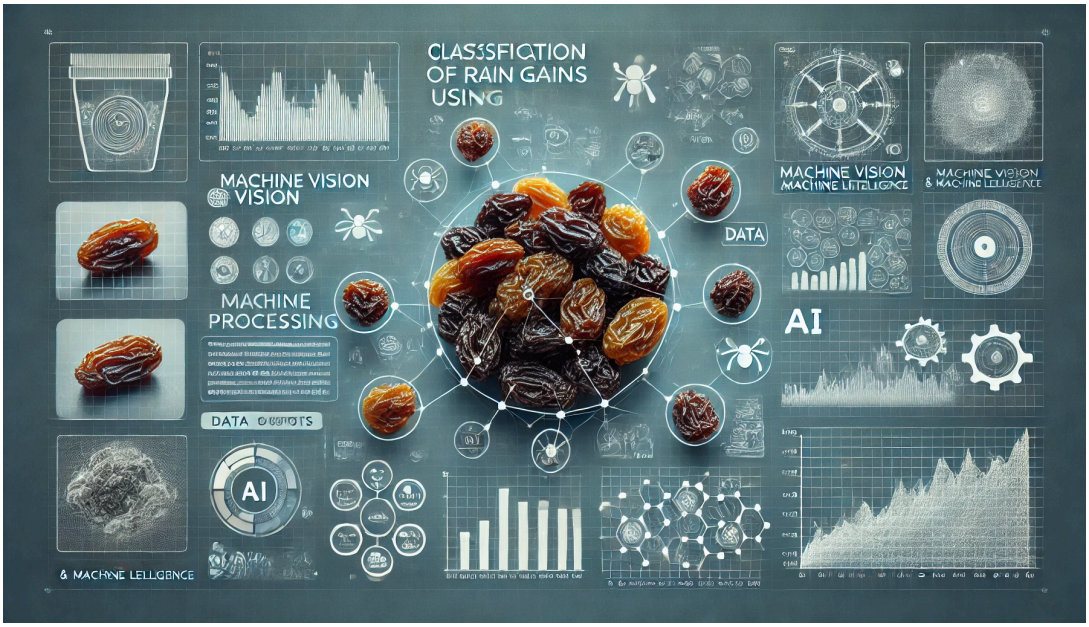
Raisin Class Prediction

If you want to be the first to be informed about new projects, please do not forget to follow us - by Fatma Nur AZMAN

Fatmanurazman.com | Linkedin | Github | Kaggle | Tableau



Understanding The Data

Data Set Information:

https://www.researchgate.net/publication/347934123_Classification_of_Raisin_Grains_Using

https://dergipark.org.tr/en/download/article-file/1227592

Project Description:

- **Raisin Class Prediction** In this study, machine vision system was developed in order to distinguish between two different variety of raisins (Kecimen and Besni) grown in Turkey. Firstly, a total of 900 pieces raisin grains were obtained, from an equal number of both varieties. These images were subjected to various preprocessing steps and 7 morphological feature extraction operations were performed using image processing techniques. In addition, minimum, mean, maximum and standard deviation tatistical information was calculated for each feature. The distributions of both raisin varieties on he features were examined and these distributions were shown on the graphs. Later, models were reated using LR, KNN, and SVM machine learning techniques and performance measurements ere performed.

About the Dataset

Dataset Descriptions:

- **Rows:** 900
- **Columns:** 8

STT	Attribute Name	Unique Values
1	Area	Gives the number of pixels within the boundaries of the raisin grain.
2	Perimeter	It measures the environment by calculating the distance between the boundaries of the raisin grain and the pixels around it.
3	MajorAxisLength	Gives the length of the main axis, which is the longest line that can be drawn on the raisin grain.
4	MinorAxisLength	Gives the length of the small axis, which is the shortest line that can be drawn on the raisin grain.
5	Eccentricity	It gives a measure of the eccentricity of the ellipse, which has the same moments as raisins.
6	ConvexArea	Gives the number of pixels of the smallest convex shell of the region formed by the raisin grain.
7	Extent	Gives the ratio of the region formed by the raisin grain to the total pixels in the bounding box.
8	Class	The class of the raisin grain, indicating the type of raisin.

Table of CONTENTS

- Understanding The Data
- Exploratory Data Analysis (EDA)
- Detection of Outliers
- Correlation
- Models
- Logistic Regression Model
- Compare Logistic Regression Models
- KNN Model
- SVM Model
- Decision Tree Classification Model
- Compare Models Performance
- Final Model and Model Deployment
- Prediction
  - Conclusion

Import Libraries and Data Review

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import plotly.express as px
import seaborn as sns

%matplotlib inline

from scipy import stats
from sklearn.model_selection import train_test_split, GridSearchCV, cross_valic
from sklearn.preprocessing import StandardScaler, MinMaxScaler, RobustScaler
from sklearn.preprocessing import PowerTransformer, OneHotEncoder
from sklearn.pipeline import Pipeline

from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier

from svm_margin_plot import plot_svm_boundary



from sklearn.metrics import make_scorer, precision_score, recall_score, f1_scor
from sklearn.metrics import PrecisionRecallDisplay, roc_curve, average_precisic
from sklearn.metrics import RocCurveDisplay, roc_auc_score, auc
from sklearn.metrics import confusion_matrix, classification_report, ConfusionM


from yellowbrick.regressor import ResidualsPlot, PredictionError

import warnings
warnings.filterwarnings("ignore")
```

In [391…
```
df0 = pd.read_excel('Raisin_Dataset.xlsx')
df = df0.copy()
```

In [3]:
```
df.shape
```

Out[3]: (900, 8)

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

Out[4]:

	Area	MajorAxisLength	MinorAxisLength	Eccentricity	ConvexArea	Extent	Peri
0	87524	442.246011	253.291155	0.819738	90546	0.758651	11
1	75166	406.690687	243.032436	0.801805	78789	0.684130	11
2	90856	442.267048	266.328318	0.798354	93717	0.637613	12
3	45928	286.540559	208.760042	0.684989	47336	0.699599	8
4	79408	352.190770	290.827533	0.564011	81463	0.792772	10

In [5]:
```
df.tail()
```

Out[5]:

	Area	MajorAxisLength	MinorAxisLength	Eccentricity	ConvexArea	Extent	P
895	83248	430.077308	247.838695	0.817263	85839	0.668793	
896	87350	440.735698	259.293149	0.808629	90899	0.636476	
897	99657	431.706981	298.837323	0.721684	106264	0.741099	
898	93523	476.344094	254.176054	0.845739	97653	0.658798	
899	85609	512.081774	215.271976	0.907345	89197	0.632020	

In [6]:
```
df.sample(5)
```

Out[6]:

	Area	MajorAxisLength	MinorAxisLength	Eccentricity	ConvexArea	Extent
592	128574	601.800235	276.248155	0.888418	135975	0.674002
883	171264	609.643147	359.224572	0.807960	174156	0.771738
558	87857	380.776487	300.908648	0.612784	90968	0.728499
34	46427	253.842028	235.906824	0.369212	48275	0.684219
850	104468	542.178172	250.063079	0.887286	108119	0.646933

Exploratory Data Analysis (EDA)

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 900 entries, 0 to 899
Data columns (total 8 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   Area             900 non-null    int64  
 1   MajorAxisLength  900 non-null    float64
 2   MinorAxisLength  900 non-null    float64
 3   Eccentricity     900 non-null    float64
 4   ConvexArea       900 non-null    int64  
 5   Extent           900 non-null    float64
 6   Perimeter        900 non-null    float64
 7   Class            900 non-null    object 
dtypes: float64(5), int64(2), object(1)
memory usage: 56.4+ KB
```

In [8]:
```
df.describe().T
```

Out[8]:

	count	mean	std	min	25%	
Area	900.0	87804.127778	39002.111390	25387.000000	59348.000000	789
MajorAxisLength	900.0	430.929950	116.035121	225.629541	345.442898	4
MinorAxisLength	900.0	254.488133	49.988902	143.710872	219.111126	2
Eccentricity	900.0	0.781542	0.090318	0.348730	0.741766	
ConvexArea	900.0	91186.090000	40769.290132	26139.000000	61513.250000	816
Extent	900.0	0.699508	0.053468	0.379856	0.670869	
Perimeter	900.0	1165.906636	273.764315	619.074000	966.410750	11

In [9]:
```
df.duplicated().sum()
```

Out[9]:
```
0
```

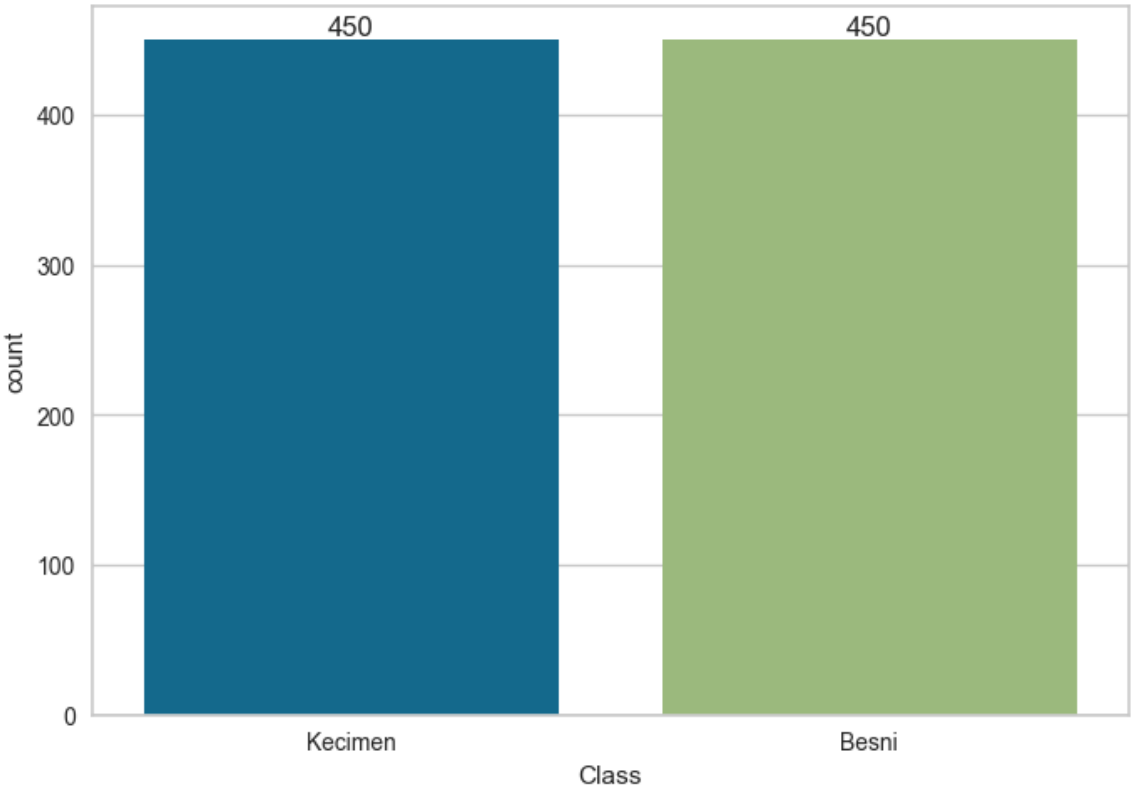
In [10]:
```
df.isnull().sum().sum()
```

Out[10]:
```
0
```

In [11]:
```
df.columns
```

Out[11]:
```
Index(['Area', 'MajorAxisLength', 'MinorAxisLength', 'Eccentricity',
       'ConvexArea', 'Extent', 'Perimeter', 'Class'],
      dtype='object')
```

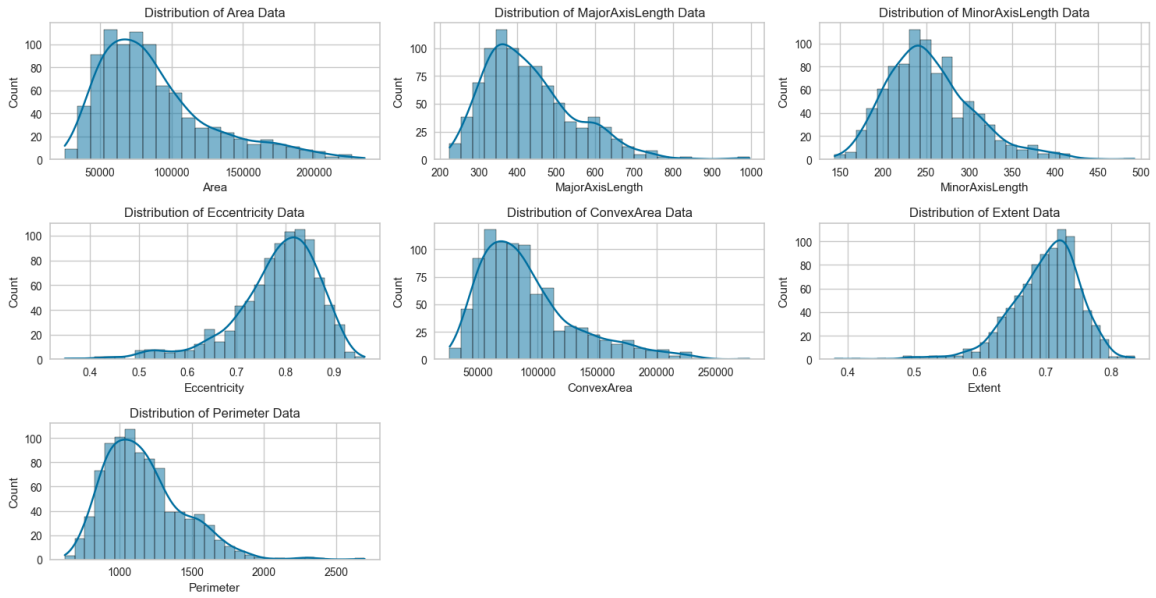
In [12]:
```
ax = sns.countplot(x="Class", data=df)
ax.bar_label(ax.containers[0]);
```



Our data is a balance data.

Features Summary

In [13]:
```
plt.figure(figsize=(15,10))
for i,col in enumerate(df.iloc[:, :-1],1):
    plt.subplot(4,3,i)
    plt.title(f"Distribution of {col} Data")
    sns.histplot(df[col],kde=True)
    plt.tight_layout()
    plt.plot()
```

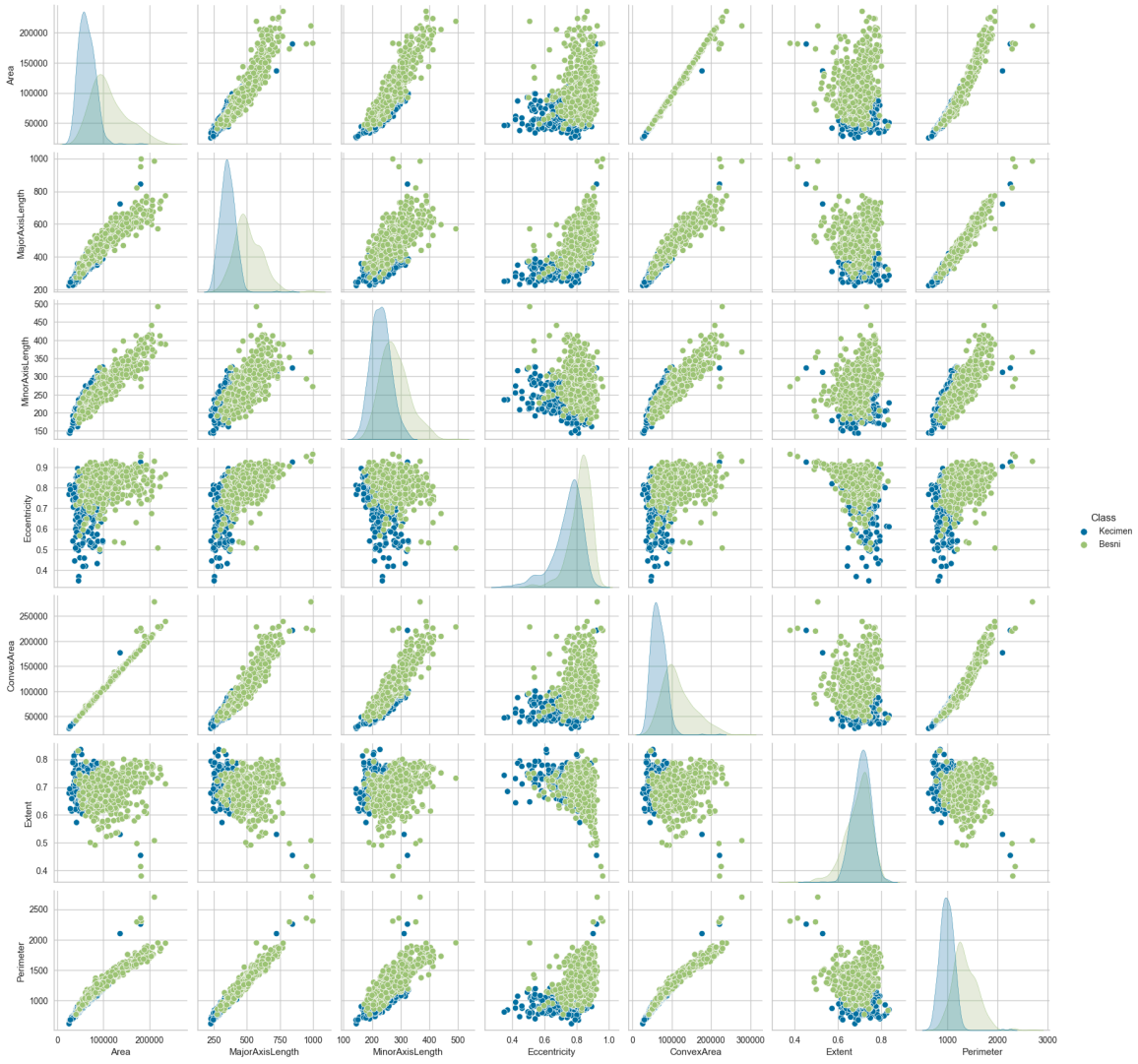


In [14]:
```
num_cols= df.select_dtypes('number').columns

skew_limit = 0.75                # define a limit above which we will log transf
skew_vals = df[num_cols].skew()


# Showing the skewed columns
skew_cols = (skew_vals
             .sort_values(ascending=False)
             .to_frame()
             .rename(columns={0:'Skew'})
             .query('abs(Skew) > {}'.format(skew_limit)))
skew_cols
```

Out[14]:

	Skew
ConvexArea	1.242904
Area	1.175237
Perimeter	1.017761
MajorAxisLength	0.989544
MinorAxisLength	0.800049
Extent	-1.151505
Eccentricity	-1.327503

In [15]:
```
sns.pairplot(df, hue= "Class");
```



In [16]:
```
%matplotlib inline
#%matplotlib notebook
from mpl_toolkits.mplot3d import Axes3D
fig = px.scatter_3d(df, x='Perimeter', y='Area', z='Extent', color='Class', opa
fig.show()
```

Encoding Class

In [392…
```
df['Class'] = OneHotEncoder(drop='first').fit_transform(
    df[['Class']]).toarray().astype(int)
```

Kecimen = 1 , besni = 0

In [359…
```
df.head()
```

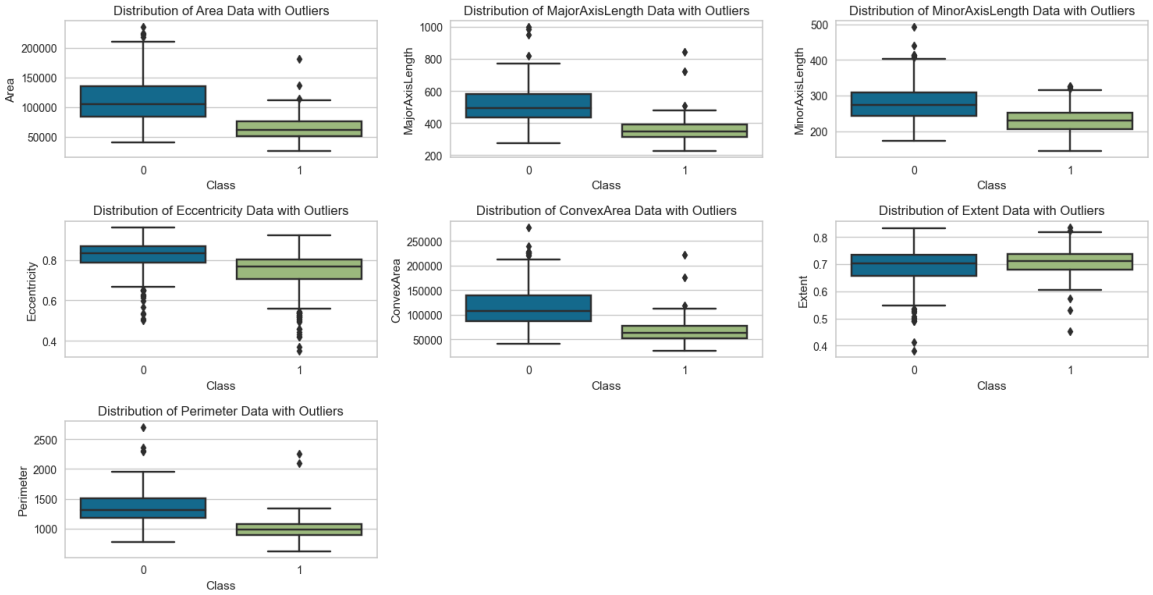
Out[359…

	Area	MajorAxisLength	MinorAxisLength	Eccentricity	ConvexArea	Extent	Peri
0	87524	442.246011	253.291155	0.819738	90546	0.758651	11
1	75166	406.690687	243.032436	0.801805	78789	0.684130	11
2	90856	442.267048	266.328318	0.798354	93717	0.637613	12
3	45928	286.540559	208.760042	0.684989	47336	0.699599	8
4	79408	352.190770	290.827533	0.564011	81463	0.792772	10

Detection of Outliers

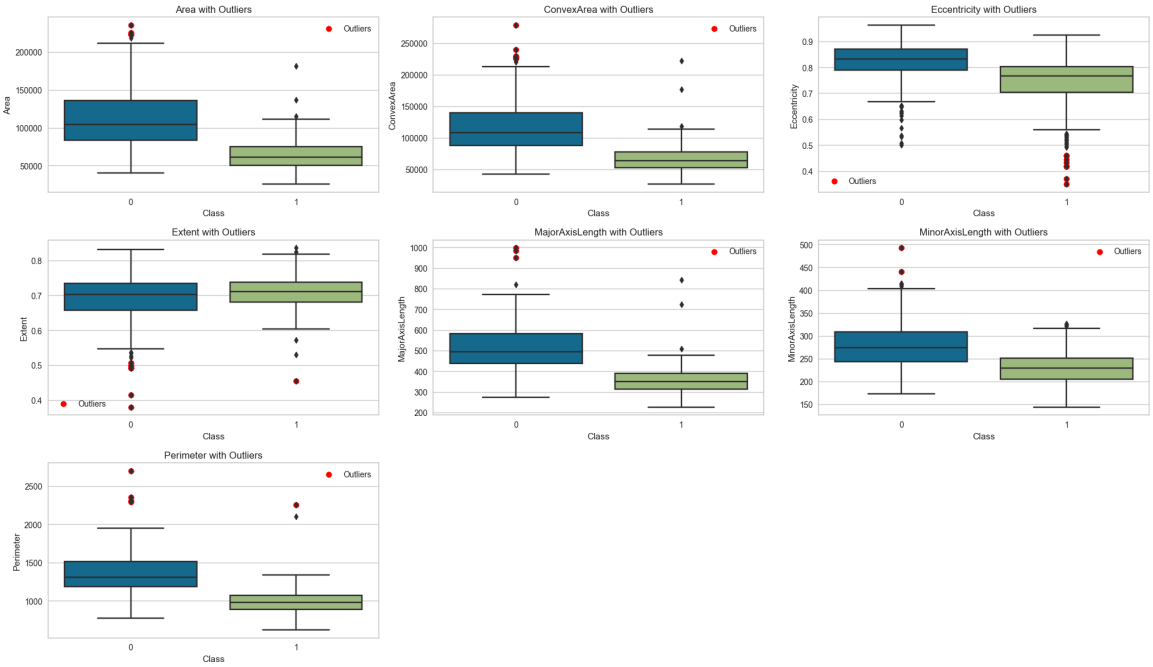
In [19]:
```
def plot_feature_outliers(df, hue_column):
    plt.figure(figsize=(15, 10))
    for i, col in enumerate(df.columns[:-1], 1):
        plt.subplot(4, 3, i)
        plt.title(f"Distribution of {col} Data with Outliers")
        sns.boxplot(x=hue_column, y=col, data=df)
        plt.tight_layout()
    plt.show()
```

In [20]:
```
plot_feature_outliers(df, 'Class')
```



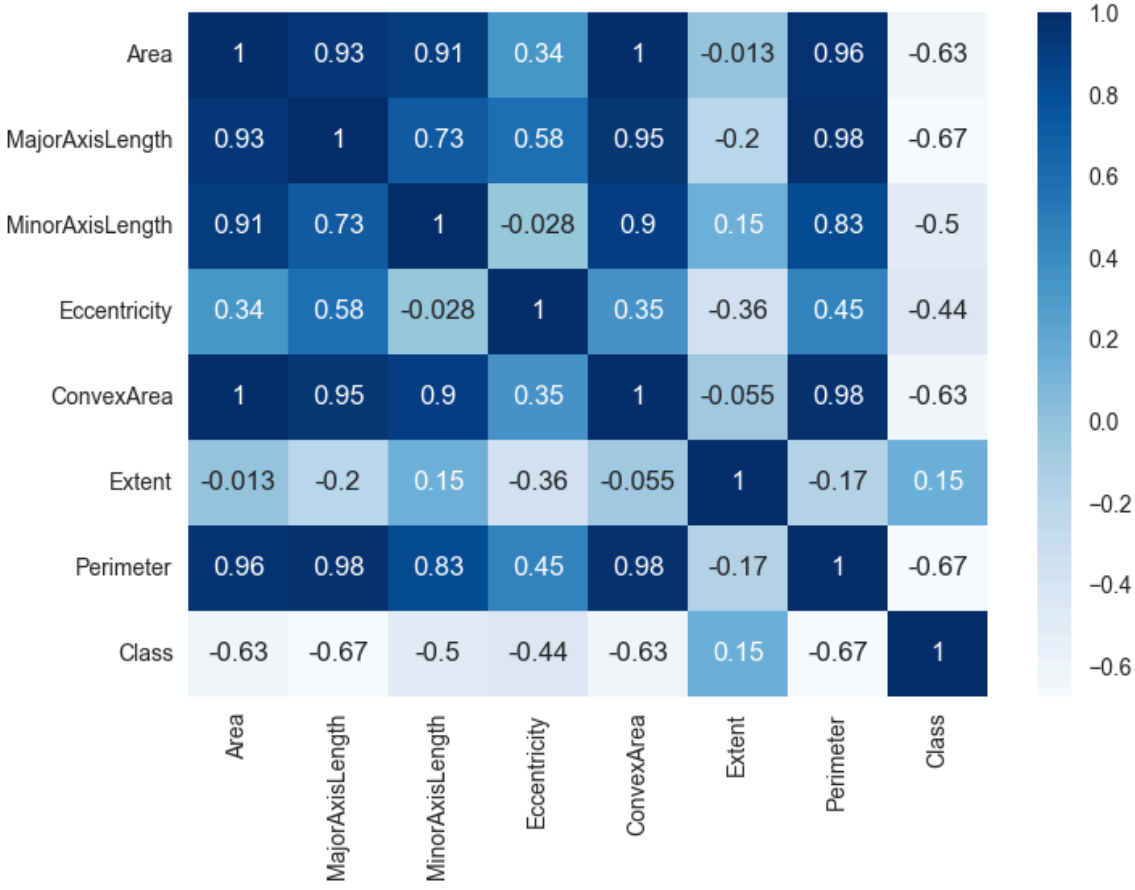
In [21]:
```
def detect_and_plot_outliers(df, target_column):
    features = df.columns.difference([target_column])

    # Create subplots
    plt.figure(figsize=(20, 15))

    for i, col in enumerate(features, 1):
        Q1 = df[col].quantile(0.25)
        Q3 = df[col].quantile(0.75)
        IQR = Q3 - Q1
        lower_bound = Q1 - 2.5 * IQR
        upper_bound = Q3 + 2.5 * IQR

        # Detect outliers
        outliers = df[(df[col] < lower_bound) | (df[col] > upper_bound)]

        # Plotting
        plt.subplot(4, 3, i)
        sns.boxplot(x=target_column, y=col, data=df)
        plt.scatter(outliers[target_column], outliers[col], color='red', label=
        plt.title(f"{col} with Outliers")
        plt.legend()
        plt.tight_layout()

    plt.show()
```

```
detect_and_plot_outliers(df, 'Class')
```



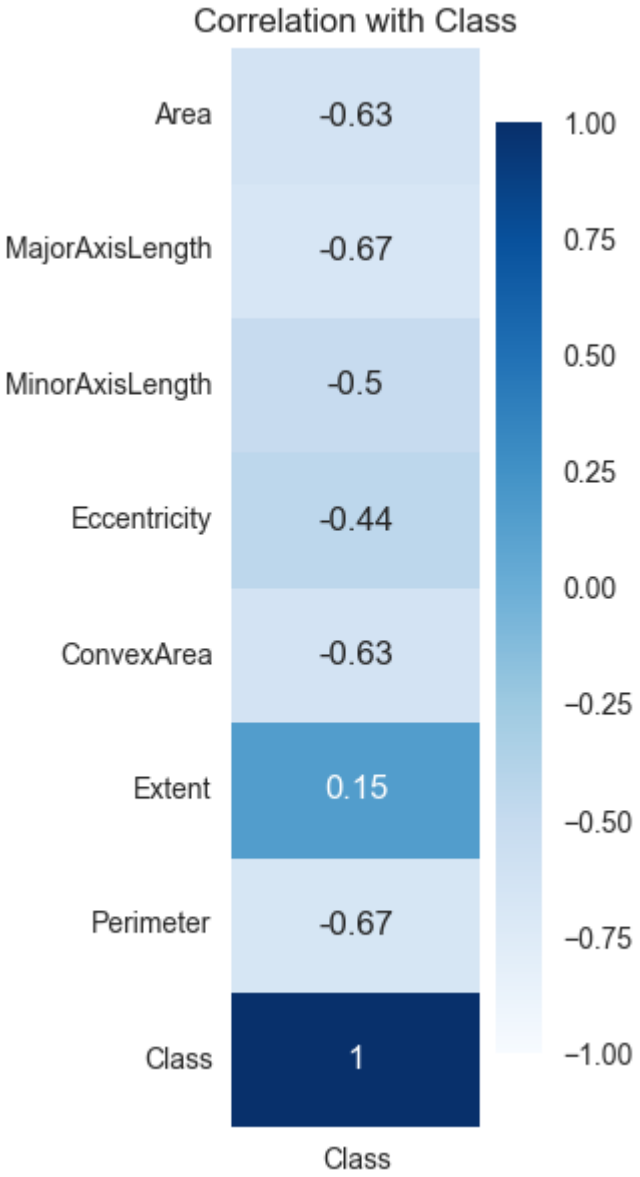
Correlation

In [22]:
```
sns.heatmap(df.corr(), annot=True, cmap="Blues");
```



In [23]:
```
def plot_target_correlation_heatmap(df, target_variable):
    df_numeric = df.select_dtypes(include=[np.number])
```

```
    df_corr_target = df_numeric.corr()

    plt.figure(figsize=(2, 7))
    sns.heatmap(df_corr_target[[target_variable]], annot=True, vmin=-1, vmax=1,
    plt.title(f'Correlation with {target_variable}')
    plt.show()
plot_target_correlation_heatmap(df, 'Class')
```



Multicollinearity

In [24]:
```
df.corr()[(df.corr() >= 0.9) & (df.corr() < 1)].any().any()
```

Out[24]: True

In [25]:
```
df.corr()[(df.corr() <= -0.9) & (df.corr() > -1)].any().any()
```

Out[25]: False

In [26]:
```
def color_correlation1(val):
    """
    Takes a scalar and returns a string with
```

```
    the css property in a variety of color scales
    for different correlations.
    """
    if val >= 0.6 and val < 0.99999 or val <= -0.6 and val > -0.99999:
        color = 'red'
    elif val < 0.6 and val >= 0.3 or val > -0.6 and val <= -0.3:
        color = 'blue'
    elif val == 1:
        color = 'green'
    else:
        color = 'black'
    return 'color: %s' % color

numeric_df = df.select_dtypes(include=[np.number])

numeric_df.corr().style.applymap(color_correlation1)
```

Out[26]:

	Area	MajorAxisLength	MinorAxisLength	Eccentricity	ConvexA
Area	1.000000	0.932774	0.906650	0.336107	0.995
MajorAxisLength	0.932774	1.000000	0.728030	0.583608	0.945
MinorAxisLength	0.906650	0.728030	1.000000	-0.027683	0.895
Eccentricity	0.336107	0.583608	-0.027683	1.000000	0.348
ConvexArea	0.995920	0.945031	0.895651	0.348210	1.000
Extent	-0.013499	-0.203866	0.145322	-0.361061	-0.054
Perimeter	0.961352	0.977978	0.827417	0.447845	0.976
Class	-0.625715	-0.673194	-0.503102	-0.438500	-0.625

In [27]:
```
df.corr()['Class'].sort_values(key=abs, ascending=False)
```

Out[27]:
```
Class             1.000000
MajorAxisLength   -0.673194
Perimeter         -0.665981
Area              -0.625715
ConvexArea        -0.625567
MinorAxisLength   -0.503102
Eccentricity      -0.438500
Extent             0.154689
Name: Class, dtype: float64
```

In [28]:
```
df.columns
```

Out[28]:
```
Index(['Area', 'MajorAxisLength', 'MinorAxisLength', 'Eccentricity',
       'ConvexArea', 'Extent', 'Perimeter', 'Class'],
      dtype='object')
```

In [393…
```
X = df.drop("Class", axis=1)
y = df['Class']
```

Models

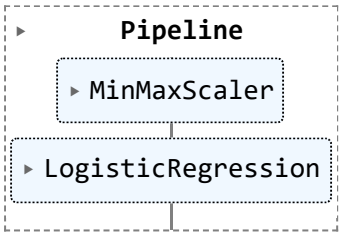
Train | Test Split

In [394…
```
X_train, X_test, y_train, y_test = train_test_split(X,
                                                    y,
                                                    test_size=0.2,
                                                    random_state=10)
```

Logistic Regression Model

In [31]:
```
operations = [("scaler", MinMaxScaler()), ("logistic", LogisticRegression())]

pipe_model = Pipeline(steps=operations)

pipe_model.fit(X_train, y_train)
```

Out[31]:



In [32]:
```
pipe_model["logistic"].coef_
```

Out[32]:
```
array([[-3.16071255, -3.44560824, -1.16070479, -3.19390504, -2.72475541,
         1.32536773, -3.06048935]])
```

In [33]:
```
pipe_model.named_steps['scaler'].get_feature_names_out()
```

Out[33]:
```
array(['Area', 'MajorAxisLength', 'MinorAxisLength', 'Eccentricity',
       'ConvexArea', 'Extent', 'Perimeter'], dtype=object)
```

In [34]:
```
pipe_model["logistic"].intercept_
```

Out[34]:
```
array([4.82289439])
```

X_test + y_yest + y_pred + y_pred_proba

In [35]:
```
y_pred = pipe_model.predict(X_test)
y_pred
```

Out[35]:
```
array([0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 0, 0, 1,
       1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 1,
       1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0,
       1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 1, 0, 1,
       0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0,
       1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1,
       1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0,
       1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1,
       1, 1, 0, 1])
```

In [36]:
```
y_pred_proba = pipe_model.predict_proba(X_test)
y_pred_proba
```

Out[36]:
```
array([[5.03398466e-01, 4.96601534e-01],
       [2.35036378e-01, 7.64963622e-01],
       [2.45416416e-01, 7.54583584e-01],
       [3.86044240e-01, 6.13955760e-01],
       [2.02524475e-01, 7.97475525e-01],
       [4.87037933e-01, 5.12962067e-01],
       [1.07866147e-01, 8.92133853e-01],
       [6.79569595e-01, 3.20430405e-01],
       [1.99529825e-01, 8.00470175e-01],
       [4.59592985e-01, 5.40407015e-01],
       [8.15770018e-01, 1.84229982e-01],
       [2.02436917e-01, 7.97563083e-01],
       [9.91752340e-01, 8.24766004e-03],
       [4.51236542e-02, 9.54876346e-01],
       [1.72699040e-01, 8.27300960e-01],
       [9.95280841e-01, 4.71915867e-03],
       [3.06134762e-01, 6.93865238e-01],
       [2.57758962e-01, 7.42241038e-01],
       [1.11445500e-01, 8.88554500e-01],
       [6.31517738e-01, 3.68482262e-01],
       [8.30903754e-01, 1.69096246e-01],
       [3.08213135e-01, 6.91786865e-01],
       [3.18193385e-01, 6.81806615e-01],
       [7.96144542e-01, 2.03855458e-01],
       [4.55751960e-01, 5.44248040e-01],
       [2.32645874e-01, 7.67354126e-01],
       [5.62869800e-01, 4.37130200e-01],
       [2.86540300e-02, 9.71345970e-01],
       [9.18748832e-01, 8.12511680e-02],
       [4.71927610e-01, 5.28072390e-01],
       [8.56192983e-01, 1.43807017e-01],
       [9.79698666e-02, 9.02030133e-01],
       [3.17260011e-01, 6.82739989e-01],
       [2.04441238e-01, 7.95558762e-01],
       [4.33607892e-01, 5.66392108e-01],
       [9.98112530e-01, 1.88747042e-03],
       [2.12968939e-01, 7.87031061e-01],
       [1.77987361e-01, 8.22012639e-01],
       [8.02105602e-02, 9.19789440e-01],
       [8.34031541e-01, 1.65968459e-01],
       [2.48238939e-01, 7.51761061e-01],
       [9.79614873e-01, 2.03851267e-02],
       [9.64497992e-01, 3.55020084e-02],
       [1.19559784e-01, 8.80440216e-01],
       [2.11739957e-01, 7.88260043e-01],
       [8.05095140e-01, 1.94904860e-01],
       [1.37175377e-01, 8.62824623e-01],
       [4.23950421e-01, 5.76049579e-01],
       [7.49251466e-01, 2.50748534e-01],
       [5.40432822e-01, 4.59567178e-01],
       [2.60643163e-01, 7.39356837e-01],
       [3.57961101e-01, 6.42038899e-01],
       [8.03428971e-01, 1.96571029e-01],
       [7.57492402e-01, 2.42507598e-01],
       [5.47892928e-01, 4.52107072e-01],
       [3.93411873e-02, 9.60658813e-01],
       [3.53905349e-01, 6.46094651e-01],
       [5.77726225e-02, 9.42227378e-01],
       [5.72273331e-01, 4.27726669e-01],
       [6.88732717e-01, 3.11267283e-01],
```

```
[9.98952591e-01, 1.04740884e-03],
[4.54926732e-01, 5.45073268e-01],
[4.12729449e-01, 5.87270551e-01],
[1.31563879e-01, 8.68436121e-01],
[4.99525527e-01, 5.00474473e-01],
[8.78907815e-01, 1.21092185e-01],
[2.24254129e-01, 7.75745871e-01],
[2.75622669e-01, 7.24377331e-01],
[8.55692326e-01, 1.44307674e-01],
[9.86738400e-01, 1.32615997e-02],
[9.82682632e-01, 1.73173676e-02],
[9.69051760e-01, 3.09482402e-02],
[1.77927979e-01, 8.22072021e-01],
[4.36148954e-01, 5.63851046e-01],
[3.88121273e-01, 6.11878727e-01],
[2.39951430e-01, 7.60048570e-01],
[5.52959536e-01, 4.47040464e-01],
[2.68772181e-01, 7.31227819e-01],
[3.14080889e-01, 6.85919111e-01],
[9.16777079e-01, 8.32229206e-02],
[5.86132370e-01, 4.13867630e-01],
[4.23962119e-02, 9.57603788e-01],
[2.36591123e-01, 7.63408877e-01],
[1.90696219e-01, 8.09303781e-01],
[9.02526964e-01, 9.74730363e-02],
[1.38312645e-01, 8.61687355e-01],
[9.02701688e-01, 9.72983115e-02],
[6.48978310e-02, 9.35102169e-01],
[7.14204461e-01, 2.85795539e-01],
[8.86863375e-01, 1.13136625e-01],
[8.68467848e-01, 1.31532152e-01],
[5.94406102e-01, 4.05593898e-01],
[2.30049836e-01, 7.69950164e-01],
[1.79888978e-01, 8.20111022e-01],
[9.72005422e-01, 2.79945784e-02],
[1.30233024e-02, 9.86976698e-01],
[8.30194362e-03, 9.91698056e-01],
[5.35662318e-01, 4.64337682e-01],
[6.24545733e-01, 3.75454267e-01],
[9.91279912e-01, 8.72008760e-03],
[1.15679071e-01, 8.84320929e-01],
[1.26151680e-01, 8.73848320e-01],
[1.24978226e-01, 8.75021774e-01],
[2.00860872e-01, 7.99139128e-01],
[4.14725454e-01, 5.85274546e-01],
[9.84929643e-01, 1.50703573e-02],
[1.66214843e-01, 8.33785157e-01],
[9.32707783e-01, 6.72922171e-02],
[2.25113839e-01, 7.74886161e-01],
[7.96319717e-01, 2.03680283e-01],
[4.64219467e-01, 5.35780533e-01],
[2.90425543e-01, 7.09574457e-01],
[3.12215530e-01, 6.87784470e-01],
[1.40153650e-01, 8.59846350e-01],
[1.79540812e-01, 8.20459188e-01],
[5.04783172e-01, 4.95216828e-01],
[9.35894955e-01, 6.41050450e-02],
[7.80947011e-01, 2.19052989e-01],
[3.64884183e-01, 6.35115817e-01],
[1.38612841e-02, 9.86138716e-01],
```

```
[5.48376388e-01, 4.51623612e-01],
[7.84259877e-01, 2.15740123e-01],
[9.37643224e-01, 6.23567756e-02],
[1.53211740e-01, 8.46788260e-01],
[9.99749695e-01, 2.50304611e-04],
[9.46340024e-01, 5.36599759e-02],
[4.98280993e-02, 9.50171901e-01],
[9.35277973e-01, 6.47220269e-02],
[7.24511372e-01, 2.75488628e-01],
[5.52579984e-01, 4.47420016e-01],
[1.27693562e-01, 8.72306438e-01],
[4.55734191e-01, 5.44265809e-01],
[1.23349650e-01, 8.76650350e-01],
[1.13871053e-01, 8.86128947e-01],
[2.73798822e-01, 7.26201178e-01],
[6.85993341e-02, 9.31400666e-01],
[1.06109291e-01, 8.93890709e-01],
[9.86873756e-01, 1.31262442e-02],
[1.24263730e-01, 8.75736270e-01],
[9.89069073e-01, 1.09309266e-02],
[2.06593112e-01, 7.93406888e-01],
[7.40992205e-01, 2.59007795e-01],
[4.84474944e-01, 5.15525056e-01],
[9.50762185e-01, 4.92378155e-02],
[1.90982104e-01, 8.09017896e-01],
[2.74531473e-01, 7.25468527e-01],
[2.90345897e-01, 7.09654103e-01],
[7.12171898e-01, 2.87828102e-01],
[9.33582505e-02, 9.06641749e-01],
[6.75072564e-01, 3.24927436e-01],
[6.94019734e-01, 3.05980266e-01],
[7.62404280e-01, 2.37595720e-01],
[5.02646664e-01, 4.97353336e-01],
[8.74834459e-01, 1.25165541e-01],
[3.70262423e-01, 6.29737577e-01],
[7.19303723e-01, 2.80696277e-01],
[2.45723077e-01, 7.54276923e-01],
[7.37734384e-01, 2.62265616e-01],
[2.99277380e-01, 7.00722620e-01],
[3.67540686e-01, 6.32459314e-01],
[6.04851036e-01, 3.95148964e-01],
[1.28073910e-01, 8.71926090e-01],
[8.75701733e-01, 1.24298267e-01],
[6.44092307e-01, 3.55907693e-01],
[9.88850844e-01, 1.11491556e-02],
[6.16005655e-01, 3.83994345e-01],
[2.19513840e-01, 7.80486160e-01],
[9.91476930e-02, 9.00852307e-01],
[7.88349895e-01, 2.11650105e-01],
[9.73890463e-01, 2.61095375e-02],
[4.66518622e-02, 9.53348138e-01],
[3.02573855e-01, 6.97426145e-01],
[9.88524140e-01, 1.14758598e-02],
[1.22767175e-01, 8.77232825e-01],
[9.99558759e-01, 4.41241377e-04],
[3.37797374e-01, 6.62202626e-01],
[1.70595284e-01, 8.29404716e-01],
[1.61116386e-01, 8.38883614e-01],
[8.01710528e-01, 1.98289472e-01],
[1.75085605e-01, 8.24914395e-01]])
```

In [37]:
```
test_data = pd.concat([X_test, y_test], axis=1)
test_data
```

Out[37]:

	Area	MajorAxisLength	MinorAxisLength	Eccentricity	ConvexArea	Extent
437	86141	414.107165	266.907528	0.764573	87883	0.718740
131	70788	362.650770	249.403725	0.725972	71954	0.746158
633	54357	362.594718	192.949367	0.846659	56006	0.699918
195	87302	392.910117	284.179068	0.690568	89605	0.738246
230	54219	344.290832	204.496689	0.804491	56526	0.688495
...	...	...	...	...	...	...
191	87036	384.969903	289.453901	0.659292	88336	0.727896
752	49691	336.678058	189.261667	0.827039	52077	0.724950
94	52243	331.451147	206.759599	0.781583	55774	0.714434
773	106938	498.433852	274.590275	0.834567	110118	0.773187
57	53006	333.381370	206.880708	0.784165	55456	0.691695

180 rows × 8 columns

In [38]:
```
test_data["pred_proba"] = y_pred_proba[:,1]
test_data
```

Out[38]:

	Area	MajorAxisLength	MinorAxisLength	Eccentricity	ConvexArea	Extent
437	86141	414.107165	266.907528	0.764573	87883	0.718740
131	70788	362.650770	249.403725	0.725972	71954	0.746158
633	54357	362.594718	192.949367	0.846659	56006	0.699918
195	87302	392.910117	284.179068	0.690568	89605	0.738246
230	54219	344.290832	204.496689	0.804491	56526	0.688495
...	...	...	...	...	...	...
191	87036	384.969903	289.453901	0.659292	88336	0.727896
752	49691	336.678058	189.261667	0.827039	52077	0.724950
94	52243	331.451147	206.759599	0.781583	55774	0.714434
773	106938	498.433852	274.590275	0.834567	110118	0.773187
57	53006	333.381370	206.880708	0.784165	55456	0.691695

180 rows × 9 columns

In [39]:
```
test_data["pred"] = y_pred
test_data
```

```
test_data[((test_data["Class"] == 0) & (test_data["pred"] == 1)) |
          ((test_data["Class"] == 1) & (test_data["pred"] == 0))]
```

Out[39]:

	Area	MajorAxisLength	MinorAxisLength	Eccentricity	ConvexArea	Extent
437	86141	414.107165	266.907528	0.764573	87883	0.718740
633	54357	362.594718	192.949367	0.846659	56006	0.699918
731	84383	403.909415	271.251509	0.740945	87629	0.674945
546	88928	395.724998	300.227002	0.651468	93821	0.671064
831	48488	275.337717	226.802199	0.566991	50103	0.669594
688	57999	311.022470	243.476121	0.622242	61519	0.656335
885	54502	346.457978	204.081212	0.808096	56464	0.636111
717	58870	355.136018	212.029460	0.802213	60490	0.734956
92	85156	422.279133	260.210623	0.787586	87057	0.702561
271	84057	452.775695	245.219982	0.840641	86710	0.622018
827	61861	345.943650	235.430468	0.732706	67390	0.702280
822	62280	388.582055	205.514392	0.848695	63596	0.791440
872	66938	356.323284	248.674245	0.716206	69880	0.708661
124	96064	411.207053	303.232452	0.675434	99609	0.731331
85	180898	843.956653	323.190569	0.923770	221396	0.454189
193	79748	441.934293	231.717531	0.851518	81718	0.738448
585	61444	371.693826	213.179806	0.819180	64321	0.737473
218	72915	414.718073	229.555770	0.832834	76912	0.679771
223	93559	438.634353	275.582973	0.777991	95750	0.687070
635	70461	376.170843	243.954829	0.761196	72819	0.728279
752	49691	336.678058	189.261667	0.827039	52077	0.724950

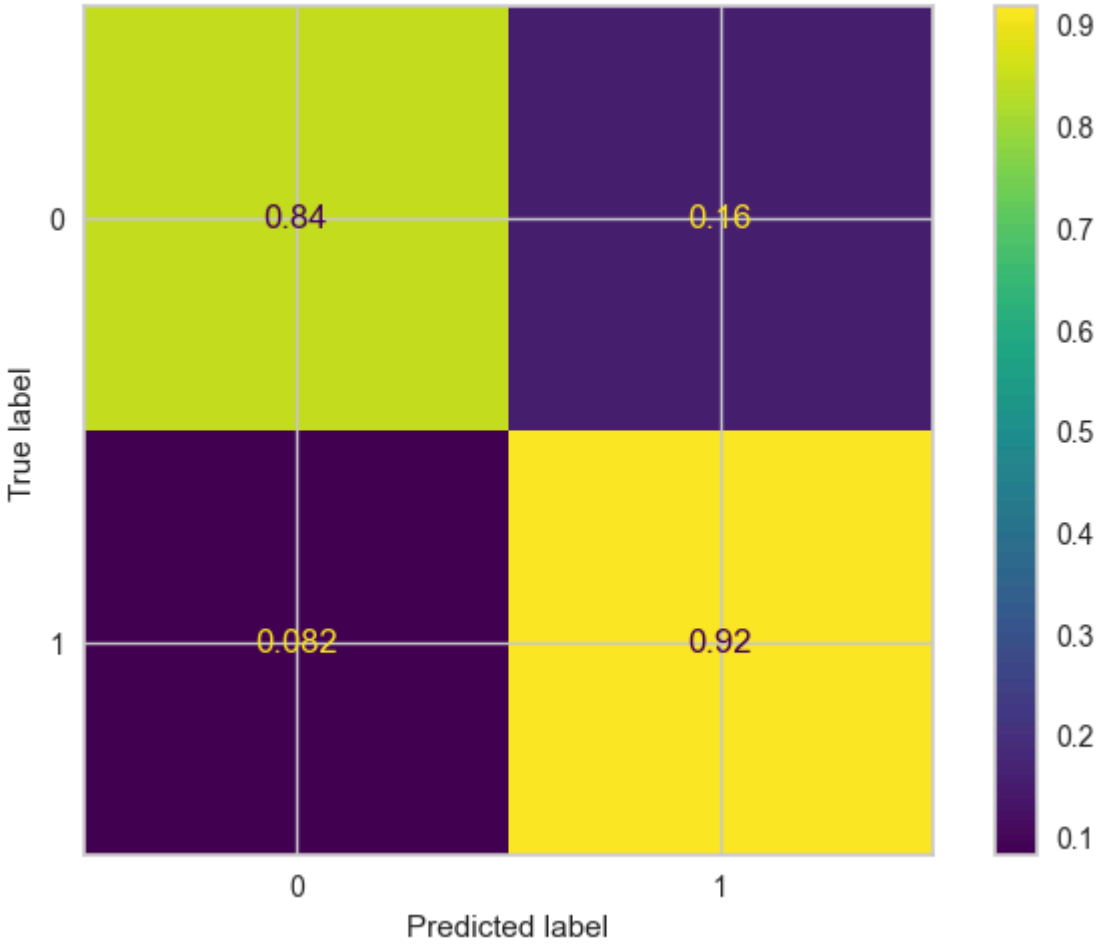
Model Performance

In [40]:
```
confusion_matrix(y_test, y_pred)
```

Out[40]:
```
array([[70, 13],
       [ 8, 89]], dtype=int64)
```

In [41]:
```
log_pipe_matrix = ConfusionMatrixDisplay.from_estimator(pipe_model,
                                                        X_test,
                                                        y_test,
                                                        normalize='true')
log_pipe_matrix
```

Out[41]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x21a2c354150>



In [42]:
```
def eval_metric(model, X_train, y_train, X_test, y_test,i):
    y_train_pred = model.predict(X_train)
    y_pred = model.predict(X_test)
    print(f"{i} Test_Set")
    print(confusion_matrix(y_test, y_pred))
    print(classification_report(y_test, y_pred))
    print()
    print(f"{i} Train_Set")
    print(confusion_matrix(y_train, y_train_pred))
    print(classification_report(y_train, y_train_pred))
```

Cross Validate

In [43]:
```
operations = [("scaler", MinMaxScaler()), ("logistic", LogisticRegression())]
model = Pipeline(steps=operations)

scores = cross_validate(model,
                        X_train,
                        y_train,
                        scoring=['accuracy', 'precision', 'recall', 'f1'],
                        cv=10,
                        return_train_score=True)
df_scores = pd.DataFrame(scores, index=range(1, 11))
df_scores
```

Out[43]:

	fit_time	score_time	test_accuracy	train_accuracy	test_precision	train_precision
1	0.008849	0.018842	0.888889	0.854938	0.885714	0.835329
2	0.015941	0.008103	0.902778	0.854938	0.911765	0.831361
3	0.015955	0.008368	0.819444	0.862654	0.843750	0.839763
4	0.010906	0.013965	0.847222	0.861111	0.815789	0.841317
5	0.013451	0.009647	0.833333	0.859568	0.780488	0.840841
6	0.012129	0.011355	0.888889	0.856481	0.864865	0.835821
7	0.008097	0.016794	0.930556	0.850309	0.916667	0.831832
8	0.008533	0.010619	0.791667	0.868827	0.744186	0.853659
9	0.008081	0.015711	0.791667	0.865741	0.800000	0.844311
10	0.008339	0.014206	0.875000	0.858025	0.846154	0.839879

In [44]:
```
df_scores.mean()[2:]
```

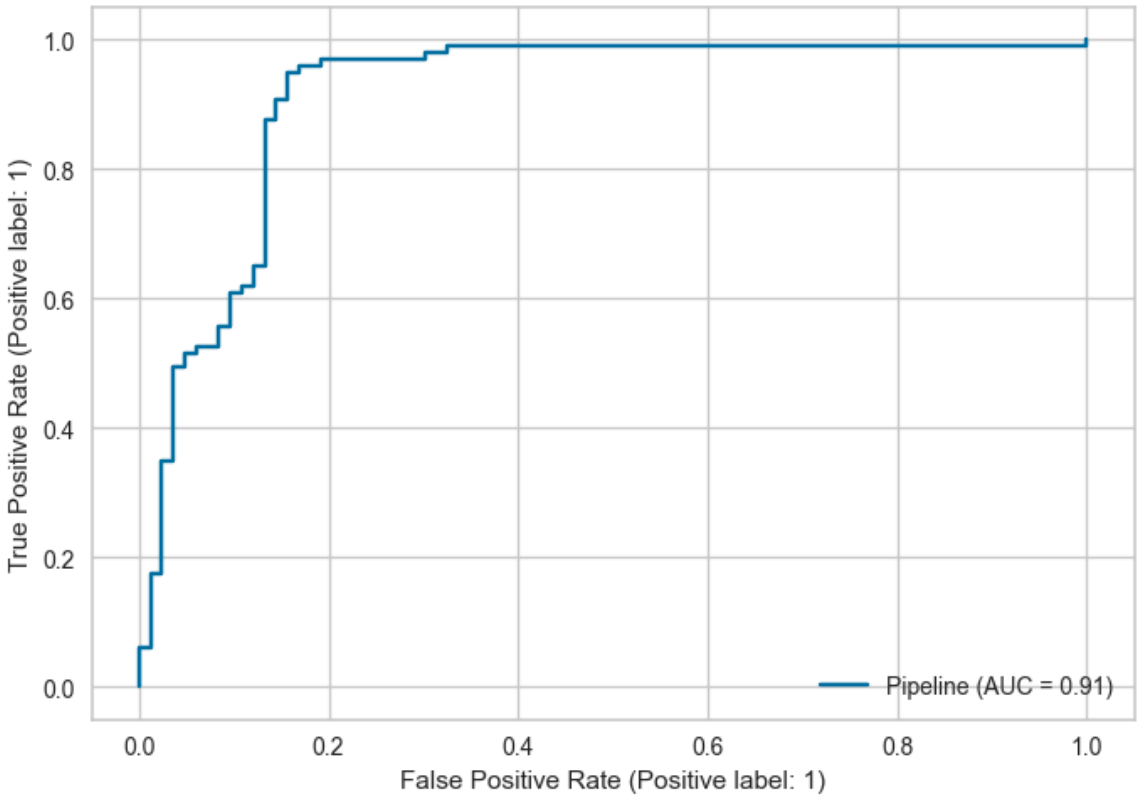
Out[44]:
```
test_accuracy      0.856944
train_accuracy     0.859259
test_precision     0.840938
train_precision    0.839411
test_recall        0.878333
train_recall       0.881651
test_f1            0.857898
train_f1           0.860002
dtype: float64
```

In [45]:
```
eval_metric(pipe_model, X_train, y_train, X_test, y_test, "logistic")
```

```
logistic Test_Set
[[70 13]
 [ 8 89]]
              precision    recall  f1-score   support

           0       0.90      0.84      0.87        83
           1       0.87      0.92      0.89        97

    accuracy                           0.88       180
   macro avg       0.88      0.88      0.88       180
weighted avg       0.88      0.88      0.88       180


logistic Train_Set
[[308  59]
 [ 42 311]]
              precision    recall  f1-score   support

           0       0.88      0.84      0.86       367
           1       0.84      0.88      0.86       353

    accuracy                           0.86       720
   macro avg       0.86      0.86      0.86       720
weighted avg       0.86      0.86      0.86       720
```

In [46]:
```
log_pipe_ROC = RocCurveDisplay.from_estimator(pipe_model, X_test, y_test)
log_pipe_ROC
```

Out[46]: <sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x21a36cb04d0>

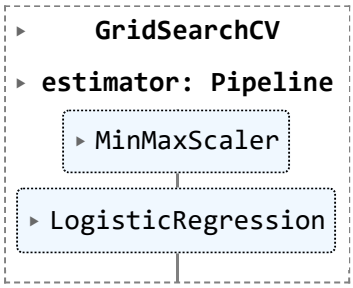


GridSearchCV

In [47]:
```
scaler = MinMaxScaler()
log_model = LogisticRegression()

model = Pipeline([("scaler", scaler), ("log_model", log_model)])
penalty = ["l1", "l2"]
C = np.logspace(-1, 5, 20)
class_weight = ["balanced", None]
solver = ["lbfgs", "liblinear", "sag", "saga"]
param_grid = {
    "log_model__penalty": penalty,
    "log_model__C": [C, 1],
    "log_model__class_weight": class_weight,
    "log_model__solver": solver
}


grid_model = GridSearchCV(
    estimator=model,
    param_grid=param_grid,
    cv=10,
    scoring='accuracy',
    n_jobs=-1)
```

In [48]:
```
grid_model.fit(X_train, y_train)
```

Out[48]:

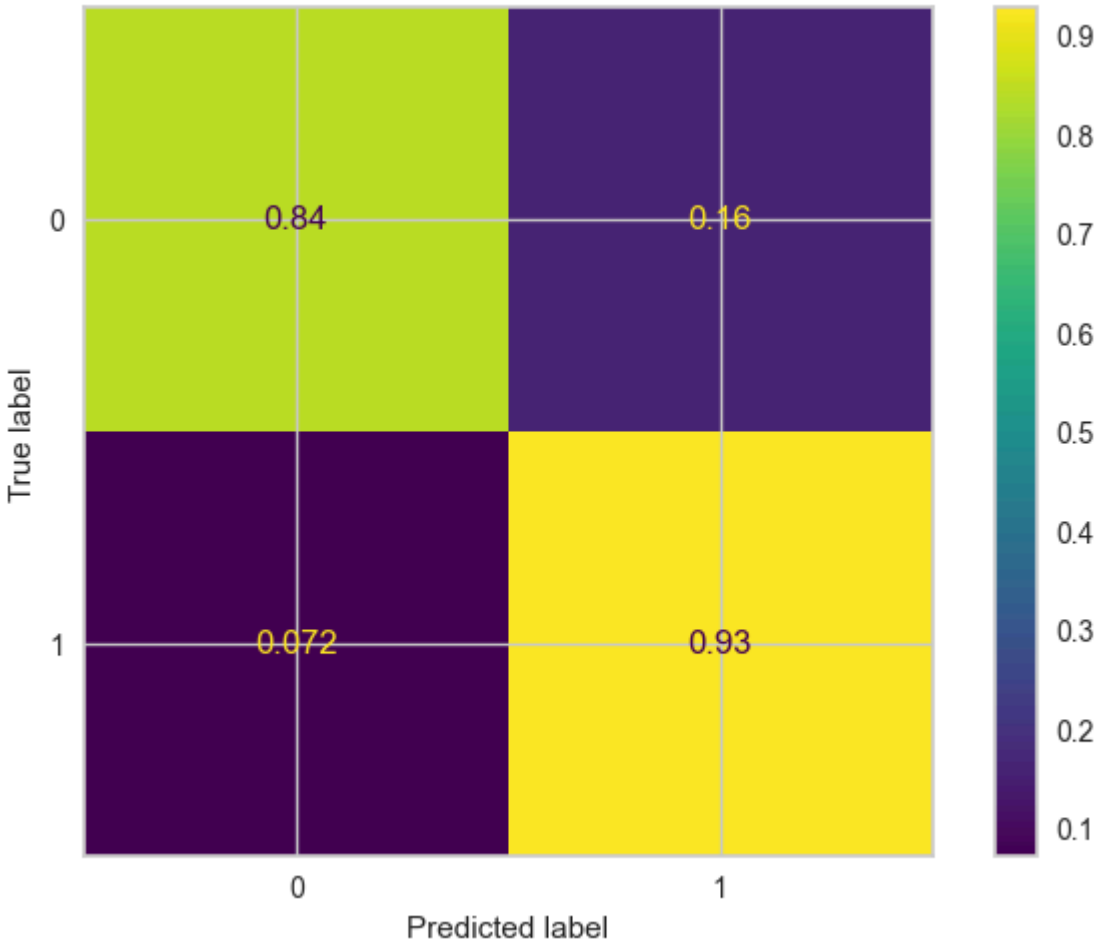


In [49]:
```
grid_model.best_params_
```

Out[49]:
```
{'log_model__C': 1,
 'log_model__class_weight': 'balanced',
 'log_model__penalty': 'l1',
 'log_model__solver': 'liblinear'}
```

In [50]:
```
log_grid_matrix = ConfusionMatrixDisplay.from_estimator(grid_model,
                                                        X_test,
                                                        y_test,
                                                        normalize='true')
log_grid_matrix
```

Out[50]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x21a3780ba50>
```



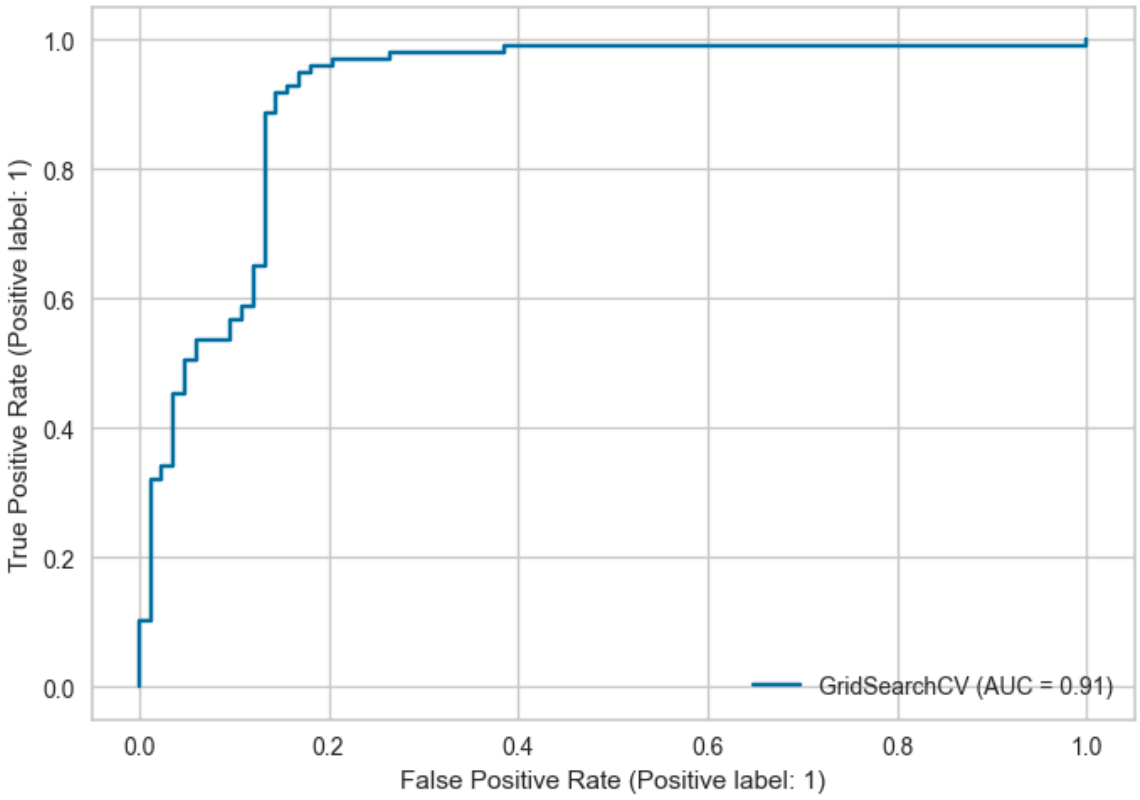
In [51]:
```
eval_metric(grid_model, X_train, y_train, X_test, y_test, "logisticgrid")
```

```
logisticgrid Test_Set
[[70 13]
 [ 7 90]]
              precision    recall  f1-score   support

           0       0.91      0.84      0.88        83
           1       0.87      0.93      0.90        97

    accuracy                           0.89       180
   macro avg       0.89      0.89      0.89       180
weighted avg       0.89      0.89      0.89       180


logisticgrid Train_Set
[[309  58]
 [ 40 313]]
              precision    recall  f1-score   support

           0       0.89      0.84      0.86       367
           1       0.84      0.89      0.86       353

    accuracy                           0.86       720
   macro avg       0.86      0.86      0.86       720
weighted avg       0.86      0.86      0.86       720
```

ROC (Receiver Operating Curve)

In [52]:
```
log_grid_ROC = RocCurveDisplay.from_estimator(grid_model, X_test, y_test)
log_grid_ROC
```

Out[52]: <sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x21a378afe90>



In [53]:
```
df1 = df.copy()
```

Logistic With Transformation

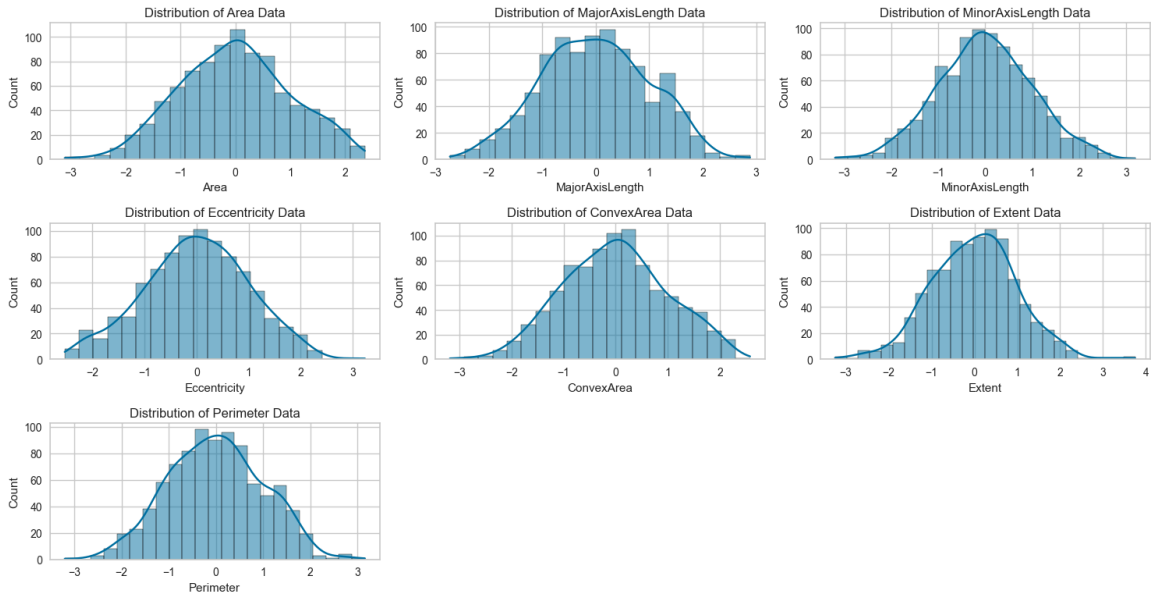
In [54]:
```
# Select the columns of interest
columns_of_interest = [
    'Area', 'MajorAxisLength', 'MinorAxisLength', 'Eccentricity', 'ConvexArea',
    'Extent', 'Perimeter'
]

# Initialize the PowerTransformer
power_transformer = PowerTransformer(method='yeo-johnson')

# Apply the Yeo-Johnson transformation to the selected columns
df1[columns_of_interest] = power_transformer.fit_transform(
    df1[columns_of_interest])
```

In [55]:
```
df1.describe().T
```

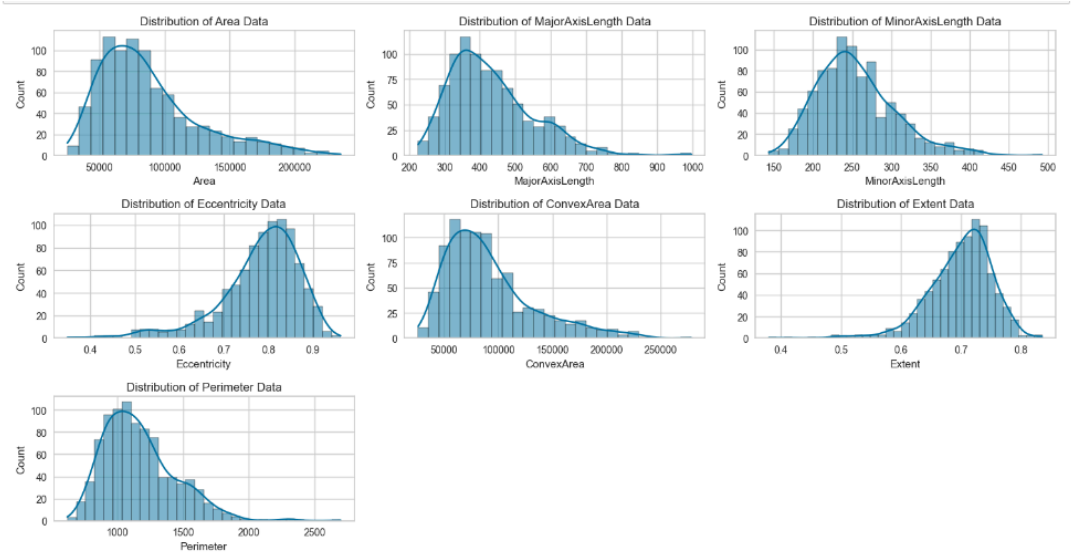
Out[55]:

	count	mean	std	min	25%	50%	75
Area	900.0	-2.210577e-15	1.000556	-3.107110	-0.710225	0.000939	0.6728
MajorAxisLength	900.0	4.357995e-15	1.000556	-2.726264	-0.710980	-0.026095	0.7043
MinorAxisLength	900.0	-7.358065e-15	1.000556	-3.195617	-0.672585	-0.006777	0.6208
Eccentricity	900.0	1.263187e-16	1.000556	-2.535103	-0.662466	0.026363	0.6830
ConvexArea	900.0	4.105358e-15	1.000556	-3.183118	-0.714198	-0.002160	0.6632
Extent	900.0	-4.026409e-16	1.000556	-3.251373	-0.672369	0.027789	0.6489
Perimeter	900.0	5.425389e-14	1.000556	-3.204105	-0.703709	-0.005496	0.6747
Class	900.0	5.000000e-01	0.500278	0.000000	0.000000	0.500000	1.0000

In [56]:
```
plt.figure(figsize=(15,10))
for i,col in enumerate(df1.iloc[:, :-1],1):
    plt.subplot(4,3,i)
    plt.title(f"Distribution of {col} Data")
    sns.histplot(df1[col],kde=True)
    plt.tight_layout()
    plt.plot()
```



no transformation



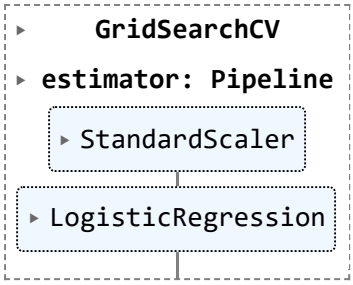
In [57]:
```
X1 = df1.drop("Class", axis=1)
y1 = df1['Class']
```

In [58]:
```
X1_train, X1_test, y1_train, y1_test = train_test_split(X1,
                                                        y1,
                                                        test_size=0.2,
                                                        random_state=10)
```

In [59]:
```
scaler = StandardScaler()
log_model = LogisticRegression()

# pipeline for logistic regression
model = Pipeline([("scaler", scaler), ("log_model", log_model)])
# l1: Lasso, l2: Ridge
penalty = ["l1", "l2"]
C = np.logspace(-1, 5, 20)
class_weight = ["balanced", None]
solver = ["lbfgs", "liblinear", "sag", "saga"]
param_grid = {
    "log_model__penalty": penalty,
    "log_model__C": [C, 1],
    "log_model__class_weight": class_weight,
    "log_model__solver": solver
}


grid_model = GridSearchCV(
    estimator=model,
    param_grid=param_grid,
    cv=10,
    scoring='accuracy',
    n_jobs=-1)
```

In [60]:
```
grid_model.fit(X1_train, y1_train)
```

Out[60]:



In [61]:
```
grid_model.best_params_
```

Out[61]:
```
{'log_model__C': 1,
 'log_model__class_weight': 'balanced',
 'log_model__penalty': 'l1',
 'log_model__solver': 'saga'}
```

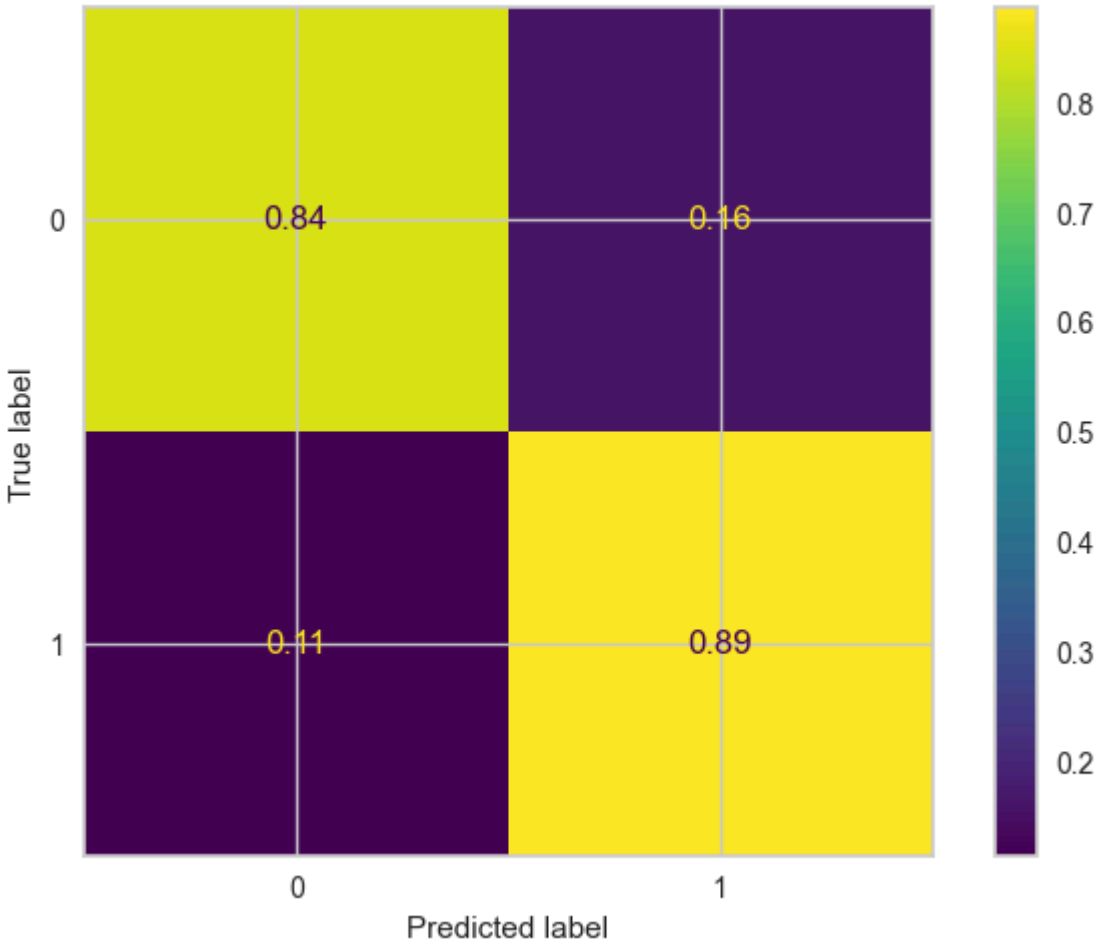
In [62]:
```
grid_model.best_index_
```

Out[62]:
```
19
```

Model Performance

In [63]:
```
log_transform_matrix = ConfusionMatrixDisplay.from_estimator(grid_model,
                                                             X1_test,
                                                             y1_test,
                                                             normalize='true')
log_transform_matrix
```

Out[63]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x21a39327fd
0>
```



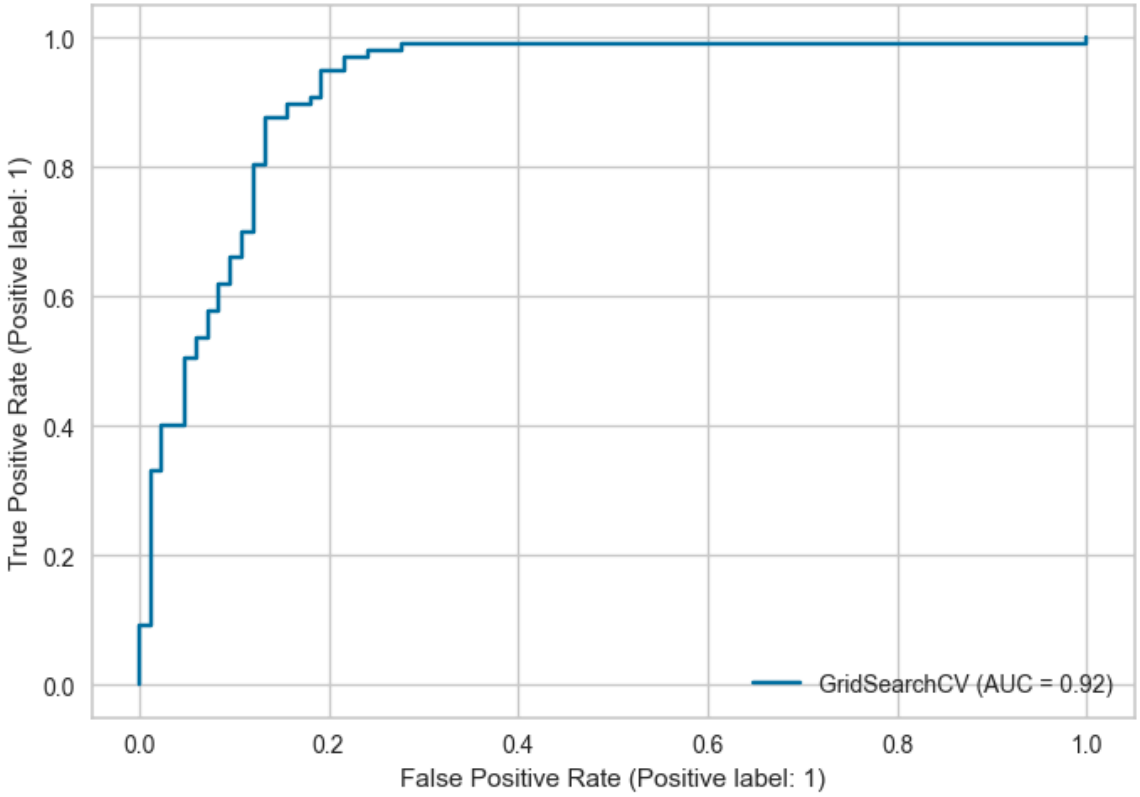
In [64]:
```
eval_metric(grid_model, X1_train, y1_train, X1_test, y1_test,
            "logisticgrid_transform")
```

```
logisticgrid_transform Test_Set
[[70 13]
 [11 86]]
              precision    recall  f1-score   support

           0       0.86      0.84      0.85        83
           1       0.87      0.89      0.88        97

    accuracy                           0.87       180
   macro avg       0.87      0.86      0.87       180
weighted avg       0.87      0.87      0.87       180


logisticgrid_transform Train_Set
[[319  48]
 [ 48 305]]
              precision    recall  f1-score   support

           0       0.87      0.87      0.87       367
           1       0.86      0.86      0.86       353

    accuracy                           0.87       720
   macro avg       0.87      0.87      0.87       720
weighted avg       0.87      0.87      0.87       720
```

In [65]:
```
log_transform_ROC = RocCurveDisplay.from_estimator(grid_model, X1_test,
                                                   y1_test)
```

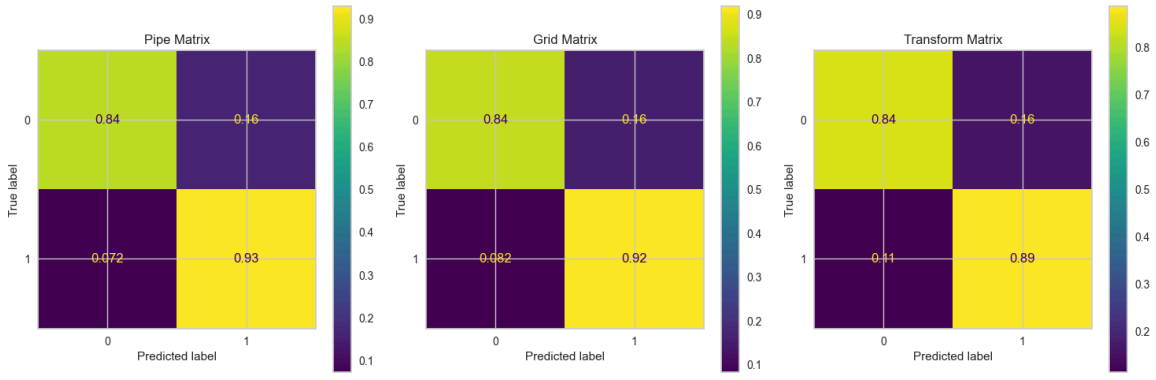
```
log_transform_ROC
```

Out[65]: <sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x21a3936be90>



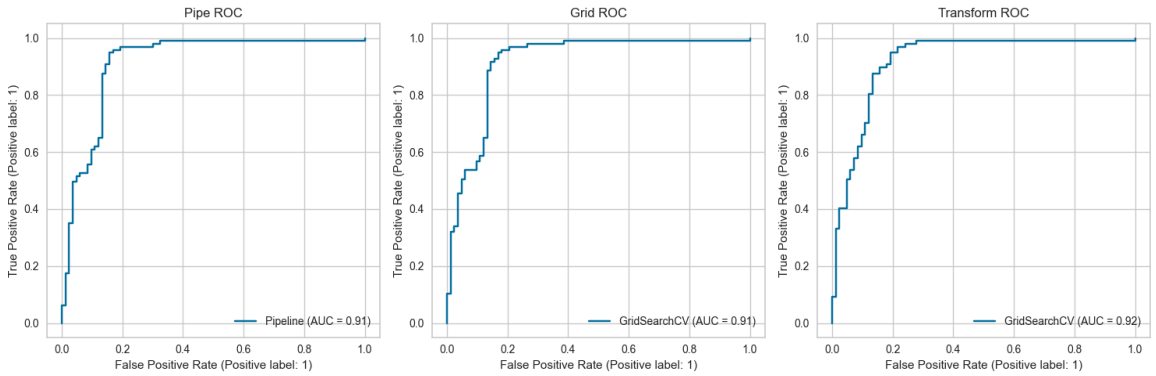
Compare Logistic Regression Models

In [66]:
```
fig, ax = plt.subplots(1, 3, figsize=(15, 5))

log_pipe_matrix.plot(ax=ax[1])
ax[0].set_title("Pipe Matrix")
log_grid_matrix.plot(ax=ax[0])
ax[1].set_title("Grid Matrix")
log_transform_matrix.plot(ax=ax[2])
ax[2].set_title("Transform Matrix")

plt.tight_layout()
plt.show()
```



In [67]:
```
fig, ax = plt.subplots(1, 3, figsize=(15, 5))
```

```
log_pipe_ROC.plot(ax=ax[0])
ax[0].set_title("Pipe ROC")
log_grid_ROC.plot(ax=ax[1])
ax[1].set_title("Grid ROC")
log_transform_ROC.plot(ax=ax[2])
ax[2].set_title("Transform ROC")

plt.tight_layout()
plt.show()
```



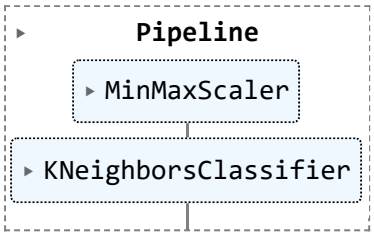

```
logistic Test_Set
[[70 13]
 [ 8 89]]
              precision    recall  f1-score   support

           0       0.90      0.84      0.87        83
           1       0.87      0.92      0.89        97

    accuracy                           0.88       180
   macro avg       0.88      0.88      0.88       180
weighted avg       0.88      0.88      0.88       180


logistic Train_Set
[[308  59]
 [ 42 311]]
              precision    recall  f1-score   support

           0       0.88      0.84      0.86       367
           1       0.84      0.88      0.86       353

    accuracy                           0.86       720
   macro avg       0.86      0.86      0.86       720
weighted avg       0.86      0.86      0.86       720
```

```
logisticgrid Test_Set
[[70 13]
 [ 7 90]]
              precision    recall  f1-score   support

           0       0.91      0.84      0.88        83
           1       0.87      0.93      0.90        97

    accuracy                           0.89       180
   macro avg       0.89      0.89      0.89       180
weighted avg       0.89      0.89      0.89       180


logisticgrid Train_Set
[[309  58]
 [ 40 313]]
              precision    recall  f1-score   support

           0       0.89      0.84      0.86       367
           1       0.84      0.89      0.86       353

    accuracy                           0.86       720
   macro avg       0.86      0.86      0.86       720
weighted avg       0.86      0.86      0.86       720



logisticgrid_transform Test_Set
[[70 13]
 [11 86]]
              precision    recall  f1-score   support

           0       0.86      0.84      0.85        83
           1       0.87      0.89      0.88        97

    accuracy                           0.87       180
   macro avg       0.87      0.86      0.87       180
weighted avg       0.87      0.87      0.87       180


logisticgrid_transform Train_Set
[[319  48]
 [ 48 305]]
              precision    recall  f1-score   support

           0       0.87      0.87      0.87       367
           1       0.86      0.86      0.86       353

    accuracy                           0.87       720
   macro avg       0.87      0.87      0.87       720
weighted avg       0.87      0.87      0.87       720
```

KNN Model

In [68]:
```
operations = [("scaler", MinMaxScaler()),
              ("knn", KNeighborsClassifier(n_neighbors=5))
              ]  #default n_neighbors=5

pipe_model = Pipeline(steps=operations)

pipe_model.fit(X_train, y_train)
```

Out[68]:



In [69]:
```
y_pred = pipe_model.predict(X_test)
y_pred
```

Out[69]:
```
array([1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 1, 0, 0, 1,
       1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 1,
       1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0,
       1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 1, 0, 1,
       0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0,
       0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1,
       1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 0, 0,
       1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1,
       1, 1, 0, 1])
```

In [70]:
```
y_pred_proba = pipe_model.predict_proba(X_test)
pd.DataFrame(y_pred_proba)
```

Out[70]:

	0	1
0	0.2	0.8
1	0.2	0.8
2	0.4	0.6
3	0.4	0.6
4	0.0	1.0
...	...	...
175	0.4	0.6
176	0.4	0.6
177	0.0	1.0
178	1.0	0.0
179	0.0	1.0

180 rows × 2 columns

In [71]:
```
y_pred = pipe_model.predict(X_test)
my_dict = {
    "Actual": y_test,
    "Pred": y_pred,
    "Proba_1": y_pred_proba[:, 1],
    "Proba_0": y_pred_proba[:, 0]
}
pd.DataFrame.from_dict(my_dict).sample(10)
```

Out[71]:

	Actual	Pred	Proba_1	Proba_0
353	1	1	0.6	0.4
647	0	0	0.4	0.6
536	0	0	0.0	1.0
752	0	1	0.6	0.4
242	1	1	0.8	0.2
318	1	1	1.0	0.0
367	1	1	1.0	0.0
192	1	1	1.0	0.0
315	1	1	1.0	0.0
594	0	0	0.0	1.0

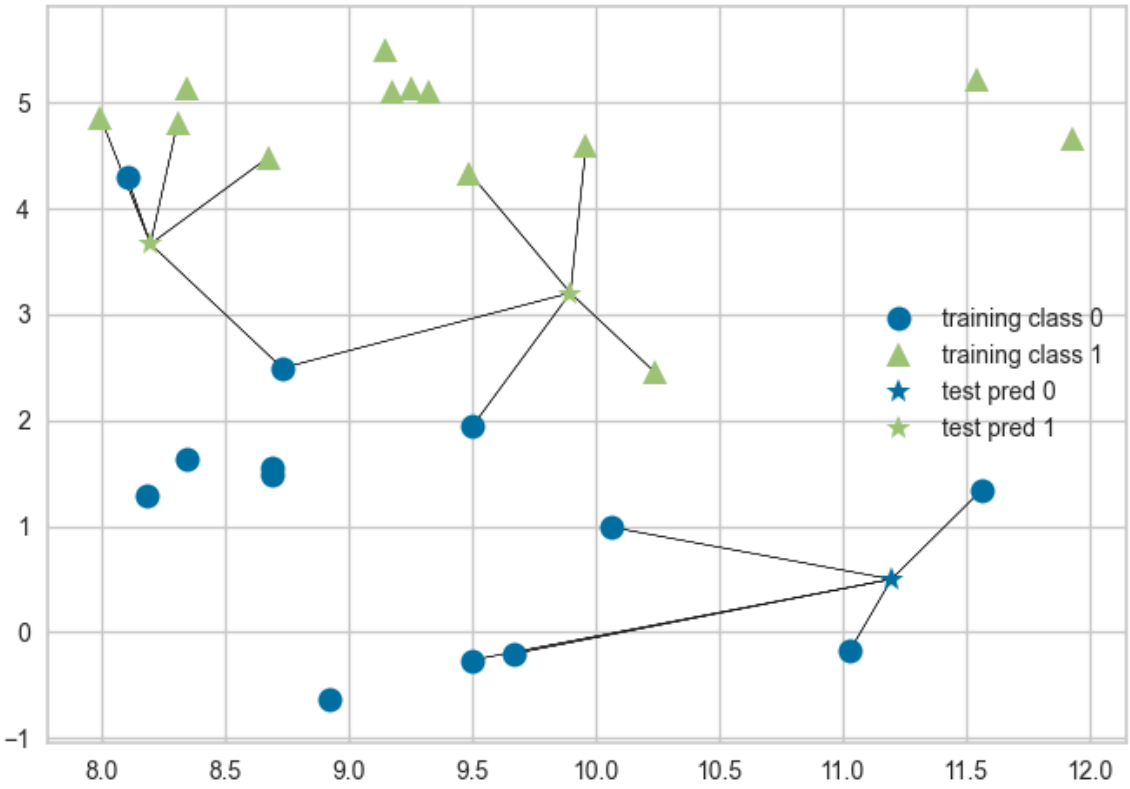
In [72]:
```
test_data["pred"] = y_pred
test_data
test_data[((test_data["Class"] == 0) & (test_data["pred"] == 1)) |
          ((test_data["Class"] == 1) & (test_data["pred"] == 0))]
```

Out[72]:

	Area	MajorAxisLength	MinorAxisLength	Eccentricity	ConvexArea	Extent
633	54357	362.594718	192.949367	0.846659	56006	0.699918
731	84383	403.909415	271.251509	0.740945	87629	0.674945
831	48488	275.337717	226.802199	0.566991	50103	0.669594
688	57999	311.022470	243.476121	0.622242	61519	0.656335
425	85954	405.915167	272.083325	0.742094	87690	0.714818
885	54502	346.457978	204.081212	0.808096	56464	0.636111
717	58870	355.136018	212.029460	0.802213	60490	0.734956
173	68627	411.888524	216.894000	0.850123	70932	0.738242
604	102013	453.893458	292.130181	0.765354	106036	0.637828
571	79492	422.567329	243.116296	0.817920	82708	0.637353
271	84057	452.775695	245.219982	0.840641	86710	0.622018
52	65727	403.194272	210.073264	0.853543	67372	0.616981
345	75540	422.098490	237.638677	0.826461	78465	0.705085
827	61861	345.943650	235.430468	0.732706	67390	0.702280
822	62280	388.582055	205.514392	0.848695	63596	0.791440
872	66938	356.323284	248.674245	0.716206	69880	0.708661
775	88747	425.007280	268.668464	0.774846	92317	0.761313
349	67798	418.408134	207.825511	0.867919	69356	0.691027
124	96064	411.207053	303.232452	0.675434	99609	0.731331
85	180898	843.956653	323.190569	0.923770	221396	0.454189
193	79748	441.934293	231.717531	0.851518	81718	0.738448
585	61444	371.693826	213.179806	0.819180	64321	0.737473
300	87429	408.926028	273.887942	0.742565	89063	0.703575
724	96442	450.408136	276.922615	0.788663	100712	0.703058
218	72915	414.718073	229.555770	0.832834	76912	0.679771
635	70461	376.170843	243.954829	0.761196	72819	0.728279
752	49691	336.678058	189.261667	0.827039	52077	0.724950

In [73]:

```
# pip install mglearn
import mglearn
mglearn.plots.plot_knn_classification(n_neighbors=5)
```



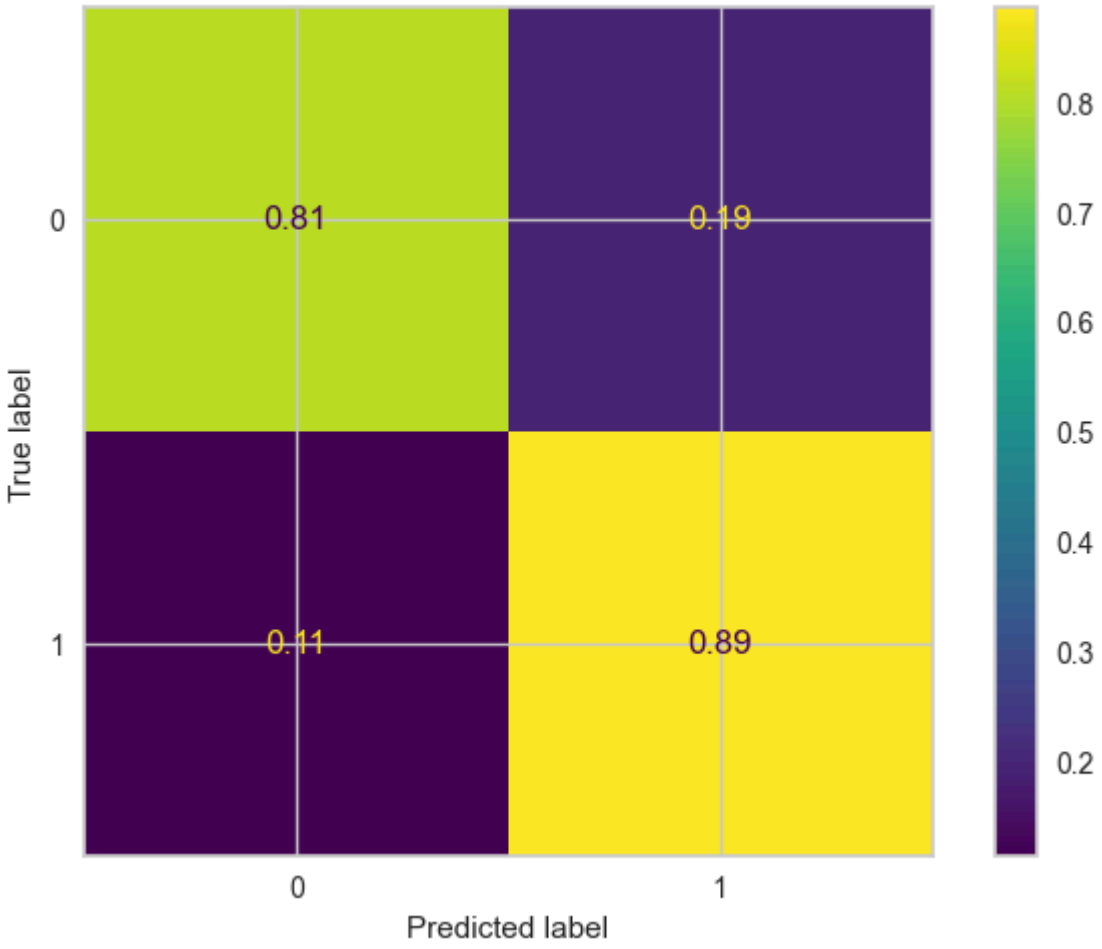
Model Performance

In [74]:
```
confusion_matrix(y_test, y_pred)
```

Out[74]:
```
array([[67, 16],
       [11, 86]], dtype=int64)
```

In [75]:
```
knn_matrix = ConfusionMatrixDisplay.from_estimator(pipe_model,
                                                   X_test,
                                                   y_test,
                                                   normalize='true')
knn_matrix
```

Out[75]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x21a321a1fd
0>
```



In [76]:
```
eval_metric(pipe_model, X_train, y_train, X_test, y_test, "knn")
```

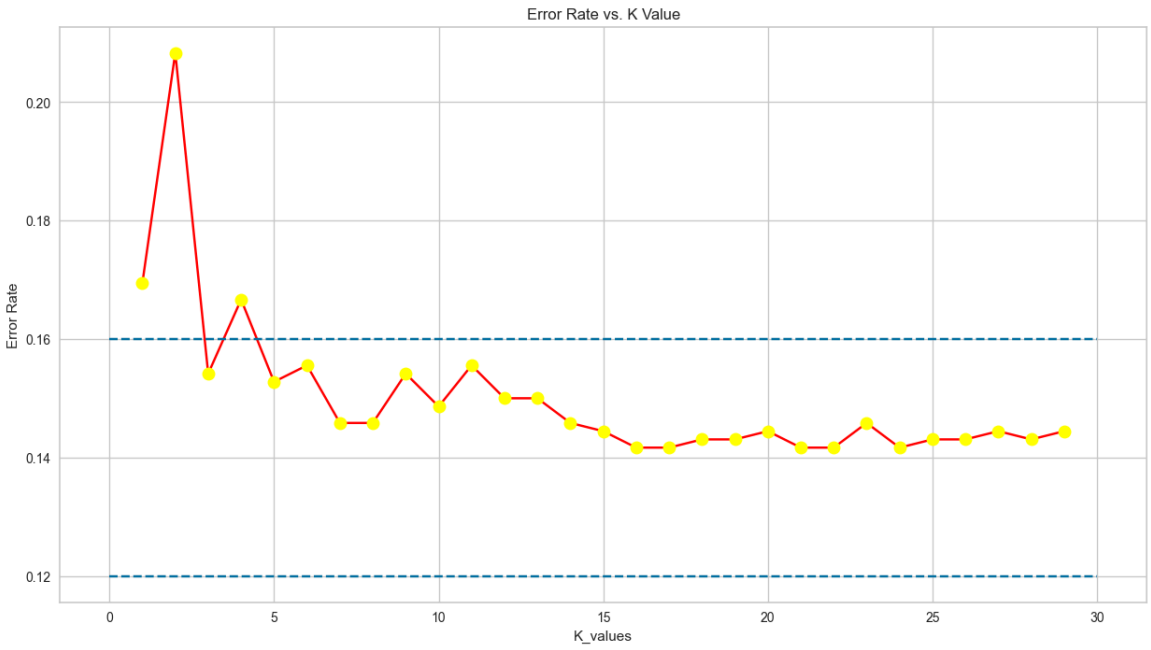
```
knn Test_Set
[[67 16]
 [11 86]]
              precision    recall  f1-score   support

           0       0.86      0.81      0.83        83
           1       0.84      0.89      0.86        97

    accuracy                           0.85       180
   macro avg       0.85      0.85      0.85       180
weighted avg       0.85      0.85      0.85       180


knn Train_Set
[[317  50]
 [ 32 321]]
              precision    recall  f1-score   support

           0       0.91      0.86      0.89       367
           1       0.87      0.91      0.89       353

    accuracy                           0.89       720
   macro avg       0.89      0.89      0.89       720
weighted avg       0.89      0.89      0.89       720
```

Elbow Method for Choosing Reasonable K Values

In [77]:
```
test_error_rates = []

for k in range(1, 30):

    operations = [("scaler", MinMaxScaler()),
                  ("knn", KNeighborsClassifier(n_neighbors=k))]

    knn_pipe_model = Pipeline(steps=operations)

    scores = cross_validate(knn_pipe_model,
                            X_train,
                            y_train,
                            scoring=['accuracy'],
                            cv=10)

    accuracy_mean = scores["test_accuracy"].mean()

    test_error = 1 - accuracy_mean

    test_error_rates.append(test_error)
```

In [78]:
```
plt.figure(figsize=(15, 8))
plt.plot(range(1, 30),
         test_error_rates,
         color='red',
         marker='o',
         markerfacecolor='yellow',
         markersize=10)

plt.title('Error Rate vs. K Value')
plt.xlabel('K_values')
plt.ylabel('Error Rate')
plt.hlines(y=0.12, xmin=0, xmax=30, colors='b', linestyles="--")
plt.hlines(y=0.16, xmin=0, xmax=30, colors='b', linestyles="--")
```

Out[78]: <matplotlib.collections.LineCollection at 0x21a390a9250>

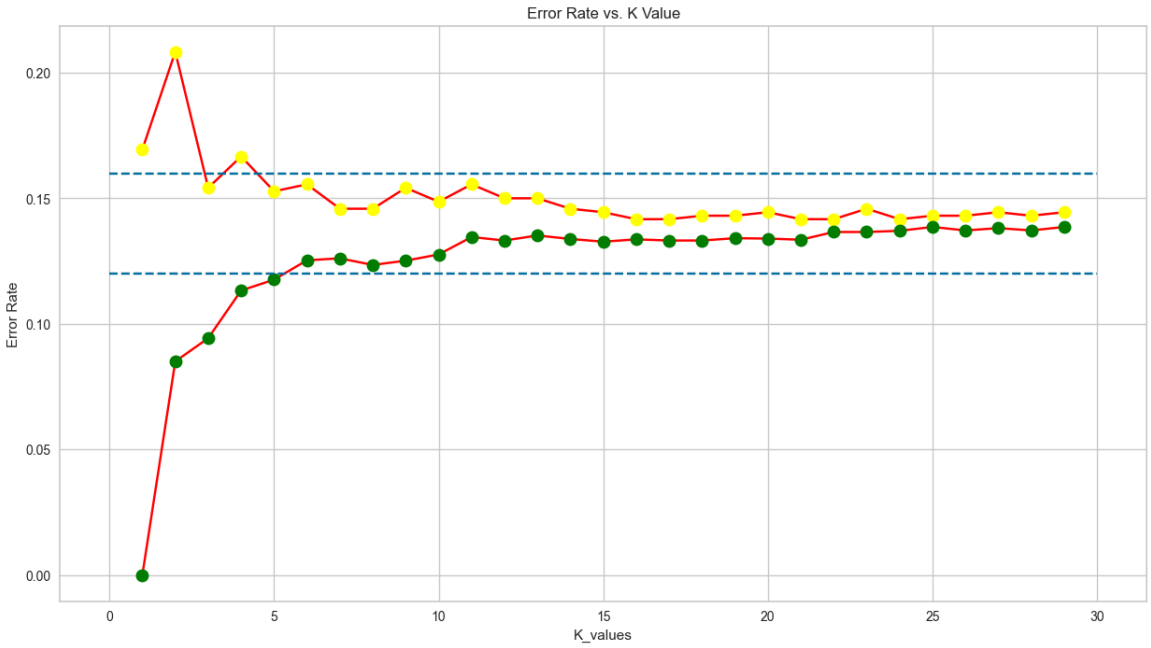


Overfiting and underfiting control for k values

In [79]:
```
test_error_rates = []
train_error_rates = []

for k in range(1, 30):

    operations = [("scaler", MinMaxScaler()),
                  ("knn", KNeighborsClassifier(n_neighbors=k))]

    knn_pipe_model = Pipeline(steps=operations)

    knn_pipe_model.fit(X_train, y_train)

    scores = cross_validate(knn_pipe_model,
                            X_train,
                            y_train,
                            scoring=['accuracy'],
                            cv=10,
                            return_train_score=True)

    accuracy_test_mean = scores["test_accuracy"].mean()
    accuracy_train_mean = scores["train_accuracy"].mean()

    test_error = 1 - accuracy_test_mean
    train_error = 1 - accuracy_train_mean
    test_error_rates.append(test_error)
    train_error_rates.append(train_error)
```

In [80]:
```
plt.figure(figsize=(15, 8))
plt.plot(range(1, 30),
         test_error_rates,
         color='red',
         marker='o',
         markerfacecolor='yellow',
         markersize=10)

plt.plot(range(1, 30),
         train_error_rates,
         color='red',
         marker='o',
         markerfacecolor='green',
         markersize=10)

plt.title('Error Rate vs. K Value')
plt.xlabel('K_values')
plt.ylabel('Error Rate')
plt.hlines(y=0.12, xmin=0, xmax=30, colors='b', linestyles="--")
plt.hlines(y=0.16, xmin=0, xmax=30, colors='b', linestyles="--")
```

Out[80]:
```
<matplotlib.collections.LineCollection at 0x21a39170dd0>
```



Scores by Various K Values

In [81]:
```
k_list = [3, 4, 6, 7, 8, 16]

for i in k_list:
    operations = [("scaler", MinMaxScaler()),
                  ("knn", KNeighborsClassifier(n_neighbors=i))]
    knn = Pipeline(steps=operations)
    knn.fit(X_train, y_train)
    print(f'WITH K={i}\n')
    eval_metric(knn, X_train, y_train, X_test, y_test, "knn_elbow")
```

```
WITH K=3

knn_elbow Test_Set
[[67 16]
 [13 84]]
              precision    recall  f1-score   support

           0       0.84      0.81      0.82        83
           1       0.84      0.87      0.85        97

    accuracy                           0.84       180
   macro avg       0.84      0.84      0.84       180
weighted avg       0.84      0.84      0.84       180


knn_elbow Train_Set
[[325  42]
 [ 26 327]]
              precision    recall  f1-score   support

           0       0.93      0.89      0.91       367
           1       0.89      0.93      0.91       353

    accuracy                           0.91       720
   macro avg       0.91      0.91      0.91       720
weighted avg       0.91      0.91      0.91       720

WITH K=4

knn_elbow Test_Set
[[73 10]
 [18 79]]
              precision    recall  f1-score   support

           0       0.80      0.88      0.84        83
           1       0.89      0.81      0.85        97

    accuracy                           0.84       180
   macro avg       0.84      0.85      0.84       180
weighted avg       0.85      0.84      0.84       180


knn_elbow Train_Set
[[335  32]
 [ 49 304]]
              precision    recall  f1-score   support

           0       0.87      0.91      0.89       367
           1       0.90      0.86      0.88       353

    accuracy                           0.89       720
   macro avg       0.89      0.89      0.89       720
weighted avg       0.89      0.89      0.89       720

WITH K=6

knn_elbow Test_Set
[[69 14]
 [14 83]]
              precision    recall  f1-score   support
```

```
              0       0.83      0.83      0.83        83
              1       0.86      0.86      0.86        97

    accuracy                           0.84       180
   macro avg       0.84      0.84      0.84       180
weighted avg       0.84      0.84      0.84       180


knn_elbow Train_Set
[[323  44]
 [ 48 305]]
              precision    recall  f1-score   support

           0       0.87      0.88      0.88       367
           1       0.87      0.86      0.87       353

    accuracy                           0.87       720
   macro avg       0.87      0.87      0.87       720
weighted avg       0.87      0.87      0.87       720

WITH K=7

knn_elbow Test_Set
[[69 14]
 [ 7 90]]
              precision    recall  f1-score   support

           0       0.91      0.83      0.87        83
           1       0.87      0.93      0.90        97

    accuracy                           0.88       180
   macro avg       0.89      0.88      0.88       180
weighted avg       0.88      0.88      0.88       180


knn_elbow Train_Set
[[313  54]
 [ 36 317]]
              precision    recall  f1-score   support

           0       0.90      0.85      0.87       367
           1       0.85      0.90      0.88       353

    accuracy                           0.88       720
   macro avg       0.88      0.88      0.87       720
weighted avg       0.88      0.88      0.87       720

WITH K=8

knn_elbow Test_Set
[[70 13]
 [ 8 89]]
              precision    recall  f1-score   support

           0       0.90      0.84      0.87        83
           1       0.87      0.92      0.89        97

    accuracy                           0.88       180
   macro avg       0.88      0.88      0.88       180
```

```
weighted avg       0.88      0.88      0.88       180


knn_elbow Train_Set
[[318  49]
 [ 41 312]]
              precision    recall  f1-score   support

           0       0.89      0.87      0.88       367
           1       0.86      0.88      0.87       353

    accuracy                           0.88       720
   macro avg       0.88      0.88      0.87       720
weighted avg       0.88      0.88      0.88       720

WITH K=16

knn_elbow Test_Set
[[71 12]
 [ 7 90]]
              precision    recall  f1-score   support

           0       0.91      0.86      0.88        83
           1       0.88      0.93      0.90        97

    accuracy                           0.89       180
   macro avg       0.90      0.89      0.89       180
weighted avg       0.90      0.89      0.89       180


knn_elbow Train_Set
[[312  55]
 [ 44 309]]
              precision    recall  f1-score   support

           0       0.88      0.85      0.86       367
           1       0.85      0.88      0.86       353

    accuracy                           0.86       720
   macro avg       0.86      0.86      0.86       720
weighted avg       0.86      0.86      0.86       720
```

Cross Validate For Optimal K Value

In [82]:
```
operations = [("scaler", MinMaxScaler()),
              ("knn", KNeighborsClassifier(n_neighbors=15))]
model = Pipeline(steps=operations)

scores = cross_validate(model,
                        X_train,
                        y_train,
                        scoring=['accuracy', 'precision', 'recall', 'f1'],
                        cv=10,
                        return_train_score=True)
df_scores = pd.DataFrame(scores, index=range(1, 11))
df_scores
```

Out[82]:

	fit_time	score_time	test_accuracy	train_accuracy	test_precision	train_precision
1	0.008718	0.023660	0.847222	0.870370	0.833333	0.846154
2	0.010515	0.023833	0.875000	0.862654	0.861111	0.831884
3	0.004745	0.019609	0.888889	0.865741	0.864865	0.844776
4	0.008305	0.018284	0.847222	0.859568	0.815789	0.840841
5	0.006938	0.021026	0.805556	0.878086	0.744186	0.854599
6	0.007801	0.017963	0.875000	0.858025	0.825000	0.832353
7	0.000000	0.016827	0.902778	0.865741	0.911765	0.838710
8	0.000000	0.016675	0.833333	0.876543	0.785714	0.858006
9	0.005512	0.016438	0.791667	0.875000	0.800000	0.845029
10	0.010510	0.014818	0.888889	0.861111	0.868421	0.825215

In [83]:
```
df_scores.mean()[2:]
```

Out[83]:
```
test_accuracy      0.855556
train_accuracy     0.867284
test_precision     0.831018
train_precision    0.841757
test_recall        0.889683
train_recall       0.898338
test_f1            0.858236
train_f1           0.869074
dtype: float64
```

Gridsearch Method for Choosing Reasonable K Values

In [143…
```
operations = [("scaler", MinMaxScaler()), ("knn", KNeighborsClassifier())]
knn_model = Pipeline(steps=operations)
```

In [144…
```
knn_model.get_params()
```

Out[144…
```
{'memory': None,
 'steps': [('scaler', MinMaxScaler()), ('knn', KNeighborsClassifier())],
 'verbose': False,
 'scaler': MinMaxScaler(),
 'knn': KNeighborsClassifier(),
 'scaler__clip': False,
 'scaler__copy': True,
 'scaler__feature_range': (0, 1),
 'knn__algorithm': 'auto',
 'knn__leaf_size': 30,
 'knn__metric': 'minkowski',
 'knn__metric_params': None,
 'knn__n_jobs': None,
 'knn__n_neighbors': 5,
 'knn__p': 2,
 'knn__weights': 'uniform'}
```

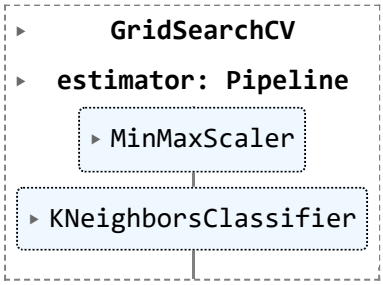
In [145…
```
k_values = range(1,30)
```

In [165…
```
param_grid = [
    {
        "knn__n_neighbors": k_values,
        "knn__metric": ['euclidean', 'manhattan'],
        "knn__weights": ['uniform']
    },
    {
        "knn__n_neighbors": k_values,
        "knn__metric": ['minkowski'],
        "knn__p": [1, 2],
        "knn__weights": ['uniform', 'distance']
    }
]
```

In [166…
```
knn_grid_model = GridSearchCV(knn_model,
                              param_grid,
                              scoring='accuracy',
                              cv=10,
                              return_train_score=True,
                              n_jobs=-1)
```

In [167…
```
knn_grid_model.fit(X_train, y_train)
```

Out[167…

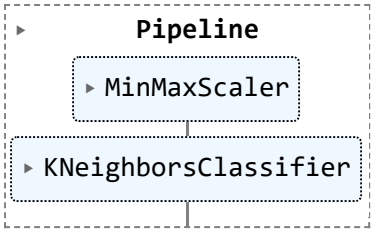


In [168…
```
knn_grid_model.best_params_
```

Out[168…
```
{'knn__metric': 'manhattan', 'knn__n_neighbors': 28, 'knn__weights': 'unifor
m'}
```

In [169…
```
knn_grid_model.best_estimator_
```

Out[169…



In [170…
```
knn_grid_model.best_index_
```

Out[170… 56

In [171…
```
pd.DataFrame(
    knn_grid_model.cv_results_).loc[56,
                                    ["mean_test_score", "mean_train_score"]]
```

Out[171…
```
mean_test_score       0.8625
mean_train_score    0.865586
Name: 56, dtype: object
```

In [172…
```
knn_grid_model.best_score_
```

Out[172… 0.8625

In [175…
```
print('WITH K=29\n')
eval_metric(knn_grid_model, X_train, y_train, X_test, y_test, "knn_grid")
```

```
WITH K=29

knn_grid Test_Set
[[71 12]
 [ 7 90]]
              precision    recall  f1-score   support

           0       0.91      0.86      0.88        83
           1       0.88      0.93      0.90        97

    accuracy                           0.89       180
   macro avg       0.90      0.89      0.89       180
weighted avg       0.90      0.89      0.89       180


knn_grid Train_Set
[[306  61]
 [ 40 313]]
              precision    recall  f1-score   support

           0       0.88      0.83      0.86       367
           1       0.84      0.89      0.86       353

    accuracy                           0.86       720
   macro avg       0.86      0.86      0.86       720
weighted avg       0.86      0.86      0.86       720
```

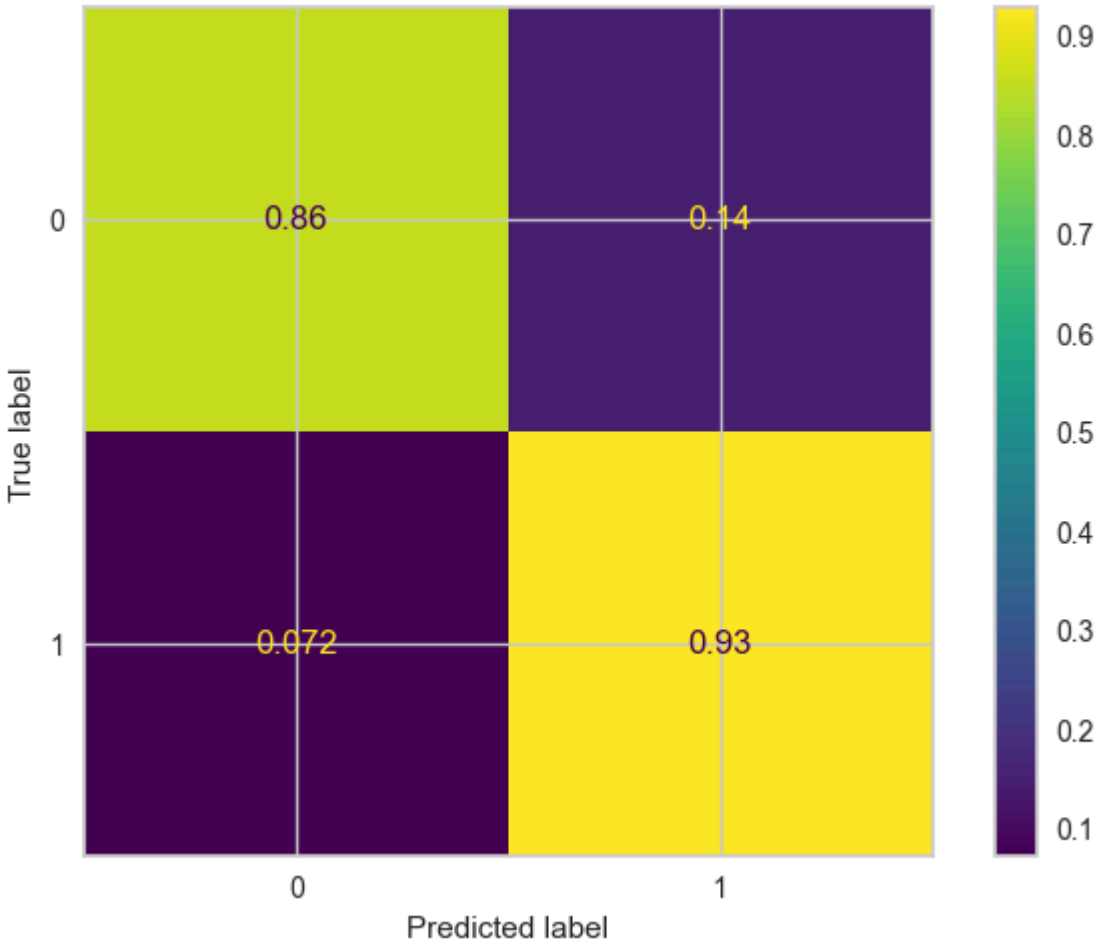
```
knn Test_Set
[[67 16]
 [11 86]]
              precision    recall  f1-score   support

           0       0.86      0.81      0.83        83
           1       0.84      0.89      0.86        97

    accuracy                           0.85       180
   macro avg       0.85      0.85      0.85       180
weighted avg       0.85      0.85      0.85       180


knn Train_Set
[[317  50]
 [ 32 321]]
              precision    recall  f1-score   support

           0       0.91      0.86      0.89       367
           1       0.87      0.91      0.89       353

    accuracy                           0.89       720
   macro avg       0.89      0.89      0.89       720
weighted avg       0.89      0.89      0.89       720
```

In [180…
```
confusion_matrix(y_test, y_pred)
```

Out[180…
```
array([[67, 16],
       [11, 86]], dtype=int64)
```

In [181…
```
knn_grid_matrix = ConfusionMatrixDisplay.from_estimator(knn_grid_model,
                                                        X_test,
                                                        y_test,
                                                        normalize='true')
knn_grid_matrix
```

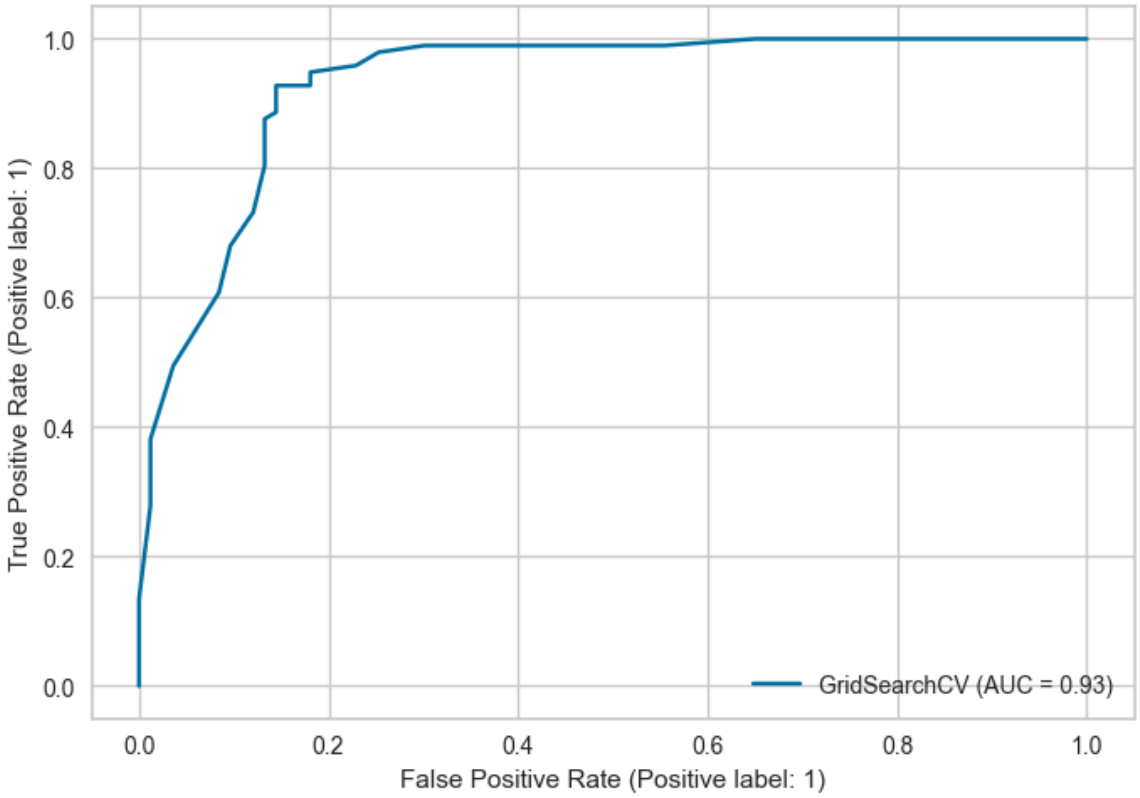
Out[181…
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x21a37d07d1
0>
```



Evaluating ROC Curves

In [182…
```
knn_grid_ROC = RocCurveDisplay.from_estimator(knn_grid_model, X_test, y_test);
knn_grid_ROC
```

Out[182…
```
<sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x21a38e70b90>
```



In [183…
```
y_pred_proba = knn.predict_proba(X_test)
roc_auc_score(y_test, y_pred_proba[:,1])
```

Out[183…
```
0.9255372003477829
```

SVM Model

In [185…
```
operations = [("scaler", MinMaxScaler()), ("SVC", SVC())]
pipe_model = Pipeline(steps=operations)
```

Model Performance

In [186…
```
pipe_model.fit(X_train, y_train)

eval_metric(pipe_model, X_train, y_train, X_test, y_test, "svm")
```

```
svm Test_Set
[[68 15]
 [ 4 93]]
              precision    recall  f1-score   support

           0       0.94      0.82      0.88        83
           1       0.86      0.96      0.91        97

    accuracy                           0.89       180
   macro avg       0.90      0.89      0.89       180
weighted avg       0.90      0.89      0.89       180


svm Train_Set
[[306  61]
 [ 34 319]]
              precision    recall  f1-score   support

           0       0.90      0.83      0.87       367
           1       0.84      0.90      0.87       353

    accuracy                           0.87       720
   macro avg       0.87      0.87      0.87       720
weighted avg       0.87      0.87      0.87       720
```

In [187…
```
operations = [("scaler", MinMaxScaler()), ("SVC", SVC())]

pipe_model = Pipeline(steps=operations)

scores = cross_validate(pipe_model,
                        X_train,
                        y_train,
                        scoring=['accuracy', 'precision', 'recall', 'f1'],
                        cv=10,
                        return_train_score=True)

df_scores = pd.DataFrame(scores, index=range(1, 11))
df_scores.mean()[2:]
```

Out[187…
```
test_accuracy      0.865278
train_accuracy     0.867901
test_precision     0.840013
train_precision    0.839448
test_recall        0.901032
train_recall       0.903376
test_f1            0.868530
train_f1           0.870225
dtype: float64
```

GridsearchCV

In [188…
```
pipe_model.get_params()
```

Out[188…
```
{'memory': None,
 'steps': [('scaler', MinMaxScaler()), ('SVC', SVC())],
 'verbose': False,
 'scaler': MinMaxScaler(),
 'SVC': SVC(),
 'scaler__clip': False,
 'scaler__copy': True,
 'scaler__feature_range': (0, 1),
 'SVC__C': 1.0,
 'SVC__break_ties': False,
 'SVC__cache_size': 200,
 'SVC__class_weight': None,
 'SVC__coef0': 0.0,
 'SVC__decision_function_shape': 'ovr',
 'SVC__degree': 3,
 'SVC__gamma': 'scale',
 'SVC__kernel': 'rbf',
 'SVC__max_iter': -1,
 'SVC__probability': False,
 'SVC__random_state': None,
 'SVC__shrinking': True,
 'SVC__tol': 0.001,
 'SVC__verbose': False}
```

In [299…
```
param_grid = {
    'SVC__C': [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8],
    'SVC__gamma': ["scale", "auto", 0.2, 0.3, 0.5],
    'SVC__kernel': ['rbf', 'linear'],
    'SVC__class_weight': ["balanced", None]
}
```

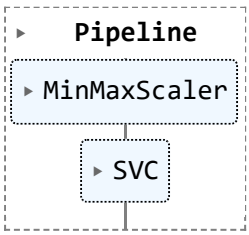
In [300…
```
operations = [("scaler", MinMaxScaler()), ("SVC", SVC(probability=True))]

pipe_model = Pipeline(steps=operations)

svm_model_grid = GridSearchCV(pipe_model,
                              param_grid,
                              scoring="accuracy",
                              cv=5,
                              return_train_score=True,
                              n_jobs=1).fit(X_train, y_train)
```

In [301…
```
svm_model_grid.best_params_
```

Out[301…
```
{'SVC__C': 0.5,
 'SVC__class_weight': 'balanced',
 'SVC__gamma': 'scale',
 'SVC__kernel': 'rbf'}
```

In [302…
```
svm_model_grid.best_estimator_
```

Out[302…



In [303…
```
svm_model_grid.best_index_
```

Out[303…
```
80
```

In [304…
```
pd.DataFrame(
    svm_model_grid.cv_results_).loc[80,
                                    ["mean_test_score", "mean_train_score"]]
```

Out[304…
```
mean_test_score       0.8625
mean_train_score    0.865278
Name: 80, dtype: object
```

In [305…
```
svm_model_grid.best_score_
```

Out[305…
```
0.8625
```

In [306…
```
eval_metric(svm_model_grid, X_train, y_train, X_test, y_test, "svm_grid")
```

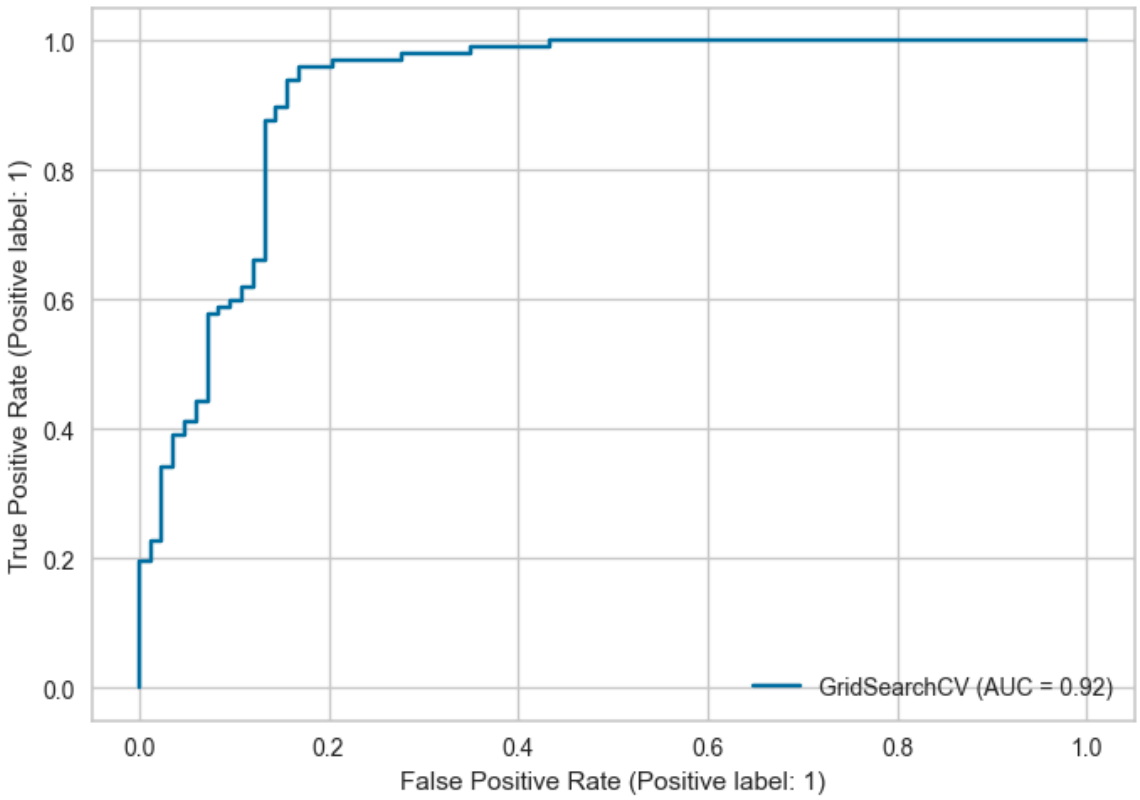
```
svm_grid Test_Set
[[68 15]
 [ 4 93]]
              precision    recall  f1-score   support

           0       0.94      0.82      0.88        83
           1       0.86      0.96      0.91        97

    accuracy                           0.89       180
   macro avg       0.90      0.89      0.89       180
weighted avg       0.90      0.89      0.89       180


svm_grid Train_Set
[[305  62]
 [ 34 319]]
              precision    recall  f1-score   support

           0       0.90      0.83      0.86       367
           1       0.84      0.90      0.87       353

    accuracy                           0.87       720
   macro avg       0.87      0.87      0.87       720
weighted avg       0.87      0.87      0.87       720
```

```
svm Test_Set
[[68 15]
 [ 4 93]]
              precision    recall  f1-score   support

           0       0.94      0.82      0.88        83
           1       0.86      0.96      0.91        97

    accuracy                           0.89       180
   macro avg       0.90      0.89      0.89       180
weighted avg       0.90      0.89      0.89       180


svm Train_Set
[[306  61]
 [ 34 319]]
              precision    recall  f1-score   support

           0       0.90      0.83      0.87       367
           1       0.84      0.90      0.87       353

    accuracy                           0.87       720
   macro avg       0.87      0.87      0.87       720
weighted avg       0.87      0.87      0.87       720
```

Evaluating ROC Curves

In [307…
```
svm_grid_ROC = RocCurveDisplay.from_estimator(svm_model_grid, X_test, y_test);
svm_grid_ROC
```

Out[307…
```
<sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x21a3bde7b10>
```



In [308…
```
y_pred_proba = svm_model_grid.predict_proba(X_test)
roc_auc_score(y_test, y_pred_proba[:,1])
```

Out[308… 0.918767854924854

In [310…
```
decision_function = svm_model_grid.decision_function(X_test)
average_precision_score(y_test, decision_function)
```

Out[310… 0.9095357041156814

In [311…
```
df1 = df.copy()
```

SVM Model Transformation

In [312…
```
# Select the columns of interest
columns_of_interest = [
    'Area', 'MajorAxisLength', 'MinorAxisLength', 'Eccentricity', 'ConvexArea',
    'Extent', 'Perimeter'
]

# Initialize the PowerTransformer
power_transformer = PowerTransformer(method='yeo-johnson')

# Apply the Yeo-Johnson transformation to the selected columns
df1[columns_of_interest] = power_transformer.fit_transform(
    df1[columns_of_interest])
```

In [313…
```
df1.describe().T
```

Out[313…

	count	mean	std	min	25%	50%	75
Area	900.0	-2.210577e-15	1.000556	-3.107110	-0.710225	0.000939	0.6728
MajorAxisLength	900.0	4.357995e-15	1.000556	-2.726264	-0.710980	-0.026095	0.7043
MinorAxisLength	900.0	-7.358065e-15	1.000556	-3.195617	-0.672585	-0.006777	0.6208
Eccentricity	900.0	1.263187e-16	1.000556	-2.535103	-0.662466	0.026363	0.6830
ConvexArea	900.0	4.105358e-15	1.000556	-3.183118	-0.714198	-0.002160	0.6632
Extent	900.0	-4.026409e-16	1.000556	-3.251373	-0.672369	0.027789	0.6489
Perimeter	900.0	5.425389e-14	1.000556	-3.204105	-0.703709	-0.005496	0.6747
Class	900.0	5.000000e-01	0.500278	0.000000	0.000000	0.500000	1.0000

In [314…
```
X1 = df1.drop("Class", axis=1)
y1 = df1['Class']
```

In [315…
```
X1_train, X1_test, y1_train, y1_test = train_test_split(X1,
                                                        y1,
                                                        test_size=0.2,
                                                        random_state=10)
```

In [316…
```
param_grid = {
    'SVC__C': [0.01, 0.1, 0.5, 1, 50],
    'SVC__gamma': ["scale", "auto", 0.2, 0.3, 0.5, 1],
    'SVC__kernel': ['rbf', 'linear']
}
```

In [317…
```
operations = [("scaler", StandardScaler()), ("SVC", SVC(probability=True))]

pipe_model = Pipeline(steps=operations)

svm_model_grid = GridSearchCV(pipe_model,
                              param_grid,
                              scoring="accuracy",
                              cv=5,
                              return_train_score=True,
                              n_jobs=1).fit(X1_train, y1_train)
```

In [318…
```
svm_model_grid.best_params_
```

Out[318…
```
{'SVC__C': 50, 'SVC__gamma': 'scale', 'SVC__kernel': 'linear'}
```

In [319…
```
svm_model_grid.best_index_
```

Out[319…
```
49
```

In [320…
```
pd.DataFrame(
    svm_model_grid.cv_results_).loc[49,
                                    ["mean_test_score", "mean_train_score"]]
```

Out[320…
```
mean_test_score     0.873611
mean_train_score    0.874653
Name: 49, dtype: object
```

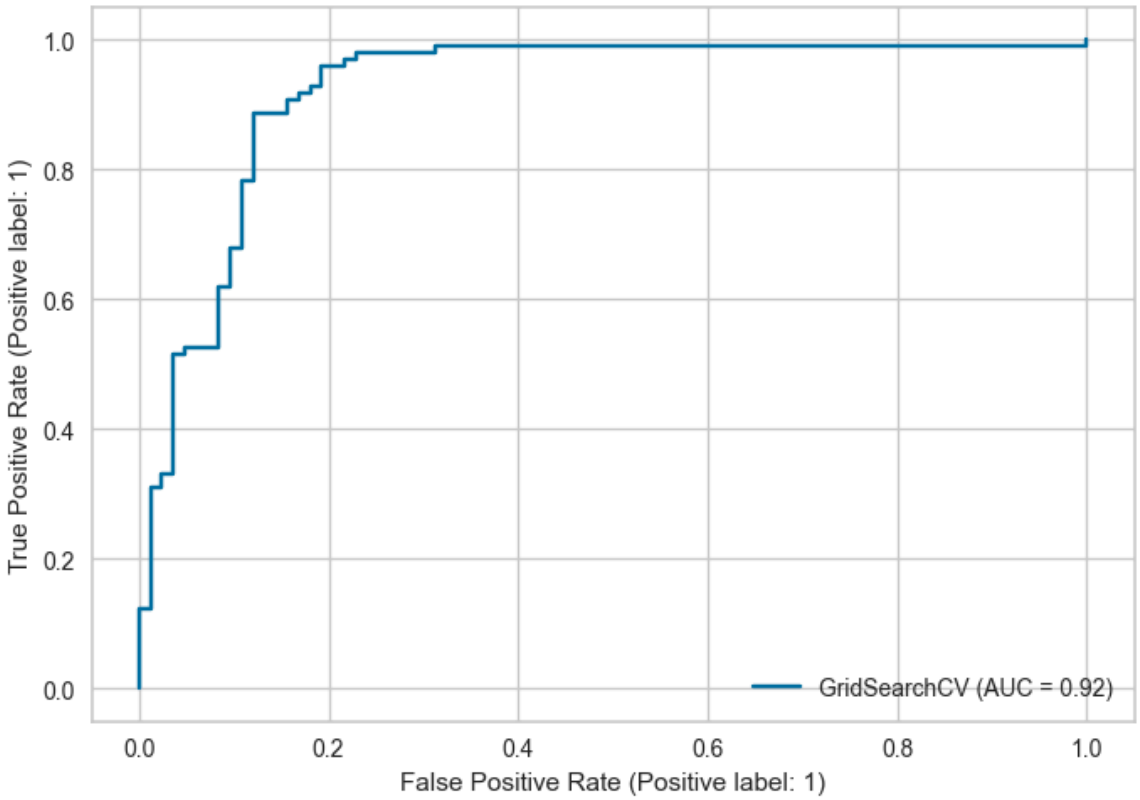
In [321…
```
svm_model_grid.best_score_
```

Out[321…
```
0.8736111111111111
```

In [322…
```
eval_metric(svm_model_grid, X1_train, y1_train, X1_test, y1_test, "svm_transfo
```

```
svm_transform__grid Test_Set
[[70 13]
 [ 9 88]]
              precision    recall  f1-score   support

           0       0.89      0.84      0.86        83
           1       0.87      0.91      0.89        97

    accuracy                           0.88       180
   macro avg       0.88      0.88      0.88       180
weighted avg       0.88      0.88      0.88       180


svm_transform__grid Train_Set
[[324  43]
 [ 47 306]]
              precision    recall  f1-score   support

           0       0.87      0.88      0.88       367
           1       0.88      0.87      0.87       353

    accuracy                           0.88       720
   macro avg       0.88      0.87      0.87       720
weighted avg       0.88      0.88      0.87       720
```

In [323…
```
RocCurveDisplay.from_estimator(svm_model_grid, X1_test, y1_test);
```

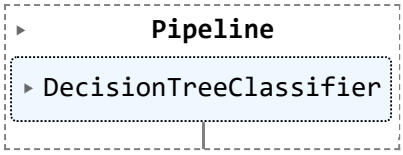


Desicion Tree Classification Model

In [327…
```
operations = [("DT_model", DecisionTreeClassifier())]
```

```
pipe_model = Pipeline(steps=operations)

pipe_model.fit(X_train, y_train)
```

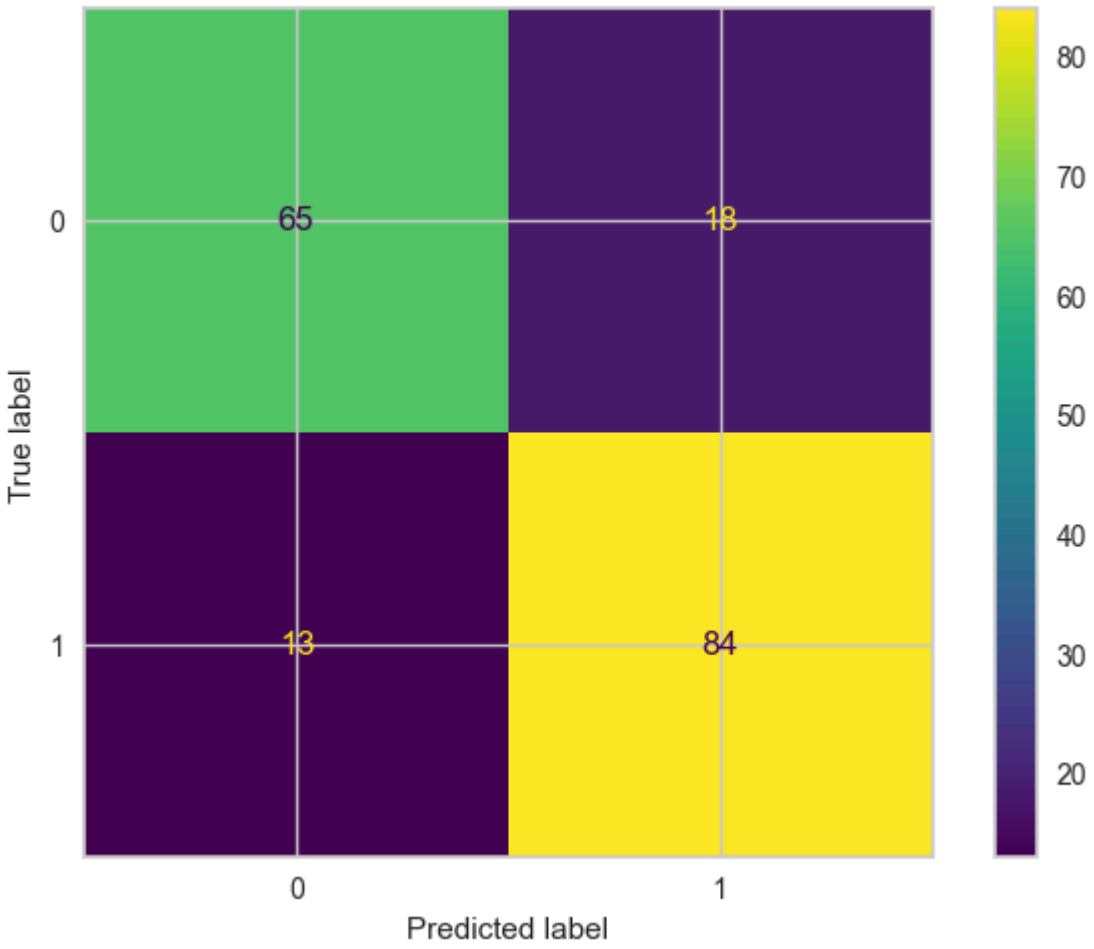
Out[327…



Model Performance

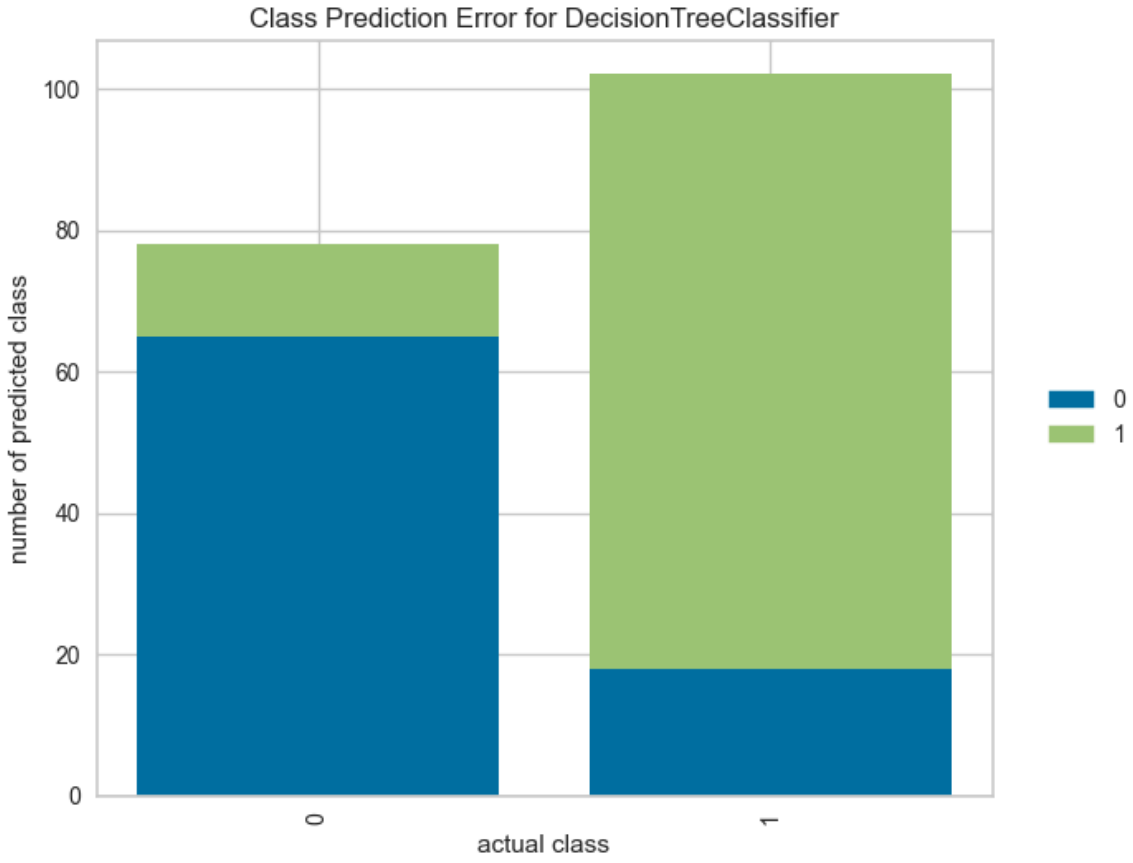
In [328…

```
ConfusionMatrixDisplay.from_estimator(pipe_model, X_test, y_test, normalize='tr
```



In [329…

```
from yellowbrick.classifier import ClassPredictionError
visualizer = ClassPredictionError(pipe_model)
# Fit the training data to the visualizer
visualizer.fit(X_train, y_train)
# Evaluate the model on the test data
visualizer.score(X_test, y_test)
# Draw visualization
visualizer.poof();
```



In [331…
```
eval_metric(pipe_model, X_train, y_train, X_test, y_test, "DT")
```

```
DT Test_Set
[[65 18]
 [13 84]]
              precision    recall  f1-score   support

           0       0.83      0.78      0.81        83
           1       0.82      0.87      0.84        97

    accuracy                           0.83       180
   macro avg       0.83      0.82      0.83       180
weighted avg       0.83      0.83      0.83       180


DT Train_Set
[[367   0]
 [  0 353]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       367
           1       1.00      1.00      1.00       353

    accuracy                           1.00       720
   macro avg       1.00      1.00      1.00       720
weighted avg       1.00      1.00      1.00       720
```

Cross Validation (CV)

In [333…
```
operations = [("DT_model", DecisionTreeClassifier())]
model = Pipeline(steps=operations)
```

```
scores = cross_validate(
    model,
    X_train,
    y_train,
    scoring=["accuracy"],
    cv=10,
    return_train_score=True)
df_scores = pd.DataFrame(scores, index=range(1, 11))
df_scores
```

Out[333...

	fit_time	score_time	test_accuracy	train_accuracy
1	0.006001	0.002000	0.861111	1.0
2	0.003844	0.002902	0.805556	1.0
3	0.005000	0.004584	0.763889	1.0
4	0.007856	0.000996	0.833333	1.0
5	0.003323	0.002002	0.791667	1.0
6	0.002992	0.001002	0.847222	1.0
7	0.004097	0.000000	0.847222	1.0
8	0.004168	0.001000	0.875000	1.0
9	0.002998	0.001909	0.694444	1.0
10	0.003997	0.000000	0.861111	1.0

In [334...
```
df_scores.mean()[2:]
```

Out[334...
```
test_accuracy     0.818056
train_accuracy    1.000000
dtype: float64
```

Overfitting Risk and Features Importance in Decision Trees

In [336...
```
pipe_model["DT_model"].feature_importances_
```

Out[336...
```
array([0.03518487, 0.14538518, 0.06489821, 0.07462943, 0.03788692,
       0.06096111, 0.58105427])
```

In [338...
```
X_train.head(1)
```

Out[338...

	Area	MajorAxisLength	MinorAxisLength	Eccentricity	ConvexArea	Extent
722	126149	478.877101	345.919972	0.691521	134778	0.624908

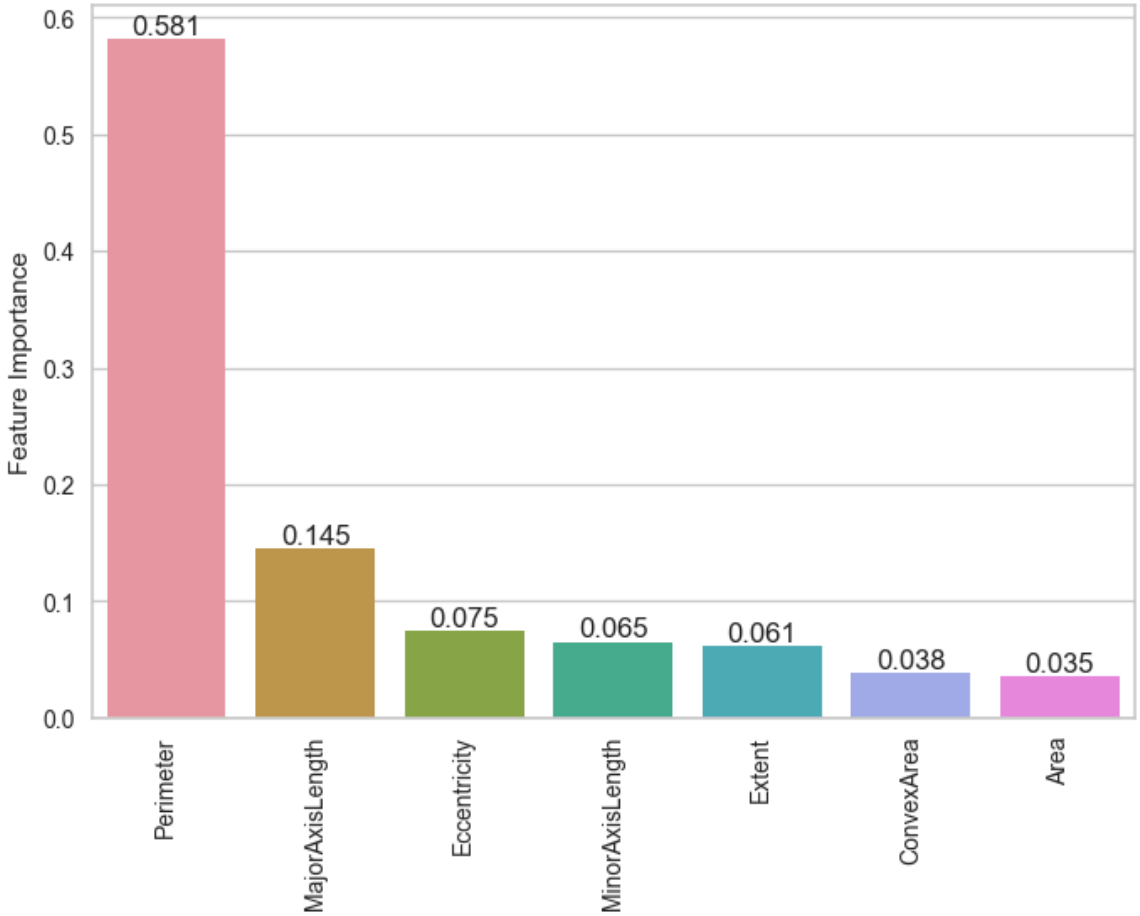
In [339...
```
df_fi = pd.DataFrame(data=pipe_model["DT_model"].feature_importances_,
                     index=X.columns,
                     columns=["Feature Importance"])
df_fi = df_fi.sort_values("Feature Importance", ascending=False)
df_fi
```

Out[339…

	Feature Importance
Perimeter	0.581054
MajorAxisLength	0.145385
Eccentricity	0.074629
MinorAxisLength	0.064898
Extent	0.060961
ConvexArea	0.037887
Area	0.035185

In [340…

```
ax = sns.barplot(x=df_fi.index, y='Feature Importance', data=df_fi)
ax.bar_label(ax.containers[0], fmt="%.3f")
plt.xticks(rotation=90)
plt.show()
```



In [407…

```
X.head()
```

Out[407…

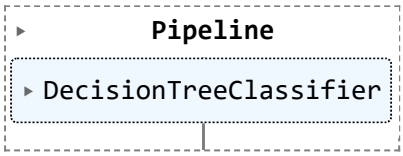
	Area	MajorAxisLength	MinorAxisLength	Eccentricity	ConvexArea	Extent	Peri
0	87524	442.246011	253.291155	0.819738	90546	0.758651	11
1	75166	406.690687	243.032436	0.801805	78789	0.684130	11
2	90856	442.267048	266.328318	0.798354	93717	0.637613	12
3	45928	286.540559	208.760042	0.684989	47336	0.699599	8
4	79408	352.190770	290.827533	0.564011	81463	0.792772	10

In [408…
```
X2 = X.drop(columns = ["Perimeter"])
```

In [409…
```
X_train2, X_test2, y_train2, y_test2 = train_test_split(X2,
                                                        y,
                                                        test_size=0.2,
                                                        random_state=10)
```

In [413…
```
operations = [("DT_model", DecisionTreeClassifier(random_state=101))]
pipe_model2 = Pipeline(steps=operations)
pipe_model2.fit(X_train2, y_train2)
```

Out[413…



In [414…
```
eval_metric(pipe_model2, X_train2, y_train2, X_test2, y_test2, "DT_drop")
```

```
DT_drop Test_Set
[[70 13]
 [18 79]]
              precision    recall  f1-score   support

           0       0.80      0.84      0.82        83
           1       0.86      0.81      0.84        97

    accuracy                           0.83       180
   macro avg       0.83      0.83      0.83       180
weighted avg       0.83      0.83      0.83       180


DT_drop Train_Set
[[367   0]
 [  0 353]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       367
           1       1.00      1.00      1.00       353

    accuracy                           1.00       720
   macro avg       1.00      1.00      1.00       720
weighted avg       1.00      1.00      1.00       720
```

In [417…
```
operations = [("DT_model", DecisionTreeClassifier(random_state=101))]
model = Pipeline(steps=operations)
```

```
scores = cross_validate(model,
                        X_train2,
                        y_train2,
                        scoring=["accuracy"],
                        cv=10,
                        return_train_score=True)
df_scores = pd.DataFrame(scores, index=range(1, 11))
df_scores
```

Out[417…

	fit_time	score_time	test_accuracy	train_accuracy
1	0.003106	0.001003	0.833333	1.0
2	0.003070	0.001006	0.833333	1.0
3	0.003557	0.002790	0.777778	1.0
4	0.004232	0.002239	0.763889	1.0
5	0.004511	0.001007	0.736111	1.0
6	0.002996	0.001003	0.791667	1.0
7	0.003485	0.001001	0.833333	1.0
8	0.001995	0.000995	0.805556	1.0
9	0.002136	0.000997	0.680556	1.0
10	0.003001	0.001066	0.791667	1.0

In [418…

```
decision_tree_model = pipe_model2.named_steps["DT_model"]
feature_importances = decision_tree_model.feature_importances_
df2_fi = pd.DataFrame(data=feature_importances, index=X_train2.columns, columns
df2_fi = df2_fi.sort_values(by="Feature Importance", ascending=False)
df2_fi
```

Out[418…

	Feature Importance
MajorAxisLength	0.659583
Extent	0.108328
Area	0.078781
Eccentricity	0.058278
ConvexArea	0.055832
MinorAxisLength	0.039198

Find Best Parameters

In [395…

```
operations = [("DT_model", DecisionTreeClassifier(random_state=101))]

model = Pipeline(steps=operations)
```

In [397…

```
param_grid = {
    "DT_model__splitter": ["best", "random"],
    "DT_model__max_features": [None, "auto", "sqrt", "log2", 2, 3, 4, 5, 6, 7],
```

```
    "DT_model__max_depth": [None, 2, 3, 4, 5, 6],
    "DT_model__min_samples_leaf": [1, 2, 3, 4],
    "DT_model__min_samples_split": [2, 3, 4],
    "DT_model__criterion": ["gini", "entropy"]
}
```

In [398…
```
grid_model = GridSearchCV(estimator=model,
                          param_grid=param_grid,
                          scoring='f1_micro',
                          cv=10,
                          n_jobs = -1,
                          return_train_score=True)
```

In [399…
```
grid_model.fit(X_train, y_train)
```

Out[399…
GridSearchCV
estimator: Pipeline
DecisionTreeClassifier

In [400…
```
grid_model.best_index_
```

Out[400… 360

In [401…
```
pd.DataFrame(grid_model.cv_results_).loc[360, ["mean_test_score", "mean_train_s
```

Out[401…
```
mean_test_score     0.866667
mean_train_score    0.869599
Name: 360, dtype: object
```

In [402…
```
grid_model.best_score_
```

Out[402… 0.8666666666666666

In [403…
```
grid_model.best_params_
```

Out[403…
```
{'DT_model__criterion': 'gini',
 'DT_model__max_depth': 2,
 'DT_model__max_features': 3,
 'DT_model__min_samples_leaf': 1,
 'DT_model__min_samples_split': 2,
 'DT_model__splitter': 'best'}
```

In [404…
```
grid_model.best_estimator_
```

Out[404…
Pipeline
DecisionTreeClassifier

In [427…
```
eval_metric(grid_model, X_train, y_train, X_test, y_test, "DT_best")
```

```
DT_best Test_Set
[[70 13]
 [ 9 88]]
              precision    recall  f1-score   support

           0       0.89      0.84      0.86        83
           1       0.87      0.91      0.89        97

    accuracy                           0.88       180
   macro avg       0.88      0.88      0.88       180
weighted avg       0.88      0.88      0.88       180


DT_best Train_Set
[[325  42]
 [ 52 301]]
              precision    recall  f1-score   support

           0       0.86      0.89      0.87       367
           1       0.88      0.85      0.86       353

    accuracy                           0.87       720
   macro avg       0.87      0.87      0.87       720
weighted avg       0.87      0.87      0.87       720
```

In [406…

```
operations = [("DT_model",
               DecisionTreeClassifier(max_depth=2,
                                      min_samples_split=2,
                                      min_samples_leaf=1,
                                      random_state=101,
                                      max_features=3
                                     ))]
scoring = ["accuracy", "precision_micro", "recall_micro", "f1_micro"]
model = Pipeline(steps=operations)
scores = cross_validate(model,
                        X_train,
                        y_train,
                        scoring=scoring,
                        cv=10,
                        return_train_score=True)
df_scores = pd.DataFrame(scores, index=range(1, 11))
df_scores.mean()[2:]
```

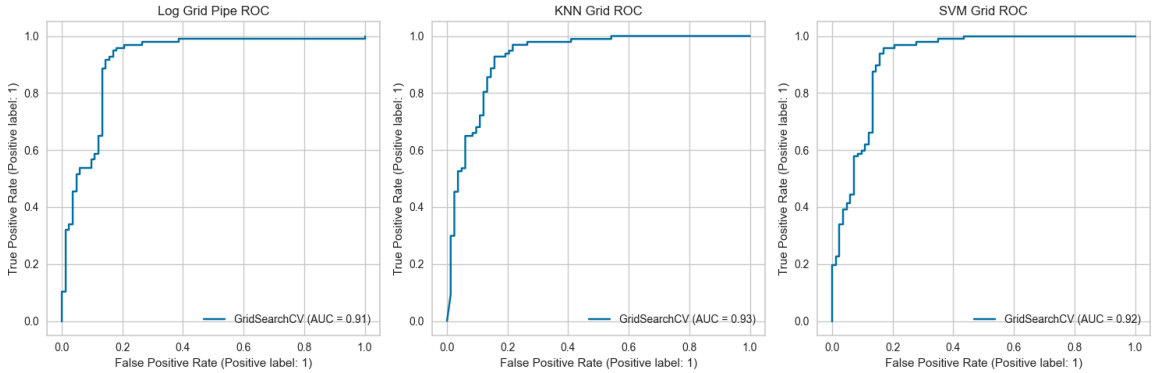
Out[406…

```
test_accuracy            0.866667
train_accuracy           0.869599
test_precision_micro     0.866667
train_precision_micro    0.869599
test_recall_micro        0.866667
train_recall_micro       0.869599
test_f1_micro            0.866667
train_f1_micro           0.869599
dtype: float64
```

Compare Models Performance

In [331…

```
fig, ax = plt.subplots(1, 3, figsize=(15, 5))

log_grid_ROC.plot(ax=ax[0])
ax[0].set_title("Log Grid Pipe ROC")
knn_grid_ROC.plot(ax=ax[1])
ax[1].set_title("KNN Grid ROC")
svm_grid_ROC.plot(ax=ax[2])
ax[2].set_title("SVM Grid ROC")

plt.tight_layout()
plt.show()
```




```
logisticgrid Test_Set
[[70 13]
 [ 7 90]]
              precision    recall  f1-score   support

           0       0.91      0.84      0.88        83
           1       0.87      0.93      0.90        97

    accuracy                           0.89       180
   macro avg       0.89      0.89      0.89       180
weighted avg       0.89      0.89      0.89       180


logisticgrid Train_Set
[[309  58]
 [ 40 313]]
              precision    recall  f1-score   support

           0       0.89      0.84      0.86       367
           1       0.84      0.89      0.86       353

    accuracy                           0.86       720
   macro avg       0.86      0.86      0.86       720
weighted avg       0.86      0.86      0.86       720
```

```
WITH K=29

knn_grid Test_Set
[[71 12]
 [ 7 90]]
              precision    recall  f1-score   support

           0       0.91      0.86      0.88        83
           1       0.88      0.93      0.90        97

    accuracy                           0.89       180
   macro avg       0.90      0.89      0.89       180
weighted avg       0.90      0.89      0.89       180


knn_grid Train_Set
[[306  61]
 [ 40 313]]
              precision    recall  f1-score   support

           0       0.88      0.83      0.86       367
           1       0.84      0.89      0.86       353

    accuracy                           0.86       720
   macro avg       0.86      0.86      0.86       720
weighted avg       0.86      0.86      0.86       720


svm_grid Test_Set
[[68 15]
 [ 4 93]]
              precision    recall  f1-score   support

           0       0.94      0.82      0.88        83
           1       0.86      0.96      0.91        97

    accuracy                           0.89       180
   macro avg       0.90      0.89      0.89       180
weighted avg       0.90      0.89      0.89       180


svm_grid Train_Set
[[305  62]
 [ 34 319]]
              precision    recall  f1-score   support

           0       0.90      0.83      0.86       367
           1       0.84      0.90      0.87       353

    accuracy                           0.87       720
   macro avg       0.87      0.87      0.87       720
weighted avg       0.87      0.87      0.87       720
```

```
DT_best Test_Set
[[70 13]
 [ 9 88]]
              precision    recall  f1-score   support

           0       0.89      0.84      0.86        83
           1       0.87      0.91      0.89        97

    accuracy                           0.88       180
   macro avg       0.88      0.88      0.88       180
weighted avg       0.88      0.88      0.88       180


DT_best Train_Set
[[325  42]
 [ 52 301]]
              precision    recall  f1-score   support

           0       0.86      0.89      0.87       367
           1       0.88      0.85      0.86       353

    accuracy                           0.87       720
   macro avg       0.87      0.87      0.87       720
weighted avg       0.87      0.87      0.87       720
```

Final Model and Model Deployment

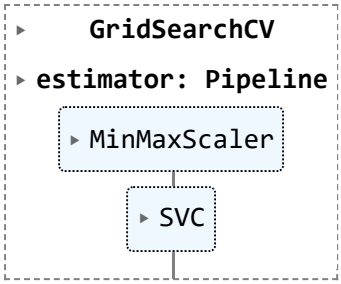
In [435…
```
param_grid = {'SVC__C': [0.5],
              'SVC__gamma':["scale"],
              'SVC__kernel':['rbf'],
              'SVC__class_weight': ["balanced"]}
```

In [437…
```
operations = [("scaler", MinMaxScaler()), ("SVC", SVC(probability=True))]

pipe_model = Pipeline(steps=operations)

final_model = GridSearchCV(pipe_model,
                           param_grid,
                           scoring="recall",
                           cv=5,
                           return_train_score=True,
                           n_jobs=-1).fit(X, y)
```

In [440…
```
import pickle
pickle.dump(final_model, open("final_pipe_model", "wb"))
```

In [441…
```
new_model = pickle.load(open("final_pipe_model", "rb"))
new_model
```

Out[441…



Prediction

In [444…
```
df.loc[[182, 725, 166]]
```

Out[444…

	Area	MajorAxisLength	MinorAxisLength	Eccentricity	ConvexArea	Extent	P
182	63968	333.012421	247.888546	0.667754	65403	0.757000	
725	79397	434.994082	235.519493	0.840745	83074	0.696117	
166	51683	325.239868	208.914825	0.766419	54510	0.728443	

In [453…
```
my_dict = {
    'Area': [63968, 79397, 51683],
    'MajorAxisLength': [333.012421, 434.994082, 325.239868],
    'MinorAxisLength': [247.888546, 235.519493, 208.914825],
    'Eccentricity': [0.667754, 0.840745, 0.766419],
    'ConvexArea': [65403, 83074, 54510],
    'Extent': [0.757, 0.696117, 0.728443],
    'Perimeter': [953.445, 1148.633, 928.217]
}
```

In [454…
```
sample = pd.DataFrame(my_dict)
sample
```

Out[454…

	Area	MajorAxisLength	MinorAxisLength	Eccentricity	ConvexArea	Extent	Peri
0	63968	333.012421	247.888546	0.667754	65403	0.757000	9
1	79397	434.994082	235.519493	0.840745	83074	0.696117	11
2	51683	325.239868	208.914825	0.766419	54510	0.728443	9

In [455…
```
new_model.predict(sample)
```

Out[455…
```
array([1, 0, 1])
```

In [457…
```
new_model.decision_function(sample)
# The SVC prediction operation is performed using the decision_function method.
# Positive values indicate class 1, while negative values indicate class 0.
# The larger the absolute value, the farther the observation is from the hyperp
```

Out[457…
```
array([ 1.69650528, -0.23482387,  1.51836999])
```

📂 Conclusion

In [ ]:
```
#   SVM grid        accuracy :  0.89   roc_auc=0.92    19  wrong prediction
```

When deciding which of these models to choose, there are several important points to consider:

1. **Accuracy**
   - Logistic Regression: 0.89
   - KNN: 0.89
   - SVM: 0.89
   - Decision Tree: 0.88
2. **AUC (Area Under the ROC Curve)**
   - Logistic Regression: 0.91
   - KNN: 0.93
   - SVM: 0.92
   - Decision Tree: AUC value is lower compared to the others.
3. **Other Metrics (Precision, Recall, F1-score)**
   - Logistic Regression and KNN models generally have balanced precision, recall, and F1-score values.
   - The SVM model has a higher F1-score for class 1 compared to the others.
   - The Decision Tree model generally has lower precision and recall values compared to the other models.
4. **Overfitting**
   - Logistic Regression: Training set accuracy: 0.86, Test set accuracy: 0.89
   - KNN: Training set accuracy: 0.86, Test set accuracy: 0.89
   - SVM: Training set accuracy: 0.87, Test set accuracy: 0.89
   - Decision Tree: Training set accuracy: 0.87, Test set accuracy: 0.88

Decision

- SVM and KNN models generally have the highest accuracy and AUC values, so these two models can be evaluated first. However:
  - The SVM model's ROC AUC value is 0.92, which shows slightly higher performance.
  - The KNN model's ROC AUC value is 0.93, which is also quite good.
- If maintaining the balance between classes and overall performance is important, you may prefer the SVM model because it has balanced and generally high precision, recall, and F1-score values.
- If the Area Under the ROC Curve (AUC) is your most important performance criterion, the KNN model can also be preferred.

In my work, maintaining the balance between classes and overall performance is important, so I chose the SVM model.

THANK YOU

If you want to be the first to be informed about new projects, please do not forget to follow us - by Fatma Nur AZMAN

Fatmanurazman.com | Linkedin | Github | Kaggle | Tableau