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

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



```
import os
dataset_path='/content/drive/MyDrive/ML_Dataset/Admission_Predict_Ver1.1.csv'
```

```
import pandas as pd
import numpy as np
```

```
import matplotlib.pyplot as plt #data visualization
import seaborn as sns #statistical data visualisation
```

```
df=pd.read_csv(dataset_path)
df.head()
```

	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

Next steps: Generate code with df | View recommended plots | New interactive sheet

```
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
```

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

```
df.describe()
```

	Serial No.	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit
count	500.000000	500.000000	500.000000	500.000000	500.000000	500.00000	500.000000	500.000000	500.00000
mean	250.500000	316.472000	107.192000	3.114000	3.374000	3.48400	8.576440	0.560000	0.72174
std	144.481833	11.295148	6.081868	1.143512	0.991004	0.92545	0.604813	0.496884	0.14114
min	1.000000	290.000000	92.000000	1.000000	1.000000	1.00000	6.800000	0.000000	0.34000
25%	125.750000	308.000000	103.000000	2.000000	2.500000	3.00000	8.127500	0.000000	0.63000
50%	250.500000	317.000000	107.000000	3.000000	3.500000	3.50000	8.560000	1.000000	0.72000
75%	375.250000	325.000000	112.000000	4.000000	4.000000	4.00000	9.040000	1.000000	0.82000
max	500.000000	340.000000	120.000000	5.000000	5.000000	5.00000	9.920000	1.000000	0.97000

```
l = df.columns
print('The columns are: ',l)
```

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

Enable browser notifications in Settings to get alerts when executions complete

```
print(df.isnull().sum())
print('\n\nNo null values')
```

```
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


No null values
```

```
df.describe().T #transpose
```

	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

```
df.describe()
```

	Serial No.	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit
count	500.000000	500.000000	500.000000	500.000000	500.000000	500.00000	500.000000	500.000000	500.00000
mean	250.500000	316.472000	107.192000	3.114000	3.374000	3.48400	8.576440	0.560000	0.72174
std	144.481833	11.295148	6.081868	1.143512	0.991004	0.92545	0.604813	0.496884	0.14114
min	1.000000	290.000000	92.000000	1.000000	1.000000	1.00000	6.800000	0.000000	0.34000
25%	125.750000	308.000000	103.000000	2.000000	2.500000	3.00000	8.127500	0.000000	0.63000
50%	250.500000	317.000000	107.000000	3.000000	3.500000	3.50000	8.560000	1.000000	0.72000
75%	375.250000	325.000000	112.000000	4.000000	4.000000	4.00000	9.040000	1.000000	0.82000
max	500.000000	340.000000	120.000000	5.000000	5.000000	5.00000	9.920000	1.000000	0.97000

```
from collections import Counter # Import the Counter class

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',
                                           'LOR ', 'CGPA', 'Research'])
```



```
df.loc[outliers_to_drop] # Show the outliers rows
```

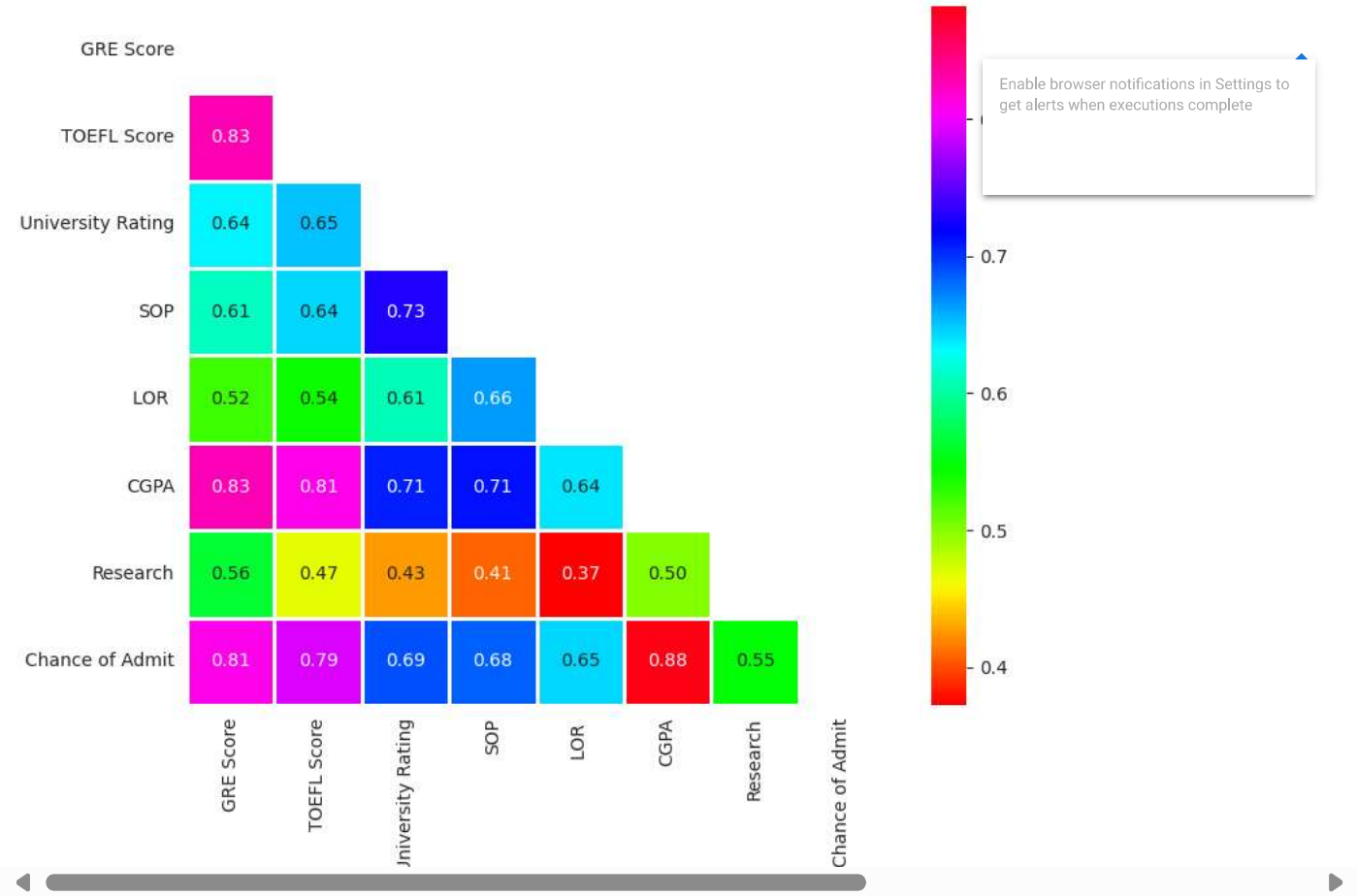
Serial No.	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit

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

	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

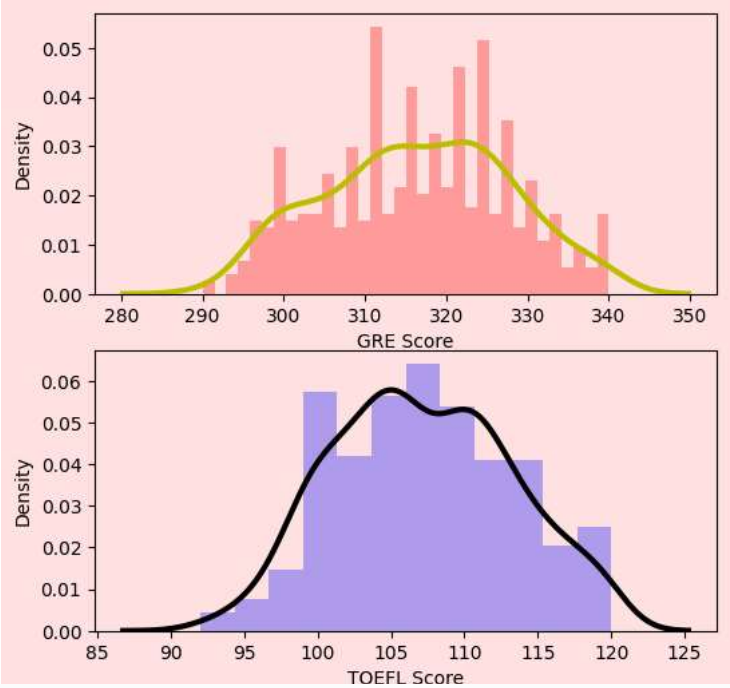
Next steps: Generate code with cols | View recommended plots | New interactive sheet

```
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")
```



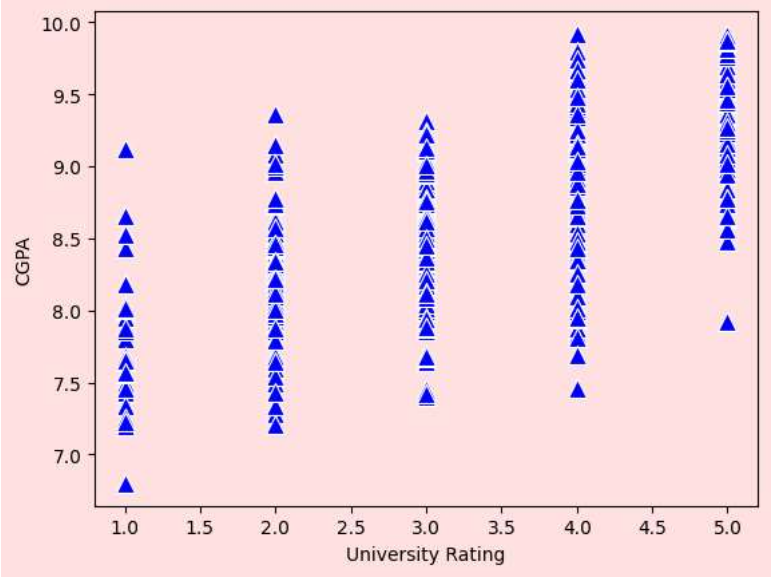

```
plt.rcParams['axes.facecolor'] = "#ffe5e5"
plt.rcParams['figure.facecolor'] = "#ffe5e5"
plt.figure(figsize=(6,6))
plt.subplot(2, 1, 1)
sns.distplot(df['GRE Score'],bins=34,color='Red',  kde_kws={"color": "y", "lw": 3, "label": "KDE"},hist_kws={"linewidth": 2,"alpha": 0.3 })
plt.subplot(2, 1, 2)
sns.distplot(df['TOEFL Score'],bins=12,color='Blue' ,kde_kws={"color": "k", "lw": 3, "label": "KDE"},hist_kws={"linewidth": 7,"alpha": 0.3 }
```

```
<ipython-input-18-6f2b0ed756ae>:5: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df['GRE Score'],bins=34,color='Red',  kde_kws={"color": "y", "lw": 3, "label": "KDE"},hist_kws={"linewidth": 2,"alpha": 0
<ipython-input-18-6f2b0ed756ae>:7: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df['TOEFL Score'],bins=12,color='Blue' ,kde_kws={"color": "k", "lw": 3, "label": "KDE"},hist_kws={"linewidth": 7,"alpha":
<Axes: xlabel='TOEFL Score', ylabel='Density'>
```



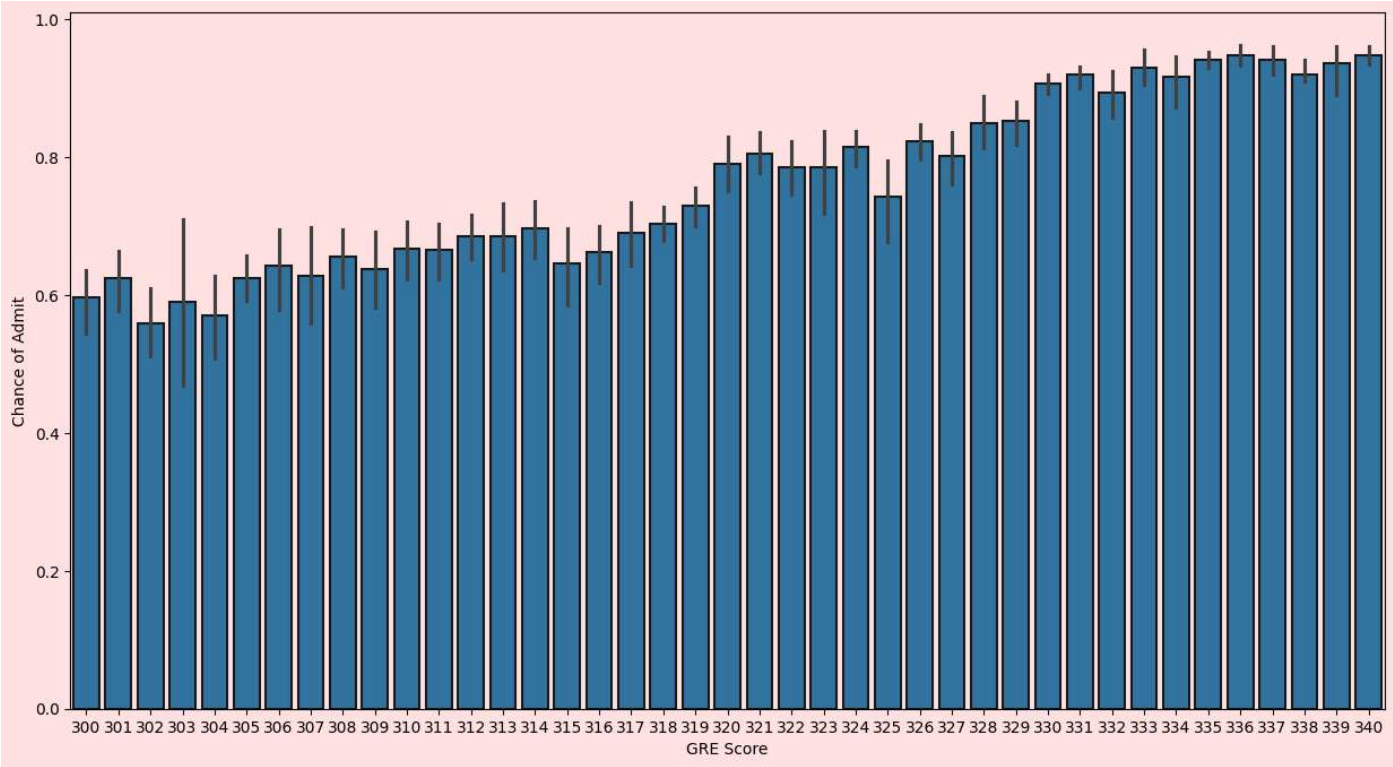

```
sns.scatterplot(x='University Rating',y='CGPA',data=df,color='Blue', marker="^", s=100)
```

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

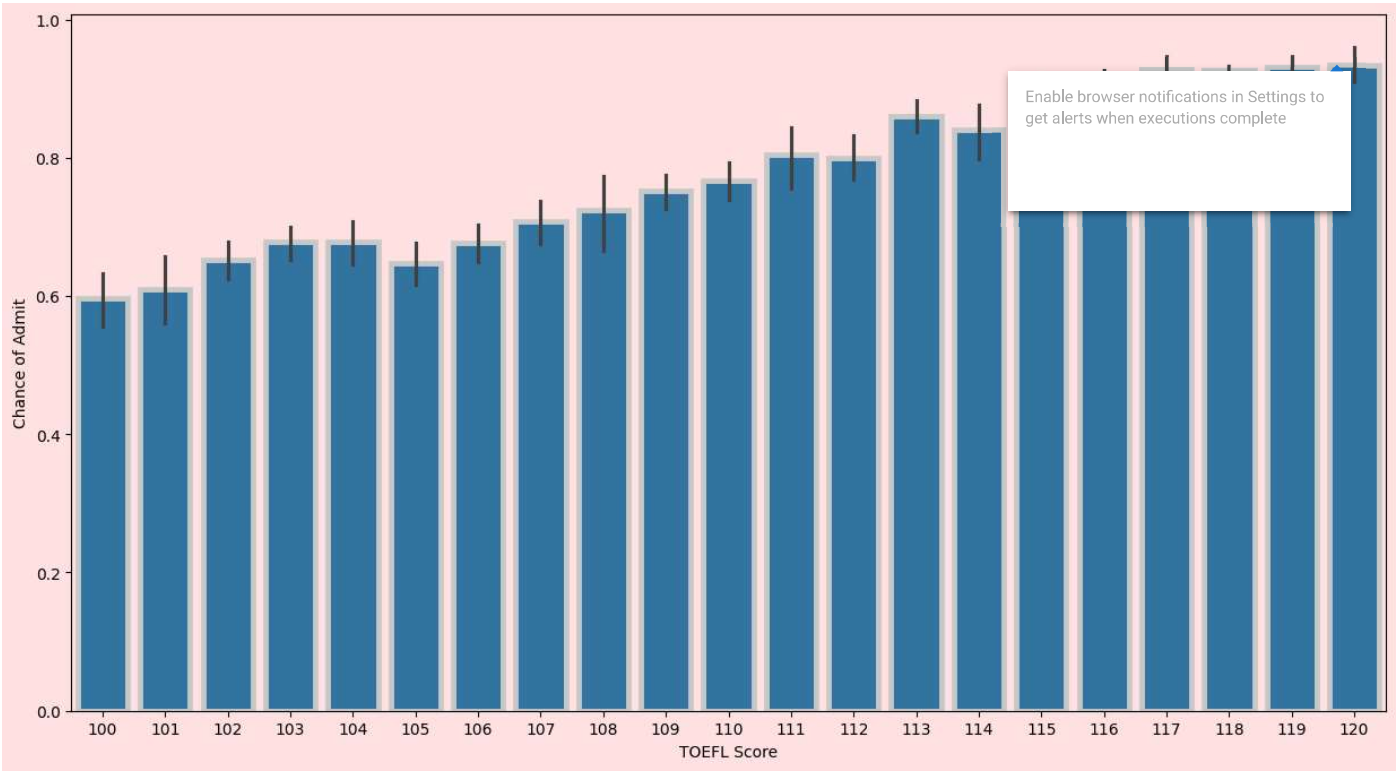


```
co_gre=df[df["GRE Score"]>=300]
co_toefel=df[df["TOEFL Score"]>=100]


import matplotlib.pyplot as plt #data visualization

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

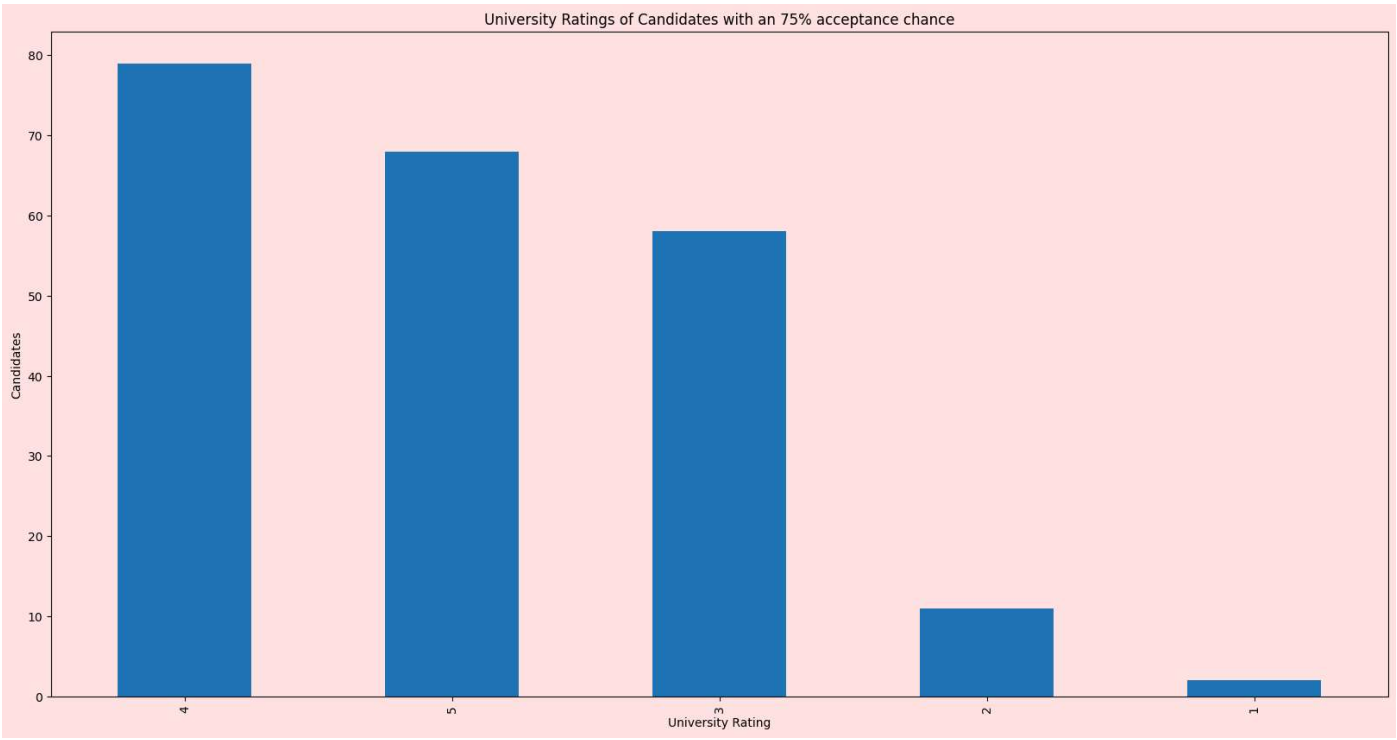
Enable browser notifications in Settings to get alerts when executions complete




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




```
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()
```



```
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%
```



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

	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

Next steps: Generate code with toppers | View recommended plots | New interactive sheet

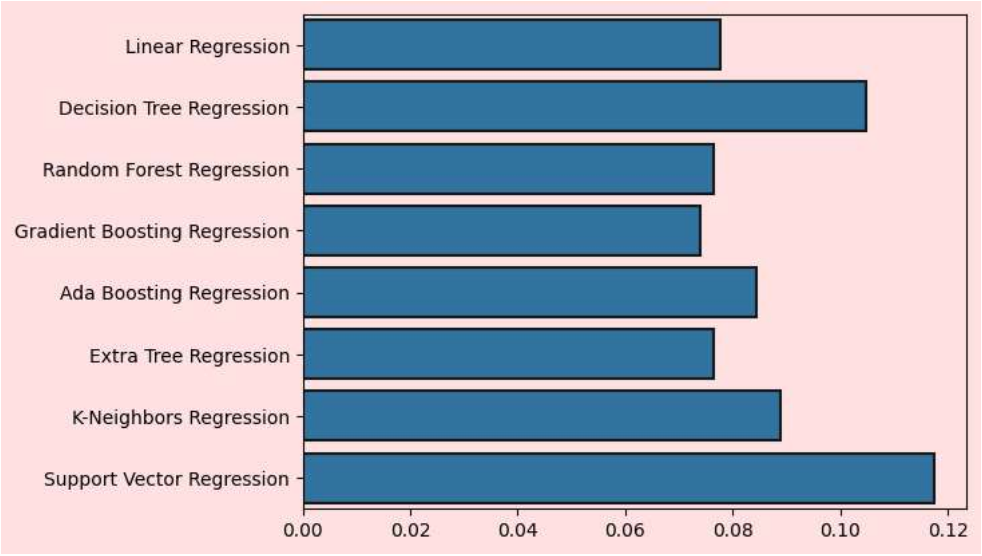
```
# reading the dataset
# Assuming the file is in the same directory as the notebook

# Specify the full path to the dataset
dataset_path = '/content/drive/MyDrive/ML_Dataset/Admission_Predict_Ver1.1.csv'
df = pd.read_csv(dataset_path, sep=",")

# it may be needed in the future.
serialNo = df["Serial No."].values

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

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


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


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)


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


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 : 0.10471389592599445
Random Forest Regression : 0.0764138972962379
Gradient Boosting Regression : 0.07387617980703885
Ada Boosting Regression : 0.0844802179885863
Extra Tree Regression : 0.07644006606485895
K-Neighbors Regression : 0.08882567196480981
Support Vector Regression : 0.11746039395819052
```

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


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

```
<Axes: >
```



```
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)


#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]


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=[]
for name,model in classifiers:
    model=model
    model.fit(X_train,y_train_c)
```