In [287…
```python
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Dataset used : "https://www.kaggle.com/mohansacharya/graduate-admissions"

train_data = pd.read_csv('Admission_Predict.csv')
```

In [288…
```python
train_data.columns = map(lambda s : s.strip(),train_data.columns)
```

In [289…
```python
train_data.head()
```

Out[289…

	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

In [290…
```python
train_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 500 entries, 0 to 499
Data columns (total 9 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Serial No.         500 non-null    int64
 1   GRE Score          500 non-null    int64
 2   TOEFL Score        500 non-null    int64
 3   University Rating  500 non-null    int64
 4   SOP                500 non-null    float64
 5   LOR                500 non-null    float64
 6   CGPA               500 non-null    float64
 7   Research           500 non-null    int64
 8   Chance of Admit    500 non-null    float64
dtypes: float64(4), int64(5)
memory usage: 35.3 KB
```

In [291…
```python
train_data.rename(columns = {'University Rating' : 'UniversityRating','Chance of Admit' : 'ChanceofAdmit','Seri
```

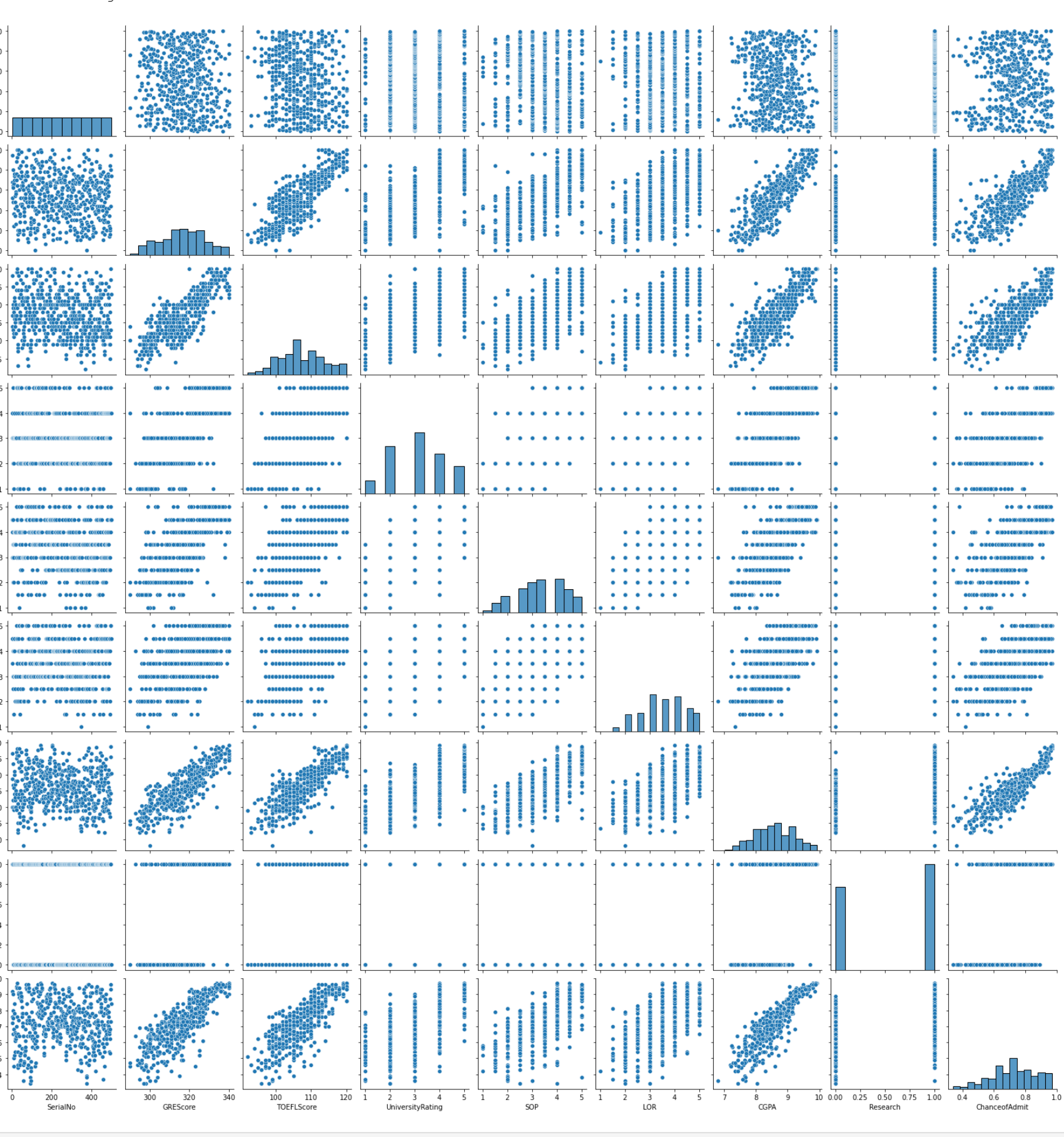
In [292…
```python
train_data.head()
```

Out[292…

	SerialNo	GREScore	TOEFLScore	UniversityRating	SOP	LOR	CGPA	Research	ChanceofAdmit
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

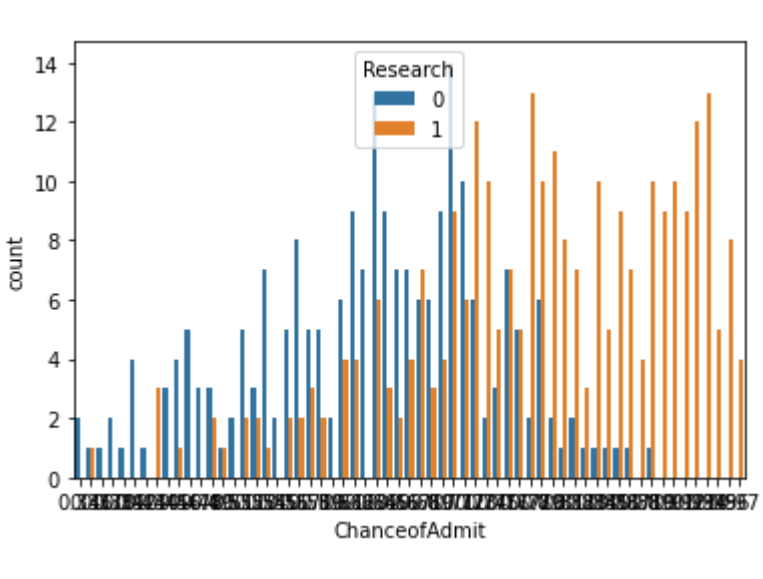
In [293…
```python
sns.pairplot(train_data)
```

Out[293…
```
<seaborn.axisgrid.PairGrid at 0x7fcefcabf5e0>
```



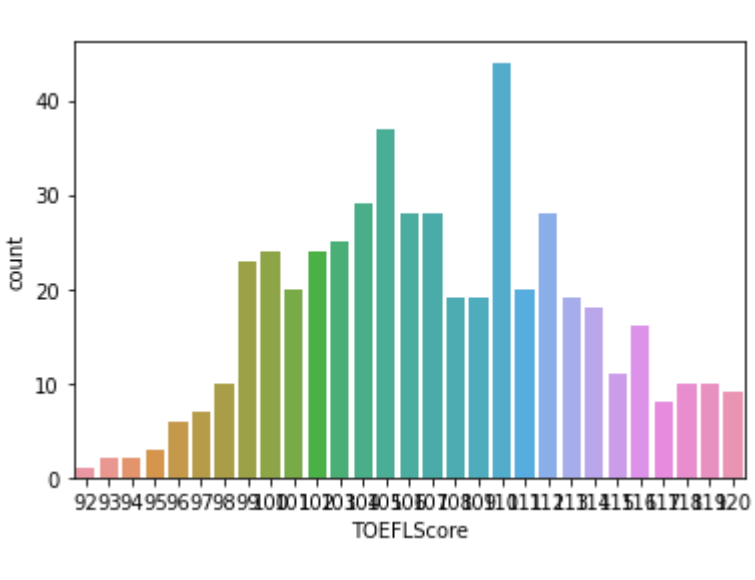
In [294…
```python
sns.countplot(x = 'ChanceofAdmit',data = train_data,hue = 'Research')
```

Out[294…
```
<AxesSubplot:xlabel='ChanceofAdmit', ylabel='count'>
```



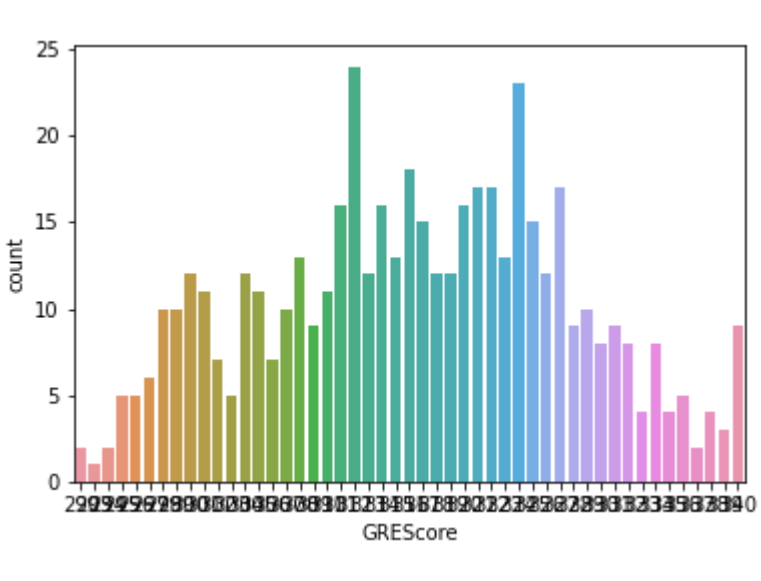
In [295…
```python
sns.countplot(x = 'TOEFLScore',data = train_data)
```

Out[295…
```
<AxesSubplot:xlabel='TOEFLScore', ylabel='count'>
```



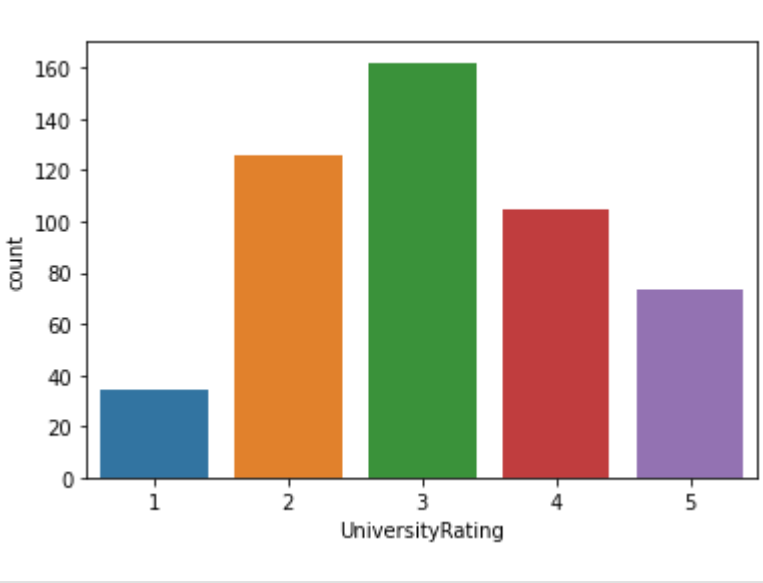
In [296…
```python
sns.countplot(x = 'GREScore',data = train_data)
```

Out[296…
```
<AxesSubplot:xlabel='GREScore', ylabel='count'>
```



In [297…
```python
sns.countplot(x = 'UniversityRating',data = train_data)
```

Out[297…
```
<AxesSubplot:xlabel='UniversityRating', ylabel='count'>
```



In [298…
```python
prob = train_data['ChanceofAdmit']
train_data.drop(['SerialNo'],axis = 1,inplace = True)
```

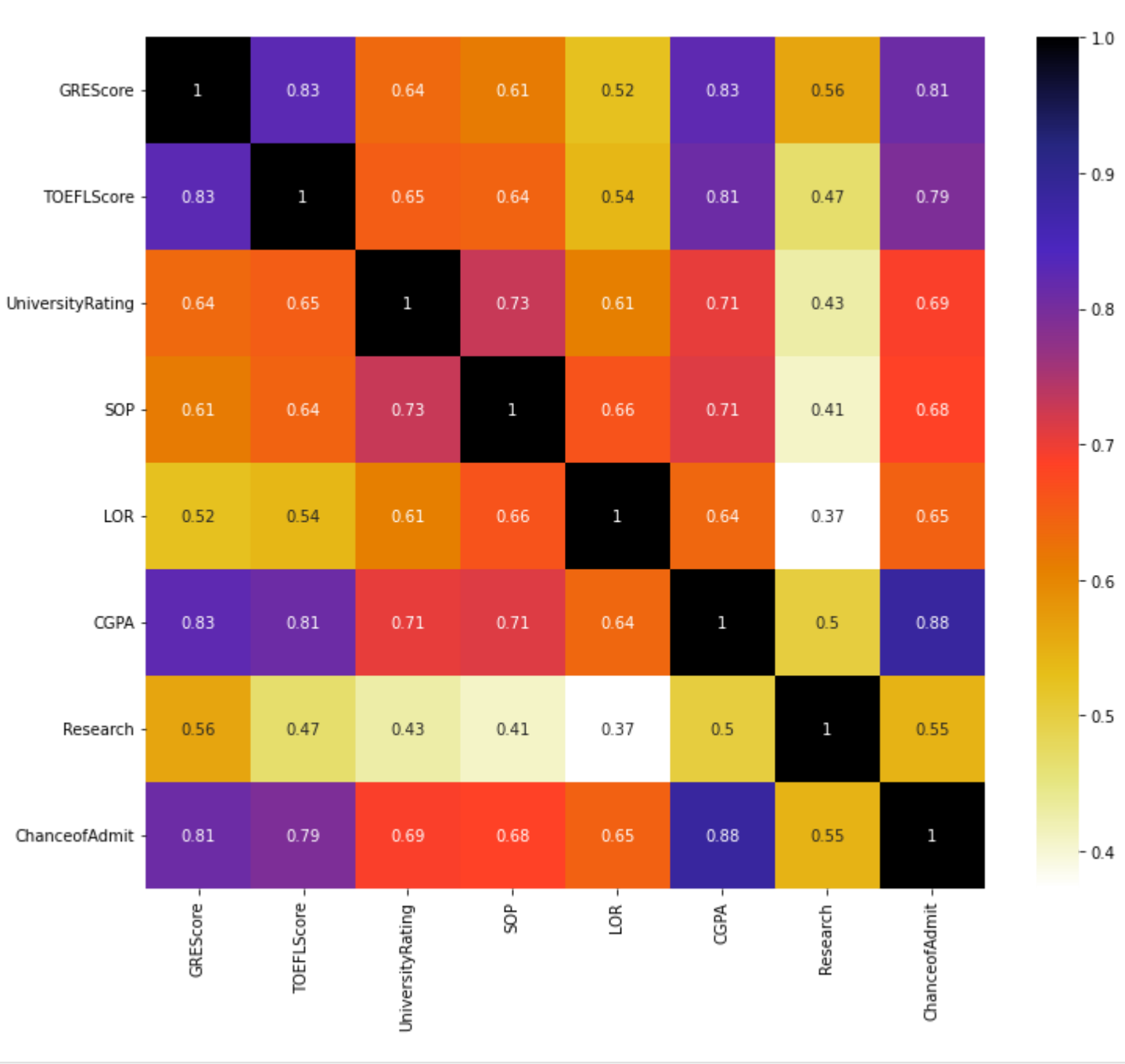
In [299…
```python
train_data.corr()
```

Out[299…

	GREScore	TOEFLScore	UniversityRating	SOP	LOR	CGPA	Research	ChanceofAdmit
GREScore	1.000000	0.827200	0.635376	0.613498	0.524679	0.825878	0.563398	0.810351
TOEFLScore	0.827200	1.000000	0.649799	0.644410	0.541563	0.810574	0.467012	0.792228
UniversityRating	0.635376	0.649799	1.000000	0.728024	0.608651	0.705254	0.427047	0.690132
SOP	0.613498	0.644410	0.728024	1.000000	0.663707	0.712154	0.408116	0.684137
LOR	0.524679	0.541563	0.608651	0.663707	1.000000	0.637469	0.372526	0.645365
CGPA	0.825878	0.810574	0.705254	0.712154	0.637469	1.000000	0.501311	0.882413
Research	0.563398	0.467012	0.427047	0.408116	0.372526	0.501311	1.000000	0.545871
ChanceofAdmit	0.810351	0.792228	0.690132	0.684137	0.645365	0.882413	0.545871	1.000000

In [300…
```python
from scipy.stats import pearsonr

cor = train_data.corr()
plt.figure(figsize = (12,10))
sns.heatmap(cor,annot = True,cmap = plt.cm.CMRmap_r)
```

Out[300…
```
<AxesSubplot:>
```

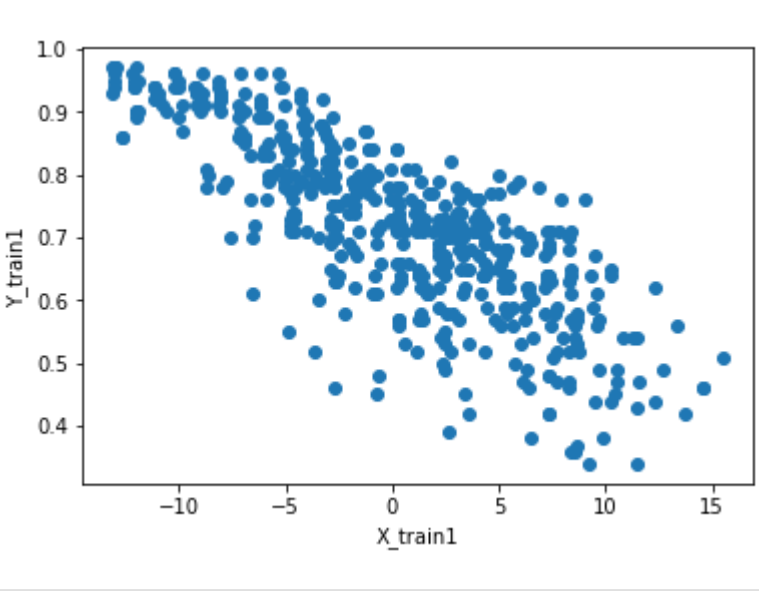


In [301…
```python
#CGPA,GREScore and TOEFLScore are highly correlated and one/two of them must be dropped
#But CGPA has a high correlation with 'Chance of Admit',so we will only consider 3 cases :
# 1) only GREScore is dropped
# 2) only TOEFLScore is dropped
# 3) both GREScore and TOEFLScore are dropped

train1 = train_data.drop(['GREScore','ChanceofAdmit'],axis = 1,inplace = False)
train2 = train_data.drop(['TOEFLScore','ChanceofAdmit'],axis = 1,inplace = False)
train3 = train_data.drop(['GREScore','TOEFLScore','ChanceofAdmit'],axis = 1,inplace = False)

from sklearn.model_selection import train_test_split

X_train1,X_test1,Y_train1,Y_test1 = train_test_split(train1,prob,train_size = 0.8)
X_train2,X_test2,Y_train2,Y_test2 = train_test_split(train2,prob,train_size = 0.8)
X_train3,X_test3,Y_train3,Y_test3 = train_test_split(train3,prob,train_size = 0.8)

X_train1.reset_index(inplace = True)
X_train2.reset_index(inplace = True)
X_train3.reset_index(inplace = True)


X_test1.reset_index(inplace = True)
X_test2.reset_index(inplace = True)
X_test3.reset_index(inplace = True)

X_train1.drop(['index'],axis = 1,inplace = True)
X_train2.drop(['index'],axis = 1,inplace = True)
X_train3.drop(['index'],axis = 1,inplace = True)
X_test1.drop(['index'],axis = 1,inplace = True)
X_test2.drop(['index'],axis = 1,inplace = True)
X_test3.drop(['index'],axis = 1,inplace = True)
```

```
/Users/vibhugarg/opt/anaconda3/lib/python3.8/site-packages/pandas/core/frame.py:4901: SettingWithCopyWarning:
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/user_guide/indexing.html#ret
urning-a-view-versus-a-copy
  return super().drop(
```

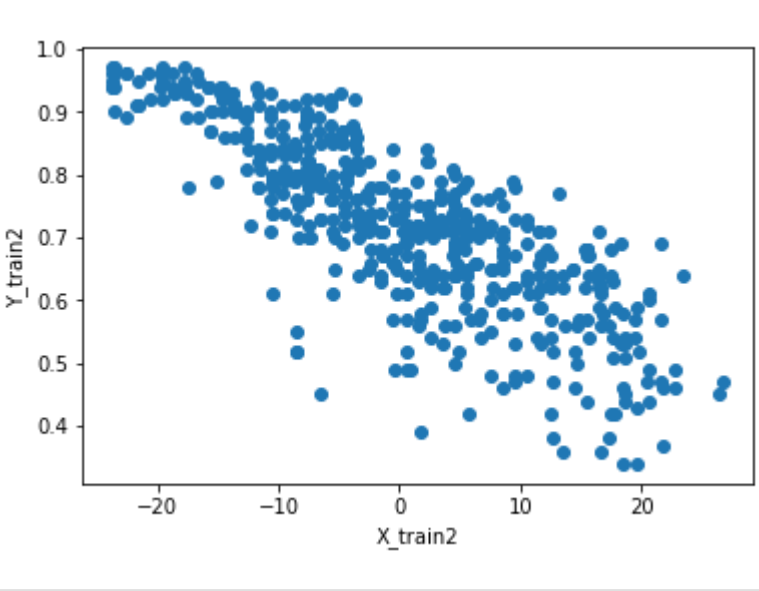
In [303…
```python
from sklearn.decomposition import PCA

pca = PCA(n_components = 1)

pcaData1 = pca.fit_transform(train1)
plt.scatter(pcaData1,prob)
plt.xlabel('X_train1')
plt.ylabel('Y_train1')
```

Out[303…
```
Text(0, 0.5, 'Y_train1')
```



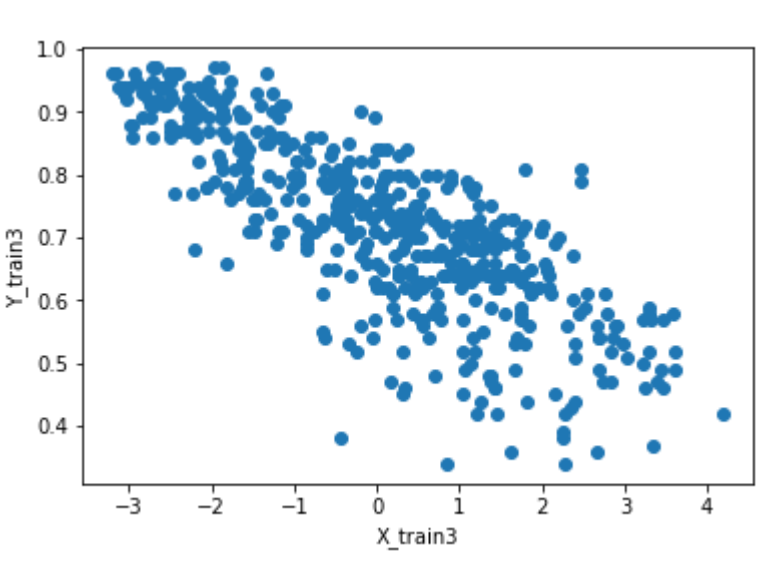
In [304…
```python
pcaData2 = pca.fit_transform(train2)
plt.scatter(pcaData2,prob)
plt.xlabel('X_train2')
plt.ylabel('Y_train2')
```

Out[304…
```
Text(0, 0.5, 'Y_train2')
```



In [305…
```python
pcaData3 = pca.fit_transform(train3)
plt.scatter(pcaData3,prob)
plt.xlabel('X_train3')
plt.ylabel('Y_train3')
```

Out[305…
```
Text(0, 0.5, 'Y_train3')
```



In [306…
```python
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import GridSearchCV

regr1 = LinearRegression(normalize = True)
regr2 = LinearRegression(normalize = True)
regr3 = LinearRegression(normalize = True)
model1 = regr1
model2 = regr2
model3 = regr3

model1.fit(X_train1,Y_train1)
model2.fit(X_train2,Y_train2)
model3.fit(X_train3,Y_train3)

print("Model1 Score :",model1.score(X_test1,Y_test1))
print("Model2 Score :",model2.score(X_test2,Y_test2))
print("Model3 Score :",model3.score(X_test3,Y_test3))
```

```
Model1 Score : 0.828677283221296
Model2 Score : 0.8019405873866431
Model3 Score : 0.8446347143626188
```

In [307…
```python
from sklearn.linear_model import Lasso

regr4 = GridSearchCV(Lasso(),{'alpha' : [0.1,0.5,1,10,100,1000],'normalize' : [True,False]},cv = 4,return_trair
regr5 = GridSearchCV(Lasso(),{'alpha' : [0.1,0.5,1,10,100,1000],'normalize' : [True,False]},cv = 4,return_trair
regr6 = GridSearchCV(Lasso(),{'alpha' : [0.1,0.5,1,10,100,1000],'normalize' : [True,False]},cv = 4,return_trair

regr4.fit(X_train1,Y_train1)
regr5.fit(X_train2,Y_train2)
regr6.fit(X_train3,Y_train3)
```

Out[307…
```
GridSearchCV(cv=4, estimator=Lasso(),
             param_grid={'alpha': [0.1, 0.5, 1, 10, 100, 1000],
                         'normalize': [True, False]},
             return_train_score=True, scoring='neg_mean_absolute_error')
```

In [308…
```python
from sklearn.linear_model import Ridge

regr7 = GridSearchCV(Ridge(),{'alpha' : [0.1,0.5,1,10,100,1000],'normalize' : [True,False]},cv = 4,return_trair
regr8 = GridSearchCV(Ridge(),{'alpha' : [0.1,0.5,1,10,100,1000],'normalize' : [True,False]},cv = 4,return_trair
regr9 = GridSearchCV(Ridge(),{'alpha' : [0.1,0.5,1,10,100,1000],'normalize' : [True,False]},cv = 4,return_trair

regr7.fit(X_train1,Y_train1)
regr8.fit(X_train2,Y_train2)
regr9.fit(X_train3,Y_train3)
```

Out[308…
```
GridSearchCV(cv=4, estimator=Ridge(),
             param_grid={'alpha': [0.1, 0.5, 1, 10, 100, 1000],
                         'normalize': [True, False]},
             return_train_score=True, scoring='neg_mean_absolute_error')
```

In [309…
```python
from sklearn.tree import DecisionTreeRegressor

regr10 = GridSearchCV(DecisionTreeRegressor(),{'max_depth' : [3,4,5,6,7,8,9]},return_train_score = True,cv = 4,
regr11 = GridSearchCV(DecisionTreeRegressor(),{'max_depth' : [3,4,5,6,7,8,9]},return_train_score = True,cv = 4,
regr12 = GridSearchCV(DecisionTreeRegressor(),{'max_depth' : [3,4,5,6,7,8,9]},return_train_score = True,cv = 4,

regr10.fit(X_train1,Y_train1)
regr11.fit(X_train2,Y_train2)
regr12.fit(X_train3,Y_train3)
```

Out[309…
```
GridSearchCV(cv=4, estimator=DecisionTreeRegressor(),
             param_grid={'max_depth': [3, 4, 5, 6, 7, 8, 9]},
             return_train_score=True, scoring='neg_mean_absolute_error')
```

In [310…
```python
from sklearn.ensemble import RandomForestRegressor

regr13 = GridSearchCV(RandomForestRegressor(),{'max_depth' : [2,4,6,8,10,None],'n_estimators' : [10,50,100,250]
regr14 = GridSearchCV(RandomForestRegressor(),{'max_depth' : [2,4,6,8,10,None],'n_estimators' : [10,50,100,250]
regr15 = GridSearchCV(RandomForestRegressor(),{'max_depth' : [2,4,6,8,10,None],'n_estimators' : [10,50,100,250]

regr13.fit(X_train1,Y_train1)
regr14.fit(X_train2,Y_train2)
regr15.fit(X_train3,Y_train3)
```

Out[310…
```
GridSearchCV(cv=4, estimator=RandomForestRegressor(),
             param_grid={'max_depth': [2, 4, 6, 8, 10, None],
                         'n_estimators': [10, 50, 100, 250]},
             return_train_score=True, scoring='neg_mean_absolute_error')
```

In [311…
```python
from sklearn.svm import SVR

regr16 = GridSearchCV(SVR(),{'C' : [0.1,0.5,1,10,100,1000],'kernel' : ['linear','rbf']},cv = 4,return_train_scc
regr17 = GridSearchCV(SVR(),{'C' : [0.1,0.5,1,10,100,1000],'kernel' : ['linear','rbf']},cv = 4,return_train_scc
regr18 = GridSearchCV(SVR(),{'C' : [0.1,0.5,1,10,100,1000],'kernel' : ['linear','rbf']},cv = 4,return_train_scc

nd1 = normalize(X_train1)
nd2 = normalize(X_train2)
nd3 = normalize(X_train3)

regr16.fit(nd1,Y_train1)
regr17.fit(nd2,Y_train2)
regr18.fit(nd3,Y_train3)
```

Out[311…
```
GridSearchCV(cv=4, estimator=SVR(),
             param_grid={'C': [0.1, 0.5, 1, 10, 100, 1000],
                         'kernel': ['linear', 'rbf']},
             return_train_score=True, scoring='neg_mean_absolute_error')
```

In [312…
```python
from sklearn.neighbors import KNeighborsRegressor

regr19 = GridSearchCV(KNeighborsRegressor(),{'n_neighbors' : [3,5,7,9,11],'p' : [1,2,3,4,5]},cv = 4,return_trai
regr20 = GridSearchCV(KNeighborsRegressor(),{'n_neighbors' : [3,5,7,9,11],'p' : [1,2,3,4,5]},cv = 4,return_trai
regr21 = GridSearchCV(KNeighborsRegressor(),{'n_neighbors' : [3,5,7,9,11],'p' : [1,2,3,4,5]},cv = 4,return_trai

regr19.fit(X_train1,Y_train1)
regr20.fit(X_train2,Y_train2)
regr21.fit(X_train3,Y_train3)
```

Out[312…
```
GridSearchCV(cv=4, estimator=KNeighborsRegressor(),
             param_grid={'n_neighbors': [3, 5, 7, 9, 11], 'p': [1, 2, 3, 4, 5]},
             return_train_score=True, scoring='neg_mean_absolute_error')
```

In [316…
```python
models1 = [regr1,regr4,regr7,regr10,regr13,regr16,regr19]
models2 = [regr2,regr5,regr8,regr11,regr14,regr17,regr20]
models3 = [regr3,regr6,regr9,regr12,regr15,regr18,regr21]

train_score1 = [models1[i].best_score_ for i in range(1,7)]
train_score2 = [models2[i].best_score_ for i in range(1,7)]
train_score3 = [models3[i].best_score_ for i in range(1,7)]

def cal_NMAE(regr,x,y):
    total = len(x)
    mae = 0.0

    result = regr.predict(x)
    mae = sum(abs(result - y))

    mae /= total
    mae *= -1
    return mae

train_score1.append(cal_MAE(regr1,X_train1,Y_train1))
train_score2.append(cal_MAE(regr2,X_train2,Y_train2))
train_score3.append(cal_MAE(regr3,X_train3,Y_train3))

test_score1 = [models1[i].score(X_test1,Y_test1) for i in range(1,7)]
test_score2 = [models2[i].score(X_test2,Y_test2) for i in range(1,7)]
test_score3 = [models3[i].score(X_test3,Y_test3) for i in range(1,7)]

test_score1.append(cal_MAE(regr1,X_test1,Y_test1))
test_score2.append(cal_MAE(regr2,X_test2,Y_test2))
test_score3.append(cal_MAE(regr3,X_test3,Y_test3))
```

In [317…
```python
# On data with no GRE Score
df1 = pd.DataFrame({'Model' : ['Lasso Regression','Ridge Regression','Decision Tree','Random Forest','SVM','KNN
df1
```

Out[317…

	Model	Train Score (MAE)	Test Score (MAE)
0	Lasso Regression	-0.068676	-0.072879
1	Ridge Regression	-0.044562	-0.044118
2	Decision Tree	-0.053434	-0.047652
3	Random Forest	-0.046365	-0.044993
4	SVM	-0.075688	-3.158275
5	KNN	-0.051907	-0.051414
6	Linear Regression	-0.043665	-0.044112

In [318…
```python
# On data with no TOEFL Score
df2 = pd.DataFrame({'Model' : ['Lasso Regression','Ridge Regression','Decision Tree','Random Forest','SVM','KNN
df2
```

Out[318…

	Model	Train Score (MAE)	Test Score (MAE)
0	Lasso Regression	-0.061899	-0.070787
1	Ridge Regression	-0.044169	-0.041071
2	Decision Tree	-0.053588	-0.050099
3	Random Forest	-0.046935	-0.042531
4	SVM	-0.068199	-0.104199
5	KNN	-0.051522	-0.056478
6	Linear Regression	-0.043574	-0.041020

In [319…
```python
# On data with no GRE and TOEFL Score
df3 = pd.DataFrame({'Model' : ['Lasso Regression','Ridge Regression','Decision Tree','Random Forest','SVM','KNN
df3
```

Out[319…

	Model	Train Score (MAE)	Test Score (MAE)
0	Lasso Regression	-0.110684	-0.108791
1	Ridge Regression	-0.047397	-0.039879
2	Decision Tree	-0.051728	-0.046398
3	Random Forest	-0.048476	-0.044258
4	SVM	-0.068740	-5.645416
5	KNN	-0.056405	-0.047964
6	Linear Regression	-0.046316	-0.039881

In [320…
```python
print("On data with no GRE Score, best model : Linear Regression , with MAE : 0.044112")
print("On data with no TOEFL Score, best model : Linear Regression , with MAE : 0.041020")
print("On data with no GRE Score, best model : Ridge Regression , with MAE : 0.039881")
```

```
On data with no GRE Score, best model : Linear Regression , with MAE : 0.044112
On data with no TOEFL Score, best model : Linear Regression , with MAE : 0.041020
On data with no GRE Score, best model : Ridge Regression , with MAE : 0.039881
```

In [321…
```python
# Overall, the optimal choice for the dataset is with no GRE and TOEFL and the optimal model : Ridge Regression
model = regr9
```

In [ ]: