```
from google.colab import files
uploaded = files.upload()
```

Choose Files weather.csv

- **weather.csv**(text/csv) - 142 bytes, last modified: 1/26/2025 - 100% done

```
Saving weather.csv to weather.csv
```

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

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

```
df
```

	sky	temp	humidity	wind	isPlay
0	sunny	warm	normal	strong	yes
1	sunny	warm	high	strong	yes
2	rainy	cold	high	strong	no
3	sunny	warm	high	weak	yes

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

```
df.describe()
```

	sky	temp	humidity	wind	isPlay
count	4	4	4	4	4
unique	2	2	2	2	2
top	sunny	warm	high	strong	yes
freq	3	3	3	3	3

```
X= df.drop(columns=['isPlay'])
X
```

	sky	temp	humidity	wind
0	sunny	warm	normal	strong
1	sunny	warm	high	strong
2	rainy	cold	high	strong
3	sunny	warm	high	weak

Next steps: Generate code with X | View recommended plots | New intera

```
y=df["isPlay"]
y
```

weather.csv

1 to 4 of 4 entries Filter

sky	temp	humidity	wind	isPlay
sunny	warm	normal	strong	yes
sunny	warm	high	strong	yes
rainy	cold	high	strong	no
sunny	warm	high	weak	yes

Show 10 per page

	isPlay
0	yes
1	yes
2	no
3	yes

```
Xn=np.array(X)
```

```
print(Xn)
```

```
[['sunny' 'warm' 'normal' 'strong']
 ['sunny' 'warm' 'high' 'strong']
 ['rainy' 'cold' 'high' 'strong']
 ['sunny' 'warm' 'high' 'weak']]
```

```
yn=np.array(y)
```

```
print(yn)
```

```
['yes' 'yes' 'no' 'yes']
```

```
def train(X,y):
  for i,val in enumerate(y):
    if val=='yes':
      h=X[i].copy()
      break
  for i,val in enumerate(X):
    if y[i]=='yes':
      for z in range(len(h)):
        if val[z] !=h[z]:
          h[z]='?'
        else:
          pass
  return h
```

```
print(train(Xn,yn))
```

```
['sunny' 'warm' '?' '?']
```

```
Start coding or generate with AI.
```

In [1]:
```
import pandas as pd
import numpy as np
import csv
```

In [2]:
```
data=pd.read_csv('Candidate_Algo_DS.csv')
```

In [3]:
```
data
```

Out[3]:

	Sunny	Warm	Normal	Strong	Warm.1	Same	Yes
0	Sunny	Warm	High	Strong	Warm	Same	Yes
1	Rainy	Cold	High	Strong	Warm	Change	No
2	Sunny	Warm	High	Strong	Cool	Change	Yes

In [4]:
```
# Open the csv file "Candidate_Algo_DS.csv"
with open("Candidate_Algo_DS.csv") as f:
 # Read the contents of the file using the csv reader
 csv_file = csv.reader(f)
 # Convert the contents to a list of lists
 data = list(csv_file)
```

In [5]:
```
data
```

Out[5]:
```
[['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same', 'Yes'],
 ['Sunny', 'Warm', 'High', 'Strong', 'Warm', 'Same', 'Yes'],
 ['Rainy', 'Cold', 'High', 'Strong', 'Warm', 'Change', 'No'],
 ['Sunny', 'Warm', 'High', 'Strong', 'Cool', 'Change', 'Yes']]
```

In [6]:
```
# Initialize the specific hypothesis with the first row of the data, excluding the las
specific = data[0][:-1]
# Initialize the general hypothesis with a list of "?" of the same length as the spec
general = [['?' for i in range(len(specific))] for j in range(len(specific))]
```

In [7]:
```
specific
```

Out[7]:
```
['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same']
```

In [8]:
```
general
```

Out[8]:
```
[['?', '?', '?', '?', '?', '?'],
 ['?', '?', '?', '?', '?', '?'],
 ['?', '?', '?', '?', '?', '?'],
 ['?', '?', '?', '?', '?', '?'],
 ['?', '?', '?', '?', '?', '?'],
 ['?', '?', '?', '?', '?', '?']]
```

In [14]:
```
# Iterate over each row in the data
for i in data:
     # If the last column of the current row is "Yes"
    if i[-1] == "Yes":
        # Iterate over each column in the current row
         for j in range(len(specific)):
             # If the current column value is not equal to the corresponding value in
             if i[j] != specific[j]:
                 # Update the corresponding value in the specific hypothesis to "?"
```

```
                specific[j] = "?"
                # Update the corresponding value in the general hypothesis to "?"
                general[j][j] = "?"


     # If the last column of the current row is "No"
    elif i[-1] == "No":
         # Iterate over each column in the current row
        for j in range(len(specific)):
         # If the current column value is not equal to the corresponding value in the
            if i[j] != specific[j]:
            # Update the corresponding value in the general hypothesis to the corresp
                general[j][j] = specific[j]
            else:
             # If the current column value is equal to the corresponding value in the
                general[j][j] = "?"


  # Print the current step of the algorithm and the values of the specific and general
    print("\nStep " + str(data.index(i)+1) + " of Candidate Elimination Algorithm")
    print(specific)
    print(general)
```

```
Step 1 of Candidate Elimination Algorithm
['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?',
'?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?',
'?', '?', '?', '?']]

Step 2 of Candidate Elimination Algorithm
['Sunny', 'Warm', '?', 'Strong', 'Warm', 'Same']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?',
'?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?',
'?', '?', '?', '?']]

Step 3 of Candidate Elimination Algorithm
['Sunny', 'Warm', '?', 'Strong', 'Warm', 'Same']
[['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?'], ['?', '?', '?',
'?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?',
'?', '?', '?', '?', 'Same']]

Step 4 of Candidate Elimination Algorithm
['Sunny', 'Warm', '?', 'Strong', '?', '?']
[['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?'], ['?', '?', '?',
'?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?',
'?', '?', '?', '?', '?']]
```

In [15]:

```
# Initialize the final general hypothesis list
gh = []
# Iterate over each list in the general hypothesis
for i in general:
    # Iterate over each value in the current list
     for j in i:
         # If the current value is not "?"
        if j != '?':
         # Add the current list to the final general hypothesis list
             gh.append(i)
             break

# Print the final specific and general hypotheses
print("\nFinal Specific hypothesis:\n", specific)
print("\nFinal General hypothesis:\n", gh)
```

```
Final Specific hypothesis:
 ['Sunny', 'Warm', '?', 'Strong', '?', '?']

Final General hypothesis:
```

In [ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [ ]:
```
from google.colab import files
uploaded = files.upload()
```

Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving car_evaluation.csv to car_evaluation.csv
```

In [ ]:
```
from google.colab import drive
drive.mount('/content/gdrive')
```

```
Mounted at /content/gdrive
```

In [ ]:
```
df=pd.read_csv('car_evaluation.csv', header=None)
```

In [ ]:
```
df.head()
```

Out[ ]:

	0	1	2	3	4	5	6
0	vhigh	vhigh	2	2	small	low	unacc
1	vhigh	vhigh	2	2	small	med	unacc
2	vhigh	vhigh	2	2	small	high	unacc
3	vhigh	vhigh	2	2	med	low	unacc
4	vhigh	vhigh	2	2	med	med	unacc

In [ ]:
```
df.shape
```

Out[ ]:
```
(1728, 7)
```

In [ ]:
```
col_names=['buying','maint','doors','persons','lug_boot','safety','class']

df.columns= col_names

col_names
```

Out[ ]:
```
['buying', 'maint', 'doors', 'persons', 'lug_boot', 'safety', 'class']
```

In [ ]:
```
df
```

Out[ ]:

	buying	maint	doors	persons	lug_boot	safety	class
0	vhigh	vhigh	2	2	small	low	unacc
1	vhigh	vhigh	2	2	small	med	unacc
2	vhigh	vhigh	2	2	small	high	unacc

3	vhigh	vhigh	2	2	med	low	unacc
4	vhigh	vhigh	2	2	med	med	unacc
...	...	...	...	...	...	...	...
1723	low	low	5more	more	med	med	good
1724	low	low	5more	more	med	high	vgood
1725	low	low	5more	more	big	low	unacc
1726	low	low	5more	more	big	med	good
1727	low	low	5more	more	big	high	vgood

1728 rows × 7 columns

In [ ]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1728 entries, 0 to 1727
Data columns (total 7 columns):
 #   Column    Non-Null Count  Dtype 
---  ------    --------------  ----- 
 0   buying    1728 non-null   object
 1   maint     1728 non-null   object
 2   doors     1728 non-null   object
 3   persons   1728 non-null   object
 4   lug_boot  1728 non-null   object
 5   safety    1728 non-null   object
 6   class     1728 non-null   object
dtypes: object(7)
memory usage: 94.6+ KB
```

In [ ]:
```
df.describe()
```

Out[ ]:

	buying	maint	doors	persons	lug_boot	safety	class
count	1728	1728	1728	1728	1728	1728	1728
unique	4	4	4	3	3	3	4
top	vhigh	vhigh	2	2	small	low	unacc
freq	432	432	432	576	576	576	1210

In [ ]:
```
col_names=['buying','maint','doors','persons','lug_boot','safety','class']

for col in col_names:
    print(df[col].value_counts())
```

```
buying
vhigh    432
high     432
med      432
low      432
Name: count, dtype: int64
maint
vhigh    432
high     432
```

```
med      432
low      432
Name: count, dtype: int64
doors
2        432
3        432
4        432
5more    432
Name: count, dtype: int64
persons
2       576
4       576
more    576
Name: count, dtype: int64
lug_boot
small    576
med      576
big      576
Name: count, dtype: int64
safety
low     576
med     576
high    576
Name: count, dtype: int64
class
unacc    1210
acc       384
good       69
vgood      65
Name: count, dtype: int64
```

In [ ]:
```
df.isnull().sum()
```

Out[ ]:

	0
buying	0
maint	0
doors	0
persons	0
lug_boot	0
safety	0
class	0

dtype: int64

In [ ]:
```
X= df.drop(['class'],axis=1)
y=df['class']
```

Sk Learn library to Divide data

In [ ]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test =train_test_split(X,y,test_size=0.33,rand
```

33% data in testing

In [ ]:
```
X_train.shape, X_test.shape
```

Out[ ]:
```
((1157, 6), (571, 6))
```

TO check data is normalize or not

In [ ]:
```
X_train.dtypes
```

Out[ ]:

	0
buying	object
maint	object
doors	object
persons	object
lug_boot	object
safety	object

dtype: object

To install Category_Encoders using pip install category_encoders

In [ ]:
```
pip install category_encoders
```

```
Collecting category_encoders
  Downloading category_encoders-2.8.0-py3-none-any.whl.metadata (7.9 kB)
Requirement already satisfied: numpy>=1.14.0 in /usr/local/lib/python3.11/di
st-packages (from category_encoders) (1.26.4)
Requirement already satisfied: pandas>=1.0.5 in /usr/local/lib/python3.11/di
st-packages (from category_encoders) (2.2.2)
Requirement already satisfied: patsy>=0.5.1 in /usr/local/lib/python3.11/dis
t-packages (from category_encoders) (1.0.1)
Requirement already satisfied: scikit-learn>=1.6.0 in /usr/local/lib/python
3.11/dist-packages (from category_encoders) (1.6.1)
Requirement already satisfied: scipy>=1.0.0 in /usr/local/lib/python3.11/dis
t-packages (from category_encoders) (1.13.1)
Requirement already satisfied: statsmodels>=0.9.0 in /usr/local/lib/python3.
11/dist-packages (from category_encoders) (0.14.4)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/pyth
on3.11/dist-packages (from pandas>=1.0.5->category_encoders) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dis
t-packages (from pandas>=1.0.5->category_encoders) (2024.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/d
ist-packages (from pandas>=1.0.5->category_encoders) (2025.1)
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.11/di
st-packages (from scikit-learn>=1.6.0->category_encoders) (1.4.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python
3.11/dist-packages (from scikit-learn>=1.6.0->category_encoders) (3.5.0)
Requirement already satisfied: packaging>=21.3 in /usr/local/lib/python3.11/
dist-packages (from statsmodels>=0.9.0->category_encoders) (24.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-pa
ckages (from python-dateutil>=2.8.2->pandas>=1.0.5->category_encoders) (1.1
7.0)
```

```
Downloading category_encoders-2.8.0-py3-none-any.whl (85 kB)
     ---------------------------------------- 0.0/85.7 kB ? eta -:--:--
     ---------------------------------------- 85.7/85.7 kB 3.7 MB/s eta 0:00:0
0
Installing collected packages: category_encoders
Successfully installed category_encoders-2.8.0
```

In [ ]:
```
import category_encoders as ce
```

In [ ]:
```
encoder=ce.OrdinalEncoder(cols=['buying','maint','doors','persons','lug_bo


X_train = encoder.fit_transform (X_train)

X_test = encoder.transform(X_test)
```

In [ ]:
```
X_train.head()
```

Out[ ]:

	buying	maint	doors	persons	lug_boot	safety
48	1	1	1	1	1	1
468	2	1	1	2	2	1
155	1	2	1	1	2	2
1721	3	3	2	1	2	2
1208	4	3	3	1	2	2

In [ ]:
```
X_test.head()
```

Out[ ]:

	buying	maint	doors	persons	lug_boot	safety
599	2	2	4	3	1	2
1201	4	3	3	2	1	3
628	2	2	2	3	3	3
1498	3	2	2	2	1	3
1263	4	3	4	1	1	1

In [ ]:
```
from sklearn.tree import DecisionTreeClassifier
```

In [ ]:
```
clf_gini =DecisionTreeClassifier(criterion='gini',max_depth=3,random_state


clf_gini.fit(X_train,y_train)
```

Out[ ]:
```
DecisionTreeClassifier(max_depth=3, random_state=0)
```
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading

this page with nbviewer.org.

In [ ]:
```
y_pred_gini=clf_gini.predict(X_test)
```

In [ ]:
```
from sklearn.metrics import accuracy_score

print('Model accuracy score with criterion gini index: {0:0.4f}',format(ac
```

```
Model accuracy score with criterion gini index: {0:0.4f} 0.8021015761821366
```

In [ ]:
```
y_pred_train_gini=clf_gini.predict(X_train)


y_pred_train_gini
```

Out[ ]:
```
array(['unacc', 'unacc', 'unacc', ..., 'unacc', 'unacc', 'acc'],
      dtype=object)
```

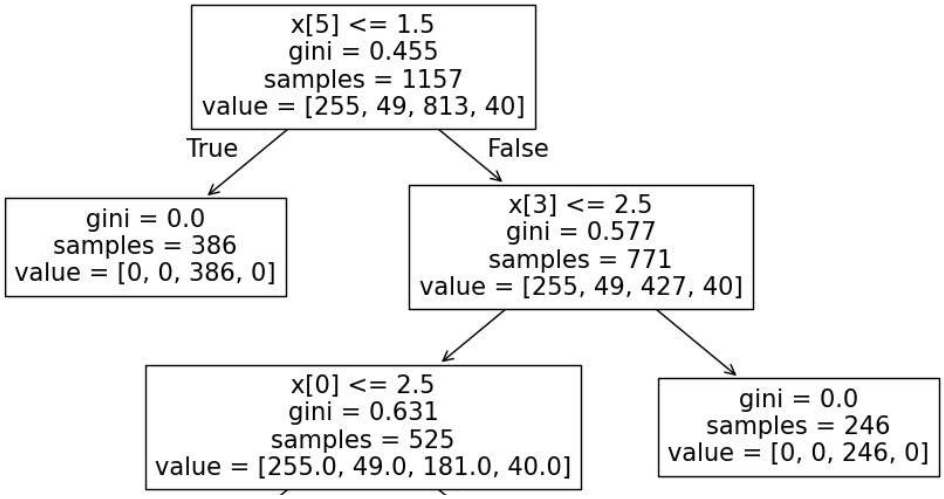
In [ ]:
```
print('Training set accuracy score : {0:0.4f}',format(accuracy_score(y_tra
```

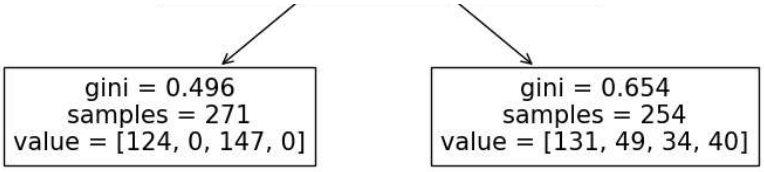
```
Training set accuracy score : {0:0.4f} 0.7865168539325843
```

In [ ]:
```
plt.figure(figsize=(12,8))

from sklearn import tree
tree.plot_tree(clf_gini.fit(X_train,y_train))
```

Out[ ]:
```
[Text(0.4, 0.875, 'x[5] <= 1.5\ngini = 0.455\nsamples = 1157\nvalue = [25
5, 49, 813, 40]'),
 Text(0.2, 0.625, 'gini = 0.0\nsamples = 386\nvalue = [0, 0, 386, 0]'),
 Text(0.30000000000000004, 0.75, 'True  '),
 Text(0.6, 0.625, 'x[3] <= 2.5\ngini = 0.577\nsamples = 771\nvalue = [255,
49, 427, 40]'),
 Text(0.5, 0.75, '  False'),
 Text(0.4, 0.375, 'x[0] <= 2.5\ngini = 0.631\nsamples = 525\nvalue = [255.
0, 49.0, 181.0, 40.0]'),
 Text(0.2, 0.125, 'gini = 0.496\nsamples = 271\nvalue = [124, 0, 147,
0]'),
 Text(0.6, 0.125, 'gini = 0.654\nsamples = 254\nvalue = [131, 49, 34, 4
0]'),
 Text(0.8, 0.375, 'gini = 0.0\nsamples = 246\nvalue = [0, 0, 246, 0]')]
```





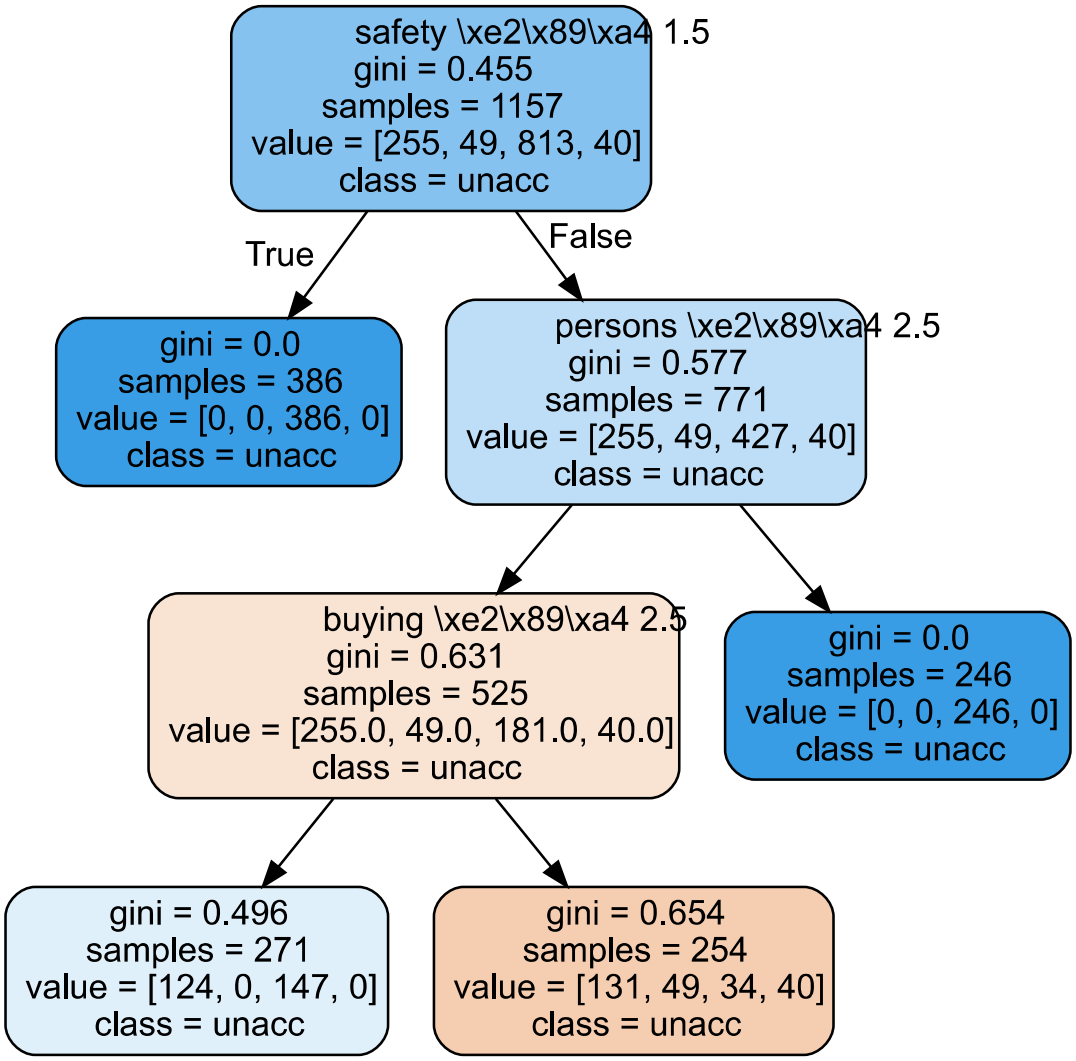
main ML / Experiment / Experiment 3 / Decision_Tree.ipynb ↑ Top

Preview Code Blame Raw

In [ ]:

```
import graphviz
dot_data = tree.export_graphviz(clf_gini, out_file=None, feature_names=X_t
graph = graphviz.Source(dot_data)
graph
```

Out[ ]:



In [ ]:

```
clf_en = DecisionTreeClassifier(criterion='entropy', max_depth=3, random_s


# fit the model
clf_en.fit(X_train, y_train)
```

Out[ ]:

```
DecisionTreeClassifier(criterion='entropy', max_depth=3, random_sta
```

te=0)

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [ ]:
```
y_pred_en = clf_en.predict(X_test)
```

In [ ]:
```
from sklearn.metrics import accuracy_score

print('Model accuracy score with criterion entropy: {0:0.4f}'. format(accu
```

```
Model accuracy score with criterion entropy: 0.8021
```

In [ ]:
```
y_pred_train_en = clf_en.predict(X_train)

y_pred_train_en
```

Out[ ]:
```
array(['unacc', 'unacc', 'unacc', ..., 'unacc', 'unacc', 'acc'],
      dtype=object)
```

In [ ]:
```
print('Training-set accuracy score: {0:0.4f}'. format(accuracy_score(y_tra
```

```
Training-set accuracy score: 0.7865
```

In [ ]:
```
# print the scores on training and test set

print('Training set score: {:.4f}'.format(clf_en.score(X_train, y_train)))

print('Test set score: {:.4f}'.format(clf_en.score(X_test, y_test)))
```

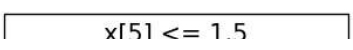
```
Training set score: 0.7865
Test set score: 0.8021
```

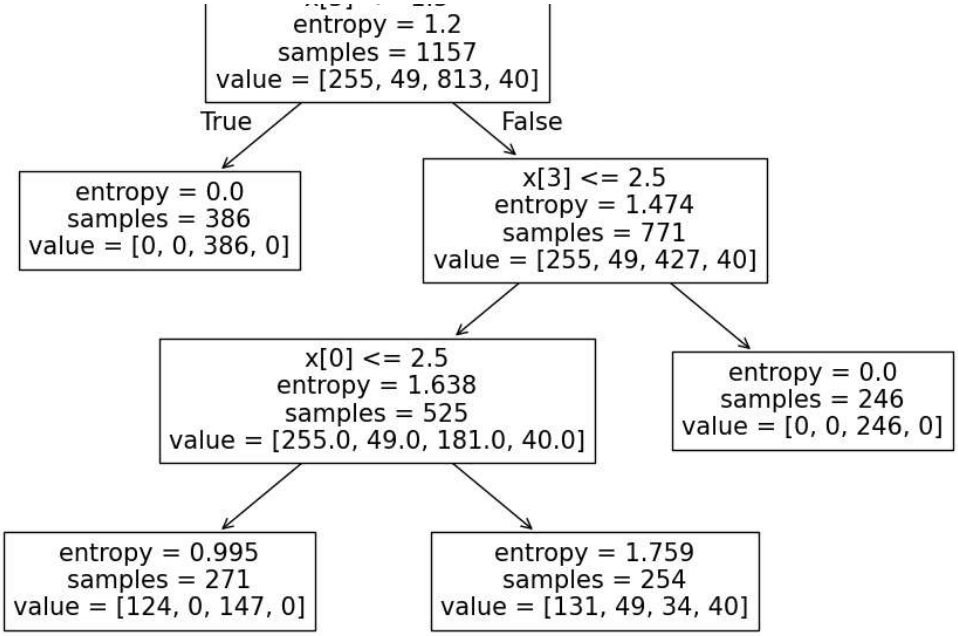
In [ ]:
```
plt.figure(figsize=(12,8))

from sklearn import tree

tree.plot_tree(clf_en.fit(X_train, y_train))
```

Out[ ]:
```
[Text(0.4, 0.875, 'x[5] <= 1.5\nentropy = 1.2\nsamples = 1157\nvalue = [25
5, 49, 813, 40]'),
 Text(0.2, 0.625, 'entropy = 0.0\nsamples = 386\nvalue = [0, 0, 386, 0]'),
 Text(0.30000000000000004, 0.75, 'True  '),
 Text(0.6, 0.625, 'x[3] <= 2.5\nentropy = 1.474\nsamples = 771\nvalue = [2
55, 49, 427, 40]'),
 Text(0.5, 0.75, '  False'),
 Text(0.4, 0.375, 'x[0] <= 2.5\nentropy = 1.638\nsamples = 525\nvalue = [2
55.0, 49.0, 181.0, 40.0]'),
 Text(0.2, 0.125, 'entropy = 0.995\nsamples = 271\nvalue = [124, 0, 147,
0]'),
 Text(0.6, 0.125, 'entropy = 1.759\nsamples = 254\nvalue = [131, 49, 34, 4
0]'),
 Text(0.8, 0.375, 'entropy = 0.0\nsamples = 246\nvalue = [0, 0, 246, 0]')]
```





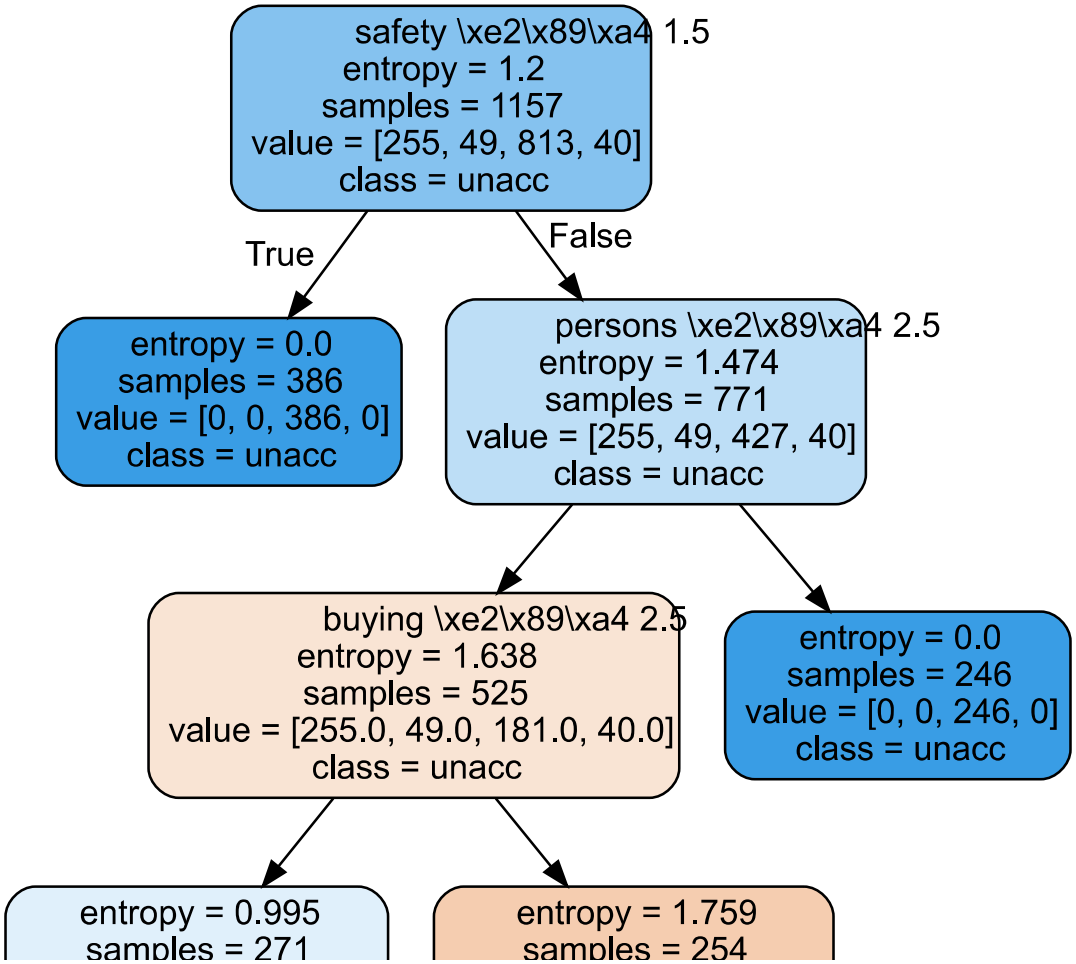
In [ ]:

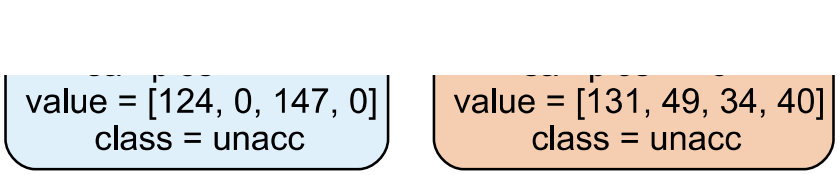
```
import graphviz
dot_data = tree.export_graphviz(clf_en, out_file=None,
                                feature_names=X_train.columns,
                                class_names=y_train,
                                filled=True, rounded=True,
                                special_characters=True)

graph = graphviz.Source(dot_data)

graph
```

Out[ ]:





In [ ]:

```
# Print the Confusion Matrix and slice it into four pieces

from sklearn.metrics import confusion_matrix

cm = confusion_matrix(y_test, y_pred_en)

print('Confusion matrix\n\n', cm)
```

```
Confusion matrix

 [[ 73   0  56   0]
 [ 20   0   0   0]
 [ 12   0 385   0]
 [ 25   0   0   0]]
```

EXPERIMENT 4

Build an Artificial Neural Network by implementing the Backpropagation algorithm and test the same using appropriate data sets.

```
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 os
dataset_path_X='/content/drive/My Drive/3RD YEAR/ML_Database/X.npy'
dataset_path_y='/content/drive/My Drive/3RD YEAR/ML_Database/y.npy'
```

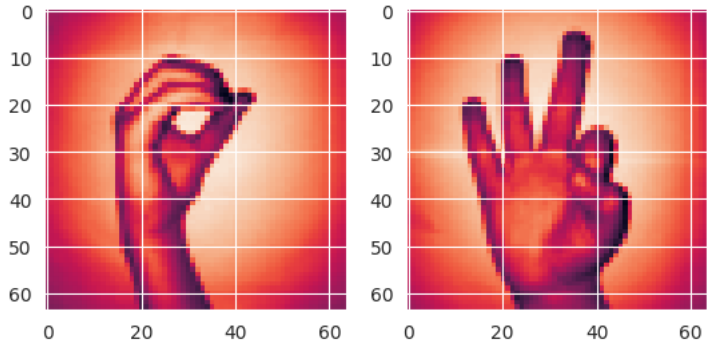
```
dataset_path_X
```

```
'/content/drive/My Drive/3RD YEAR/ML_Database/X.npy'
```

```
from re import X
X=np.load(dataset_path_X)
```

```
image_size=64
plt.subplot(1,2,1)
plt.imshow(X[399].reshape(image_size,image_size))
plt.subplot(1,2,2)
plt.imshow(X[189].reshape(image_size,image_size))
```

```
<matplotlib.image.AxesImage at 0x7b49798435d0>
```




```
X = np.concatenate((X[204:409], X[822:1027] ), axis=0) # from 0 to 204 is zero sign and from 205 to 410 is one s
z = np.zeros(205)
o = np.ones(205)
Y = np.concatenate((z, o), axis=0).reshape(X.shape[0],1)
print("X shape: " , X.shape)
print("Y shape: " , Y.shape)
```

```
X shape:  (410, 64, 64)
Y shape:  (410, 1)
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.15, random_state=42)
number_of_train = X_train.shape[0]
number_of_test = X_test.shape[0]
```

```
print("Number of training examples are:-",number_of_train)
print("Number of test examples are:-",number_of_test)
```

```
Number of training examples are:- 348
Number of test examples are:- 62
```



```
# prompt: I have to use ann algo by using tensorflow keras model train it and Build an Artificial Neural Network
# implementing the Backpropagation
# algorithm and test the same using
# appropriate data sets.

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

# Define the model
model = Sequential()
model.add(Dense(128, activation='relu', input_shape=(image_size*image_size,)))  # Input layer
model.add(Dense(64, activation='relu'))  # Hidden layer
model.add(Dense(1, activation='sigmoid'))  # Output layer (sigmoid for binary classification)

# Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Reshape the input data
X_train = X_train.reshape(number_of_train, image_size * image_size)
X_test = X_test.reshape(number_of_test, image_size*image_size)

# Train the model
model.fit(X_train, Y_train, epochs=10, batch_size=32, validation_data=(X_test, Y_test))

# Evaluate the model
loss, accuracy = model.evaluate(X_test, Y_test)
print(f'Test Loss: {loss:.4f}')
print(f'Test Accuracy: {accuracy:.4f}')

# Make predictions (optional)
predictions = model.predict(X_test)
```

```
Epoch 1/10
11/11 ━━━━━━━━━━━━━━━━━━━━ 2s 44ms/step - accuracy: 0.4806 - loss: 1.1873 - val_accuracy: 0.5323 - val_loss: 0.7181
Epoch 2/10
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step - accuracy: 0.5053 - loss: 0.7164 - val_accuracy: 0.5323 - val_loss: 0.6503
Epoch 3/10
11/11 ━━━━━━━━━━━━━━━━━━━━ 1s 24ms/step - accuracy: 0.6508 - loss: 0.6513 - val_accuracy: 0.8548 - val_loss: 0.6082
Epoch 4/10
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step - accuracy: 0.8412 - loss: 0.6020 - val_accuracy: 0.7742 - val_loss: 0.5575
Epoch 5/10
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - accuracy: 0.7957 - loss: 0.5442 - val_accuracy: 0.5645 - val_loss: 0.5814
Epoch 6/10
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step - accuracy: 0.7806 - loss: 0.5126 - val_accuracy: 0.8387 - val_loss: 0.4356
Epoch 7/10
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step - accuracy: 0.8667 - loss: 0.4187 - val_accuracy: 0.9516 - val_loss: 0.3507
Epoch 8/10
11/11 ━━━━━━━━━━━━━━━━━━━━ 1s 25ms/step - accuracy: 0.9074 - loss: 0.3736 - val_accuracy: 0.9516 - val_loss: 0.2947
Epoch 9/10
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - accuracy: 0.9208 - loss: 0.3044 - val_accuracy: 0.9355 - val_loss: 0.2547
Epoch 10/10
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.9235 - loss: 0.2704 - val_accuracy: 0.9355 - val_loss: 0.2473
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step - accuracy: 0.9362 - loss: 0.2365
Test Loss: 0.2473
Test Accuracy: 0.9355
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 51ms/step
```

In [ ]:
```
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 [ ]:
```
from google.colab import drive
drive.mount('/content/drive')
```

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

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

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

In [ ]:
```
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'])
```

In [ ]:
```
df.loc[outliers_to_drop]
```

Out[ ]:

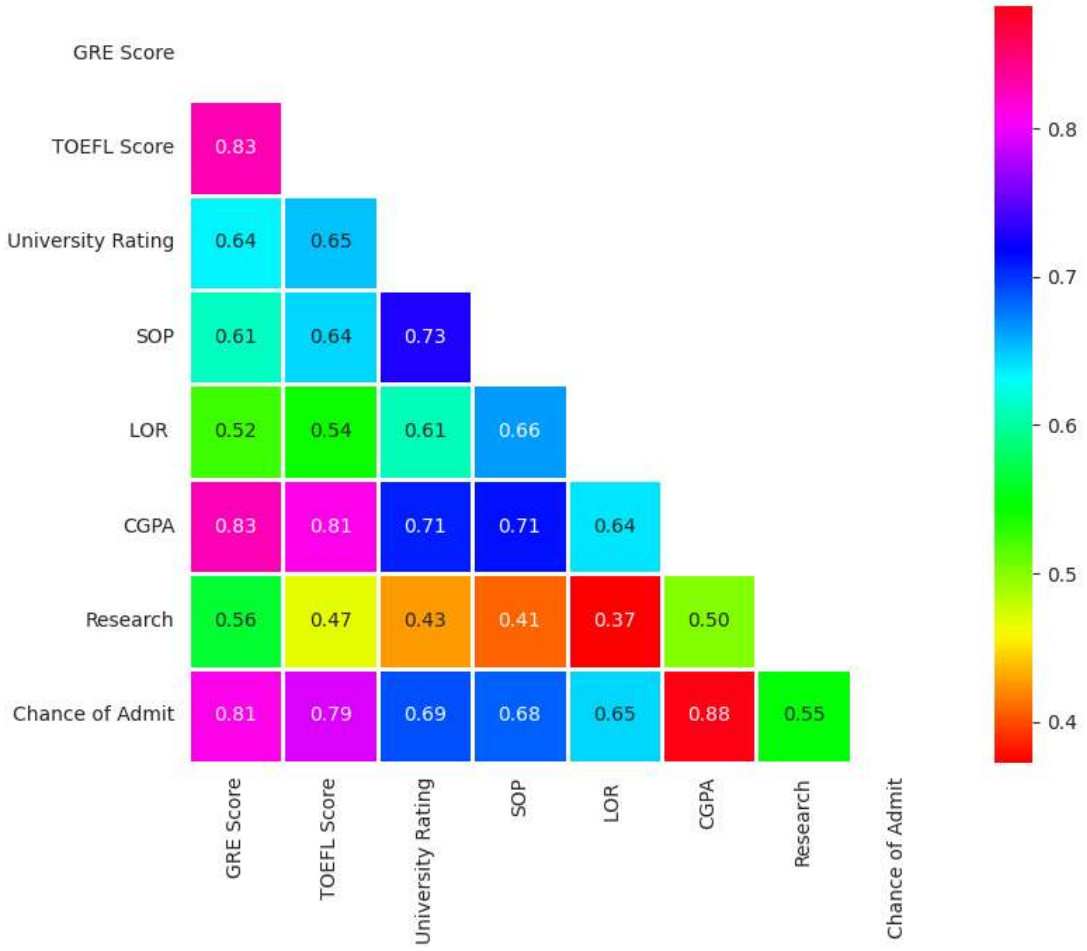
Serial No.	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit

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

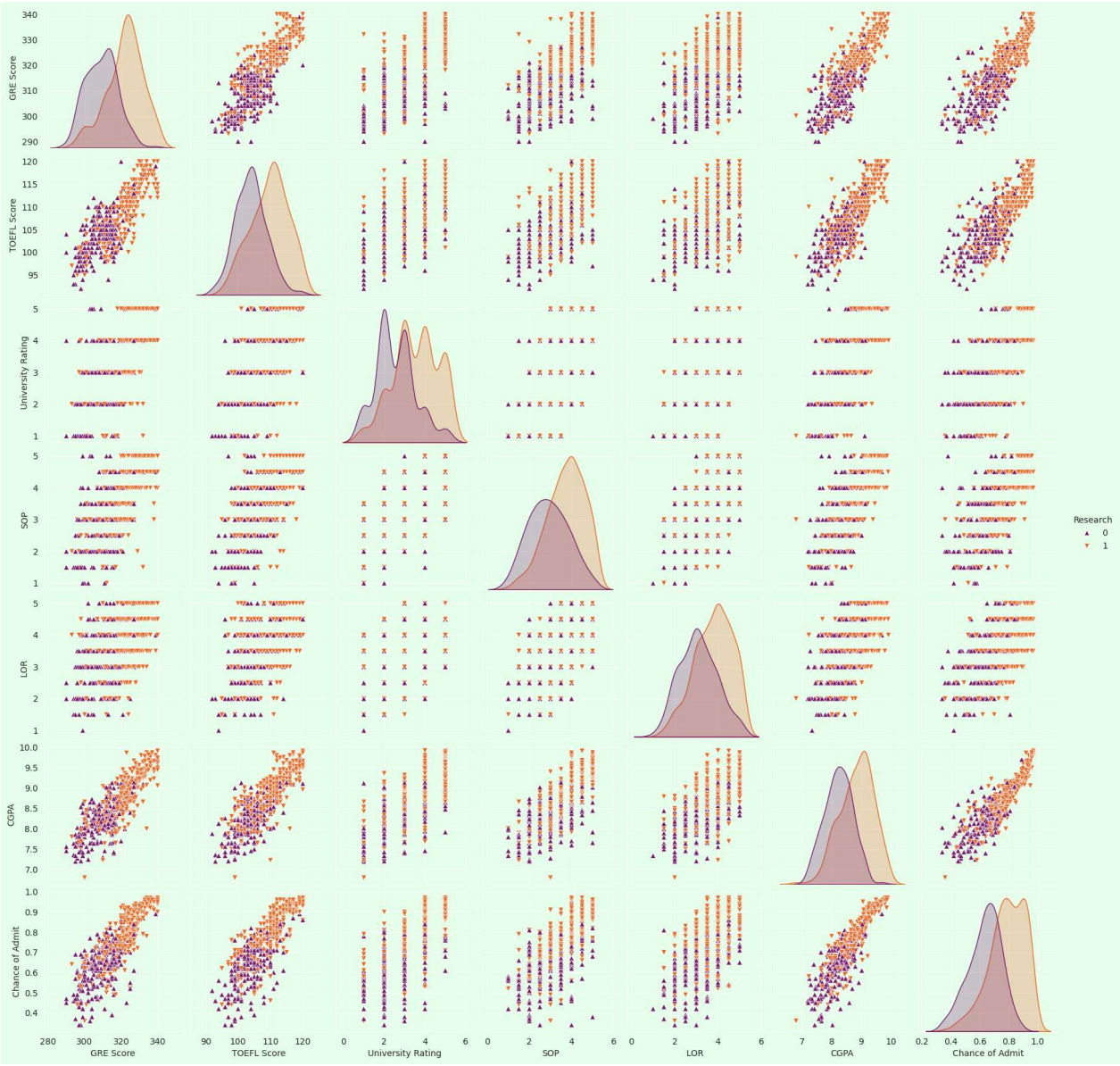
Out[ ]:

	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')
```



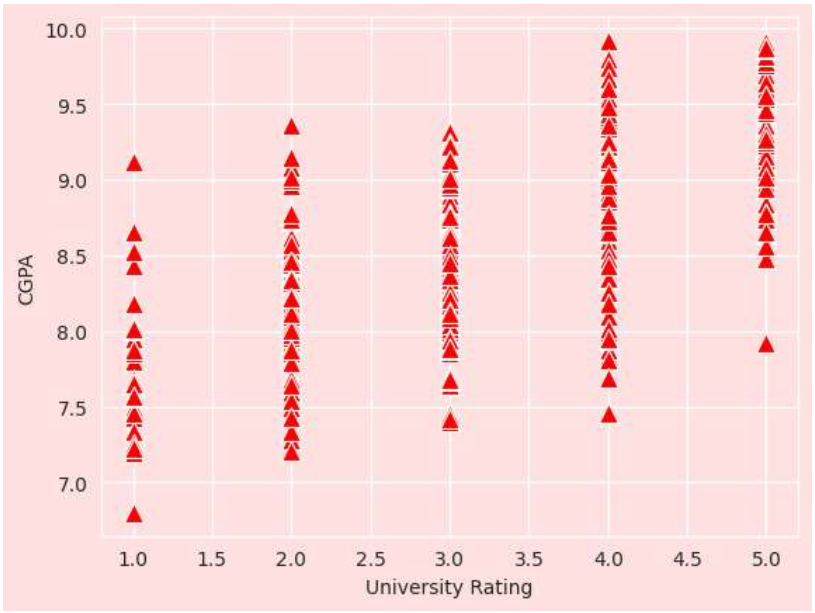
In [ ]:

```
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'>

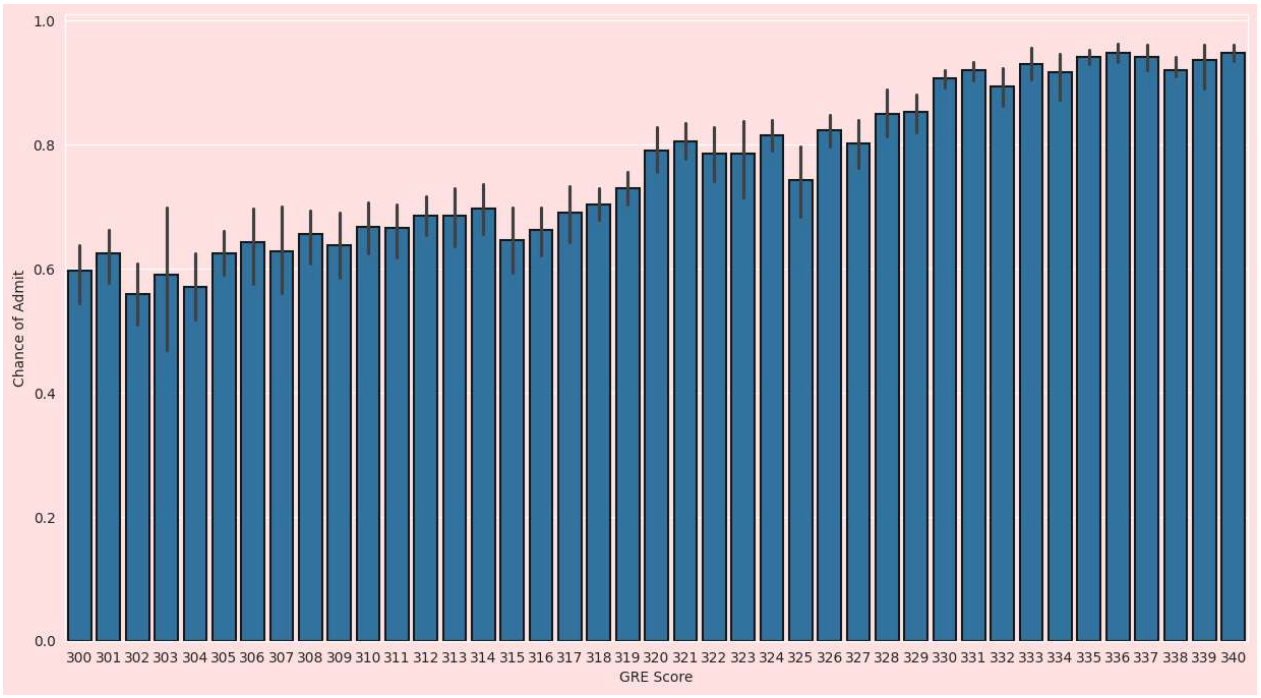
In [ ]:
```
sns.scatterplot(x='University Rating',y='CGPA',data=df,color='Red', marker="^", s=100)
```

Out[ ]: <Axes: xlabel='University Rating', ylabel='CGPA'>



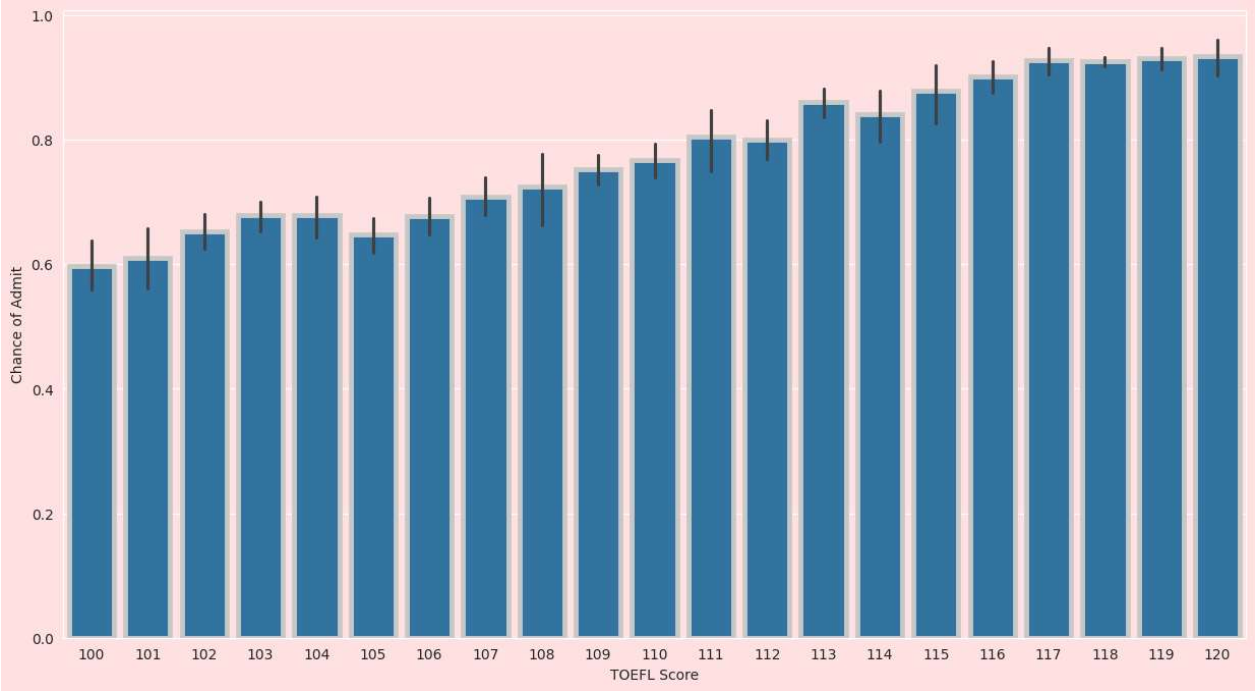
In [ ]:
```
co_gre=df[df["GRE Score"]>=300]
co_toefel=df[df["TOEFL Score"]>=100]
```

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()
```



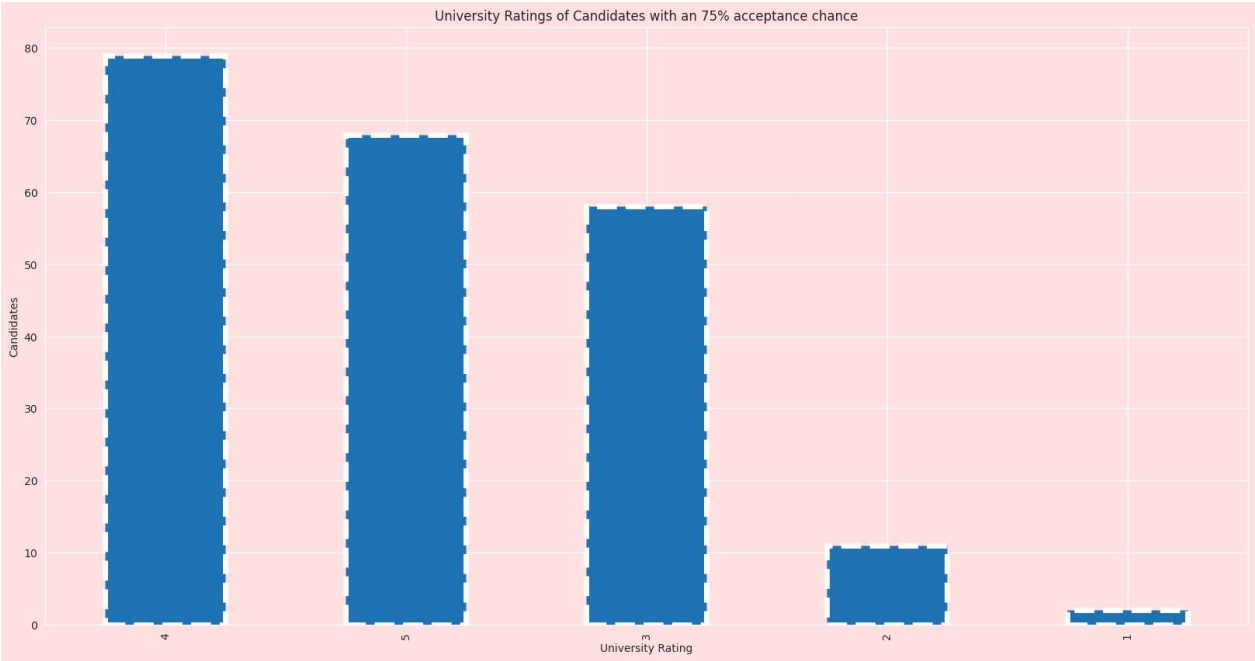
In [ ]:

```
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'],ascer
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)
```

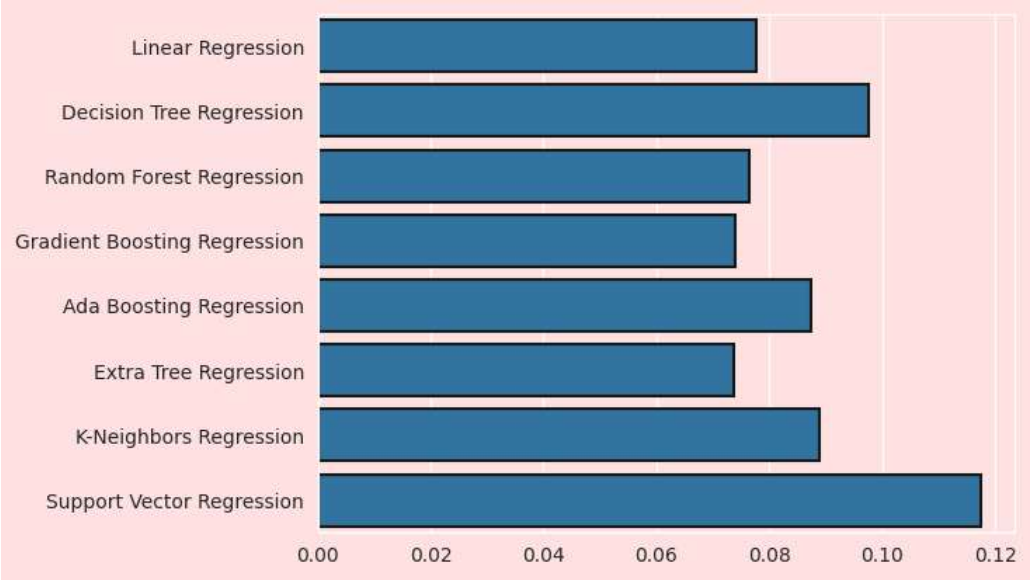
```
Results...

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 Boo
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=[]
```

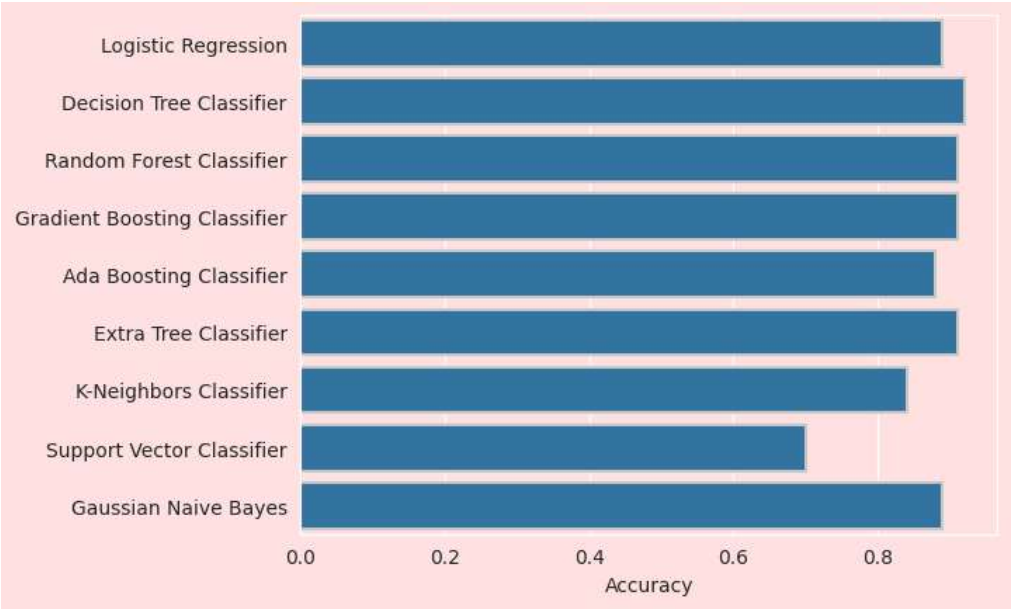
```
cla_pred=[]
for name,model in classifiers:
    model=model
    model.fit(X_train,y_train_c)
    predictions = model.predict(X_test)
    cla_pred.append(accuracy_score(y_test_c,predictions))
    print(name,accuracy_score(y_test_c,predictions))
```

```
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 [ ]:
```
```