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]:
```
# Dependencies
import numpy as np
import pandas as pd
```

In [2]:
```
# import sys
# print(sys.version)
```

In [3]:
```
# import tensorflow
# tensorflow.keras.__version__
```

In [4]:
```
import warnings
warnings.simplefilter('ignore')
```

Read the CSV and Perform Basic Data Cleaning

In [5]:
```
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[5]:

	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 [6]:
```
# https://exoplanetarchive.ipac.caltech.edu/docs/API_kepcandidate_columns.html#stellar_param
df.columns
```

Out[6]:
```
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 [7]:
```python
Xtemp = df[['koi_fpflag_nt', 'koi_fpflag_ss', 'koi_fpflag_co', 'koi_fpflag_ec', 'koi_period','koi_time0bk','
Xtemp
```

Out[7]:

	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 [8]:
```python
y = df[['koi_disposition']]
data_binary_encoded = pd.get_dummies(y, columns=["koi_disposition"])
data_binary_encoded.columns = [["candidate","confirmed","false_positive"]]
```

In [9]:
```python
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier(n_estimators=200)
rf = rf.fit(Xtemp, data_binary_encoded)
```

In [10]:
```python
# Random Forests in sklearn will automatically calculate feature importance
importances = rf.feature_importances_
importances
```

Out[10]:
```
array([0.12752843, 0.08704186, 0.12940321, 0.04593622, 0.03687452,
       0.02781514, 0.01640036, 0.01679574, 0.0404346 , 0.02933802,
       0.05497453, 0.08285962, 0.0319641 , 0.03005543, 0.12911168,
       0.01991671, 0.01639255, 0.01777266, 0.02057601, 0.01919567,
       0.01961292])
```

In [11]:
```python
# We can sort the features by their importance
sorted(zip(rf.feature_importances_, Xtemp), reverse=True)
```

Out[11]:
```
[(0.1294032116996266, 'koi_fpflag_co'),
 (0.12911167730728298, 'koi_model_snr'),
 (0.12752843471127012, 'koi_fpflag_nt'),
 (0.08704186358110598, 'koi_fpflag_ss'),
 (0.08285962337499243, 'koi_prad'),
 (0.05497453333589291, 'koi_depth'),
 (0.045936223412756876, 'koi_fpflag_ec'),
 (0.0404345958488925, 'koi_impact'),
 (0.03687451966067239, 'koi_period'),
 (0.03196410473017362, 'koi_teq'),
 (0.030055430705632446, 'koi_insol'),
 (0.0293380245843238, 'koi_duration'),
 (0.027815144239945407, 'koi_time0bk'),
 (0.0205760105080307, 'ra'),
 (0.019916705989532525, 'koi_steff'),
 (0.01961291982187025, 'koi_kepmag'),
 (0.01919567013134733, 'dec'),
 (0.017772659476365783, 'koi_srad'),
 (0.016795738843094037, 'koi_srad'),
 (0.016400356497123885, 'koi_slogg'),
 (0.01639255154006741, 'koi_slogg')]
```

In [12]:
```python
X = Xtemp.drop(columns=['ra','dec','koi_kepmag','koi_srad','koi_slogg'])
X
```

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
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 [ ]:

Select your features (columns)

In [13]:
```python
# Set features. This will also be used as your x values.
selected_features = X.columns
selected_features
```

Out[13]:
```
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

Use `koi_disposition` for the y values

In [14]:
```python
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 [15]:
```python
X_train.head()
```

Out[15]:

	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

Pre-processing

Scale the data using the MinMaxScaler and perform some feature selection

In [16]:
```python
# Scale your data
from sklearn.preprocessing import LabelEncoder, MinMaxScaler, StandardScaler
from tensorflow.keras.utils import to_categorical
```

In [17]:
```
# Using MinMaxScaler
X_scaler = MinMaxScaler().fit(X_train)
X_train_scaled = X_scaler.transform(X_train)
X_test_scaled = X_scaler.transform(X_test)
```

In [18]:
```
# Step 1: Label-encode data set
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 [19]:
```
# Step 2: Convert encoded labels to one-hot-encoding
y_train_categorical = to_categorical(encoded_y_train)
y_test_categorical = to_categorical(encoded_y_test)
```

In [20]:
```
# print(X_train_scaled.shape, y_train_categorical.shape)
```

In [21]:
```
# Using StandardScaler
X2_scaler = StandardScaler().fit(X_train)
X2_train_scaled = X2_scaler.transform(X_train)
X2_test_scaled = X2_scaler.transform(X_test)
```

In [ ]:

Train the Model

In [22]:
```
# create deep learning model
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
```

In [23]:
```
model = Sequential()
# give no. of columns in y_train_categorical as input_dim
model.add(Dense(units=100, activation='relu', input_dim=14))
model.add(Dense(units=100, activation='relu'))
# give of columns in y_train_categorical as units
model.add(Dense(units=3, activation='softmax'))
```

In [24]:
```
# Compile and fit the model
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 100)               1500
_________________________________________________________________
dense_1 (Dense)              (None, 100)               10100
_________________________________________________________________
dense_2 (Dense)              (None, 3)                 303
=================================================================
Total params: 11,903
Trainable params: 11,903
Non-trainable params: 0
_________________________________________________________________
```

In [25]:
```
model2 = Sequential()
model2.add(Dense(units=100, activation='relu', input_dim=14))
model2.add(Dense(units=100, activation='relu'))
model2.add(Dense(units=3, activation='softmax'))
```

In [26]:
```python
# Compile and fit the model
model2.compile(optimizer='adam',
               loss='categorical_crossentropy',
               metrics=['accuracy'])
model2.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_3 (Dense)              (None, 100)               1500      
_________________________________________________________________
dense_4 (Dense)              (None, 100)               10100     
_________________________________________________________________
dense_5 (Dense)              (None, 3)                 303       
=================================================================
Total params: 11,903
Trainable params: 11,903
Non-trainable params: 0
_________________________________________________________________
```

In [27]:
```python
model.fit(
    X_train_scaled,
    y_train_categorical,
    epochs=60,
    shuffle=True,
    verbose=2
)
```

```
Train on 5243 samples
Epoch 1/60
5243/5243 - 1s - loss: 0.5349 - accuracy: 0.7063
Epoch 2/60
5243/5243 - 0s - loss: 0.3943 - accuracy: 0.7822
Epoch 3/60
5243/5243 - 0s - loss: 0.3843 - accuracy: 0.7829
Epoch 4/60
5243/5243 - 0s - loss: 0.3792 - accuracy: 0.7902
Epoch 5/60
5243/5243 - 0s - loss: 0.3760 - accuracy: 0.7974
Epoch 6/60
5243/5243 - 0s - loss: 0.3727 - accuracy: 0.7952
Epoch 7/60
5243/5243 - 0s - loss: 0.3706 - accuracy: 0.8013
Epoch 8/60
5243/5243 - 0s - loss: 0.3676 - accuracy: 0.8053
Epoch 9/60
5243/5243 - 0s - loss: 0.3636 - accuracy: 0.8053
```

In [ ]:
```python
# model_loss, model_accuracy = model.evaluate(
#     X_test_scaled, y_test_categorical, verbose=2)
# print(
#     f"Deep Neural Network - Loss: {model_loss}, Accuracy: {model_accuracy}")
```

In [ ]:
```python
# model.fit(
#     X_train_scaled,
#     y_train_categorical,
#     epochs=80,
#     shuffle=True,
#     verbose=2
# )
```

In [ ]:
```python
# model_loss, model_accuracy = model.evaluate(
#     X_test_scaled, y_test_categorical, verbose=2)
# print(
#     f"Deep Neural Network - Loss: {model_loss}, Accuracy: {model_accuracy}")
```

In [ ]:
```python
# model.fit(
#     X_train_scaled,
#     y_train_categorical,
#     epochs=100,
#     shuffle=True,
#     verbose=2
# )
```

In [ ]:
```
# model_loss, model_accuracy = model.evaluate(
#     X_test_scaled, y_test_categorical, verbose=2)
# print(
#     f"Deep Neural Network - Loss: {model_loss}, Accuracy: {model_accuracy}")
```

In [28]:
```
# This one scores better than epochs 60,80,and 100
model.fit(
    X_train_scaled,
    y_train_categorical,
    epochs=150,
    shuffle=True,
    verbose=2
)
```

```
Train on 5243 samples
Epoch 1/150
5243/5243 - 0s - loss: 0.2666 - accuracy: 0.8802
Epoch 2/150
5243/5243 - 0s - loss: 0.2762 - accuracy: 0.8753
Epoch 3/150
5243/5243 - 0s - loss: 0.2685 - accuracy: 0.8835
Epoch 4/150
5243/5243 - 0s - loss: 0.2681 - accuracy: 0.8806
Epoch 5/150
5243/5243 - 0s - loss: 0.2631 - accuracy: 0.8869
Epoch 6/150
5243/5243 - 0s - loss: 0.2697 - accuracy: 0.8766
Epoch 7/150
5243/5243 - 0s - loss: 0.2637 - accuracy: 0.8795
Epoch 8/150
5243/5243 - 0s - loss: 0.2619 - accuracy: 0.8804
Epoch 9/150
5243/5243 - 0s - loss: 0.2662 - accuracy: 0.8791
```

In [29]:
```
# Testing with StandardScaler
model2.fit(
    X2_train_scaled,
    y_train_categorical,
    epochs=150,
    shuffle=True,
    verbose=2
)
```

```
Train on 5243 samples
Epoch 1/150
5243/5243 - 1s - loss: 0.4786 - accuracy: 0.7706
Epoch 2/150
5243/5243 - 0s - loss: 0.3630 - accuracy: 0.8053
Epoch 3/150
5243/5243 - 0s - loss: 0.3530 - accuracy: 0.8156
Epoch 4/150
5243/5243 - 0s - loss: 0.3431 - accuracy: 0.8228
Epoch 5/150
5243/5243 - 0s - loss: 0.3362 - accuracy: 0.8247
Epoch 6/150
5243/5243 - 0s - loss: 0.3314 - accuracy: 0.8272
Epoch 7/150
5243/5243 - 0s - loss: 0.3349 - accuracy: 0.8270
Epoch 8/150
5243/5243 - 0s - loss: 0.3262 - accuracy: 0.8302
Epoch 9/150
5243/5243 - 0s - loss: 0.3212 - accuracy: 0.8302
```

In [30]:
```
# MinMaxScaler Accuracy
model_loss, model_accuracy = model.evaluate(
    X_test_scaled, y_test_categorical, verbose=2)
print(
    f"Deep Neural Network - Loss: {model_loss}, Accuracy: {model_accuracy}")
```

```
1748/1 - 0s - loss: 0.2667 - accuracy: 0.8902
Deep Neural Network - Loss: 0.2905025056284134, Accuracy: 0.8901602029800415
```

In [31]:
```
# StandardScaler
model_loss, model_accuracy = model2.evaluate(
    X2_test_scaled, y_test_categorical, verbose=2)
print(
    f"Deep Neural Network - Loss: {model_loss}, Accuracy: {model_accuracy}")
```

```
1748/1 - 0s - loss: 0.3756 - accuracy: 0.8930
Deep Neural Network - Loss: 0.41243264065454427, Accuracy: 0.8930205702781677
```

In [32]:
```
# make predictions and print the results
encoded_predictions = model.predict_classes(X_test_scaled[:5])
prediction_labels = label_encoder.inverse_transform(encoded_predictions)
print(f"Predicted classes: {prediction_labels}")
print(f"Actual Labels: {y_test.values[:5].tolist()}")
```

```
Predicted classes: ['CONFIRMED' 'FALSE POSITIVE' 'FALSE POSITIVE' 'CONFIRMED'
 'FALSE POSITIVE']
Actual Labels: [['CONFIRMED'], ['FALSE POSITIVE'], ['FALSE POSITIVE'], ['CONFIRMED'], ['FALSE POSITIVE']]
```

Deep Learning with MinMaxScaler scored (89.58%) better than StandardScaler