```
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

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

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

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

df.head(10).T
```


```
{"summary":"{\n  \"name\": \"df\",\n  \"rows\": 500,\n  \"fields\": [\
n    {\n      \"column\": \"Serial No.\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 144,\n        \"min\": 1,\n
\"max\": 500,\n        \"num_unique_values\": 500,\n
\"samples\": [\n          362,\n          74,\n          375\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"GRE
Score\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 11,\n        \"min\": 290,\n        \"max\": 340,\n
\"num_unique_values\": 49,\n        \"samples\": [\n          307,\n
335,\n          297\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"TOEFL Score\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 6,\n        \"min\": 92,\n
\"max\": 120,\n        \"num_unique_values\": 29,\n
\"samples\": [\n          94,\n          119,\n          112\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"University Rating\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 1,\n        \"min\": 1,\n
\"max\": 5,\n        \"num_unique_values\": 5,\n        \"samples\":
[\n          3,\n          1,\n          2\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"SOP\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 0.9910036207566069,\n
\"min\": 1.0,\n        \"max\": 5.0,\n        \"num_unique_values\":
9,\n        \"samples\": [\n          1.0,\n          4.0,\n
```

```
5.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"LOR
\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 0.9254495738978181,\n        \"min\": 1.0,\n        \"max\":
5.0,\n        \"num_unique_values\": 9,\n        \"samples\": [\n
5.0,\n          3.5,\n          1.5\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"CGPA\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 0.6048128003332052,\n
\"min\": 6.8,\n        \"max\": 9.92,\n        \"num_unique_values\":
184,\n        \"samples\": [\n          9.6,\n          8.9,\n
8.24\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Research\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0,\n        \"min\": 0,\n
\"max\": 1,\n        \"num_unique_values\": 2,\n        \"samples\":
[\n          0,\n          1\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"Chance of Admit \",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 0.1411404039503023,\n
\"min\": 0.34,\n        \"max\": 0.97,\n        \"num_unique_values\":
61,\n        \"samples\": [\n          0.92,\n          0.9\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe","variable_name":"df"}
```


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

```
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]
```


```
{"summary":"{\n  \"name\": \"df\",\n  \"rows\": 0,\n  \"fields\": [\n
{\n      \"column\": \"Serial No.\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": null,\n        \"min\":
null,\n        \"max\": null,\n        \"num_unique_values\": 0,\n
\"samples\": [],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"GRE
Score\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": null,\n        \"min\": null,\n        \"max\": null,\n
\"num_unique_values\": 0,\n        \"samples\": [],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"TOEFL Score\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
null,\n        \"min\": null,\n        \"max\": null,\n
\"num_unique_values\": 0,\n        \"samples\": [],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"University Rating\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
null,\n        \"min\": null,\n        \"max\": null,\n
\"num_unique_values\": 0,\n        \"samples\": [],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"SOP\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": null,\n        \"min\":
null,\n        \"max\": null,\n        \"num_unique_values\": 0,\n
\"samples\": [],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"LOR
\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": null,\n        \"min\": null,\n        \"max\": null,\n
\"num_unique_values\": 0,\n        \"samples\": [],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"CGPA\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": null,\n        \"min\":
```

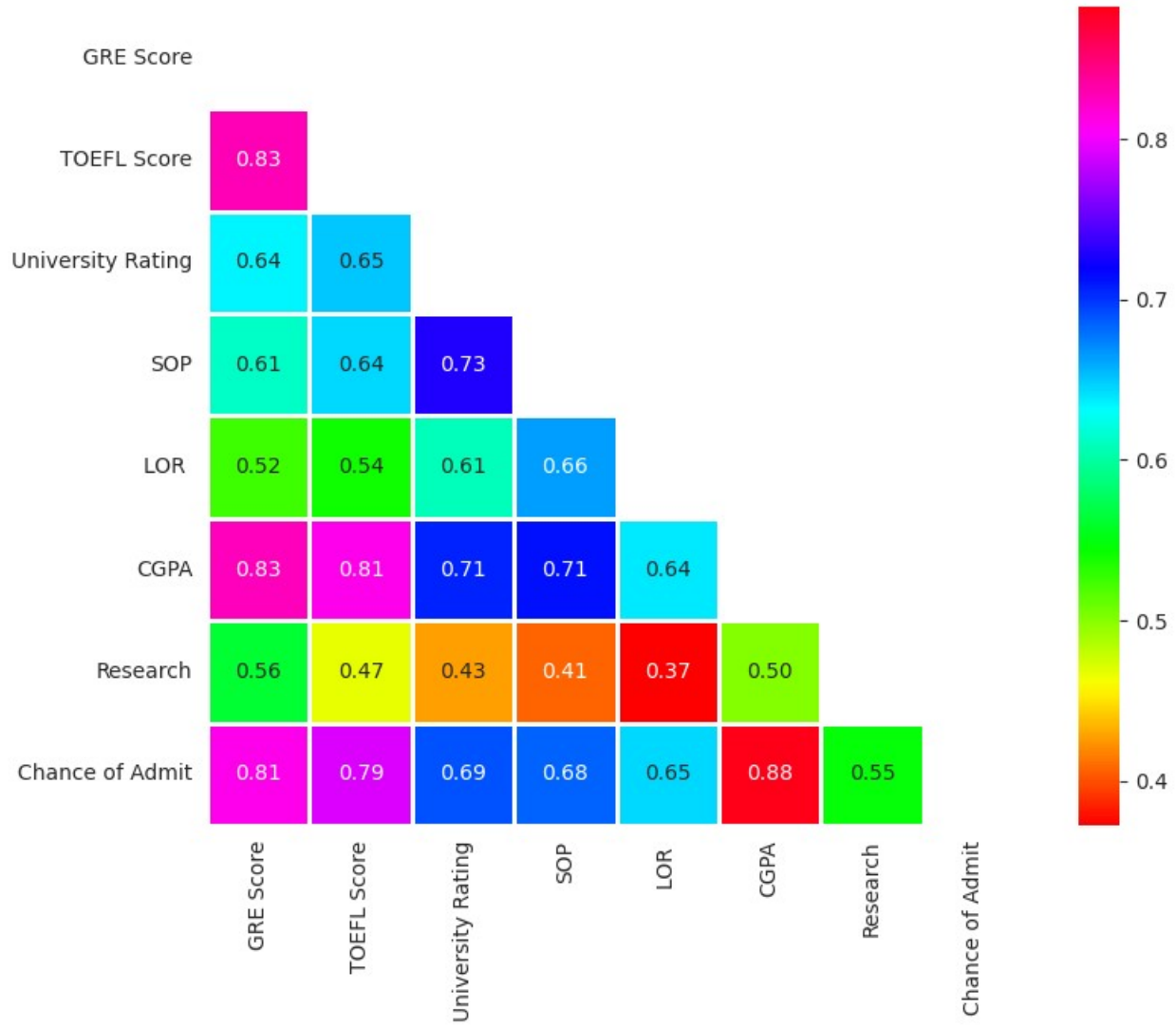
```
null,\n        \"max\": null,\n        \"num_unique_values\": 0,\n
\"samples\": [],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Research\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": null,\n        \"min\": null,\n
\"max\": null,\n        \"num_unique_values\": 0,\n
\"samples\": [],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Chance of Admit\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": null,\n        \"min\": null,\n
\"max\": null,\n        \"num_unique_values\": 0,\n
\"samples\": [],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}
```


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

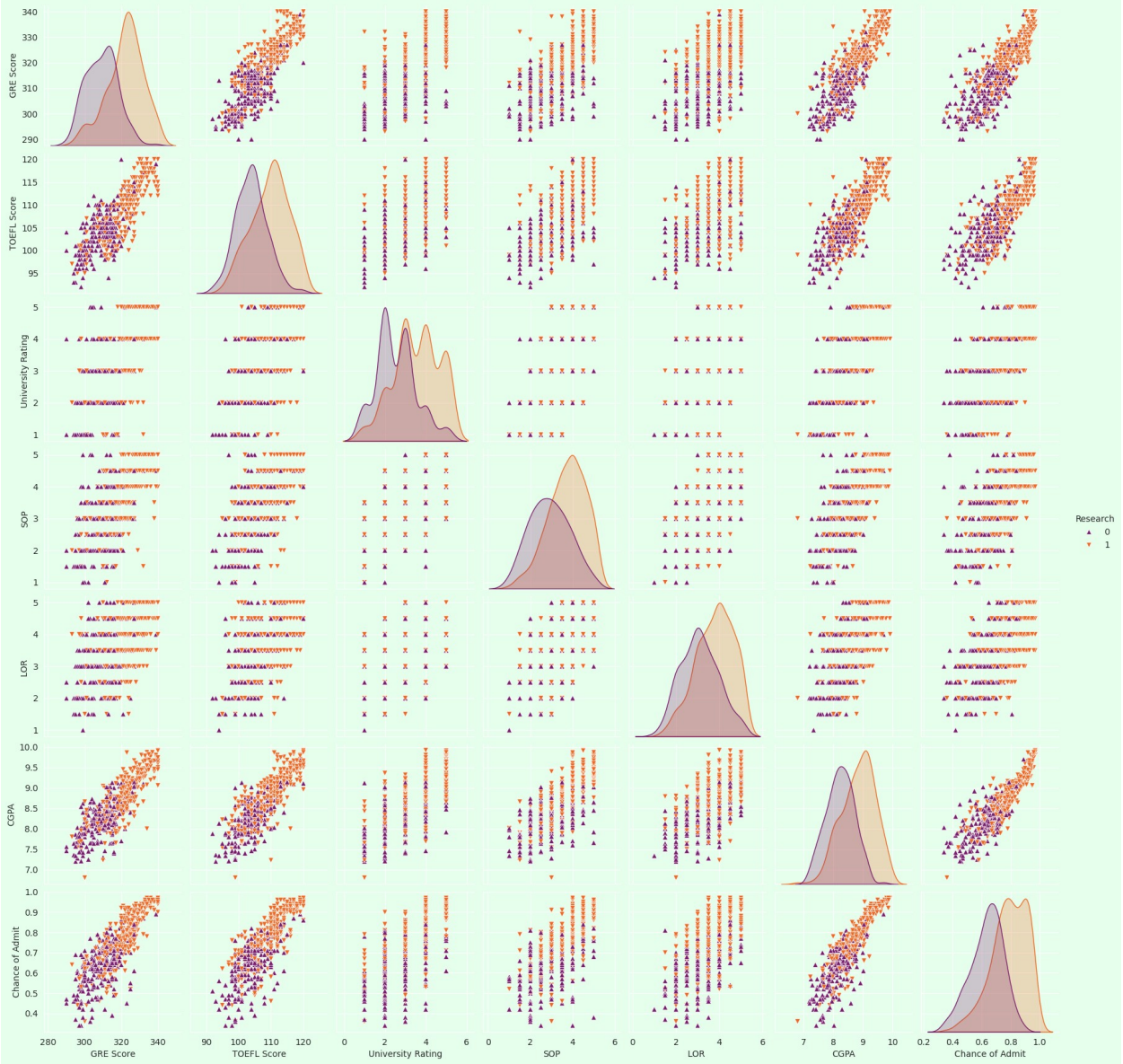

```
{"summary":"{\n  \"name\": \"cols\",\n  \"rows\": 500,\n  \"fields\":
[\n    {\n      \"column\": \"GRE Score\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 11,\n        \"min\": 290,\n
\"max\": 340,\n        \"num_unique_values\": 49,\n
\"samples\": [\n          307,\n          335,\n          297\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"TOEFL Score\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
6,\n        \"min\": 92,\n        \"max\": 120,\n
\"num_unique_values\": 29,\n        \"samples\": [\n          94,\n
119,\n          112\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"University Rating\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 1,\n        \"min\": 1,\n
\"max\": 5,\n        \"num_unique_values\": 5,\n        \"samples\":
[\n          3,\n          1,\n          2\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"SOP\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 0.9910036207566069,\n
\"min\": 1.0,\n        \"max\": 5.0,\n        \"num_unique_values\":
9,\n        \"samples\": [\n          1.0,\n          4.0,\n
5.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"LOR
\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 0.9254495738978181,\n        \"min\": 1.0,\n        \"max\":
5.0,\n        \"num_unique_values\": 9,\n        \"samples\": [\n
5.0,\n          3.5,\n          1.5\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"CGPA\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 0.6048128003332052,\n
\"min\": 6.8,\n        \"max\": 9.92,\n        \"num_unique_values\":
184,\n        \"samples\": [\n          9.6,\n          8.9,\n
8.24\n        ],\n        \"semantic_type\": \"\",\n
```

\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"Research\",\n   \"properties\": {\n   \"dtype\":
\"number\",\n   \"std\": 0,\n   \"min\": 0,\n
\"max\": 1,\n   \"num_unique_values\": 2,\n   \"samples\":
[\n   0,\n   1\n   ],\n   \"semantic_type\":
\"\",\n   \"description\": \"\"\n   }\n   },\n   {\n
\"column\": \"Chance of Admit\",\n   \"properties\": {\n
\"dtype\": \"number\",\n   \"std\": 0.1411404039503023,\n
\"min\": 0.34,\n   \"max\": 0.97,\n   \"num_unique_values\":
61,\n   \"samples\": [\n   0.92,\n   0.9\
n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   }\n]\
n}","type":"dataframe","variable_name":"cols"}

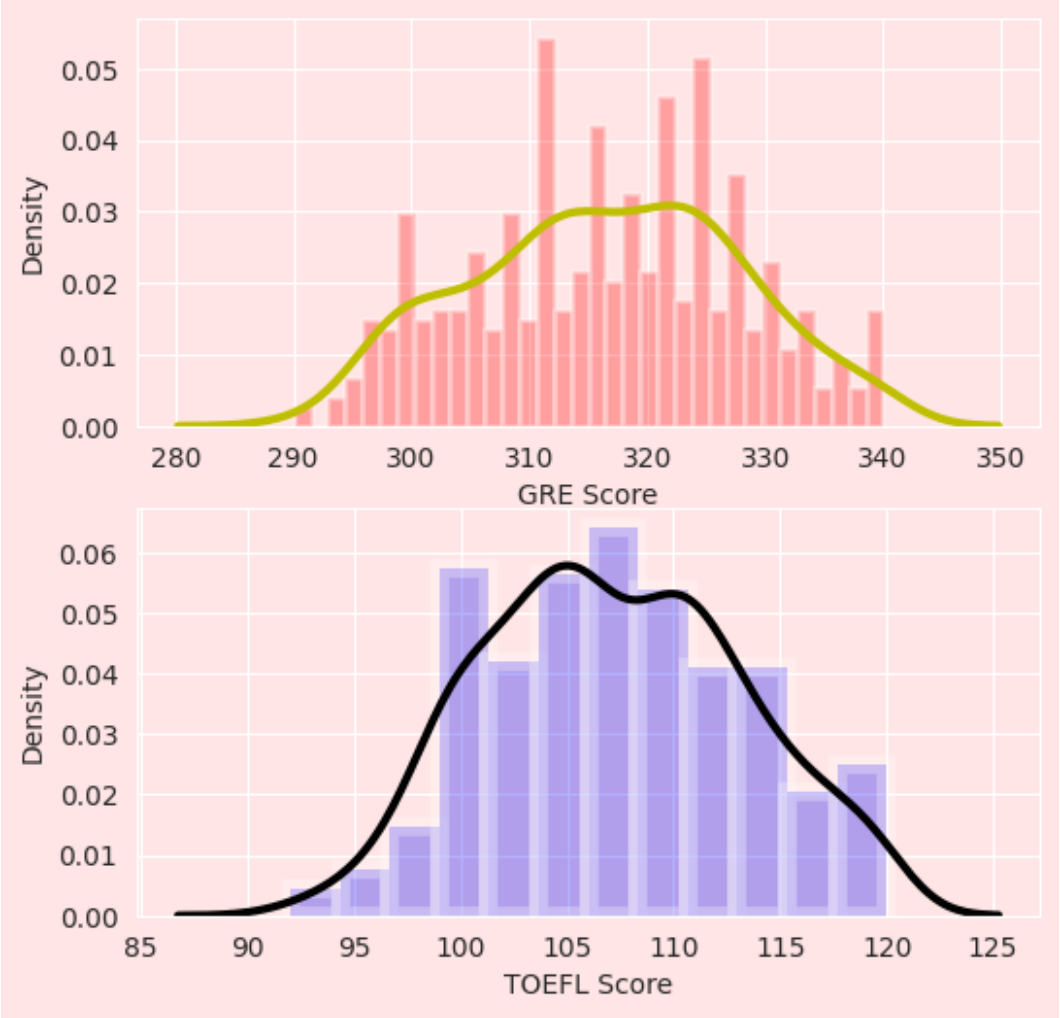
```
  corr = cols.corr()
mask = np.zeros_like(corr)
mask[np.triu_indices_from(mask)] = True
with sns.axes_style("white"):
    f, ax = plt.subplots(figsize=(9, 7))
    ax =
sns.heatmap(corr,mask=mask,square=True,annot=True,fmt='0.2f',linewidth
s=.8,cmap="hsv")
```




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

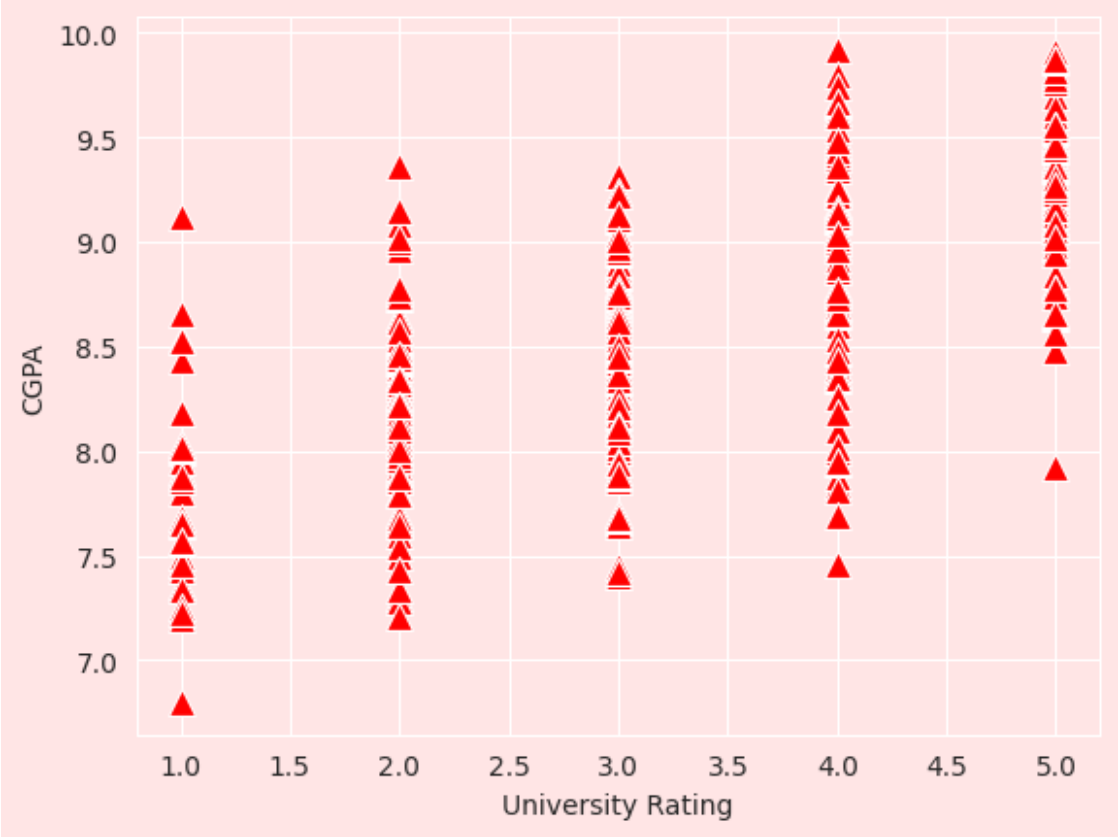



```
plt.rcParams['axes.facecolor'] = "#ffe5e5"
plt.rcParams['figure.facecolor'] = "#ffe5e5"
plt.figure(figsize=(6,6))
plt.subplot(2, 1, 1)
sns.distplot(df['GRE Score'],bins=34,color='Red',  kde_kws={"color":
"y", "lw": 3, "label": "KDE"},hist_kws={"linewidth": 2,"alpha": 0.3 })
plt.subplot(2, 1, 2)
sns.distplot(df['TOEFL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

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

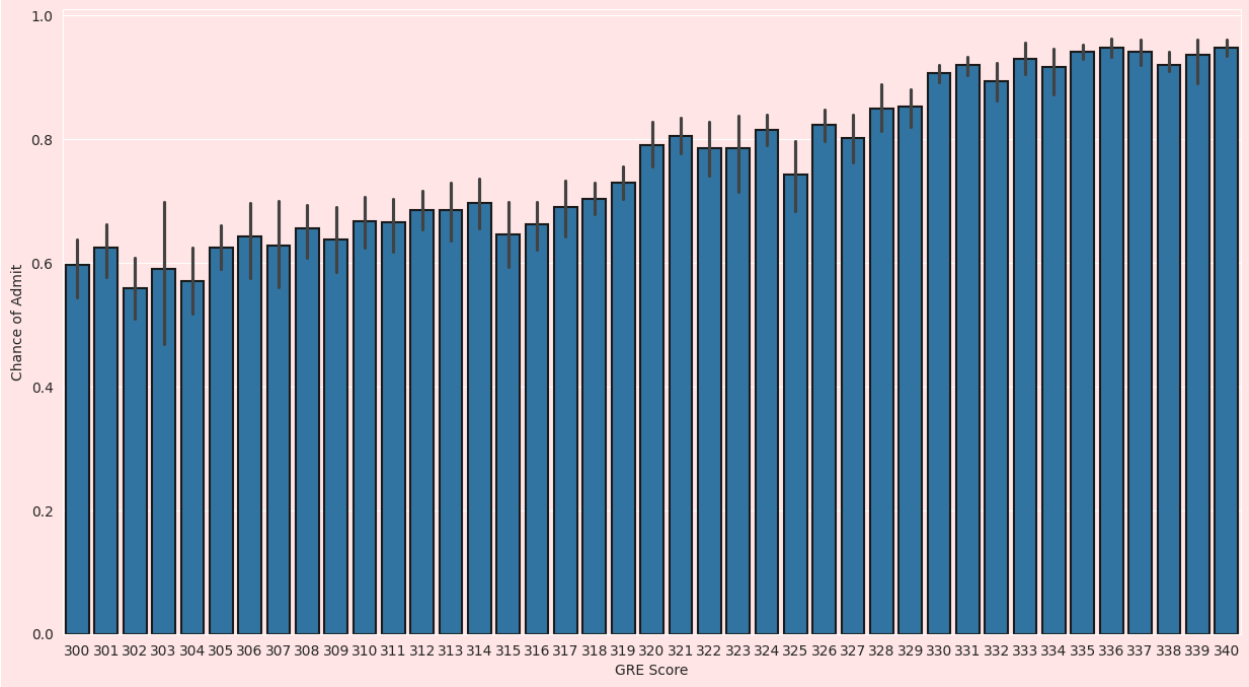



```
sns.scatterplot(x='University Rating',y='CGPA',data=df,color='Red',
marker="^", s=100)
```

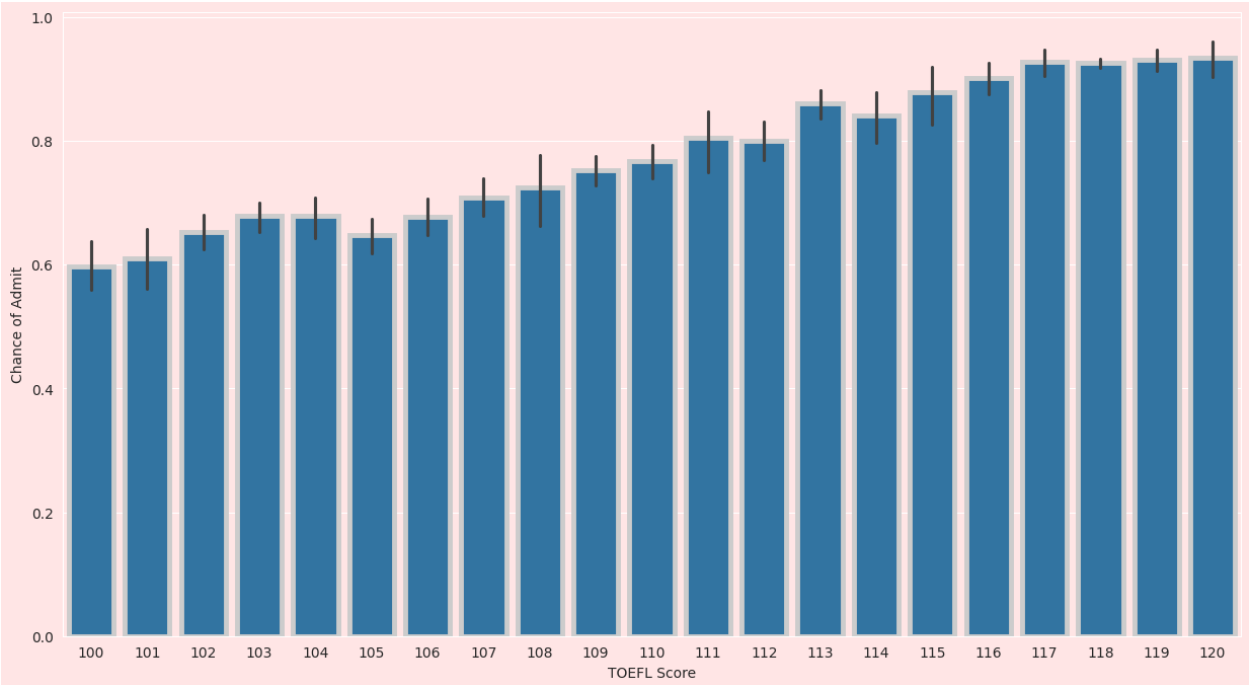
```
<Axes: xlabel='University Rating', ylabel='CGPA'>
```




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

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()
```




```
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()
```



```
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%
```

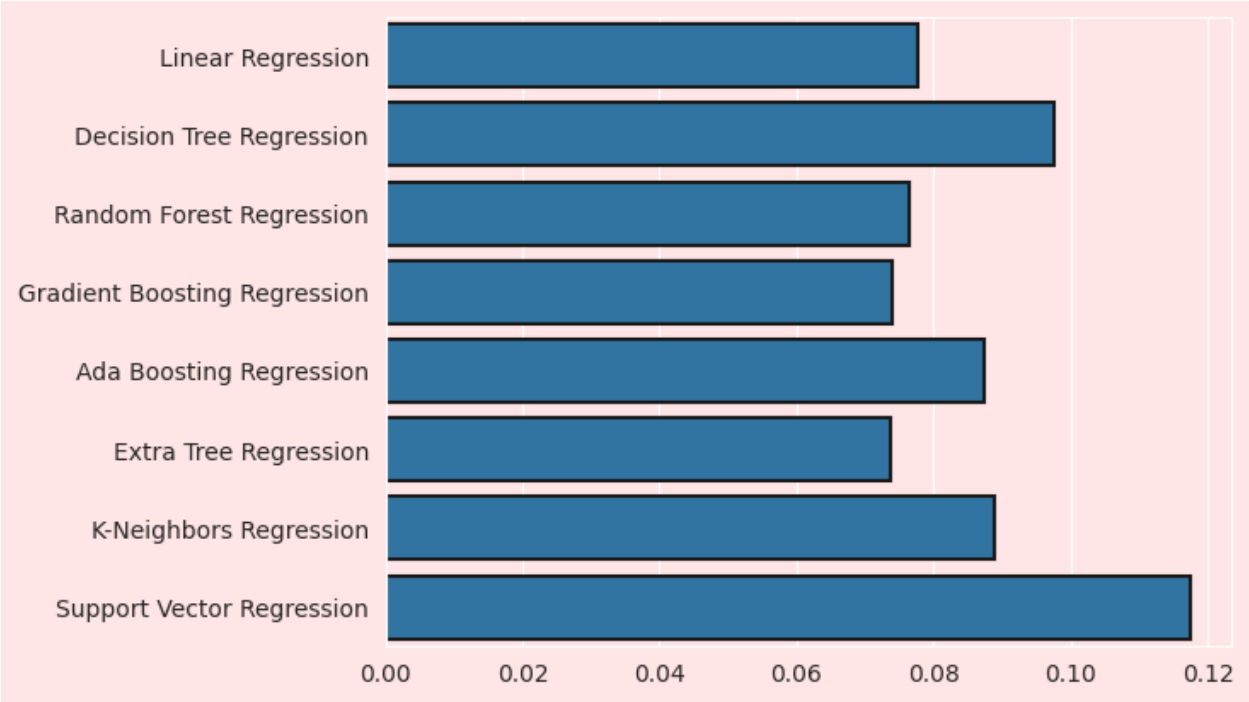
```
toppers=df[(df['GRE Score']>=330) & (df['TOEFL Score']>=115) &
(df['CGPA']>=9.5)].sort_values(by=['Chance of Admit'],ascending=False)
toppers
```


```
{"summary":"{\n  \"name\": \"toppers\",\n  \"rows\": 23,\n
\"fields\": [\n    {\n      \"column\": \"Serial No.\",\n
```

\"properties\": {\n   \"dtype\": \"number\",\n   \"std\":
158,\n   \"min\": 1,\n   \"max\": 498,\n
\"num_unique_values\": 23,\n   \"samples\": [\n   194,\n
24,\n   203\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"GRE
Score\",\n   \"properties\": {\n   \"dtype\": \"number\",\n
\"std\": 2,\n   \"min\": 330,\n   \"max\": 340,\n
\"num_unique_values\": 10,\n   \"samples\": [\n   332,\n
336,\n   339\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"TOEFL Score\",\n   \"properties\": {\n   \"dtype\":
\"number\",\n   \"std\": 1,\n   \"min\": 115,\n
\"max\": 120,\n   \"num_unique_values\": 6,\n   \"samples\":
[\n   120,\n   119,\n   115\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"University Rating\",\n
\"properties\": {\n   \"dtype\": \"number\",\n   \"std\":
0,\n   \"min\": 4,\n   \"max\": 5,\n
\"num_unique_values\": 2,\n   \"samples\": [\n   4,\n
5\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"SOP\",\n   \"properties\": {\n   \"dtype\": \"number\",\n
\"std\": 0.3679340893028889,\n   \"min\": 4.0,\n   \"max\":
5.0,\n   \"num_unique_values\": 3,\n   \"samples\": [\n
4.5,\n   4.0\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"LOR
\",\n   \"properties\": {\n   \"dtype\": \"number\",\n
\"std\": 0.543557306504609,\n   \"min\": 3.5,\n   \"max\":
5.0,\n   \"num_unique_values\": 4,\n   \"samples\": [\n
4.0,\n   5.0\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"CGPA\",\n   \"properties\": {\n   \"dtype\": \"number\",\n
\"std\": 0.13054613165513018,\n   \"min\": 9.5,\n   \"max\":
9.92,\n   \"num_unique_values\": 17,\n   \"samples\": [\n
9.91,\n   9.92\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"Research\",\n   \"properties\": {\n   \"dtype\":
\"number\",\n   \"std\": 0,\n   \"min\": 0,\n
\"max\": 1,\n   \"num_unique_values\": 2,\n   \"samples\":
[\n   0,\n   1\n   ],\n   \"semantic_type\":
\"\",\n   \"description\": \"\"\n   }\n   },\n   {\n
\"column\": \"Chance of Admit\",\n   \"properties\": {\n
\"dtype\": \"number\",\n   \"std\": 0.019172608920184114,\n
\"min\": 0.89,\n   \"max\": 0.97,\n   \"num_unique_values\":
7,\n   \"samples\": [\n   0.97,\n   0.96\
n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   }\n]\
n}","type":"dataframe","variable_name":"toppers"}

```
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 : 0.09757561170702442
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

y_ax=['Linear Regression' ,'Decision Tree Regression', 'Random Forest
Regression','Gradient Boosting Regression', 'Ada Boosting
Regression','Extra Tree Regression' ,'K-Neighbors Regression',
'Support Vector Regression' ]
x_ax=reg_pred

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

<Axes: >
```



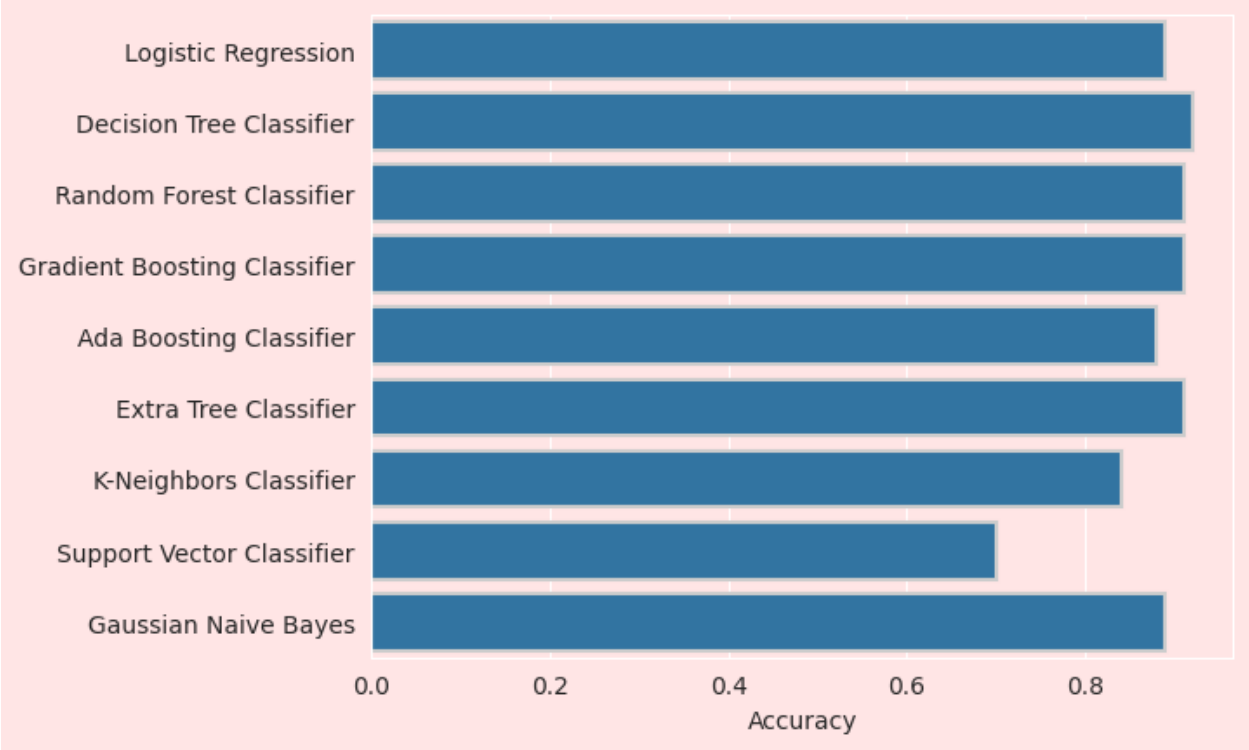

```
from sklearn.model_selection import train_test_split

X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.20,rand
om_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)
    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
```

```
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

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

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


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
0.0
0.2
0.4
0.6
0.8
Accuracy