In [1]:
```
# Importing some important libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import pickle
from pandas_profiling import ProfileReport
```

In [2]:
```
# This library helps us in creating statiscal model of the dataset
# By this lib. we can check p-value and 100 - pvalue will give us
# contribution factor of the column or feature corresponding to label or ouput feature
import statsmodels.formula.api as snf
```

In [3]:
```
# Importing modules from sklearn
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import Ridge,Lasso,RidgeCV,LassoCV,ElasticNet,ElasticNetCV,Li
from sklearn.model_selection import train_test_split
```

Admission Dataset

In [4]:
```
df = pd.read_csv('Admission_Prediction.csv')
```

In [5]:
```
df.head()
```

Out[5]:

	Serial No.	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit
0	1	337.0	118.0	4.0	4.5	4.5	9.65	1	0.92
1	2	324.0	107.0	4.0	4.0	4.5	8.87	1	0.76
2	3	NaN	104.0	3.0	3.0	3.5	8.00	1	0.72
3	4	322.0	110.0	3.0	3.5	2.5	8.67	1	0.80
4	5	314.0	103.0	2.0	2.0	3.0	8.21	0	0.65

EDA And Feature Engineering

In [15]:
```
# checking null values
df.isnull().sum()
```

Out[15]:
```
Serial No.            0
GRE Score            15
TOEFL Score          10
University Rating    15
SOP                   0
LOR                   0
CGPA                  0
Research              0
Chance of Admit       0
dtype: int64
```

In [16]:
```
# Filling null values with mean
df['GRE Score'] = df['GRE Score'].fillna(df['GRE Score'].mean())
```

In [17]:
```
df.isnull().sum()
```

Out[17]:
```
Serial No.            0
GRE Score             0
TOEFL Score          10
University Rating    15
SOP                   0
LOR                   0
CGPA                  0
Research              0
Chance of Admit       0
dtype: int64
```

In [18]:
```
df['TOEFL Score'] = df['TOEFL Score'].fillna(df['TOEFL Score'].mean())
```

In [19]:
```
df['University Rating'] = df['University Rating'].fillna(df['University Rating'].mean()
```

In [22]:
```
df.isnull().sum()
```

Out[22]:
```
Serial No.           0
GRE Score            0
TOEFL Score          0
University Rating    0
SOP                  0
LOR                  0
CGPA                 0
Research             0
Chance of Admit      0
dtype: int64
```

In [23]:
```
df.drop(columns = 'Serial No.',inplace = True)
```

In [24]:
```
df.head()
```

Out[24]:

	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit
0	337.000000	118.0	4.0	4.5	4.5	9.65	1	0.92
1	324.000000	107.0	4.0	4.0	4.5	8.87	1	0.76
2	316.558763	104.0	3.0	3.0	3.5	8.00	1	0.72
3	322.000000	110.0	3.0	3.5	2.5	8.67	1	0.80
4	314.000000	103.0	2.0	2.0	3.0	8.21	0	0.65

In [25]:
```
y = df['Chance of Admit']
```

In [26]:
```
X = df.drop(columns='Chance of Admit')
```

In [27]:
```
scaler = StandardScaler()
```

In [28]:
```
# StandardScaling of the dataset
arr = scaler.fit_transform(X)
```

In [68]:
```
# Means used for the features for transformation
scaler.mean_
```

Out[68]:
```
array([316.55876289, 107.1877551 ,   3.12164948,   3.374     ,
         3.484     ,   8.57644   ,   0.56      ])
```

In [29]:
```
df1 = pd.DataFrame(arr)
```

In [37]:
```
#df1.profile_report()
```

In [38]:
```
df1
```

Out[38]:

	0	1	2	3	4	5	6
0	1.842741e+00	1.788542	0.778906	1.137360	1.098944	1.776806	0.886405
1	6.708143e-01	-0.031058	0.778906	0.632315	1.098944	0.485859	0.886405
2	5.124333e-15	-0.527313	-0.107877	-0.377773	0.017306	-0.954043	0.886405
3	4.905178e-01	0.465197	-0.107877	0.127271	-1.064332	0.154847	0.886405
4	-2.306679e-01	-0.692731	-0.994659	-1.387862	-0.523513	-0.606480	-1.128152
...	...	...	...	...	...	...	...
495	1.392000e+00	0.134360	1.665688	1.137360	0.558125	0.734118	0.886405
496	1.842741e+00	1.623124	1.665688	1.642404	1.639763	2.140919	0.886405
497	1.211704e+00	2.119379	1.665688	1.137360	1.639763	1.627851	0.886405
498	-4.109644e-01	-0.692731	0.778906	0.632315	1.639763	-0.242367	-1.128152
499	9.412590e-01	0.961451	0.778906	1.137360	1.098944	0.767220	-1.128152

500 rows × 7 columns

In [39]:
```
df1.describe()
```

Out[39]:

	0	1	2	3	4	5	
count	5.000000e+02	5.000000e+02	5.000000e+02	5.000000e+02	5.000000e+02	5.000000e+02	5.00
mean	4.350520e-15	9.419132e-16	5.608847e-16	2.926548e-16	-1.332268e-17	3.091971e-15	-2.2
std	1.001002e+00	1.001002e+00	1.001002e+00	1.001002e+00	1.001002e+00	1.001002e+00	1.00
min	-2.394225e+00	-2.512331e+00	-1.881441e+00	-2.397950e+00	-2.686789e+00	-2.940115e+00	-1.12
25%	-6.814090e-01	-6.927310e-01	-9.946589e-01	-8.828175e-01	-5.235128e-01	-7.430227e-01	-1.12
50%	5.124333e-15	-3.105811e-02	-1.078766e-01	1.272712e-01	1.730621e-02	-2.720919e-02	8.8
75%	6.708143e-01	7.960330e-01	7.789057e-01	6.323155e-01	5.581253e-01	7.672196e-01	8.8
max	2.113186e+00	2.119379e+00	1.665688e+00	1.642404e+00	1.639763e+00	2.223672e+00	8.8

checking multicollinearity using statsmodels

In [40]:
```
## To check multicollinearity in the dataset
from statsmodels.stats.outliers_influence import variance_inflation_factor
vif_df = pd.DataFrame()
```

In [44]:
```
vif_df['vif'] = [variance_inflation_factor(arr,i) for i in range(arr.shape[1])]
```

In [45]:
```
vif_df['feature'] = X.columns
```

In [46]:
```
vif_df
```

Out[46]:

	vif	feature
0	4.153268	GRE Score
1	3.792866	TOEFL Score
2	2.508768	University Rating
3	2.775750	SOP
4	2.037308	LOR
5	4.651670	CGPA
6	1.459311	Research

THERE IS NO MULTICOLLINEARITY AMONG FEATURES IN THE DATASET.

In [49]:
```
arr
```

Out[49]:
```
array([[ 1.84274116e+00,  1.78854223e+00,  7.78905651e-01, ...,
         1.09894429e+00,  1.77680627e+00,  8.86405260e-01],
       [ 6.70814288e-01, -3.10581135e-02,  7.78905651e-01, ...,
         1.09894429e+00,  4.85859428e-01,  8.86405260e-01],
       [ 5.12433309e-15, -5.27312752e-01, -1.07876604e-01, ...,
         1.73062093e-02, -9.54042814e-01,  8.86405260e-01],
       ...,
       [ 1.21170361e+00,  2.11937866e+00,  1.66568791e+00, ...,
         1.63976333e+00,  1.62785086e+00,  8.86405260e-01],
       [-4.10964364e-01, -6.92730965e-01,  7.78905651e-01, ...,
         1.63976333e+00, -2.42366993e-01, -1.12815215e+00],
       [ 9.41258951e-01,  9.61451165e-01,  7.78905651e-01, ...,
         1.09894429e+00,  7.67219636e-01, -1.12815215e+00]])
```

Training and Testing data

In [51]:
```
# Train Test split
X_train,X_test,y_train,y_test = train_test_split(arr,y,test_size=0.25,random_state= 100
```

In [52]:
```
X_train
```

Out[52]:
```
array([[-0.41096436, -0.52731275, -0.1078766 , ...,  0.01730621,
        -0.25891759, -1.12815215],
       [ 0.12992496, -0.19647633, -0.1078766 , ..., -0.52351283,
         0.12174622, -1.12815215],
       [-1.13215013, -1.02356739, -0.99465886, ..., -1.06433187,
        -1.51676323, -1.12815215],
       ...,
       [-1.04200191, -0.85814918, -0.99465886, ..., -1.06433187,
        -0.65613201, -1.12815215],
       [-0.50111259, -0.85814918, -0.1078766 , ...,  0.55812525,
         0.10519562,  0.88640526],
       [-1.31244657, -0.85814918, -1.88144112, ..., -2.14596996,
        -0.95404281, -1.12815215]])
```

Linear Regression

In [53]:
```
# object
lin = LinearRegression()
```

In [56]:
```
# training model to get slope and intercept for features of dataset
lin.fit(X_train,y_train)
```

Out[56]:
```
LinearRegression()
```

In [57]:
```
# slopes
lin.coef_
```

Out[57]:
```
array([0.015458  , 0.01908417, 0.00381077, 0.00315846, 0.01678637,
       0.07622763, 0.01400522])
```

In [58]:
```
# Intercept
lin.intercept_
```

Out[58]:
```
0.7181156002659718
```

In [59]:
```
# saving trained model using pickle
pickle.dump(lin,open('admission_model.pkl','wb'))
```

In [60]:
```
ls
```

```
 Volume in drive C is OS
 Volume Serial Number is B4B1-1DA2

 Directory of C:\Users\aniyant\python\Ineuron\Ineuron Previous Batch Lec\ML and EDA

07-Sep-22  03:40 PM    <DIR>          .
07-Sep-22  03:40 PM    <DIR>          ..
07-Sep-22  03:07 PM    <DIR>          .ipynb_checkpoints
07-Sep-22  03:40 PM               548 admission_model.pkl
22-Jun-22  12:30 PM            16,085 Admission_Prediction.csv
20-Jun-22  06:44 PM         1,697,600 ads.html
28-Aug-22  10:10 AM        18,165,424 EDA.ipynb
13-Jun-22  02:55 PM            51,260 FitBit data.csv
27-Aug-22  07:56 PM        19,156,131 FitBit_EDA _And_Feature Engineering.ipynb
20-Jun-22  07:34 PM               452 linear.sav
07-Sep-22  03:40 PM            52,034 LInear_Ridge_Lasso_ElasticNet.ipynb
07-Sep-22  03:04 PM            32,447 LinearRegression.ipynb
07-Sep-22  03:04 PM        27,473,937 Logistics regression implementation.ipynb
              10 File(s)     66,645,918 bytes
               3 Dir(s)  28,583,088,128 bytes free
```

In [62]:
```
# transforming data for testing purpose
test1 = scaler.transform([[337.000000,118.0,4.0,4.5,4.5,9.65,1]])
```

In [69]:
```
test1
```

Out[69]:
```
array([[1.84274116, 1.78854223, 0.77890565, 1.13735981, 1.09894429,
        1.77680627, 0.88640526]])
```

In [70]:
```
# Making prediction
lin.predict(test1)[0]
```

Out[70]:
```
0.9535973940109868
```

In [71]:
```
test2 = scaler.transform([[324.000000,107.0,4.0,4.0,4.5,8.87,1]])
```

In [72]:
```
lin.predict(test2)
```

Out[72]: array([0.8007552])

In [73]:
```
# Loading saved pickle file to make prediction on it
lin_pkl = pickle.load(open('admission_model.pkl','rb'))
```

In [74]:
```
lin_pkl.predict(test2)
```

Out[74]: array([0.8007552])

In [77]:
```
# R2 score of the model
lin.score(X_test,y_test) # dataset get out of train test split
```

Out[77]: 0.8262844735686963

In [78]:
```
# Let's create a function create adjusted R-squared

def adj_r2(x,y):
    r2 = lin.score(x,y)
    n = x.shape[0]
    p = x.shape[1]

    adjusted_r2 = 1- (1-r2) * (n-1)/(n-p-1)
    return adjusted_r2
```

In [80]:
```
adj_r2(X_test,y_test)
```

Out[80]: 0.8158912369446012

Regularization : LASSO

In [81]:
```
# LassoCV is used for hyperparameter tuning to get the best alpha value
lasso_cv = LassoCV(cv=10,max_iter=200000,normalize=True)
```

In [83]:
```
# Training model on the dataset and alpha values are chosen implicitly
lasso_cv.fit(X_train,y_train)
```

Out[83]: LassoCV(cv=10, max_iter=200000, normalize=True)

In [85]:
```
# Best chosen alpha
lasso_cv.alpha_
```

Out[85]: 9.742054000449748e-06

In [86]:
```
# Training model using lasso, alpha value is selected from above
lasso= Lasso(alpha=lasso_cv.alpha_)
lasso.fit(X_train,y_train)
```

Out[86]: Lasso(alpha=9.742054000449748e-06)

In [88]:
```
# slopes
lasso.coef_
```

Out[88]:
```
array([0.01545838, 0.01908149, 0.00380865, 0.00315615, 0.01678179,
       0.07622787, 0.01399974])
```

In [90]:
```
# intercept
lasso.intercept_
```

Out[90]:
```
0.7181154878580782
```

In [91]:
```
# alpha
lasso.alpha
```

Out[91]:
```
9.742054000449748e-06
```

In [92]:
```
# R2 Score
lasso.scoe(X_test,y_test)
```

Out[92]:
```
0.8262977741326504
```

Ridge

In [94]:
```
# RidgeCV is used to calculate alpha values
ridge_cv = RidgeCV(alphas=np.random.uniform(0,10,50), cv = 10,normalize = True)
```

In [95]:
```
# Training to get best alpha
ridge_cv.fit(X_train,y_train)
```

Out[95]:
```
RidgeCV(alphas=array([6.86802509, 2.20666752, 0.9387914 , 4.16756124, 9.65121867,
       8.9378751 , 4.36916388, 5.51593048, 4.06133823, 3.79821316,
       2.27419958, 1.61706836, 4.43216293, 5.562069  , 1.33518031,
       7.33488434, 1.42019074, 0.23461087, 8.51258018, 0.60772957,
       9.84839238, 8.59064879, 5.05054261, 3.44782082, 0.46148754,
       2.09412024, 9.51910449, 9.6367529 , 0.03986625, 8.25365922,
       5.42463656, 1.4489003 , 9.63892675, 6.25523996, 6.46287   ,
       0.24984325, 2.85870413, 8.16365081, 6.0746236 , 6.92300657,
       9.63748785, 6.29513815, 4.17636606, 9.48413879, 1.45232261,
       0.25601542, 7.24626929, 3.80921251, 7.71775441, 3.63783528]),
        cv=10, normalize=True)
```

In [97]:
```
# Best aplpha
ridge_cv.alpha_
```

Out[97]:
```
0.03986624728800292
```

In [100…
```
# this generates 50 data points between 0 to 10
np.random.uniform(0,10,50)
```

Out[100…
```
array([3.97786974, 4.39128124, 1.55104035, 5.99861076, 6.61045805,
       7.56510343, 9.03875899, 7.3601661 , 3.37044407, 5.33785276,
       9.31563898, 7.1017156 , 4.46891991, 9.12100597, 6.64233325,
       1.67502726, 7.19066812, 2.90365246, 4.92802075, 0.97947282,
       9.48743829, 3.5659289 , 6.4359049 , 0.66730316, 4.25655261,
       9.26489184, 6.70363506, 1.92814794, 9.92306351, 3.97372924,
       8.03096164, 8.75658177, 0.39315309, 3.34207194, 4.74369993,
       1.02405844, 6.62576395, 2.77917884, 0.97324526, 0.46830156,
       9.29621595, 5.12957902, 6.94824045, 1.74275946, 4.68124524,
       3.97374255, 8.51964676, 4.32560317, 6.16940685, 2.47488927])
```

In [104…
```
# Using Above calculated alpha to train model using ridge
ridge = Ridge(alpha=ridge_cv.alpha_)
```

```
ridge.fit(X_train,y_train)
```

Out[104… Ridge(alpha=0.03986624728800292)

In [105…
```
# slopes
ridge.coef_
```

Out[105…
```
array([0.01547134, 0.0190883 , 0.00381551, 0.00316578, 0.01678822,
       0.07619546, 0.01400377])
```

In [106…
```
# intercepts
ridge.intercept_
```

Out[106… 0.7181163666123268

In [107…
```
# alpha
ridge.alpha
```

Out[107… 0.03986624728800292

In [109…
```
# R2 score
ridge.score(X_test,y_test)
```

Out[109… 0.8262980573952111

ElasticNet

In [110…
```
# Using ElasticNetCV to calculate alpha
elastic_cv = ElasticNetCV(alphas=None,cv = 10)
elastic_cv.fit(X_train,y_train)
```

Out[110… ElasticNetCV(cv=10)

In [112…
```
# selected alpha
elastic_cv.alpha_
```

Out[112… 0.00032049828688228085

In [113…
```
elastic_cv.l1_ratio_
```

Out[113… 0.5

In [114…
```
# Training ElastiNet model using above selected alpha
elastic = ElasticNet(elastic_cv.alpha_,elastic_cv.l1_ratio_)
elastic.fit(X_train,y_train)
```

```
C:\Users\aniyant\anaconda3\lib\site-packages\sklearn\utils\validation.py:67: FutureWarni
ng: Pass l1_ratio=0.5 as keyword args. From version 0.25 passing these as positional arg
uments will result in an error
  warnings.warn("Pass {} as keyword args. From version 0.25 "
```

Out[114… ElasticNet(alpha=0.00032049828688228085)

In [116…
```
# slope
elastic.coef_
```

Out[116…
```
array([0.01546633, 0.01904823, 0.00378117, 0.00313248, 0.01671488,
```

```
       0.07619495, 0.01391656])
```

In [117…

```
# intercept
elastic.intercept_
```

Out[117…

```
0.7181145693194342
```

In [118…

```
# R2 score
elastic.score(X_test,y_test)
```

Out[118…

```
0.8265118379982933
```

In [ ]: