In [ ]:

```
# Update sklearn to prevent version mismatches
!pip install sklearn --upgrade
```

In [ ]:

```
# install joblib. This will be used to save your model.
# Restart your kernel after installing
!pip install joblib
```

In [1]:

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

In [2]:

```
import warnings
warnings.simplefilter('ignore')
```

Read the CSV and Perform Basic Data Cleaning

In [3]:

```
df = pd.read_csv("exoplanet_data.csv")
# Drop the null columns where all values are null
df = df.dropna(axis='columns', how='all')
# Drop the null rows
df = df.dropna()
df.head()
```

Out[3]:

	koi_disposition	koi_fpflag_nt	koi_fpflag_ss	koi_fpflag_co	koi_fpflag_ec	koi_period	koi_period_err1	koi_period_err2	koi_time0bk	koi_time0bk
0	CONFIRMED	0	0	0	0	54.418383	2.479000e-04	-2.479000e-04	162.513840	0.0
1	FALSE POSITIVE	0	1	0	0	19.899140	1.490000e-05	-1.490000e-05	175.850252	0.0
2	FALSE POSITIVE	0	1	0	0	1.736952	2.630000e-07	-2.630000e-07	170.307565	0.0
3	CONFIRMED	0	0	0	0	2.525592	3.760000e-06	-3.760000e-06	171.595550	0.0
4	CONFIRMED	0	0	0	0	4.134435	1.050000e-05	-1.050000e-05	172.979370	0.0

5 rows × 41 columns

In [4]:

```
# https://exoplanetarchive.ipac.caltech.edu/docs/API_kepcandidate_columns.html#stellar_param
df.columns
```

Out[4]:

```
Index(['koi_disposition', 'koi_fpflag_nt', 'koi_fpflag_ss', 'koi_fpflag_co',
       'koi_fpflag_ec', 'koi_period', 'koi_period_err1', 'koi_period_err2',
       'koi_time0bk', 'koi_time0bk_err1', 'koi_time0bk_err2', 'koi_impact',
       'koi_impact_err1', 'koi_impact_err2', 'koi_duration',
       'koi_duration_err1', 'koi_duration_err2', 'koi_depth', 'koi_depth_err1',
       'koi_depth_err2', 'koi_prad', 'koi_prad_err1', 'koi_prad_err2',
       'koi_teq', 'koi_insol', 'koi_insol_err1', 'koi_insol_err2',
       'koi_model_snr', 'koi_tce_plnt_num', 'koi_steff', 'koi_steff_err1',
       'koi_steff_err2', 'koi_slogg', 'koi_slogg_err1', 'koi_slogg_err2',
       'koi_srad', 'koi_srad_err1', 'koi_srad_err2', 'ra', 'dec',
       'koi_kepmag'],
      dtype='object')
```

In [5]:
```python
# Based on prior analysis with feature...
Xtemp = df[['koi_fpflag_nt', 'koi_fpflag_ss', 'koi_fpflag_co', 'koi_fpflag_ec', 'koi_period','koi_time0bk','
Xtemp
```

Out[5]:

	koi_fpflag_nt	koi_fpflag_ss	koi_fpflag_co	koi_fpflag_ec	koi_period	koi_time0bk	koi_slogg	koi_srad	koi_impact	koi_duration	...	koi_pra
0	0	0	0	0	54.418383	162.513840	4.467	0.927	0.586	4.50700	...	2.
1	0	1	0	0	19.899140	175.850252	4.544	0.868	0.969	1.78220	...	14.
2	0	1	0	0	1.736952	170.307565	4.564	0.791	1.276	2.40641	...	33.
3	0	0	0	0	2.525592	171.595550	4.438	1.046	0.701	1.65450	...	2.
4	0	0	0	0	4.134435	172.979370	4.486	0.972	0.762	3.14020	...	2.
...	...	...	...	...	...	...	...	...	...	...	...	
6986	0	0	0	1	8.589871	132.016100	4.296	1.088	0.765	4.80600	...	1.
6987	0	1	1	0	0.527699	131.705093	4.529	0.903	1.252	3.22210	...	29.
6988	0	0	0	0	1.739849	133.001270	4.444	1.031	0.043	3.11400	...	0.
6989	0	0	1	0	0.681402	132.181750	4.447	1.041	0.147	0.86500	...	1.
6990	0	0	1	1	4.856035	135.993300	4.385	1.193	0.134	3.07800	...	1.

6991 rows × 21 columns

In [6]:
```python
y = df[['koi_disposition']]
data_binary_encoded = pd.get_dummies(y, columns=["koi_disposition"])
data_binary_encoded.columns = [["candidate","confirmed","false_positive"]]
```

In [7]:
```python
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier(n_estimators=200)
rf = rf.fit(Xtemp, data_binary_encoded)
```

In [8]:
```python
# Random Forests in sklearn will automatically calculate feature importance
importances = rf.feature_importances_
importances
```

Out[8]:
```
array([0.13008624, 0.10483403, 0.12352244, 0.04581723, 0.04106112,
       0.02581772, 0.01685325, 0.01722201, 0.04005631, 0.02922445,
       0.04333847, 0.08014678, 0.03041426, 0.03204579, 0.12807676,
       0.02012986, 0.01593547, 0.01669276, 0.02018772, 0.01911327,
       0.01942407])
```

In [9]:
```python
# We can sort the features by their importance
sorted(zip(rf.feature_importances_, Xtemp), reverse=True)
```

Out[9]:
```
[(0.13008624213084874, 'koi_fpflag_nt'),
 (0.1280767574640183, 'koi_model_snr'),
 (0.1235224368390042, 'koi_fpflag_co'),
 (0.10483403059720878, 'koi_fpflag_ss'),
 (0.08014677860960959, 'koi_prad'),
 (0.045817230982805254, 'koi_fpflag_ec'),
 (0.04333847209702059, 'koi_depth'),
 (0.04106111510753319, 'koi_period'),
 (0.040056309240269664, 'koi_impact'),
 (0.032045792989303455, 'koi_insol'),
 (0.03041426236240638, 'koi_teq'),
 (0.02922444511349618, 'koi_duration'),
 (0.025817720359986765, 'koi_time0bk'),
 (0.020187718365325492, 'ra'),
 (0.020129855461575637, 'koi_steff'),
 (0.019424067265294873, 'koi_kepmag'),
 (0.01911327046420486, 'dec'),
 (0.017222008201840857, 'koi_srad'),
 (0.01685325358020019, 'koi_slogg'),
 (0.01669276056609946, 'koi_srad'),
 (0.015935472201947708, 'koi_slogg')]
```

In [10]:
```
# removing features less than 0.2107
X = Xtemp.drop(columns=['ra','dec','koi_kepmag','koi_srad','koi_slogg'])
X
```

Out[10]:

	koi_fpflag_nt	koi_fpflag_ss	koi_fpflag_co	koi_fpflag_ec	koi_period	koi_time0bk	koi_impact	koi_duration	koi_depth	koi_prad	koi_teq
0	0	0	0	0	54.418383	162.513840	0.586	4.50700	874.8	2.83	443
1	0	1	0	0	19.899140	175.850252	0.969	1.78220	10829.0	14.60	638
2	0	1	0	0	1.736952	170.307565	1.276	2.40641	8079.2	33.46	1395
3	0	0	0	0	2.525592	171.595550	0.701	1.65450	603.3	2.75	1406
4	0	0	0	0	4.134435	172.979370	0.762	3.14020	686.0	2.77	1160
...	...	...	...	...	...	...	...	...	...	...	...
6986	0	0	0	1	8.589871	132.016100	0.765	4.80600	87.7	1.11	929
6987	0	1	1	0	0.527699	131.705093	1.252	3.22210	1579.2	29.35	2088
6988	0	0	0	0	1.739849	133.001270	0.043	3.11400	48.5	0.72	1608
6989	0	0	1	0	0.681402	132.181750	0.147	0.86500	103.6	1.07	2218
6990	0	0	1	1	4.856035	135.993300	0.134	3.07800	76.7	1.05	1266

6991 rows × 14 columns

In [11]:
```
target_names = y['koi_disposition'].unique().tolist()
target_names
```

Out[11]:
```
['CONFIRMED', 'FALSE POSITIVE', 'CANDIDATE']
```

Select your features (columns)

In [12]:
```
# Set features. This will also be used as your x values.
selected_features = X.columns
selected_features
```

Out[12]:
```
Index(['koi_fpflag_nt', 'koi_fpflag_ss', 'koi_fpflag_co', 'koi_fpflag_ec',
       'koi_period', 'koi_time0bk', 'koi_impact', 'koi_duration', 'koi_depth',
       'koi_prad', 'koi_teq', 'koi_insol', 'koi_model_snr', 'koi_steff'],
      dtype='object')
```

Create a Train Test Split

In [13]:
```
from sklearn.model_selection import train_test_split
# random stat 42
X42_train, X42_test, y42_train, y42_test = train_test_split(X, y, random_state=42)
```

In [14]:
```
X42_train.head()
```

Out[14]:

	koi_fpflag_nt	koi_fpflag_ss	koi_fpflag_co	koi_fpflag_ec	koi_period	koi_time0bk	koi_impact	koi_duration	koi_depth	koi_prad	koi_teq
6122	0	0	0	0	6.768901	133.077240	0.150	3.61600	123.1	1.24	1017
6370	0	1	0	1	0.733726	132.020050	0.291	2.30900	114.6	0.86	1867
2879	1	0	0	0	7.652707	134.460380	0.970	79.89690	641.1	3.21	989
107	0	0	0	0	7.953547	174.662240	0.300	2.63120	875.4	2.25	696
29	0	0	0	0	4.959319	172.258529	0.831	2.22739	9802.0	12.21	1103

In [15]:
```
# random stat 1
X1_train, X1_test, y1_train, y1_test = train_test_split(X, y, random_state=1)
```

In [16]:
```
X1_train.head()
```

Out[16]:

	koi_fpflag_nt	koi_fpflag_ss	koi_fpflag_co	koi_fpflag_ec	koi_period	koi_time0bk	koi_impact	koi_duration	koi_depth	koi_prad	koi_teq
3563	0	0	0	0	10.548413	139.064020	1.0170	1.8720	102.9	3.89	899
4099	0	0	0	0	24.754385	140.207320	0.7090	3.3900	593.3	2.10	491
5460	0	0	0	0	1.057336	131.792007	0.2620	1.5795	47337.0	14.59	1276
1091	0	0	0	0	201.118319	187.569860	0.0010	10.3280	584.8	2.28	300
5999	0	0	0	0	91.649983	175.715600	0.2136	10.2940	193.6	2.27	568

In [17]:
```
# random stat 21
X21_train, X21_test, y21_train, y21_test = train_test_split(X, y, random_state=21)
```

In [18]:
```
X21_train.head()
```

Out[18]:

	koi_fpflag_nt	koi_fpflag_ss	koi_fpflag_co	koi_fpflag_ec	koi_period	koi_time0bk	koi_impact	koi_duration	koi_depth	koi_prad	koi_teq
6966	1	0	1	0	361.901618	405.302100	0.093	4.9840	184.1	0.95	198
1714	0	0	0	0	6.739683	132.292960	0.662	4.1830	142.0	1.48	1011
225	0	0	0	0	3.166354	170.966145	0.032	3.3129	1473.4	3.82	1273
5266	0	0	0	0	25.090157	138.498800	0.935	8.2730	37.7	1.35	852
5468	0	1	0	0	7.234966	134.582307	0.548	7.0245	164850.0	34.03	843

Pre-processing

Scale the data using the MinMaxScaler and perform some feature selection

In [19]:
```
from sklearn.preprocessing import MinMaxScaler, StandardScaler
```

In [20]:
```
# StandardScaler
X42S_scaler = StandardScaler().fit(X42_train)
X42S_train_scaled = X42S_scaler.transform(X42_train)
X42S_test_scaled = X42S_scaler.transform(X42_test)
```

In [21]:
```
# MinMaxScaler
X42M_scaler = MinMaxScaler().fit(X42_train)
X42M_train_scaled = X42M_scaler.transform(X42_train)
X42M_test_scaled = X42M_scaler.transform(X42_test)
```

Train the Model

In [22]:
```
# Create the SVC Model
# 'linear', 'poly', 'rbf', 'sigmoid', 'precomputed' or a callable (default = 'rbf')
from sklearn.svm import SVC
model = SVC(kernel='linear')
model
```

Out[22]:
```
SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='linear',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

In [23]:
```
model2 = SVC(kernel='rbf')
model2
```

Out[23]:
```
SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='rbf',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

In [24]:
```
model3 = SVC(kernel='sigmoid')
model3
```

Out[24]:
```
SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='sigmoid',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

Testing random state 42

In [25]:
```
# StandardScaler score - Linear Kernel
model.fit(X42S_train_scaled, y42_train)
print(f"Training Data Score: {model.score(X42S_train_scaled, y42_train)}")
print(f"Testing Data Score: {model.score(X42S_test_scaled, y42_test)}")
```

```
Training Data Score: 0.8083158497043678
Testing Data Score: 0.8049199084668193
```

In [27]:
```
# MinMaxScaler Score - Linear Kernel
model.fit(X42M_train_scaled, y42_train)
print(f"Training Data Score: {model.score(X42M_train_scaled, y42_train)}")
print(f"Testing Data Score: {model.score(X42M_test_scaled, y42_test)}")
```

```
Training Data Score: 0.7818043105092505
Testing Data Score: 0.7929061784897025
```

In [28]:
```
# StandardScaler score - rbf Kernel
model2.fit(X42S_train_scaled, y42_train)
print(f"Training Data Score: {model2.score(X42S_train_scaled, y42_train)}")
print(f"Testing Data Score: {model2.score(X42S_test_scaled, y42_test)}")
```

```
Training Data Score: 0.8189967575815373
Testing Data Score: 0.816933638443936
```

In [29]:
```
# MinMaxScaler Score - rbf Kernel
model2.fit(X42M_train_scaled, y42_train)
print(f"Training Data Score: {model2.score(X42M_train_scaled, y42_train)}")
print(f"Testing Data Score: {model2.score(X42M_test_scaled, y42_test)}")
```

```
Training Data Score: 0.7922944878886133
Testing Data Score: 0.8003432494279176
```

In [30]:
```
# StandardScaler score - sigmoid Kernel
model3.fit(X42S_train_scaled, y42_train)
print(f"Training Data Score: {model3.score(X42S_train_scaled, y42_train)}")
print(f"Testing Data Score: {model3.score(X42S_test_scaled, y42_test)}")
```

```
Training Data Score: 0.7148579057791341
Testing Data Score: 0.7191075514874142
```

In [31]:
```
# MinMaxScaler Score - sigmoid Kernel
model3.fit(X42M_train_scaled, y42_train)
print(f"Training Data Score: {model3.score(X42M_train_scaled, y42_train)}")
print(f"Testing Data Score: {model3.score(X42M_test_scaled, y42_test)}")
```

```
Training Data Score: 0.7806599275224109
Testing Data Score: 0.7934782608695652
```

StandardScaler scores better than MinMaxScaler with random_state 42

```
StandardScaler better scores (81.69%) with rbf kernel than others
```

Testing random state 1

In [33]:
```
X1S_scaler = StandardScaler().fit(X1_train)
X1S_train_scaled = X1S_scaler.transform(X1_train)
X1S_test_scaled = X1S_scaler.transform(X1_test)
```

In [34]:
```
X1M_scaler = MinMaxScaler().fit(X1_train)
X1M_train_scaled = X42M_scaler.transform(X1_train)
X1M_test_scaled = X42M_scaler.transform(X1_test)
```

In [35]:
```
# StandardScaler score with random state 1
model.fit(X1S_train_scaled, y1_train)
print(f"Training Data Score: {model.score(X1S_train_scaled, y1_train)}")
print(f"Testing Data Score: {model.score(X1S_test_scaled, y1_test)}")
```

```
Training Data Score: 0.795346175853519
Testing Data Score: 0.8094965675057209
```

In [36]:
```
# MinMaxScaler Score with random state 1
model.fit(X1M_train_scaled, y1_train)
print(f"Training Data Score: {model.score(X1M_train_scaled, y1_train)}")
print(f"Testing Data Score: {model.score(X1M_test_scaled, y1_test)}")
```

```
Training Data Score: 0.7798970055311845
Testing Data Score: 0.7889016018306636
```

In [37]:
```
# StandardScaler score with random state 1
model2.fit(X1S_train_scaled, y1_train)
print(f"Training Data Score: {model2.score(X1S_train_scaled, y1_train)}")
print(f"Testing Data Score: {model2.score(X1S_test_scaled, y1_test)}")
```

```
Training Data Score: 0.8170894526034713
Testing Data Score: 0.8255148741418764
```

In [38]:
```
# MinMaxScaler Score with random state 1
model2.fit(X1M_train_scaled, y1_train)
print(f"Training Data Score: {model2.score(X1M_train_scaled, y1_train)}")
print(f"Testing Data Score: {model2.score(X1M_test_scaled, y1_test)}")
```

```
Training Data Score: 0.7856189204653824
Testing Data Score: 0.7951945080091534
```

In [39]:
```
# StandardScaler score with random state 1
model3.fit(X1S_train_scaled, y1_train)
print(f"Training Data Score: {model3.score(X1S_train_scaled, y1_train)}")
print(f"Testing Data Score: {model3.score(X1S_test_scaled, y1_test)}")
```

```
Training Data Score: 0.7030326149151249
Testing Data Score: 0.7288329519450801
```

With random state 1 the Standard Scaler scored highest 82.55%

Testing random state 21 (rest of the tests only applying StandardScaler)

In [42]:
```
X21S_scaler = StandardScaler().fit(X21_train)
X21S_train_scaled = X21S_scaler.transform(X21_train)
X21S_test_scaled = X21S_scaler.transform(X21_test)
```

In [43]:
```
# StandardScaler score with random state 21
model.fit(X21S_train_scaled, y21_train)
print(f"Training Data Score: {model.score(X21S_train_scaled, y21_train)}")
print(f"Testing Data Score: {model.score(X21S_test_scaled, y21_test)}")
```

```
Training Data Score: 0.8033568567613961
Testing Data Score: 0.7911899313501144
```

In [44]:
```
# StandardScaler score with random state 21
model2.fit(X21S_train_scaled, y21_train)
print(f"Training Data Score: {model2.score(X21S_train_scaled, y21_train)}")
print(f"Testing Data Score: {model2.score(X21S_test_scaled, y21_test)}")
```

```
Training Data Score: 0.8256723250047683
Testing Data Score: 0.8077803203661327
```

Overall StandardScaler with kernel rbf scores better (82.55%) with random_stat 1

Hyperparameter Tuning

In [45]:
```
# Create the GridSearch estimator along with a parameter object containing the values to adjust
from sklearn.model_selection import GridSearchCV
param_grid = {'C': [1, 5, 10, 50],
              'gamma': [0.0001, 0.0005, 0.001, 0.005]}
grid = GridSearchCV(model2, param_grid, verbose=3)
```

In [46]:
```
# Fit the model using the grid search estimator.
# This will take the SVC model and try each combination of parameters
grid.fit(X21S_train_scaled, y21_train)
```

```
Fitting 5 folds for each of 16 candidates, totalling 80 fits
[CV] C=1, gamma=0.0001 ...............................................

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.

[CV] ................... C=1, gamma=0.0001, score=0.500, total=   0.6s
[CV] C=1, gamma=0.0001 ...............................................

[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.6s remaining:    0.0s

[CV] ................... C=1, gamma=0.0001, score=0.500, total=   0.6s
[CV] C=1, gamma=0.0001 ...............................................

[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    1.2s remaining:    0.0s
```

In [47]:
```
# from scipy.stats import expon
# param_grid = {'C': [1, 10, 100, 1000],
#                'gamma': [0.001, 0.0001]}
param_grid = {'C': [1, 10, 100, 1000],
              'gamma': [1e-3, 1e-4]}
grid2 = GridSearchCV(model2, param_grid, verbose=3)
```

In [48]:
```python
# Fit the model using the grid search estimator.
# This will take the SVC model and try each combination of parameters
grid2.fit(X21S_train_scaled, y21_train)
```

```
Fitting 5 folds for each of 8 candidates, totalling 40 fits
[CV] C=1, gamma=0.001 ................................................

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.

[CV] .................... C=1, gamma=0.001, score=0.777, total=   0.4s
[CV] C=1, gamma=0.001 ................................................

[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.4s remaining:    0.0s

[CV] .................... C=1, gamma=0.001, score=0.770, total=   0.4s
[CV] C=1, gamma=0.001 ................................................

[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    0.8s remaining:    0.0s

[CV] .................... C=1, gamma=0.001, score=0.755, total=   0.4s
[CV] C=1, gamma=0.001 ................................................
[CV] .................... C=1, gamma=0.001, score=0.759, total=   0.4s
[CV] C=1, gamma=0.001 ................................................
[CV] .................... C=1, gamma=0.001, score=0.770, total=   0.4s
[CV] C=1, gamma=0.0001 ...............................................
[CV] ................... C=1, gamma=0.0001, score=0.500, total=   0.6s
[CV] C=1, gamma=0.0001 ...............................................
[CV] ................... C=1, gamma=0.0001, score=0.500, total=   0.6s
[CV] C=1, gamma=0.0001 ...............................................
[CV] ................... C=1, gamma=0.0001, score=0.500, total=   0.6s
[CV] C=1, gamma=0.0001 ...............................................
[CV] ................... C=1, gamma=0.0001, score=0.500, total=   0.6s
[CV] C=1, gamma=0.0001 ...............................................
[CV] ................... C=1, gamma=0.0001, score=0.500, total=   0.6s
[CV] C=10, gamma=0.001 ...............................................
[CV] ................... C=10, gamma=0.001, score=0.789, total=   0.2s
[CV] C=10, gamma=0.001 ...............................................
[CV] ................... C=10, gamma=0.001, score=0.784, total=   0.3s
[CV] C=10, gamma=0.001 ...............................................
[CV] ................... C=10, gamma=0.001, score=0.782, total=   0.3s
[CV] C=10, gamma=0.001 ...............................................
[CV] ................... C=10, gamma=0.001, score=0.792, total=   0.3s
[CV] C=10, gamma=0.001 ...............................................
[CV] ................... C=10, gamma=0.001, score=0.781, total=   0.3s
[CV] C=10, gamma=0.0001 ..............................................
[CV] .................. C=10, gamma=0.0001, score=0.777, total=   0.4s
[CV] C=10, gamma=0.0001 ..............................................
[CV] .................. C=10, gamma=0.0001, score=0.772, total=   0.4s
[CV] C=10, gamma=0.0001 ..............................................
[CV] .................. C=10, gamma=0.0001, score=0.753, total=   0.4s
[CV] C=10, gamma=0.0001 ..............................................
[CV] .................. C=10, gamma=0.0001, score=0.760, total=   0.4s
[CV] C=10, gamma=0.0001 ..............................................
[CV] .................. C=10, gamma=0.0001, score=0.770, total=   0.4s
[CV] C=100, gamma=0.001 ..............................................
[CV] .................. C=100, gamma=0.001, score=0.798, total=   0.2s
[CV] C=100, gamma=0.001 ..............................................
[CV] .................. C=100, gamma=0.001, score=0.808, total=   0.2s
[CV] C=100, gamma=0.001 ..............................................
[CV] .................. C=100, gamma=0.001, score=0.788, total=   0.3s
[CV] C=100, gamma=0.001 ..............................................
[CV] .................. C=100, gamma=0.001, score=0.804, total=   0.2s
[CV] C=100, gamma=0.001 ..............................................
[CV] .................. C=100, gamma=0.001, score=0.809, total=   0.3s
[CV] C=100, gamma=0.0001 .............................................
[CV] ................. C=100, gamma=0.0001, score=0.787, total=   0.3s
[CV] C=100, gamma=0.0001 .............................................
[CV] ................. C=100, gamma=0.0001, score=0.783, total=   0.3s
[CV] C=100, gamma=0.0001 .............................................
[CV] ................. C=100, gamma=0.0001, score=0.781, total=   0.3s
[CV] C=100, gamma=0.0001 .............................................
[CV] ................. C=100, gamma=0.0001, score=0.791, total=   0.3s
[CV] C=100, gamma=0.0001 .............................................
[CV] ................. C=100, gamma=0.0001, score=0.782, total=   0.3s
[CV] C=1000, gamma=0.001 .............................................
[CV] ................. C=1000, gamma=0.001, score=0.818, total=   0.3s
[CV] C=1000, gamma=0.001 .............................................
[CV] ................. C=1000, gamma=0.001, score=0.821, total=   0.3s
[CV] C=1000, gamma=0.001 .............................................
```

```
[CV] ................. C=1000, gamma=0.001, score=0.810, total=   0.3s
[CV] C=1000, gamma=0.001 .............................................
[CV] ................. C=1000, gamma=0.001, score=0.828, total=   0.3s
[CV] C=1000, gamma=0.001 .............................................
[CV] ................. C=1000, gamma=0.001, score=0.815, total=   0.3s
[CV] C=1000, gamma=0.0001 ............................................
[CV] ................ C=1000, gamma=0.0001, score=0.793, total=   0.4s
[CV] C=1000, gamma=0.0001 ............................................
[CV] ................ C=1000, gamma=0.0001, score=0.806, total=   0.4s
[CV] C=1000, gamma=0.0001 ............................................
[CV] ................ C=1000, gamma=0.0001, score=0.788, total=   0.4s
[CV] C=1000, gamma=0.0001 ............................................
[CV] ................ C=1000, gamma=0.0001, score=0.802, total=   0.4s
[CV] C=1000, gamma=0.0001 ............................................
[CV] ................ C=1000, gamma=0.0001, score=0.805, total=   0.4s

[Parallel(n_jobs=1)]: Done  40 out of  40 | elapsed:   14.4s finished
```

Out[48]:
```
GridSearchCV(cv=None, error_score=nan,
             estimator=SVC(C=1.0, break_ties=False, cache_size=200,
                           class_weight=None, coef0=0.0,
                           decision_function_shape='ovr', degree=3,
                           gamma='scale', kernel='rbf', max_iter=-1,
                           probability=False, random_state=None, shrinking=True,
                           tol=0.001, verbose=False),
             iid='deprecated', n_jobs=None,
             param_grid={'C': [1, 10, 100, 1000], 'gamma': [0.001, 0.0001]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=3)
```

In [49]:
```
param_grid = {'C': [1, 50, 500, 5000],'gamma': [0.0001, 0.0003, 0.0009]}
grid3 = GridSearchCV(model2, param_grid, verbose=3)
grid3.fit(X21S_train_scaled, y21_train)
```

```
Fitting 5 folds for each of 12 candidates, totalling 60 fits
[CV] C=1, gamma=0.0001 ...............................................

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.

[CV] ................... C=1, gamma=0.0001, score=0.500, total=   0.6s
[CV] C=1, gamma=0.0001 ...............................................

[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.6s remaining:    0.0s

[CV] ................... C=1, gamma=0.0001, score=0.500, total=   0.6s
[CV] C=1, gamma=0.0001 ...............................................

[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    1.2s remaining:    0.0s
```

In [ ]:

Comparing results with grid 1, 2 and 3

In [50]:
```
print(grid.best_params_)
print(grid.best_score_)
```

```
{'C': 50, 'gamma': 0.005}
0.8163287100037111
```

In [51]:
```
# Make predictions with the hypertuned model
predictions = grid.predict(X21S_test_scaled)
```

In [52]:
```
# Calculate classification report
from sklearn.metrics import classification_report
print(classification_report(y21_test, predictions,
                            target_names=target_names))
```

```
                precision    recall  f1-score   support

     CONFIRMED       0.66      0.40      0.50       411
FALSE POSITIVE       0.60      0.79      0.68       455
     CANDIDATE       0.98      1.00      0.99       882

      accuracy                           0.80      1748
     macro avg       0.75      0.73      0.72      1748
  weighted avg       0.80      0.80      0.79      1748
```

In [53]:
```
print(grid2.best_params_)
print(grid2.best_score_)
```

```
{'C': 1000, 'gamma': 0.001}
0.818425763540704
```

In [54]:
```
# Make predictions with the hypertuned model
predictions = grid2.predict(X21S_test_scaled)
# Calculate classification report
from sklearn.metrics import classification_report
print(classification_report(y21_test, predictions,
                            target_names=target_names))
```

```
                precision    recall  f1-score   support

     CONFIRMED       0.66      0.42      0.51       411
FALSE POSITIVE       0.61      0.78      0.68       455
     CANDIDATE       0.98      1.00      0.99       882

      accuracy                           0.80      1748
     macro avg       0.75      0.73      0.73      1748
  weighted avg       0.81      0.80      0.80      1748
```

In [55]:
```
print(grid3.best_params_)
print(grid3.best_score_)
```

```
{'C': 5000, 'gamma': 0.0009}
0.8272002052117975
```

In [56]:
```
# Make predictions with the hypertuned model
predictions = grid3.predict(X21S_test_scaled)
# Calculate classification report
from sklearn.metrics import classification_report
print(classification_report(y21_test, predictions,
                            target_names=target_names))
```

```
                precision    recall  f1-score   support

     CONFIRMED       0.67      0.49      0.57       411
FALSE POSITIVE       0.63      0.75      0.69       455
     CANDIDATE       0.98      1.00      0.99       882

      accuracy                           0.81      1748
     macro avg       0.76      0.75      0.75      1748
  weighted avg       0.81      0.81      0.81      1748
```

Best score ends with 82.72%