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]:
```
Xtemp = df[['koi_fpflag_nt', 'koi_fpflag_ss', 'koi_fpflag_co', 'koi_fpflag_ec', 'koi_period','koi_time0bk','
Xtemp
```

Out[4]:

	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_pr
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 [5]:
```
data = df[['koi_disposition']]
data_binary_encoded = pd.get_dummies(data, columns=["koi_disposition"])
data_binary_encoded.columns = [["candidate","confirmed","false_positive"]]
y = data_binary_encoded
```

In [6]:
```
# Using RandomForestClassifier to find features importance
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier(n_estimators=200)
rf = rf.fit(Xtemp, y)
```

In [7]:
```
# Random Forests in sklearn will automatically calculate feature importance
importances = rf.feature_importances_
importances
```

Out[7]:
```
array([0.1275446 , 0.09728054, 0.12407088, 0.04611336, 0.03970133,
       0.02631832, 0.01728792, 0.01640372, 0.04143043, 0.03037272,
       0.04744568, 0.08161766, 0.03084387, 0.03441217, 0.12485427,
       0.02047564, 0.01661677, 0.01709949, 0.02072587, 0.01949034,
       0.01989443])
```

In [8]:
```
# We can sort the features by their importance
sorted(zip(rf.feature_importances_, Xtemp), reverse=True)
```

Out[8]:
```
[(0.1275445980385035, 'koi_fpflag_nt'),
 (0.12485427465751037, 'koi_model_snr'),
 (0.12407087655657575, 'koi_fpflag_co'),
 (0.09728054080021772, 'koi_fpflag_ss'),
 (0.08161765543333321, 'koi_prad'),
 (0.04744567680861078, 'koi_depth'),
 (0.0461133572376748, 'koi_fpflag_ec'),
 (0.04143042563106936, 'koi_impact'),
 (0.03970133388263601, 'koi_period'),
 (0.03441217169634627, 'koi_insol'),
 (0.03084386723142717, 'koi_teq'),
 (0.030372717309541324, 'koi_duration'),
 (0.02631831713839617, 'koi_time0bk'),
 (0.020725867840814927, 'ra'),
 (0.020475636075140715, 'koi_steff'),
 (0.019894434293683035, 'koi_kepmag'),
 (0.019490337112497626, 'dec'),
 (0.017287922775048026, 'koi_slogg'),
 (0.01709949432229022, 'koi_srad'),
 (0.016616772792990776, 'koi_slogg'),
 (0.01640372236569231, 'koi_srad')]
```

In [9]:
```
# Removing features less than 0.26
X = Xtemp.drop(columns=['ra','dec','koi_kepmag','koi_srad','koi_slogg'])
X
```

Out[9]:

	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	k
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

Create a Train Test Split

Use `koi_disposition` for the y values

In [10]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42)
```

In [11]:
```
X_train.head()
```

Out[11]:

	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	k
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	

Train the Model

In [12]:
```
# Create and score a Random Forest classifier
rf = RandomForestClassifier(n_estimators=200)
rf = rf.fit(X_train, y_train)
rf.score(X_test, y_test)
```

Out[12]:
```
0.8913043478260869
```

In [13]:
```
rf = RandomForestClassifier(n_estimators=500)
rf = rf.fit(X_train, y_train)
rf.score(X_test, y_test)
```

Out[13]:
```
0.8953089244851259
```

In [14]:
```
rf = RandomForestClassifier(n_estimators=700)
rf = rf.fit(X_train, y_train)
rf.score(X_test, y_test)
```

Out[14]:
```
0.8930205949656751
```

Best score with 500 estimators (89.53%) after that it goes down..

Hyperparameter Tuning

In [15]:
```
from sklearn.ensemble import RandomForestRegressor
rf = RandomForestRegressor(random_state = 42)
from pprint import pprint
# Look at parameters used by our current forest
print('Parameters currently in use:\n')
pprint(rf.get_params())
```

```
Parameters currently in use:

{'bootstrap': True,
 'ccp_alpha': 0.0,
 'criterion': 'mse',
 'max_depth': None,
 'max_features': 'auto',
 'max_leaf_nodes': None,
 'max_samples': None,
 'min_impurity_decrease': 0.0,
 'min_impurity_split': None,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'n_estimators': 100,
 'n_jobs': None,
 'oob_score': False,
 'random_state': 42,
 'verbose': 0,
 'warm_start': False}
```

In [16]:
```
from sklearn.model_selection import RandomizedSearchCV
# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 200, stop = 2000, num = 10)]
# Number of features to consider at every split
max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(10, 110, num = 11)]
max_depth.append(None)
# Minimum number of samples required to split a node
min_samples_split = [5, 10, 14]
# Minimum number of samples required at each leaf node
min_samples_leaf = [4, 9, 14]
# Method of selecting samples for training each tree
bootstrap = [True, False]
# Create the random grid
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'bootstrap': bootstrap}
pprint(random_grid)
```

```
{'bootstrap': [True, False],
 'max_depth': [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, None],
 'max_features': ['auto', 'sqrt'],
 'min_samples_leaf': [4, 9, 14],
 'min_samples_split': [5, 10, 14],
 'n_estimators': [200, 400, 600, 800, 1000, 1200, 1400, 1600, 1800, 2000]}
```

In [17]:
```
# Use the random grid to search for best hyperparameters
# First create the base model to tune
rf = RandomForestRegressor()
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores
rf_random = RandomizedSearchCV(estimator = rf, param_distributions = random_grid, n_iter = 100, cv = 3, verb
# Fit the random search model
rf_random.fit(X_train, y_train)
```

```
Fitting 3 folds for each of 100 candidates, totalling 300 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 12 concurrent workers.
[Parallel(n_jobs=-1)]: Done  17 tasks      | elapsed:   21.9s
[Parallel(n_jobs=-1)]: Done 138 tasks      | elapsed:  3.9min
[Parallel(n_jobs=-1)]: Done 300 out of 300 | elapsed:  9.5min finished
```

Out[17]:
```
RandomizedSearchCV(cv=3, estimator=RandomForestRegressor(), n_iter=100,
                   n_jobs=-1,
                   param_distributions={'bootstrap': [True, False],
                                        'max_depth': [10, 20, 30, 40, 50, 60,
                                                      70, 80, 90, 100, 110,
                                                      None],
                                        'max_features': ['auto', 'sqrt'],
                                        'min_samples_leaf': [4, 9, 14],
                                        'min_samples_split': [5, 10, 14],
                                        'n_estimators': [200, 400, 600, 800,
                                                         1000, 1200, 1400, 1600,
                                                         1800, 2000]},
                   random_state=42, verbose=2)
```

In [18]:
```
rf_random.best_params_
```

Out[18]:
```
{'n_estimators': 1600,
 'min_samples_split': 14,
 'min_samples_leaf': 9,
 'max_features': 'auto',
 'max_depth': None,
 'bootstrap': True}
```

In [19]:
```
print(rf_random.best_params_)
print(rf_random.best_score_)
```

```
{'n_estimators': 1600, 'min_samples_split': 14, 'min_samples_leaf': 9, 'max_features': 'auto', 'max_dept
h': None, 'bootstrap': True}
0.7232106682662275
```

In [20]:
```
from sklearn.model_selection import GridSearchCV
# Create the parameter grid based on the results of random search
param_grid = {
    'bootstrap': [True],
    'max_features': [13, 14],
    'min_samples_leaf': [8, 9, 10],
    'min_samples_split': [3, 5, 7],
    'n_estimators': [200, 300, 500, 1600]
}
```

In [21]:
```
# Create a based model
rf = RandomForestRegressor()
```

In [22]:
```
# Instantiate the grid search model
grid_search = GridSearchCV(estimator = rf, param_grid = param_grid,
                           cv = 3, n_jobs = -1, verbose = 2)
```

In [23]:
```
# Fit the grid search to the data
grid_search.fit(X_train, y_train)
grid_search.best_params_
```

```
Fitting 3 folds for each of 72 candidates, totalling 216 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 12 concurrent workers.
[Parallel(n_jobs=-1)]: Done  17 tasks      | elapsed:   17.8s
[Parallel(n_jobs=-1)]: Done 138 tasks      | elapsed:  2.9min
[Parallel(n_jobs=-1)]: Done 216 out of 216 | elapsed:  4.9min finished
```

Out[23]:
```
{'bootstrap': True,
 'max_features': 13,
 'min_samples_leaf': 8,
 'min_samples_split': 5,
 'n_estimators': 300}
```

In [24]:
```
print(grid_search.best_params_)
print(grid_search.best_score_)
```

```
{'bootstrap': True, 'max_features': 13, 'min_samples_leaf': 8, 'min_samples_split': 5, 'n_estimators': 30
0}
0.724903625743842
```

Unable to bring best score with Hyperparameter tuning also it's time consuming