In [ ]:

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from collections import Counter

import warnings
warnings.filterwarnings('ignore')

sns.set_style('darkgrid')
from matplotlib import pyplot
```

In [ ]:

```python
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

In [ ]:

```python
import pandas as pd
df = pd.read_csv('/content/drive/My Drive/ML_Dataset/Admission_Predict_Ver1.1.csv')
```

In [ ]:

```python
df.head(10).T
```

Out[ ]:

	0	1	2	3	4	5	6	7	8	9
Serial No.	1.00	2.00	3.00	4.00	5.00	6.00	7.00	8.00	9.0	10.00
GRE Score	337.00	324.00	316.00	322.00	314.00	330.00	321.00	308.00	302.0	323.00
TOEFL Score	118.00	107.00	104.00	110.00	103.00	115.00	109.00	101.00	102.0	108.00
University Rating	4.00	4.00	3.00	3.00	2.00	5.00	3.00	2.00	1.0	3.00
SOP	4.50	4.00	3.00	3.50	2.00	4.50	3.00	3.00	2.0	3.50
LOR	4.50	4.50	3.50	2.50	3.00	3.00	4.00	4.00	1.5	3.00
CGPA	9.65	8.87	8.00	8.67	8.21	9.34	8.20	7.90	8.0	8.60
Research	1.00	1.00	1.00	1.00	0.00	1.00	1.00	0.00	0.0	0.00
Chance of Admit	0.92	0.76	0.72	0.80	0.65	0.90	0.75	0.68	0.5	0.45

In [ ]:

```python
df=df.rename(columns = {'Chance of Admit ':'Chance of Admit'})
```

In [ ]:

```python
def detect_outliers(df,n,features):
    """
    Takes a dataframe df of features and returns a list of the indices
    corresponding to the observations containing more than n outliers according
    to the Tukey method.
    """
    outlier_indices = []

    # iterate over features(columns)
    for col in features:
        # 1st quartile (25%)
        Q1 = np.percentile(df[col], 25)
        # 3rd quartile (75%)
        Q3 = np.percentile(df[col],75)
        # Interquartile range (IQR)
        IQR = Q3 - Q1

        # outlier step
        outlier_step = 1.5 * IQR

        # Determine a list of indices of outliers for feature col
        outlier_list_col = df[(df[col] < Q1 - outlier_step) | (df[col] > Q3 + outlier_step )].index

        # append the found outlier indices for col to the list of outlier indices
        outlier_indices.extend(outlier_list_col)

    # select observations containing more than 2 outliers
    outlier_indices = Counter(outlier_indices)
    multiple_outliers = list( k for k, v in outlier_indices.items() if v > n )

    return multiple_outliers
outliers_to_drop=detect_outliers(df,2,['GRE Score', 'TOEFL Score', 'University Rating', 'SOP',
outliers_to_drop=detect_outliers(df,2,['GRE Score', 'TOEFL Score', 'University Rating', 'SOP',
       'LOR ', 'CGPA', 'Research'])
```

In [ ]:

```
df.loc[outliers_to_drop]
```

Out[ ]:

Serial No.	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit

In [ ]:

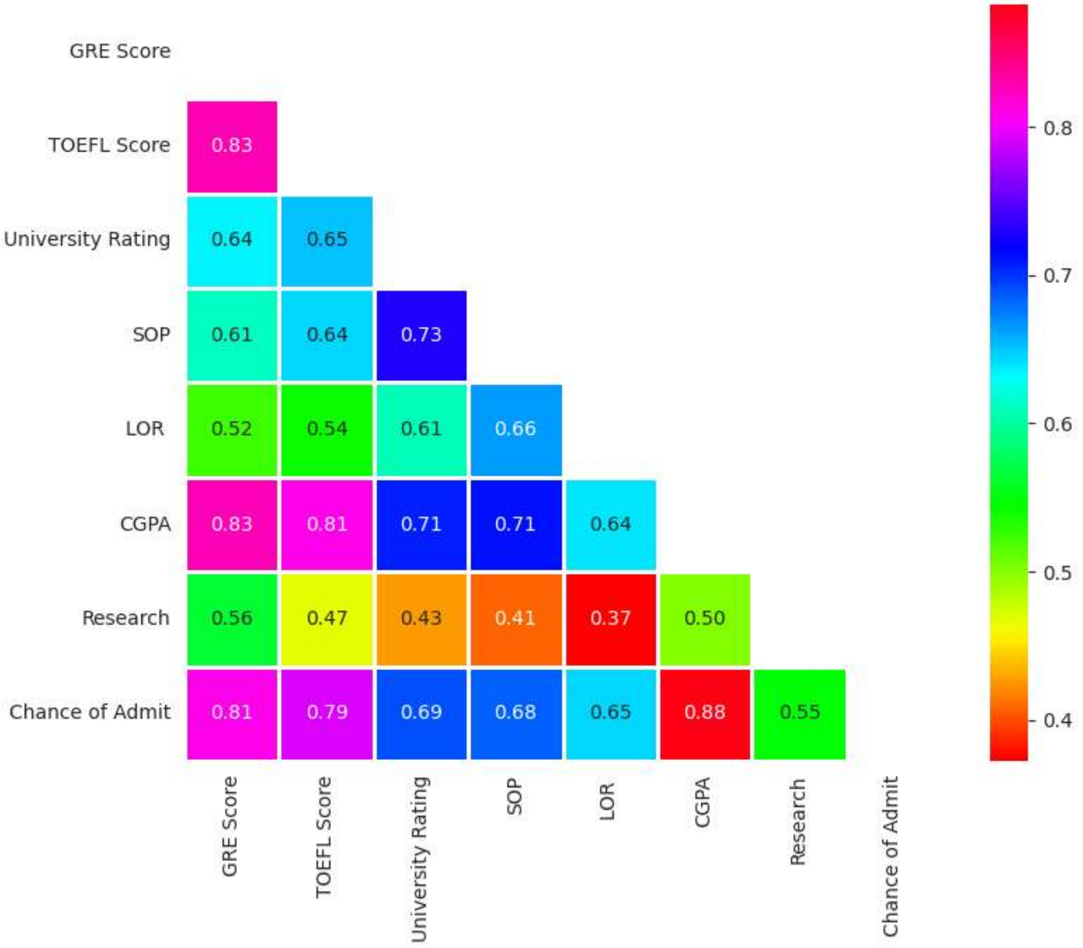
```
cols=df.drop(labels='Serial No.',axis=1)
cols.head().T
```

Out[ ]:

	0	1	2	3	4
GRE Score	337.00	324.00	316.00	322.00	314.00
TOEFL Score	118.00	107.00	104.00	110.00	103.00
University Rating	4.00	4.00	3.00	3.00	2.00
SOP	4.50	4.00	3.00	3.50	2.00
LOR	4.50	4.50	3.50	2.50	3.00
CGPA	9.65	8.87	8.00	8.67	8.21
Research	1.00	1.00	1.00	1.00	0.00
Chance of Admit	0.92	0.76	0.72	0.80	0.65

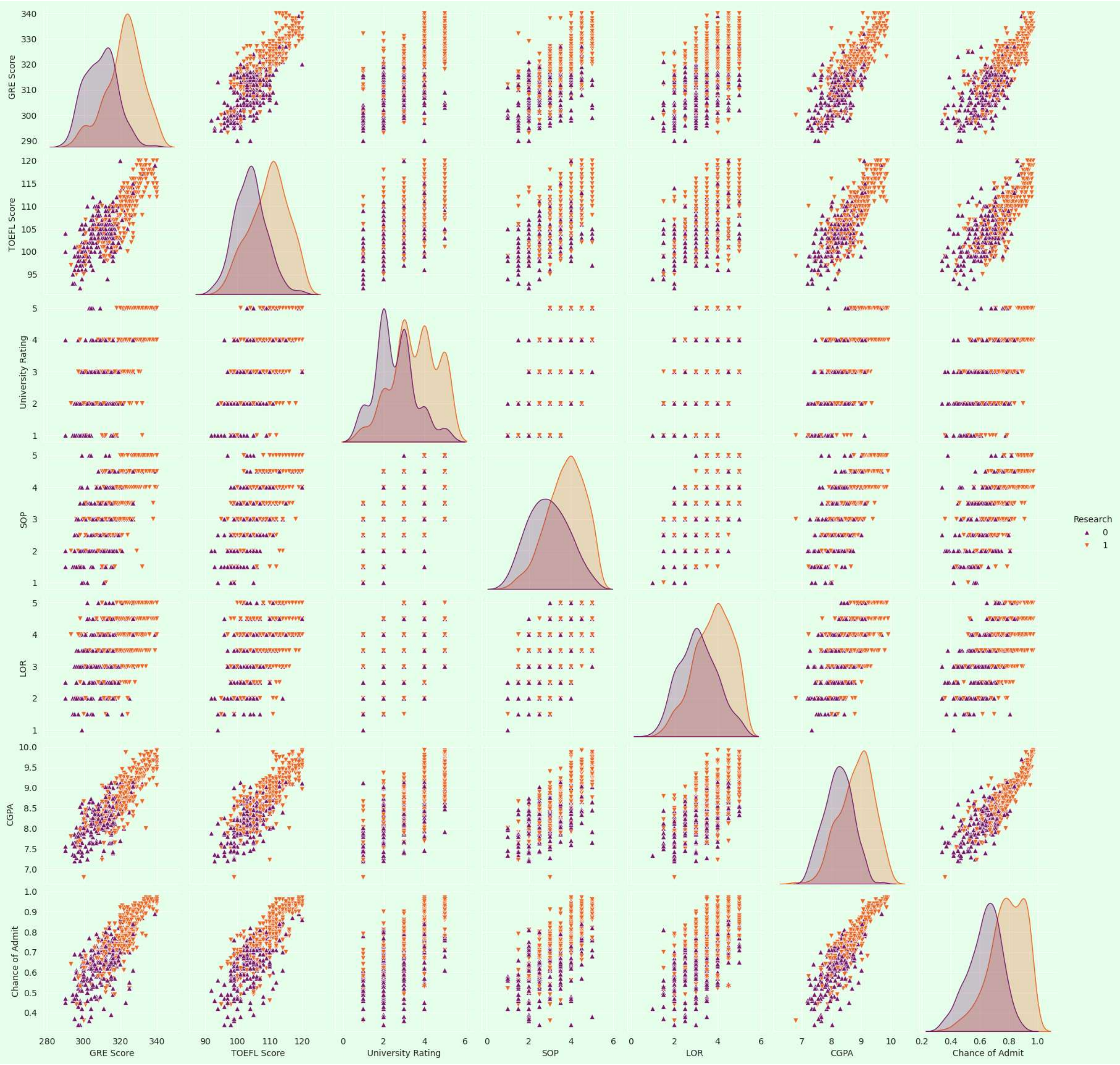
In [ ]:

```
  corr = cols.corr()
mask = np.zeros_like(corr)
mask[np.triu_indices_from(mask)] = True
with sns.axes_style("white"):
    f, ax = plt.subplots(figsize=(9, 7))
    ax = sns.heatmap(corr,mask=mask,square=True,annot=True,fmt='0.2f',linewidths=.8,cmap="hsv")
```



In [ ]:

```
plt.rcParams['axes.facecolor'] = "#e6ffed"
plt.rcParams['figure.facecolor'] = "#e6ffed"
g = sns.pairplot(data=cols,hue='Research',markers=["^", "v"],palette='inferno')
```

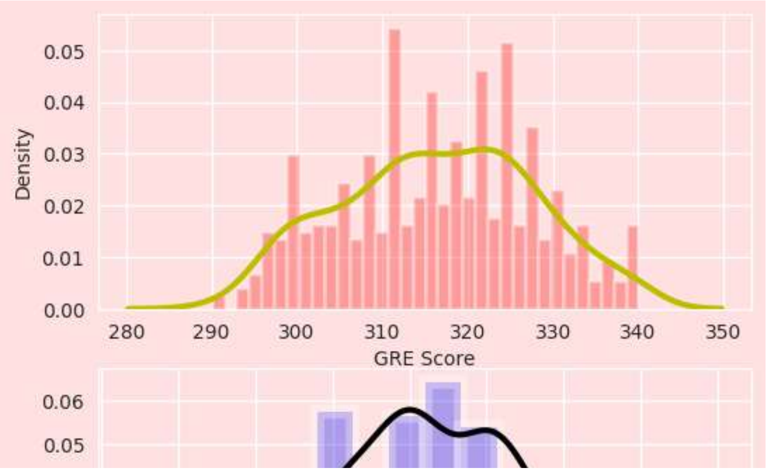


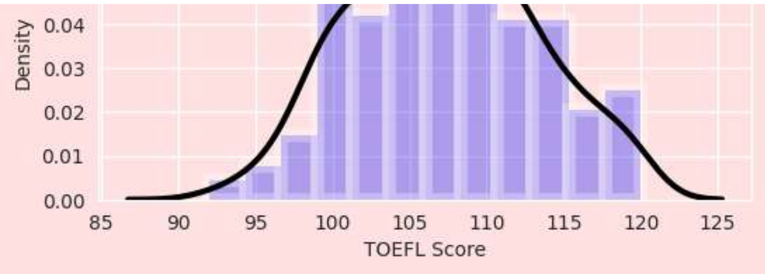
In [ ]:

```
plt.rcParams['axes.facecolor'] = "#ffe5e5"
plt.rcParams['figure.facecolor'] = "#ffe5e5"
plt.figure(figsize=(6,6))
plt.subplot(2, 1, 1)
sns.distplot(df['GRE Score'],bins=34,color='Red',  kde_kws={"color": "y", "lw": 3, "label": "KDE"},hist_kws={"linewidth":
plt.subplot(2, 1, 2)
sns.distplot(df['TOEFL Score'],bins=12,color='Blue' ,kde_kws={"color": "k", "lw": 3, "label": "KDE"},hist_kws={"linewidth'
```

Out[ ]:

```
<Axes: xlabel='TOEFL Score', ylabel='Density'>
```



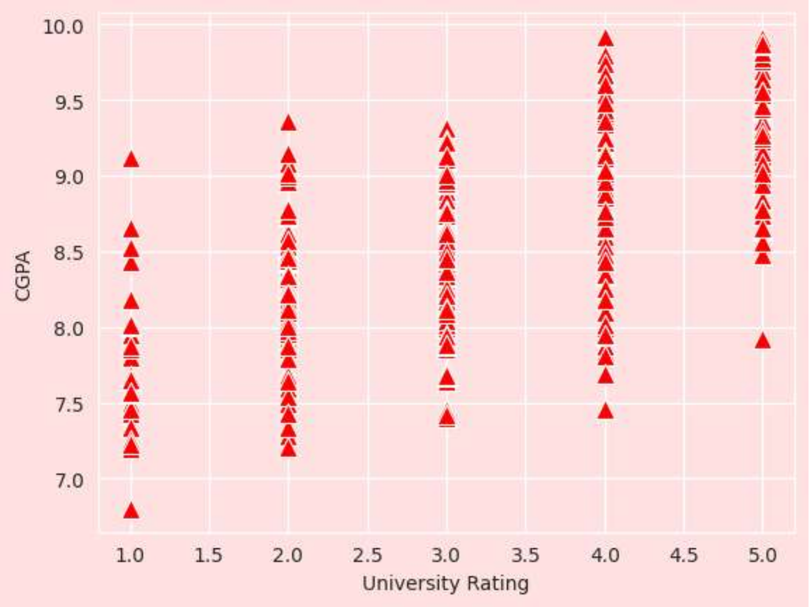


In [ ]:

```
sns.scatterplot(x='University Rating',y='CGPA',data=df,color='Red', marker="^", s=100)
```

Out[ ]:

```
<Axes: xlabel='University Rating', ylabel='CGPA'>
```

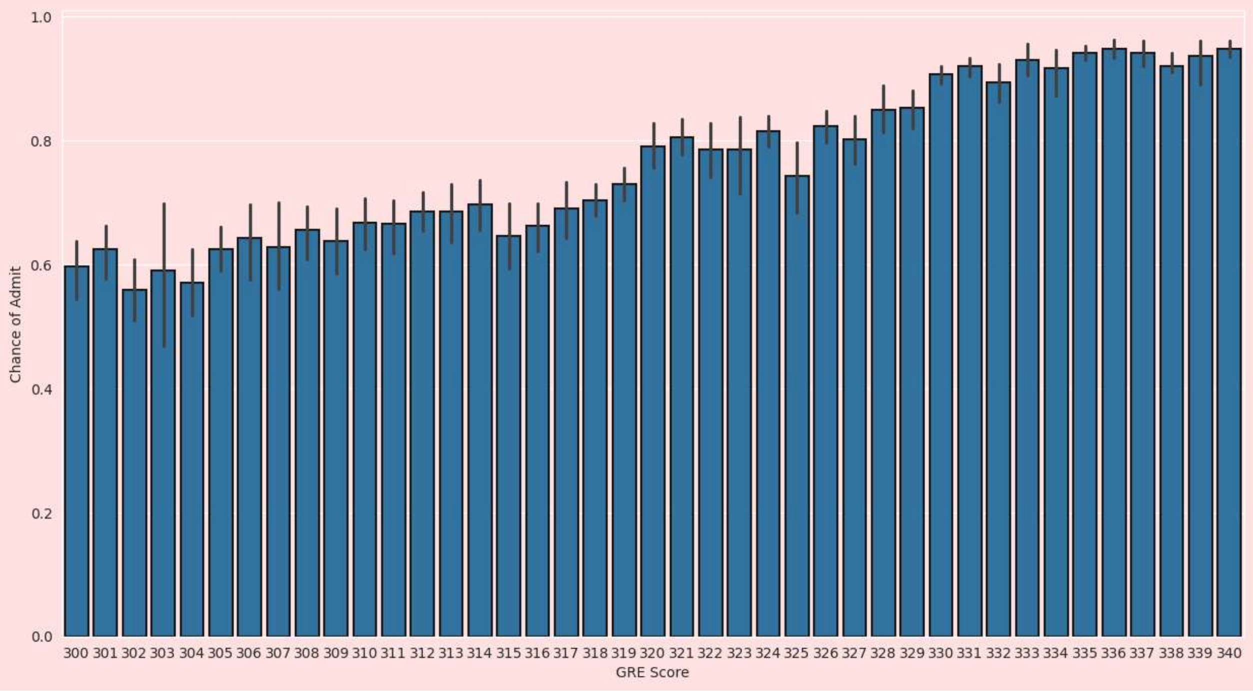


In [ ]:

```
co_gre=df[df["GRE Score"]>=300]
co_toefel=df[df["TOEFL Score"]>=100]
```

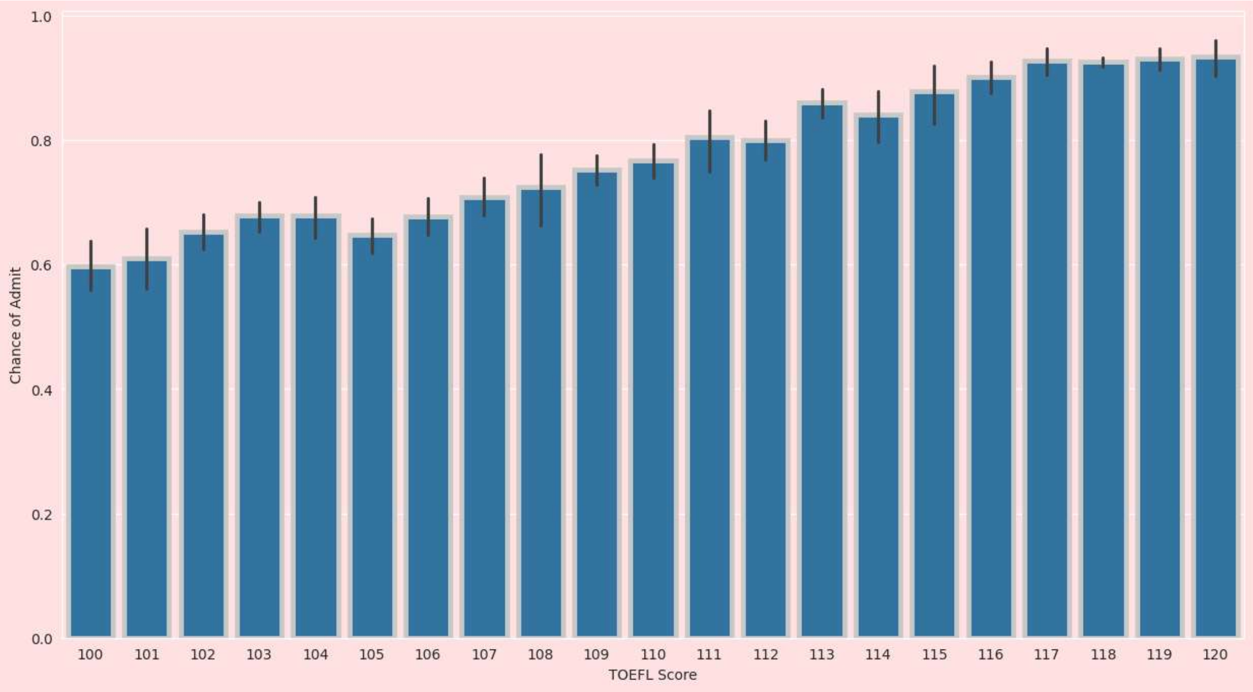
In [ ]:

```
fig, ax = pyplot.subplots(figsize=(15,8))
sns.barplot(x='GRE Score',y='Chance of Admit',data=co_gre, linewidth=1.5,edgecolor="0.1")
plt.show()
```



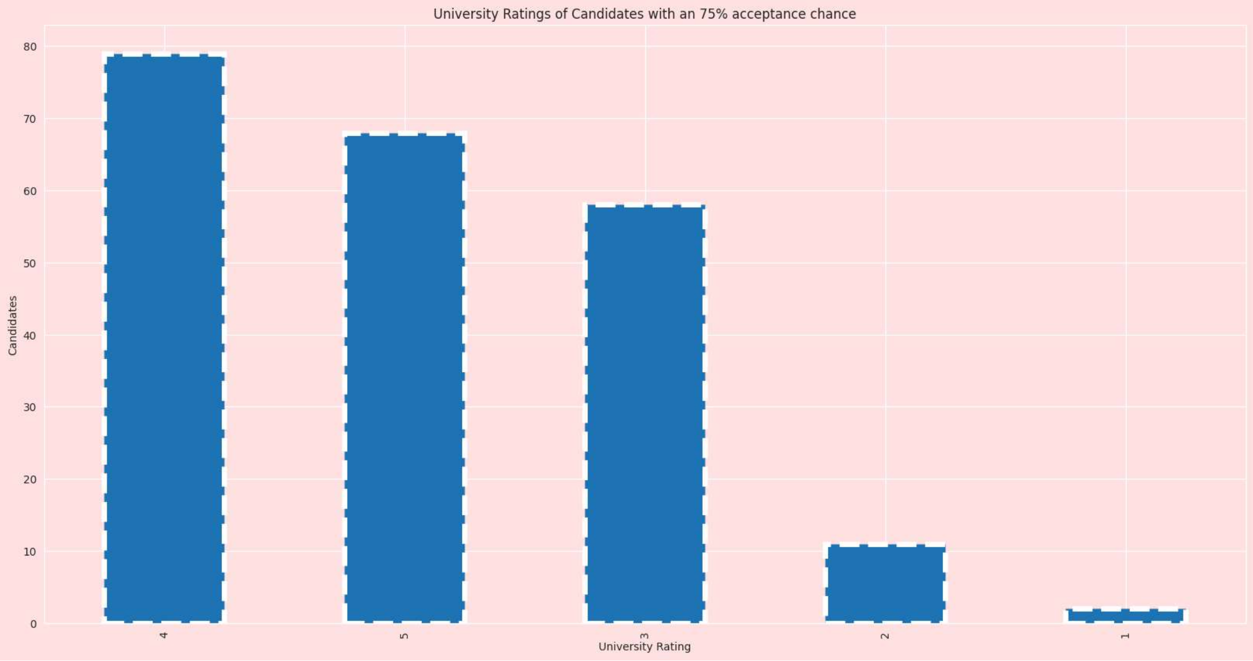
In [ ]:

```
fig, ax = pyplot.subplots(figsize=(15,8))
sns.barplot(x='TOEFL Score',y='Chance of Admit',data=co_toefel, linewidth=3.5,edgecolor="0.8")
plt.show()
```



In [ ]:

```
s = df[df["Chance of Admit"] >= 0.75]["University Rating"].value_counts().head(5)
plt.title("University Ratings of Candidates with an 75% acceptance chance")
s.plot(kind='bar',figsize=(20, 10),linestyle='dashed',linewidth=5)
plt.xlabel("University Rating")
plt.ylabel("Candidates")
plt.show()
```



In [ ]:

```
print("Average GRE Score :{0:.2f} out of 340".format(df['GRE Score'].mean()))
print('Average TOEFL Score:{0:.2f} out of 120'.format(df['TOEFL Score'].mean()))
print('Average CGPA:{0:.2f} out of 10'.format(df['CGPA'].mean()))
print('Average Chance of getting admitted:{0:.2f}%'.format(df['Chance of Admit'].mean()*100))
```

```
Average GRE Score :316.47 out of 340
Average TOEFL Score:107.19 out of 120
Average CGPA:8.58 out of 10
Average Chance of getting admitted:72.17%
```

In [ ]:

```
toppers=df[(df['GRE Score']>=330) & (df['TOEFL Score']>=115) & (df['CGPA']>=9.5)].sort_values(by=['Chance of Admit'],ascen
toppers
```

Out[ ]:

	Serial No.	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit
202	203	340	120	5	4.5	4.5	9.91	1	0.97
143	144	340	120	4	4.5	4.0	9.92	1	0.97
24	25	336	119	5	4.0	3.5	9.80	1	0.97
203	204	334	120	5	4.0	5.0	9.87	1	0.97
213	214	333	119	5	5.0	4.5	9.78	1	0.96
385	386	335	117	5	5.0	5.0	9.82	1	0.96
148	149	339	116	4	4.0	3.5	9.80	1	0.96
81	82	340	120	4	5.0	5.0	9.50	1	0.96
496	497	337	117	5	5.0	5.0	9.87	1	0.96
23	24	334	119	5	5.0	4.5	9.70	1	0.95
212	213	338	120	4	5.0	5.0	9.66	1	0.95
399	400	333	117	4	5.0	4.0	9.66	1	0.95
372	373	336	119	4	4.5	4.0	9.62	1	0.95
120	121	335	117	5	5.0	5.0	9.56	1	0.94
70	71	332	118	5	5.0	5.0	9.64	1	0.94
193	194	336	118	5	4.5	5.0	9.53	1	0.94
25	26	340	120	5	4.5	4.5	9.60	1	0.94
423	424	334	119	5	4.5	5.0	9.54	1	0.94
497	498	330	120	5	4.5	5.0	9.56	1	0.93
361	362	334	116	4	4.0	3.5	9.54	1	0.93
253	254	335	115	4	4.5	4.5	9.68	1	0.93
0	1	337	118	4	4.5	4.5	9.65	1	0.92
47	48	339	119	5	4.5	4.0	9.70	0	0.89

In [ ]:

```
serialNo = df["Serial No."].values

df.drop(["Serial No."],axis=1,inplace = True)

df=df.rename(columns = {'Chance of Admit ':'Chance of Admit'})
```

In [ ]:

```
X=df.drop('Chance of Admit',axis=1)
y=df['Chance of Admit']
```

In [ ]:

```
from sklearn.model_selection import train_test_split
from sklearn import preprocessing

#Normalisation works slightly better for Regression.
X_norm=preprocessing.normalize(X)
X_train,X_test,y_train,y_test=train_test_split(X_norm,y,test_size=0.20,random_state=101)
```

In [ ]:

```
from sklearn.linear_model import LinearRegression,LogisticRegression
from sklearn.tree import DecisionTreeRegressor,DecisionTreeClassifier
from sklearn.ensemble import RandomForestRegressor,RandomForestClassifier
from sklearn.ensemble import GradientBoostingRegressor,GradientBoostingClassifier
from sklearn.ensemble import AdaBoostRegressor,AdaBoostClassifier
from sklearn.ensemble import ExtraTreesRegressor,ExtraTreesClassifier
from sklearn.neighbors import KNeighborsRegressor,KNeighborsClassifier
from sklearn.svm import SVR,SVC
from sklearn.naive_bayes import GaussianNB

from sklearn.metrics import accuracy_score,mean_squared_error
```

In [ ]:

```
regressors=[['Linear Regression :',LinearRegression()],
regressors=[['Linear Regression :',LinearRegression()],
        ['Decision Tree Regression :',DecisionTreeRegressor()],
        ['Random Forest Regression :',RandomForestRegressor()],
        ['Gradient Boosting Regression :', GradientBoostingRegressor()],
        ['Ada Boosting Regression :',AdaBoostRegressor()],
        ['Extra Tree Regression :', ExtraTreesRegressor()],
        ['K-Neighbors Regression :',KNeighborsRegressor()],
        ['Support Vector Regression :',SVR()]]
reg_pred=[]
print('Results...\n')
for name,model in regressors:
    model=model
    model.fit(X_train,y_train)
    predictions = model.predict(X_test)
    rms=np.sqrt(mean_squared_error(y_test, predictions))
    reg_pred.append(rms)
    print(name,rms)
```

```
Results...

Linear Regression : 0.07765759656302859
Decision Tree Regression : 0.09757561170702442
Random Forest Regression : 0.07632554225159488
Gradient Boosting Regression : 0.07385762575059364
Ada Boosting Regression : 0.08731672803937653
Extra Tree Regression : 0.07381146455666628
K-Neighbors Regression : 0.08882567196480981
Support Vector Regression : 0.11746039395819052
```

In [ ]:

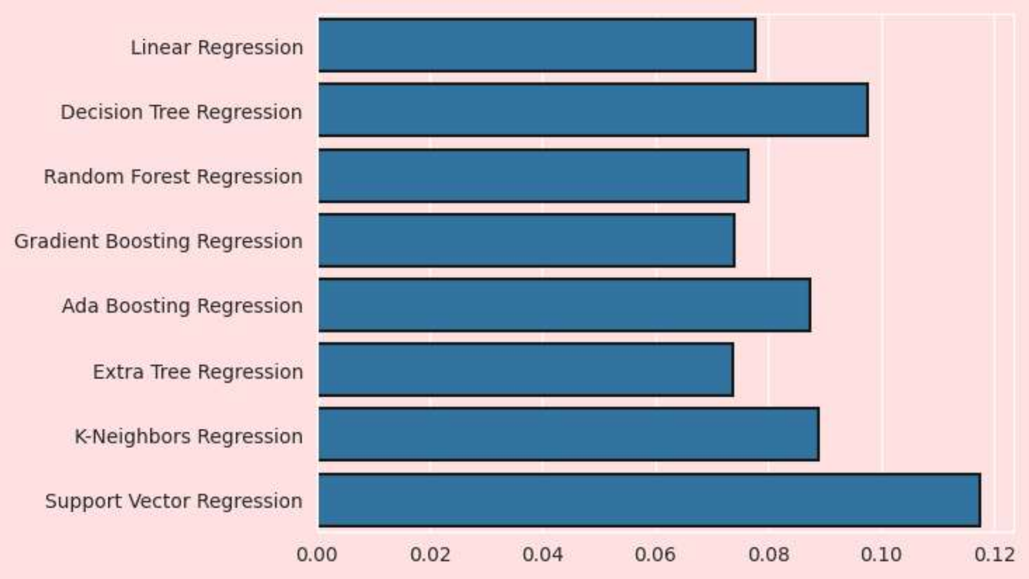
```
y_ax=['Linear Regression' ,'Decision Tree Regression', 'Random Forest Regression','Gradient Boosting Regression', 'Ada Bo
x_ax=reg_pred
```

In [ ]:

```
sns.barplot(x=x_ax,y=y_ax,linewidth=1.5,edgecolor="0.1")
```

Out[ ]:

```
<Axes: >
```



In [ ]:

```
from sklearn.model_selection import train_test_split

X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.20,random_state=101)
```

In [ ]:

```
#If Chance of Admit greater than 80% we classify it as 1
y_train_c = [1 if each > 0.8 else 0 for each in y_train]
y_test_c  = [1 if each > 0.8 else 0 for each in y_test]
```

In [ ]:

```
classifiers=[['Logistic Regression :',LogisticRegression()],
        ['Decision Tree Classification :',DecisionTreeClassifier()],
        ['Random Forest Classification :',RandomForestClassifier()],
        ['Gradient Boosting Classification :', GradientBoostingClassifier()],
        ['Ada Boosting Classification :',AdaBoostClassifier()],
        ['Extra Tree Classification :', ExtraTreesClassifier()],
        ['K-Neighbors Classification :',KNeighborsClassifier()],
        ['Support Vector Classification :',SVC()],
        ['Gausian Naive Bayes :',GaussianNB()]]
cla_pred=[]
```

```
cla_pred=[]
for name,model in classifiers:
    model=model
    model.fit(X_train,y_train_c)
    predictions = model.predict(X_test)
    cla_pred.append(accuracy_score(y_test_c,predictions))
    print(name,accuracy_score(y_test_c,predictions))
```

```
Logistic Regression : 0.89
Decision Tree Classification : 0.92
Random Forest Classification : 0.91
Gradient Boosting Classification : 0.91
Ada Boosting Classification : 0.88
Extra Tree Classification : 0.91
K-Neighbors Classification : 0.84
Support Vector Classification : 0.7
Gausian Naive Bayes : 0.89
```

In [ ]:

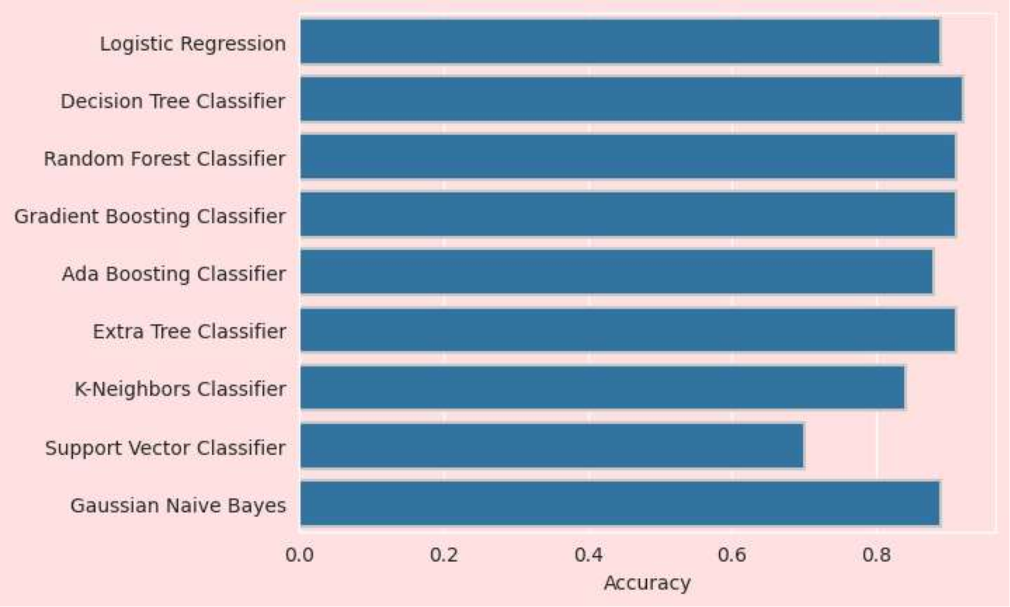
```
y_ax=['Logistic Regression' ,
      'Decision Tree Classifier',
      'Random Forest Classifier',
      'Gradient Boosting Classifier',
      'Ada Boosting Classifier',
      'Extra Tree Classifier' ,
      'K-Neighbors Classifier',
      'Support Vector Classifier',
       'Gaussian Naive Bayes']
x_ax=cla_pred
```

In [ ]:

```
sns.barplot(x=x_ax,y=y_ax,linewidth=1.5,edgecolor="0.8")
plt.xlabel('Accuracy')
```

Out[ ]:

```
Text(0.5, 0, 'Accuracy')
```



In [ ]: