Importing libraries

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

Loading Datasets

```
df=pd.read_csv("/content/Movie_collection_train.csv")
df
```

	Collection	Marketin_expense	Production_expense	Multiplex_coverage	Budget
0	48000	20.1264	59.62	0.462	36524.125
1	43200	20.5462	69.14	0.531	35668.655
2	69400	20.5458	69.14	0.531	39912.675
3	66800	20.6474	59.36	0.542	38873.890
4	72400	21.3810	59.36	0.542	39701.585
...	...	...	...	...	...
395	26200	194.3350	91.20	0.307	35946.405
396	25000	137.4410	91.20	0.307	35579.775
397	17000	173.4404	91.20	0.307	31924.585
398	10000	787.0360	91.20	0.307	30291.415
399	12600	218.3310	91.20	0.307	32507.860

400 rows × 19 columns

Data preprocessing

EDD (Extended Data Dictionary)

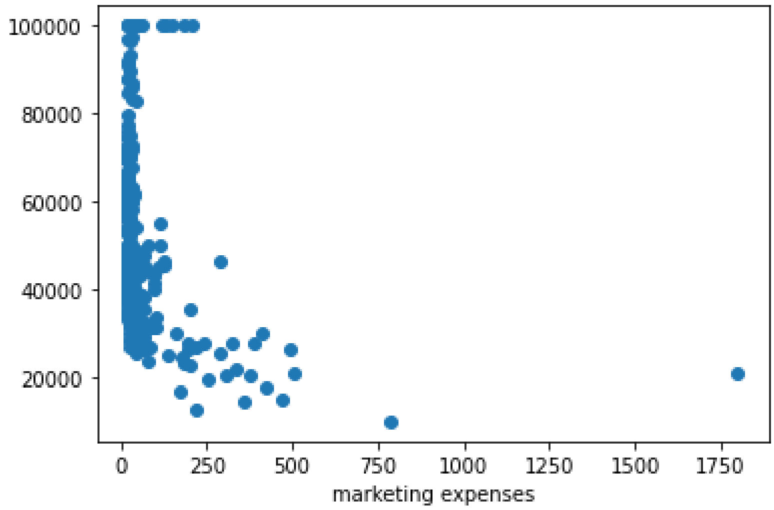
```
df.describe()
```

	Collection	Marketin_expense	Production_expense	Multiplex_coverage	
count	400.000000	400.000000	400.000000	400.000000	40(
mean	48646.500000	55.017180	73.832700	0.469881	3519
std	18308.499136	119.755634	13.023426	0.113920	407
min	10000.000000	20.126400	55.920000	0.129000	1978
25%	37800.000000	21.321950	63.250000	0.419000	3272
50%	45000.000000	23.214700	69.030000	0.494500	3459
75%	56500.000000	34.638300	82.840000	0.558000	3714
max	100000.000000	1799.524000	106.300000	0.615000	4877

Outliers detection and Treatment

```
plt.scatter(df.Marketin_expense,df.Collection)
plt.xlabel('marketing expenses')
```

```
Text(0.5, 0, 'marketing expenses')
```




```
plt.scatter(df.Twitter_hastags,df.Collection)
plt.xlabel('Twitter_hastags')
```

```
Text(0.5, 0, 'Twitter_hastags')
```


```
uv=np.percentile(df.Twitter_hastags,[99])[0]
df.Twitter_hastags[df.Twitter_hastags>3*uv]=3*uv
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:2: SettingWithCopyWarnir
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/us
```

```
#EDD after outliers treatment
df.describe()
```

	Collection	Marketin_expense	Production_expense	Multiplex_coverage	
count	400.000000	400.000000	400.000000	400.000000	40(
mean	48646.500000	55.017180	73.832700	0.469881	3519
std	18308.499136	119.755634	13.023426	0.113920	407
min	10000.000000	20.126400	55.920000	0.129000	1978
25%	37800.000000	21.321950	63.250000	0.419000	3272
50%	45000.000000	23.214700	69.030000	0.494500	3459
75%	56500.000000	34.638300	82.840000	0.558000	3714
max	100000.000000	1799.524000	106.300000	0.615000	4877

Detecting and imputing missing values

```
#detecting the number of missing value in a particular column
df.isna().sum()
```

```
Collection            0
Marketin_expense      0
Production_expense    0
Multiplex_coverage    0
Budget                0
Movie_length          0
Lead_ Actor_Rating    0
Lead_Actress_rating   0
```

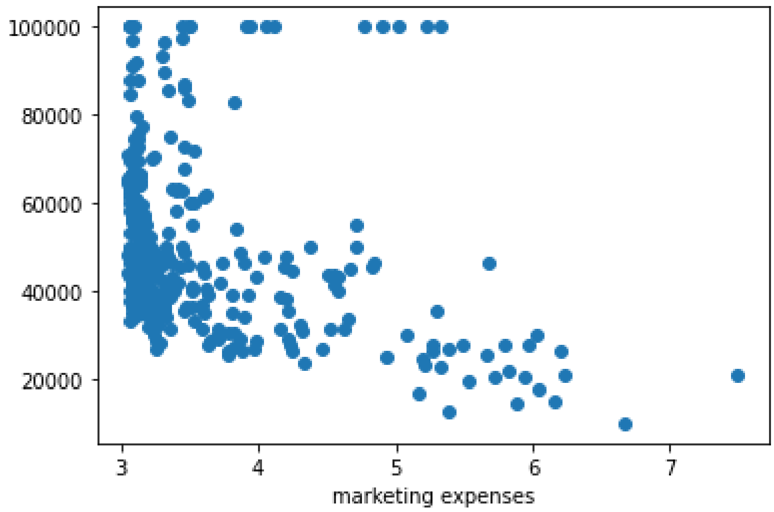
```
Director_rating        0
Producer_rating        0
Critic_rating          0
Trailer_views          0
Time_taken             8
Twitter_hastags        0
Genre                  0
Avg_age_actors         0
MPAA_film_rating       0
Num_multiplex          0
3D_available           0
dtype: int64
```

```
#Imputation of missing values
df.Time_taken=df.Time_taken.fillna(df.Time_taken.mean())
```

Variable Transformation

```
df.Marketin_expense=np.log(1+df.Marketin_expense)
plt.scatter(df.Marketin_expense,df.Collection)
plt.xlabel('marketing expenses')
```

```
Text(0.5, 0, 'marketing expenses')
```



Deletion of unnecessary variables

```
del df['MPAA_film_rating']
```

Handling qualitative data

Dummy variable creation:

```
df=pd.get_dummies(df)
df.head()
```

	Collection	Marketin_expense	Production_expense	Multiplex_coverage	Budget	M
0	48000	3.050523	59.62	0.462	36524.125	
1	43200	3.070199	69.14	0.531	35668.655	
2	69400	3.070181	69.14	0.531	39912.675	
3	66800	3.074885	59.36	0.542	38873.890	
4	72400	3.108212	59.36	0.542	39701.585	

5 rows × 22 columns

```
#delete unnecessary columns
del df['3D_available_NO']
```

```
del df['Genre_Action']
```

Correlation Analysis

```
df.corr()
```

	Collection	Marketin_expense	Production_expense	Multiplex_cove
Collection	1.000000	-0.309711	-0.373947	0.30
Marketin_expense	-0.309711	1.000000	0.615376	-0.70
Production_expense	-0.373947	0.615376	1.000000	-0.74
Multiplex_coverage	0.303971	-0.707648	-0.747325	1.00
Budget	0.754353	-0.306339	-0.403334	0.31
Movie_length	-0.278718	0.533402	0.609577	-0.71
Lead_ Actor_Rating	-0.110412	0.557617	0.668242	-0.74
Lead_Actress_rating	-0.109230	0.558538	0.669666	-0.75

from the correlation analysis, we can see that the feature which is atmost correlated with the 'collection'(target) is 'Budget'.

Simple Linear regression

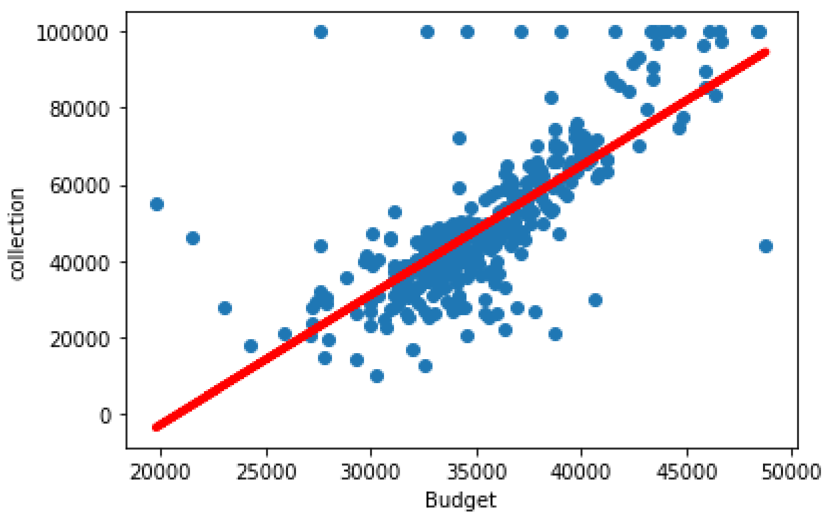
Time_taken	0.140573	0.048600	0.024382	0.02

```
from sklearn.linear_model import LinearRegression
lr=LinearRegression()
y=df['Collection']
x=df[['Budget']]
lr.fit(x,y)
regression_line=lr.predict(x)
print(lr.intercept_,lr.coef_)
```

```
-70624.10545910442 [3.388584]
```

```
plt.scatter(x,y)
plt.plot(x,regression_line,color='red',linewidth=4)
plt.xlabel('Budget')
plt.ylabel('collection')
```

```
Text(0, 0.5, 'collection')
```



Multiple Linear Regression

```
x_multi=df.drop("Collection",axis=1)
x_multi.head()
```

	Marketin_expense	Production_expense	Multiplex_coverage	Budget	Movie_length
0	3.050523	59.62	0.462	36524.125	138.7
1	3.070199	69.14	0.531	35668.655	152.4
2	3.070181	69.14	0.531	39912.675	134.6
3	3.074885	59.36	0.542	38873.890	119.3
4	3.108212	59.36	0.542	39701.585	127.7

```
y_multi=df['Collection']
```

```
#model taining without splitting
lr.fit(x_multi,y_multi)
print(lr.intercept_,lr.coef_)
```

```
-159361.76886924737 [ 9.66671082e+02 -6.25478537e+01  2.65168392e+04  2.15738234e+00
 -3.60671093e+01  8.53583151e+03 -1.40727631e+04  1.21806614e+04
 -2.63253967e+03  3.67503970e+03  1.00653022e-01  3.40347440e+01
  5.55368261e+00  5.09699981e+01  1.56458024e+01  4.17396939e+03
  4.47170229e+03  3.20132381e+03  2.47259466e+03]
```

```
#Splitting the datainto train and test data
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x_multi,y_multi,test_size=0.2,random_state=
```

```
#model training and prediction for test data
lr.fit(x_train,y_train)
y_pred_test=lr.predict(x_test)
y_pred_test
```

```
array([45488.53544783, 54358.27864328, 67145.76839519, 62049.59116147,
       38457.69377522, 31682.95458895, 49841.28055941, 74948.46422632,
       10671.57552665, 23480.3775603 , 44332.22143222, 29666.04230181,
       51504.70956988, 54463.76135676, 36708.58392172, 33673.12805413,
       48859.26214456, 27480.75340518, 45188.83051011, 62087.90804133,
       53833.26983156, 81817.99161128, 38300.70384266, 60410.17265884,
       59743.50022123, 54331.43601473, 49476.51896328, 44485.88069553,
       39811.87249076, 48334.38418292, 44698.26207927, 69098.39070335,
       46925.82034464, 83286.15091819, 57971.48545952, 46934.94055094,
```

```
       64750.23530748, 57584.40230335, 68773.26005041, 16511.05594827,
       53894.89981711, 48120.44626486, 46438.13453717, 51498.10693162,
       34908.4104853 , 32088.97185921, -4849.27043985, 35576.97397823,
       51025.05636312, 78458.12562106, 40910.34310117, 16190.38289093,
       45628.69350475, 48047.98818485, 77287.1225122 , 51778.0959102 ,
       57303.04220641, 35912.71287506, 37182.22902129, 42202.26213704,
       44527.1448397 , 55677.9124057 , 65748.51987116, 32461.59098058,
       53808.65872169, 50396.05707858, 31098.52069279, 51009.85134364,
       46601.95467898, 31177.42081495, 37469.99901127, 72621.67075029,
       42404.48783908, 26366.03986558, 42078.69628949, 50577.00325088,
       45772.96165894, 68085.68845915, 46972.44996106, 52696.58519309])
```

```
#prediction for train data
y_pred_train=lr.predict(x_train)
y_pred_train
```

```
       60733.61364416, 31536.76651645, 46544.93823102, 71573.97483765,
       57979.88371011, 43775.02310178, 49330.75958126, 54532.56066561,
       49840.15263797, 44975.37434802, 57486.68408903, 38477.02364143,
       89402.35202699, 54234.7181923 , 47479.77758898, 36127.08840485,
       61458.24750691, 41096.18127279, 46967.00000732, 53024.34138131,
       46419.23085269, 49474.42501302, 49972.69787281, 49607.01720519,
       40668.66532728, 35637.23413959, 72294.38233501, 49441.73627818,
       32604.2663275 , 41468.84346996, 45531.78627398, 41685.88640366,
       29899.06549298, 53019.64898771, 50672.19339665, 62512.30056934,
       59389.3928435 , 63291.6814904 , 49949.73173896, 46521.8364547 ,
       77103.15320792, 14310.60495119, 47116.59851252, 35216.22706258,
       47217.3942881 , 48220.62873862, 40677.52696827, 31092.47053489,
       54597.5505031 , 60105.53486218, 80408.04449448, 22287.2383199 ,
       28040.60530253, 51785.63942462, 68693.41659352, 34498.94473221,
       52156.78861987, 40996.2832104 , 57243.39353005, 37852.98176401,
       46858.33236282, 41277.58921616, 26383.57094104, 48639.20842789,
        9621.38834229,  7492.61473269, 84351.69947946, 65203.48045128,
       38615.99196613, 74918.67007858, 45825.51601571, 69036.31403395,
       31349.92790976, 39270.23248561, 45120.57885994, 67378.80443067,
       36884.60624686, 58754.18958537, 42199.96922457, 29744.2548982 ,
       71218.60048678, 38266.34009842, 51286.5822251 , 41939.13715493,
       46531.56627972, 46400.8543773 , 44385.00117074, 45923.34824991,
       43176.52743117, 44413.87831759, 70768.30980019, 43015.14753359,
       49879.13828726, 58294.56602458, 37922.64242486, 76599.41512619,
       67266.06438015, 66588.58506123, 60457.70321344, 42443.82179989,
       48327.43207688, 46156.29174583, 54269.11896162, 33500.30855887,
       33092.50986024, 74137.1602352 , 34947.36814304, 62101.26312156,
       69428.36208304, 40051.42476574, 56815.59952052, 41619.45066318,
       60060.58372333, 47635.48483303, 30283.82728765, 36991.11714764,
       52767.0508465 , 38831.22171325, 28372.652671  , 27418.48955385,
       37599.86094689, 31863.24676558, 42338.22889901, 33883.89209858,
       56559.2125105 , 47572.32778411, 51950.84452123, 47451.60417318,
       60102.77657616, 32182.50430025, 50685.01993292, 56215.43104316,
       50742.73939991, 37588.79172447, 77122.21609818, 10456.01835904,
       57588.65340464, 65326.03040406, 45004.28365441, 64459.41654165,
       57487.44002625, 43714.17620021, 44066.20278057, 45748.97644203,
       46645.8167382 , 13375.47013239, 33657.58596607, 66500.87833673,
       63123.12938248, 24312.5110594 , 25453.84398557, 71728.05671031,
       39975.04833093, 51305.27581253, 16532.72879613, 82321.72039002,
       52614.73828008, 77502.37208168, 43478.0355201 , 44323.01815679,
       41037.83267547, 34710.25666025, 51602.98342346, 49658.75658591,
       88152.69185695, 46954.32919438, 70522.96311179, 61660.73690354,
       52472.72305573, 94469.78242376, 36499.19983978, 51080.66054386,
```

```
       43703.07142082, 48023.15849807, 71533.17878962, 48877.26122623,
       44861.30932936, 49671.13888432, 50513.19058831, 48857.5778953 ,
       66807.38191913, 48409.54365655, 64328.30158081, 92396.86581688,
       36656.33364757, 31624.39614794, 74741.76129472, 36574.30055723,
       24233.24086998, 29279.03306374, 49637.06469705, 50372.10300021,
       40535.37673032, 34880.87869255, 13631.23256985, 57343.81108731,
       66478.71058868, 59877.05154308, 65069.20461343, 46451.25284643,
       42398.11964448, 55812.19785914, 56851.54804409, 37104.9576238 ,
       47908.92665077, 38878.44820401, 52441.51477845, 35018.38742911,
       49417.25920967, 41696.34287815,  4813.13511441, 50756.83598393,
       10805.54987196, 65075.76401798, 53307.67510328, 55244.94866428,
       63794.35781942, 51119.26885006, 56121.86172262, 59826.46757957,
       49953.50898721, 54633.04243858, 19570.52384162, 38199.39106414,
       39344.16641561, 81227.53050991, 58784.95629607, 39148.65816953,
       65560.81167802, 43413.71782734, 33021.1448313 , 39080.58172074])
```

```
#Accuracy of the model using r square method
from sklearn.metrics import r2_score
r2_score(y_test,y_pred_test)
```

```
0.6294584067165659
```

```
r2_score(y_train,y_pred_train)
```

```
0.7032392467424771
```

Linear models other than OLS (Ordinary least squares)

Standardization of data

```
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
x_train_s=sc.fit_transform(x_train)
x_test_s=sc.fit_transform(x_test)
```

Shrinkage methods:

1.Ridge

```
from sklearn.linear_model import Ridge
rd=Ridge(alpha=0.5)
rd.fit(x_train_s,y_train)
y_pred_s=rd.predict(x_test_s)
y_pred_s
```

```
array([46476.89368336, 55171.44958513, 68087.95396647, 64294.11329753,
       40618.54867448, 33251.8886704 , 50549.05425177, 75145.16479465,
       12807.82863039, 23948.81988917, 45190.76696496, 30322.32651643,
       52191.08070669, 55742.59519036, 37730.5987516 , 35906.81790053,
```

```
       50344.33798305, 29424.36817545, 46665.25568393, 63755.90684701,
       55406.00847628, 83689.58725283, 39578.11081731, 62938.66728494,
       60032.10593604, 55427.80627091, 51305.50242923, 45862.74022745,
       41596.32088238, 49451.3879843 , 45248.14765387, 70063.64171512,
       48268.73702111, 86373.22901507, 58993.10429251, 48673.09171808,
       65187.90776484, 58284.06105986, 70529.05171182, 19643.46676431,
       56562.84162096, 49545.03652992, 47329.11468561, 52716.80411284,
       35395.93415325, 34300.35633012, -3512.31754669, 37010.40735489,
       52080.90609385, 80437.6380752 , 42914.95048787, 17199.02314522,
       46029.94760216, 48646.3606515 , 78093.32768666, 51837.80027793,
       59263.61785434, 37535.16625031, 38821.83419345, 42850.79068548,
       47201.49128987, 57713.44131744, 67671.60335295, 34010.12918811,
       55458.11678033, 51668.42466188, 33011.45383581, 52651.69460268,
       46925.77634934, 31505.90075162, 38751.27472268, 73142.9394788 ,
       43616.03847094, 28526.91175363, 43306.45155085, 50727.78930822,
       46073.35594772, 68617.40095389, 48645.83001415, 53385.98898261])
```

```
#accuracy
r2_score(y_test,y_pred_s)
```

```
0.6357729421143091
```

To find the alpha value for which r^2 value is maximum, we use validation curve

```
from sklearn.model_selection import validation_curve
```

```
param_range=np.logspace(-2,8,100)
param_range
```

```
array([1.00000000e-02, 1.26185688e-02, 1.59228279e-02, 2.00923300e-02,
       2.53536449e-02, 3.19926714e-02, 4.03701726e-02, 5.09413801e-02,
       6.42807312e-02, 8.11130831e-02, 1.02353102e-01, 1.29154967e-01,
       1.62975083e-01, 2.05651231e-01, 2.59502421e-01, 3.27454916e-01,
       4.13201240e-01, 5.21400829e-01, 6.57933225e-01, 8.30217568e-01,
       1.04761575e+00, 1.32194115e+00, 1.66810054e+00, 2.10490414e+00,
       2.65608778e+00, 3.35160265e+00, 4.22924287e+00, 5.33669923e+00,
       6.73415066e+00, 8.49753436e+00, 1.07226722e+01, 1.35304777e+01,
       1.70735265e+01, 2.15443469e+01, 2.71858824e+01, 3.43046929e+01,
       4.32876128e+01, 5.46227722e+01, 6.89261210e+01, 8.69749003e+01,
       1.09749877e+02, 1.38488637e+02, 1.74752840e+02, 2.20513074e+02,
       2.78255940e+02, 3.51119173e+02, 4.43062146e+02, 5.59081018e+02,
       7.05480231e+02, 8.90215085e+02, 1.12332403e+03, 1.41747416e+03,
       1.78864953e+03, 2.25701972e+03, 2.84803587e+03, 3.59381366e+03,
       4.53487851e+03, 5.72236766e+03, 7.22080902e+03, 9.11162756e+03,
       1.14975700e+04, 1.45082878e+04, 1.83073828e+04, 2.31012970e+04,
       2.91505306e+04, 3.67837977e+04, 4.64158883e+04, 5.85702082e+04,
       7.39072203e+04, 9.32603347e+04, 1.17681195e+05, 1.48496826e+05,
       1.87381742e+05, 2.36448941e+05, 2.98364724e+05, 3.76493581e+05,
       4.75081016e+05, 5.99484250e+05, 7.56463328e+05, 9.54548457e+05,
       1.20450354e+06, 1.51991108e+06, 1.91791026e+06, 2.42012826e+06,
       3.05385551e+06, 3.85352859e+06, 4.86260158e+06, 6.13590727e+06,
       7.74263683e+06, 9.77009957e+06, 1.23284674e+07, 1.55567614e+07,
       1.96304065e+07, 2.47707636e+07, 3.12571585e+07, 3.94420606e+07,
       4.97702356e+07, 6.28029144e+07, 7.92482898e+07, 1.00000000e+08])
```

```
train_score,test_score=validation_curve(Ridge(),x_train_s,y_train,param_name="alpha",param_


print(train_score)
print(test_score)
```

```
 -1.22483281e-02]
[ 4.59375557e-03  1.35418612e-03 -9.16867841e-03 -3.83845052e-03
 -1.33004330e-02]
[ 3.01170536e-03  3.79146462e-05 -1.04886807e-02 -4.96000592e-03
 -1.41389291e-02]
[ 1.75210926e-03 -1.00996895e-03 -1.15389895e-02 -5.85234586e-03
 -1.48064189e-02]
[ 7.50202331e-04 -1.84341185e-03 -1.23740160e-02 -6.56174137e-03
 -1.53372902e-02]
[-4.61274377e-05 -2.50580496e-03 -1.30374491e-02 -7.12533490e-03
 -1.57591958e-02]
[-6.78678470e-04 -3.03194166e-03 -1.35642739e-02 -7.57286267e-03
 -1.60943059e-02]
[-1.18089285e-03 -3.44965348e-03 -1.39824448e-02 -7.92808153e-03
 -1.63603524e-02]
[-1.57947390e-03 -3.78116012e-03 -1.43142610e-02 -8.20993963e-03
 -1.65714905e-02]
[-1.89571061e-03 -4.04417388e-03 -1.45774857e-02 -8.43352971e-03
 -1.67390033e-02]
[-2.14655433e-03 -4.25279681e-03 -1.47862542e-02 -8.61086094e-03
 -1.68718735e-02]
[-2.34548931e-03 -4.41824580e-03 -1.49518049e-02 -8.75148090e-03
 -1.69772458e-02]
[-2.50323342e-03 -4.54943606e-03 -1.50830673e-02 -8.86297514e-03
 -1.70607988e-02]
[-2.62830048e-03 -4.65344910e-03 -1.51871320e-02 -8.95136721e-03
 -1.71270427e-02]
[-2.72745016e-03 -4.73590718e-03 -1.52696277e-02 -9.02143826e-03
 -1.71795585e-02]
[-2.80604730e-03 -4.80127237e-03 -1.53350206e-02 -9.07698212e-03
 -1.72211881e-02]
[-2.86834847e-03 -4.85308482e-03 -1.53868536e-02 -9.12100830e-03
 -1.72541862e-02]
[-2.91773006e-03 -4.89415264e-03 -1.54279369e-02 -9.15590372e-03
 -1.72803413e-02]
[-2.95686975e-03 -4.92670278e-03 -1.54604987e-02 -9.18356115e-03
 -1.73010716e-02]
[-2.98789083e-03 -4.95250110e-03 -1.54863060e-02 -9.20548133e-03
 -1.73175018e-02]
[-3.01247673e-03 -4.97294765e-03 -1.55067595e-02 -9.22285405e-03
 -1.73305237e-02]
[-3.03196203e-03 -4.98915232e-03 -1.55229695e-02 -9.23662247e-03
 -1.73408440e-02]
[-3.04740467e-03 -5.00199497e-03 -1.55358163e-02 -9.24753424e-03
 -1.73490231e-02]
[-3.05964326e-03 -5.01217300e-03 -1.55459975e-02 -9.25618196e-03
 -1.73555052e-02]
[-3.06934247e-03 -5.02023920e-03 -1.55540663e-02 -9.26303534e-03
 -1.73606424e-02]
[-3.07702915e-03 -5.02663170e-03 -1.55604607e-02 -9.26846666e-03
 -1.73647136e-02]
[-3.08312084e-03 -5.03169775e-03 -1.55655284e-02 -9.27277097e-03
 -1.73679400e-02]
[-3.08794850e-03 -5.03571259e-03 -1.55695444e-02 -9.27618212e-03
```

```
  -1.73704969e-02]
 [-3.09177438e-03 -5.03889432e-03 -1.55727271e-02 -9.27888542e-03
  -1.73725233e-02]
 [-3.09480636e-03 -5.04141582e-03 -1.55752494e-02 -9.28102777e-03
```

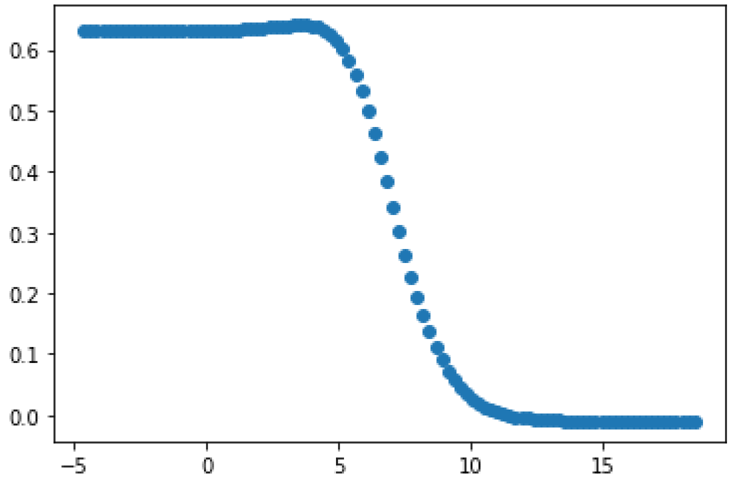
```
test_mean=np.mean(test_score, axis=1)
train_mean=np.mean(train_score,axis=1)
```

```
max(test_mean)
```

```
0.6407034912553751
```

```
plt.scatter(x=np.log(param_range),y=test_mean)
```

```
<matplotlib.collections.PathCollection at 0x7f5409f91b90>
```




```
#to find the loction of max test_mean in param_range
np.where(test_mean==max(test_mean))
```

```
(array([35]),)
```

```
param_range[35]
```

```
34.30469286314919
```

Model training using the best alpha value for greater accuracy

```
#model training using ridge
rd_best=Ridge(alpha=param_range[35])
rd_best.fit(x_train_s,y_train)
```

```
Ridge(alpha=34.30469286314919)
```

```
#prediction
r2_score(y_test,rd_best.predict(x_test_s))
```

```
0.6408784721817569
```

```
r2_score(y_train,rd_best.predict(x_train_s))
```

```
0.695293521696044
```

2.Lasso

```
from sklearn.linear_model import Lasso
ls=Lasso(alpha=0.4)
ls.fit(x_train_s,y_train)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_coordinate_descent.py:64
  coef_, l1_reg, l2_reg, X, y, max_iter, tol, rng, random, positive
Lasso(alpha=0.4)
```

To find the alpha value for which r^2 value is maximum, we use validation curve

```
train_lsc,test_lsc=validation_curve(Lasso(),x_train_s,y_train,param_name='alpha',param_ran
train_ls_mean=np.mean(train_lsc,axis=1)
test_lsc_mean=np.mean(test_lsc,axis=1)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_coordinate_descent.py
  coef_, l1_reg, l2_reg, X, y, max_iter, tol, rng, random, positive
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_coordinate_descent.py
  coef_, l1_reg, l2_reg, X, y, max_iter, tol, rng, random, positive
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_coordinate_descent.py
  coef_, l1_reg, l2_reg, X, y, max_iter, tol, rng, random, positive
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_coordinate_descent.py
  coef_, l1_reg, l2_reg, X, y, max_iter, tol, rng, random, positive
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_coordinate_descent.py
  coef_, l1_reg, l2_reg, X, y, max_iter, tol, rng, random, positive
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_coordinate_descent.py
  coef_, l1_reg, l2_reg, X, y, max_iter, tol, rng, random, positive
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_coordinate_descent.py
  coef_, l1_reg, l2_reg, X, y, max_iter, tol, rng, random, positive
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_coordinate_descent.py
  coef_, l1_reg, l2_reg, X, y, max_iter, tol, rng, random, positive
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_coordinate_descent.py
  coef_, l1_reg, l2_reg, X, y, max_iter, tol, rng, random, positive
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_coordinate_descent.py
  coef_, l1_reg, l2_reg, X, y, max_iter, tol, rng, random, positive
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_coordinate_descent.py
  coef_, l1_reg, l2_reg, X, y, max_iter, tol, rng, random, positive
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_coordinate_descent.py
  coef_, l1_reg, l2_reg, X, y, max_iter, tol, rng, random, positive
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_coordinate_descent.py
  coef_, l1_reg, l2_reg, X, y, max_iter, tol, rng, random, positive
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_coordinate_descent.py
  coef_, l1_reg, l2_reg, X, y, max_iter, tol, rng, random, positive
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_coordinate_descent.py
  coef_, l1_reg, l2_reg, X, y, max_iter, tol, rng, random, positive
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_coordinate_descent.py
  coef_, l1_reg, l2_reg, X, y, max_iter, tol, rng, random, positive
```

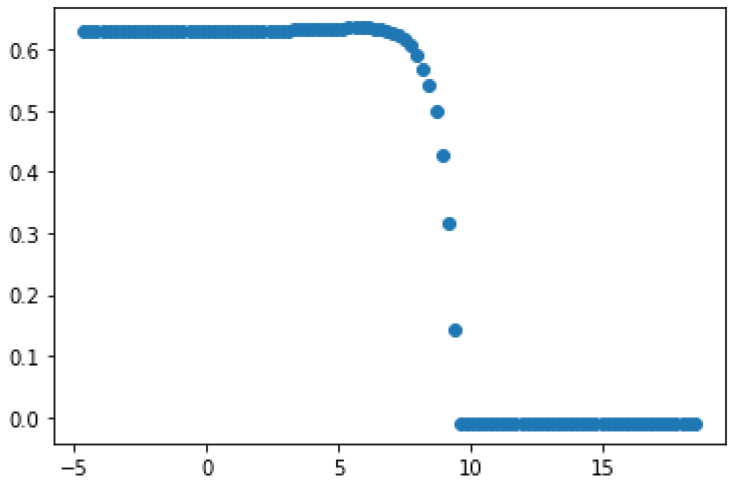
```
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_coordinate_descent.py
  coef_, l1_reg, l2_reg, X, y, max_iter, tol, rng, random, positive
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_coordinate_descent.py
  coef_, l1_reg, l2_reg, X, y, max_iter, tol, rng, random, positive
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_coordinate_descent.py
  coef_, l1_reg, l2_reg, X, y, max_iter, tol, rng, random, positive
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_coordinate_descent.py
  coef_, l1_reg, l2_reg, X, y, max_iter, tol, rng, random, positive
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_coordinate_descent.py
  coef_, l1_reg, l2_reg, X, y, max_iter, tol, rng, random, positive
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_coordinate_descent.py
  coef_, l1_reg, l2_reg, X, y, max_iter, tol, rng, random, positive
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_coordinate_descent.py
  coef_, l1_reg, l2_reg, X, y, max_iter, tol, rng, random, positive
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_coordinate_descent.py
  coef_, l1_reg, l2_reg, X, y, max_iter, tol, rng, random, positive
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_coordinate_descent.py
  coef_, l1_reg, l2_reg, X, y, max_iter, tol, rng, random, positive
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_coordinate_descent.py
  coef_, l1_reg, l2_reg, X, y, max_iter, tol, rng, random, positive
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_coordinate_descent.py
  coef_, l1_reg, l2_reg, X, y, max_iter, tol, rng, random, positive
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_coordinate_descent.py
  coef_, l1_reg, l2_reg, X, y, max_iter, tol, rng, random, positive
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_coordinate_descent.py
```

```
max(test_lsc_mean)
```

```
0.6358097798977403
```

```
plt.scatter(x=np.log(param_range),y=test_lsc_mean)
```

```
<matplotlib.collections.PathCollection at 0x7f5409e38310>
```




```
np.where(test_lsc_mean==max(test_lsc_mean))
```

```
(array([44]),)
```

```
param_range[44]
```

```
278.2559402207126
```

```
#model training using Lasso
ls_best=Lasso(alpha=param_range[44])
ls_best.fit(x_train_s,y_train)
```

```
Lasso(alpha=278.2559402207126)
```

```
#model testing/prediction
r2_score(y_test,ls_best.predict(x_test_s))
```

```
0.6369803779229701
```

```
r2_score(y_train,ls_best.predict(x_train_s))
```

```
0.6923236555033697
```

0s completed at 9:55 PM