For Master's Degree Admission in worldclass university Prediction based on various scores as shown below:

For Master's studies dataset is a dataset which describes the probability of selections for world class students such as Indian Students are dependent on the following Scores:

1. GRE Score
2. TOEFL Score
3. University Rating (out of 5)
4. SOP (Statement of Purpose)and LOR(Letter of Recommendation) Strength(out of 5)
5. Undergraduate CGPA (Also called CPI) (out of 10)
6. Research Experience (0 for no experience and 1 for having an experience)
7. Chance of Admit (out of 1)

In [1]:
```
# Import the libraries

import pandas as pd
import numpy as np
```

In [2]:
```
# Read the dataset

df = pd.read_csv(r"https://raw.githubusercontent.com/srinivasav22/Graduate-Admission-P
```

In [3]:
```
# Fetch the dataset

df
```

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
...	...	...	...	...	...	...	...	...	...
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

Perform the EDA on To_Graduate_Chance_of_Admission_Prediction

In [4]:
```
# Let us check the number of columns and number of rows in the datset
# Observation 1 : There are 500 rows and 9 variables in the datsets
df.shape
```

Out[4]:
```
(500, 9)
```

In [5]:
```
#Let us understand the basic information from the dataset
# Observation 2 : There are 500 rows and 9 features in the dataset
# Observation 3 : All the features are numeric features with integers and float dtypes
df.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 [6]:
```
# Basic statistics of the dataset
df.describe()
```

Out[6]:

	Serial No.	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research
count	500.000000	500.000000	500.000000	500.000000	500.000000	500.00000	500.000000	500.000000
mean	250.500000	316.472000	107.192000	3.114000	3.374000	3.48400	8.576440	0.560000
std	144.481833	11.295148	6.081868	1.143512	0.991004	0.92545	0.604813	0.496884
min	1.000000	290.000000	92.000000	1.000000	1.000000	1.00000	6.800000	0.000000
25%	125.750000	308.000000	103.000000	2.000000	2.500000	3.00000	8.127500	0.000000
50%	250.500000	317.000000	107.000000	3.000000	3.500000	3.50000	8.560000	1.000000
75%	375.250000	325.000000	112.000000	4.000000	4.000000	4.00000	9.040000	1.000000
max	500.000000	340.000000	120.000000	5.000000	5.000000	5.00000	9.920000	1.000000

In [7]:
```
# Transposing the basic statistics of the datset for more clarity
# Looks like there are some outliers in the dataset
df.describe().T
```

Out[7]:

	count	mean	std	min	25%	50%	75%	max
Serial No.	500.0	250.50000	144.481833	1.00	125.7500	250.50	375.25	500.00
GRE Score	500.0	316.47200	11.295148	290.00	308.0000	317.00	325.00	340.00
TOEFL Score	500.0	107.19200	6.081868	92.00	103.0000	107.00	112.00	120.00
University Rating	500.0	3.11400	1.143512	1.00	2.0000	3.00	4.00	5.00
SOP	500.0	3.37400	0.991004	1.00	2.5000	3.50	4.00	5.00
LOR	500.0	3.48400	0.925450	1.00	3.0000	3.50	4.00	5.00
CGPA	500.0	8.57644	0.604813	6.80	8.1275	8.56	9.04	9.92
Research	500.0	0.56000	0.496884	0.00	0.0000	1.00	1.00	1.00
Chance of Admit	500.0	0.72174	0.141140	0.34	0.6300	0.72	0.82	0.97

In [8]:
```
# Checking the features in the dataset
df.columns
```

Out[8]:
```
Index(['Serial No.', 'GRE Score', 'TOEFL Score', 'University Rating', 'SOP',
       'LOR ', 'CGPA', 'Research', 'Chance of Admit '],
      dtype='object')
```

In [9]:
```
# checking the dtypes of the features
df.dtypes
```

Out[9]:
```
Serial No.             int64
GRE Score              int64
TOEFL Score            int64
University Rating      int64
SOP                  float64
LOR                  float64
CGPA                 float64
Research               int64
Chance of Admit      float64
dtype: object
```

In [10]:
```
# check the null values
#Observation 4 : There are no null values in the dataset
df.isnull().sum()
```

Out[10]:
```
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 [11]:
```
# check for the duplicated values
# Observation 5 : There are no duplicate values in the dataset
df.duplicated().sum()
```

Out[11]:
```
0
```

In [12]:
```
# Finding the numerical features
numerical_features = [features for features in df.columns if df[features].dtypes != "O
```

In [13]:
```
# Finding the categorical features in the datset
categorical_features = [features for features in df.columns if df[features].dtypes =='
```

In [14]:
```
# Finding discreet features in the datset
discreet_features = [features for features in numerical_features if len(df[features].u
```

In [15]:
```
# Finding the continous features in the dataset
continous_features = [features for features in numerical_features if features not in d
```

In [16]:
```
# Convert Float features to integer types
df['SOP'] = df['SOP'].astype(int)
```

In [17]:
```
df['SOP'].dtypes
```

Out[17]:
```
dtype('int32')
```

In [18]:
```
# Convert rest of the float features into integer types in one shot
# Observation 6 : Here, we observed that the column names has space in it hence it nee
#df = df.astype({'LOR': int, 'CGPA': int, 'Chance of Admit': int })
```

In [19]:
```
# Removing space in the column names
df.columns
```

Out[19]:
```
Index(['Serial No.', 'GRE Score', 'TOEFL Score', 'University Rating', 'SOP',
       'LOR ', 'CGPA', 'Research', 'Chance of Admit '],
      dtype='object')
```

In [20]:
```
df = df.rename(columns = {'LOR ': 'LOR', 'Chance of Admit ': 'Chance of Admit'})
```

In [21]:
```
df['LOR']
```

Out[21]:
```
0      4.5
1      4.5
2      3.5
3      2.5
4      3.0
      ...
495    4.0
496    5.0
497    5.0
498    5.0
499    4.5
Name: LOR, Length: 500, dtype: float64
```

In [22]:
```
df.columns
```

Out[22]:
```
Index(['Serial No.', 'GRE Score', 'TOEFL Score', 'University Rating', 'SOP',
       'LOR', 'CGPA', 'Research', 'Chance of Admit'],
      dtype='object')
```

In [23]:
```
#Let us now convert the dtypes
df = df.astype({'LOR': int, 'CGPA': int})
```

In [24]:
```
df.dtypes
```

Out[24]:
```
Serial No.             int64
GRE Score              int64
TOEFL Score            int64
University Rating      int64
SOP                    int32
LOR                    int32
CGPA                   int32
Research               int64
Chance of Admit      float64
dtype: object
```

Univariate Analysis

In [25]:
```
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
```

In [32]:
```
df
```

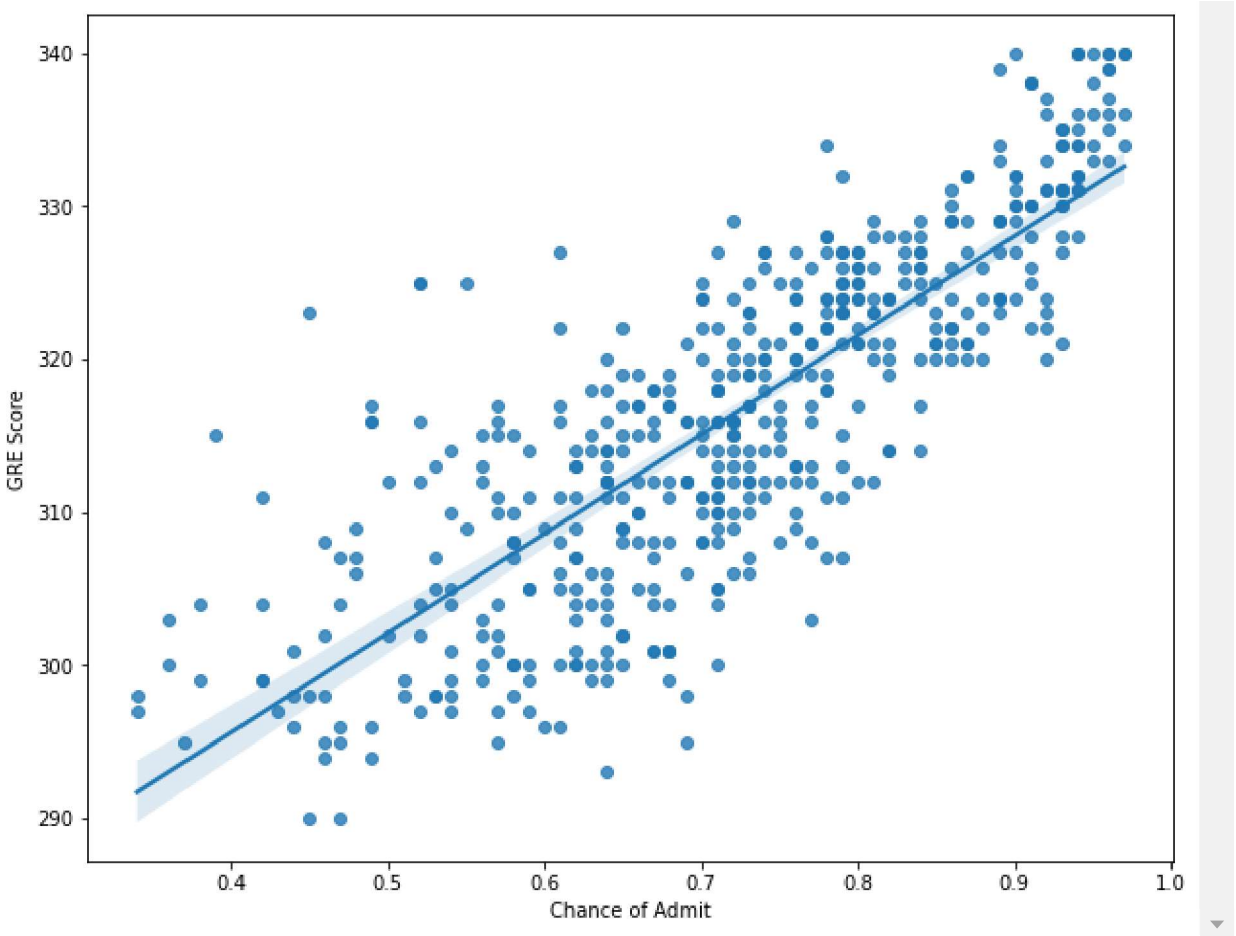
Out[32]:

	Serial No.	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit
0	1	337	118	4	4	4	9	1	0.92
1	2	324	107	4	4	4	8	1	0.76
2	3	316	104	3	3	3	8	1	0.72
3	4	322	110	3	3	2	8	1	0.80
4	5	314	103	2	2	3	8	0	0.65
...	...	...	...	...	...	...	...	...	...
495	496	332	108	5	4	4	9	1	0.87
496	497	337	117	5	5	5	9	1	0.96
497	498	330	120	5	4	5	9	1	0.93
498	499	312	103	4	4	5	8	0	0.73
499	500	327	113	4	4	4	9	0	0.84

500 rows × 9 columns

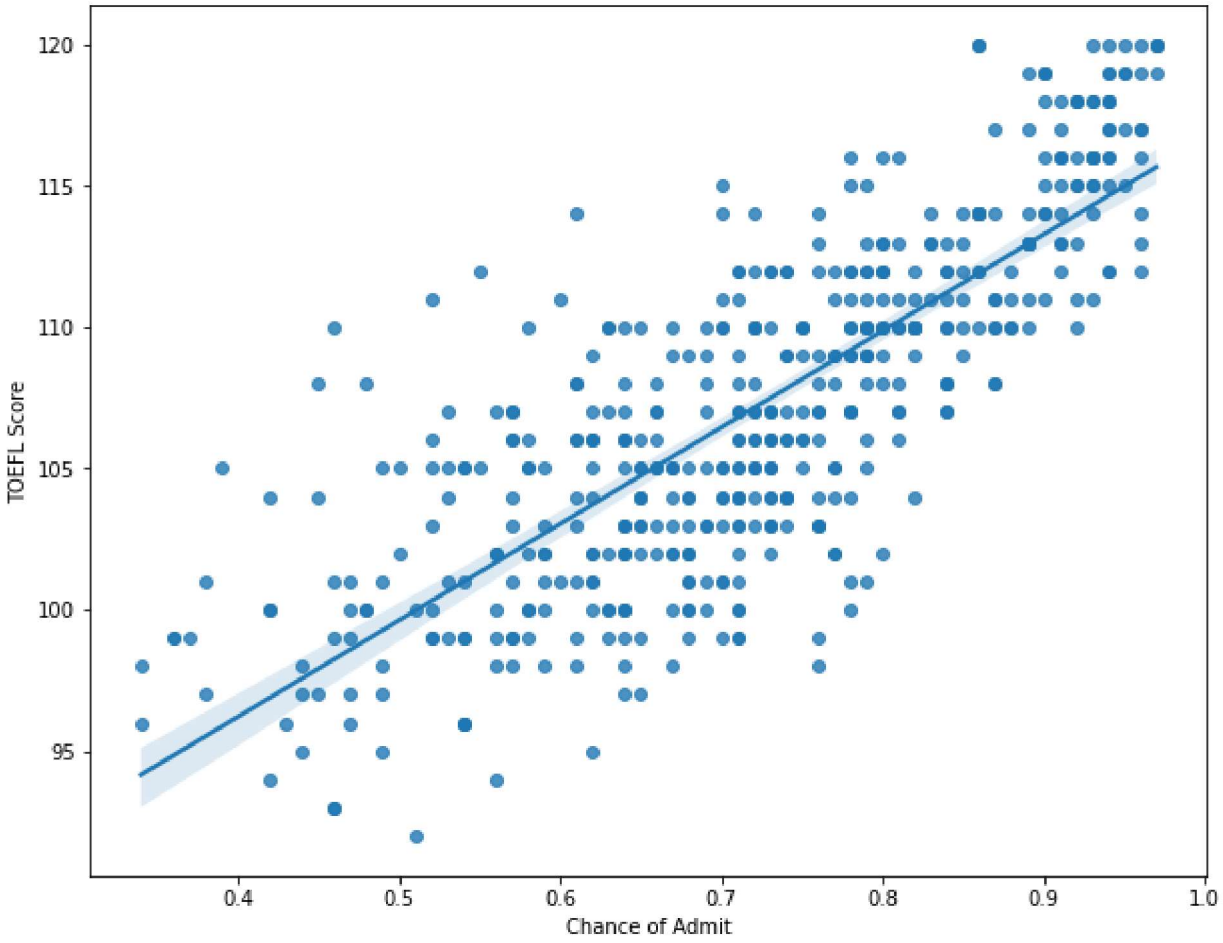
In [43]:
```
# Impact GREscore has on Target feature
plt.figure(figsize = (10,8))
sns.regplot(data = df, y = df['GRE Score'], x = 'Chance of Admit')
```

Out[43]:
```
<AxesSubplot:xlabel='Chance of Admit', ylabel='GRE Score'>
```



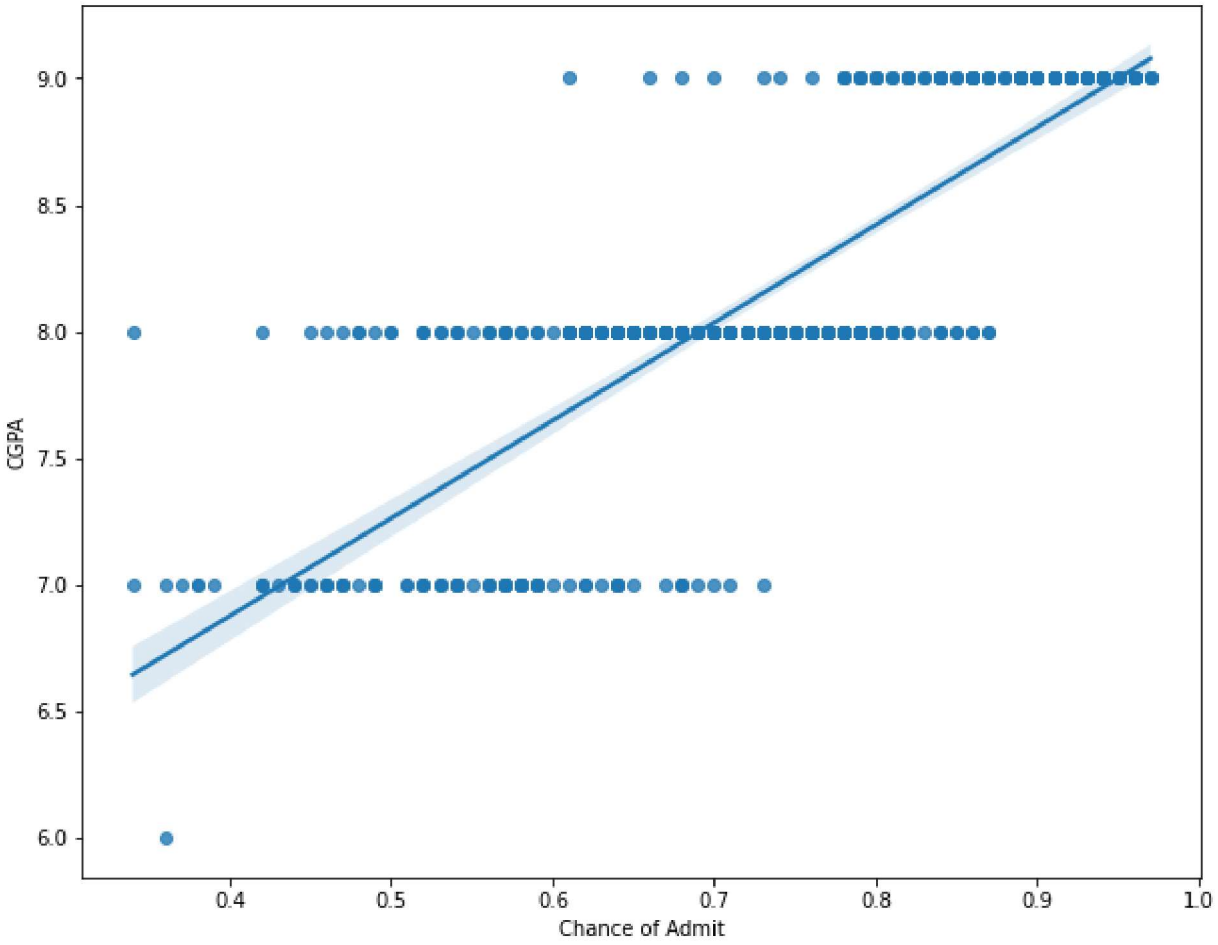
In [42]:
```
# Impact TOEFL Score has on Target feature
plt.figure(figsize = (10,8))
sns.regplot(data = df, y = df['TOEFL Score'], x = 'Chance of Admit')
```

Out[42]:
```
<AxesSubplot:xlabel='Chance of Admit', ylabel='TOEFL Score'>
```



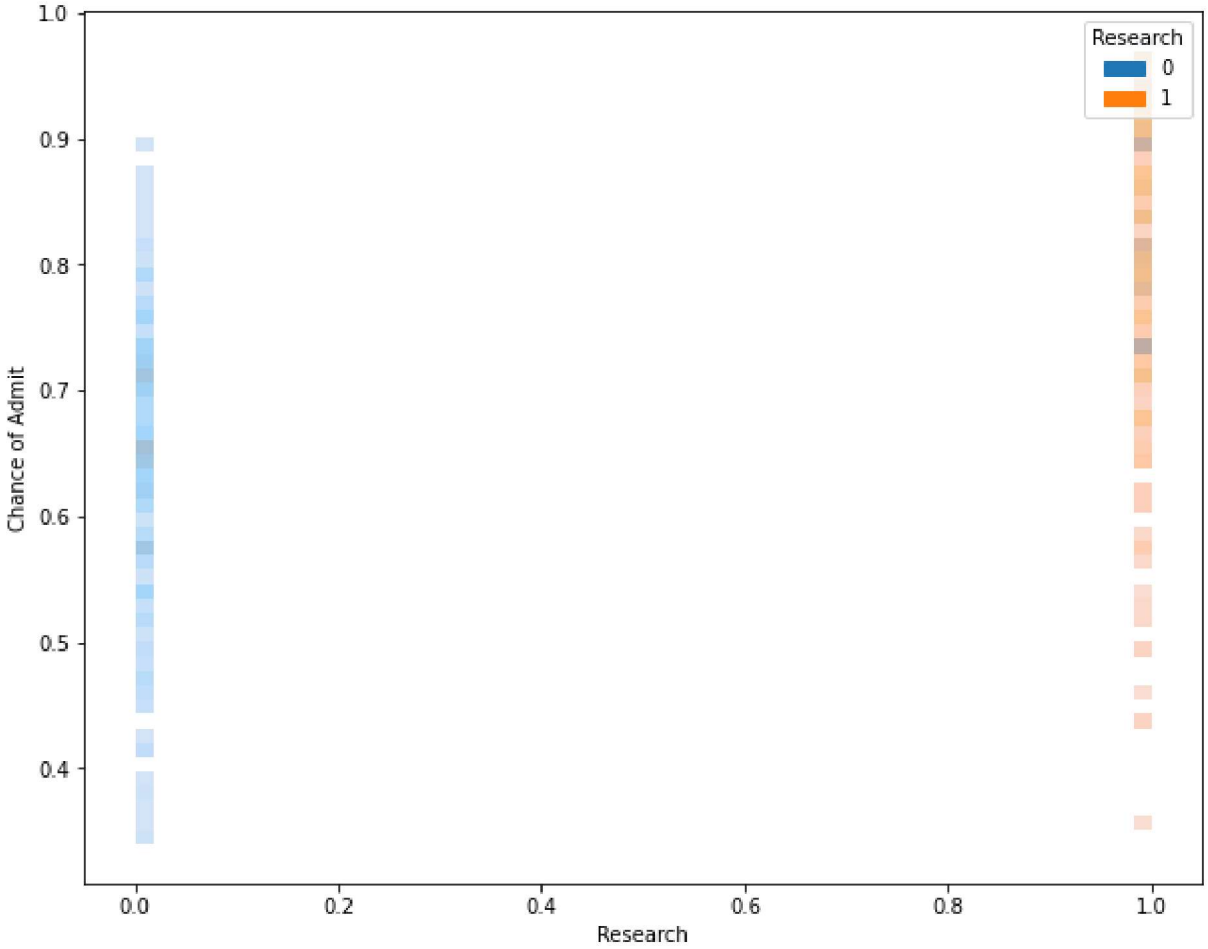
In [44]:
```
# Impact CGPA Score has on Target feature
plt.figure(figsize = (10,8))
sns.regplot(data = df, y = df['CGPA'], x = 'Chance of Admit')
```

Out[44]:
```
<AxesSubplot:xlabel='Chance of Admit', ylabel='CGPA'>
```



In [76]:
```
# Impact Research has on Target feature
plt.figure(figsize = (10,8))
sns.histplot(x = df['Research'],y = 'Chance of Admit', hue = 'Research',data = df, alp
```

Out[76]:
```
<AxesSubplot:xlabel='Research', ylabel='Chance of Admit'>
```



Bivariate Analysis

In [60]:
```
df
```

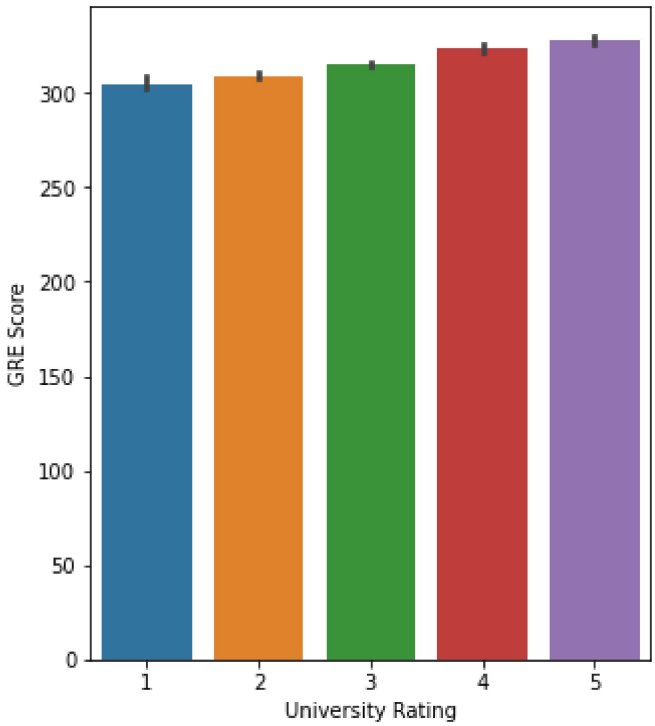
Out[60]:

	Serial No.	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit
0	1	337	118	4	4	4	9	1	0.92
1	2	324	107	4	4	4	8	1	0.76
2	3	316	104	3	3	3	8	1	0.72
3	4	322	110	3	3	2	8	1	0.80
4	5	314	103	2	2	3	8	0	0.65
...	...	...	...	...	...	...	...	...	...
495	496	332	108	5	4	4	9	1	0.87
496	497	337	117	5	5	5	9	1	0.96
497	498	330	120	5	4	5	9	1	0.93
498	499	312	103	4	4	5	8	0	0.73
499	500	327	113	4	4	4	9	0	0.84

500 rows × 9 columns

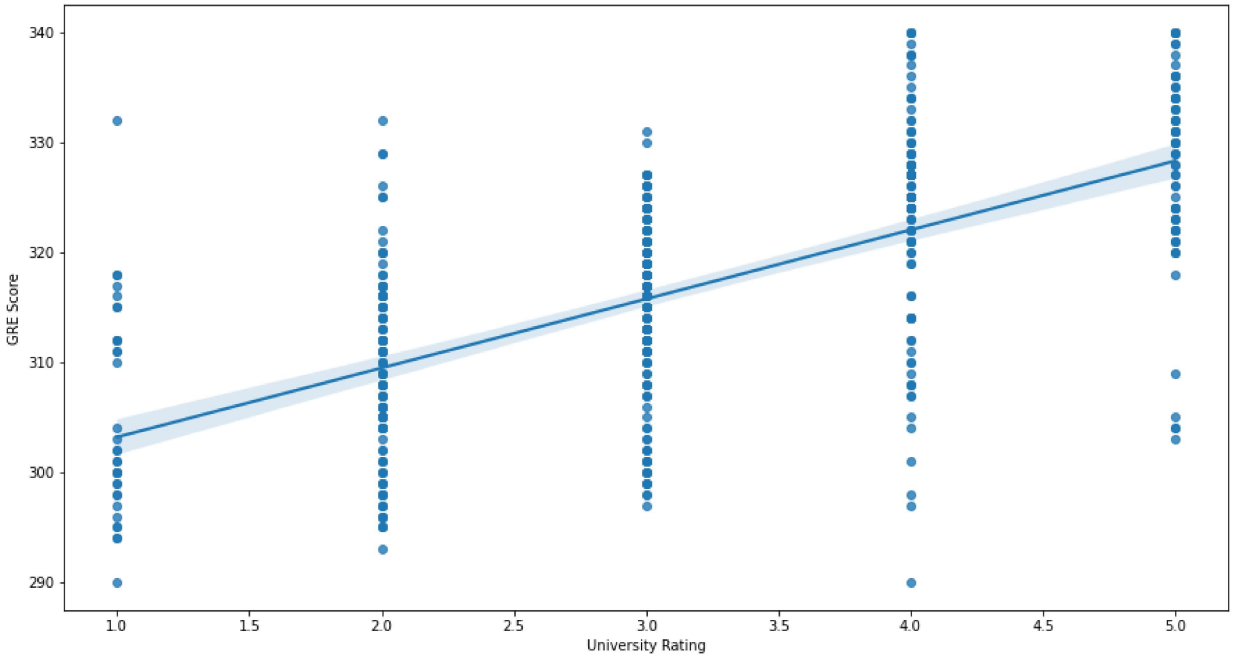
In [87]:
```
plt.figure(figsize = (5,6))
sns.barplot(y = df['GRE Score'], x = 'University Rating',data = df)
```

Out[87]:
```
<AxesSubplot:xlabel='University Rating', ylabel='GRE Score'>
```



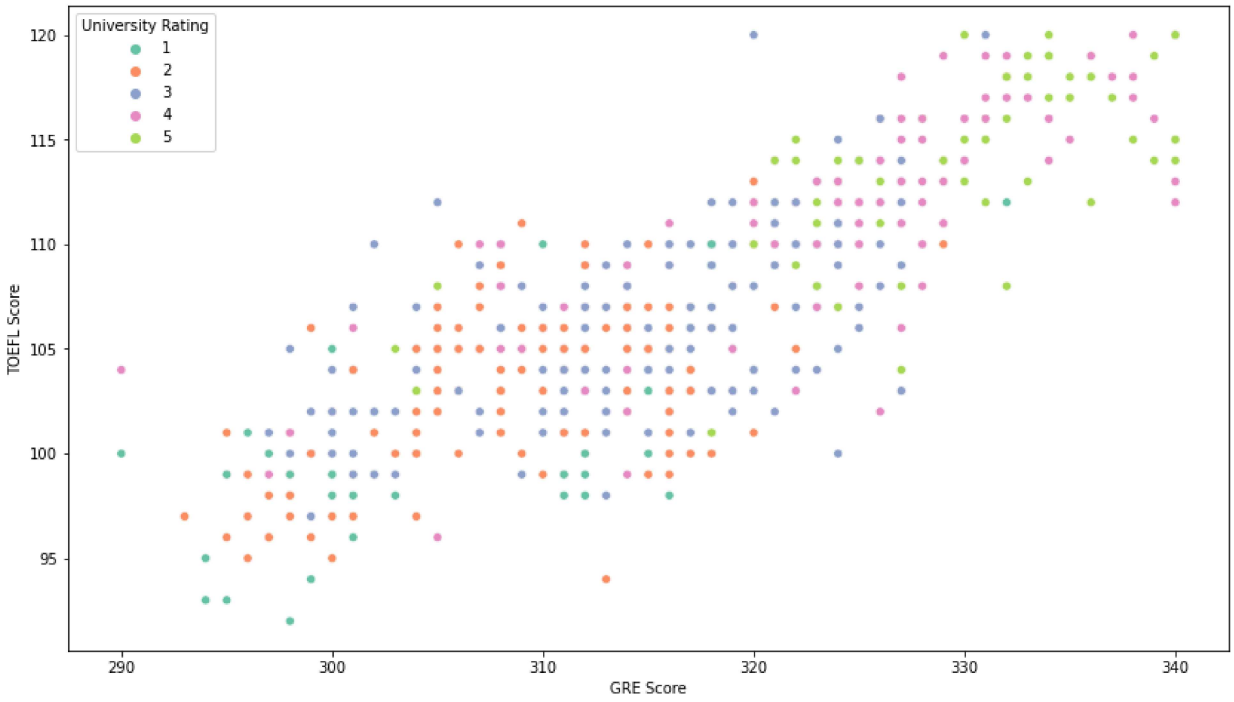
In [91]:
```
plt.figure(figsize = (15,8))
sns.regplot(y = df['GRE Score'], x = 'University Rating',data = df)
```

Out[91]:
```
<AxesSubplot:xlabel='University Rating', ylabel='GRE Score'>
```



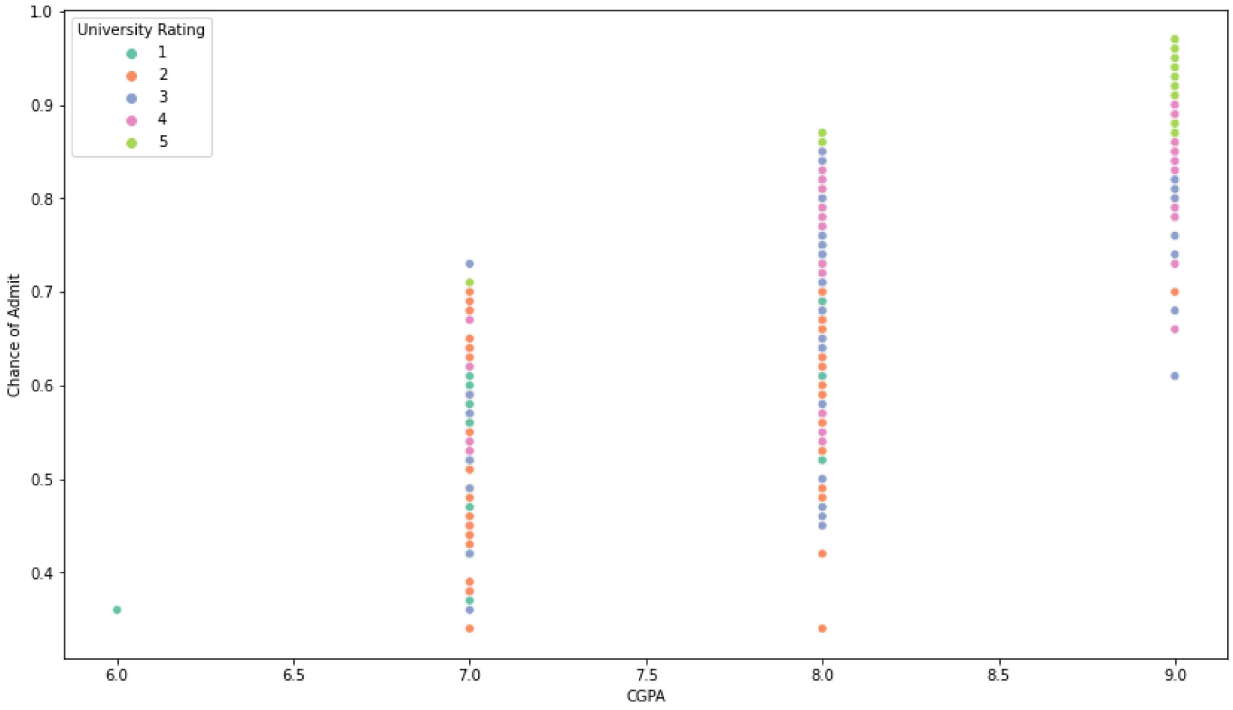
In [102…
```
plt.figure(figsize = (14, 8))
sns.scatterplot(x = df['GRE Score'], y = 'TOEFL Score', hue = 'University Rating', dat
```

Out[102]:
```
<AxesSubplot:xlabel='GRE Score', ylabel='TOEFL Score'>
```



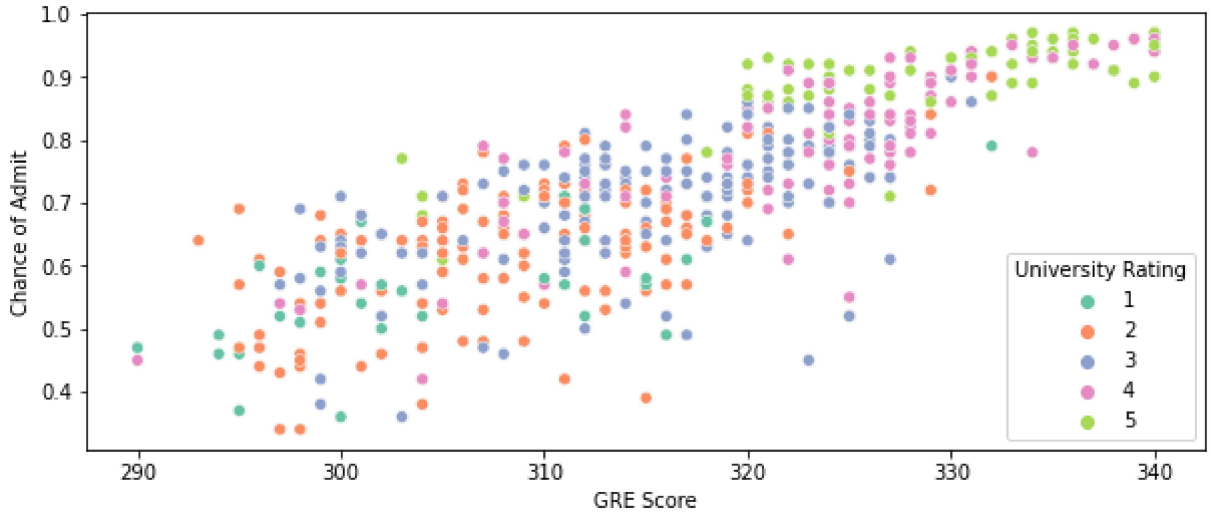
In [104…
```
plt.figure(figsize = (14, 8))
sns.scatterplot(x = df['CGPA'], y = 'Chance of Admit', hue = 'University Rating', data
```

Out[104]:
```
<AxesSubplot:xlabel='CGPA', ylabel='Chance of Admit'>
```



In [109…
```
plt.figure(figsize = (10, 4))
sns.scatterplot(x = df['GRE Score'], y = 'Chance of Admit', hue = 'University Rating',
```

Out[109]:
```
<AxesSubplot:xlabel='GRE Score', ylabel='Chance of Admit'>
```



Mutlivariate Analysis

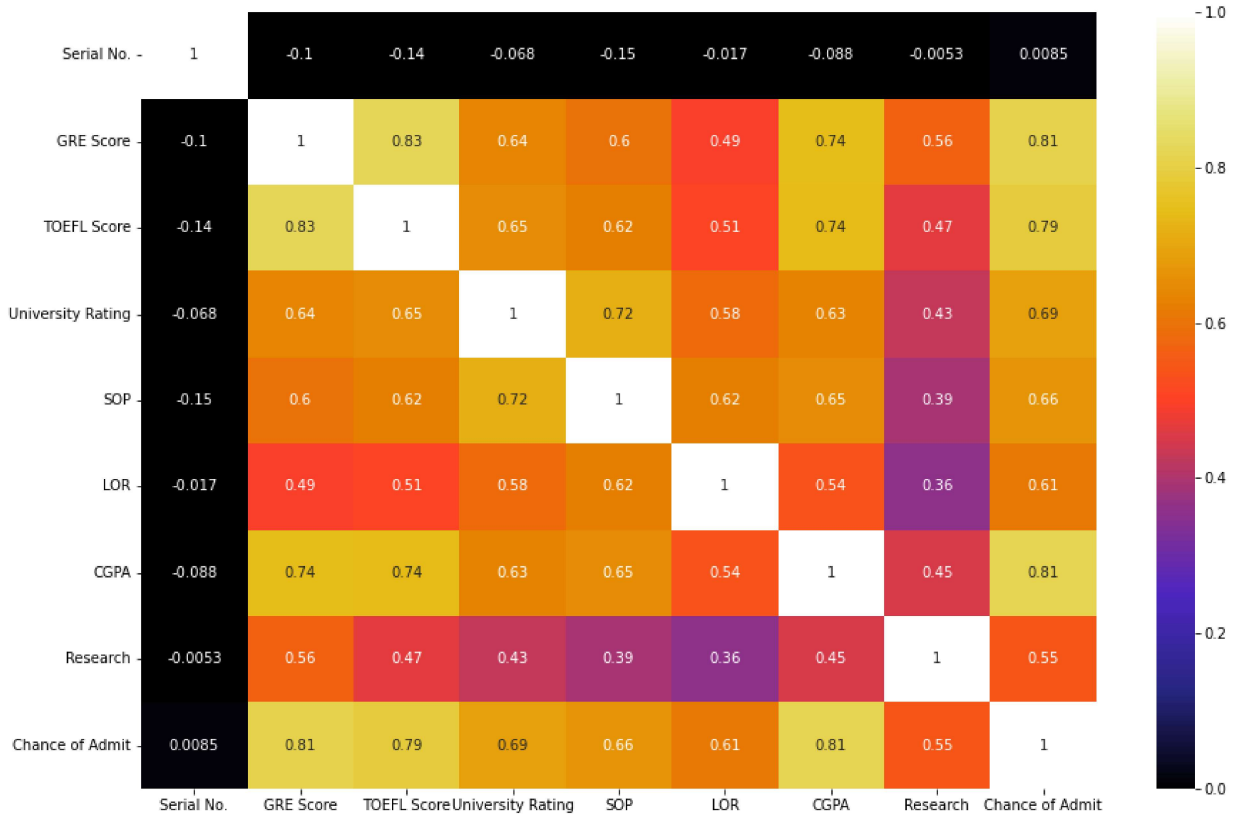
In [110…
```
df.corr()
```

Out[110]:

	Serial No.	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Cl A
Serial No.	1.000000	-0.103839	-0.141696	-0.067641	-0.152894	-0.017465	-0.087635	-0.005332	0.0(
GRE Score	-0.103839	1.000000	0.827200	0.635376	0.597419	0.493999	0.743827	0.563398	0.8
TOEFL Score	-0.141696	0.827200	1.000000	0.649799	0.623507	0.510702	0.740427	0.467012	0.7
University Rating	-0.067641	0.635376	0.649799	1.000000	0.715252	0.582600	0.631043	0.427047	0.6
SOP	-0.152894	0.597419	0.623507	0.715252	1.000000	0.621785	0.648813	0.392705	0.6(
LOR	-0.017465	0.493999	0.510702	0.582600	0.621785	1.000000	0.536366	0.356890	0.6
CGPA	-0.087635	0.743827	0.740427	0.631043	0.648813	0.536366	1.000000	0.451194	0.8
Research	-0.005332	0.563398	0.467012	0.427047	0.392705	0.356890	0.451194	1.000000	0.5
Chance of Admit	0.008505	0.810351	0.792228	0.690132	0.664467	0.611591	0.813652	0.545871	1.0(

In [124…
```
plt.figure(figsize = (15, 10))
sns.heatmap(df.corr(), data = df, annot = True, cmap="CMRmap",vmin=0, vmax=1,)
```

Out[124]: <AxesSubplot:>



In [133…
```
df.drop('Serial No.', axis = 1, inplace = True)
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
Input In [133], in <cell line: 1>()
----> 1 df.drop('Serial No.', axis = 1, inplace = True)

File ~\Anaconda3\lib\site-packages\pandas\util\_decorators.py:311, in deprecate_nonke
yword_arguments.<locals>.decorate.<locals>.wrapper(*args, **kwargs)
    305 if len(args) > num_allow_args:
    306     warnings.warn(
    307         msg.format(arguments=arguments),
    308         FutureWarning,
    309         stacklevel=stacklevel,
    310     )
--> 311 return func(*args, **kwargs)

File ~\Anaconda3\lib\site-packages\pandas\core\frame.py:4954, in DataFrame.drop(self,
labels, axis, index, columns, level, inplace, errors)
   4806 @deprecate_nonkeyword_arguments(version=None, allowed_args=["self", "labels"
])
   4807 def drop(
   4808     self,
   (...)
   4815     errors: str = "raise",
   4816 ):
   4817     """
   4818     Drop specified labels from rows or columns.
   4819
   (...)
   4952             weight  1.0     0.8
   4953     """
-> 4954     return super().drop(
   4955         labels=labels,
   4956         axis=axis,
   4957         index=index,
   4958         columns=columns,
   4959         level=level,
   4960         inplace=inplace,
   4961         errors=errors,
   4962     )

File ~\Anaconda3\lib\site-packages\pandas\core\generic.py:4267, in NDFrame.drop(self,
labels, axis, index, columns, level, inplace, errors)
   4265 for axis, labels in axes.items():
   4266     if labels is not None:
-> 4267         obj = obj._drop_axis(labels, axis, level=level, errors=errors)
   4269 if inplace:
   4270     self._update_inplace(obj)

File ~\Anaconda3\lib\site-packages\pandas\core\generic.py:4311, in NDFrame._drop_axis
(self, labels, axis, level, errors, consolidate, only_slice)
   4309         new_axis = axis.drop(labels, level=level, errors=errors)
   4310     else:
-> 4311         new_axis = axis.drop(labels, errors=errors)
   4312     indexer = axis.get_indexer(new_axis)
   4314 # Case for non-unique axis
   4315 else:

File ~\Anaconda3\lib\site-packages\pandas\core\indexes\base.py:6644, in Index.drop(se
lf, labels, errors)
   6642 if mask.any():
```

```
   6643     if errors != "ignore":
-> 6644         raise KeyError(f"{list(labels[mask])} not found in axis")
   6645     indexer = indexer[~mask]
   6646 return self.delete(indexer)

KeyError: "['Serial No.'] not found in axis"
```

In [135…
```
df
```

Out[135]:

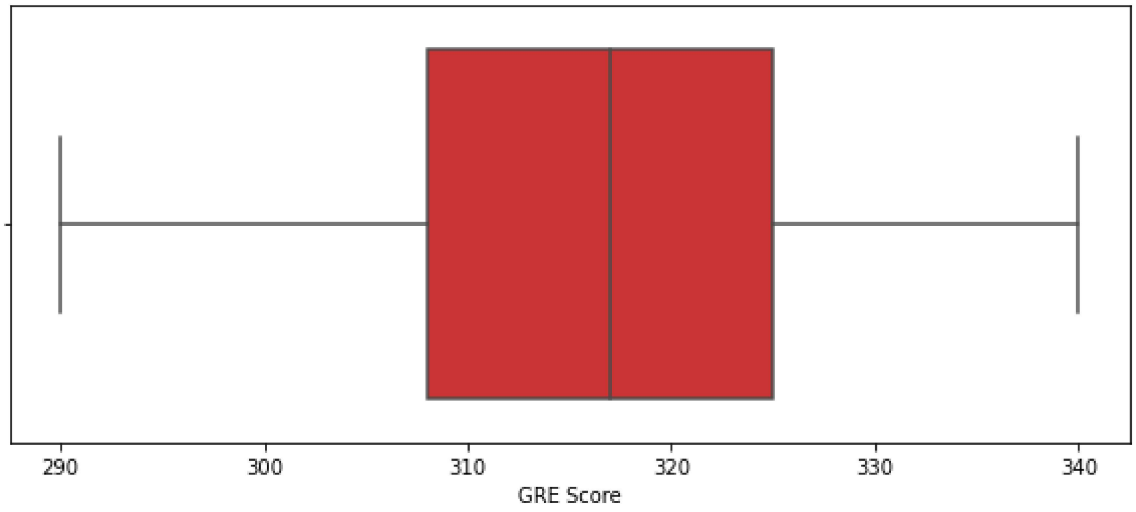
	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit
0	337	118	4	4	4	9	1	0.92
1	324	107	4	4	4	8	1	0.76
2	316	104	3	3	3	8	1	0.72
3	322	110	3	3	2	8	1	0.80
4	314	103	2	2	3	8	0	0.65
…	…	…	…	…	…	…	…	…
495	332	108	5	4	4	9	1	0.87
496	337	117	5	5	5	9	1	0.96
497	330	120	5	4	5	9	1	0.93
498	312	103	4	4	5	8	0	0.73
499	327	113	4	4	4	9	0	0.84

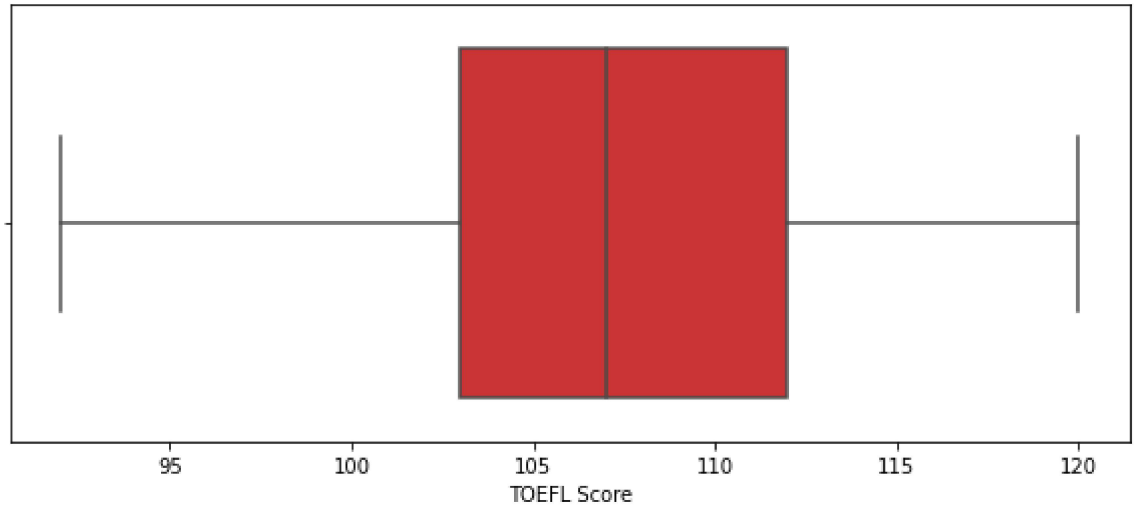
500 rows × 8 columns

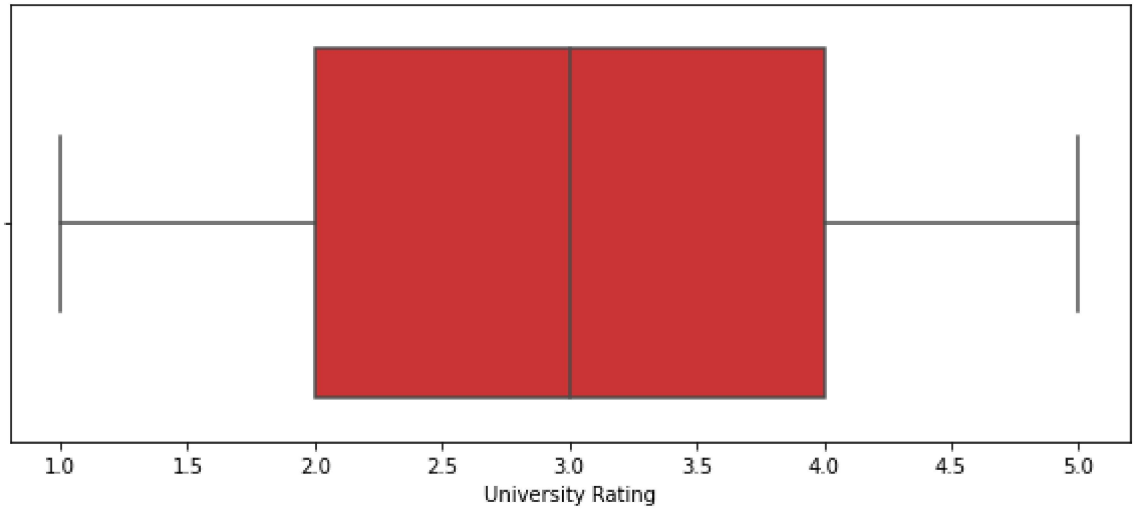
In [143…
```
features_without_target = [features for features in df.columns if features not in 'Cha
features_without_target
```

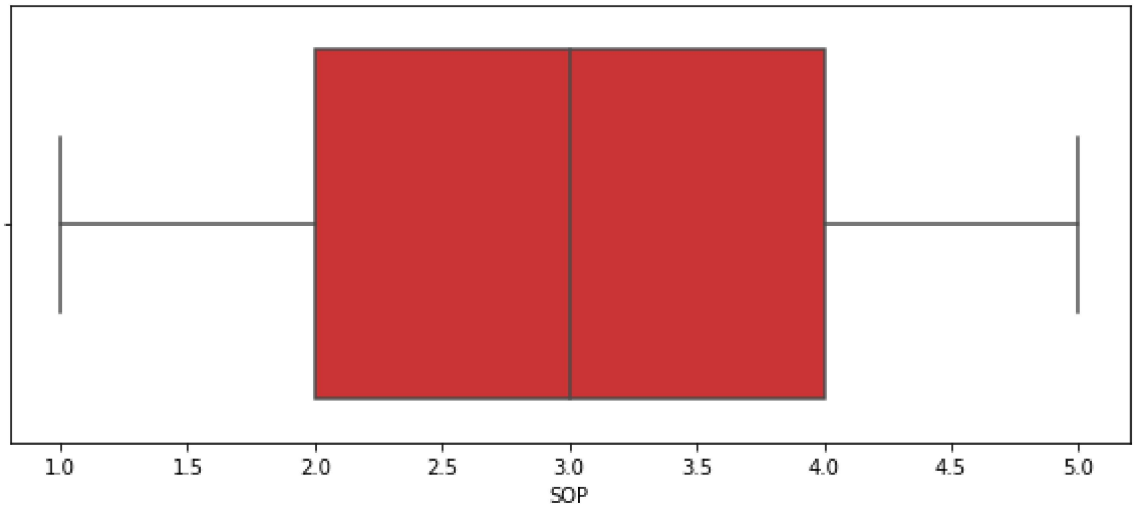
Out[143]:
```
['GRE Score',
 'TOEFL Score',
 'University Rating',
 'SOP',
 'LOR',
 'CGPA',
 'Research']
```

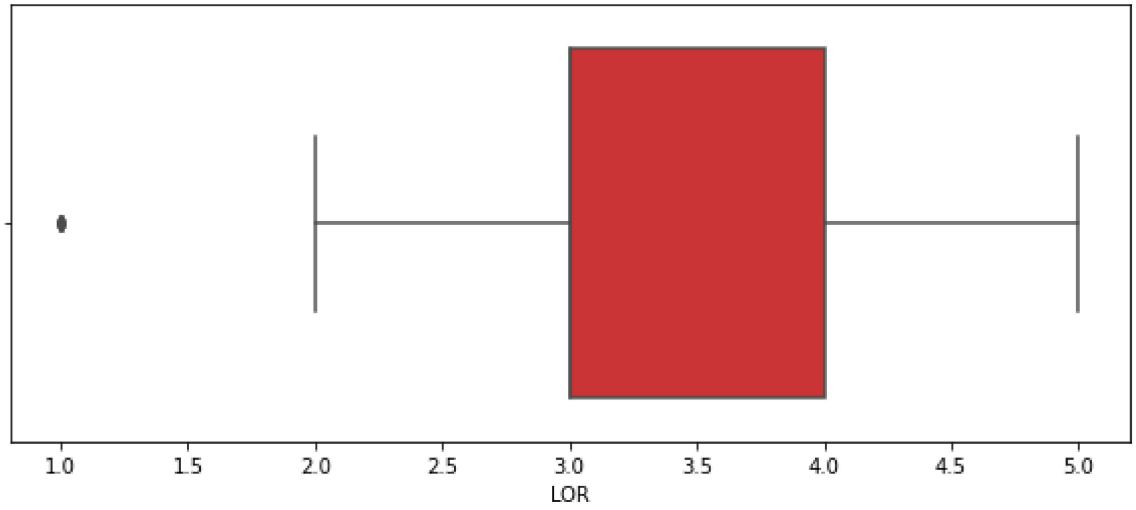
In [156…
```
for features in features_without_target:
    df1 = df.copy()
    plt.figure(figsize = (10,4))
    sns.boxplot(x = df1[features], palette = 'Set1')

#     for features in df.columns:
#     df1 = df.copy()
#     plt.figure(figsize = (8,6))
#     sns.boxplot(x = df1[features],color = 'b' )
```

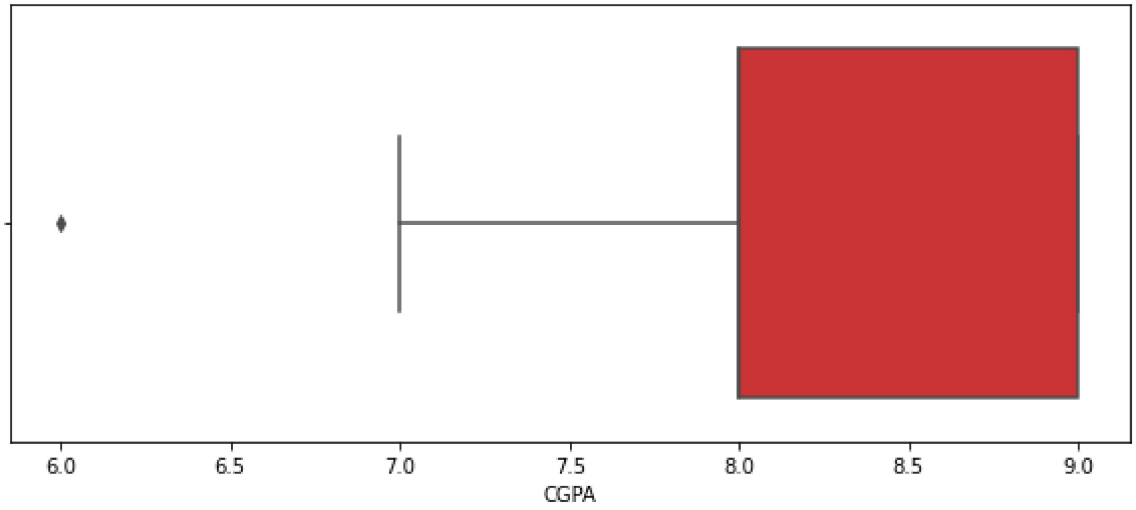

290
300
310
320
330
340
GRE Score

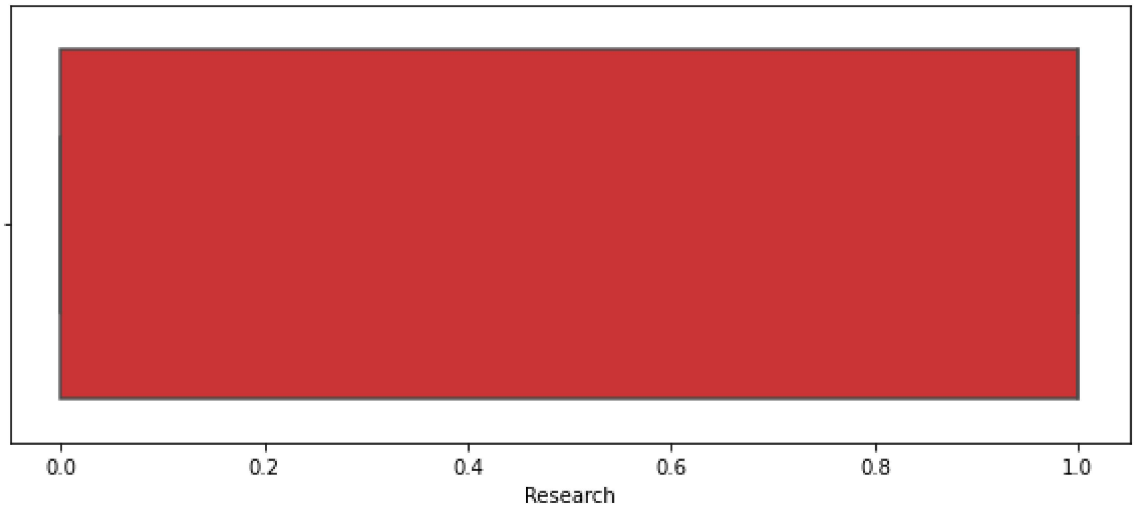

95
100
105
110
115
120
TOEFL Score


1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
University Rating


1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
SOP


1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
LOR


6.0
6.5
7.0
7.5
8.0
8.5
9.0
CGPA



Model Building

In [164…
```
X = df.iloc[:,:-1]
```

In [165…
```
y = df['Chance of Admit']
```

In [166…
```
X
```

Out[166]:

	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research
0	337	118	4	4	4	9	1
1	324	107	4	4	4	8	1
2	316	104	3	3	3	8	1
3	322	110	3	3	2	8	1
4	314	103	2	2	3	8	0
...	...	...	...	...	...	...	...
495	332	108	5	4	4	9	1
496	337	117	5	5	5	9	1
497	330	120	5	4	5	9	1
498	312	103	4	4	5	8	0
499	327	113	4	4	4	9	0

500 rows × 7 columns

In [167…
```
y
```

Out[167]:
```
0      0.92
1      0.76
2      0.72
3      0.80
4      0.65
       ...
495    0.87
496    0.96
497    0.93
498    0.73
499    0.84
Name: Chance of Admit, Length: 500, dtype: float64
```

Split the dataset into train, test, split

In [168…
```
from sklearn.model_selection import train_test_split
```

In [169…
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state
```

Standard Scaling

In [170…
```
from sklearn import preprocessing
```

Instantiate Standard Scaling

In [171…
```
scaler = preprocessing.StandardScaler()
```

Fit and Transform the X_train dataset to calcualte mean and Standard deviation

In [172…
```
X_train_transform = scaler.fit_transform(X_train)
```

In [173…
```
print(scaler.mean_)
```

```
[315.72238806 106.89850746   3.07761194   3.10149254   3.21791045
   8.09850746   0.54925373]
```

Transform and preserve the X_test dataset with the same pattern from the X_train features to calculate the mean and standard deviation

In [306…
```
X_test_transform = scaler.transform(X_test)
X_test_transform
```

Out[306]:
```
array([[ 1.41759906,  1.49957357,  1.68511343, ...,  1.85527233,
         1.31371417,  0.90589855],
       [ 0.80797687,  1.170051  ,  1.68511343, ...,  1.85527233,
         1.31371417,  0.90589855],
       [ 0.89506575,  0.18148332, -0.06803253, ..., -0.2268591 ,
        -0.14355155, -1.10387637],
       ...,
       [ 2.11431013,  1.00528972,  0.80854045, ...,  1.85527233,
         1.31371417,  0.90589855],
       [ 0.72088799, -1.13660692, -0.06803253, ...,  1.85527233,
        -0.14355155,  0.90589855],
       [ 0.37253245,  0.51100588, -0.94460551, ..., -0.2268591 ,
        -0.14355155, -1.10387637]])
```

In [175…
```
print(scaler.mean_)
```

```
[315.72238806 106.89850746   3.07761194   3.10149254   3.21791045
   8.09850746   0.54925373]
```

Linear Regression Model

In [176…
```
from sklearn.linear_model import LinearRegression
```

In [177…
```
regression=LinearRegression()
```

In [178…
```
regression
```

Out[178]:
```
LinearRegression()
```

In [179…
```
regression.fit(X_train,y_train)
```

Out[179]:
```
LinearRegression()
```

In [180…
```
print(regression.coef_)
```

```
[0.00343479 0.00316098 0.00885074 0.00246264 0.02388638 0.0735812
 0.02100568]
```

In [183…
```
print(regression.intercept_)
```

```
-1.4272523895959213
```

In [184…
```
## PRediction for the test data
reg_pred= regression.predict(X_test)
```

In [185…
```
reg_pred
```

Out[185]:
```
array([0.93900717, 0.90617904, 0.72812421, 0.86937494, 0.6861542 ,
       0.71587723, 0.63535443, 0.88737035, 0.62188908, 0.69718349,
       0.86864451, 0.88858055, 0.87769815, 0.66739321, 0.78475417,
       0.76234535, 0.88596271, 0.92517895, 0.76499896, 0.64082088,
       0.68811861, 0.61568685, 0.58920388, 0.86711996, 0.4637427 ,
       0.94876392, 0.69717702, 0.54207218, 0.68772815, 0.68651624,
       0.95660566, 0.75923137, 0.57523662, 0.66416779, 0.73222975,
       0.90298597, 0.75743301, 0.6188816 , 0.64164232, 0.70499116,
       0.91575298, 0.63274109, 0.68812508, 0.89533349, 0.91199388,
       0.53195815, 0.61842022, 0.74618407, 0.75762227, 0.72331837,
       0.7806872 , 0.69276728, 0.5801321 , 0.62821103, 0.71705705,
       0.77031667, 0.88132051, 0.58562555, 0.75968656, 0.93557238,
       0.67419664, 0.88541957, 0.77101501, 0.49073786, 0.77663862,
       0.50861016, 0.72062891, 0.72538408, 0.50920828, 0.77225927,
       0.90037262, 0.5192252 , 0.63559418, 0.71957767, 0.75900328,
       0.65806099, 0.6419502 , 0.84537739, 0.84389786, 0.75240752,
       0.65677817, 0.69127959, 0.68936934, 0.76568223, 0.77722031,
       0.77429909, 0.79840709, 0.64319896, 0.76325952, 0.97027867,
       0.87438001, 0.66204341, 0.50729328, 0.77192101, 0.74663319,
       0.69429157, 0.71927994, 0.92046584, 0.49867511, 0.68449456,
       0.73511691, 0.92511451, 0.71849058, 0.73308158, 0.88217602,
       0.45791619, 0.76532524, 0.45908954, 0.75545185, 0.82228011,
       0.5122847 , 0.72508635, 0.68078595, 0.68222118, 0.86000416,
       0.65709531, 0.73771014, 0.4985796 , 0.76438715, 0.73566454,
       0.63918615, 0.92741997, 0.78097941, 0.82332964, 0.8042336 ,
       0.65381121, 0.68505036, 0.75975579, 0.71981913, 0.94532912,
       0.88217602, 0.71560342, 0.82446893, 0.74931745, 0.68909724,
       0.59924161, 0.70780938, 0.65450955, 0.86711996, 0.83067424,
       0.52489822, 0.59876659, 0.50924235, 0.82410606, 0.6580535 ,
       0.87742434, 0.83058459, 0.93037975, 0.88663345, 0.64234066,
       0.93697552, 0.47906611, 0.68243531, 0.65924081, 0.86684615,
       0.69487802, 0.91930442, 0.61763284, 0.76520859, 0.86341135,
       0.84167695, 0.58374799, 0.94815183, 0.76720788, 0.7074493 ])
```

In [186…
```
# variance score: 1 means perfect prediction
print('Variance score: {}'.format(regression.score(X_test, y_test)))
```

```
Variance score: 0.7799364853355912
```

In [187…
```
from sklearn.metrics import r2_score
score=r2_score(y_test,reg_pred)
print(score)
```

```
0.7799364853355912
```

In [188…
```
from sklearn import datasets, linear_model, metrics
mae = metrics.mean_absolute_error(y_test, reg_pred)
mse = metrics.mean_squared_error(y_test, reg_pred)
rmse = np.sqrt(mse) # or mse**(0.5)
r2 = metrics.r2_score(y_test, reg_pred)
AR2 = 1 - (1-score)*(len(y_test)-1)/(len(y_test)-X_test.shape[1]-1)

print("Results of sklearn.metrics:")
print("MAE:",mae)
print("MSE:", mse)
print("RMSE:", rmse)
print("R-Squared:", r2)
print("Adjusted R2:", AR2)
```

```
Results of sklearn.metrics:
MAE: 0.04324901286761981
MSE: 0.0037868578102297668
RMSE: 0.06153745046904175
R-Squared: 0.7799364853355912
Adjusted R2: 0.7701247362741208
```

SVR Model

In [192…
```
from sklearn.svm import SVR
```

In [236…
```
SVR_model = SVR(kernel="rbf", C=100, gamma=0.1, epsilon=0.1)
```

In [237…
```
SVR_model.fit(X_train_transform,y_train)
```

Out[237]:
```
SVR(C=100, gamma=0.1)
```

In [238…
```
SVR_model.score(X_train_transform,y_train)
```

Out[238]:
```
0.7880256729363578
```

In [301…
```
SVR_model_predict = SVR_model.predict(X_test_transform)
SVR_model_predict
```

Out[301]:
```
array([0.89970366, 0.87749825, 0.47498261, 0.89964699, 0.76130666,
       0.75121705, 0.59550308, 0.89997643, 0.59739356, 0.67628267,
       0.8624619 , 0.859698  , 0.89863678, 0.68480852, 0.87982167,
       0.82099682, 0.79995371, 0.77161453, 0.75934591, 0.63269344,
       0.75533093, 0.5098556 , 0.50784825, 0.88128363, 0.48546556,
       0.89320417, 0.73312772, 0.55227078, 0.64804897, 0.60969407,
       0.88977246, 0.73603428, 0.54504218, 0.66578544, 0.74831613,
       0.84971375, 0.80118047, 0.59926096, 0.6308987 , 0.68506474,
       0.87426779, 0.65422364, 0.70900839, 0.92111545, 0.9098089 ,
       0.67045151, 0.54543461, 0.55892993, 0.81319953, 0.80166502,
       0.89397336, 0.61335805, 0.97600219, 0.57464118, 0.57859722,
       0.81618082, 0.8806735 , 0.74280641, 0.8387207 , 0.8985166 ,
       0.6624973 , 0.8612041 , 0.83207822, 0.43507223, 0.7739341 ,
       0.46946151, 0.7104138 , 0.78342426, 0.553765  , 0.72532517,
       0.89305656, 0.54074141, 0.62715687, 0.75531448, 0.80272922,
       0.65578221, 0.58579696, 0.93471521, 0.87007759, 0.80454126,
       0.57817168, 0.76184993, 0.68600867, 0.70015309, 0.80134157,
       0.86095585, 0.71705963, 0.66171092, 0.80227924, 0.84695058,
       0.84343598, 0.68271426, 0.53841044, 0.7797425 , 0.80455675,
       0.76905791, 0.63849127, 0.90200284, 0.46854354, 0.74884747,
       0.74609389, 0.89713453, 0.74458578, 0.71486939, 0.88853902,
       0.42157443, 0.78325497, 0.44961552, 0.7520029 , 0.80918766,
       0.49075211, 0.74280774, 0.73866647, 0.72775909, 0.78414092,
       0.61301637, 0.6283413 , 0.45098091, 0.75217669, 0.74519547,
       0.68936028, 0.85575469, 0.73195518, 0.82029706, 0.82248759,
       0.63457287, 0.65311322, 0.48108197, 0.6473792 , 0.89167923,
       0.88853902, 0.75341359, 0.79527554, 0.6939899 , 0.7291101 ,
       0.72640841, 0.65088494, 0.60037303, 0.88128363, 0.84081932,
       0.55361018, 0.5641435 , 0.52864225, 0.7273436 , 0.53604912,
       0.89675719, 0.77805995, 0.90893472, 0.86084393, 0.60936815,
       0.90857268, 0.42501951, 0.67306351, 0.63537617, 0.86296525,
       0.67274709, 0.83116313, 0.54131147, 0.76814287, 0.8517566 ,
       0.84724377, 0.63762113, 0.82352421, 0.6685352 , 0.68609245])
```

In [303…
```
X_test_transform[0]
```

Out[303]:
```
array([1.41759906, 1.49957357, 1.68511343, 1.82495098, 1.85527233,
       1.31371417, 0.90589855])
```

In [305…
```
SVR_model.predict([[1.41759906, 1.49957357, 1.68511343, 1.82495098, 1.85527233,
       1.31371417, 0.90589855]])
```

Out[305]:
```
array([0.89970366])
```

GridsearchCV

In [265…
```
from sklearn.model_selection import GridSearchCV
```

In [292…
```
grid_param = {'kernel' : ['linear', 'poly', 'sigmoid'],
              'gamma': ['scale', 'auto'],
              'C': [1.0],
              'epsilon': [0.1],
              'verbose': [False],
              'max_iter': [-1]}
```

In [293…
```
grid_search = GridSearchCV(estimator= SVR_model,
                           param_grid = grid_param,cv = 6)
```

In [294…
```
grid_search = grid_search.fit(X_train_transform, y_train)
```

In [295…
```
grid_search
```

Out[295]:
```
GridSearchCV(cv=6, estimator=SVR(C=100, gamma=0.1),
             param_grid={'C': [1.0], 'epsilon': [0.1],
                         'gamma': ['scale', 'auto'],
                         'kernel': ['linear', 'poly', 'sigmoid'],
                         'max_iter': [-1], 'verbose': [False]})
```

In [296…
```
accuracy = grid_search.best_score_*100
accuracy
```

Out[296]:
```
75.6152870491771
```

Thank you for your Time

In [ ]:
```
```