1. Import Libraries

In [36]:

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
from xgboost import XGBClassifier
RANDOM_STATE = 55 # for reproducibility
```

2. Data Exploration

In [2]:

```
# Read the excel file and check first few lines
pumpkin = pd.read_excel("/Users/xinyizhang/Desktop/Weill Cornell Medicine/自学
pumpkin.head()
```

Out[2]:

	Area	Perimeter	Major_Axis_Length	Minor_Axis_Length	Convex_Area	Equiv_Diameter	E
0	56276	888.242	326.1485	220.2388	56831	267.6805	
1	76631	1068.146	417.1932	234.2289	77280	312.3614	
2	71623	1082.987	435.8328	211.0457	72663	301.9822	
3	66458	992.051	381.5638	222.5322	67118	290.8899	
4	66107	998.146	383.8883	220.4545	67117	290.1207	

In [3]:

```
pumpkin.info()
pumpkin.shape
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2500 entries, 0 to 2499
Data columns (total 13 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   Area               2500 non-null   int64  
 1   Perimeter          2500 non-null   float64
 2   Major_Axis_Length  2500 non-null   float64
 3   Minor_Axis_Length  2500 non-null   float64
 4   Convex_Area        2500 non-null   int64  
 5   Equiv_Diameter     2500 non-null   float64
 6   Eccentricity       2500 non-null   float64
 7   Solidity           2500 non-null   float64
 8   Extent             2500 non-null   float64
 9   Roundness          2500 non-null   float64
 10  Aspect_Ration      2500 non-null   float64
 11  Compactness        2500 non-null   float64
 12  Class              2500 non-null   object 
dtypes: float64(10), int64(2), object(1)
memory usage: 254.0+ KB
```

Out[3]:
```
(2500, 13)
```

The data has 2500 rows and 13 columns, among which the first 12 columns are features and the last column is the target variable

In [4]:
```
# Summarize the unique labels in the Class column
class_summary = pumpkin['Class'].value_counts()
class_summary
```

Out[4]:
```
Class
Çerçevelik       1300
Ürgüp Sivrisi    1200
Name: count, dtype: int64
```

In [5]:
```
# Summarize the numeric pumpkin data
pumpkin_summary = pumpkin.describe()
pumpkin_summary
```

Out[5]:

	Area	Perimeter	Major_Axis_Length	Minor_Axis_Length	Convex_Area	E
count	2500.000000	2500.000000	2500.000000	2500.000000	2500.000000	
mean	80658.220800	1130.279015	456.601840	225.794921	81508.084400	
std	13664.510228	109.256418	56.235704	23.297245	13764.092788	
min	47939.000000	868.485000	320.844600	152.171800	48366.000000	
25%	70765.000000	1048.829750	414.957850	211.245925	71512.000000	
50%	79076.000000	1123.672000	449.496600	224.703100	79872.000000	
75%	89757.500000	1203.340500	492.737650	240.672875	90797.750000	
max	136574.000000	1559.450000	661.911300	305.818000	138384.000000	

In [6]:
```
# Describe the data by unique pumpkin seeds classes
class_description = pumpkin.groupby('Class').describe()
class_description
```

Out[6]:

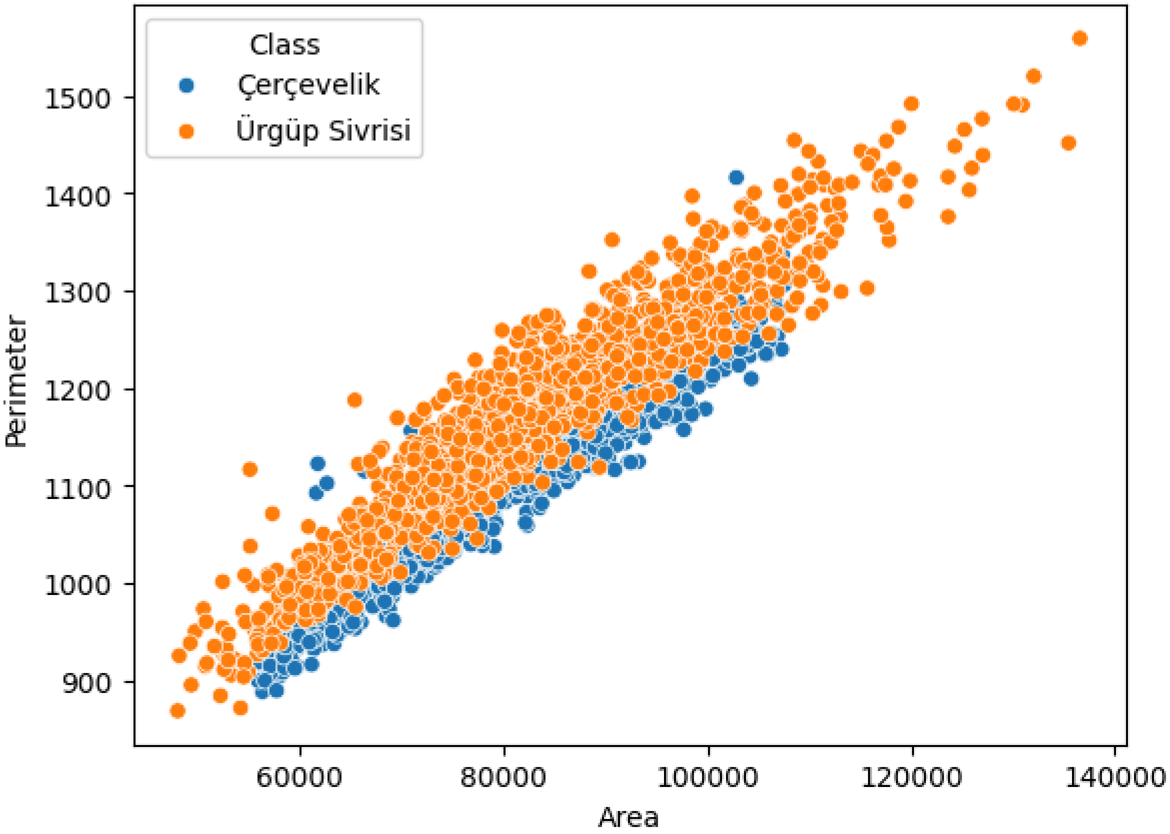
								Are
	count	mean	std	min	25%	50%	75%	ma
Class								
Çerçevelik	1300.0	78423.154615	11246.499728	55811.0	69777.75	76718.5	86277.75	107476.
Ürgüp Sivrisi	1200.0	83079.542500	15519.323847	47939.0	72482.50	81657.0	93815.75	136574.

2 rows × 96 columns

Visualization

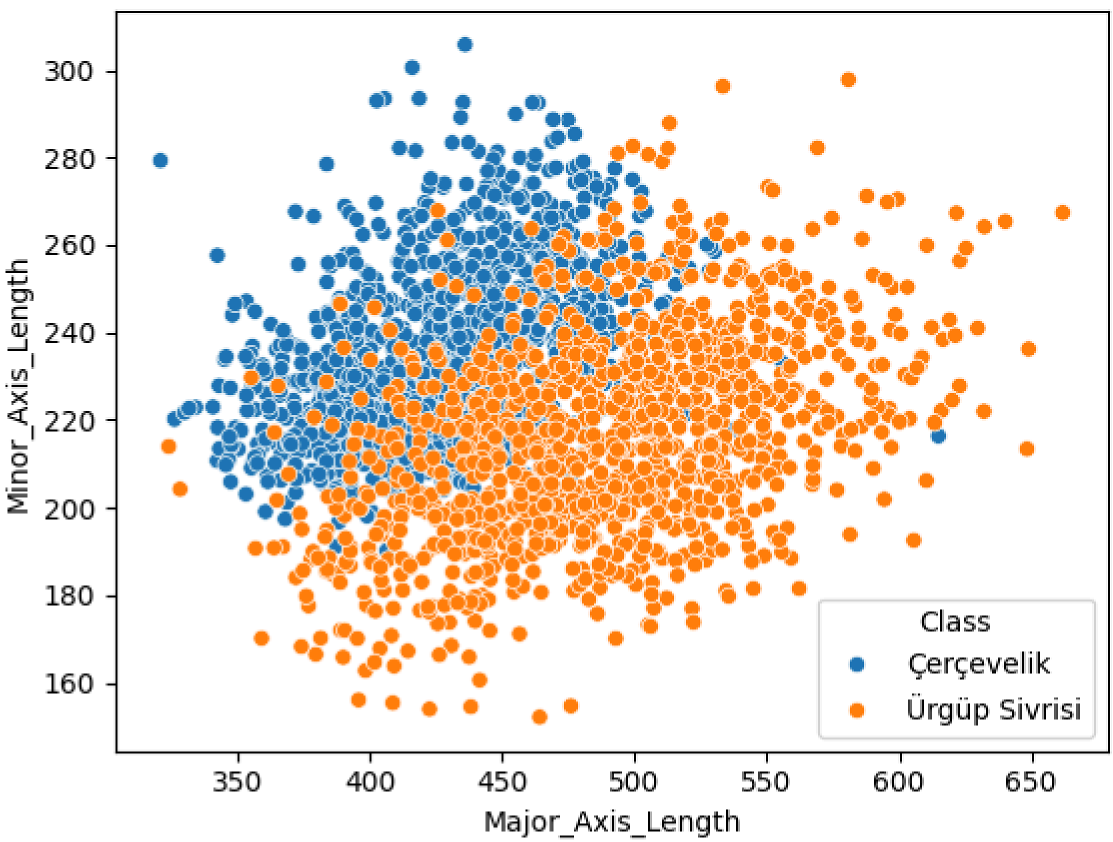
In [7]:
```
sns.scatterplot(data = pumpkin, x = 'Area', y = 'Perimeter', hue = 'Class')
```

Out[7]:
```
<Axes: xlabel='Area', ylabel='Perimeter'>
```



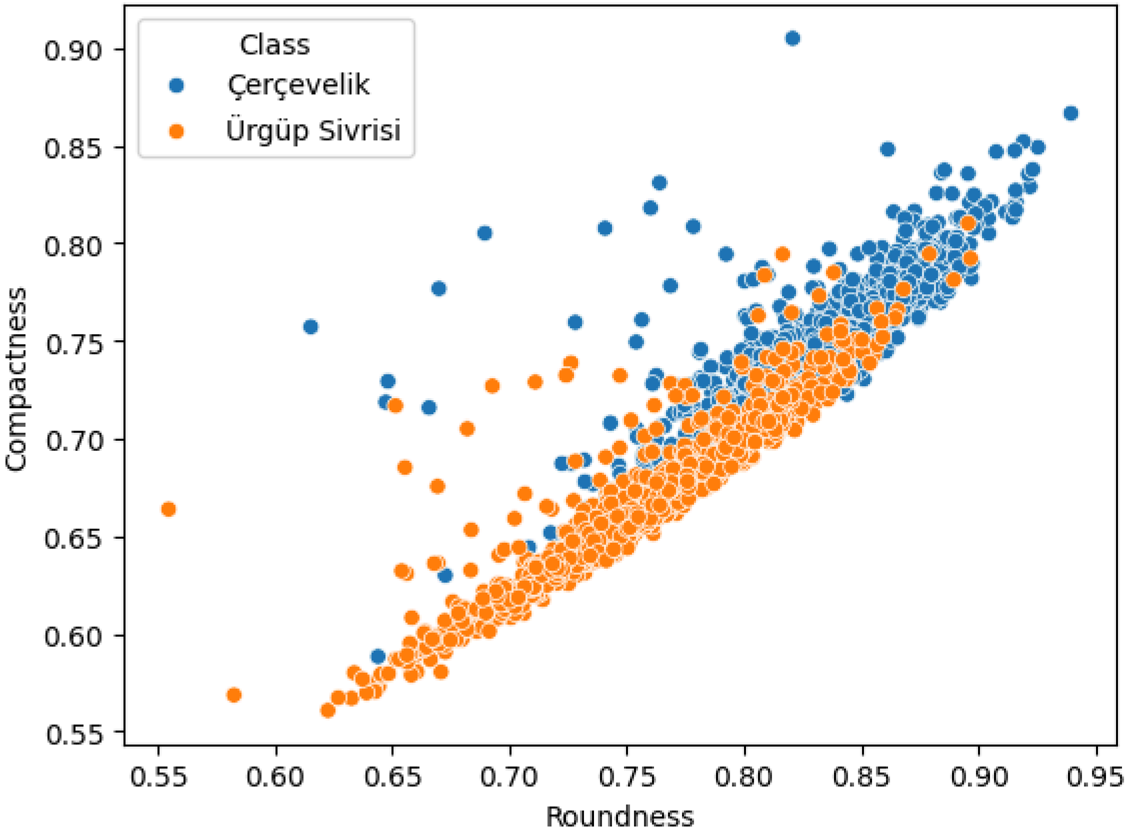
In [8]:
```
sns.scatterplot(data = pumpkin, x = 'Major_Axis_Length', y = 'Minor_Axis_Leng
```

Out[8]: <Axes: xlabel='Major_Axis_Length', ylabel='Minor_Axis_Length'>



In [9]:
```
sns.scatterplot(data = pumpkin, x = 'Roundness', y = 'Compactness', hue = 'Cl
```

Out[9]: <Axes: xlabel='Roundness', ylabel='Compactness'>



3. Random Forest Building

In [10]:

```
features = [x for x in pumpkin.columns if x not in 'Class'] # Removing our ta
```

Split the data into training and test set

In [11]:

```
X_train, X_test, y_train, y_test = train_test_split(pumpkin[features], pumpki
```

In [12]:

```
print(f'Train samples: {len(X_train)}')
print(f'Validation samples: {len(X_test)}')
```

```
Train samples: 2000
Validation samples: 500
```

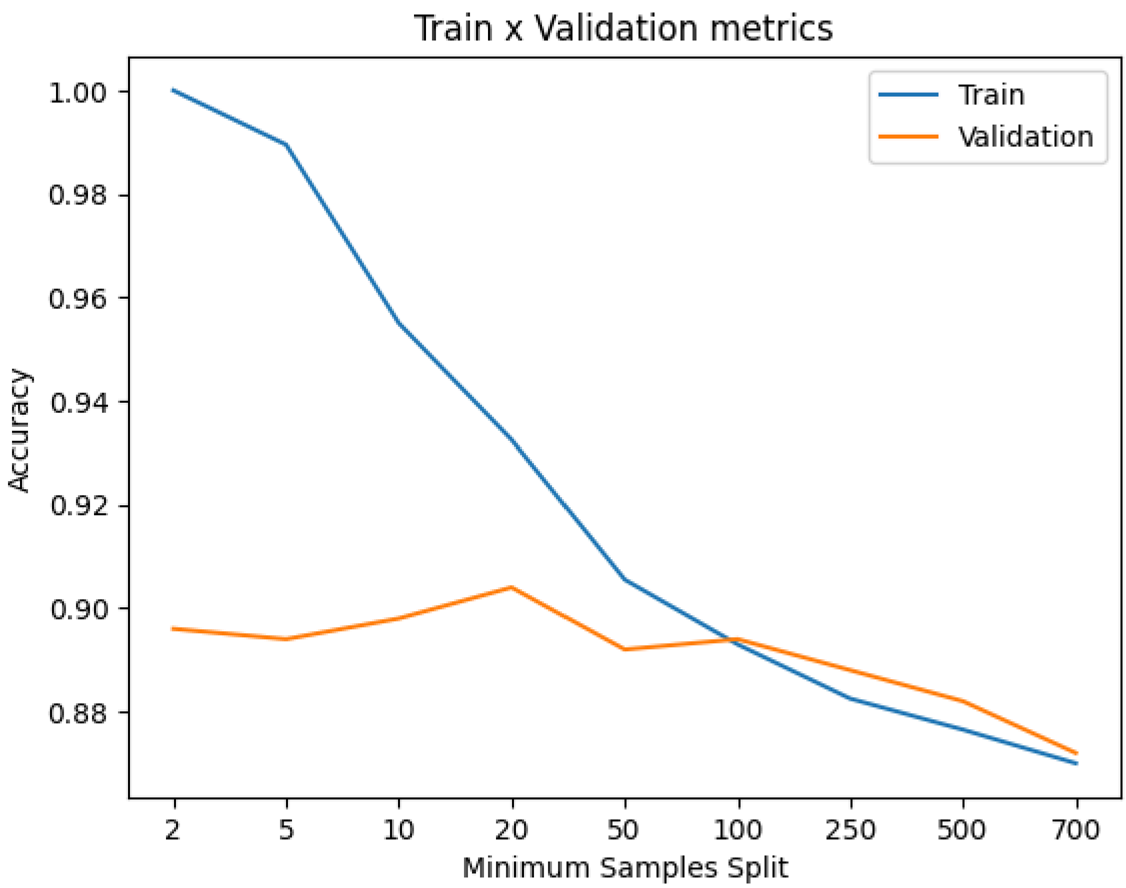
Choose the most suitable hyperparamter

In [13]:

```
# Minimum samples per leaf
min_samples_split_list = [2, 5, 10, 20, 50, 100, 250, 500, 700]

# Maximum depth of the tree
max_depth_list = [2, 4, 8, 16, 32, 64, None]

# Number of trees in the forest
n_list = [10, 50, 100, 200, 500]
```

In [14]:
```
accuracy_list_train = []
accuracy_list_test = []
for min_samples_split in min_samples_split_list:
    model = RandomForestClassifier(min_samples_split = min_samples_split,
                                   random_state = RANDOM_STATE).fit(X_train,y
    predictions_train = model.predict(X_train) # The predicted values for the
    predictions_test = model.predict(X_test) # The predicted values for the t
    accuracy_train = accuracy_score(predictions_train,y_train)
    accuracy_test = accuracy_score(predictions_test,y_test)
    accuracy_list_train.append(accuracy_train)
    accuracy_list_test.append(accuracy_test)

plt.title('Train x Validation metrics')
plt.xlabel('Minimum Samples Split')
plt.ylabel('Accuracy')
plt.xticks(ticks = range(len(min_samples_split_list)),labels=min_samples_spli
plt.plot(accuracy_list_train)
plt.plot(accuracy_list_test)
plt.legend(['Train','Validation'])
```

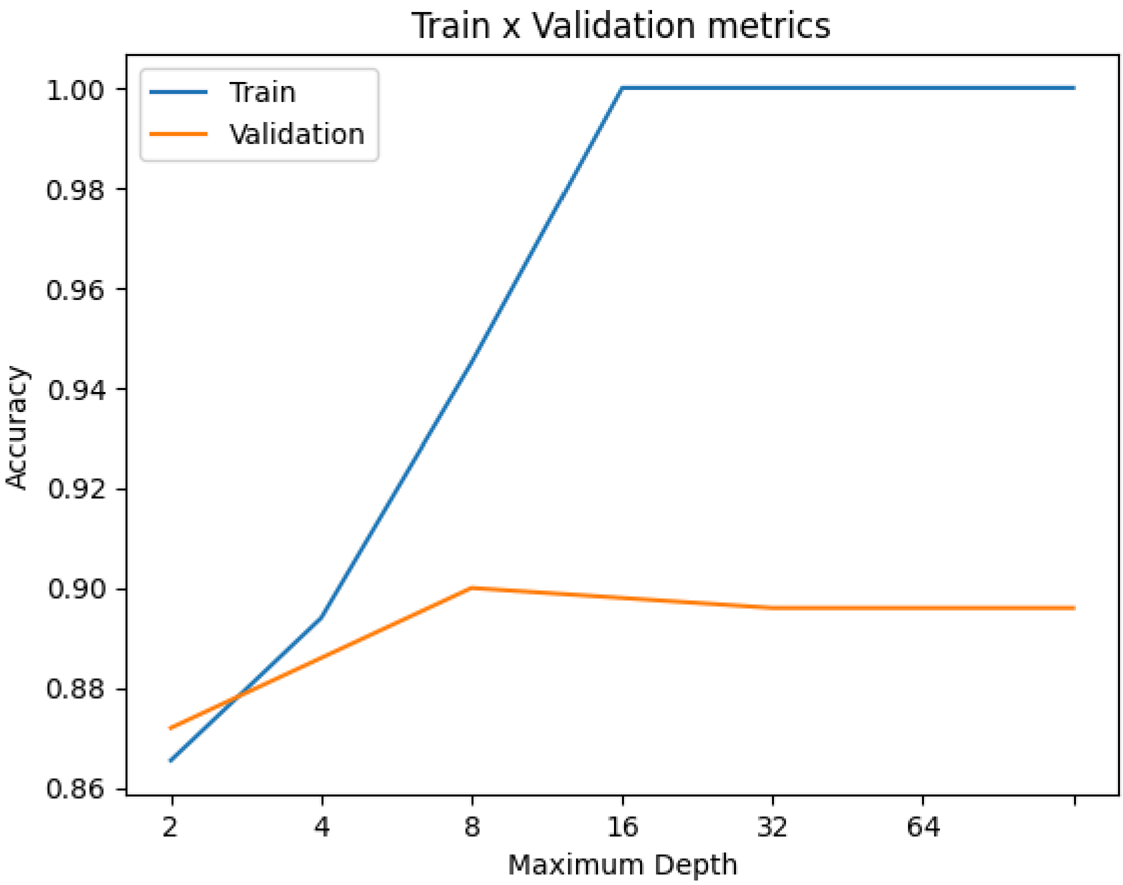
Out[14]: <matplotlib.legend.Legend at 0x17dc6cf40>



In [15]:
```
accuracy_list_train = []
accuracy_list_test = []
for max_depth in max_depth_list:
    model = RandomForestClassifier(max_depth = max_depth,
                                   random_state = RANDOM_STATE).fit(X_train,y
    predictions_train = model.predict(X_train) # The predicted values for the
    predictions_test = model.predict(X_test) # The predicted values for the t
    accuracy_train = accuracy_score(predictions_train,y_train)
    accuracy_test = accuracy_score(predictions_test,y_test)
    accuracy_list_train.append(accuracy_train)
    accuracy_list_test.append(accuracy_test)
```

```
plt.title('Train x Validation metrics')
plt.xlabel('Maximum Depth')
plt.ylabel('Accuracy')
plt.xticks(ticks = range(len(max_depth_list )),labels=max_depth_list)
plt.plot(accuracy_list_train)
plt.plot(accuracy_list_test)
plt.legend(['Train','Validation'])
```

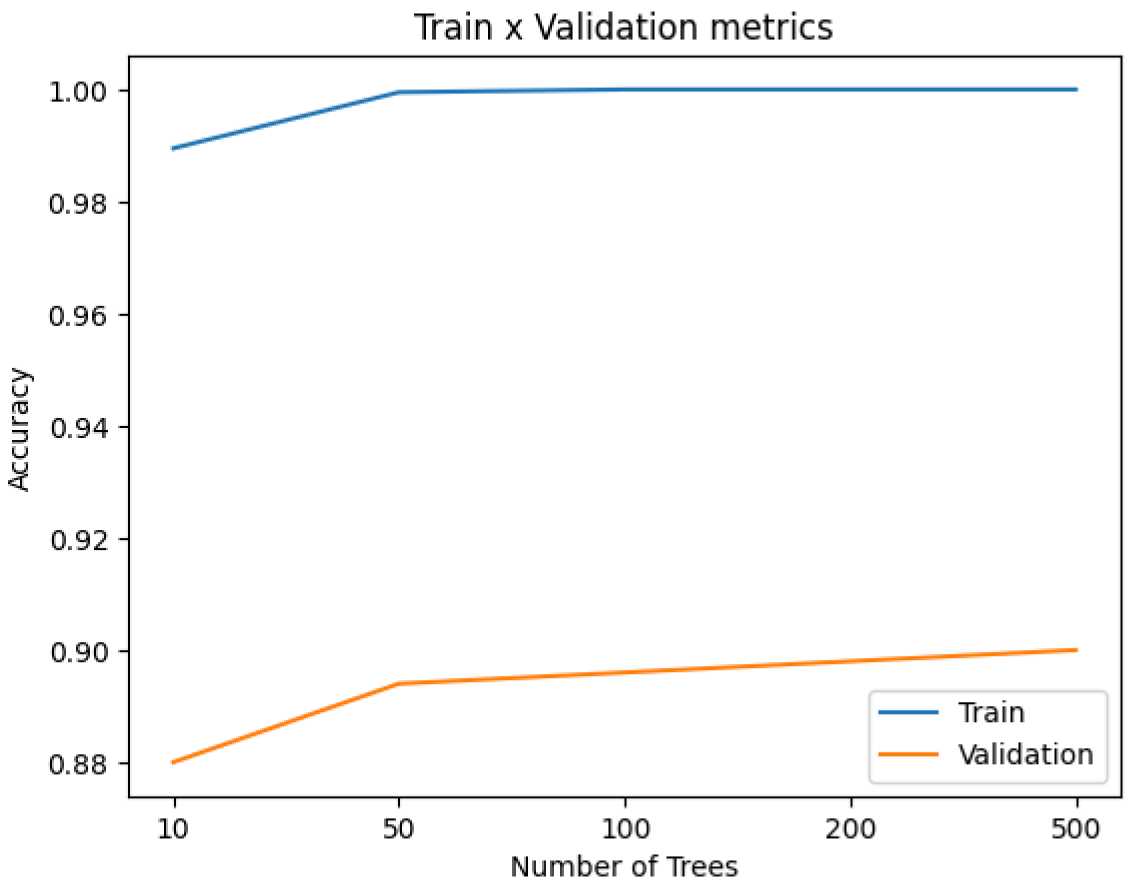
Out[15]: <matplotlib.legend.Legend at 0x17421c310>



In [16]:

```
accuracy_list_train = []
accuracy_list_test = []
for n_estimators in n_list:
    model = RandomForestClassifier(n_estimators = n_estimators,
                                   random_state = RANDOM_STATE).fit(X_train,y
    predictions_train = model.predict(X_train) # The predicted values for the
    predictions_test = model.predict(X_test) # The predicted values for the t
    accuracy_train = accuracy_score(predictions_train,y_train)
    accuracy_test = accuracy_score(predictions_test,y_test)
    accuracy_list_train.append(accuracy_train)
    accuracy_list_test.append(accuracy_test)

plt.title('Train x Validation metrics')
plt.xlabel('Number of Trees')
plt.ylabel('Accuracy')
plt.xticks(ticks = range(len(n_list)),labels=n_list)
plt.plot(accuracy_list_train)
plt.plot(accuracy_list_test)
plt.legend(['Train','Validation'])
```

Out[16]: <matplotlib.legend.Legend at 0x17fcb5a60>



In order to achieve the best result, choose the following numbers as hyperparamters: Maximum Sample Split: 20 Only try to split a node if there are at least 20 samples in it Maximum Depth: 8 Number of Trees: 100

Final Random Forest Model

In [17]:

```
random_forest_model = RandomForestClassifier(n_estimators = 100,
                                             max_depth = 8,
                                             min_samples_split = 20).fit(X_tr
```

Evaluation

Classification Report

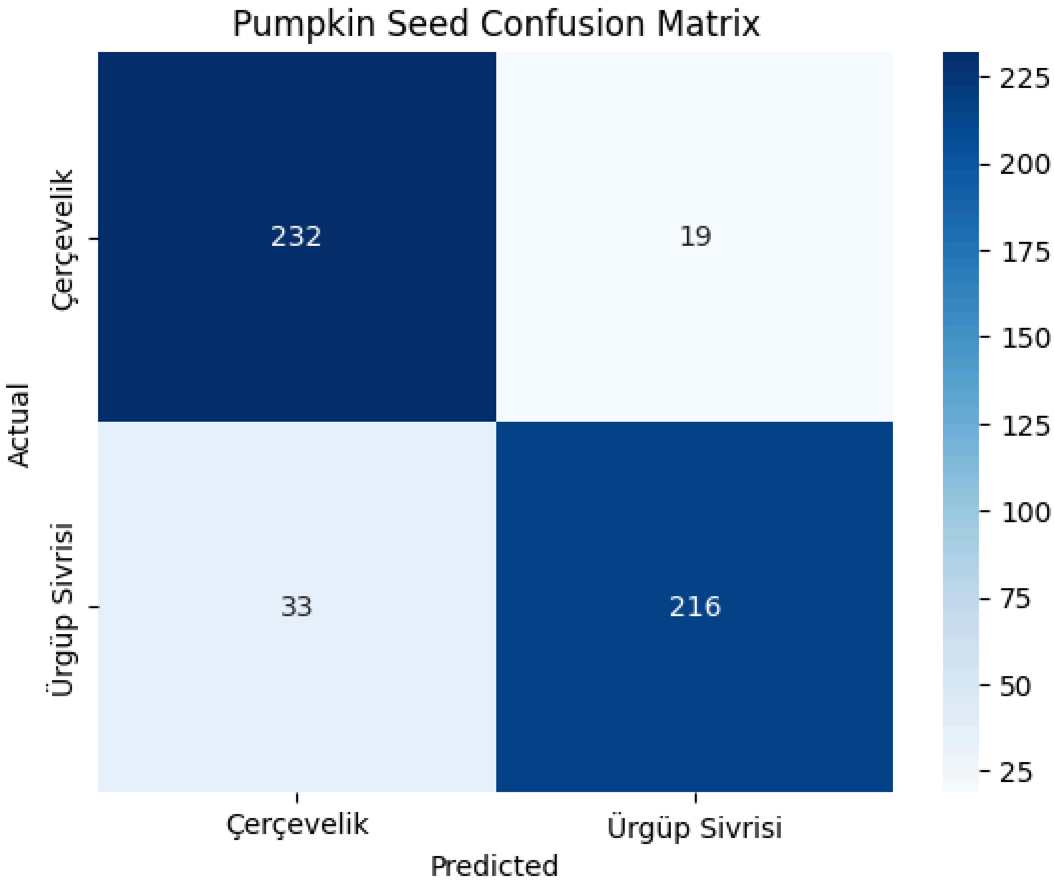
In [18]:

```
y_test_pred = random_forest_model.predict(X_test)
print(classification_report(y_test, y_test_pred))
```

```
               precision    recall  f1-score   support

   Çerçevelik       0.88      0.92      0.90       251
 Ürgüp Sivrisi       0.92      0.87      0.89       249

     accuracy                           0.90       500
    macro avg       0.90      0.90      0.90       500
 weighted avg       0.90      0.90      0.90       500
```

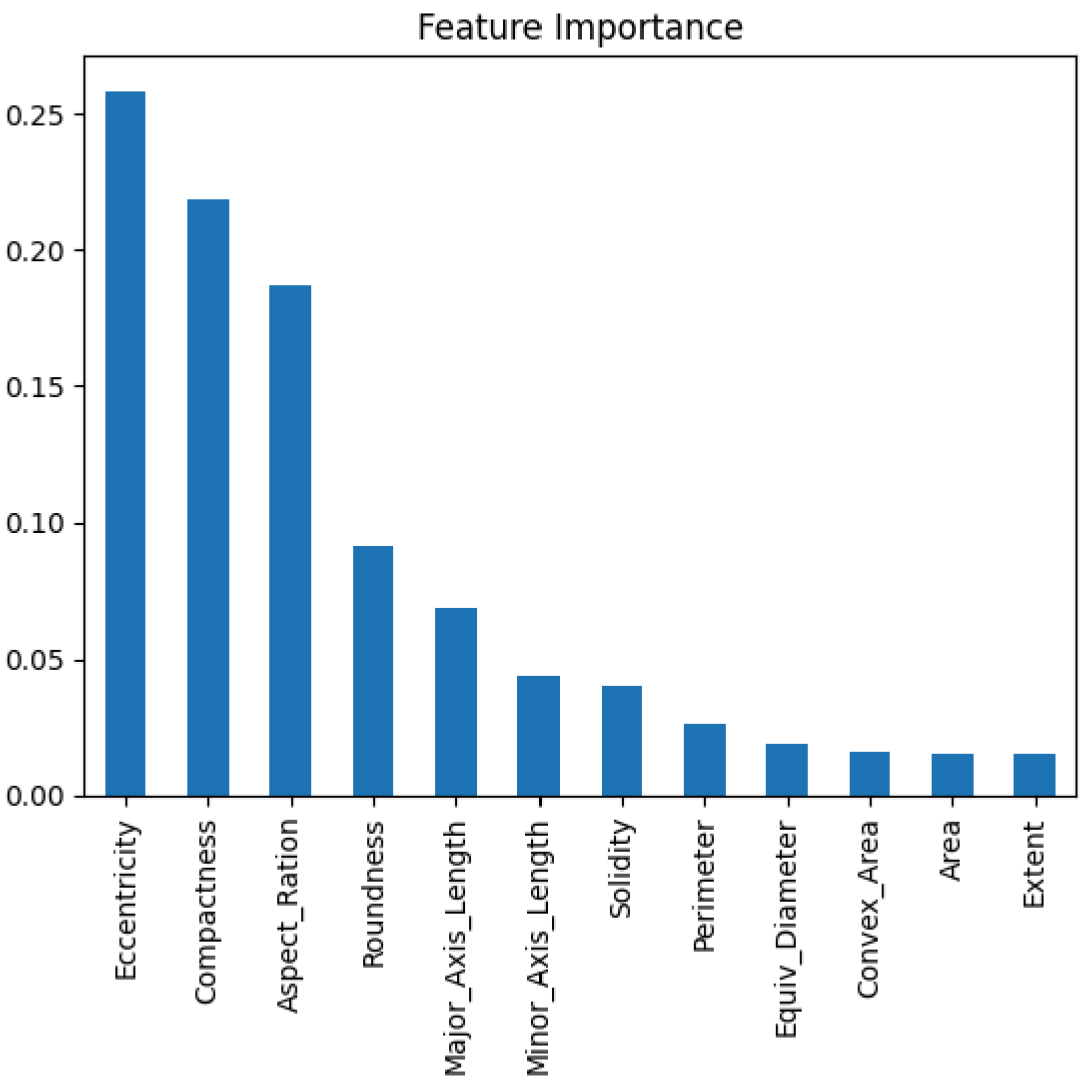
Confusion Matrix

In [19]:
```
class_labels = pumpkin['Class'].unique()
cm = confusion_matrix(y_test, y_test_pred)
sns.heatmap(cm, annot = True, fmt = 'd', cmap = 'Blues', xticklabels = class_
plt.title('Pumpkin Seed Confusion Matrix')
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```



Importance

In [43]:
```
importances = random_forest_model.feature_importances_
feat_importances = pd.Series(importances, index = features)
feat_importances.sort_values(ascending = False).plot(kind = 'bar')
plt.title("Feature Importance")
plt.show()
```



4. XGBoost

The boosting methods train several trees, but instead of them being uncorrelated to each other, now the trees are fit one after the other in order to minimize the error.

The model has the same parameters as a decision tree, plus the learning rate.

- The learning rate is the size of the step on the Gradient Descent method that the XGBoost uses internally to minimize the error on each train step.

Split the data into training and testing set

In [20]:
```
X_train, X_test, y_train, y_test = train_test_split(pumpkin[features], pumpki
```

In [21]:
```
# Encode labels
label_encoder = LabelEncoder()
y_train_encoded = label_encoder.fit_transform(y_train)
y_test_encoded = label_encoder.transform(y_test)
```

Choose the most suitable hyperparamters

In [45]:

```
# Define the learning rate
learning_rate = [0.1, 0.2, 0.3, 0.5, 0.7, 1.0]

# Define the maximum depth of the tree
max_depth_list = [2, 4, 8, 16, 32, 64, None]
```

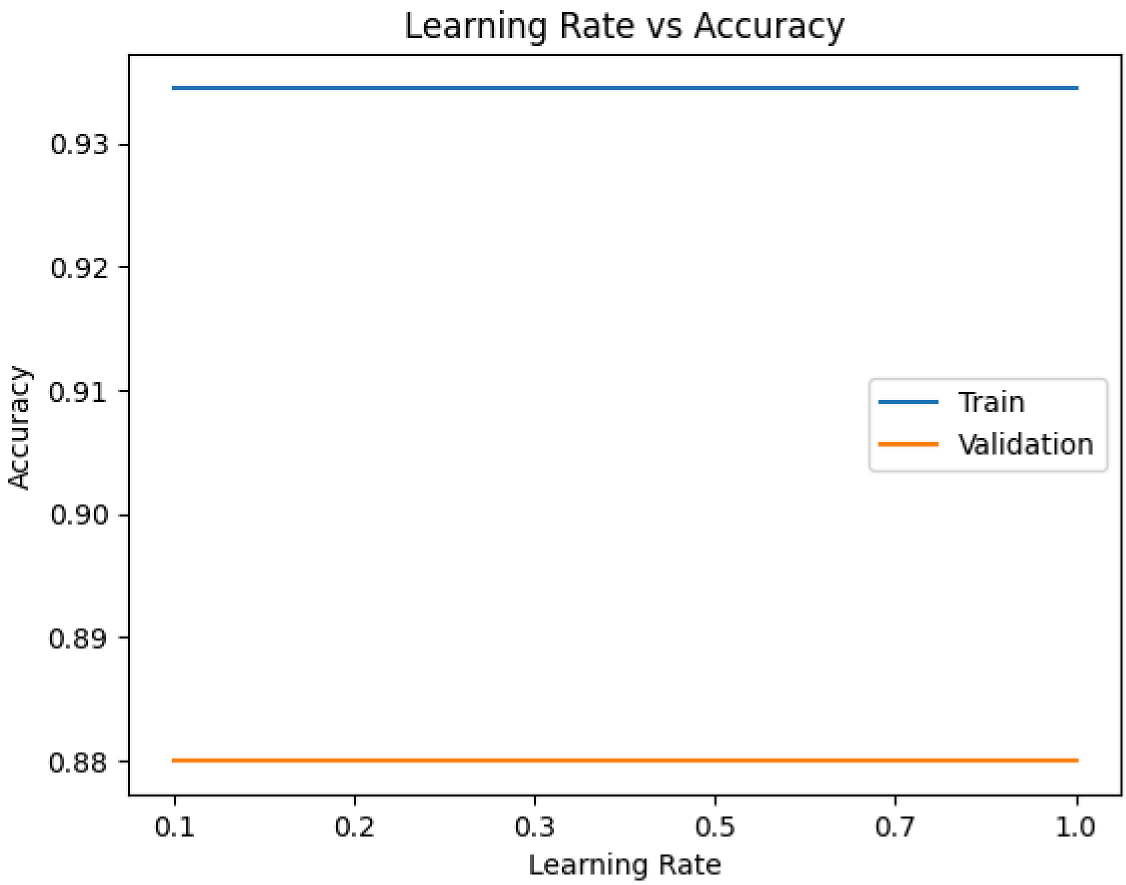
In [47]:

```
accuracy_list_test_xgb = []
accuracy_list_train_xgb = []
for lr in learning_rate:
    # Number of trees can be set to a higher value for XGBoost since model wi
    gb_model = XGBClassifier(n_estimators = 500, learning_rate = lr, verbosit
    xgb_model.fit(X_train, y_train_encoded, eval_set=[(X_test, y_test_encoded
    prediction_train = xgb_model.predict(X_train)
    prediction_test = xgb_model.predict(X_test)
    accuracy_train = accuracy_score(prediction_train, y_train_encoded)
    accuracy_test = accuracy_score(prediction_test, y_test_encoded)
    accuracy_list_train_xgb.append(accuracy_train)
    accuracy_list_test_xgb.append(accuracy_test)

plt.title('Learning Rate vs Accuracy')
plt.xlabel('Learning Rate')
plt.ylabel('Accuracy')
plt.xticks(ticks=range(len(learning_rate)), labels = learning_rate)
plt.plot(accuracy_list_train_xgb)
plt.plot(accuracy_list_test_xgb)
plt.legend(['Train', "Validation"])
plt.show()
```

```
[0]	validation_0-logloss:0.52957
[1]	validation_0-logloss:0.43944
[2]	validation_0-logloss:0.38977
[3]	validation_0-logloss:0.35359
[4]	validation_0-logloss:0.32719
[5]	validation_0-logloss:0.31539
[6]	validation_0-logloss:0.30277
[7]	validation_0-logloss:0.29750
[8]	validation_0-logloss:0.29406
[9]	validation_0-logloss:0.29429
[10]	validation_0-logloss:0.29286
[11]	validation_0-logloss:0.29258
[12]	validation_0-logloss:0.29295
[13]	validation_0-logloss:0.29271
[14]	validation_0-logloss:0.29479
[15]	validation_0-logloss:0.29615
[16]	validation_0-logloss:0.29605
[17]	validation_0-logloss:0.29656
[18]	validation_0-logloss:0.29605
[19]	validation_0-logloss:0.29830
[20]	validation_0-logloss:0.30036
[21]	validation_0-logloss:0.30155
[0]	validation_0-logloss:0.52957
[1]	validation_0-logloss:0.43944
[2]	validation_0-logloss:0.38977
[3]	validation_0-logloss:0.35359
[4]	validation_0-logloss:0.32719
[5]	validation_0-logloss:0.31539
[6]	validation_0-logloss:0.30277
[7]	validation_0-logloss:0.29750
[8]	validation_0-logloss:0.29406
[9]	validation_0-logloss:0.29429
[10]	validation_0-logloss:0.29286
```

```
[11]	validation_0-logloss:0.29258
[12]	validation_0-logloss:0.29295
[13]	validation_0-logloss:0.29271
[14]	validation_0-logloss:0.29479
[15]	validation_0-logloss:0.29615
[16]	validation_0-logloss:0.29605
[17]	validation_0-logloss:0.29656
[18]	validation_0-logloss:0.29605
[19]	validation_0-logloss:0.29830
[20]	validation_0-logloss:0.30036
[0]	validation_0-logloss:0.52957
[1]	validation_0-logloss:0.43944
[2]	validation_0-logloss:0.38977
[3]	validation_0-logloss:0.35359
[4]	validation_0-logloss:0.32719
[5]	validation_0-logloss:0.31539
[6]	validation_0-logloss:0.30277
[7]	validation_0-logloss:0.29750
[8]	validation_0-logloss:0.29406
[9]	validation_0-logloss:0.29429
[10]	validation_0-logloss:0.29286
[11]	validation_0-logloss:0.29258
[12]	validation_0-logloss:0.29295
[13]	validation_0-logloss:0.29271
[14]	validation_0-logloss:0.29479
[15]	validation_0-logloss:0.29615
[16]	validation_0-logloss:0.29605
[17]	validation_0-logloss:0.29656
[18]	validation_0-logloss:0.29605
[19]	validation_0-logloss:0.29830
[20]	validation_0-logloss:0.30036
[21]	validation_0-logloss:0.30155
[0]	validation_0-logloss:0.52957
[1]	validation_0-logloss:0.43944
[2]	validation_0-logloss:0.38977
[3]	validation_0-logloss:0.35359
[4]	validation_0-logloss:0.32719
[5]	validation_0-logloss:0.31539
[6]	validation_0-logloss:0.30277
[7]	validation_0-logloss:0.29750
[8]	validation_0-logloss:0.29406
[9]	validation_0-logloss:0.29429
[10]	validation_0-logloss:0.29286
[11]	validation_0-logloss:0.29258
[12]	validation_0-logloss:0.29295
[13]	validation_0-logloss:0.29271
[14]	validation_0-logloss:0.29479
[15]	validation_0-logloss:0.29615
[16]	validation_0-logloss:0.29605
[17]	validation_0-logloss:0.29656
[18]	validation_0-logloss:0.29605
[19]	validation_0-logloss:0.29830
[20]	validation_0-logloss:0.30036
[0]	validation_0-logloss:0.52957
[1]	validation_0-logloss:0.43944
[2]	validation_0-logloss:0.38977
[3]	validation_0-logloss:0.35359
[4]	validation_0-logloss:0.32719
[5]	validation_0-logloss:0.31539
[6]	validation_0-logloss:0.30277
[7]	validation_0-logloss:0.29750
[8]	validation_0-logloss:0.29406
[9]	validation_0-logloss:0.29429
[10]	validation_0-logloss:0.29286
```

```
[11]	validation_0-logloss:0.29258
[12]	validation_0-logloss:0.29295
[13]	validation_0-logloss:0.29271
[14]	validation_0-logloss:0.29479
[15]	validation_0-logloss:0.29615
[16]	validation_0-logloss:0.29605
[17]	validation_0-logloss:0.29656
[18]	validation_0-logloss:0.29605
[19]	validation_0-logloss:0.29830
[20]	validation_0-logloss:0.30036
[0]	validation_0-logloss:0.52957
[1]	validation_0-logloss:0.43944
[2]	validation_0-logloss:0.38977
[3]	validation_0-logloss:0.35359
[4]	validation_0-logloss:0.32719
[5]	validation_0-logloss:0.31539
[6]	validation_0-logloss:0.30277
[7]	validation_0-logloss:0.29750
[8]	validation_0-logloss:0.29406
[9]	validation_0-logloss:0.29429
[10]	validation_0-logloss:0.29286
[11]	validation_0-logloss:0.29258
[12]	validation_0-logloss:0.29295
[13]	validation_0-logloss:0.29271
[14]	validation_0-logloss:0.29479
[15]	validation_0-logloss:0.29615
[16]	validation_0-logloss:0.29605
[17]	validation_0-logloss:0.29656
[18]	validation_0-logloss:0.29605
[19]	validation_0-logloss:0.29830
[20]	validation_0-logloss:0.30036
```



In [46]:

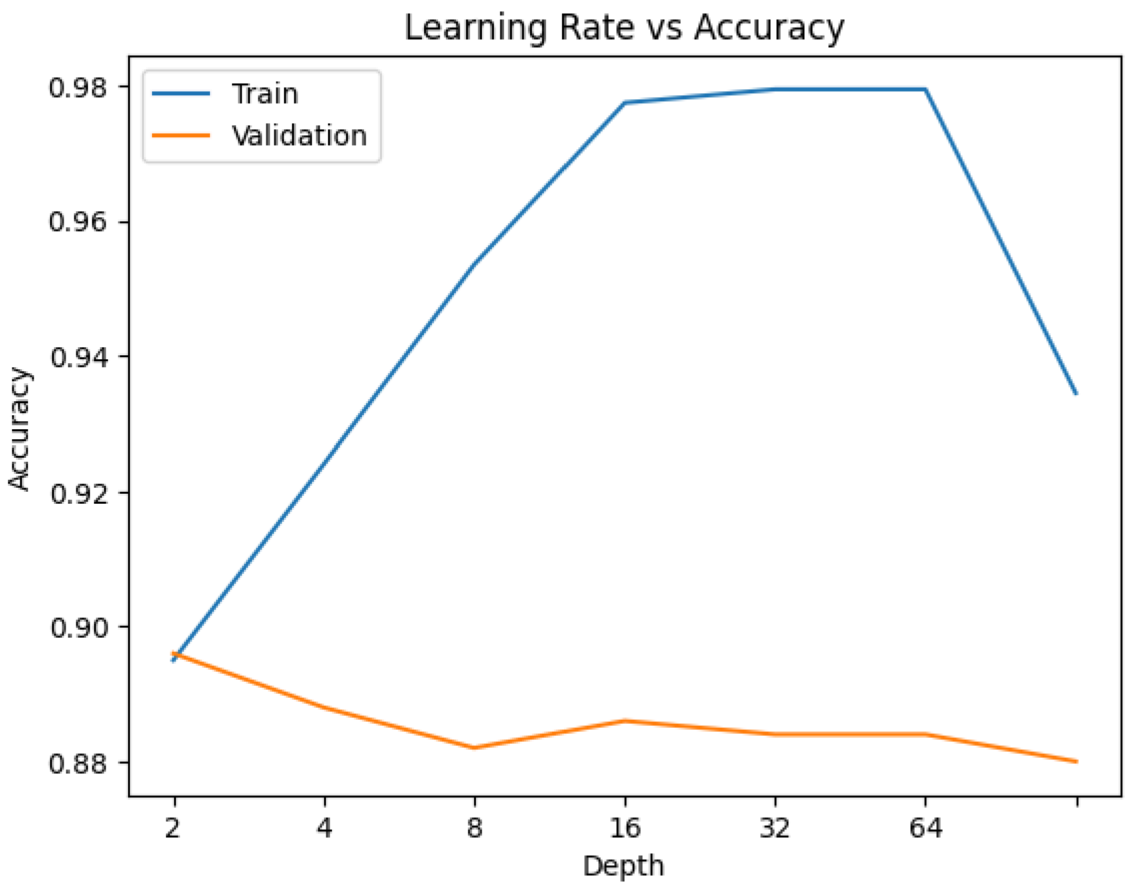
```
accuracy_list_test_xgb = []
accuracy_list_train_xgb = []
for depth in max_depth_list:
```

```
    # Number of trees can be set to a higher value for XGBoost since model wi
    xgb_model = XGBClassifier(n_estimators = 500, max_depth = depth, verbosit
    xgb_model.fit(X_train, y_train_encoded, eval_set=[(X_test, y_test_encoded
    prediction_train = xgb_model.predict(X_train)
    prediction_test = xgb_model.predict(X_test)
    accuracy_train = accuracy_score(prediction_train, y_train_encoded)
    accuracy_test = accuracy_score(prediction_test, y_test_encoded)
    accuracy_list_train_xgb.append(accuracy_train)
    accuracy_list_test_xgb.append(accuracy_test)

plt.title('Learning Rate vs Accuracy')
plt.xlabel('Depth')
plt.ylabel('Accuracy')
plt.xticks(ticks=range(len(max_depth_list)), labels = max_depth_list)
plt.plot(accuracy_list_train_xgb)
plt.plot(accuracy_list_test_xgb)
plt.legend(['Train', "Validation"])
plt.show()
```

```
[0]	validation_0-logloss:0.54215
[1]	validation_0-logloss:0.45459
[2]	validation_0-logloss:0.40414
[3]	validation_0-logloss:0.37337
[4]	validation_0-logloss:0.35276
[5]	validation_0-logloss:0.33243
[6]	validation_0-logloss:0.32235
[7]	validation_0-logloss:0.31157
[8]	validation_0-logloss:0.30627
[9]	validation_0-logloss:0.30165
[10]	validation_0-logloss:0.29612
[11]	validation_0-logloss:0.29369
[12]	validation_0-logloss:0.29106
[13]	validation_0-logloss:0.28805
[14]	validation_0-logloss:0.28599
[15]	validation_0-logloss:0.28437
[16]	validation_0-logloss:0.28431
[17]	validation_0-logloss:0.28405
[18]	validation_0-logloss:0.28320
[19]	validation_0-logloss:0.28203
[20]	validation_0-logloss:0.28238
[21]	validation_0-logloss:0.28216
[22]	validation_0-logloss:0.28221
[23]	validation_0-logloss:0.28378
[24]	validation_0-logloss:0.28327
[25]	validation_0-logloss:0.28504
[26]	validation_0-logloss:0.28550
[27]	validation_0-logloss:0.28530
[28]	validation_0-logloss:0.28395
[0]	validation_0-logloss:0.53268
[1]	validation_0-logloss:0.44039
[2]	validation_0-logloss:0.38649
[3]	validation_0-logloss:0.35107
[4]	validation_0-logloss:0.33148
[5]	validation_0-logloss:0.31687
[6]	validation_0-logloss:0.31061
[7]	validation_0-logloss:0.30357
[8]	validation_0-logloss:0.29831
[9]	validation_0-logloss:0.29613
[10]	validation_0-logloss:0.29318
[11]	validation_0-logloss:0.29221
[12]	validation_0-logloss:0.28951
[13]	validation_0-logloss:0.28946
[14]	validation_0-logloss:0.29206
[15]	validation_0-logloss:0.29127
```

```
[16]	validation_0-logloss:0.28799
[17]	validation_0-logloss:0.28873
[18]	validation_0-logloss:0.28920
[19]	validation_0-logloss:0.28884
[20]	validation_0-logloss:0.28987
[21]	validation_0-logloss:0.29086
[22]	validation_0-logloss:0.29051
[23]	validation_0-logloss:0.28783
[24]	validation_0-logloss:0.28787
[25]	validation_0-logloss:0.28888
[26]	validation_0-logloss:0.28959
[27]	validation_0-logloss:0.28966
[28]	validation_0-logloss:0.29026
[29]	validation_0-logloss:0.29129
[30]	validation_0-logloss:0.29195
[31]	validation_0-logloss:0.29206
[32]	validation_0-logloss:0.29246
[33]	validation_0-logloss:0.29040
[0]	validation_0-logloss:0.52777
[1]	validation_0-logloss:0.43750
[2]	validation_0-logloss:0.38840
[3]	validation_0-logloss:0.35246
[4]	validation_0-logloss:0.32736
[5]	validation_0-logloss:0.31123
[6]	validation_0-logloss:0.30156
[7]	validation_0-logloss:0.29543
[8]	validation_0-logloss:0.29108
[9]	validation_0-logloss:0.28816
[10]	validation_0-logloss:0.29117
[11]	validation_0-logloss:0.29301
[12]	validation_0-logloss:0.29640
[13]	validation_0-logloss:0.29625
[14]	validation_0-logloss:0.29656
[15]	validation_0-logloss:0.30112
[16]	validation_0-logloss:0.30018
[17]	validation_0-logloss:0.29921
[18]	validation_0-logloss:0.30100
[0]	validation_0-logloss:0.52703
[1]	validation_0-logloss:0.43674
[2]	validation_0-logloss:0.38237
[3]	validation_0-logloss:0.34501
[4]	validation_0-logloss:0.32304
[5]	validation_0-logloss:0.30740
[6]	validation_0-logloss:0.29668
[7]	validation_0-logloss:0.29257
[8]	validation_0-logloss:0.29151
[9]	validation_0-logloss:0.28918
[10]	validation_0-logloss:0.29037
[11]	validation_0-logloss:0.29218
[12]	validation_0-logloss:0.29063
[13]	validation_0-logloss:0.29522
[14]	validation_0-logloss:0.29485
[15]	validation_0-logloss:0.29618
[16]	validation_0-logloss:0.30019
[17]	validation_0-logloss:0.30195
[18]	validation_0-logloss:0.30447
[19]	validation_0-logloss:0.30894
[0]	validation_0-logloss:0.52703
[1]	validation_0-logloss:0.43674
[2]	validation_0-logloss:0.38237
[3]	validation_0-logloss:0.34501
[4]	validation_0-logloss:0.32304
[5]	validation_0-logloss:0.30740
[6]	validation_0-logloss:0.29668
```

```
[7]	validation_0-logloss:0.29257
[8]	validation_0-logloss:0.29168
[9]	validation_0-logloss:0.29008
[10]	validation_0-logloss:0.28863
[11]	validation_0-logloss:0.29265
[12]	validation_0-logloss:0.29404
[13]	validation_0-logloss:0.29427
[14]	validation_0-logloss:0.29844
[15]	validation_0-logloss:0.29993
[16]	validation_0-logloss:0.29865
[17]	validation_0-logloss:0.30166
[18]	validation_0-logloss:0.30280
[19]	validation_0-logloss:0.30454
[0]	validation_0-logloss:0.52703
[1]	validation_0-logloss:0.43674
[2]	validation_0-logloss:0.38237
[3]	validation_0-logloss:0.34501
[4]	validation_0-logloss:0.32304
[5]	validation_0-logloss:0.30740
[6]	validation_0-logloss:0.29668
[7]	validation_0-logloss:0.29257
[8]	validation_0-logloss:0.29168
[9]	validation_0-logloss:0.29008
[10]	validation_0-logloss:0.28863
[11]	validation_0-logloss:0.29265
[12]	validation_0-logloss:0.29404
[13]	validation_0-logloss:0.29427
[14]	validation_0-logloss:0.29844
[15]	validation_0-logloss:0.29993
[16]	validation_0-logloss:0.29865
[17]	validation_0-logloss:0.30166
[18]	validation_0-logloss:0.30280
[19]	validation_0-logloss:0.30454
[20]	validation_0-logloss:0.30455
[0]	validation_0-logloss:0.52957
[1]	validation_0-logloss:0.43944
[2]	validation_0-logloss:0.38977
[3]	validation_0-logloss:0.35359
[4]	validation_0-logloss:0.32719
[5]	validation_0-logloss:0.31539
[6]	validation_0-logloss:0.30277
[7]	validation_0-logloss:0.29750
[8]	validation_0-logloss:0.29406
[9]	validation_0-logloss:0.29429
[10]	validation_0-logloss:0.29286
[11]	validation_0-logloss:0.29258
[12]	validation_0-logloss:0.29295
[13]	validation_0-logloss:0.29271
[14]	validation_0-logloss:0.29479
[15]	validation_0-logloss:0.29615
[16]	validation_0-logloss:0.29605
[17]	validation_0-logloss:0.29656
[18]	validation_0-logloss:0.29605
[19]	validation_0-logloss:0.29830
[20]	validation_0-logloss:0.30036
```



A learning rate of 0.1 and maximum depth of 2 will be chosen.

Final XGBoost Model

In [48]:
```
xgb_model = XGBClassifier(n_estimators = 500, learning_rate = 0.1, max_depth
xgb_model.fit(X_train, y_train_encoded, eval_set=[(X_test, y_test_encoded)])
```

```
[0]	validation_0-logloss:0.63768
[1]	validation_0-logloss:0.58984
[2]	validation_0-logloss:0.55032
[3]	validation_0-logloss:0.51793
[4]	validation_0-logloss:0.49032
[5]	validation_0-logloss:0.46623
[6]	validation_0-logloss:0.44569
[7]	validation_0-logloss:0.42838
[8]	validation_0-logloss:0.41388
[9]	validation_0-logloss:0.40094
[10]	validation_0-logloss:0.39106
[11]	validation_0-logloss:0.38106
[12]	validation_0-logloss:0.37201
[13]	validation_0-logloss:0.36577
[14]	validation_0-logloss:0.35889
[15]	validation_0-logloss:0.35374
[16]	validation_0-logloss:0.34795
[17]	validation_0-logloss:0.34181
[18]	validation_0-logloss:0.33680
[19]	validation_0-logloss:0.33331
[20]	validation_0-logloss:0.32946
[21]	validation_0-logloss:0.32474
[22]	validation_0-logloss:0.32158
[23]	validation_0-logloss:0.31762
[24]	validation_0-logloss:0.31474
```

```
[25]	validation_0-logloss:0.31208
[26]	validation_0-logloss:0.30892
[27]	validation_0-logloss:0.30728
[28]	validation_0-logloss:0.30577
[29]	validation_0-logloss:0.30347
[30]	validation_0-logloss:0.30225
[31]	validation_0-logloss:0.30010
[32]	validation_0-logloss:0.29872
[33]	validation_0-logloss:0.29715
[34]	validation_0-logloss:0.29587
[35]	validation_0-logloss:0.29480
[36]	validation_0-logloss:0.29413
[37]	validation_0-logloss:0.29373
[38]	validation_0-logloss:0.29266
[39]	validation_0-logloss:0.29171
[40]	validation_0-logloss:0.29106
[41]	validation_0-logloss:0.29047
[42]	validation_0-logloss:0.29003
[43]	validation_0-logloss:0.28936
[44]	validation_0-logloss:0.28903
[45]	validation_0-logloss:0.28828
[46]	validation_0-logloss:0.28838
[47]	validation_0-logloss:0.28799
[48]	validation_0-logloss:0.28719
[49]	validation_0-logloss:0.28699
[50]	validation_0-logloss:0.28654
[51]	validation_0-logloss:0.28635
[52]	validation_0-logloss:0.28575
[53]	validation_0-logloss:0.28559
[54]	validation_0-logloss:0.28547
[55]	validation_0-logloss:0.28495
[56]	validation_0-logloss:0.28515
[57]	validation_0-logloss:0.28514
[58]	validation_0-logloss:0.28502
[59]	validation_0-logloss:0.28507
[60]	validation_0-logloss:0.28473
[61]	validation_0-logloss:0.28470
[62]	validation_0-logloss:0.28547
[63]	validation_0-logloss:0.28556
[64]	validation_0-logloss:0.28537
[65]	validation_0-logloss:0.28519
[66]	validation_0-logloss:0.28461
[67]	validation_0-logloss:0.28474
[68]	validation_0-logloss:0.28461
[69]	validation_0-logloss:0.28509
[70]	validation_0-logloss:0.28488
[71]	validation_0-logloss:0.28504
[72]	validation_0-logloss:0.28507
[73]	validation_0-logloss:0.28469
[74]	validation_0-logloss:0.28458
[75]	validation_0-logloss:0.28510
[76]	validation_0-logloss:0.28464
[77]	validation_0-logloss:0.28419
[78]	validation_0-logloss:0.28462
[79]	validation_0-logloss:0.28435
[80]	validation_0-logloss:0.28529
[81]	validation_0-logloss:0.28499
[82]	validation_0-logloss:0.28516
[83]	validation_0-logloss:0.28499
[84]	validation_0-logloss:0.28521
[85]	validation_0-logloss:0.28496
[86]	validation_0-logloss:0.28458
[87]	validation_0-logloss:0.28536
```

Out[48]:

```
▾ XGBClassifier
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_roun
ds=10,
              enable_categorical=False, eval_metric=None, feature_typ
es=None,
              gamma=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=0.1, max_bi
n=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=2, max_leaves=None,
```

Best Iteration

In [49]:

```
xgb_model.best_iteration
```

Out[49]:

```
77
```

Although set the number of trees to be 500, but actually the best number is 77.

Evaluation

In [50]:

```
y_pred = xgb_model.predict(X_test)
```

Classification Report

In [51]:

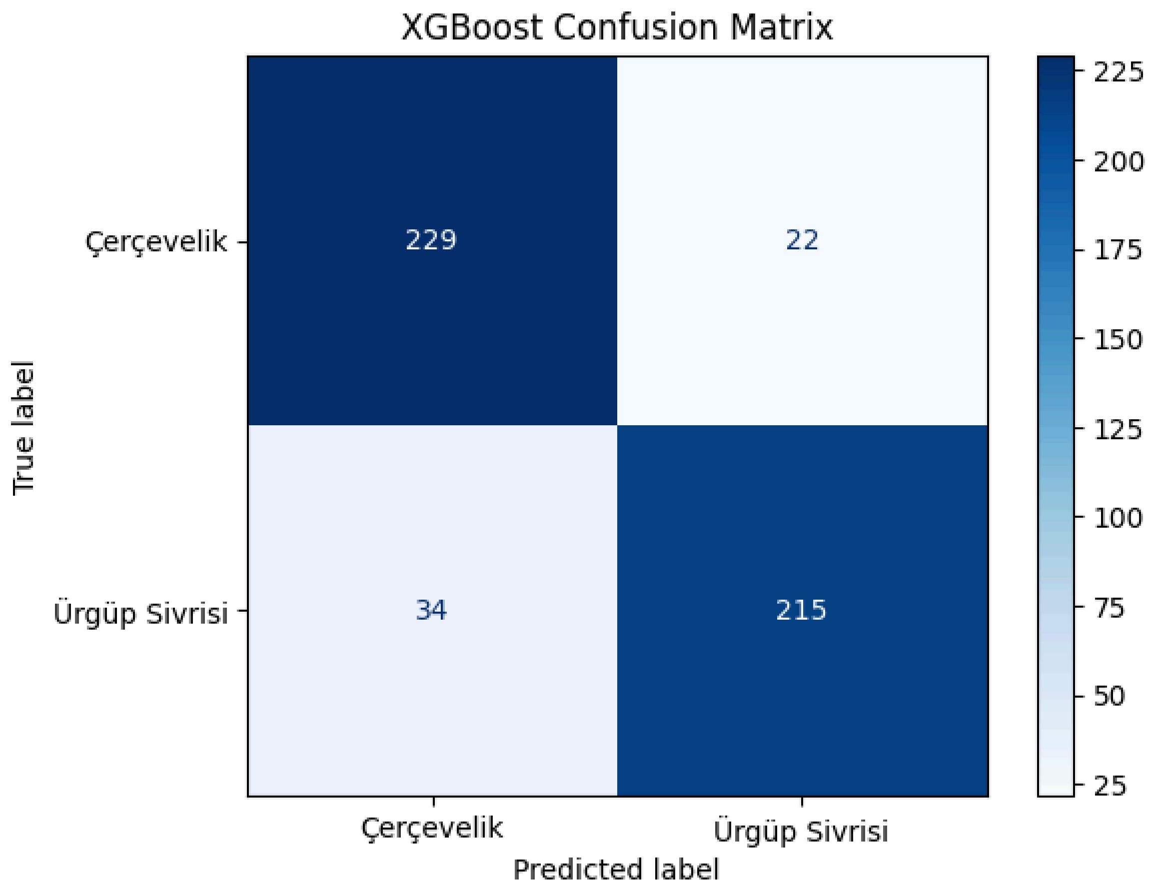
```
print(classification_report(y_test_encoded, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.87      0.91      0.89       251
           1       0.91      0.86      0.88       249

    accuracy                           0.89       500
   macro avg       0.89      0.89      0.89       500
weighted avg       0.89      0.89      0.89       500
```

Confusion Matrix

In [52]:

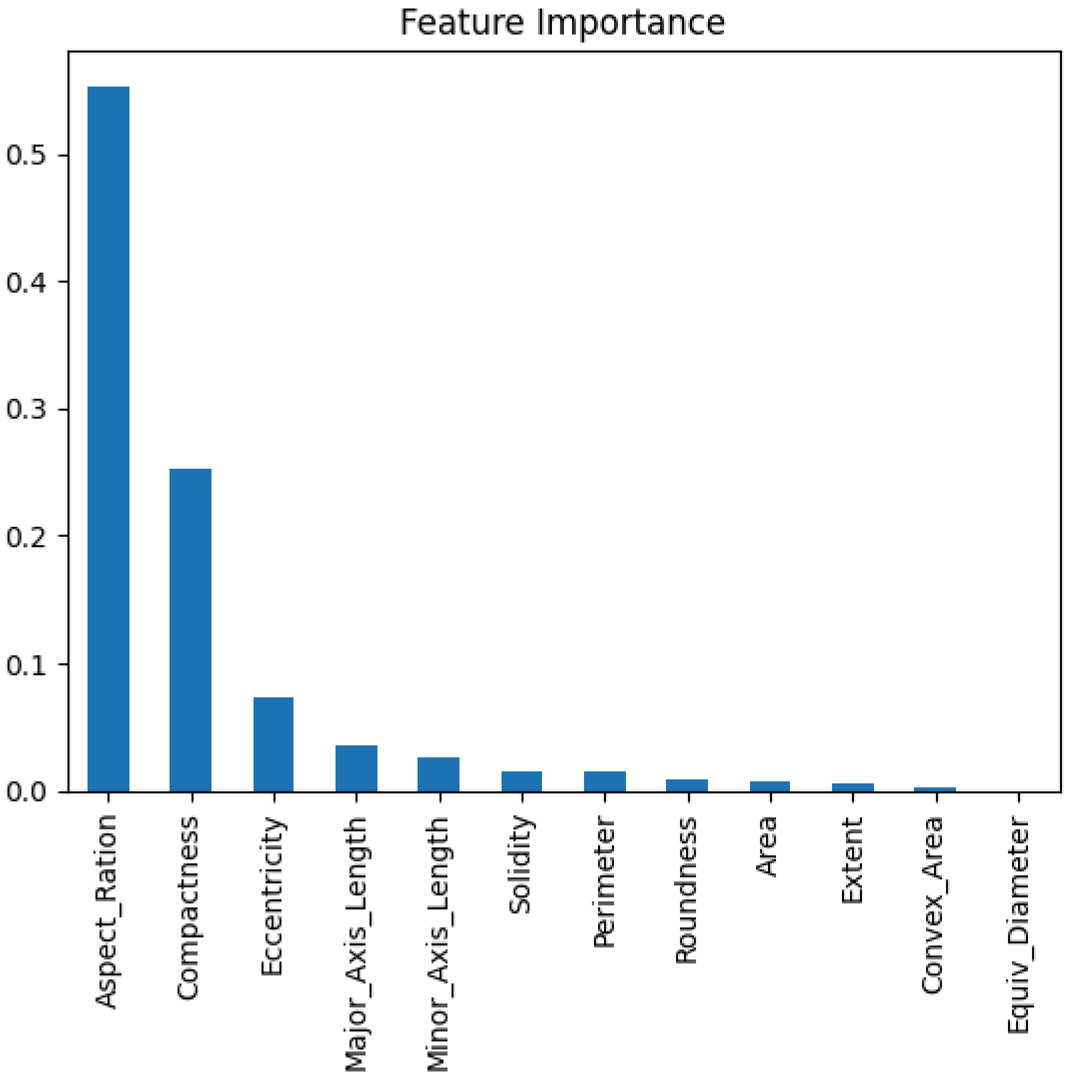
```
cm = confusion_matrix(y_test_encoded, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix = cm, display_labels = class_l
disp.plot(cmap='Blues')
plt.title("XGBoost Confusion Matrix")
plt.show()
```



Importance

In [53]:

```
importances = xgb_model.feature_importances_
feat_importances = pd.Series(importances, index = features)
feat_importances.sort_values(ascending = False).plot(kind = 'bar')
plt.title("Feature Importance")
plt.show()
```



5. Conclusion

- Random Forest performs slightly better on this dataset in terms of overall accuracy (0.90) and F1-score (Çerçevelik: 0.90, Ürgüp Sivrisi: 0.89).
- XGBoost is also showing good F1-score (Çerçevelik: 0.89, Ürgüp Sivrisi: 0.88) and accuracy (0.89).
- Both models are strong classifiers.
- Top three important features for Random Forest are Eccentricity, Compactness, and Aspect Ration.
- Top three important features for XGBoost are Aspect Ration, Compactness, and Eccentricity.
- Top features are the same except their ranking.