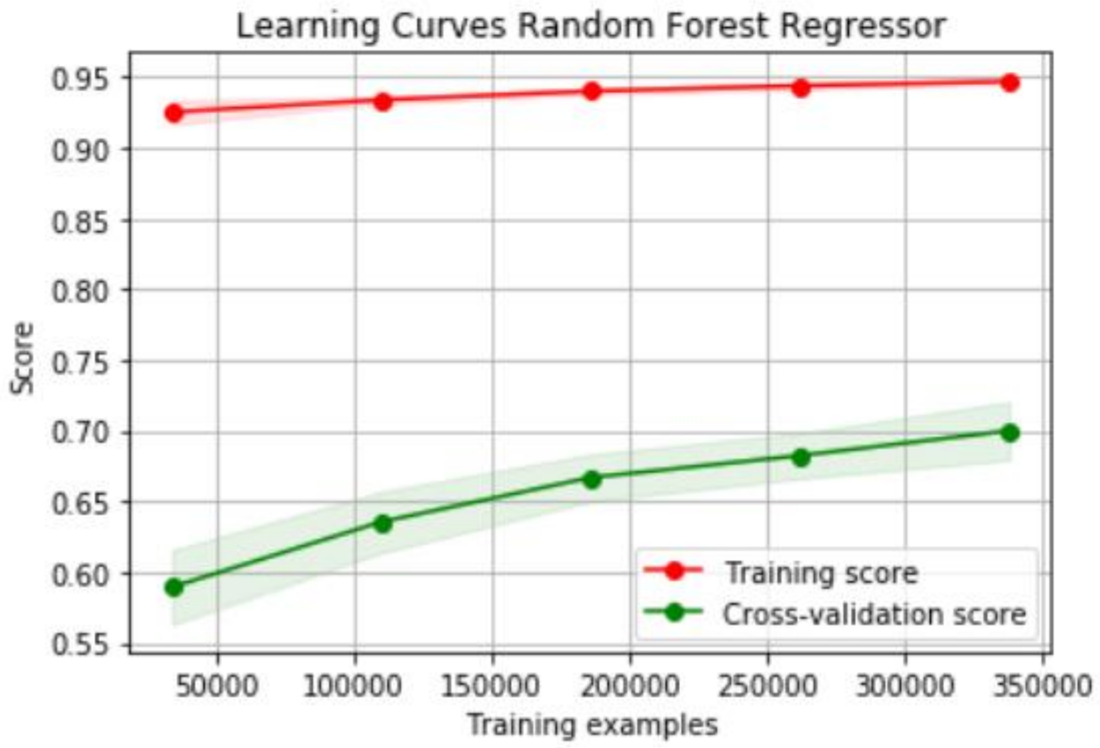
We are doing it for 3 distinct types which are as follows.

```
X, y = x_train.values, y_train.values


title = "Learning Curves (Linear Regression)"
# Cross validation with 100 iterations to get smoother mean test and train
# score curves, each time with 20% data randomly selected as a validation set.
cv = ShuffleSplit(n_splits=100, test_size=0.2, random_state=0)

estimator = LinearRegression()
plot_learning_curve(estimator, title, X, y, cv=cv, n_jobs=4)

title = "Learning Curves (Extra Trees Regressor)"
# Cross validation with 100 iterations to get smoother mean test and train
# score curves, each time with 20% data randomly selected as a validation set.
cv = ShuffleSplit(n_splits=100, test_size=0.2, random_state=0)

estimator = ExtraTreesRegressor()
plot_learning_curve(estimator, title, X, y, cv=cv, n_jobs=4)

title = "Learning Curves Random Forest Regressor"
cv = ShuffleSplit(n_splits=10, test_size=0.2, random_state=0)
estimator = RandomForestRegressor()
plot_learning_curve(estimator, title, X, y, cv=cv, n_jobs=4)

plt.show()
```

We received graphs as output which is as follows.







Docker

Docker is a computer program that performs operating-system-level virtualization also known as containerization. It is developed by Docker, Inc. Docker is primarily developed for Linux, where it uses the resource isolation features of the Linux kernel such as cgroups and kernel namespaces, and a union-capable file system such as OverlayFS and others to allow independent "containers" to run within a single Linux instance, avoiding the overhead of starting and maintaining virtual machines (VMs). The Linux kernel's support for namespaces mostly isolates an application's view of the operating environment, including process trees, network, user IDs and mounted file systems, while the kernel's cgroups provide resource limiting for memory and CPU. Since version 0.9, Docker includes the libcontainer library as its own way to directly use virtualization facilities provided by the Linux kernel, in addition to using abstracted virtualization interfaces via libvirt, LXC and systemd-nspawn.

Pipeline

Airflow