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

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([0.12346593, 0.09419654, 0.12582219, 0.0451646 , 0.04132743,
       0.02811616, 0.01605033, 0.01679236, 0.03981774, 0.02874018,
       0.04922634, 0.08625493, 0.0311314 , 0.03333052, 0.12676818,
       0.02028909, 0.0166809 , 0.01735946, 0.02046538, 0.01906963,
       0.01993072])
```

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

Out[8]:
```
[(0.12676817914075886, 'koi_model_snr'),
 (0.12582219050085736, 'koi_fpflag_co'),
 (0.12346592514282172, 'koi_fpflag_nt'),
 (0.09419654311719489, 'koi_fpflag_ss'),
 (0.08625493046596698, 'koi_prad'),
 (0.04922633953812942, 'koi_depth'),
 (0.0451646027973805, 'koi_fpflag_ec'),
 (0.04132743071504333, 'koi_period'),
 (0.039817735262521955, 'koi_impact'),
 (0.0333305187187091, 'koi_insol'),
 (0.03113140411547599, 'koi_teq'),
 (0.02874018139161716, 'koi_duration'),
 (0.02811615911611042, 'koi_time0bk'),
 (0.020465383335579426, 'ra'),
 (0.0202890859067854, 'koi_steff'),
 (0.019930715102662335, 'koi_kepmag'),
 (0.019069625020265706, 'dec'),
 (0.017359462879260164, 'koi_srad'),
 (0.016792361851902703, 'koi_srad'),
 (0.016680897721283037, 'koi_slogg'),
 (0.01605032815967361, 'koi_slogg')]
```

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 [23]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=1)
```

In [24]:
```
X_train.head()
```

Out[24]:

	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

Train the Model

In [25]:
```
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression()
classifier
```

Out[25]:
```
LogisticRegression()
```

In [26]:
```
classifier.fit(X_train, y_train)
```

Out[26]:
```
LogisticRegression()
```

In [27]:
```
print(f"Training Data Score: {classifier.score(X_train, y_train)}")
print(f"Testing Data Score: {classifier.score(X_test, y_test)}")
```

```
Training Data Score: 0.6004196070951745
Testing Data Score: 0.6098398169336384
```

In [15]:
```
from sklearn.preprocessing import LabelEncoder
label_encoder = LabelEncoder()
label_encoder.fit(y_train)
encoded_y_train = label_encoder.transform(y_train)
encoded_y_test = label_encoder.transform(y_test)
```

In [ ]:
```
# from tensorflow.keras.utils import to_categorical
# y_train_categorical = to_categorical(encoded_y_train)
# y_test_categorical = to_categorical(encoded_y_test)
```

In [ ]:
```
# # Using StandardScaler
# from sklearn.preprocessing import StandardScaler
# X2_scaler = StandardScaler().fit(X_train)
# X2_train_scaled = X2_scaler.transform(X_train)
# X2_test_scaled = X2_scaler.transform(X_test)
```

In [16]:
```
classifier2 = LogisticRegression()
classifier2.fit(X_train, encoded_y_train)
```

Out[16]:
```
LogisticRegression()
```

In [17]:
```
print(f"Training Data Score: {classifier2.score(X_train, encoded_y_train)}")
print(f"Testing Data Score: {classifier2.score(X_test, encoded_y_test)}")
```

```
Training Data Score: 0.6004196070951745
Testing Data Score: 0.6098398169336384
```

In [ ]:
```
# predictions = classifier.predict(X_test)
# print(f"First 10 Predictions:   {predictions[:10]}")
# print(f"First 10 Actual labels: {y_test[:10]}")
```

In [ ]:
```
# pd.DataFrame({"Prediction": predictions, "Actual": y_test['koi_disposition']}).reset_index(drop=True)
```

Logistic Regression scores 60.98%. I have also noticed random state 1 scores better than random state 42