In [4]:
```
import pandas as pd
import numpy as np
```

In [5]:
```
df = pd.read_csv("koi.csv")
```

In [6]:
```
df.head(5)
```

Out[6]:

	kepid	kepoi_name	kepler_name	koi_disposition	koi_pdisposition	koi_score	koi_fpf
0	10797460	K00752.01	Kepler-227 b	CONFIRMED	CANDIDATE	1.000	
1	10797460	K00752.02	Kepler-227 c	CONFIRMED	CANDIDATE	0.969	
2	10811496	K00753.01	NaN	CANDIDATE	CANDIDATE	0.000	
3	10848459	K00754.01	NaN	FALSE POSITIVE	FALSE POSITIVE	0.000	
4	10854555	K00755.01	Kepler-664 b	CONFIRMED	CANDIDATE	1.000	

5 rows × 49 columns

In [7]:
```
df.columns
```

Out[7]:
```
Index(['kepid', 'kepoi_name', 'kepler_name', 'koi_disposition',
       'koi_pdisposition', 'koi_score', '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_teq_err1', 'koi_teq_err2', 'koi_insol',
       'koi_insol_err1', 'koi_insol_err2', 'koi_model_snr', 'koi_tce_plnt_num',
       'koi_tce_delivname', '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')
```

Dropping all the irrelevant columns - identifiers, light curve fit errors, etc

In [8]:
```
drop_cols = [
    "kepid", "kepoi_name", "kepler_name",
    "koi_pdisposition", "koi_score", "koi_tce_delivname",
    "koi_fpflag_nt", "koi_fpflag_ss", "koi_fpflag_co", "koi_fpflag_ec",
    "koi_time0bk", 'koi_model_snr', 'koi_tce_plnt_num', 'ra', 'dec', 'koi_kepmag'
]

err_cols = [c for c in df.columns if c.endswith("_err1") or c.endswith("_err2")]

df1 = df.drop(columns = drop_cols + err_cols, errors="ignore")
```

In [9]:
```
df1.columns
```

Out[9]:
```
Index(['koi_disposition', 'koi_period', 'koi_impact', 'koi_duration',
       'koi_depth', 'koi_prad', 'koi_teq', 'koi_insol', 'koi_steff',
       'koi_slogg', 'koi_srad'],
      dtype='object')
```

In [10]:
```
df1.head(5)
```

Out[10]:

	koi_disposition	koi_period	koi_impact	koi_duration	koi_depth	koi_prad	koi_teq	koi_
0	CONFIRMED	9.488036	0.146	2.95750	615.8	2.26	793.0	9
1	CONFIRMED	54.418383	0.586	4.50700	874.8	2.83	443.0	
2	CANDIDATE	19.899140	0.969	1.78220	10829.0	14.60	638.0	3
3	FALSE POSITIVE	1.736952	1.276	2.40641	8079.2	33.46	1395.0	89
4	CONFIRMED	2.525592	0.701	1.65450	603.3	2.75	1406.0	92

In [11]:
```
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9564 entries, 0 to 9563
Data columns (total 11 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   koi_disposition  9564 non-null   object 
 1   koi_period       9564 non-null   float64
 2   koi_impact       9201 non-null   float64
 3   koi_duration     9564 non-null   float64
 4   koi_depth        9201 non-null   float64
 5   koi_prad         9201 non-null   float64
 6   koi_teq          9201 non-null   float64
 7   koi_insol        9243 non-null   float64
 8   koi_steff        9201 non-null   float64
 9   koi_slogg        9201 non-null   float64
 10  koi_srad         9201 non-null   float64
dtypes: float64(10), object(1)
memory usage: 822.0+ KB
```

In [12]:
```
miss_per_col = df1.isna().mean()*100
print("Missing % per column:\n", miss_per_col)
```

```
Missing % per column:
 koi_disposition    0.000000
koi_period         0.000000
koi_impact         3.795483
koi_duration       0.000000
koi_depth          3.795483
koi_prad           3.795483
koi_teq            3.795483
koi_insol          3.356336
koi_steff          3.795483
koi_slogg          3.795483
koi_srad           3.795483
dtype: float64
```

In [13]:
```
complete = df1.dropna()
incomplete = df1[df1.isna().any(axis=1)]

comp_dist = complete['koi_disposition'].value_counts(normalize=True)
incomp_dist = incomplete['koi_disposition'].value_counts(normalize=True)

print("Complete data disposition distribution:\n", comp_dist)
print("\nIncomplete data disposition distribution:\n", incomp_dist)
```

```
Complete data disposition distribution:
 koi_disposition
FALSE POSITIVE    0.497989
CONFIRMED         0.298228
CANDIDATE         0.203782
Name: proportion, dtype: float64

Incomplete data disposition distribution:
 koi_disposition
FALSE POSITIVE    0.707989
CANDIDATE         0.286501
CONFIRMED         0.005510
Name: proportion, dtype: float64
```

In [14]:
```
from scipy.stats import chi2_contingency

table = pd.crosstab(df1['koi_disposition'], df1.isna().any(axis=1))
chi2, p, dof, expected = chi2_contingency(table)
p
```

Out[14]:
```
np.float64(1.7869303561526572e-32)
```

In [15]:
```
print("No. of K_disposition values: ",df1["koi_disposition"].value_counts())
```

```
No. of K_disposition values:  koi_disposition
FALSE POSITIVE    4839
CONFIRMED         2746
CANDIDATE         1979
Name: count, dtype: int64
```

In [16]:
```
from sklearn.model_selection import train_test_split

x = df1.drop(columns=["koi_disposition"], errors="ignore")
y = df1["koi_disposition"]
```

```
x_train, x_test, y_train, y_test  = train_test_split(x, y, test_size=0.2, random_st
```

In [18]:
```
x_train.head(5)
```

Out[18]:

	koi_period	koi_impact	koi_duration	koi_depth	koi_prad	koi_teq	koi_insol	koi_ste
8117	0.580725	0.292	1.8200	64.2	0.75	2191.0	5418.89	5989
5245	14.532605	0.112	2.4212	328.9	1.69	746.0	73.37	6003
5865	1.339670	0.602	1.8950	28.6	1.27	3448.0	33385.02	8867
4050	160.412202	0.903	4.9200	2538.0	4.45	270.0	1.26	5269
5130	11.937835	0.036	4.6520	246.0	1.15	671.0	47.91	5469

In [25]:
```
print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(7651, 10)
(1913, 10)
(7651,)
(1913,)
```

In [19]:
```
y_train.head(5)
```

Out[19]:
```
8117    FALSE POSITIVE
5245         CONFIRMED
5865    FALSE POSITIVE
4050    FALSE POSITIVE
5130         CANDIDATE
Name: koi_disposition, dtype: object
```

In [31]:
```
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.ensemble import RandomForestClassifier

pipe = Pipeline([
    ('imputer', SimpleImputer(strategy='median', add_indicator=True)),
    ('clf', RandomForestClassifier(class_weight='balanced'))
])

pipe.fit(x_train, y_train)
y_pred = pipe.predict(x_test)
```

In [32]:
```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

acc  = accuracy_score(y_test, y_pred)
prec = precision_score(y_test, y_pred, average='macro')
rec  = recall_score(y_test, y_pred, average='macro')
f1   = f1_score(y_test, y_pred, average='macro')
```

```
print(acc, prec, rec, f1)
print(classification_report(y_test, y_pred))
```

```
0.7208572922111867 0.6651573298613878 0.6601180320401993 0.6592360293838311
                precision    recall  f1-score   support

     CANDIDATE       0.48      0.36      0.41       405
     CONFIRMED       0.73      0.78      0.75       569
FALSE POSITIVE       0.79      0.84      0.82       939

      accuracy                           0.72      1913
     macro avg       0.67      0.66      0.66      1913
  weighted avg       0.71      0.72      0.71      1913
```