```
import pandas as pd
data=pd.read_csv(r'https://d2beiqkhq929f0.cloudfront.net/public_assets/assets/000/001/839/original/Jamboree_Admission.csv')
```

```
data
```

	Serial No.	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit
0	1	337	118	4	4.5	4.5	9.65	1	0.92
1	2	324	107	4	4.0	4.5	8.87	1	0.76
2	3	316	104	3	3.0	3.5	8.00	1	0.72
3	4	322	110	3	3.5	2.5	8.67	1	0.80
4	5	314	103	2	2.0	3.0	8.21	0	0.65
...	...	...	...	...	...	...	...	...	...
495	496	332	108	5	4.5	4.0	9.02	1	0.87
496	497	337	117	5	5.0	5.0	9.87	1	0.96
497	498	330	120	5	4.5	5.0	9.56	1	0.93
498	499	312	103	4	4.0	5.0	8.43	0	0.73
499	500	327	113	4	4.5	4.5	9.04	0	0.84

500 rows × 9 columns

Next steps: Generate code with data View recommended plots

```
# data preprocessing
data.isnull().sum().sum()
```

```
0
```

```
data.shape
```

```
(500, 9)
```

```
data.duplicated().any()
```

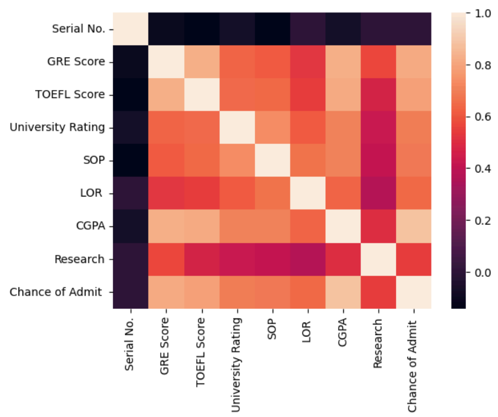
```
False
```

```
data.head(1)
data.corr()
```

	Serial No.	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit
Serial No.	1.000000	-0.103839	-0.141696	-0.067641	-0.137352	-0.003694	-0.074289	-0.005332	0.008505
GRE Score	-0.103839	1.000000	0.827200	0.635376	0.613498	0.524679	0.825878	0.563398	0.810351
TOEFL Score	-0.141696	0.827200	1.000000	0.649799	0.644410	0.541563	0.810574	0.467012	0.792228
University Rating	-0.067641	0.635376	0.649799	1.000000	0.728024	0.608651	0.705254	0.427047	0.690132
SOP	-0.137352	0.613498	0.644410	0.728024	1.000000	0.663707	0.712154	0.408116	0.684137
LOR	-0.003694	0.524679	0.541563	0.608651	0.663707	1.000000	0.637469	0.372526	0.645365
CGPA	-0.074289	0.825878	0.810574	0.705254	0.712154	0.637469	1.000000	0.501311	0.882413
Research	-0.005332	0.563398	0.467012	0.427047	0.408116	0.372526	0.501311	1.000000	0.545871
Chance of Admit	0.008505	0.810351	0.792228	0.690132	0.684137	0.645365	0.882413	0.545871	1.000000

```
# data visualisation
import seaborn as sns
sns.heatmap(data.corr())
```

```
<Axes: >
```




```
# data visualisation
```

```
import matplotlib.pyplot as plt
```

```
data['Chance of Admit ']=data['Chance of Admit ']*10
data
```

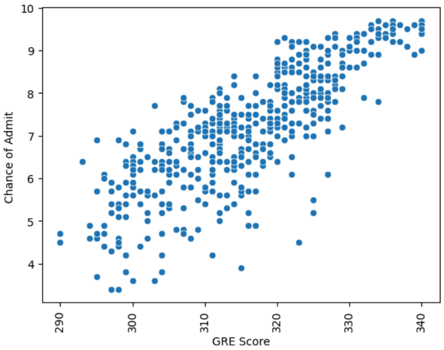
	Serial No.	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit
0	1	337	118	4	4.5	4.5	9.65	1	9.2
1	2	324	107	4	4.0	4.5	8.87	1	7.6
2	3	316	104	3	3.0	3.5	8.00	1	7.2
3	4	322	110	3	3.5	2.5	8.67	1	8.0
4	5	314	103	2	2.0	3.0	8.21	0	6.5
...	...	...	...	...	...	...	...	...	...
495	496	332	108	5	4.5	4.0	9.02	1	8.7
496	497	337	117	5	5.0	5.0	9.87	1	9.6
497	498	330	120	5	4.5	5.0	9.56	1	9.3
498	499	312	103	4	4.0	5.0	8.43	0	7.3
499	500	327	113	4	4.5	4.5	9.04	0	8.4

500 rows × 9 columns

Next steps: Generate code with data View recommended plots

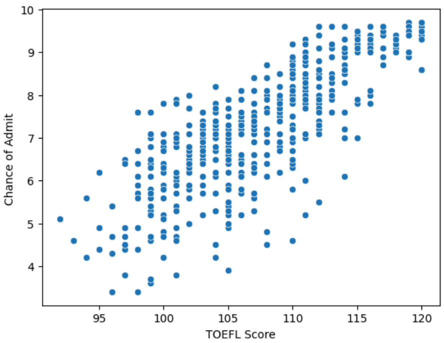
```
sns.scatterplot(x='GRE Score',y='Chance of Admit ',data=data)
plt.xticks(rotation=90)
```

```
(array([280., 290., 300., 310., 320., 330., 340., 350.]),
 [Text(280.0, 0, '280'),
  Text(290.0, 0, '290'),
  Text(300.0, 0, '300'),
  Text(310.0, 0, '310'),
  Text(320.0, 0, '320'),
  Text(330.0, 0, '330'),
  Text(340.0, 0, '340'),
  Text(350.0, 0, '350')])
```



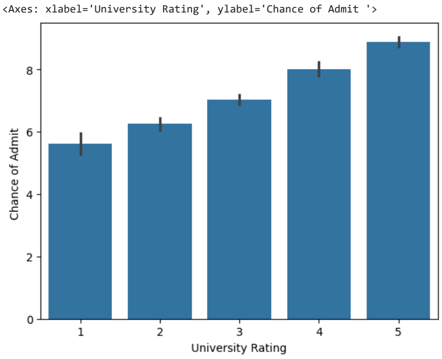

```
sns.scatterplot(x='TOEFL Score',y='Chance of Admit ',data=data)
```

```
<Axes: xlabel='TOEFL Score', ylabel='Chance of Admit '>
```



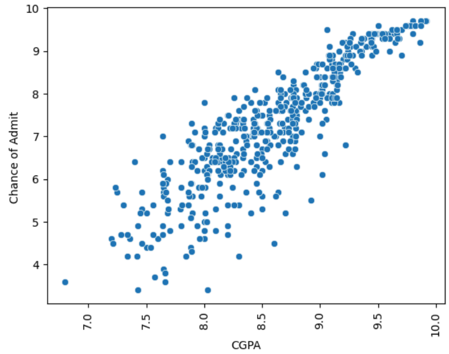

```
data
sns.barplot(x='University Rating',y='Chance of Admit ',data=data)
```

```
<Axes: xlabel='University Rating', ylabel='Chance of Admit '>
```




```
sns.scatterplot(x='CGPA',y='Chance of Admit ',data=data)
plt.xticks(rotation=90)
```

```
(array([ 6.5,  7. ,  7.5,  8. ,  8.5,  9. ,  9.5, 10. , 10.5]),
 [Text(6.5, 0, '6.5'),
  Text(7.0, 0, '7.0'),
  Text(7.5, 0, '7.5'),
  Text(8.0, 0, '8.0'),
  Text(8.5, 0, '8.5'),
  Text(9.0, 0, '9.0'),
  Text(9.5, 0, '9.5'),
  Text(10.0, 0, '10.0'),
  Text(10.5, 0, '10.5')])
```




```
# model training and testing
from sklearn.linear_model import LinearRegression
```

```
from sklearn.model_selection import train_test_split
x=data.iloc[:,0:8]
x
y=data.iloc[:,-1]
y
```

```
0      9.2
1      7.6
2      7.2
3      8.0
4      6.5
      ...
495    8.7
496    9.6
497    9.3
498    7.3
499    8.4
Name: Chance of Admit , Length: 500, dtype: float64
```

```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3,random_state=67)
```

```
x_train
```

	Serial No.	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research
324	325	315	104	3	3.0	2.5	8.33	0
79	80	294	93	1	1.5	2.0	7.36	0
336	337	319	110	3	3.0	2.5	8.79	0
58	59	300	99	1	3.0	2.0	6.80	1
36	37	299	106	2	4.0	4.0	8.40	0
...	...	...	...	...	...	...	...	...
7	8	308	101	2	3.0	4.0	7.90	0
453	454	319	103	3	2.5	4.0	8.76	1
202	203	340	120	5	4.5	4.5	9.91	1
309	310	308	110	4	3.5	3.0	8.60	0
323	324	305	102	2	2.0	2.5	8.18	0

350 rows × 8 columns

Next steps: Generate code with x_train View recommended plots

```
lr=LinearRegression()
model=lr.fit(x_train,y_train)
# firstly model will be trained means here based on all the train records mathematical function will be created it is a model
# by using gradient descent optimization takes place
# loss function like we can use sum of squares of errors,mean square error, mean absolute error we can choose any thing where
# this loss function is minimum that value related weights or coefficients we are going to consider
# based on that weight values  model is going to predict target variables
```

```
model.coef_
```

```
array([9.11379347e-04, 2.07464324e-02, 2.85944897e-02, 5.47027306e-02,
       3.07003309e-02, 1.72226278e-01, 1.20106338e+00, 2.30932496e-01])
```

```
model.intercept_
```

```
-13.932238739014954
```

```
y_pred=model.predict(x_test)
```

```
from sklearn import metrics
```

```
metrics.r2_score(y_pred,y_test)
# it is nearer to 1 so it is a good model
# r2 score value ranges from 0 to 1
```

```
0.8154902126690262
```

```
# multi_colinearity
```

```
# multi collinearity
# if there is a relationship is there between independent features or not  we are going to find out
# if there is a relation ship is there between independent features we cant find out actual regression coefficients then it is very difficult t
# based on weight values importance on features will be there
# based on ols and vif we can find multi collinearity
```

```
data
from statsmodels.stats.outliers_influence import variance_inflation_factor
from sklearn.preprocessing import StandardScaler
```

```
p=StandardScaler().fit_transform(x_test)
data=pd.DataFrame(p,columns=x_test.columns)
```

```
s=pd.DataFrame()
```

```
s['features']=data.columns
s
```

	features
0	Serial No.
1	GRE Score
2	TOEFL Score
3	University Rating
4	SOP
5	LOR
6	CGPA
7	Research

Next steps: Generate code with s View recommended plots

```
s.shape
data
```

	Serial No.	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research
0	0.054560	-0.869836	0.085023	0.642635	1.159202	1.513344	-0.513790	-1.241923
1	1.155586	-0.483814	-0.255069	-0.241725	-0.357196	1.513344	-0.099637	-1.241923
2	-0.751923	0.867263	0.085023	-0.241725	-0.357196	-0.055791	0.476576	-1.241923
3	-0.429330	-0.001287	-0.765207	-0.241725	-0.862662	-0.578837	-0.153657	0.805203
4	1.029354	-1.931397	-1.105299	-0.241725	-1.368128	0.467254	-1.720235	0.805203
...	...	...	...	...	...	...	...	...
145	1.309870	-0.483814	0.255069	-1.126085	-0.862662	0.467254	0.710662	-1.241923
146	1.050392	-0.387309	-2.295622	-1.126085	-0.862662	-2.147972	-0.891929	-1.241923
147	-0.057646	1.735813	1.275345	0.642635	1.159202	0.990299	1.899101	0.805203
148	-1.404123	-0.966342	-1.105299	-0.241725	0.653736	-0.578837	-0.765883	-1.241923
149	-1.628536	-0.676825	-1.445392	-1.126085	-1.873594	-1.624927	-2.386481	-1.241923

150 rows × 8 columns

Next steps: Generate code with data View recommended plots

```
len(data.values[0])
```

8

```
s['vif']=[variance_inflation_factor(data.values,i) for i in range(data.shape[1])]
```

```
s
```

	features	vif
0	Serial No.	1.024350
1	GRE Score	4.741730
2	TOEFL Score	4.483460
3	University Rating	3.017187
4	SOP	3.169092
5	LOR	2.084630
6	CGPA	5.851363
7	Research	1.525741

Next steps: Generate code with s View recommended plots

```
# all features  vif values are less than 10 so no need to remove columns
# vif is grater than or equal to 10 then we should have to remove columns which are strongly correlated
# vif it is 5 to 10 it is still more but our choice to remove or not
# vif is less than 5  then it is not a problem
```

```
# we can use ols method to see weather which column is more corelated based on hypothesis concept
```

```
# if null hypothesis is accepted for a particular column it says that this column is irrelevant in order to find the
# target features
# one  industry rule is there adjusted r2 value >0.85 or vif greater than  5 then only we are going to remove columns
# in both of them whichever  is correct we can remove column
```

```
import  statsmodels.api as sm
```

```
x_sm=sm.add_constant(x_train)
```

```
x_sm
```

	const	Serial No.	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research
324	1.0	325	315	104	3	3.0	2.5	8.33	0
79	1.0	80	294	93	1	1.5	2.0	7.36	0
336	1.0	337	319	110	3	3.0	2.5	8.79	0
58	1.0	59	300	99	1	3.0	2.0	6.80	1
36	1.0	37	299	106	2	4.0	4.0	8.40	0
...	...	...	...	...	...	...	...	...	...
7	1.0	8	308	101	2	3.0	4.0	7.90	0
453	1.0	454	319	103	3	2.5	4.0	8.76	1
202	1.0	203	340	120	5	4.5	4.5	9.91	1
309	1.0	310	308	110	4	3.5	3.0	8.60	0
323	1.0	324	305	102	2	2.0	2.5	8.18	0

350 rows × 9 columns

Next steps: Generate code with `x_sm` View recommended plots

```
k=sm.OLS(y_train,x_sm).fit()
k
```

<statsmodels.regression.linear_model.RegressionResultsWrapper at 0x7f5d0a8db100>

```
x_sm.shape
y_train.shape
```

(350,)

```
k.summary()
```

OLS Regression Results

Dep. Variable:	Chance of Admit	**R-squared:**	0.832
Model:	OLS	**Adj. R-squared:**	0.828
Method:	Least Squares	**F-statistic:**	210.5
Date:	Mon, 18 Mar 2024	**Prob (F-statistic):**	6.09e-127
Time:	07:14:47	**Log-Likelihood:**	-318.91
No. Observations:	350	**AIC:**	655.8
Df Residuals:	341	**BIC:**	690.5
Df Model:	8		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	-13.9322	1.267	-10.998	0.000	-16.424	-11.441
Serial No.	0.0009	0.000	3.879	0.000	0.000	0.001
GRE Score	0.0207	0.006	3.492	0.001	0.009	0.032
TOEFL Score	0.0286	0.010	2.747	0.006	0.008	0.049
University Rating	0.0547	0.046	1.196	0.233	-0.035	0.145
SOP	0.0307	0.057	0.540	0.590	-0.081	0.143
LOR	0.1722	0.052	3.321	0.001	0.070	0.274
CGPA	1.2011	0.112	10.744	0.000	0.981	1.421
Research	0.2309	0.080	2.871	0.004	0.073	0.389

Omnibus:	69.304	**Durbin-Watson:**	1.959
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	140.625
Skew:	-1.039	**Prob(JB):**	2.91e-31
Kurtosis:	5.308	**Cond. No.**	1.65e+04

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.65e+04. This might indicate that there are
strong multicollinearity or other numerical problems.

```
# feature scaling
# StandardScaler
from sklearn.preprocessing import StandardScaler
st=StandardScaler()
l=st.fit_transform(data)
pd.DataFrame(l,columns=data.columns)
```

	Serial No.	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research
0	0.054560	-0.869836	0.085023	0.642635	1.159202	1.513344	-0.513790	-1.241923
1	1.155586	-0.483814	-0.255069	-0.241725	-0.357196	1.513344	-0.099637	-1.241923
2	-0.751923	0.867263	0.085023	-0.241725	-0.357196	-0.055791	0.476576	-1.241923
3	-0.429330	-0.001287	-0.765207	-0.241725	-0.862662	-0.578837	-0.153657	0.805203
4	1.029354	-1.931397	-1.105299	-0.241725	-1.368128	0.467254	-1.720235	0.805203
...	...	...	...	...	...	...	...	...
145	1.309870	-0.483814	0.255069	-1.126085	-0.862662	0.467254	0.710662	-1.241923
146	1.050392	-0.387309	-2.295622	-1.126085	-0.862662	-2.147972	-0.891929	-1.241923
147	-0.057646	1.735813	1.275345	0.642635	1.159202	0.990299	1.899101	0.805203
148	-1.404123	-0.966342	-1.105299	-0.241725	0.653736	-0.578837	-0.765883	-1.241923
149	-1.628536	-0.676825	-1.445392	-1.126085	-1.873594	-1.624927	-2.386481	-1.241923

150 rows × 8 columns

```
# model performance evaluation  by using Mae,mse,r2,adjusted r2
```

```
# train and test performances are checked
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import adjusted_rand_score
```

```
mean_squared_error(y_pred,y_test)
```

0.29078083444414665

```
mean_absolute_error(y_pred,y_test)
```

0.41689893712161696

```
from sklearn.linear_model import Lasso,Ridge
```

```
lasso_model=Lasso(alpha=0.1) # alpha is tuner or hyperparameter which regulates l1 regularaizer
ridge_model=Ridge(alpha=0.1) # alpha is tuner or hyperparameter which regulates l2 regularaizer
```

```
lasso=lasso_model.fit(x_train,y_train)
ridge=ridge_model.fit(x_train,y_train)
y_lassopredict=lasso.predict(x_test)
```

```
y_ridge=ridge.fit(x_train,y_train)
```

Generate   create a dataframe with 2 columns and 10 rows   Close

Generate is available for a limited time for unsubscribed users. Upgrade to Colab Pro

```
y_lassopredict
```

```
array([7.05629936, 7.30947643, 7.76087589, 6.84387658, 5.7200202 ,
       8.65027748, 7.27869425, 6.2478123 , 5.11588513, 6.2436753 ,
       5.34641447, 6.82072971, 7.58801227, 9.30564406, 8.24216202,
       6.77697273, 6.33368873, 6.01233491, 7.26243725, 5.00803003,
       7.68501303, 6.68899431, 7.02132287, 6.30952546, 8.36690628,
       5.42106524, 8.17830667, 6.42472314, 6.43606181, 6.59873572,
       7.01300546, 9.68197672, 8.77301848, 7.81604879, 5.91327976,
       8.72041373, 8.96253667, 8.40054136, 8.03742465, 7.94013844,
       8.04150117, 5.98127945, 7.94822912, 6.33174485, 8.02933346,
       6.10211864, 8.46496534, 8.26407751, 5.8074222 , 7.51219535,
       8.07878771, 7.88998228, 8.13682675, 6.45457343, 6.49061727,
       7.45380257, 8.43844696, 7.59725465, 7.86942512, 8.30406521,
       7.03528768, 7.23827327, 8.17194755, 7.17320493, 8.11871286,
       7.62919498, 9.70229721, 9.23772387, 9.2640647 , 8.07629779,
       8.53540787, 7.83698779, 7.08536693, 8.09798994, 7.38025593,
       7.89118968, 6.62812615, 5.04016567, 6.48782465, 8.44828567,
       9.20582716, 6.67949571, 6.74772246, 9.44600997, 9.03852767,
       6.27367872, 7.58550459, 6.98148968, 6.16251808, 6.17332305,
       7.57007168, 8.09406874, 6.55319667, 6.06690878, 6.23502438,
       6.57552815, 4.85770585, 8.35128077, 8.02182004, 5.75866143,
       5.52238004, 9.35748323, 9.20857864, 6.95048945, 8.16729028,
       6.51752913, 6.1633283 , 7.53296864, 7.73212129, 9.16608927,
       9.61873933, 7.23659182, 7.0598029 , 6.42434215, 9.04603982,
       6.28325007, 6.54413668, 8.73098293, 6.50321167, 7.72837721,
       7.10957218, 9.5817492 , 6.78798018, 7.69181567, 6.59765118,
       5.2501381 , 8.24541336, 8.13881567, 7.18187059, 8.28739663,
       6.13901853, 8.0399477 , 8.33864989, 7.29236255, 6.58824287,
       6.44246198, 6.01197075, 6.57948801, 6.80241581, 7.29956569,
       6.16497485, 5.2708988 , 4.59717691, 9.48454855, 7.58144818,
       7.46727578, 5.92300022, 9.26165393, 5.98414188, 5.37980136])
```

```
y_ridge_predict=y_ridge.predict(x_test)
```

```
y_ridge_predict
```

```
array([ 7.02829404,  7.37113562,  7.59653661,  7.01746629,  5.84848008,
        8.62583264,  7.5343564 ,  5.94085444,  5.20304946,  6.20759601,
        5.12636906,  6.40448723,  7.48050315,  9.71100614,  8.73003355,
        6.5270648 ,  6.31209726,  5.79332745,  7.40142651,  4.69809296,
        7.85761607,  6.57959345,  6.86699317,  6.21097313,  8.59057679,
        5.14094285,  8.83561569,  5.62090187,  6.91277575,  6.47124763,
        6.55539629,  9.74831833,  8.96015447,  7.52790253,  5.8401609 ,
        8.88821582,  9.1030486 ,  8.52053702,  7.77868441,  7.98026571,
        7.81144684,  6.23173787,  8.00656473,  6.4357932 ,  8.15684664,
        6.42180036,  8.73685024,  8.59111911,  5.90293018,  7.43254648,
        7.93686322,  7.89860995,  7.82850787,  6.37796471,  6.62952593,
        7.42569227,  8.50336225,  7.37989226,  7.65448877,  8.31649083,
        6.74454263,  7.22709206,  8.27849552,  7.48681688,  8.0191577 ,
        7.75145006, 10.02786759,  9.46310784,  9.21798344,  8.44979635,
        8.81111659,  7.9257854 ,  7.05797352,  8.1939668 ,  8.03594586,
        7.8619827 ,  6.41275821,  5.53215144,  6.18333469,  8.55730188,
        9.09088283,  6.52525748,  7.19337191,  9.75260959,  8.79664937,
        6.26124395,  7.63347546,  6.9719238 ,  6.6555036 ,  6.15049562,
        7.99765777,  8.2579309 ,  6.79479963,  6.03306453,  5.80847123,
        6.57139697,  4.90304415,  8.43190315,  8.33949775,  5.69078021,
        5.6455095 ,  9.58003186,  9.21875041,  6.70695993,  8.47038771,
        6.5862863 ,  6.22288536,  8.1077011 ,  7.63929496,  9.27650973,
        9.78511449,  7.29801827,  7.3540355 ,  6.2956326 ,  9.33252745,
        6.57154322,  6.41523611,  8.79247155,  6.21202186,  7.88520182,
        7.65499237,  9.52462265,  6.77301109,  7.58112741,  7.06758907,
        5.50553856,  8.36412137,  8.36333946,  7.30218784,  8.36474227,
        6.44131993,  7.59496956,  8.44131173,  7.18397753,  6.27917781,
        6.35470094,  5.92528991,  6.35295043,  7.29114124,  7.28187686,
        5.58105424,  5.26960363,  4.66655451,  9.53954619,  7.44631487,
        7.77304365,  5.8331482 ,  9.52575097,  6.03389237,  4.62718069])
```

```
from sklearn.preprocessing import mean_squared_error
```

```
mean_squared_error(y_ridge_predict,y_test)
```

```
0.2906790062456789
```

```
mean_squared_error(y_lassopredict,y_test)
```

```
0.3210632996410194
```

```
# insights
# model is good
# here mean squared error and mean absolute error are low

# based on values of mean squared error and mean absolute error model performance is good
# mean squared error range between 0 to infinity
# mean absolute error range between 0 to infinity
# r2 score range between 0 to 1
# mean absolute error we cant defferentiate it
# we have scaled the total data frame such that when model is created then which feature is having more value while finding
# the target variable  which feature having more value that value is considered as an important feature
# in finding target variable so the values  are scaled such that range between 0 to 1 or
# -1 to +1
# then weight values now plays a key role  which weight value is more means for a feature that feature is considered as important feature
# lasso regression model is created in that l1 regularizer along with hyperparameter in  order to convert
#irrelevant features weight values will to zero based on gradient descent
# ridge regression model contains l2 regularizer with hyper parameter it will increase or decrease loss function based on the
#model  weather model  is facing  overfitting or underfitting problems
```

```
# recommendations
# based on this results we cant use this model in real world inustry because the model should follow some rules
# model should  make sure that independent and dependent features have linear relationship  is there or not
```