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_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 [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([0.12662648, 0.10317236, 0.12504482, 0.04473652, 0.03843636,
       0.02586495, 0.01612839, 0.01645708, 0.04245329, 0.030154  ,
       0.0521294 , 0.07478961, 0.03029361, 0.03502636, 0.12446381,
       0.02117092, 0.01678824, 0.01741316, 0.02071905, 0.01893659,
       0.01919501])
```

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

Out[8]:
```
[(0.12662647597194454, 'koi_fpflag_nt'),
 (0.12504482068001485, 'koi_fpflag_co'),
 (0.12446381064718784, 'koi_model_snr'),
 (0.10317236467561273, 'koi_fpflag_ss'),
 (0.07478960721157546, 'koi_prad'),
 (0.052129398937604494, 'koi_depth'),
 (0.04473651607033153, 'koi_fpflag_ec'),
 (0.04245328880577117, 'koi_impact'),
 (0.03843635602170486, 'koi_period'),
 (0.035026364740105446, 'koi_insol'),
 (0.030293612548449096, 'koi_teq'),
 (0.030153999545387552, 'koi_duration'),
 (0.025864949628388614, 'koi_time0bk'),
 (0.021170915307488915, 'koi_steff'),
 (0.020719045798171895, 'ra'),
 (0.01919500695295812, 'koi_kepmag'),
 (0.018936587637840292, 'dec'),
 (0.01741316142283267, 'koi_srad'),
 (0.01678823980007196, 'koi_slogg'),
 (0.01645708396003472, 'koi_srad'),
 (0.01612839363652317, '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

Select your features (columns)

In [10]:
```
# Set features. This will also be used as your x values.
selected_features = X.columns
```

Create a Train Test Split

Use `koi_disposition` for the y values

In [11]:
```
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 [12]:
```
X_train.head()
```

Out[12]:

	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	

Pre-processing

Scale the data using the MinMaxScaler and perform some feature selection

In [13]:
```
# Scale your data
from sklearn.preprocessing import StandardScaler
X_scaler = StandardScaler().fit(X_train)
y_scaler = StandardScaler().fit(y_train)
```

In [14]:
```
X_train_scaled = X_scaler.transform(X_train)
X_test_scaled = X_scaler.transform(X_test)
y_train_scaled = y_scaler.transform(y_train)
y_test_scaled = y_scaler.transform(y_test)
```

Train the Model

In [15]:
```
from sklearn.linear_model import LinearRegression
model = LinearRegression()
model.fit(X_train_scaled, y_train_scaled)
```

Out[15]:
```
LinearRegression()
```

In [16]:
```
print(f"Training Data Score: {model.score(X_train_scaled, y_train)}")
print(f"Testing Data Score: {model.score(X_test_scaled, y_test)}")
```

```
Training Data Score: -0.7016866962127946
Testing Data Score: -0.6855398916933263
```

In [17]:
```
predictions = model.predict(X_test_scaled)
# predictions
```

Comparing the scores

In [18]:
```
from sklearn.metrics import mean_squared_error

MSE = mean_squared_error(y_test_scaled, predictions)
r2 = model.score(X_test_scaled, y_test_scaled)
print(f"MSE: {MSE}, R2: {r2}")
```

```
MSE: 0.560809699443665, R2: 0.44829163453463244
```

In [19]:
```
# LASSO model
# Note: Use an alpha of .01 when creating the model for this activity
from sklearn.linear_model import Lasso

lasso = Lasso(alpha=.01).fit(X_train_scaled, y_train_scaled)

predictions = lasso.predict(X_test_scaled)

MSE = mean_squared_error(y_test_scaled, predictions)
r2 = lasso.score(X_test_scaled, y_test_scaled)
print(f"MSE: {MSE}, R2: {r2}")
```

```
MSE: 0.5605954827898855, R2: 0.448535814565778
```

In [20]:
```
# Ridge model
# Note: Use an alpha of .01 when creating the model for this activity
from sklearn.linear_model import Ridge

ridge = Ridge(alpha=.01).fit(X_train_scaled, y_train_scaled)

predictions = ridge.predict(X_test_scaled)

MSE = mean_squared_error(y_test_scaled, predictions)
r2 = ridge.score(X_test_scaled, y_test_scaled)

print(f"MSE: {MSE}, R2: {r2}")
```

```
MSE: 0.560809726274668, R2: 0.44829160919997424
```

In [21]:
```
# ElasticNet model
# Note: Use an alpha of .01 when creating the model for this activity
from sklearn.linear_model import ElasticNet

elasticnet = ElasticNet(alpha=.01).fit(X_train_scaled, y_train_scaled)
predictions = elasticnet.predict(X_test_scaled)

MSE = mean_squared_error(y_test_scaled, predictions)
r2 = elasticnet.score(X_test_scaled, y_test_scaled)

print(f"MSE: {MSE}, R2: {r2}")
```

```
MSE: 0.5605503626540483, R2: 0.44856466122540367
```

In [22]:
```
# Create and score a decision tree classifier
from sklearn import tree
clf = tree.DecisionTreeClassifier()
clf = clf.fit(X_train, y_train)
clf.score(X_test, y_test)
```

Out[22]:
```
0.8649885583524027
```

In [23]:
```
# 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[23]:
```
0.8901601830663616
```

In [24]:
```
rf = RandomForestClassifier(n_estimators=500)
rf = rf.fit(X_train, y_train)
rf.score(X_test, y_test)
```

Out[24]:
```
0.8930205949656751
```

Decision Tree Classifier (86.49%) and Random Forest Classifier (89.30%) scores are better than other models.