World University

Admission open for 2023

REGISTER NOW!

CALL NOW: +123-456-7890

Importing Required Libraries

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from warnings import filterwarnings
filterwarnings('ignore')
from scipy import stats
from sklearn.model_selection import train_test_split
import statsmodels.api as sma
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score , mean_squared_error
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor , AdaBoostRegressor , GradientBoostingRegressor
from sklearn.neighbors import KNeighborsRegressor
from xgboost import XGBRegressor
from catboost import CatBoostRegressor
from sklearn.linear_model import LinearRegression
```

In [2]:

```
plt.rcParams['figure.figsize'] = [15,8]
```

Loading dataset and reading first five rows

In [3]:

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

df.head()
```

Out[3]:

	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

Checking for shape and dimension

In [4]:

```
print(f'The dataset has {df.shape[0]} rows and {df.shape[1]} columns')
```

```
The dataset has 400 rows and 9 columns
```

In [5]:

```
print(f'The dataset is {df.ndim} dimensions')
```

```
The dataset is 2 dimensions
```

Checking for info

In [6]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 400 entries, 0 to 399
Data columns (total 9 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   Serial No.         400 non-null    int64  
 1   GRE Score          400 non-null    int64  
 2   TOEFL Score        400 non-null    int64  
 3   University Rating  400 non-null    int64  
 4   SOP                400 non-null    float64
 5   LOR                400 non-null    float64
 6   CGPA               400 non-null    float64
 7   Research           400 non-null    int64  
 8   Chance of Admit    400 non-null    float64
dtypes: float64(4), int64(5)
memory usage: 28.2 KB
```

Checking for missing values

In [7]:

```
missing_values = pd.DataFrame({'Number of missing values' : df.isnull().sum(),
                              'Percentage of missing values' : (df.isnull().sum() / len(df)) * 100})

missing_values
```

Out[7]:

	Number of missing values	Percentage of missing values
Serial No.	0	0.0
GRE Score	0	0.0
TOEFL Score	0	0.0
University Rating	0	0.0
SOP	0	0.0
LOR	0	0.0
CGPA	0	0.0
Research	0	0.0
Chance of Admit	0	0.0

Dropping of irrelevant columns

We can drop serial no as it is unique identifer

In [8]:

```
df.drop(columns = 'Serial No.' , inplace = True)
```

Univariate Analysis

In [9]:

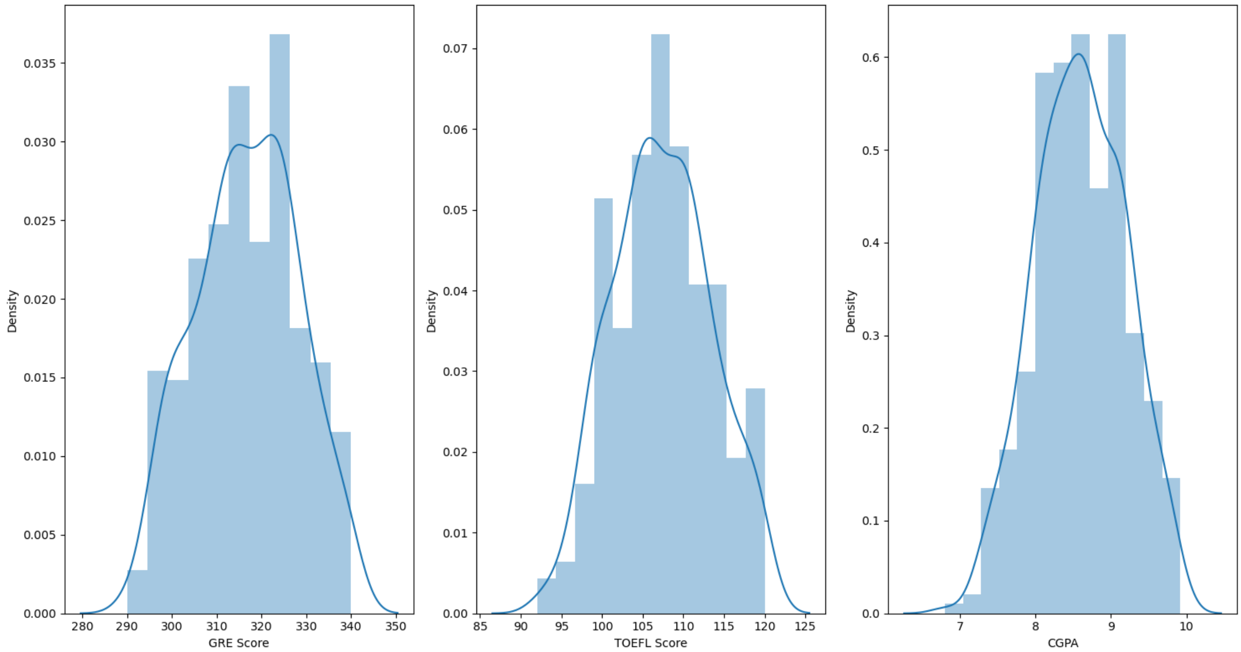
```
num_cols = ['GRE Score','TOEFL Score','CGPA']
cat_cols = ['University Rating','SOP','LOR ','Research']
```

In [10]:

```
f , ax = plt.subplots(1,3)

for i,v in zip(num_cols , ax.flatten()):
    sns.distplot(df[i] , ax = v)

plt.tight_layout()
plt.show()
```

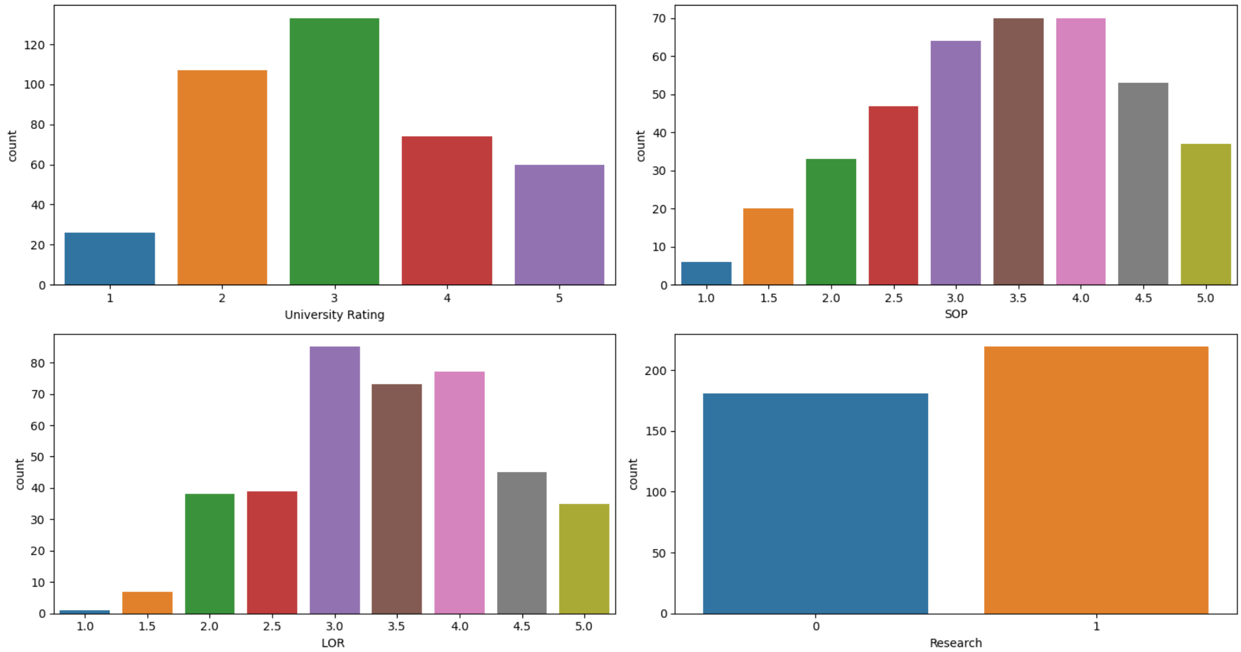


In [11]:

```
f , ax = plt.subplots(2,2)

for i , v in zip(cat_cols , ax.flatten()):
    sns.countplot(df[i] , ax = v)

plt.tight_layout()
plt.show()
```



Bivariate Analysis

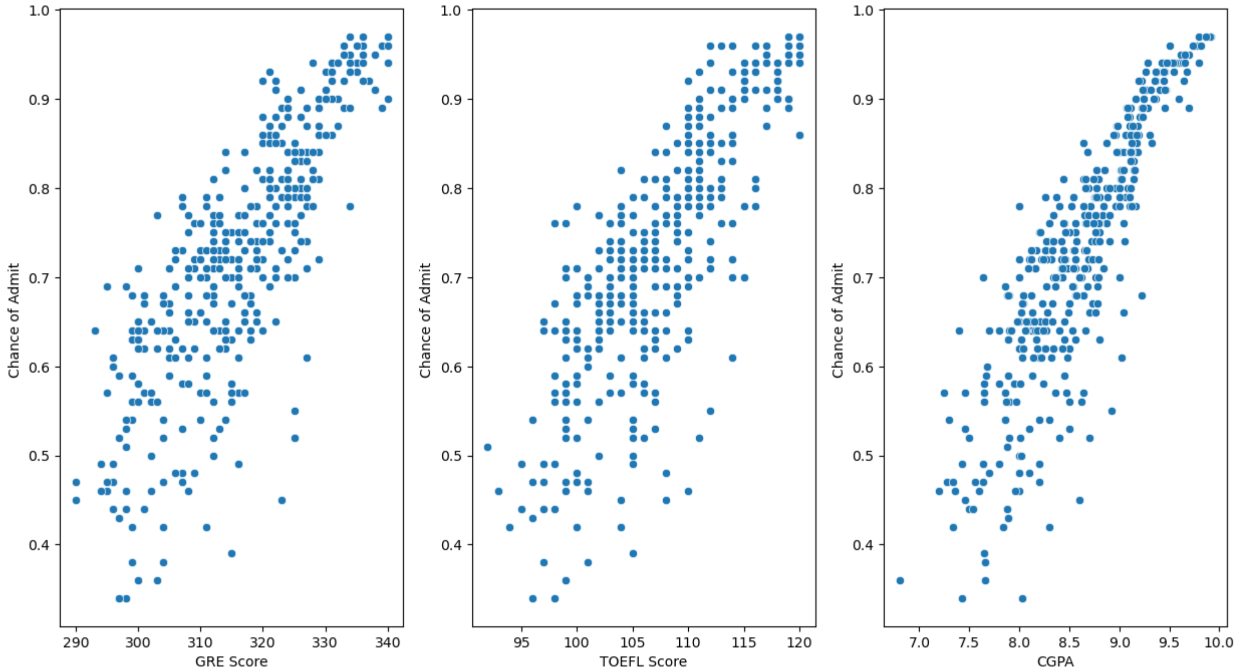
Num vs Num - Scatter Plot

In [12]:

```
f ,ax = plt.subplots(1,3)

for i,v in zip(num_cols , ax.flatten()):
    sns.scatterplot(x = df[i] , y = df['Chance of Admit '] , ax = v)

plt.tight_layout
plt.show()
```



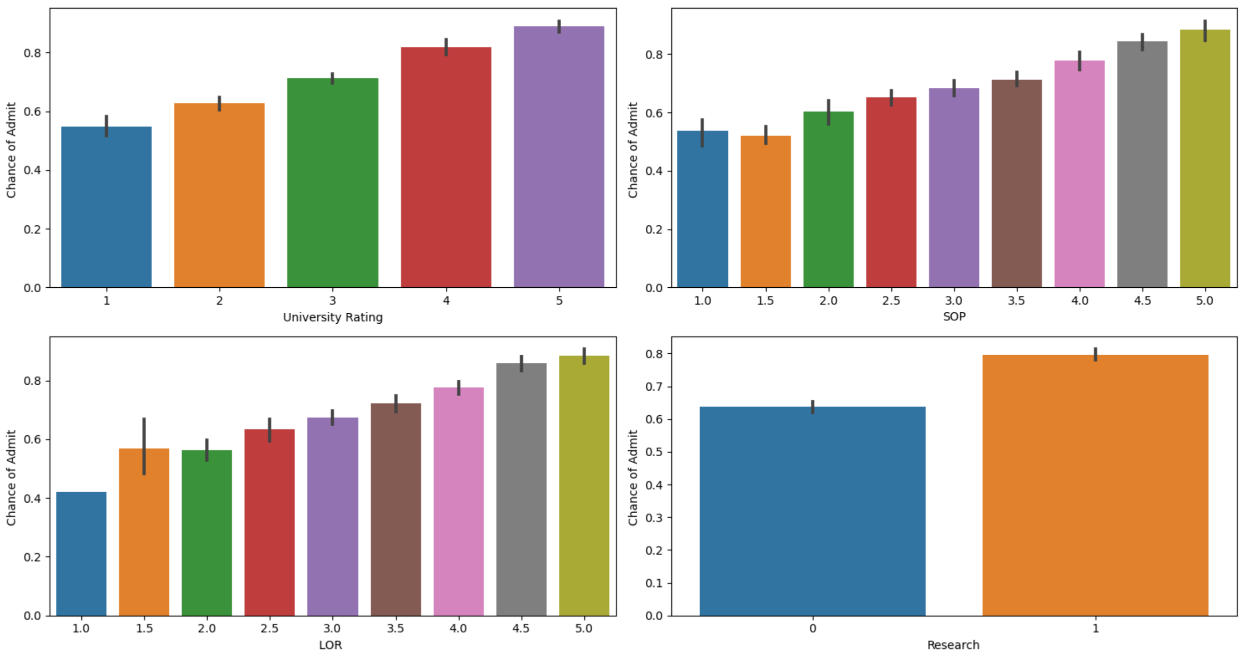
Cat vs Num - Barplot

In [13]:

```
f , ax = plt.subplots(2,2)

for i,v in zip(cat_cols , ax.flatten()):
    sns.barplot(x = df[i] , y = df['Chance of Admit '] , ax = v)

plt.tight_layout()
plt.show()
```



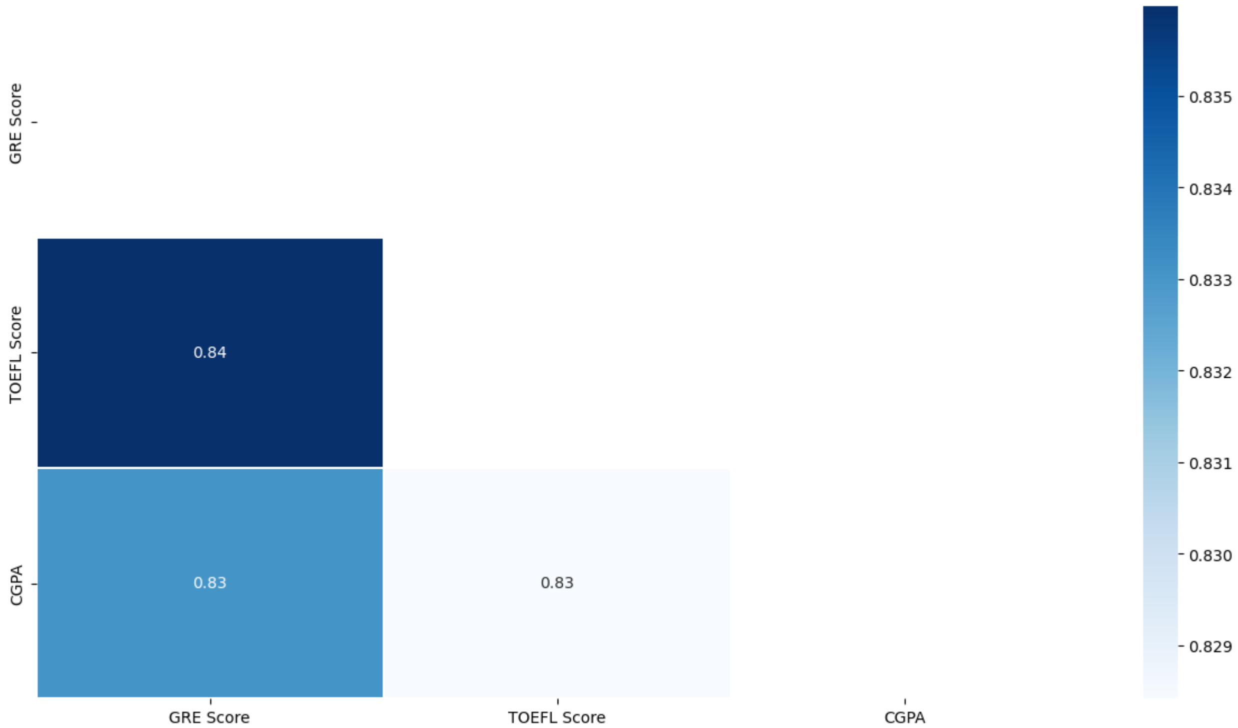
Multivariate Analysis

In [14]:

```
sns.heatmap(df[num_cols].corr() , annot = True , mask = np.triu(df[num_cols].corr()) , cmap = 'Blues',linewidths=0.1
```

Out[14]:

<AxesSubplot:>



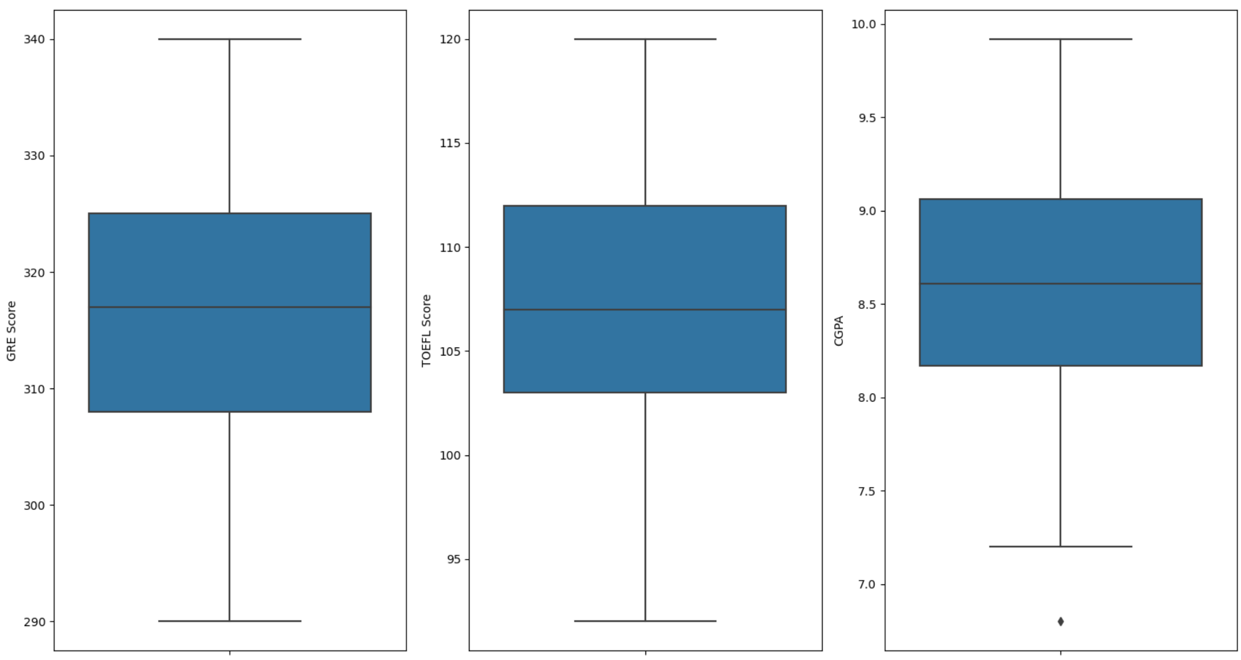
Checking and Treating of Outliers

In [15]:

```
f , ax = plt.subplots(1,3)

for i,v in zip(num_cols , ax.flatten()):
    sns.boxplot(y = df[i] , ax = v)

plt.tight_layout()
plt.show()
```



From above boxplot it is clearly evident that there are no outliers present in the given dataset

Performing Statistical Test

H0 : There is no significant relationship between dependent and independent variable

Ha : There is significant relationship between dependent and independent variable

consider significane level 0.05

In [16]:

```
statistical_result = pd.DataFrame(columns = ['Columns','Pvalue','Remarks'])
```

In [17]:

```
# Num vs Num - Pearsonr test


for i in num_cols:
    stat , pval = stats.pearsonr(df[i] , df['Chance of Admit '])

    statistical_result = statistical_result.append({'Columns' : i,
                                                    'Pvalue': '{:f}'.format(pval),
                                                    'Remarks' : 'Reject H0' if pval < 0.05 else 'Failed to reject H0
                                                  ignore_index=True)
```

In [18]:

```
# Num vs Cat - Anova test

for i in cat_cols:
    groups = [df.loc[df[i] == subclass , 'Chance of Admit '] for subclass in df[i].unique()]

    stat , pval = stats.f_oneway(*groups)

    statistical_result = statistical_result.append({'Columns' : i,
                                                    'Pvalue': '{:f}'.format(pval),
                                                    'Remarks' : 'Reject H0' if pval < 0.05 else 'Failed to reject H0
                                                  ignore_index=True)
```

In [19]:

```
statistical_result
```

Out[19]:

	Columns	Pvalue	Remarks
0	GRE Score	0.000000	Reject H0
1	TOEFL Score	0.000000	Reject H0
2	CGPA	0.000000	Reject H0
3	University Rating	0.000000	Reject H0
4	SOP	0.000000	Reject H0
5	LOR	0.000000	Reject H0
6	Research	0.000000	Reject H0

From above statistical result it is clearly evident that all the independent variables are significant to dependent variable chance of admit

Splitting of data into 70% train and 30% test

In [20]:

```
x = df.drop(columns = 'Chance of Admit ')
y = df['Chance of Admit ']

xtrain , xtest , ytrain , ytest = train_test_split(x,y,test_size = 0.30 , random_state = 24)
```

Building Models

Linear Regression using statsmodel

In [21]:

```
model = sma.OLS(ytrain , sma.add_constant(xtrain)).fit()

model.summary()
```

Out[21]:

OLS Regression Results

Dep. Variable:	Chance of Admit	**R-squared:**	0.816
Model:	OLS	**Adj. R-squared:**	0.811
Method:	Least Squares	**F-statistic:**	172.2
Date:	Tue, 01 Aug 2023	**Prob (F-statistic):**	4.62e-96
Time:	19:55:30	**Log-Likelihood:**	381.25
No. Observations:	280	**AIC:**	-746.5
Df Residuals:	272	**BIC:**	-717.4
Df Model:	7		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	-1.3172	0.149	-8.815	0.000	-1.611	-1.023
GRE Score	0.0018	0.001	2.531	0.012	0.000	0.003
TOEFL Score	0.0038	0.001	2.992	0.003	0.001	0.006
University Rating	0.0055	0.005	1.047	0.296	-0.005	0.016
SOP	-0.0057	0.006	-0.882	0.379	-0.018	0.007
LOR	0.0259	0.007	3.869	0.000	0.013	0.039
CGPA	0.1105	0.014	7.737	0.000	0.082	0.139
Research	0.0245	0.009	2.607	0.010	0.006	0.043

Omnibus:	54.280	**Durbin-Watson:**	1.870
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	95.914
Skew:	-1.052	**Prob(JB):**	1.49e-21
Kurtosis:	4.947	**Cond. No.**	1.33e+04

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.33e+04. This might indicate that there are
strong multicollinearity or other numerical problems.

In [22]:

```
pred_train = model.predict(sma.add_constant(xtrain))
pred_test = model.predict(sma.add_constant(xtest))

r2_train = r2_score(ytrain,pred_train)
r2_test = r2_score(ytest,pred_test)
rmse_train = np.sqrt(mean_squared_error(ytrain,pred_train))
rmse_test = np.sqrt(mean_squared_error(ytest,pred_test))
```

In [23]:

```
# Creating a dataframe to store values of train and test data

performance_df = pd.DataFrame(columns = ['Model','Train R2','Test R2','Train RMSE','Test RMSE','Remarks'])

# Creating a user defined function that will create a model and will append the values to dataframe

def model_performnace(model , name , xtrain = xtrain , xtest = xtest , ytrain = ytrain , ytest = ytest ):

    global performance_df

    model = model.fit(xtrain , ytrain)

    pred_train = model.predict(xtrain)
    pred_test = model.predict(xtest)

    r2_train = r2_score(ytrain , pred_train)
    r2_test = r2_score(ytest , pred_test)
    rmse_train = np.sqrt(mean_squared_error(ytrain , pred_train))
    rmse_test = np.sqrt(mean_squared_error(ytest , pred_test))

    def remark(train,test):
        if abs(train - test) > 0.1 or train > 0.95:
            return 'Over Fit'
        elif train > 0.81 and test > 0.76:
            return 'Good Fit'
        else :
            return 'Under Fit'

    performance_df = performance_df.append({'Model':name ,
                                           'Train R2':r2_train,
                                           'Test R2':r2_test,
                                           'Train RMSE':'{:f}'.format(rmse_train),
                                           'Test RMSE':'{:f}'.format(rmse_test),
                                           'Remarks':remark(r2_train,r2_test)},
                                          ignore_index=True)

```

In [24]:

```
# Creating a user defined function that will highlight the rows which are good fit

def highlight(df):
    color_green = ['background-color : lightgreen']*len(df)
    color_white = ['background-color : white']*len(df)

    if df['Remarks'] == 'Good Fit':
        return color_green

    else:
        return color_white
```

In [25]:

```
# Appending values of our base model to dataframe

performance_df = performance_df.append({'Model':'Base Model','Train R2':r2_train,'Test R2':r2_test,
                                       'Train RMSE':'{:f}'.format(rmse_train),'Test RMSE':'{:f}'.format(rmse_test),
                                        'Remarks':'Base'},ignore_index=True)

performance_df
```

Out[25]:

	Model	Train R2	Test R2	Train RMSE	Test RMSE	Remarks
0	Base Model	0.815868	0.76753	0.062004	0.066146	Base

In [26]:

```
# Decision tree

model_performnace(DecisionTreeRegressor() , 'DecisionTRee')
```

In [27]:

```
# Random Forest

model_performnace(RandomForestRegressor() , 'RandomForest')
```

In [28]:

```
# KNN

model_performnace(KNeighborsRegressor() , 'KNN')
```

In [29]:

```
# AdaBoost

model_performnace(AdaBoostRegressor() , 'AdaBoost')
```

In [30]:

```
# GradientBoost

model_performnace(GradientBoostingRegressor() , 'GradientBoosting')
```

In [31]:

```
# XGBoost

model_performnace(XGBRegressor() , 'XGBoost')
```

In [32]:

```
# Catboost

model_performnace(CatBoostRegressor() , 'Catboost')
```

```
63:     learn: 0.0627331        total: 232ms    remaining: 3.4s
64:     learn: 0.0624420        total: 233ms    remaining: 3.36s
65:     learn: 0.0621249        total: 234ms    remaining: 3.31s
66:     learn: 0.0617638        total: 235ms    remaining: 3.27s
67:     learn: 0.0614712        total: 236ms    remaining: 3.23s
68:     learn: 0.0611334        total: 237ms    remaining: 3.2s
69:     learn: 0.0608349        total: 238ms    remaining: 3.16s
70:     learn: 0.0604866        total: 239ms    remaining: 3.12s
71:     learn: 0.0602221        total: 239ms    remaining: 3.09s
72:     learn: 0.0599786        total: 240ms    remaining: 3.05s
73:     learn: 0.0596738        total: 241ms    remaining: 3.02s
74:     learn: 0.0594794        total: 242ms    remaining: 2.99s
75:     learn: 0.0593250        total: 243ms    remaining: 2.96s
76:     learn: 0.0590900        total: 244ms    remaining: 2.93s
77:     learn: 0.0588792        total: 245ms    remaining: 2.9s
78:     learn: 0.0586299        total: 246ms    remaining: 2.87s
79:     learn: 0.0583325        total: 247ms    remaining: 2.85s
80:     learn: 0.0581256        total: 248ms    remaining: 2.82s
81:     learn: 0.0579609        total: 249ms    remaining: 2.79s
82:     learn: 0.0577826        total: 250ms    remaining: 2.77s
```

In [33]:

```
performance_df
```

Out[33]:

	Model	Train R2	Test R2	Train RMSE	Test RMSE	Remarks
0	Base Model	0.815868	0.767530	0.062004	0.066146	Base
1	DecisionTRee	1.000000	0.614393	0.000000	0.085191	Over Fit
2	RandomForest	0.967439	0.737626	0.026074	0.070272	Over Fit
3	KNN	0.811396	0.632535	0.062752	0.083163	Over Fit
4	AdaBoost	0.839607	0.698930	0.057869	0.075276	Over Fit
5	GradientBoosting	0.940036	0.711380	0.035383	0.073703	Over Fit
6	XGBoost	0.999807	0.655497	0.002007	0.080523	Over Fit
7	Catboost	0.987785	0.717429	0.015970	0.072926	Over Fit

From the given dataframe, it is evident that linear regression, being a simple algorithm, outperforms other more complex algorithms. This serves as a great illustration of how complex algorithms can lead to overfitting when dealing with small and straightforward datasets. In such cases, the simplicity and generalization ability of linear regression prove to be advantageous.