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

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

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]:
```
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]:
```
y = df[['koi_disposition']]
```

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

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

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

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

based on past history directly removing insignificant features based on feature_importance

In [7]:
```
X = Xtemp.drop(columns=['ra','dec','koi_kepmag','koi_srad','koi_slogg'])
X
```

Out[7]:

	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 [8]:
```
# Set features. This will also be used as your x values.
selected_features = X.columns
selected_features
```

Out[8]:
```
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 [9]:
```
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 [10]:
```
print(X.shape, y.shape)
```

```
(6991, 14) (6991, 1)
```

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
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 [12]:
```
# Scale your data
from sklearn.preprocessing import LabelEncoder, MinMaxScaler
from tensorflow.keras.utils import to_categorical
```

In [13]:
```
X_scaler = MinMaxScaler().fit(X_train)
X_train_scaled = X_scaler.transform(X_train)
X_test_scaled = X_scaler.transform(X_test)
```

In [14]:
```
# 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 [15]:
```
# 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 [16]:
```
print(X_train_scaled.shape, y_train_categorical.shape)
```

```
(5243, 14) (5243, 3)
```

In [ ]:

Train the Model

In [17]:
```
# create deep learning model
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
```

In [18]:
```
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 [19]:
```
# Compile and fit the model
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

In [20]:
```
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 [21]:
```
model.fit(
    X_train_scaled,
    y_train_categorical,
    epochs=15,
    shuffle=True,
    verbose=2
)
```

```
Train on 5243 samples
Epoch 1/15
5243/5243 - 1s - loss: 0.5409 - accuracy: 0.7044
Epoch 2/15
5243/5243 - 0s - loss: 0.3787 - accuracy: 0.7833
Epoch 3/15
5243/5243 - 0s - loss: 0.3708 - accuracy: 0.7910
Epoch 4/15
5243/5243 - 0s - loss: 0.3657 - accuracy: 0.7978
Epoch 5/15
5243/5243 - 0s - loss: 0.3618 - accuracy: 0.7940
Epoch 6/15
5243/5243 - 0s - loss: 0.3589 - accuracy: 0.8039
Epoch 7/15
5243/5243 - 0s - loss: 0.3557 - accuracy: 0.8051
Epoch 8/15
5243/5243 - 0s - loss: 0.3530 - accuracy: 0.8070
Epoch 9/15
5243/5243 - 0s - loss: 0.3503 - accuracy: 0.8135
Epoch 10/15
5243/5243 - 0s - loss: 0.3494 - accuracy: 0.8114
Epoch 11/15
5243/5243 - 0s - loss: 0.3467 - accuracy: 0.8148
Epoch 12/15
5243/5243 - 0s - loss: 0.3440 - accuracy: 0.8177
Epoch 13/15
5243/5243 - 0s - loss: 0.3415 - accuracy: 0.8150
Epoch 14/15
5243/5243 - 0s - loss: 0.3388 - accuracy: 0.8230
Epoch 15/15
5243/5243 - 0s - loss: 0.3365 - accuracy: 0.8230
```

Out[21]:
```
<tensorflow.python.keras.callbacks.History at 0x7f8813038390>
```

In [22]:
```
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: 0.3523 - accuracy: 0.8026
Deep Neural Network - Loss: 0.3657203559471759, Accuracy: 0.8026315569877625
```

In [ ]:
```
# model.fit(
#     X_train_scaled,
#     y_train_categorical,
#     epochs=80,
#     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 [23]:
```python
model.fit(
    X_train_scaled,
    y_train_categorical,
    epochs=100,
    shuffle=True,
    verbose=2
)
```

```
Train on 5243 samples
Epoch 1/100
5243/5243 - 0s - loss: 0.3362 - accuracy: 0.8230
Epoch 2/100
5243/5243 - 0s - loss: 0.3304 - accuracy: 0.8270
Epoch 3/100
5243/5243 - 0s - loss: 0.3293 - accuracy: 0.8282
Epoch 4/100
5243/5243 - 0s - loss: 0.3250 - accuracy: 0.8325
Epoch 5/100
5243/5243 - 0s - loss: 0.3202 - accuracy: 0.8390
Epoch 6/100
5243/5243 - 0s - loss: 0.3201 - accuracy: 0.8369
Epoch 7/100
5243/5243 - 0s - loss: 0.3187 - accuracy: 0.8371
Epoch 8/100
5243/5243 - 0s - loss: 0.3163 - accuracy: 0.8411
Epoch 9/100
5243/5243 - 0s - loss: 0.3118 - accuracy: 0.8436
```

In [24]:
```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}")
```

```
1748/1 - 0s - loss: 0.3060 - accuracy: 0.8993
Deep Neural Network - Loss: 0.2669288119134968, Accuracy: 0.8993135094642639
```

In [25]:
```python
# 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: ['FALSE POSITIVE' 'CANDIDATE' 'FALSE POSITIVE' 'FALSE POSITIVE'
 'FALSE POSITIVE']
Actual Labels: [['FALSE POSITIVE'], ['CANDIDATE'], ['FALSE POSITIVE'], ['FALSE POSITIVE'], ['FALSE POSITIV
E']]
```

In [ ]:

Deep Learning model with random state 42 (V6) accuracy slightly better than random state 1 (V5). The best accuracy is 89.93% and loss rate is 26.69%