Barak Bonker 316177708
Amit Avigdor 316178144

Exploratory Data Analysis - Mushrooms

Importing the required libraries for EDA

In [1]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn import preprocessing
import numpy as np

from sklearn.metrics import accuracy_score, recall_score, precision_score, f1_score, r
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.cluster import KMeans
%matplotlib inline
sns.set(color_codes=True)
```

Loading the data into the data frame

In [2]:
```
# from google.colab import files
# uploaded = files.upload()
# import io
# df = pd.read_csv(io.BytesIO(uploaded['mushrooms.csv']))
df = pd.read_csv("mushrooms.csv")
```

In [3]:
```
df.head(5)
```

Out[3]:

	class	cap-shape	cap-surface	cap-color	bruises	odor	gill-attachment	gill-spacing	gill-size	gill-color	...	stalk-surface-below-ring	stalk-color-above-ring
0	p	x	s	n	t	p	f	c	n	k	...	s	w
1	e	x	s	y	t	a	f	c	b	k	...	s	w
2	e	b	s	w	t	l	f	c	b	n	...	s	w
3	p	x	y	w	t	p	f	c	n	n	...	s	w
4	e	x	s	g	f	n	f	w	b	k	...	s	w

5 rows × 23 columns

Checking the types of data

In [4]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8124 entries, 0 to 8123
Data columns (total 23 columns):
 #   Column                    Non-Null Count  Dtype 
---  ------                    --------------  ----- 
 0   class                     8124 non-null   object
 1   cap-shape                 8124 non-null   object
 2   cap-surface               8124 non-null   object
 3   cap-color                 8124 non-null   object
 4   bruises                   8124 non-null   object
 5   odor                      8124 non-null   object
 6   gill-attachment           8124 non-null   object
 7   gill-spacing              8124 non-null   object
 8   gill-size                 8124 non-null   object
 9   gill-color                8124 non-null   object
 10  stalk-shape               8124 non-null   object
 11  stalk-root                8124 non-null   object
 12  stalk-surface-above-ring  8124 non-null   object
 13  stalk-surface-below-ring  8124 non-null   object
 14  stalk-color-above-ring    8124 non-null   object
 15  stalk-color-below-ring    8124 non-null   object
 16  veil-type                 8124 non-null   object
 17  veil-color                8124 non-null   object
 18  ring-number               8124 non-null   object
 19  ring-type                 8124 non-null   object
 20  spore-print-color         8124 non-null   object
 21  population                8124 non-null   object
 22  habitat                   8124 non-null   object
dtypes: object(23)
memory usage: 1.4+ MB
```

Dropping the duplicate rows

In [5]:
```
df.shape
```

Out[5]:
```
(8124, 23)
```

In [6]:
```
duplicate_rows_df = df[df.duplicated()]
print("number of duplicate rows: ", duplicate_rows_df.shape)
```

```
number of duplicate rows:  (0, 23)
```

In [7]:
```
df.count()
```

Out[7]:

```
class                       8124
cap-shape                   8124
cap-surface                 8124
cap-color                   8124
bruises                     8124
odor                        8124
gill-attachment             8124
gill-spacing                8124
gill-size                   8124
gill-color                  8124
stalk-shape                 8124
stalk-root                  8124
stalk-surface-above-ring    8124
stalk-surface-below-ring    8124
stalk-color-above-ring      8124
stalk-color-below-ring      8124
veil-type                   8124
veil-color                  8124
ring-number                 8124
ring-type                   8124
spore-print-color           8124
population                  8124
habitat                     8124
dtype: int64
```

In [8]:

```
df = df.drop_duplicates()
df.head(5)
```

Out[8]:

	class	cap-shape	cap-surface	cap-color	bruises	odor	gill-attachment	gill-spacing	gill-size	gill-color	...	stalk-surface-below-ring	stalk-color-above-ring
0	p	x	s	n	t	p	f	c	n	k	...	s	w
1	e	x	s	y	t	a	f	c	b	k	...	s	w
2	e	b	s	w	t	l	f	c	b	n	...	s	w
3	p	x	y	w	t	p	f	c	n	n	...	s	w
4	e	x	s	g	f	n	f	w	b	k	...	s	w

5 rows × 23 columns

In [9]:

```
df.count()
```

Out[9]:
```
class                       8124
cap-shape                   8124
cap-surface                 8124
cap-color                   8124
bruises                     8124
odor                        8124
gill-attachment             8124
gill-spacing                8124
gill-size                   8124
gill-color                  8124
stalk-shape                 8124
stalk-root                  8124
stalk-surface-above-ring    8124
stalk-surface-below-ring    8124
stalk-color-above-ring      8124
stalk-color-below-ring      8124
veil-type                   8124
veil-color                  8124
ring-number                 8124
ring-type                   8124
spore-print-color           8124
population                  8124
habitat                     8124
dtype: int64
```

Dropping the missing or null values

In [10]:
```
print(df.isnull().sum())
```

```
class                       0
cap-shape                   0
cap-surface                 0
cap-color                   0
bruises                     0
odor                        0
gill-attachment             0
gill-spacing                0
gill-size                   0
gill-color                  0
stalk-shape                 0
stalk-root                  0
stalk-surface-above-ring    0
stalk-surface-below-ring    0
stalk-color-above-ring      0
stalk-color-below-ring      0
veil-type                   0
veil-color                  0
ring-number                 0
ring-type                   0
spore-print-color           0
population                  0
habitat                     0
dtype: int64
```

In [11]:
```
for column in df.columns:
    df[column] = df[column].replace('?', np.nan)
    if df[column].dtypes == "object":
        if df[column].str.contains('.').any():
            df[column] = pd.to_numeric(df[column], errors='ignore')
```

In [12]:
```
df = df.dropna()
df.count()
```

Out[12]:
```
class                     5644
cap-shape                 5644
cap-surface               5644
cap-color                 5644
bruises                   5644
odor                      5644
gill-attachment           5644
gill-spacing              5644
gill-size                 5644
gill-color                5644
stalk-shape               5644
stalk-root                5644
stalk-surface-above-ring  5644
stalk-surface-below-ring  5644
stalk-color-above-ring    5644
stalk-color-below-ring    5644
veil-type                 5644
veil-color                5644
ring-number               5644
ring-type                 5644
spore-print-color         5644
population                5644
habitat                   5644
dtype: int64
```

In [13]:
```
print(df.isnull().sum())
```

```
class                     0
cap-shape                 0
cap-surface               0
cap-color                 0
bruises                   0
odor                      0
gill-attachment           0
gill-spacing              0
gill-size                 0
gill-color                0
stalk-shape               0
stalk-root                0
stalk-surface-above-ring  0
stalk-surface-below-ring  0
stalk-color-above-ring    0
stalk-color-below-ring    0
veil-type                 0
veil-color                0
ring-number               0
ring-type                 0
spore-print-color         0
population                0
habitat                   0
dtype: int64
```

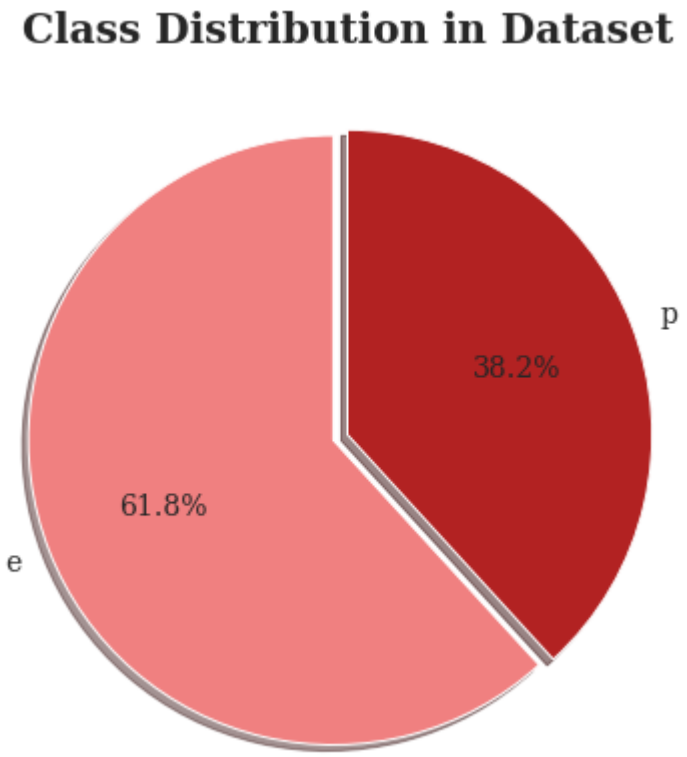
In [14]:
```
labels =df['veil-type'].value_counts(sort = True).index
values = df['veil-type'].value_counts(sort = True)
print(values)
```

```
p    5644
Name: veil-type, dtype: int64
```

we will remove the column because he only has one value

Describing the classify column

In [15]:
```
labels =df['class'].value_counts(sort = True).index
sizes = df['class'].value_counts(sort = True)

colors = ['lightcoral','firebrick']
explode = (0.05,0)

plt.figure(figsize=(7,7))
plt.pie(sizes, explode=explode, labels=labels, colors=colors, autopct='%1.1f%%',
        shadow=True, startangle=90, textprops={'fontsize': 14, 'fontfamily':'serif'})

plt.title('Class Distribution in Dataset', fontsize=20, fontweight='bold', fontfamily=
plt.show()
```

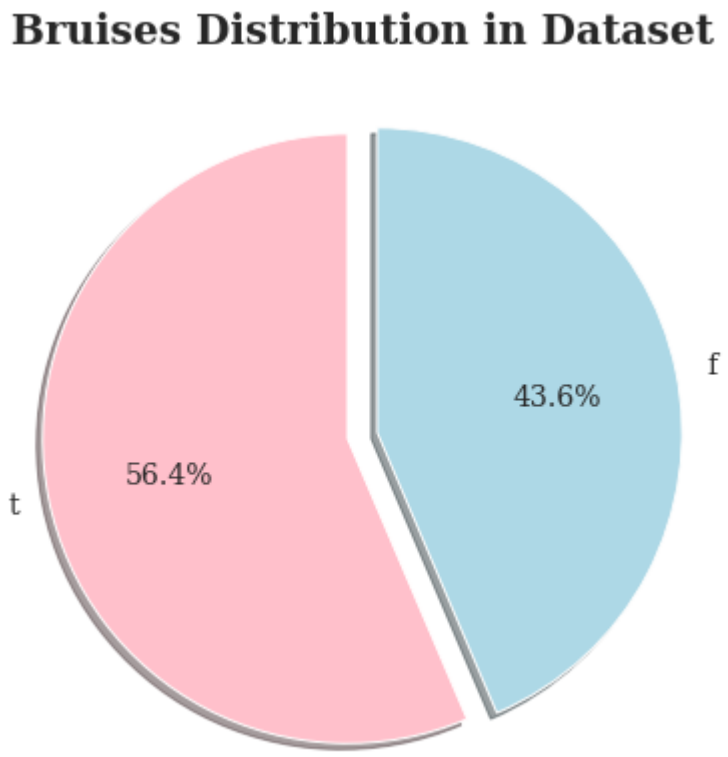


Categorical variables

In [16]:
```
labels =df['bruises'].value_counts(sort = True).index
values = df['bruises'].value_counts(sort = True)
print(values)

colors = ['pink','lightblue','mediumaquamarine']
explode = (0.05, 0.05)

plt.figure(figsize=(7,7))
plt.pie(values, labels=labels, explode=explode, colors=colors, autopct='%1.1f%%', shad
```

```
        startangle=90, textprops={'fontsize': 14, 'fontfamily':'serif'})

plt.title('Bruises Distribution in Dataset', fontsize=20, fontweight='bold', fontfamil
plt.show()
```

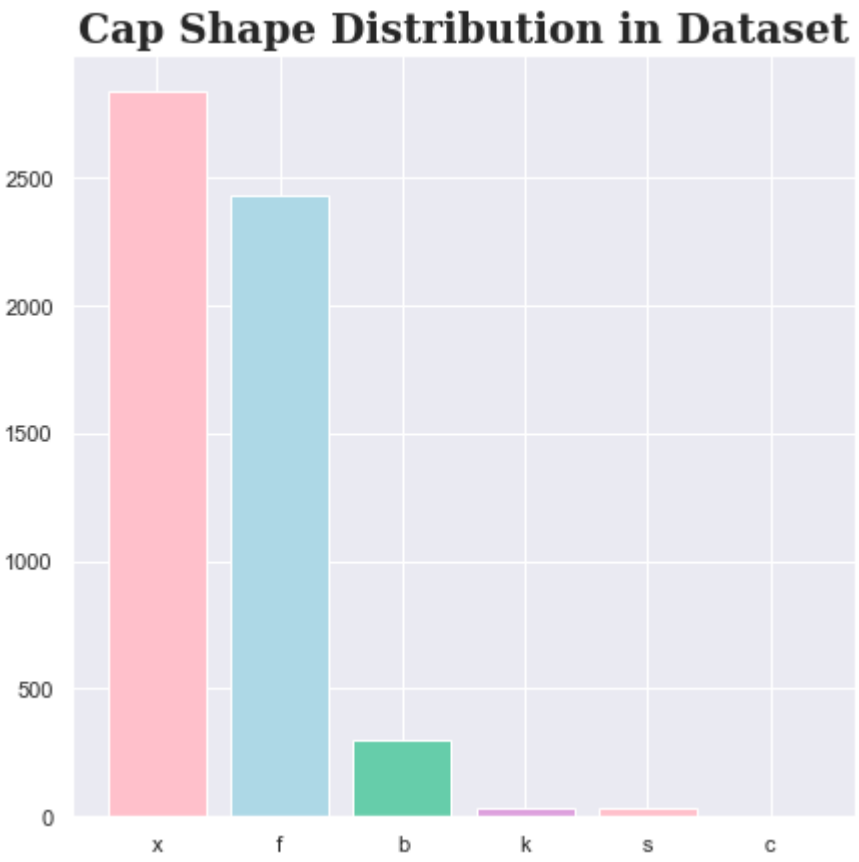
```
t    3184
f    2460
Name: bruises, dtype: int64
```



In [17]:

```
labels =df['cap-shape'].value_counts(sort = True).index
values = df['cap-shape'].value_counts(sort = True)
print(values)

labels =df['cap-shape'].value_counts(sort = True).index
values = df['cap-shape'].value_counts(sort = True)

plt.figure(figsize=(7,7))

plt.bar(labels,values, color = ['pink','lightblue','mediumaquamarine','plum'])

plt.title('Cap Shape Distribution in Dataset', fontsize=20, fontweight='bold', fontfam
plt.show()
```

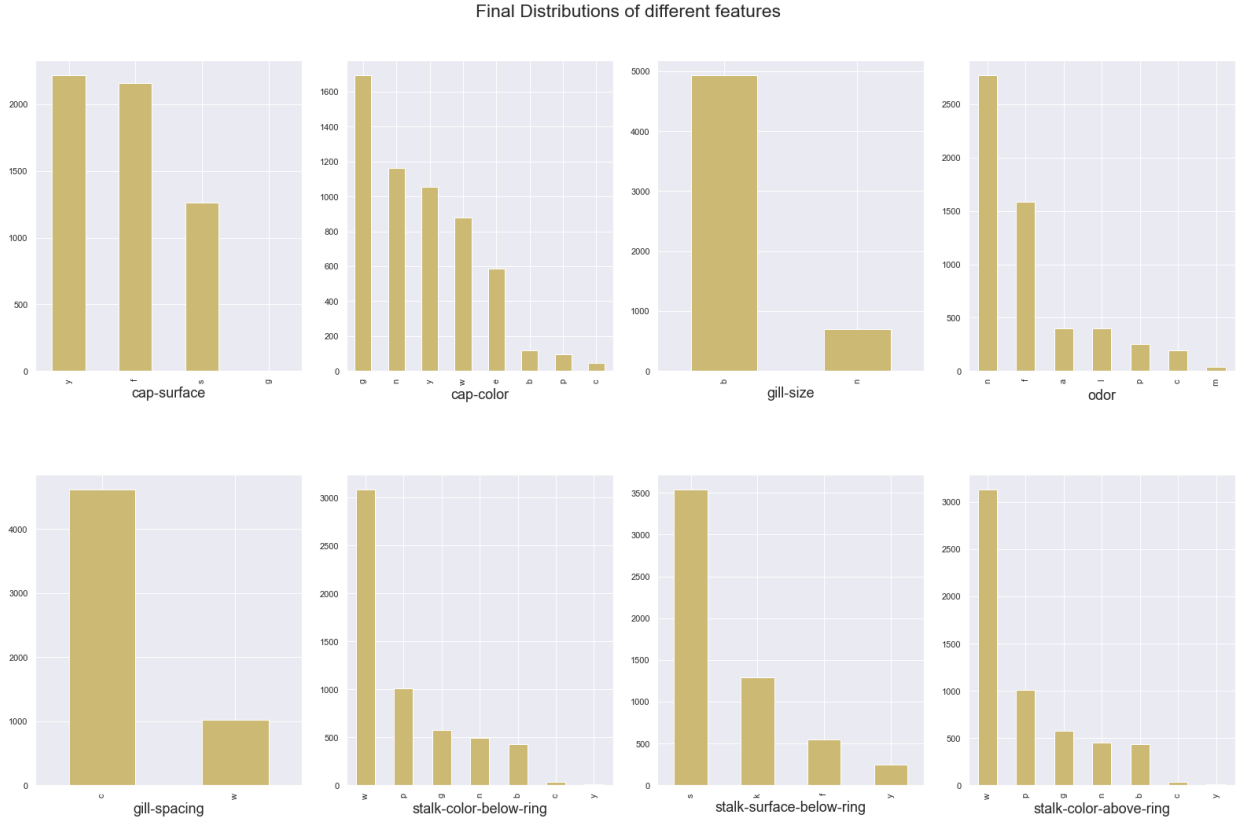
```
x    2840
f    2432
b     300
k      36
s      32
c       4
Name: cap-shape, dtype: int64
```



In [18]:
```
labels =df['cap-surface'].value_counts(sort = True).index
values = df['cap-surface'].value_counts(sort = True)
print(values)
```

```
y    2220
f    2160
s    1260
g       4
Name: cap-surface, dtype: int64
```

In [19]:
```
def plot_dist(col, ax):
    if col != 'height':
        df[col].value_counts().plot(kind='bar', facecolor='y', ax=ax)
    else:
        df[col].plot('density', ax=ax, bw_method = 0.15, color='y')
        ax.set_xlim(130,200)
        ax.set_ylim(0, 0.07)
    ax.set_xlabel('{}'.format(col), fontsize=18)
    # ax.set_title("{} on Modcloth".format(col), fontsize= 18)
    return ax

f, ax = plt.subplots(2,4, figsize = (22,15))
f.tight_layout(h_pad=9, w_pad=2, rect=[0, 0.03, 1, 0.93])
cols = ['cap-surface','cap-color', 'gill-size', 'odor', 'gill-spacing', 'stalk-color-b
k = 0
for i in range(2):
    for j in range(4):
        plot_dist(cols[k], ax[i][j])
        k += 1
__ = plt.suptitle("Final Distributions of different features", fontsize= 23)
```



In [20]:
```
for column in df.columns:
    df[column] = LabelEncoder().fit_transform(df[column])
```

In [21]:
```
classify_col = df.pop('class')
classify_col
```

Out[21]:
```
0       1
1       0
2       0
3       1
4       0
       ..
7986    0
8001    0
8038    0
8095    1
8114    1
Name: class, Length: 5644, dtype: int32
```

In [22]:
```
df.head(5)
```

Out[22]:

	cap-shape	cap-surface	cap-color	bruises	odor	gill-attachment	gill-spacing	gill-size	gill-color	stalk-shape	...	stalk-surface-below-ring	stall colo above rin
0	5	2	4	1	6	1	0	1	2	0	...	2	
1	5	2	7	1	0	1	0	0	2	0	...	2	
2	0	2	6	1	3	1	0	0	3	0	...	2	
3	5	3	6	1	6	1	0	1	3	0	...	2	
4	5	2	3	0	5	1	1	0	2	1	...	2	

5 rows × 22 columns

Models

common function for models

In [23]:
```
def confusionMatrix(predicted_test, title):
  background_color = "#fbfbfb"
  cm = confusion_matrix(class_test, predicted_test)

  fig = plt.figure(figsize=(7,5)) # create figure
  gs = fig.add_gridspec(1, 1)
  gs.update(wspace=0.1, hspace=0.8)
  ax0 = fig.add_subplot(gs[0, :])

  ax0.set_facecolor(background_color) # axes background color

  # Overall
  sns.heatmap(cm, annot=True,fmt="d", linewidths=5,cbar=False,ax=ax0,
              yticklabels=['Actual 0','Actual 1'],xticklabels=['Predicted 0','Predicte


  ax0.tick_params(axis=u'both', which=u'both',length=0)
  background_color = "#fbfbfb"
  fig.patch.set_facecolor(background_color) # figure background color
  ax0.set_facecolor(background_color)


  ax0.text(0,-0.75,title,fontsize=18,fontweight='bold',fontfamily='serif')


  plt.show()
```

In [24]:
```
def summery(predicted_test, title):
  background_color = "#fbfbfb"
  summery_test = pd.DataFrame(data=[f1_score(class_test,predicted_test),accuracy_score
                                    precision_score(class_test, predicted_test), roc_auc
              columns=['Test values'],
              index=["F-measure","Accuracy", "Recall", "Precision", "ROC AUC Score"])
```

```
fig = plt.figure(figsize=(10,1)) # create figure
gs = fig.add_gridspec(1, 1)
gs.update(wspace=0.1, hspace=0.5)
ax0 = fig.add_subplot(gs[0, :])


sns.heatmap(summery_test.T,annot=True,fmt=".1%",vmin=0,vmax=0.95, linewidths=2.5,cba
fig.patch.set_facecolor(background_color) # figure background color
ax0.set_facecolor(background_color)


ax0.text(0,-0.5,title,fontsize=18,fontweight='bold',fontfamily='serif')
ax0.tick_params(axis=u'both', which=u'both',length=0)

plt.show()
```

Test 1 - splitting the data so 80% is for training

In [25]:
```
data_train, data_test, class_train, class_test = train_test_split(df, classify_col, tr
```

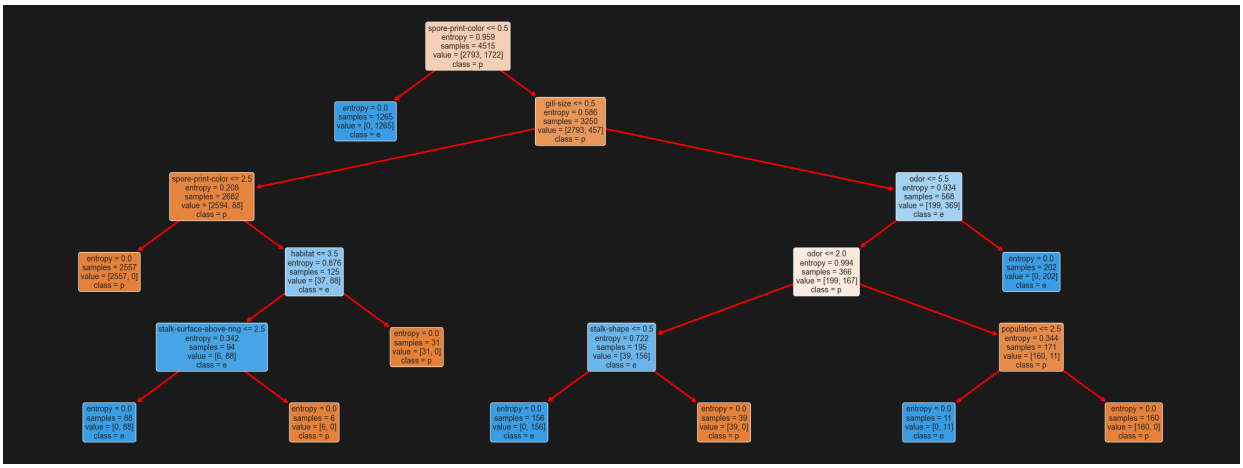
Decision Tree

In [26]:
```
dtc = DecisionTreeClassifier(criterion='entropy')
dtc.fit(data_train, class_train)
predicted_train = dtc.predict(data_train)
print("Train accuracy score using Decision Tree is: {}%".format(accuracy_score(class_t

predicted_test = dtc.predict(data_test)
print("Test accuracy score using Decision Tree is: {}%".format(accuracy_score(class_te
```

```
Train accuracy score using Decision Tree is: 100.0%
Test accuracy score using Decision Tree is: 100.0%
```

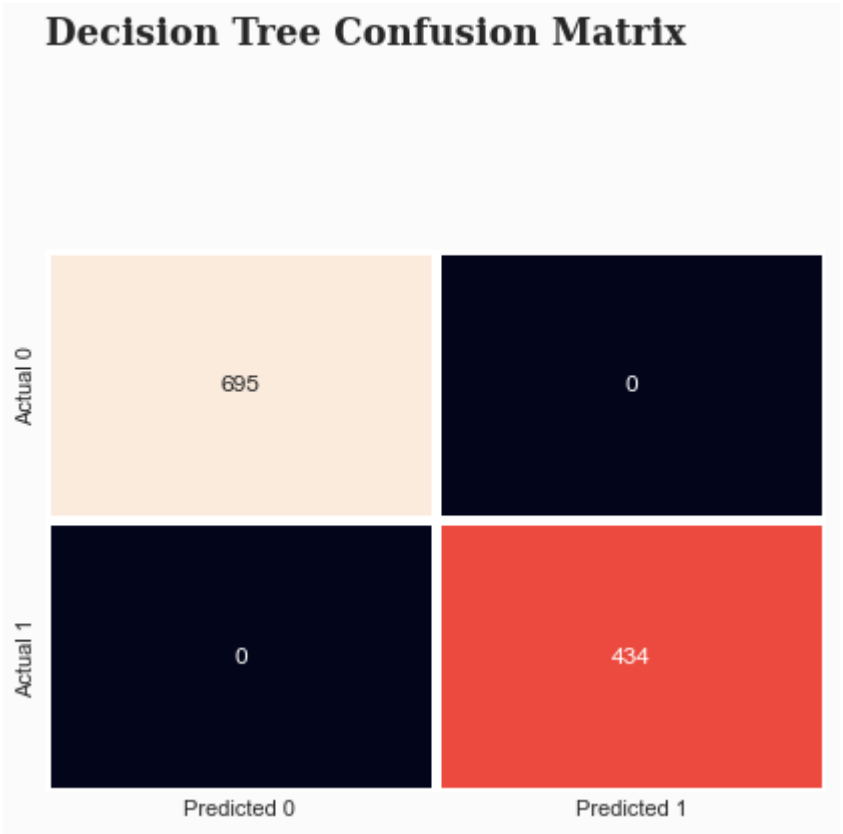
In [27]:
```
from sklearn import tree

import matplotlib.pyplot as plt

plt.figure(figsize=(40,15), facecolor ='k')

a = tree.plot_tree(dtc,

                   feature_names = data_train.columns,

                   class_names = ['p', 'e'],

                   rounded = True,

                   filled = True,
```

```
                    fontsize=14)
for o in a:
    arrow = o.arrow_patch
    if arrow is not None:
        arrow.set_edgecolor('red')
        arrow.set_linewidth(3)

plt.show()
```



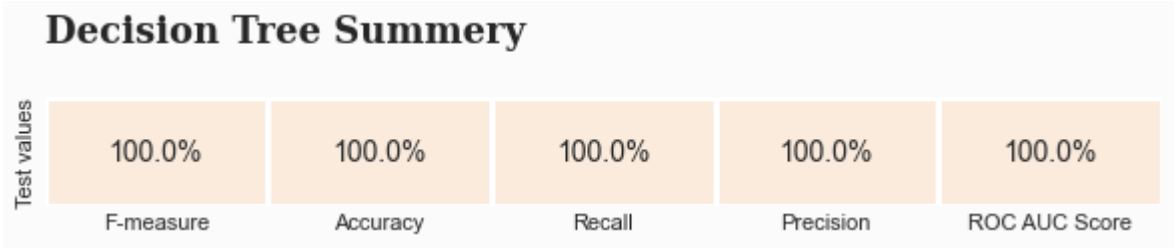
Confusion matrix DT

In [28]:
```
confusionMatrix(predicted_test, 'Decision Tree Confusion Matrix')
```



Summery

In [29]:
```
summery(predicted_test, 'Decision Tree Summery')
```



NB

sklearn implement

In [30]:
```
gnb = GaussianNB()
gnb.fit(data_train, class_train)
predicted_train_NB = gnb.predict(data_train)

print("Train accuracy score using Decision Tree is: {}%".format(accuracy_score(class_t

predicted_test_NB = gnb.predict(data_test)
print("Test accuracy score using Decision Tree is: {}%".format(accuracy_score(class_te
```

```
Train accuracy score using Decision Tree is: 70.58693244739757%
Test accuracy score using Decision Tree is: 69.9734278122232%
```

self implement

In [65]:
```
class selfNaiveBayes:
    """
        class that builds self made naive bayes model and predict by new data
    """
    def __init__(self, data_train, class_train):
        """
            init the naive bayes with the train data
            :param data_train: the train data
            :param class_train: the train classify column
        """
        self.data_train = data_train
        self.class_train = class_train
        self.bayesCalcs = {}
        self.pClass = class_train.value_counts() / (len(class_train))
        self.pClass.index.name = class_train.name
        temp = data_train.join(class_train.to_frame(), how="inner")
        for column in temp.columns:
            if column != class_train.name:
                self.bayesCalcs[column] = temp.groupby([class_train.name, column]).siz

    def calcBayes(self,*args):
        """
            calculate the bayes probability
            :param args: the data (row of data from data_test)
            :return: the predicted classify
        """
        classify = ("", 0)
        calc = 1
```

```
        for classOpt in self.class_train.unique():
            calc = 1
            for column in self.data_train.columns:
                try:
                    calc *= self.bayesCalcs[column][classOpt][args[0][self.data_train.
                except KeyError:
                    calc *= 1
            calc *= self.pClass[classOpt]
            if calc > classify[1]:
                classify = (classOpt, calc)
        return classify[0]
```

In [66]:
```
model = selfNaiveBayes(data_train,class_train)
```

In [33]:
```
predicted_test_NB = np.array([])
for row in data_test.values.tolist():
        # run the test prediction for each row of the given data and append it to the
  if predicted_test_NB.size == 0:predicted_test_NB = np.array([model.calcBayes(row)])
  else:
        predicted_test_NB = np.append(predicted_test_NB, model.calcBayes(row))

    # prediction of train data for the self-made NB model
predicted_train_NB = np.array([])
for row in data_train.values.tolist():
  # run the train prediction for each row of the given data and append it to the predi
  if predicted_train_NB.size == 0:predicted_train_NB = np.array([model.calcBayes(row)]
  else:
    predicted_train_NB = np.append(predicted_train_NB, model.calcBayes(row))
# return train_pred_array, test_pred_array
```

In [67]:
```
predicted_test_NB
```

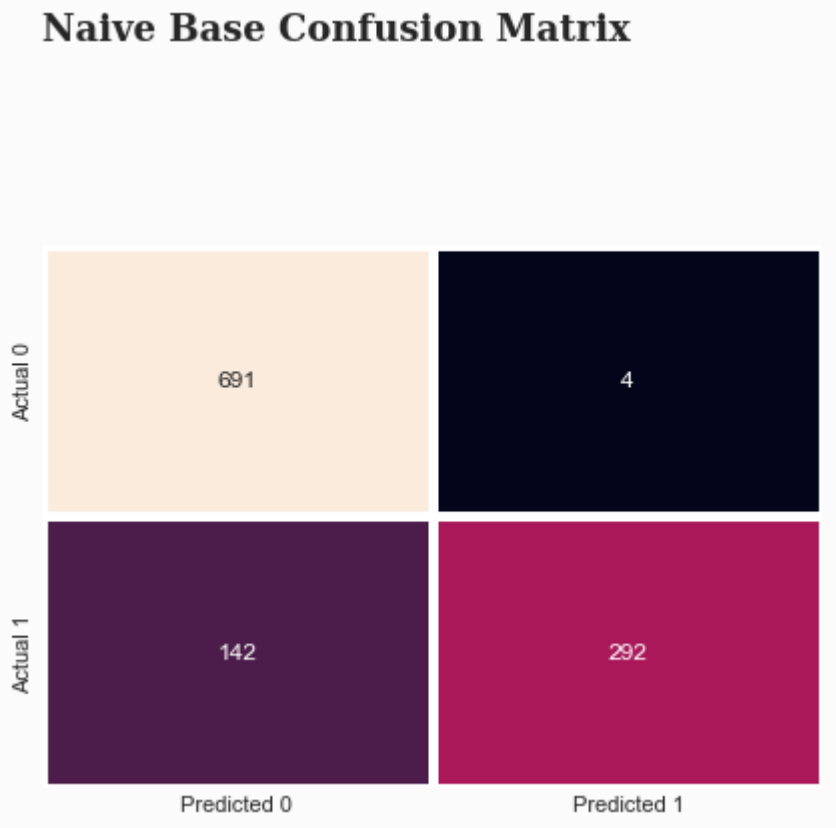
Out[67]:
```
array([0, 0, 0, ..., 0, 0, 0])
```

In [68]:
```
print("Train accuracy score using Decision Tree is: {}%".format(accuracy_score(class_t
print("Test accuracy score using Decision Tree is: {}%".format(accuracy_score(class_te
```

```
Train accuracy score using Decision Tree is: 69.76990214229039%
Test accuracy score using Decision Tree is: 72.08803005904456%
```

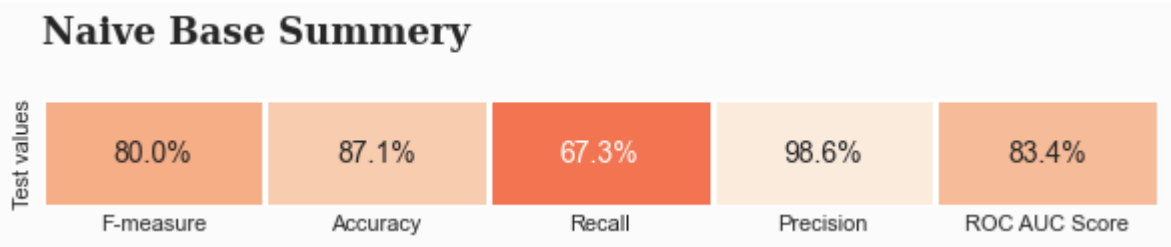
Confusion matrix NB

In [36]:
```
confusionMatrix(predicted_test_NB, 'Naive Base Confusion Matrix')
```



Summery

In [37]:
```
summery(predicted_test_NB, 'Naive Base Summery')
```



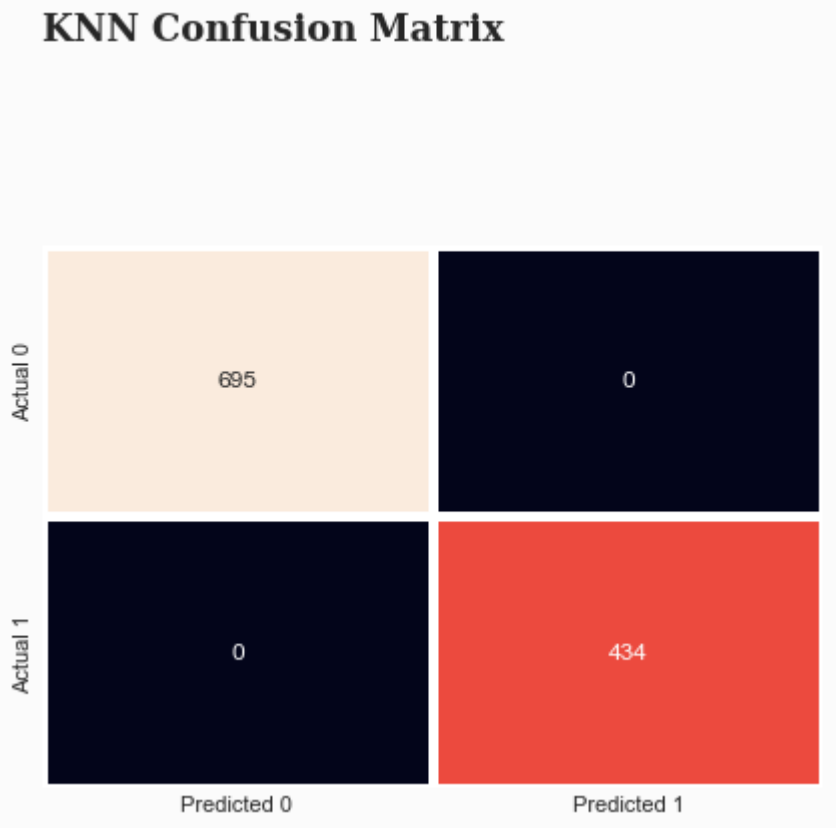
KNN

In [38]:
```
knn = KNeighborsClassifier()
knn.fit(data_train, class_train)
predicted_train_KNN = knn.predict(data_train)

print("Train accuracy score using Decision Tree is: {}%".format(accuracy_score(class_t

predicted_test_KNN = knn.predict(data_test)
print("Test accuracy score using Decision Tree is: {}%".format(accuracy_score(class_te
```

```
Train accuracy score using Decision Tree is: 100.0%
Test accuracy score using Decision Tree is: 100.0%
```

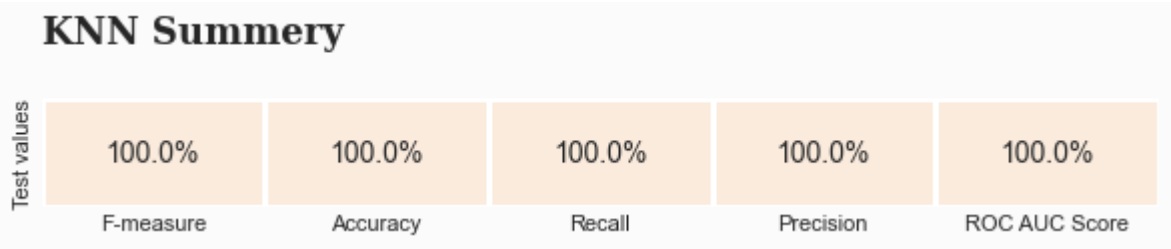
Confusion matrix KNN

In [39]:
```
confusionMatrix(predicted_test_KNN, 'KNN Confusion Matrix')
```



Summery

In [40]:
```
summery(predicted_test_KNN, 'KNN Summery')
```



K-Means

In [41]:
```
km = KMeans()
km.fit(data_train)

predicted_train_KM = km.predict(data_train)
print("Train accuracy score using Decision Tree is: {}%".format(accuracy_score(class_t

predicted_test_KM = km.predict(data_test)
print("Test accuracy score using Decision Tree is: {}%".format(accuracy_score(class_te
```

```
Train accuracy score using Decision Tree is: 14.507198228128459%
Test accuracy score using Decision Tree is: 12.843224092116918%
```

Test 2 - splitting the data so 67% is for training

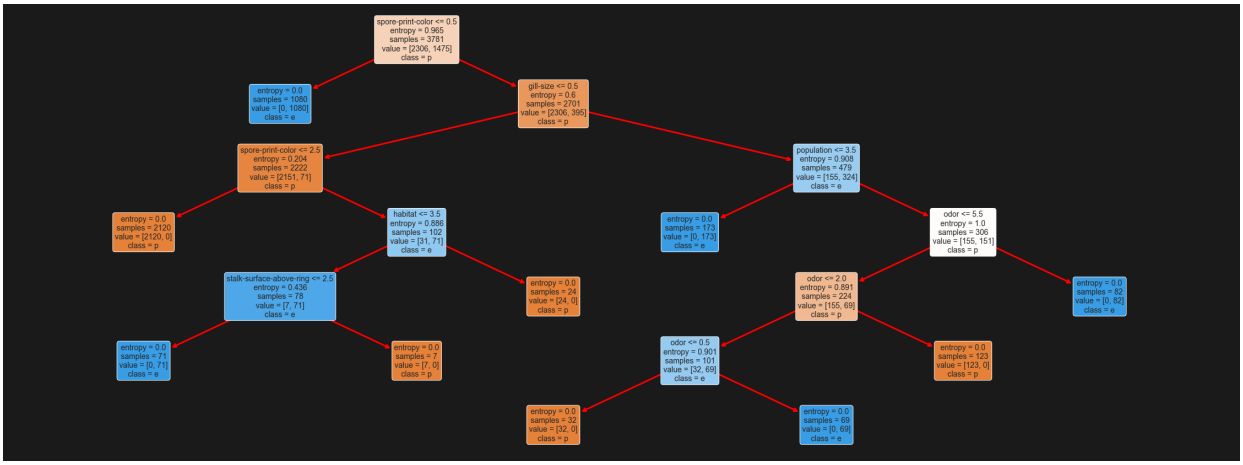
In [42]:
```
data_train, data_test, class_train, class_test = train_test_split(df, classify_col, tr
```

Decision Tree

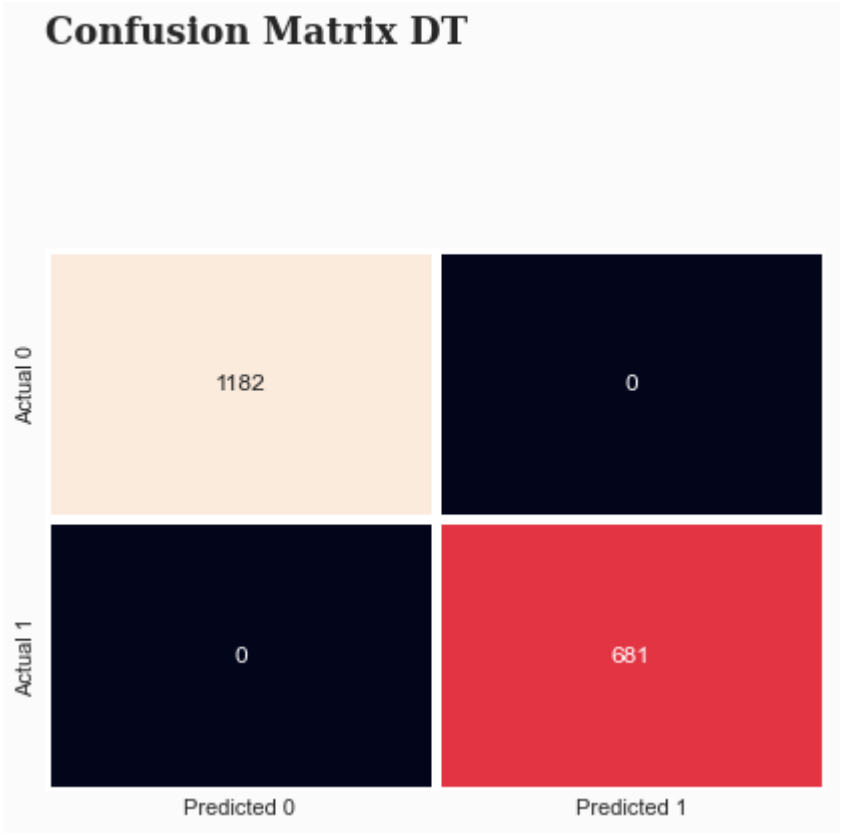
In [43]:
```
dtc = DecisionTreeClassifier(criterion='entropy')
dtc.fit(data_train, class_train)
predicted_train = dtc.predict(data_train)
print("Train accuracy score using Decision Tree is: {}%".format(accuracy_score(class_t

predicted_test = dtc.predict(data_test)
print("Test accuracy score using Decision Tree is: {}%".format(accuracy_score(class_te
```

```
Train accuracy score using Decision Tree is: 100.0%
Test accuracy score using Decision Tree is: 100.0%
```

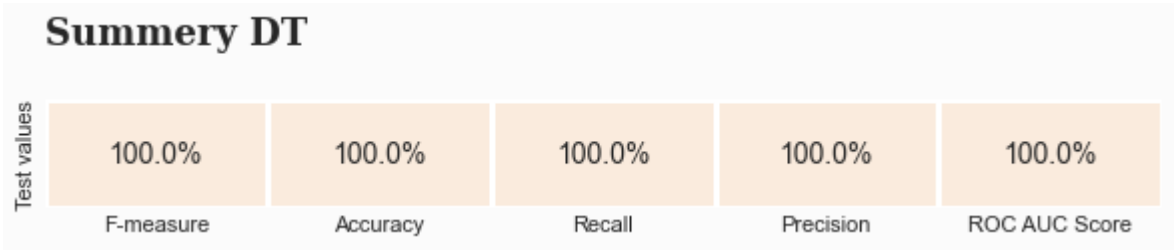
In [44]:
```
from sklearn import tree

import matplotlib.pyplot as plt

plt.figure(figsize=(40,15), facecolor ='k')

a = tree.plot_tree(dtc,

                   feature_names = data_train.columns,

                   class_names = ['p', 'e'],

                   rounded = True,

                   filled = True,

                   fontsize=14)
for o in a:
    arrow = o.arrow_patch
    if arrow is not None:
        arrow.set_edgecolor('red')
        arrow.set_linewidth(3)

plt.show()
```



In [45]:
```
confusionMatrix(predicted_test,'Confusion Matrix DT')
```



In [46]:
```
summery(predicted_test,'Summery DT')
```



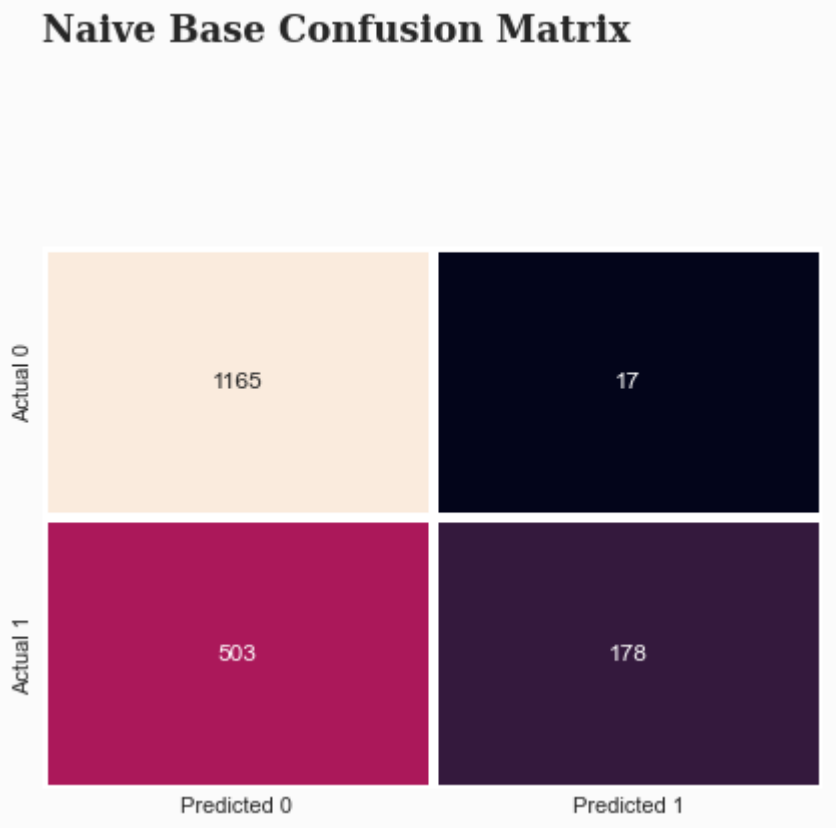
NB

In [47]:
```
gnb = GaussianNB()
```

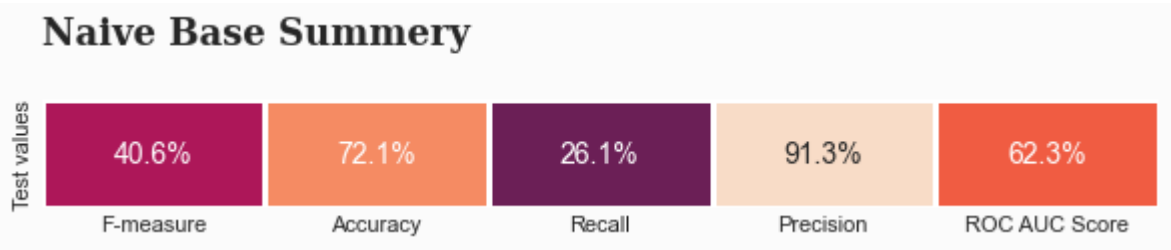
```
gnb.fit(data_train, class_train)
predicted_train_NB = gnb.predict(data_train)

print("Train accuracy score using Decision Tree is: {}%".format(accuracy_score(class_t

predicted_test_NB = gnb.predict(data_test)
print("Test accuracy score using Decision Tree is: {}%".format(accuracy_score(class_te
```

```
Train accuracy score using Decision Tree is: 69.76990214229039%
Test accuracy score using Decision Tree is: 72.08803005904456%
```

In [48]:
```
confusionMatrix(predicted_test_NB, 'Naive Base Confusion Matrix')
```



In [49]:
```
summery(predicted_test_NB, 'Naive Base Summery')
```



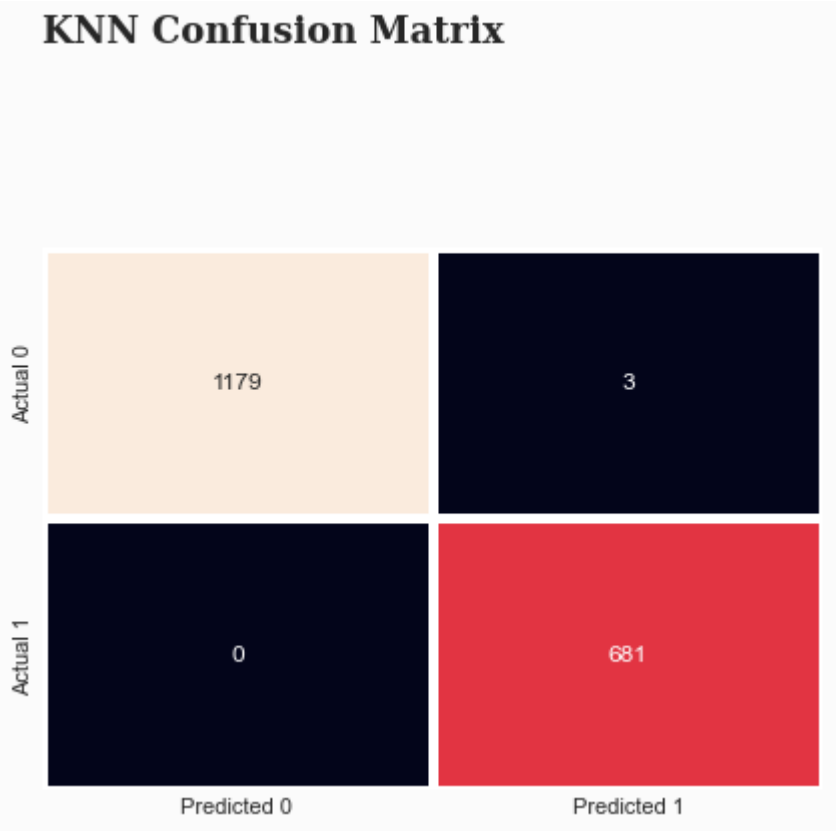
KNN

In [50]:
```
knn = KNeighborsClassifier()
knn.fit(data_train, class_train)
predicted_train_KNN = knn.predict(data_train)

print("Train accuracy score using Decision Tree is: {}%".format(accuracy_score(class_t
```

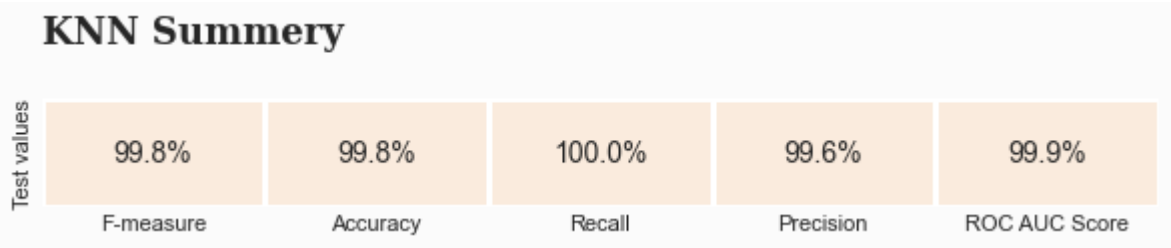
```
predicted_test_KNN = knn.predict(data_test)
print("Test accuracy score using Decision Tree is: {}%".format(accuracy_score(class_te
```

```
Train accuracy score using Decision Tree is: 99.9471039407564%
Test accuracy score using Decision Tree is: 99.8389694041868%
```

In [51]:
```
confusionMatrix(predicted_test_KNN, 'KNN Confusion Matrix')
```



In [52]:
```
summery(predicted_test_KNN, 'KNN Summery')
```



K-Means

In [53]:
```
km = KMeans()
km.fit(data_train)

predicted_train_KM = km.predict(data_train)
print("Train accuracy score using Decision Tree is: {}%".format(accuracy_score(class_t

predicted_test_KM = km.predict(data_test)
print("Test accuracy score using Decision Tree is: {}%".format(accuracy_score(class_te
```

```
Train accuracy score using Decision Tree is: 13.938111610685002%
Test accuracy score using Decision Tree is: 14.492753623188406%
```

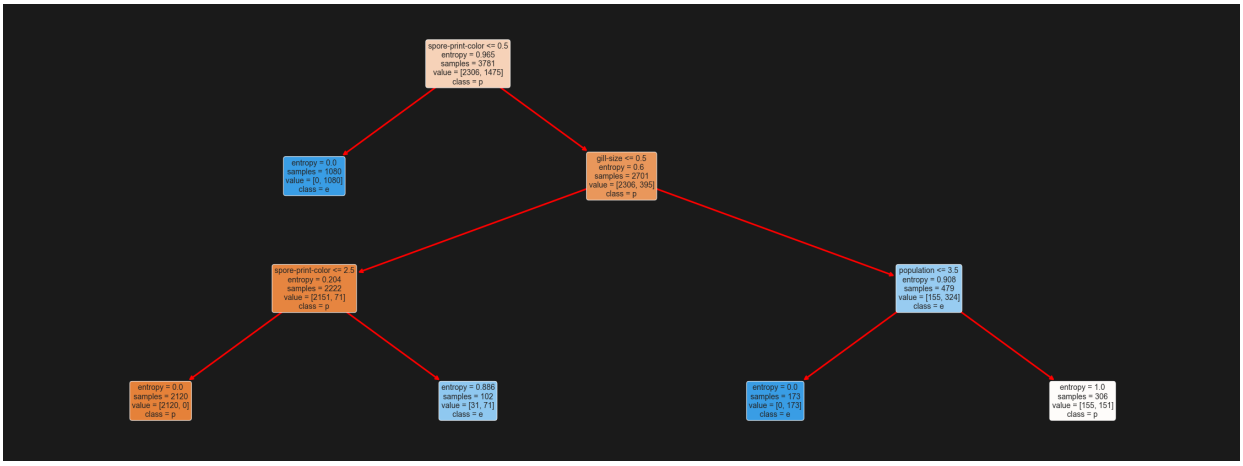
Test #3 - light Hyperparameter tuning

Decision Tree

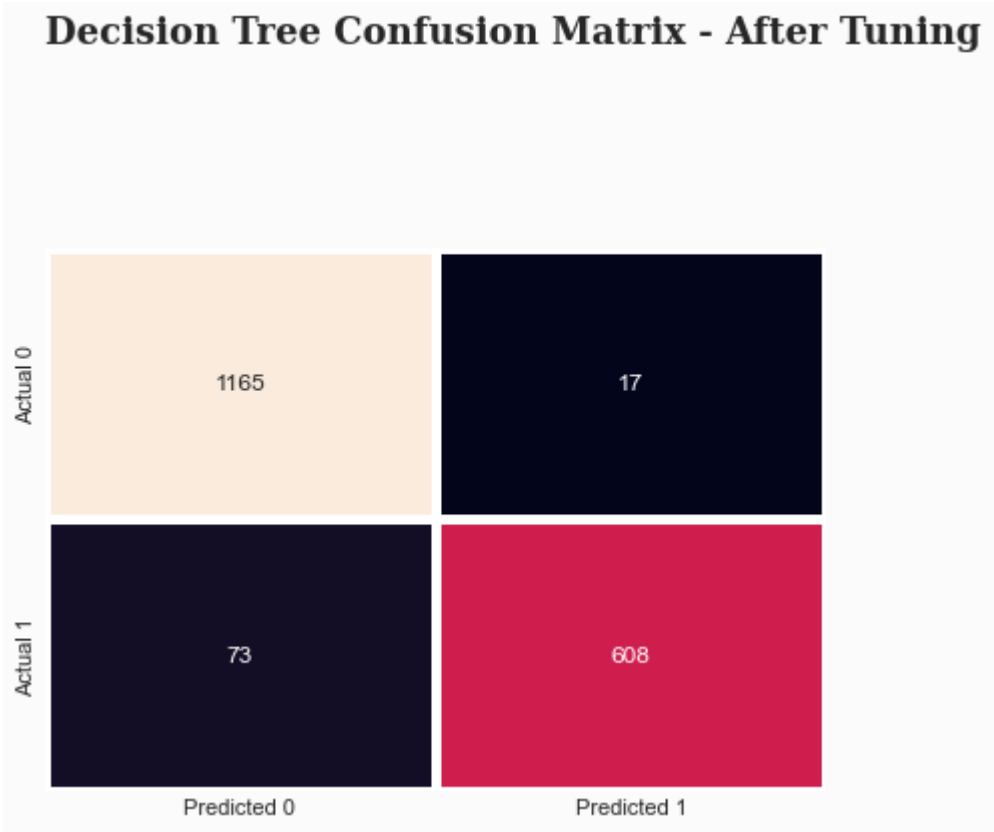
In [54]:
```
dtc = DecisionTreeClassifier(max_depth=3,criterion='entropy')
dtc.fit(data_train, class_train)
predicted_train = dtc.predict(data_train)
print("Train accuracy score using Decision Tree is: {}%".format(accuracy_score(class_t

predicted_test = dtc.predict(data_test)
print("Test accuracy score using Decision Tree is: {}%".format(accuracy_score(class_te
```

```
Train accuracy score using Decision Tree is: 95.18645860883365%
Test accuracy score using Decision Tree is: 95.16908212560386%
```

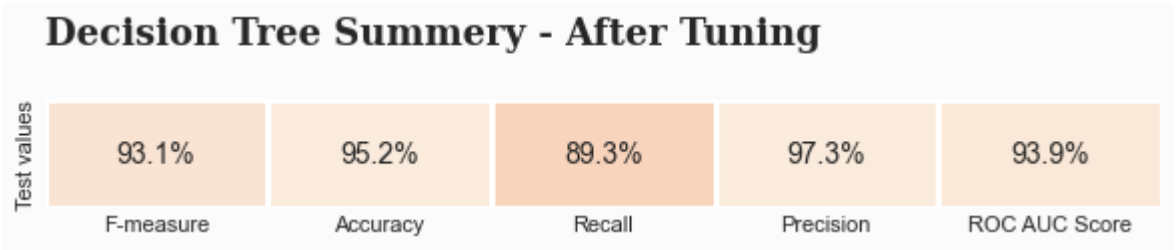
In [55]:
```
from sklearn import tree

import matplotlib.pyplot as plt

plt.figure(figsize=(40,15), facecolor ='k')

a = tree.plot_tree(dtc,

                   feature_names = data_train.columns,

                   class_names = ['p', 'e'],

                   rounded = True,

                   filled = True,

                   fontsize=14)
for o in a:
    arrow = o.arrow_patch
    if arrow is not None:
        arrow.set_edgecolor('red')
        arrow.set_linewidth(3)

plt.show()
```



In [56]:
```
confusionMatrix(predicted_test, 'Decision Tree Confusion Matrix - After Tuning')
```



In [57]:
```
summery(predicted_test, 'Decision Tree Summery - After Tuning')
```



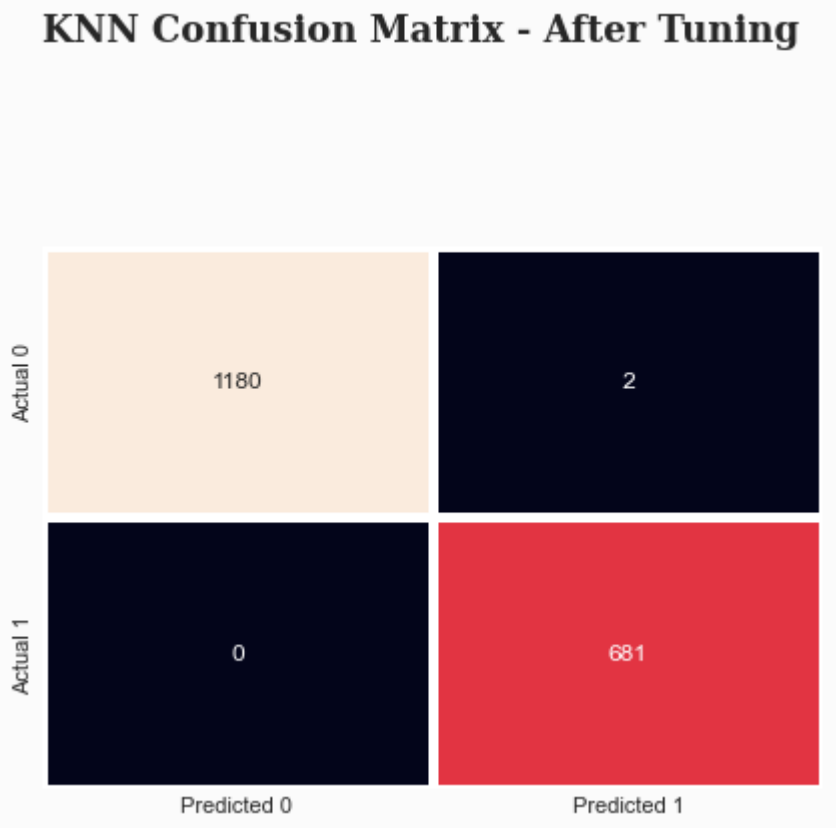
KNN

In [58]:
```
knn = KNeighborsClassifier(n_neighbors=6)
```

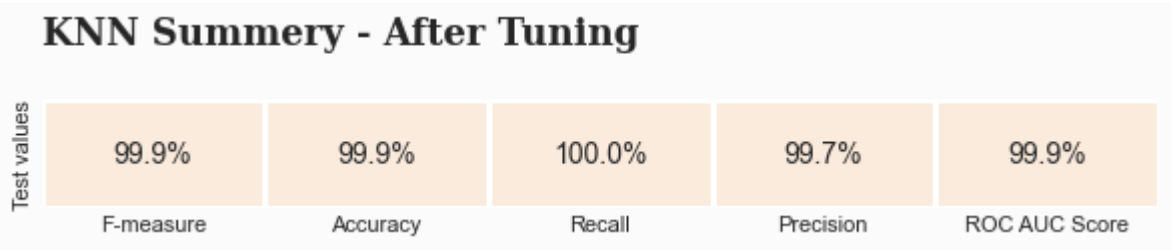
```
knn.fit(data_train, class_train)
predicted_train_KNN = knn.predict(data_train)

print("Train accuracy score using Decision Tree is: {}%".format(accuracy_score(class_t

predicted_test_KNN = knn.predict(data_test)
print("Test accuracy score using Decision Tree is: {}%".format(accuracy_score(class_te
```

```
Train accuracy score using Decision Tree is: 99.97355197037821%
Test accuracy score using Decision Tree is: 99.89264626945786%
```

In [59]:
```
confusionMatrix(predicted_test_KNN, 'KNN Confusion Matrix - After Tuning')
```



In [60]:
```
summery(predicted_test_KNN, 'KNN Summery - After Tuning')
```



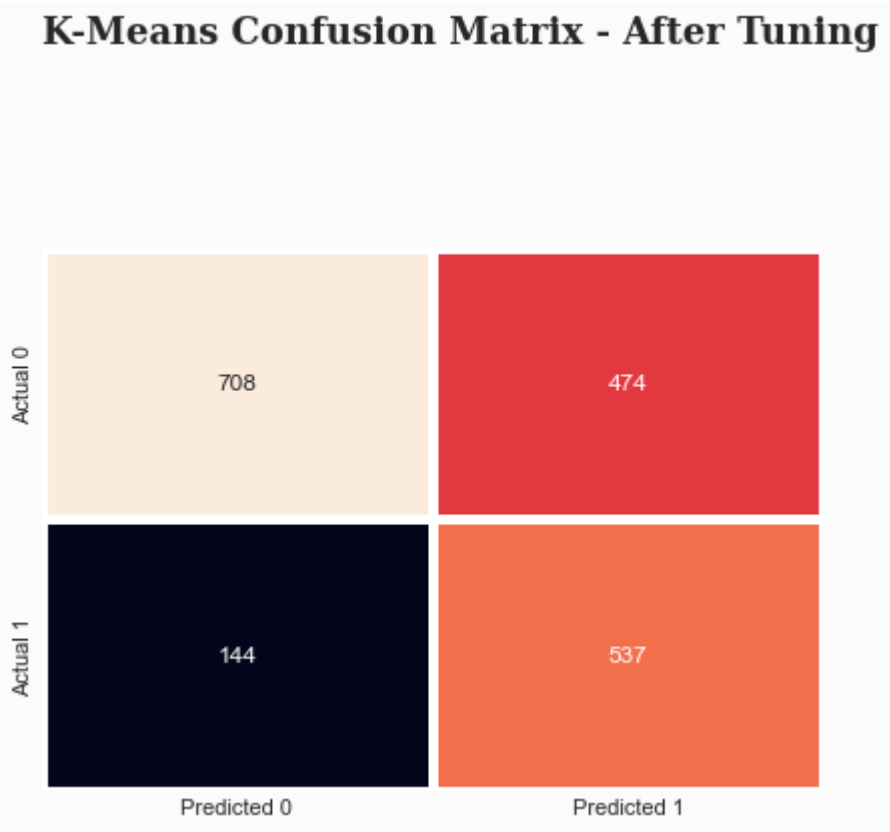
K-Means

In [61]:
```
km = KMeans(n_clusters=2)
km.fit(data_train)

predicted_train_KM = km.predict(data_train)
print("Train accuracy score using Decision Tree is: {}%".format(accuracy_score(class_t

predicted_test_KM = km.predict(data_test)
print("Test accuracy score using Decision Tree is: {}%".format(accuracy_score(class_te
```

```
Train accuracy score using Decision Tree is: 67.62761174292515%
Test accuracy score using Decision Tree is: 66.82769726247987%
```

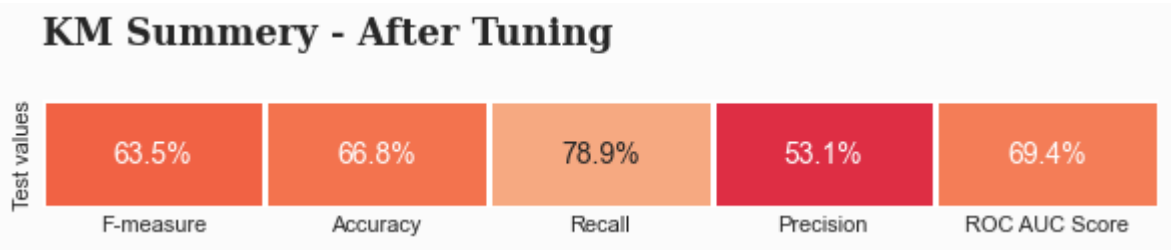
confusion matrix

In [62]:
```
confusionMatrix(predicted_test_KM, 'K-Means Confusion Matrix - After Tuning')
```



summery

In [63]:
```
summery(predicted_test_KM, 'KM Summery - After Tuning')
```



סיכום ומסקנות

ניתן לראות לאורך שלושת הניסויים כי אחוזי הדיוק עולים בין ניסוי לניסוי. אפשר לראות כי יש במספר מודלים אובר פיטינג שמסדרים לאחר היפרטיונינג וכאלה שלא. לאחר ההיפר טיונינג של העץ החלטה הוא המודל המומלץ מבחינת אחוזי דיוק
אך לפי שינוי אחוזי אימון ובדיקה למרות אחוזים יחסית נמוכים הוא הנאיב בייס. אם מגדילים את גודל קובץ האימון ניתן לראות שיפור בדיוק.

In [ ]: