# Comparison of Neural Networks, Decision Tree, and Support Vector Machine to classify exoplanets using NASA Kepler Data

**Prepared by:**

Nor El Islam Messedad

**Date:** 10/03/2023

## 1. Abstract

This project aims to explore the use of machine learning algorithms to automatically classify exoplanets using data from the Kepler Mission dataset. The classification task is to predict the exoplanet's disposition label, which indicates whether the observed signal is due to 'CANDIDATE', 'FALSE POSITIVE', or 'CONFIRMED'. The performance of three machine learning algorithms - Neural Networks, Decision Tree, and Support Vector Machine – were evaluated using standard evaluation metrics. The best-performing algorithm using K-fold cross validation was SVM up to 0.852 when the hyperparameters are optimized using GridSearchCv. When implementing the model using the same hyperparameters SVM also outperformed Decision Tree and NN with 0.752 average accuracy.

## 2. Background and problem to be addressed.

In recent times, astronomy has experienced significant advancements aided by modern instruments, resulting in a vast amount of data for astronomers to work with. However, this abundance of data also presents several challenges when applying state-of-the-art technologies to gauge matters and build predictive models [1].

In exoplanet research, scientists believe that there could be planets orbiting stars besides the sun, which may have natural conditions suitable for supporting life. However, the foremost goal to discover such habitable planets is to locate new exoplanets in the universe [2]. This is also underpinned by the Kepler Mission which collected substantial data in the field of exoplanet research and provided wealth of information about exoplanet populations and their characteristics for further research.

The Kepler Mission was a NASA space observatory designed to discover Earth-size planets orbiting other stars, also known as exoplanets. During its mission from 2009 to 2018, the spacecraft monitored over 150,000 stars and surveying a substantial number of stars in the Milky-Way Galaxy. The primary objective of the Kepler Mission is to conduct a survey within our segment of the Milky Way galaxy and identify numerous planets that are smaller or equal in size to Earth and are located within or around the habitable zone. The mission aims to estimate the proportion of stars, which is roughly hundreds of billions in our galaxy, that could potentially possess these planets [3]. Although Kepler was officially decommissioned in October 2018 due to fuel depletion, the statistical data it gathered has the potential to reveal new exoplanets for several years to come [4].

The Kepler mission data has created an opportunity to explore exoplanets in a new way, with the aim of automating the analysis of scientific observations and generating data for exoplanet identification[4]. Therefore, we attempt to leverage machine learning algorithms to classify exoplanets automatically. In this project, we aim to compare the performance of three machine learning algorithms, namely Neural Networks, Decision Tree, and Support Vector Machine, for the classification of exoplanets using the Kepler Mission dataset.

The classification task is to predict the exoplanet's disposition "**Disposition Using Kepler Data**", which is a multi-class label indicating whether the observed signal is due to 'CANDIDATE', 'FALSE POSITIVE', or 'CONFIRMED'. The dataset was acquired from the NASA Exoplanet Archive database provided by the NASA Exoplanet Science Institute.

The dataset represents a tabular dataset containing features such as various types of periods, durations, and transit depths of the observed signals, which can be used as input to train our machines using machine learning algorithms. The performance of each algorithm will be evaluated using standard evaluation metrics such as accuracy, precision, recall, and F1-score, and the results will be compared to identify the best-performing algorithm for the exoplanet classification task.

After conducting the Exploratory Data Analysis, we will perform data cleaning and pre-processing on the entire dataset. Afterwards, we will use the k-fold cross-validation method to evaluate our selected models and retrieve the best performant model on the pre-processed data. To apply optimal hyperparameters, we will employ GridSearchCV to compare at what hyperparameter settings we can achieve optimal results per each model.

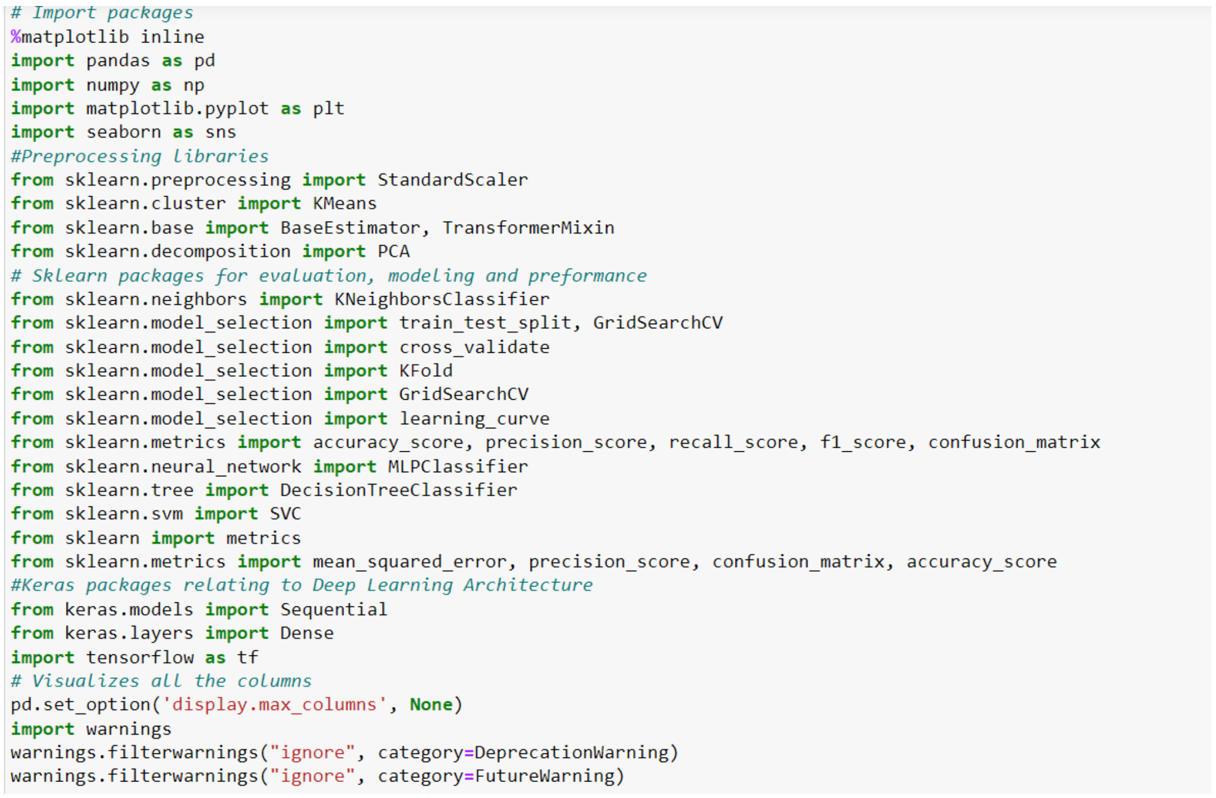
To implement the optimal hyperparameters, we will create the model using a dataset copy prior to the pre-processing phase. At this stage we will split the dataset into 80% training and 20% test. The same pre-processing steps will be implemented on this dataset copy by fitting solely on the training set and transforming both sets. Thus, we can set-up the models hyperparameters as of the GridSearchCV outputs to train the model and test the generalizability on the test set and output the necessary performance metrics.

## 3. Exploratory Data Analysis

### 3.1. Data Understanding

Since data understanding and pre-processing steps are integral parts of the machine learning/data science process, we will be dedicating this section to highlighting the most key information relating to our dataset prior to constructing any classification model for the purpose of the coursework. To do so, we first import the necessary packages for EDA, data pre-processing, data visualization, modelling, evaluation and performance metrics as follows:

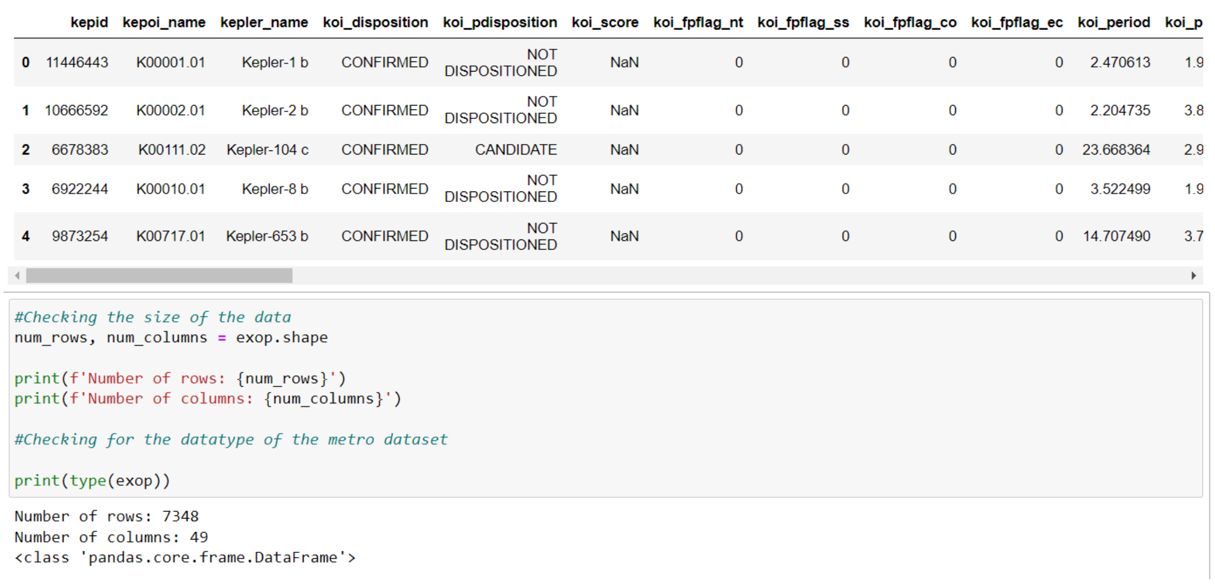
**Figure 01: Packages importing**



```python
# Import packages
%matplotlib inline
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
#Preprocessing libraries
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.decomposition import PCA
# Sklearn packages for evaluation, modeling and preformance
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.model_selection import cross_validate
from sklearn.model_selection import KFold
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import learning_curve
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix
from sklearn.neural_network import MLPClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn import metrics
from sklearn.metrics import mean_squared_error, precision_score, confusion_matrix, accuracy_score
#Keras packages relating to Deep Learning Architecture
from keras.models import Sequential
from keras.layers import Dense
import tensorflow as tf
# Visualizes all the columns
pd.set_option('display.max_columns', None)
import warnings
warnings.filterwarnings("ignore", category=DeprecationWarning)
warnings.filterwarnings("ignore", category=FutureWarning)
```

Once libraries are imported, we load our dataset named ‘keplerm.csv’ by reading it using the ‘***pd.read_csv()***’ function and print the dataset head using the ‘***head()***’ function to print the tabular output representing the first five rows of the dataset and have a glance over how our dataset is appearing and assessing the header names. We also use the ***“. shape”*** attribute to access the size of the dataset for both the columns/features and the rows/instances. As per the out, our dataset encompasses 7348 rows and 49 columns as shown in the figure below:

**Figure 02: dataset shape and table**



| | 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_p |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 11446443 | K00001.01 | Kepler-1 b | CONFIRMED | NOT DISPOSITIONED | NaN | 0 | 0 | 0 | 0 | 2.470613 | 1.9 |
| **1** | 10666592 | K00002.01 | Kepler-2 b | CONFIRMED | NOT DISPOSITIONED | NaN | 0 | 0 | 0 | 0 | 2.204735 | 3.8 |
| **2** | 6678383 | K00111.02 | Kepler-104 c | CONFIRMED | CANDIDATE | NaN | 0 | 0 | 0 | 0 | 23.668364 | 2.9 |
| **3** | 6922244 | K00010.01 | Kepler-8 b | CONFIRMED | NOT DISPOSITIONED | NaN | 0 | 0 | 0 | 0 | 3.522499 | 1.9 |
| **4** | 9873254 | K00717.01 | Kepler-653 b | CONFIRMED | NOT DISPOSITIONED | NaN | 0 | 0 | 0 | 0 | 14.707490 | 3.7 |

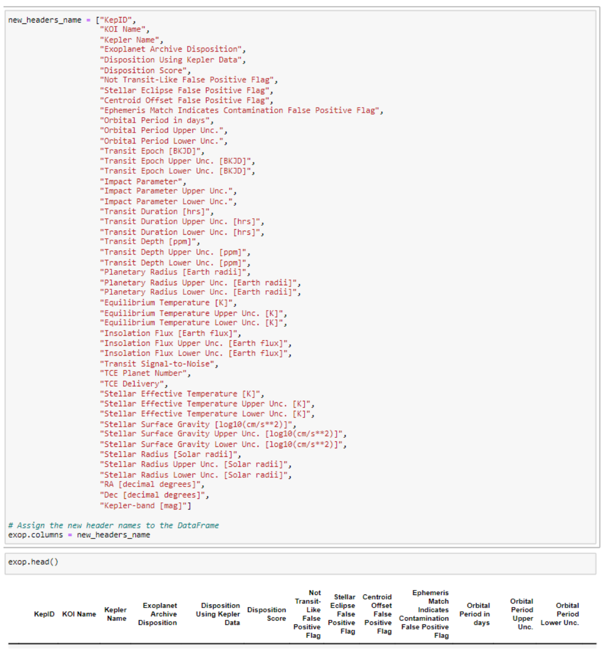
```python
#Checking the size of the data
num_rows, num_columns = exop.shape

print(f'Number of rows: {num_rows}')
print(f'Number of columns: {num_columns}')

#Checking for the datatype of the metro dataset

print(type(exop))
```

```
Number of rows: 7348
Number of columns: 49
<class 'pandas.core.frame.DataFrame'>
```

As demonstrated above, our dataset is characterised by a high dimensionality since it encompasses 49 features as a Pandas DataFrame datatype. Hence, we will be considering several steps for feature selection in the pre-processing phase starting from correlation-based selection to applying Principal Component Analysis (PCA).

We can also notice that the header names are abbreviated and do not reflect the designation in a readable manner per each column. Upon the download of our Kepler data in csv, the file provides the complete labelling of each column. We will then manually copy the appropriate labels and tweak the header names by setting up a list of the new columns then assigning the list to the data frame as follows:

**Figure 03: Columns rename**



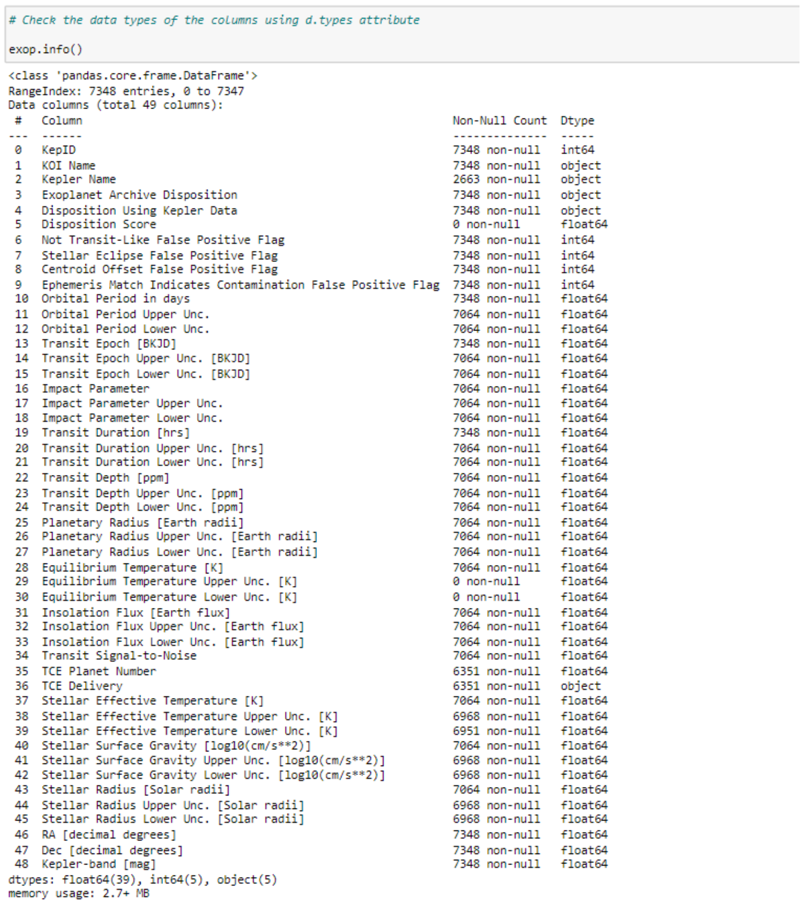
```python
new_headers_name = ["KepID",
                    "KOI Name",
                    "Kepler Name",
                    "Exoplanet Archive Disposition",
                    "Disposition Using Kepler Data",
                    "Disposition Score",
                    "Not Transit-Like False Positive Flag",
                    "Stellar Eclipse False Positive Flag",
                    "Centroid Offset False Positive Flag",
                    "Ephemeris Match Indicates Contamination False Positive Flag",
                    "Orbital Period in days",
                    "Orbital Period Upper Unc.",
                    "Orbital Period Lower Unc.",
                    "Transit Epoch [BKJD]",
                    "Transit Epoch Upper Unc. [BKJD]",
                    "Transit Epoch Lower Unc. [BKJD]",
                    "Impact Parameter",
                    "Impact Parameter Upper Unc.",
                    "Impact Parameter Lower Unc.",
                    "Transit Duration [hrs]",
                    "Transit Duration Upper Unc. [hrs]",
                    "Transit Duration Lower Unc. [hrs]",
                    "Transit Depth [ppm]",
                    "Transit Depth Upper Unc. [ppm]",
                    "Transit Depth Lower Unc. [ppm]",
                    "Planetary Radius [Earth radii]",
                    "Planetary Radius Upper Unc. [Earth radii]",
                    "Planetary Radius Lower Unc. [Earth radii]",
                    "Equilibrium Temperature [K]",
                    "Equilibrium Temperature Upper Unc. [K]",
                    "Equilibrium Temperature Lower Unc. [K]",
                    "Insolation Flux [Earth flux]",
                    "Insolation Flux Upper Unc. [Earth flux]",
                    "Insolation Flux Lower Unc. [Earth flux]",
                    "Transit Signal-to-Noise",
                    "TCE Planet Number",
                    "TCE Delivery",
                    "Stellar Effective Temperature [K]",
                    "Stellar Effective Temperature Upper Unc. [K]",
                    "Stellar Effective Temperature Lower Unc. [K]",
                    "Stellar Surface Gravity [log10(cm/s**2)]",
                    "Stellar Surface Gravity Upper Unc. [log10(cm/s**2)]",
                    "Stellar Surface Gravity Lower Unc. [log10(cm/s**2)]",
                    "Stellar Radius [Solar radii]",
                    "Stellar Radius Upper Unc. [Solar radii]",
                    "Stellar Radius Lower Unc. [Solar radii]",
                    "RA [decimal degrees]",
                    "Dec [decimal degrees]",
                    "Kepler-band [mag]"]

# Assign the new header names to the DataFrame
exop.columns = new_headers_name
```

```python
exop.head()
```

| | KepID | KOI Name | Kepler Name | Exoplanet Archive Disposition | Disposition Using Kepler Data | Disposition Score | Not Transit-Like False Positive Flag | Stellar Eclipse False Positive Flag | Centroid Offset False Positive Flag | Ephemeris Match Indicates Contamination False Positive Flag | Orbital Period in days | Orbital Period Upper Unc. | Orbital Period Lower Unc. |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

We can verify the change by displaying the DataFrame as shown in Figure 03. We also would like to access more information about the dataset such datatypes per each attribute and the non-null count using the the '***info()***' function in which we can extract the following:

**Figure 04: Summary of the Kepler DataFrame**

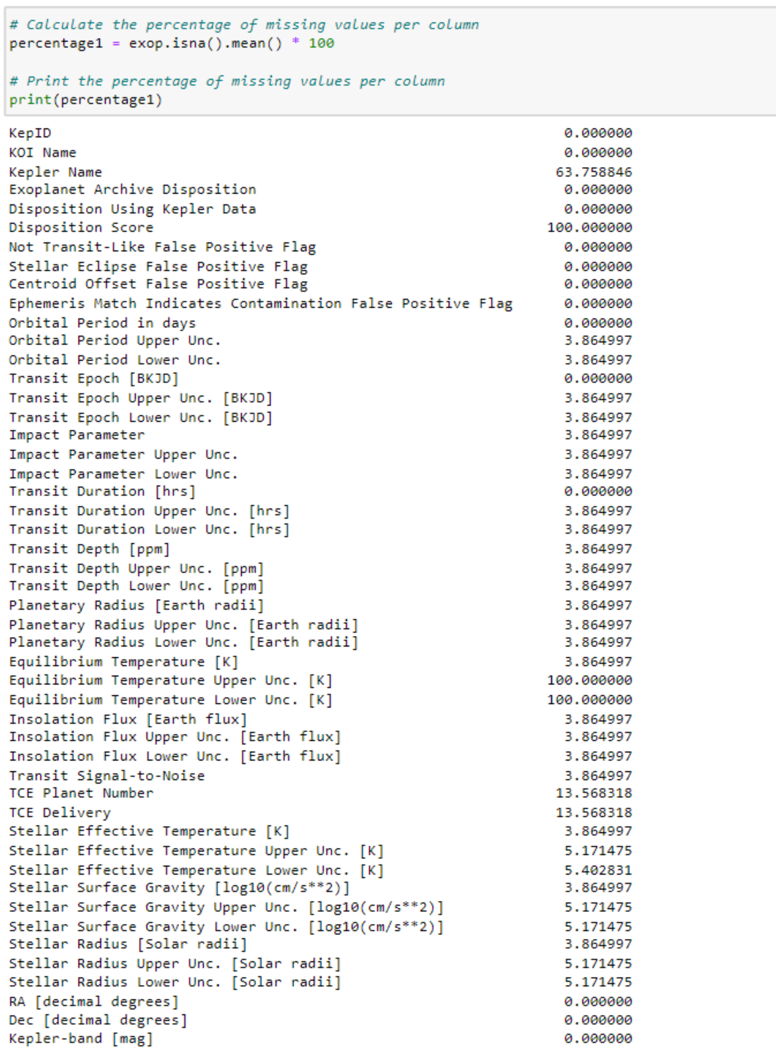


```
# Check the data types of the columns using d.types attribute

exop.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7348 entries, 0 to 7347
Data columns (total 49 columns):
 #   Column                                                Non-Null Count  Dtype  
---  ------                                                --------------  -----  
 0   KepID                                                 7348 non-null   int64  
 1   KOI Name                                              7348 non-null   object 
 2   Kepler Name                                           2663 non-null   object 
 3   Exoplanet Archive Disposition                         7348 non-null   object 
 4   Disposition Using Kepler Data                         7348 non-null   object 
 5   Disposition Score                                     0 non-null      float64
 6   Not Transit-Like False Positive Flag                  7348 non-null   int64  
 7   Stellar Eclipse False Positive Flag                   7348 non-null   int64  
 8   Centroid Offset False Positive Flag                   7348 non-null   int64  
 9   Ephemeris Match Indicates Contamination False Positive Flag  7348 non-null   int64  
 10  Orbital Period in days                                7348 non-null   float64
 11  Orbital Period Upper Unc.                             7064 non-null   float64
 12  Orbital Period Lower Unc.                             7064 non-null   float64
 13  Transit Epoch [BKJD]                                  7348 non-null   float64
 14  Transit Epoch Upper Unc. [BKJD]                       7064 non-null   float64
 15  Transit Epoch Lower Unc. [BKJD]                       7064 non-null   float64
 16  Impact Parameter                                      7064 non-null   float64
 17  Impact Parameter Upper Unc.                           7064 non-null   float64
 18  Impact Parameter Lower Unc.                           7064 non-null   float64
 19  Transit Duration [hrs]                                7348 non-null   float64
 20  Transit Duration Upper Unc. [hrs]                     7064 non-null   float64
 21  Transit Duration Lower Unc. [hrs]                     7064 non-null   float64
 22  Transit Depth [ppm]                                   7064 non-null   float64
 23  Transit Depth Upper Unc. [ppm]                        7064 non-null   float64
 24  Transit Depth Lower Unc. [ppm]                        7064 non-null   float64
 25  Planetary Radius [Earth radii]                        7064 non-null   float64
 26  Planetary Radius Upper Unc. [Earth radii]             7064 non-null   float64
 27  Planetary Radius Lower Unc. [Earth radii]             7064 non-null   float64
 28  Equilibrium Temperature [K]                           7064 non-null   float64
 29  Equilibrium Temperature Upper Unc. [K]                0 non-null      float64
 30  Equilibrium Temperature Lower Unc. [K]                0 non-null      float64
 31  Insolation Flux [Earth flux]                          7064 non-null   float64
 32  Insolation Flux Upper Unc. [Earth flux]               7064 non-null   float64
 33  Insolation Flux Lower Unc. [Earth flux]               7064 non-null   float64
 34  Transit Signal-to-Noise                               7064 non-null   float64
 35  TCE Planet Number                                     6351 non-null   float64
 36  TCE Delivery                                          6351 non-null   object 
 37  Stellar Effective Temperature [K]                     7064 non-null   float64
 38  Stellar Effective Temperature Upper Unc. [K]          6968 non-null   float64
 39  Stellar Effective Temperature Lower Unc. [K]          6951 non-null   float64
 40  Stellar Surface Gravity [log10(cm/s**2)]              7064 non-null   float64
 41  Stellar Surface Gravity Upper Unc. [log10(cm/s**2)]   6968 non-null   float64
 42  Stellar Surface Gravity Lower Unc. [log10(cm/s**2)]   6968 non-null   float64
 43  Stellar Radius [Solar radii]                          7064 non-null   float64
 44  Stellar Radius Upper Unc. [Solar radii]               6968 non-null   float64
 45  Stellar Radius Lower Unc. [Solar radii]               6968 non-null   float64
 46  RA [decimal degrees]                                  7348 non-null   float64
 47  Dec [decimal degrees]                                 7348 non-null   float64
 48  Kepler-band [mag]                                     7348 non-null   float64
dtypes: float64(39), int64(5), object(5)
memory usage: 2.7+ MB
```

As observable from the summary, we can spot we have mixed datatypes ranging from object to integer to float. We also notice that some columns consist entirely of null values and some others contain some missing values as records. We also use another method to print the percentage of missing values using the ***'isna()'*** function to have clear visibility over the approach we will take to handle missing values.

**Figure 05: Percentage of missing values**



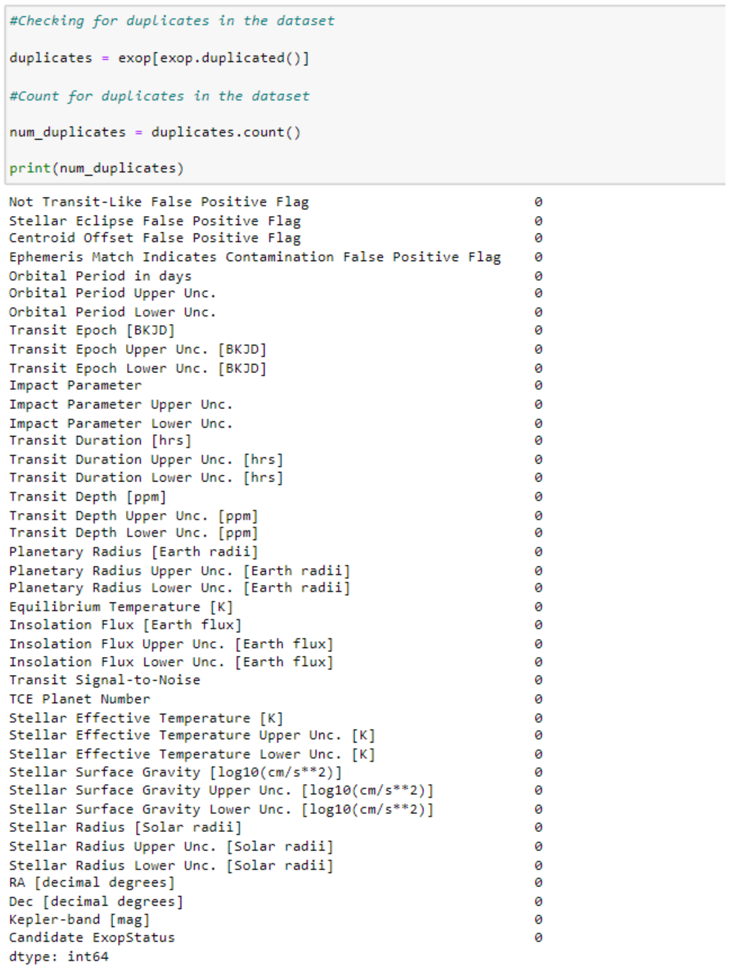
```
# Calculate the percentage of missing values per column
percentage1 = exop.isna().mean() * 100

# Print the percentage of missing values per column
print(percentage1)
```

```
KepID                                                         0.000000
KOI Name                                                      0.000000
Kepler Name                                                  63.758846
Exoplanet Archive Disposition                                 0.000000
Disposition Using Kepler Data                                 0.000000
Disposition Score                                           100.000000
Not Transit-Like False Positive Flag                          0.000000
Stellar Eclipse False Positive Flag                           0.000000
Centroid Offset False Positive Flag                           0.000000
Ephemeris Match Indicates Contamination False Positive Flag   0.000000
Orbital Period in days                                        0.000000
Orbital Period Upper Unc.                                     3.864997
Orbital Period Lower Unc.                                     3.864997
Transit Epoch [BKJD]                                          0.000000
Transit Epoch Upper Unc. [BKJD]                               3.864997
Transit Epoch Lower Unc. [BKJD]                               3.864997
Impact Parameter                                              3.864997
Impact Parameter Upper Unc.                                   3.864997
Impact Parameter Lower Unc.                                   3.864997
Transit Duration [hrs]                                        0.000000
Transit Duration Upper Unc. [hrs]                             3.864997
Transit Duration Lower Unc. [hrs]                             3.864997
Transit Depth [ppm]                                           3.864997
Transit Depth Upper Unc. [ppm]                                3.864997
Transit Depth Lower Unc. [ppm]                                3.864997
Planetary Radius [Earth radii]                                3.864997
Planetary Radius Upper Unc. [Earth radii]                     3.864997
Planetary Radius Lower Unc. [Earth radii]                     3.864997
Equilibrium Temperature [K]                                   3.864997
Equilibrium Temperature Upper Unc. [K]                      100.000000
Equilibrium Temperature Lower Unc. [K]                      100.000000
Insolation Flux [Earth flux]                                  3.864997
Insolation Flux Upper Unc. [Earth flux]                       3.864997
Insolation Flux Lower Unc. [Earth flux]                       3.864997
Transit Signal-to-Noise                                       3.864997
TCE Planet Number                                            13.568318
TCE Delivery                                                 13.568318
Stellar Effective Temperature [K]                             3.864997
Stellar Effective Temperature Upper Unc. [K]                  5.171475
Stellar Effective Temperature Lower Unc. [K]                  5.402831
Stellar Surface Gravity [log10(cm/s**2)]                      3.864997
Stellar Surface Gravity Upper Unc. [log10(cm/s**2)]           5.171475
Stellar Surface Gravity Lower Unc. [log10(cm/s**2)]           5.171475
Stellar Radius [Solar radii]                                  3.864997
Stellar Radius Upper Unc. [Solar radii]                       5.171475
Stellar Radius Lower Unc. [Solar radii]                       5.171475
RA [decimal degrees]                                          0.000000
Dec [decimal degrees]                                         0.000000
Kepler-band [mag]                                             0.000000
```

As pe the above, our dataset contains missing in most of its attributes. We will be handling the missing values in the following Data Cleansing phase. We can also notice that the dataset contains two class labels '*Disposition Using Kepler Data*' and *'Exoplanet Archive Disposition'*. Since we will only be interested in predicting *'Disposition Using Kepler Data'* using the method developed by the Kepler mission team, it will not be necessary to keep 'Exoplanet Archive Disposition' column.

Following this step, we would also be interested in identifying duplicates if any in order to avoid negatively impacting the accuracy of the model in the training process. We employ the ***'duplicated()'*** function as follows:

**Figure 06: Identifying duplicates**


```
#Checking for duplicates in the dataset

duplicates = exop[exop.duplicated()]

#Count for duplicates in the dataset

num_duplicates = duplicates.count()

print(num_duplicates)
```

```
Not Transit-Like False Positive Flag                              0
Stellar Eclipse False Positive Flag                               0
Centroid Offset False Positive Flag                               0
Ephemeris Match Indicates Contamination False Positive Flag       0
Orbital Period in days                                            0
Orbital Period Upper Unc.                                         0
Orbital Period Lower Unc.                                         0
Transit Epoch [BKJD]                                              0
Transit Epoch Upper Unc. [BKJD]                                   0
Transit Epoch Lower Unc. [BKJD]                                   0
Impact Parameter                                                  0
Impact Parameter Upper Unc.                                       0
Impact Parameter Lower Unc.                                       0
Transit Duration [hrs]                                            0
Transit Duration Upper Unc. [hrs]                                 0
Transit Duration Lower Unc. [hrs]                                 0
Transit Depth [ppm]                                               0
Transit Depth Upper Unc. [ppm]                                    0
Transit Depth Lower Unc. [ppm]                                    0
Planetary Radius [Earth radii]                                    0
Planetary Radius Upper Unc. [Earth radii]                         0
Planetary Radius Lower Unc. [Earth radii]                         0
Equilibrium Temperature [K]                                       0
Insolation Flux [Earth flux]                                      0
Insolation Flux Upper Unc. [Earth flux]                           0
Insolation Flux Lower Unc. [Earth flux]                           0
Transit Signal-to-Noise                                           0
TCE Planet Number                                                 0
Stellar Effective Temperature [K]                                 0
Stellar Effective Temperature Upper Unc. [K]                      0
Stellar Effective Temperature Lower Unc. [K]                      0
Stellar Surface Gravity [log10(cm/s**2)]                          0
Stellar Surface Gravity Upper Unc. [log10(cm/s**2)]               0
Stellar Surface Gravity Lower Unc. [log10(cm/s**2)]               0
Stellar Radius [Solar radii]                                      0
Stellar Radius Upper Unc. [Solar radii]                           0
Stellar Radius Lower Unc. [Solar radii]                           0
RA [decimal degrees]                                              0
Dec [decimal degrees]                                             0
Kepler-band [mag]                                                 0
Candidate ExopStatus                                              0
dtype: int64
```

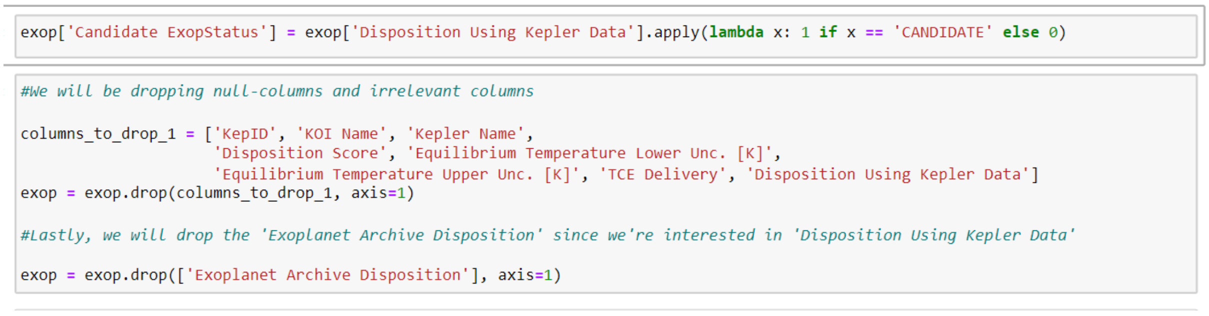
According to Figure 06, our dataset contains no duplicates, hence no further cleansing will be carried out with regards to duplicates.

## 3.2. Data Cleansing

During this phase, we will initially be dropping irrelevant columns that will not provide any useful information for the selected machine learning models such columns the ID number attribute such ***KepID, KOI Name*** as they do not have any correlation with the target variable. We can also establish from Figure 05 that 4 Columns contain null values for more than 60% (i.e., ***Kepler Name, Disposition Score, Equilibrium Temperature Lower Unc. [K], Equilibrium Temperature Upper Unc. [K], TCE Delivery,***). Such information will not have any significance or impact for the model's performance.

We may also drop *'Exoplanet Archive Disposition'* as established by the foregoing. We will be dropping the above Columns using the ***'drop()'***function. Since we are interested in classifying the exoplanets as 'CANDIDATE' or not, we will create a new column entitled '***Candidate ExopStatus***' in the exop dataset for binary class CADIDATE being 1 and any other class being 0 since we're interested in a predictive model for exoplanet candidates. The values in this column are based on the values in another column called 'Disposition Using Kepler Data', therefore, this column will also be dropped.

**Figure 07: Dropping unnecessary columns**

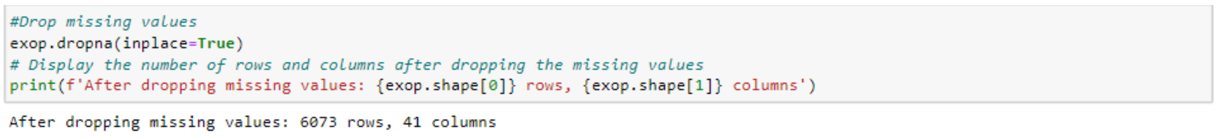


```
exop['Candidate ExopStatus'] = exop['Disposition Using Kepler Data'].apply(lambda x: 1 if x == 'CANDIDATE' else 0)
```

```
#We will be dropping null-columns and irrelevant columns

columns_to_drop_1 = ['KepID', 'KOI Name', 'Kepler Name',
                     'Disposition Score', 'Equilibrium Temperature Lower Unc. [K]',
                     'Equilibrium Temperature Upper Unc. [K]', 'TCE Delivery', 'Disposition Using Kepler Data']
exop = exop.drop(columns_to_drop_1, axis=1)

#Lastly, we will drop the 'Exoplanet Archive Disposition' since we're interested in 'Disposition Using Kepler Data'

exop = exop.drop(['Exoplanet Archive Disposition'], axis=1)
```

As shown in figure 05, for Columns with records of missing values less than 20%, we will be satisfied with dropping the records providing that it will not entail of a substantial loss of information and that the dataset is considerably large as well. For this purpose, we will use the ***'dropna'*** attribute as follows:

**Figure 08: Dropping missing values**



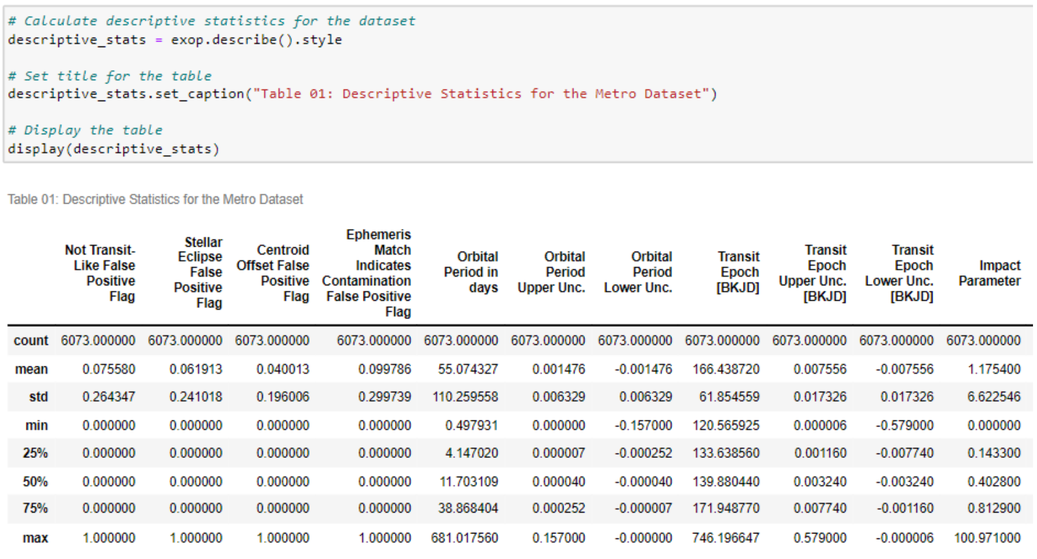
```
#Drop missing values
exop.dropna(inplace=True)
# Display the number of rows and columns after dropping the missing values
print(f'After dropping missing values: {exop.shape[0]} rows, {exop.shape[1]} columns')
```

```
After dropping missing values: 6073 rows, 41 columns
```

After dropping missing values and irrelevant columns, we can notice the reduction of the dataset size to 6073 rows and 41 attributes. For the next step, we will be identifying outliers in the Kepler dataset. This is also crucial as outliers might skew the distribution of data, create bias, increase the unrepresentative variability of the data, and create numerical issues. For this reason, removing outliers can lead to better performance and more accurate predictions.

To do this, we will be observing the distribution of the dataset as well as the statistical description using the ***'describe()'*** function which will output the mean, std, the interquartile range and min and max values per each columns.

**Figure 09: Statistical table**



```
# Calculate descriptive statistics for the dataset
descriptive_stats = exop.describe().style

# Set title for the table
descriptive_stats.set_caption("Table 01: Descriptive Statistics for the Metro Dataset")

# Display the table
display(descriptive_stats)
```

Table 01: Descriptive Statistics for the Metro Dataset

| | Not Transit-Like False Positive Flag | Stellar Eclipse False Positive Flag | Centroid Offset False Positive Flag | Ephemeris Match Indicates Contamination False Positive Flag | Orbital Period in days | Orbital Period Upper Unc. | Orbital Period Lower Unc. | Transit Epoch [BKJD] | Transit Epoch Upper Unc. [BKJD] | Transit Epoch Lower Unc. [BKJD] | Impact Parameter |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **count** | 6073.000000 | 6073.000000 | 6073.000000 | 6073.000000 | 6073.000000 | 6073.000000 | 6073.000000 | 6073.000000 | 6073.000000 | 6073.000000 | 6073.000000 |
| **mean** | 0.075580 | 0.061913 | 0.040013 | 0.099786 | 55.074327 | 0.001476 | -0.001476 | 166.438720 | 0.007556 | -0.007556 | 1.175400 |
| **std** | 0.264347 | 0.241018 | 0.196006 | 0.299739 | 110.259558 | 0.006329 | 0.006329 | 61.854559 | 0.017326 | 0.017326 | 6.622546 |
| **min** | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.497931 | 0.000000 | -0.157000 | 120.565925 | 0.000006 | -0.579000 | 0.000000 |
| **25%** | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 4.147020 | 0.000007 | -0.000252 | 133.638560 | 0.001160 | -0.007740 | 0.143300 |
| **50%** | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 11.703109 | 0.000040 | -0.000040 | 139.880440 | 0.003240 | -0.003240 | 0.402800 |
| **75%** | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 38.868404 | 0.000252 | -0.000007 | 171.948770 | 0.007740 | -0.001160 | 0.812900 |
| **max** | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 681.017560 | 0.157000 | -0.000000 | 746.196647 | 0.579000 | -0.000006 | 100.971000 |

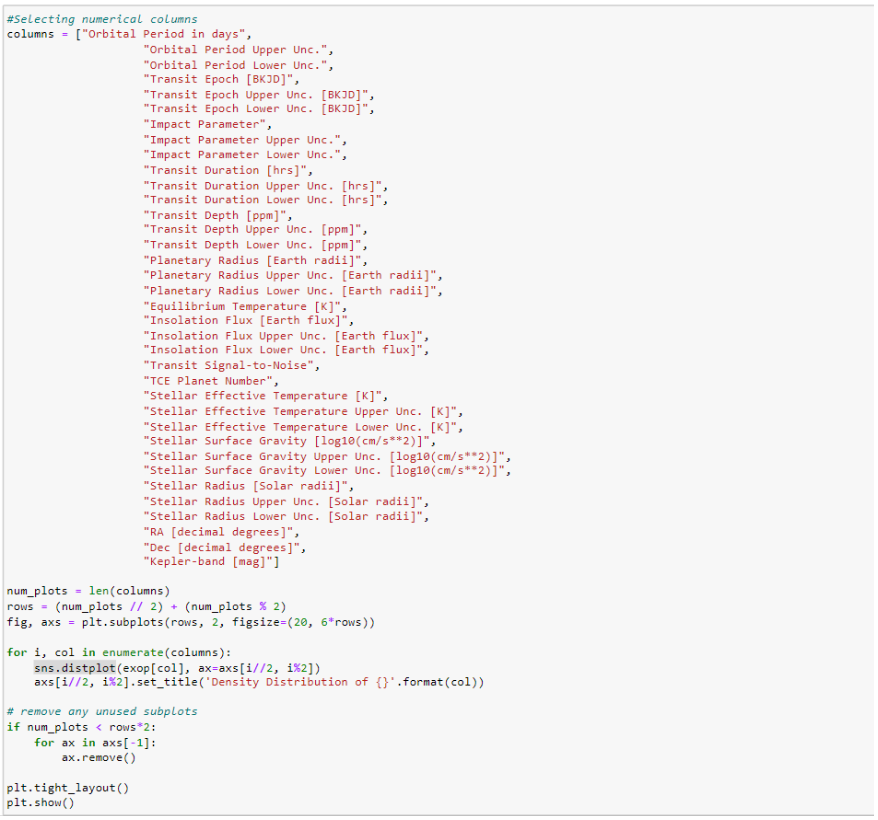
As per the above, we can notice that the mean and standard deviation of each feature vary widely. For example, the mean of the "Not Transit-Like False Positive Flag" column is 0.076, which means that only about 7.6% of the candidates are flagged as not transit-like false positives, while the mean of the "Transit Signal-to-Noise" column is 236, which suggests a much higher level of signal-to-noise ratio.

It is important to note that some of the features have negative values, such as the "Orbital Period Lower Unc." and "Transit Epoch Lower Unc." columns, indicating that the uncertainty range for these features extends below the mean value.

We can also notice that most of the columns would have the mean substantially different from the median, indicating that the distribution is skewed for the majority of the columns. We can also observe that there are some columns where the minimum and maximum values are significantly different from the mean and median values, suggesting the presence of outliers.

We may also plot the histogram for each Column to visualize the skewness of the distribution in the dataset using density distribution using the '***sns.distplot()***' function as follows:

**Figure 10: Density Distribution code snippet**



```python
#Selecting numerical columns
columns = ["Orbital Period in days",
                    "Orbital Period Upper Unc.",
                    "Orbital Period Lower Unc.",
                    "Transit Epoch [BKJD]",
                    "Transit Epoch Upper Unc. [BKJD]",
                    "Transit Epoch Lower Unc. [BKJD]",
                    "Impact Parameter",
                    "Impact Parameter Upper Unc.",
                    "Impact Parameter Lower Unc.",
                    "Transit Duration [hrs]",
                    "Transit Duration Upper Unc. [hrs]",
                    "Transit Duration Lower Unc. [hrs]",
                    "Transit Depth [ppm]",
                    "Transit Depth Upper Unc. [ppm]",
                    "Transit Depth Lower Unc. [ppm]",
                    "Planetary Radius [Earth radii]",
                    "Planetary Radius Upper Unc. [Earth radii]",
                    "Planetary Radius Lower Unc. [Earth radii]",
                    "Equilibrium Temperature [K]",
                    "Insolation Flux [Earth flux]",
                    "Insolation Flux Upper Unc. [Earth flux]",
                    "Insolation Flux Lower Unc. [Earth flux]",
                    "Transit Signal-to-Noise",
                    "TCE Planet Number",
                    "Stellar Effective Temperature [K]",
                    "Stellar Effective Temperature Upper Unc. [K]",
                    "Stellar Effective Temperature Lower Unc. [K]",
                    "Stellar Surface Gravity [log10(cm/s**2)]",
                    "Stellar Surface Gravity Upper Unc. [log10(cm/s**2)]",
                    "Stellar Surface Gravity Lower Unc. [log10(cm/s**2)]",
                    "Stellar Radius [Solar radii]",
                    "Stellar Radius Upper Unc. [Solar radii]",
                    "Stellar Radius Lower Unc. [Solar radii]",
                    "RA [decimal degrees]",
                    "Dec [decimal degrees]",
                    "Kepler-band [mag]"]

num_plots = len(columns)
rows = (num_plots // 2) + (num_plots % 2)
fig, axs = plt.subplots(rows, 2, figsize=(20, 6*rows))

for i, col in enumerate(columns):
    sns.distplot(exop[col], ax=axs[i//2, i%2])
    axs[i//2, i%2].set_title('Density Distribution of {}'.format(col))

# remove any unused subplots
if num_plots < rows*2:
    for ax in axs[-1]:
        ax.remove()

plt.tight_layout()
plt.show()
```

**Figure 11: Four examples of skewed distribution in the dataset**

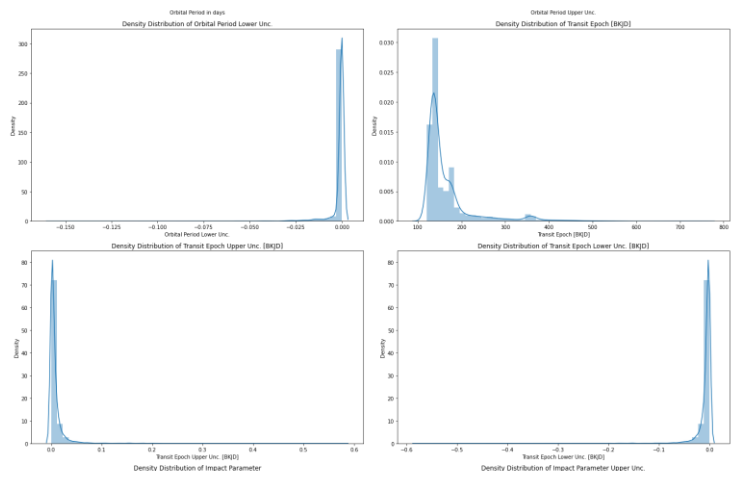
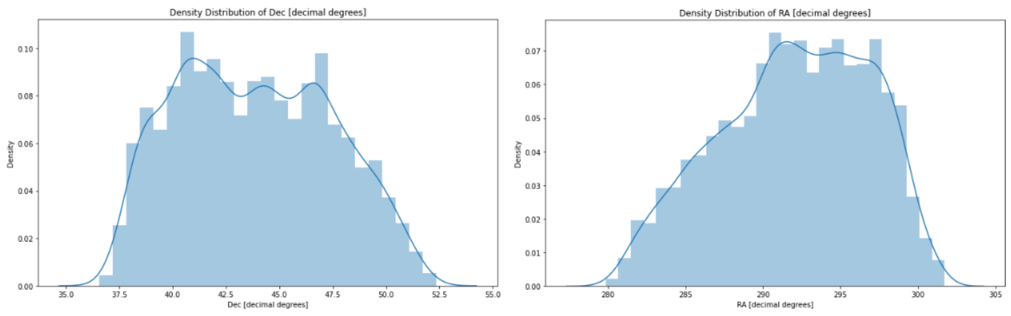
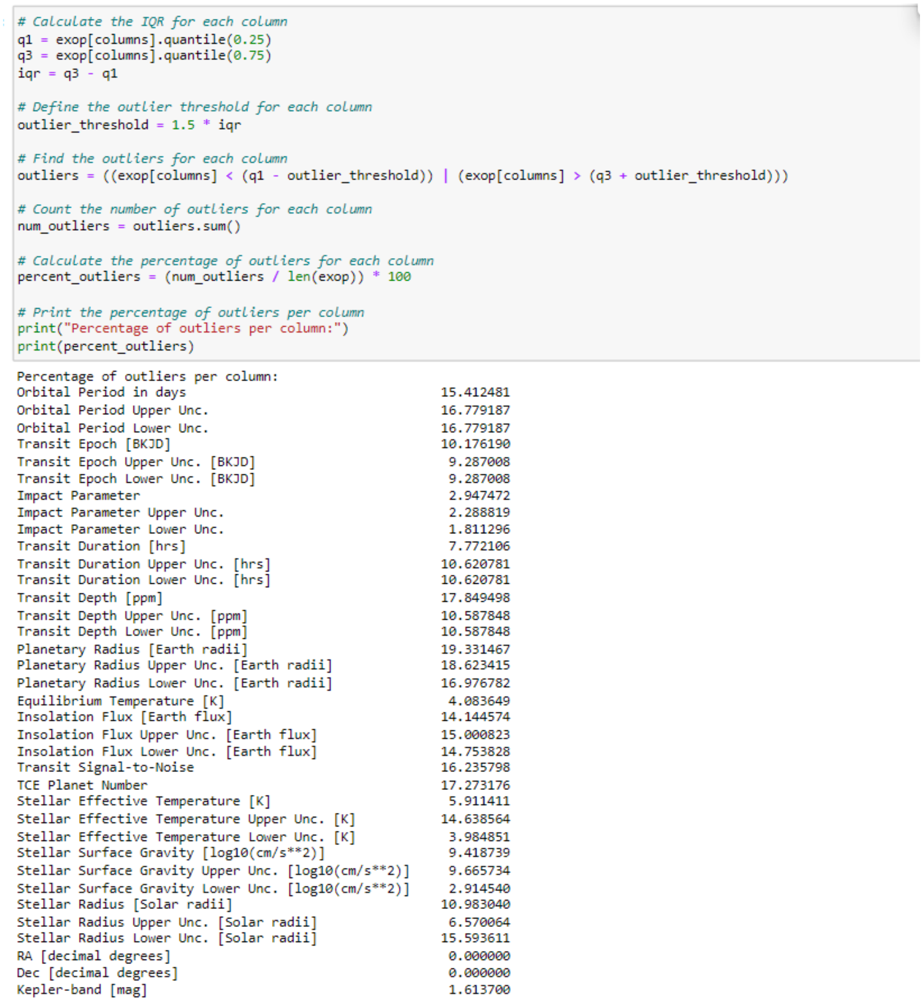


**Figure 12: Two examples of a normal distribution in the dataset**



As demonstrated in Figure 11 and 12, we provided a sample density distribution of the 36 continuous numerical attributes. As confirmed in the statistical description, most of the columns are characterized by a highly skewed distribution which imply the presence of outliers in these columns. To confirm our assumption, we will use the quartile method to identify data points that ranges 1.5 below or above the interquartile range. We are mostly interested in not only identifying outliers but also perceive the ratio of outliers in each columns to make a decision on the handling approach as follows:

**Figure 13: Outliers percentage**

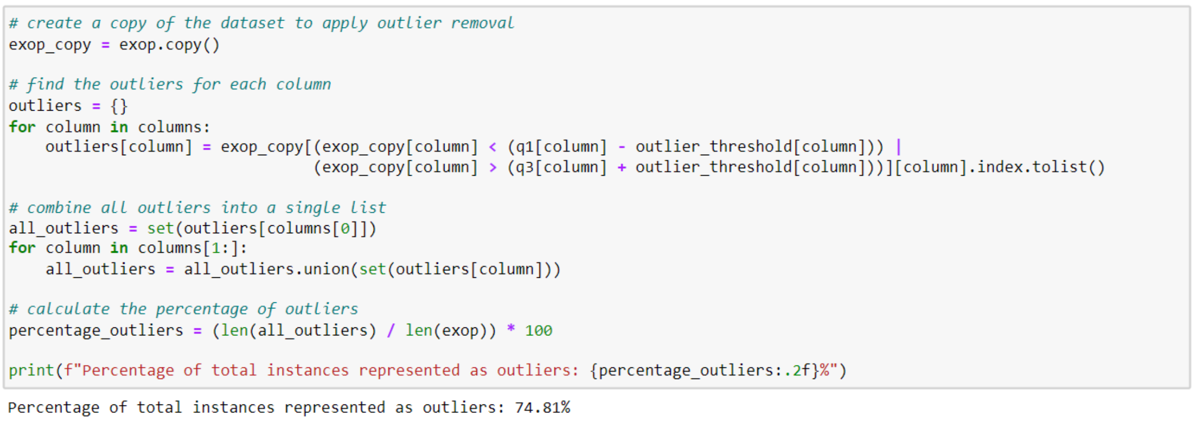


```
# Calculate the IQR for each column
q1 = exop[columns].quantile(0.25)
q3 = exop[columns].quantile(0.75)
iqr = q3 - q1

# Define the outlier threshold for each column
outlier_threshold = 1.5 * iqr

# Find the outliers for each column
outliers = ((exop[columns] < (q1 - outlier_threshold)) | (exop[columns] > (q3 + outlier_threshold)))

# Count the number of outliers for each column
num_outliers = outliers.sum()

# Calculate the percentage of outliers for each column
percent_outliers = (num_outliers / len(exop)) * 100

# Print the percentage of outliers per column
print("Percentage of outliers per column:")
print(percent_outliers)
```

```
Percentage of outliers per column:
Orbital Period in days                                  15.412481
Orbital Period Upper Unc.                               16.779187
Orbital Period Lower Unc.                               16.779187
Transit Epoch [BKJD]                                    10.176190
Transit Epoch Upper Unc. [BKJD]                          9.287008
Transit Epoch Lower Unc. [BKJD]                          9.287008
Impact Parameter                                         2.947472
Impact Parameter Upper Unc.                              2.288819
Impact Parameter Lower Unc.                              1.811296
Transit Duration [hrs]                                   7.772106
Transit Duration Upper Unc. [hrs]                       10.620781
Transit Duration Lower Unc. [hrs]                       10.620781
Transit Depth [ppm]                                     17.849498
Transit Depth Upper Unc. [ppm]                          10.587848
Transit Depth Lower Unc. [ppm]                          10.587848
Planetary Radius [Earth radii]                          19.331467
Planetary Radius Upper Unc. [Earth radii]               18.623415
Planetary Radius Lower Unc. [Earth radii]               16.976782
Equilibrium Temperature [K]                              4.083649
Insolation Flux [Earth flux]                            14.144574
Insolation Flux Upper Unc. [Earth flux]                 15.000823
Insolation Flux Lower Unc. [Earth flux]                 14.753828
Transit Signal-to-Noise                                 16.235798
TCE Planet Number                                       17.273176
Stellar Effective Temperature [K]                        5.911411
Stellar Effective Temperature Upper Unc. [K]            14.638564
Stellar Effective Temperature Lower Unc. [K]             3.984851
Stellar Surface Gravity [log10(cm/s**2)]                 9.418739
Stellar Surface Gravity Upper Unc. [log10(cm/s**2)]      9.665734
Stellar Surface Gravity Lower Unc. [log10(cm/s**2)]      2.914540
Stellar Radius [Solar radii]                            10.983040
Stellar Radius Upper Unc. [Solar radii]                  6.570064
Stellar Radius Lower Unc. [Solar radii]                 15.593611
RA [decimal degrees]                                     0.000000
Dec [decimal degrees]                                    0.000000
Kepler-band [mag]                                        1.613700
```

Bearing in mind that the selected algorithms SVMs, NNs, and DTs are all sensitive to the scale and distribution of the data, and outliers can skew these properties which would result in poor performance of the algorithms or significantly overfit or underfit the model. As displayed in Figure 13, we can outline a large number of outliers in each column. Most of the columns entail a 10-18% whereas only a handful of columns have a percentage of outlier as little as 1-9%. We can test the impact of outlier removal on the loss of information in the dataset by calculating the total number of instances represented as outliers by combining all of the outliers in a list using ***'for loop'*** as follows:

**Figure 14: Outlier instances percentage**

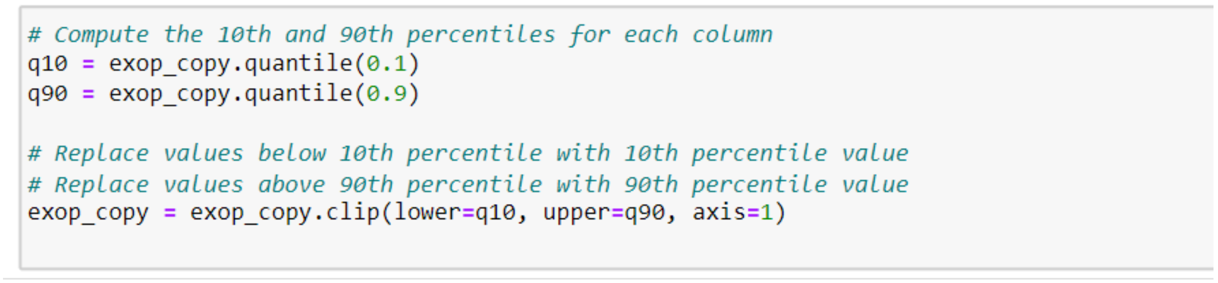


```
# create a copy of the dataset to apply outlier removal
exop_copy = exop.copy()

# find the outliers for each column
outliers = {}
for column in columns:
    outliers[column] = exop_copy[(exop_copy[column] < (q1[column] - outlier_threshold[column])) |
                                 (exop_copy[column] > (q3[column] + outlier_threshold[column]))][column].index.tolist()

# combine all outliers into a single list
all_outliers = set(outliers[columns[0]])
for column in columns[1:]:
    all_outliers = all_outliers.union(set(outliers[column]))

# calculate the percentage of outliers
percentage_outliers = (len(all_outliers) / len(exop)) * 100

print(f"Percentage of total instances represented as outliers: {percentage_outliers:.2f}%")
```

```
Percentage of total instances represented as outliers: 74.81%
```

We have created a new copy of the dataset entitled ***'exop_copy'*** to preform the subsequent steps for redundancy purposes. Interpreting the output, we can asses that the removal of outliers will result in the loss of the majority of data, which consequently will bias the model upon the training. In addition, we should be aware that not all datapoints ranging 1.5 outside the IQR are necessarily outliers or anomalies, which astronomical data as such having a large range, the data points could be representative of explicit information that we do not want to undermine. To circumvent this outcome, we will use the 10th and 90th percentile method which will replace values in the dataset that are below the 10th percentile with the 10th percentile value, and similarly, replace values above the 90th percentile with the 90th percentile value using ***'clip'*** function.

**Figure 15: Replacing outlier values**



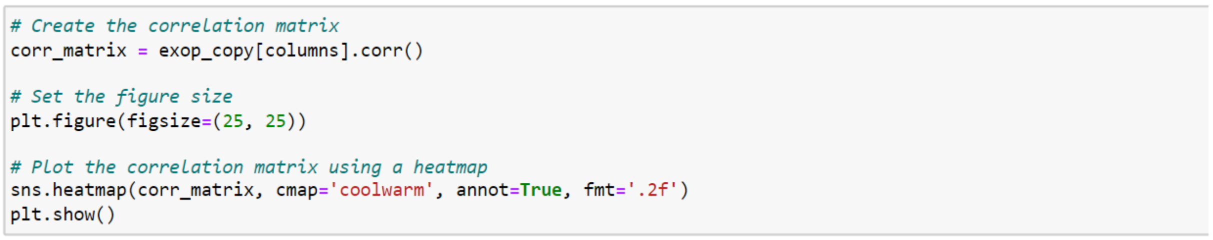
```
# Compute the 10th and 90th percentiles for each column
q10 = exop_copy.quantile(0.1)
q90 = exop_copy.quantile(0.9)

# Replace values below 10th percentile with 10th percentile value
# Replace values above 90th percentile with 90th percentile value
exop_copy = exop_copy.clip(lower=q10, upper=q90, axis=1)
```

## 4. Data pre-processing and feature selection (support with Python code snippets)

In the Data Pre-processing portion, we will be preforming several techniques to attempt attaining optimal results on our selected models. For this reason, we will be utilising feature selection methods, namely correlation-based approached pertaining to the correlation matrix, standardization of the dataset, dimensionality reduction using PCA, and finagling oversampling the minority class using the Synthetic Minority Oversampling Technique (SMOTE).

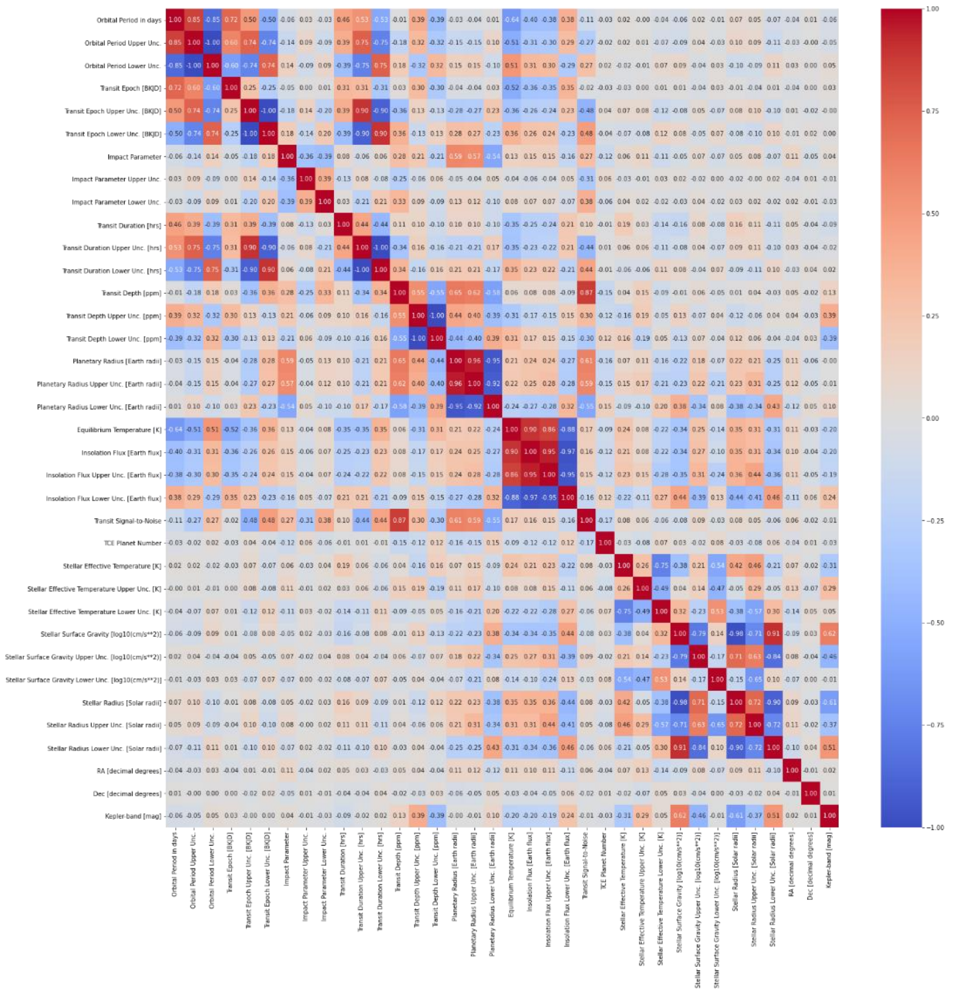
At this stage the features of our dataset are up to 40 excluding the class label. This means that this dataset has a high dimensionality, which renders feature selection a valuable option. We would like to identify the highly correlated features int the dataset excluding the class label. These correlated features represent redundancy in the model as they would convey the same information value and could lead the model to overfitting by focusing on the noise as opposed to the actual patterns in the dataset. To do so, we can visualize the correlation matrix using '***.corr()***'function as illustrated:

Figure 16: Correlation matrix code snippet



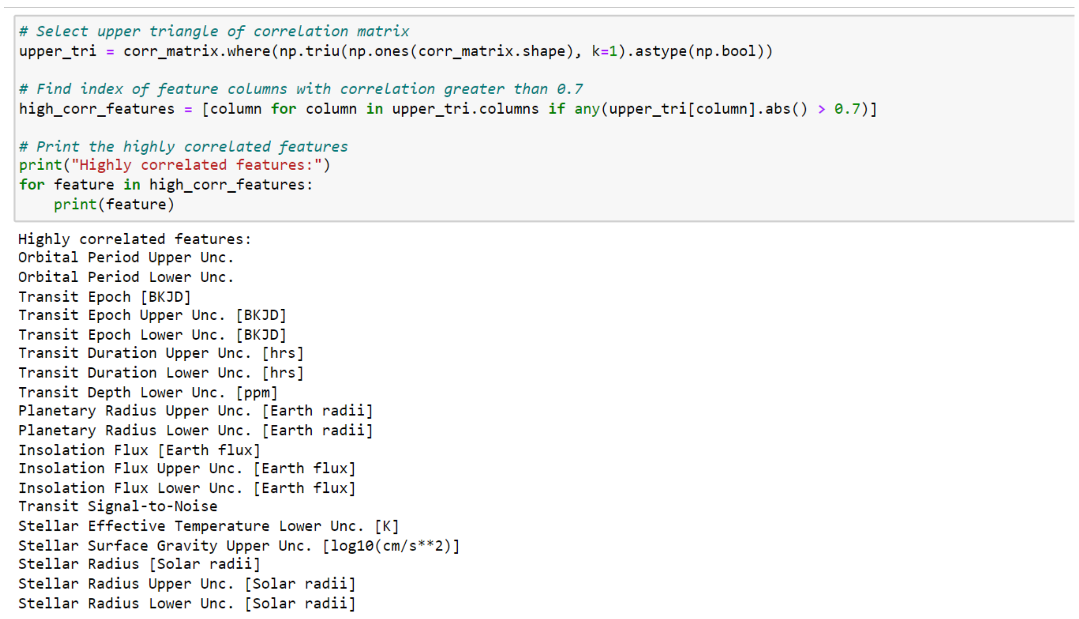
```
# Create the correlation matrix
corr_matrix = exop_copy[columns].corr()

# Set the figure size
plt.figure(figsize=(25, 25))

# Plot the correlation matrix using a heatmap
sns.heatmap(corr_matrix, cmap='coolwarm', annot=True, fmt='.2f')
plt.show()
```

Figure 17: Correlation heatmap



Given that our features too large to spot correlated features visually, we will print the list of highly correlated by finding the index of feature columns with a correlation greater than 0.7 by iterating through the columns of the upper triangle of the correlation matrix as follows:

**Figure 18: Highly correlated features**

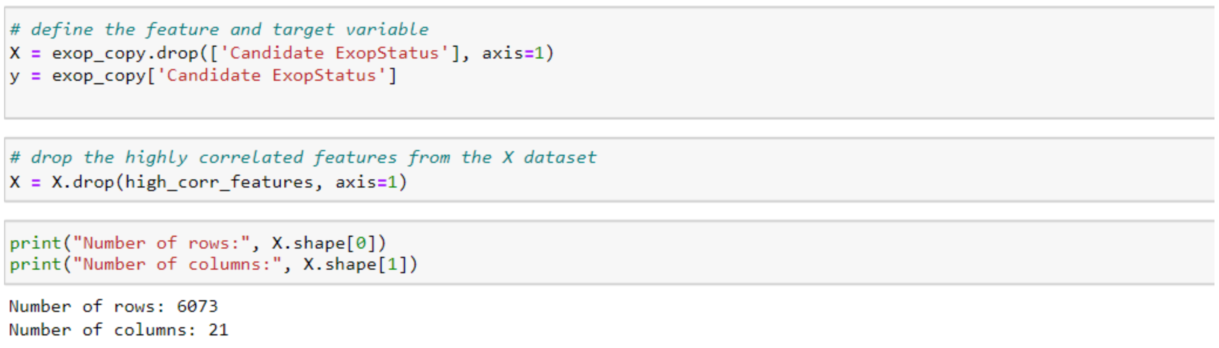


```python
# Select upper triangle of correlation matrix
upper_tri = corr_matrix.where(np.triu(np.ones(corr_matrix.shape), k=1).astype(np.bool))

# Find index of feature columns with correlation greater than 0.7
high_corr_features = [column for column in upper_tri.columns if any(upper_tri[column].abs() > 0.7)]

# Print the highly correlated features
print("Highly correlated features:")
for feature in high_corr_features:
    print(feature)
```

```
Highly correlated features:
Orbital Period Upper Unc.
Orbital Period Lower Unc.
Transit Epoch [BKJD]
Transit Epoch Upper Unc. [BKJD]
Transit Epoch Lower Unc. [BKJD]
Transit Duration Upper Unc. [hrs]
Transit Duration Lower Unc. [hrs]
Transit Depth Lower Unc. [ppm]
Planetary Radius Upper Unc. [Earth radii]
Planetary Radius Lower Unc. [Earth radii]
Insolation Flux [Earth flux]
Insolation Flux Upper Unc. [Earth flux]
Insolation Flux Lower Unc. [Earth flux]
Transit Signal-to-Noise
Stellar Effective Temperature Lower Unc. [K]
Stellar Surface Gravity Upper Unc. [log10(cm/s**2)]
Stellar Radius [Solar radii]
Stellar Radius Upper Unc. [Solar radii]
Stellar Radius Lower Unc. [Solar radii]
```

Before proceeding with feature selection and transformation on our dataset, we first define the features, represented by ***'X'*** and target variable represented by **'y'**. This will be for the purpose of preparing the dataset for the K-fold cross validation to evaluate the selected models. Afterwards, we will proceed with dropping the highly correlated columns outputted in Figure 18 above as of the below code. This will reduce the dimensionality of our features from 40 to

**Figure 19: Defining X, y and dropping highly correlated features**



```python
# define the feature and target variable
X = exop_copy.drop(['Candidate ExopStatus'], axis=1)
y = exop_copy['Candidate ExopStatus']
```

```python
# drop the highly correlated features from the X dataset
X = X.drop(high_corr_features, axis=1)
```

```python
print("Number of rows:", X.shape[0])
print("Number of columns:", X.shape[1])
```

```
Number of rows: 6073
Number of columns: 21
```

The next pre-processing step would be to scale the dataset. This is a crucial segment of the pre-processing phase given that the Kepler dataset given its variety of astronomical measurements of celestial objects it includes measurements of various physical quantities, such as orbital periods, star temperature and planet radius, that are measured in different units. Hence, scaling these measurements to a similar range will ensure that each measurement contributes equally to the selected classification models. We implement the scaling using the ***'StandardScaler'*** class from the Scikit-learn library on another version of the ***'X'*** entitled ***'X_pr'*** in order to preserve the ***'X'*** only for model implementation, whereas ***'X_pr'*** will be dedication for model evaluation using k-fold cross validation. This approach is envisaged to apply good practices with regards to information leakage and avoid any potential of overfitting the model.

Granted that all the features in the dataset are in numerical values, we will not apply any custom-based transformation and ***'fit_transform'*** the scaler to the entire '***X_pr***' dataset as follows:

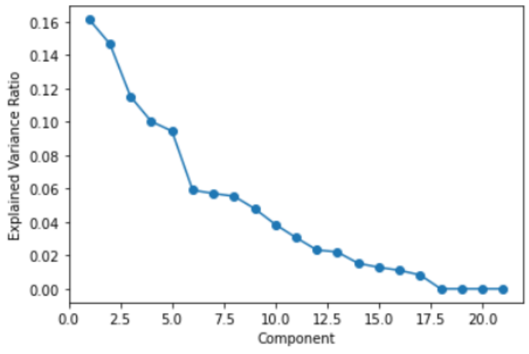
**Figure 20: Apply standardization using StandardScaler**

```
#Scaling the dataset

# make a copy of X and y
X_pr = X.copy()
y_pr = y.copy()

scaler = StandardScaler()
X_pr = scaler.fit_transform(X_pr)
```

Once our data is scaled to set in the same range, we will also be using PCA in order to reduce further the dimensionality bearing 21 features and hence avoid the complexity of high-dimensional. PCA will allow us to reduce the number of features in our dataset while retaining as much of the original information as possible and facilitate the model to learn as well as avoid overfitting. To determine the number of the PCA component, we will plot the variance ratio for each component to determine the number of principal components to retain in the transformed data by fitting the '***PCA()'*** into '***X_pr'*** as follows:

**Figure 21: PCA variance ratio per component**

```
# create a PCA object with n_components=None
pca = PCA(n_components=None)

# fit the PCA model on the data
pca.fit(X_pr)

# plot the explained variance ratio for each component
plt.plot(range(1, len(pca.explained_variance_ratio_)+1), pca.explained_variance_ratio_, marker='o')
plt.xlabel('Component')
plt.ylabel('Explained Variance Ratio')
plt.show()
```



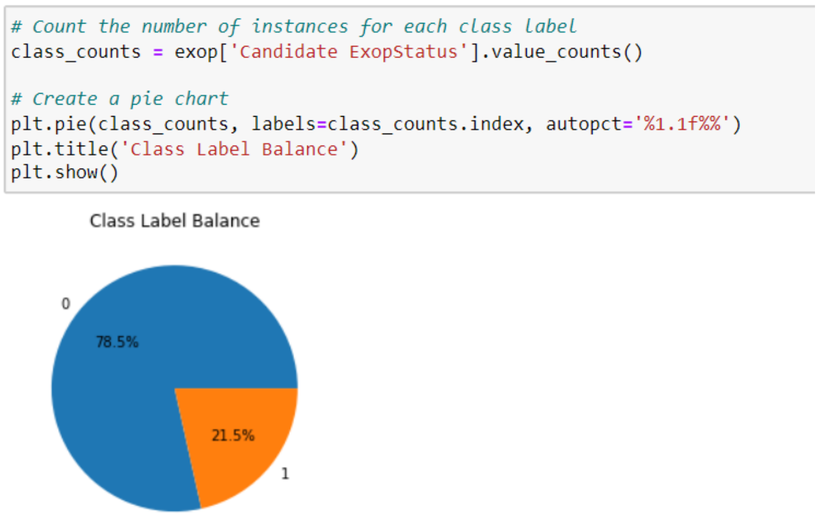
As per the graph, it seems that the component number 5 can retain up to 10% of variance in the dataset whereas at component number 2 it can reach up to 16% of the variance. Choosing to retain only 2 principal components from 21 to 2 may result in loss of information and lead to suboptimal model performance. Hence, 5 principal components is a more moderate reduction to not undermine the model performance. We apply the PCA as follows:

**Figure 22: Applying PCA using 5 components**

```
# instantiate PCA
pca = PCA(n_components=5)

# fit and transform PCA on X_pr
X_pr = pca.fit_transform(X_pr)
```

Considering using SMOTE would imply that the class label is imbalanced. Initially, we attempt to visualize the class label count in a pie chart to make an informed decision using the ***‘value_count()’*** function and ***‘plt’***:

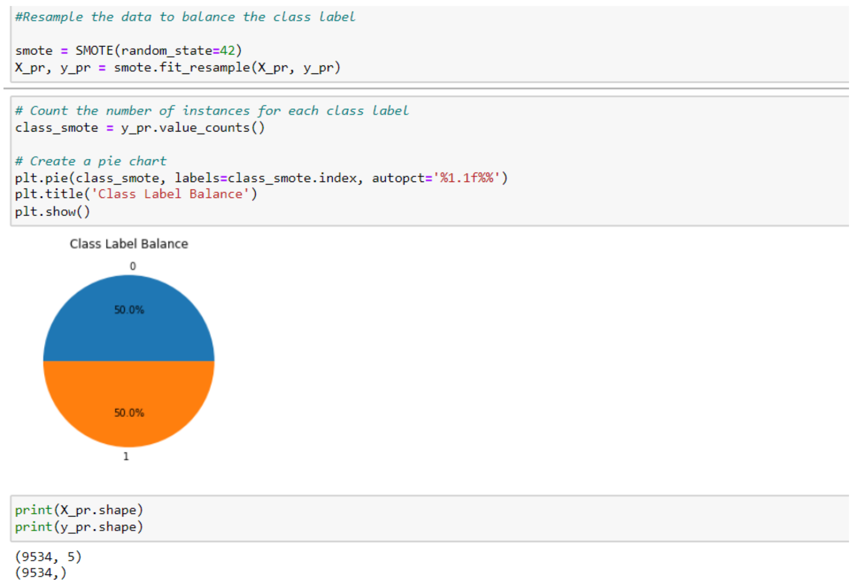
**Figure 23: Class label balance**



```
# Count the number of instances for each class label
class_counts = exop['Candidate ExopStatus'].value_counts()

# Create a pie chart
plt.pie(class_counts, labels=class_counts.index, autopct='%1.1f%%')
plt.title('Class Label Balance')
plt.show()
```

As displayed in the pie chart, the candidate class has merely 21.5% whereas the non-candidate class is much more significant for over 78.5%. This will necessitate balancing the target label to optimize the performance of the model using the ***‘SMOTE’*** function. We also recheck the balance and the shape of the dataset:

**Figure 24: oversampling the minority class**



```
#Resample the data to balance the class label

smote = SMOTE(random_state=42)
X_pr, y_pr = smote.fit_resample(X_pr, y_pr)
```

```
# Count the number of instances for each class label
class_smote = y_pr.value_counts()

# Create a pie chart
plt.pie(class_smote, labels=class_smote.index, autopct='%1.1f%%')
plt.title('Class Label Balance')
plt.show()
```

```
print(X_pr.shape)
print(y_pr.shape)
```

```
(9534, 5)
(9534,)
```

## 5. Machine learning model N (iterate for each of the three models)

### 5.1. Summary of the approach

Our selected ML algorithm consist of running and fine tunning Support Vector Machine (SVM), Decision Tree, and Neural Networks. We will be using k-fold cross validation in order to evaluate and compare the results of each ML algorithm by splitting the data into 10 subsets and using K-1 subsets for training the model and the remaining subset for testing the model through each iteration and averaging the results to acquire the final results. Afterwards, we use GridSearch for hyperparameter tuning that involves searching through a predefined hyperparameter space and evaluating the model's performance for each combination of hyperparameters to output the hyperparameters with the best performance to select the optimal hyperparameters. In the final stage, we will be building the model using the best hyperparameters per each model by splitting the dataset into 80% training and 20% test to verify the generalizability of the models.

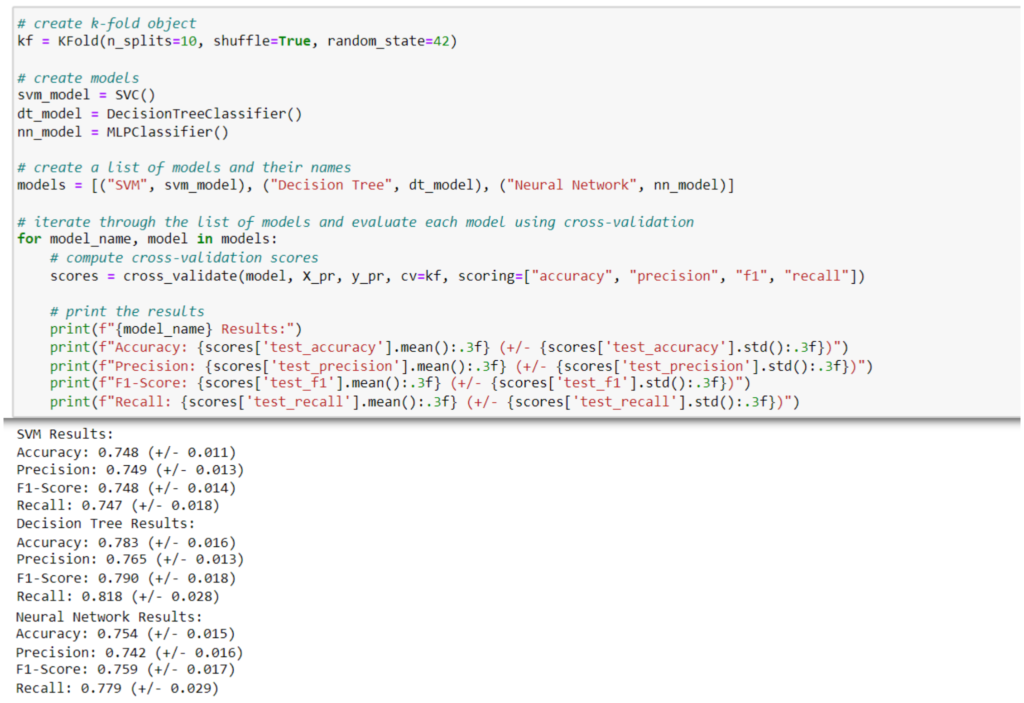
Our selection of the algorithms is based on the popularity of such algorithms in the exoplanet research and for similar classification tasks more generally. Ameya *et al* [4] trained the model using the same Kepler dataset by employing GridSearch on each classifier with different values for its hyperparameters to choose the optimum set of values that provide maximum performance using SVM, Random Forest, and Neural Networks. Similarly, Rutuja *et al* [5] used various supervised learning algorithms to predict the habitability of recently observed exoplanets using the Kepler dataset such as CART, SVM, FNN, Random Forest, Logistic Regression and Naive Bayes. The common approach for Vendant *et al* [2] was to use Decision Tree as part of the selected algorithms on the same Kepler dataset.

Given that the Kepler dataset is complex and has a high number of features, selecting these three algorithms is a reasonable choice as they are capable of handling complex data. Decisions Trees is feasibly interpretable and provides insights into the decision-making process, whereas NN is particularly effective when there are large amounts of data and complexity despite its interpretability limitation. SVM also has shown to perform well on classification tasks with non-linear decision boundaries.

### 5.2. Model training and evaluation

We start off our model training and evaluation by employ the k-fold cross validation on our selected models to compare performance using the sklearn library. The KFold object is created with a specified number of splits (10), and the shuffle and random state parameters are set for reproducibility as demonstrated below:

**Figure 25: K-fold cross-validation**



```
# create k-fold object
kf = KFold(n_splits=10, shuffle=True, random_state=42)

# create models
svm_model = SVC()
dt_model = DecisionTreeClassifier()
nn_model = MLPClassifier()

# create a list of models and their names
models = [("SVM", svm_model), ("Decision Tree", dt_model), ("Neural Network", nn_model)]

# iterate through the list of models and evaluate each model using cross-validation
for model_name, model in models:
    # compute cross-validation scores
    scores = cross_validate(model, X_pr, y_pr, cv=kf, scoring=["accuracy", "precision", "f1", "recall"])

    # print the results
    print(f"{model_name} Results:")
    print(f"Accuracy: {scores['test_accuracy'].mean():.3f} (+/- {scores['test_accuracy'].std():.3f})")
    print(f"Precision: {scores['test_precision'].mean():.3f} (+/- {scores['test_precision'].std():.3f})")
    print(f"F1-Score: {scores['test_f1'].mean():.3f} (+/- {scores['test_f1'].std():.3f})")
    print(f"Recall: {scores['test_recall'].mean():.3f} (+/- {scores['test_recall'].std():.3f})")
```

```
SVM Results:
Accuracy: 0.748 (+/- 0.011)
Precision: 0.749 (+/- 0.013)
F1-Score: 0.748 (+/- 0.014)
Recall: 0.747 (+/- 0.018)
Decision Tree Results:
Accuracy: 0.783 (+/- 0.016)
Precision: 0.765 (+/- 0.013)
F1-Score: 0.790 (+/- 0.018)
Recall: 0.818 (+/- 0.028)
Neural Network Results:
Accuracy: 0.754 (+/- 0.015)
Precision: 0.742 (+/- 0.016)
F1-Score: 0.759 (+/- 0.017)
Recall: 0.779 (+/- 0.029)
```

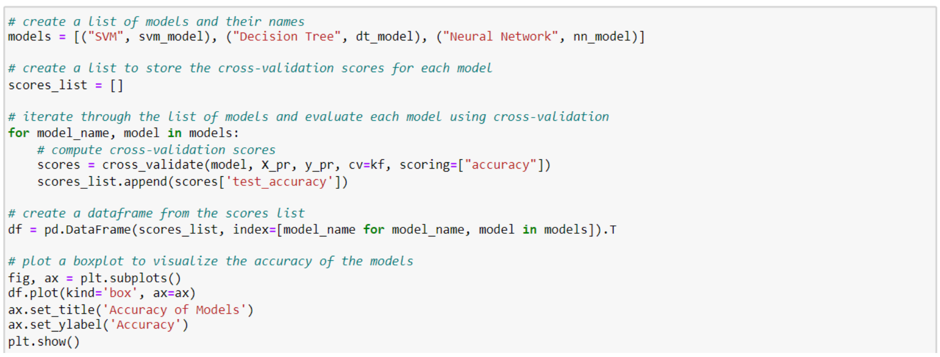
As per the results, we can establish that the decision tree model has the highest accuracy, precision, F1-score, and recall among the three models. This suggests that the decision tree model is the best performer for this particular k-fold cross-validation. the SVM and neural network models have relatively similar performance, with the neural network model having slightly higher precision and F1-score, but slightly lower accuracy and recall than the SVM model as summarized in Table 01:

**Table 01: Comparison of Classification Performance for Exoplanet Detection**

| **Model** | **Accuracy** | **Precision** | **F1-Score** | **Recall** |
|---|---|---|---|---|
| SVM | 0.748 (+/- 0.011) | 0.749 (+/- 0.013) | 0.748 (+/- 0.014) | 0.747 (+/- 0.018) |
| Decision Tree | 0.783 (+/- 0.016) | 0.765 (+/- 0.013) | 0.790 (+/- 0.018) | 0.818 (+/- 0.028) |
| Neural Network | 0.754 (+/- 0.015) | 0.742 (+/- 0.016) | 0.759 (+/- 0.017) | 0.779 (+/- 0.029) |

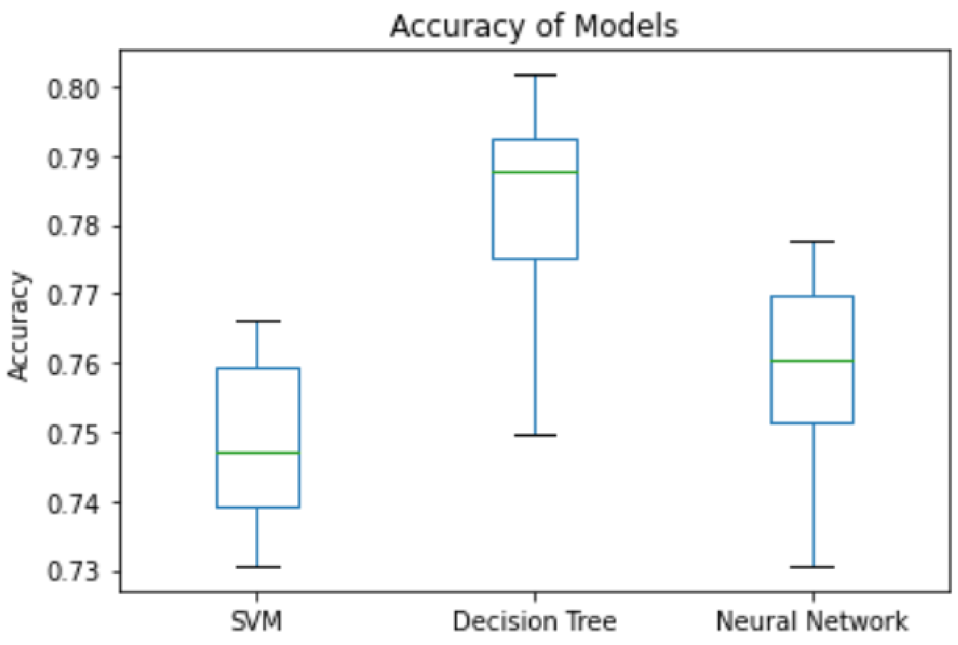
We may also visualize the average accuracy of each model for comparison using the boxplot:

**Figure 26: Boxplot code snippet**



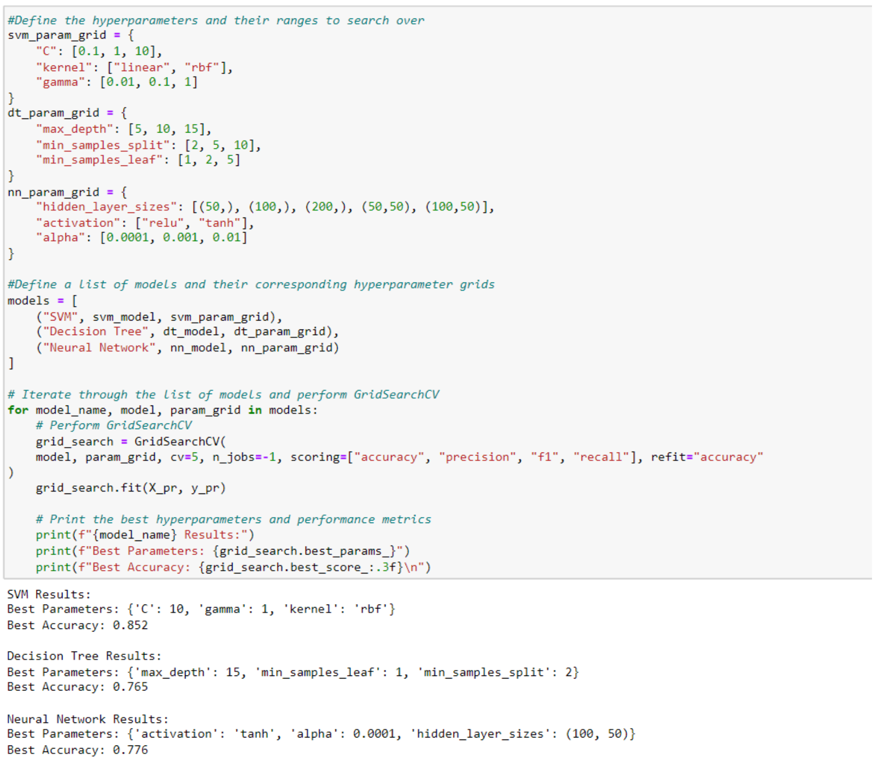
```
# create a list of models and their names
models = [("SVM", svm_model), ("Decision Tree", dt_model), ("Neural Network", nn_model)]

# create a list to store the cross-validation scores for each model
scores_list = []

# iterate through the list of models and evaluate each model using cross-validation
for model_name, model in models:
    # compute cross-validation scores
    scores = cross_validate(model, X_pr, y_pr, cv=kf, scoring=["accuracy"])
    scores_list.append(scores['test_accuracy'])

# create a dataframe from the scores list
df = pd.DataFrame(scores_list, index=[model_name for model_name, model in models]).T

# plot a boxplot to visualize the accuracy of the models
fig, ax = plt.subplots()
df.plot(kind='box', ax=ax)
ax.set_title('Accuracy of Models')
ax.set_ylabel('Accuracy')
plt.show()
```

**Figure 27: Boxplot visualization**



Following the K-fold cross-validation, we will be running GridSearchCV in order to output the optimal hyperparameters for each model. We use different parameter grids to perform a search through iterations for the best hyperparameters of our selected models. For each model, the code prints out the best hyperparameters found during the search, as well as the best accuracy score obtained with those hyperparameters:

**Figure 28: Using GridSearchCV for hyperparameter optimization**



```
#Define the hyperparameters and their ranges to search over
svm_param_grid = {
    "C": [0.1, 1, 10],
    "kernel": ["linear", "rbf"],
    "gamma": [0.01, 0.1, 1]
}
dt_param_grid = {
    "max_depth": [5, 10, 15],
    "min_samples_split": [2, 5, 10],
    "min_samples_leaf": [1, 2, 5]
}
nn_param_grid = {
    "hidden_layer_sizes": [(50,), (100,), (200,), (50,50), (100,50)],
    "activation": ["relu", "tanh"],
    "alpha": [0.0001, 0.001, 0.01]
}

#Define a list of models and their corresponding hyperparameter grids
models = [
    ("SVM", svm_model, svm_param_grid),
    ("Decision Tree", dt_model, dt_param_grid),
    ("Neural Network", nn_model, nn_param_grid)
]

# Iterate through the list of models and perform GridSearchCV
for model_name, model, param_grid in models:
    # Perform GridSearchCV
    grid_search = GridSearchCV(
    model, param_grid, cv=5, n_jobs=-1, scoring=["accuracy", "precision", "f1", "recall"], refit="accuracy"
)
    grid_search.fit(X_pr, y_pr)

    # Print the best hyperparameters and performance metrics
    print(f"{model_name} Results:")
    print(f"Best Parameters: {grid_search.best_params_}")
    print(f"Best Accuracy: {grid_search.best_score_:.3f}\n")
```

```
SVM Results:
Best Parameters: {'C': 10, 'gamma': 1, 'kernel': 'rbf'}
Best Accuracy: 0.852

Decision Tree Results:
Best Parameters: {'max_depth': 15, 'min_samples_leaf': 1, 'min_samples_split': 2}
Best Accuracy: 0.765

Neural Network Results:
Best Parameters: {'activation': 'tanh', 'alpha': 0.0001, 'hidden_layer_sizes': (100, 50)}
Best Accuracy: 0.776
```

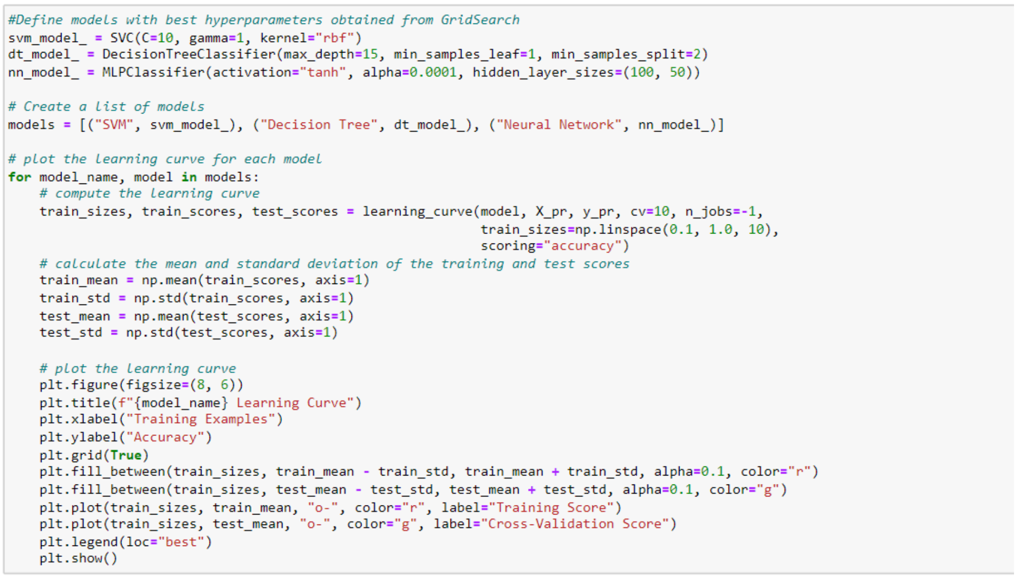
**Table 02: Optimized hyperparameters per each model**

| Model | **Optimized hyperparameters** | **Best Accuracy** |
|---|---|---|
| SVM | {'C': 10, 'gamma': 1, 'kernel': 'rbf'} | 0.852 |
| Decision Tree | {'max_depth': 15, 'min_samples_leaf': 1, 'min_samples_split': 2} | 0.765 |
| Neural Network | {'activation': 'tanh', 'alpha': 0.0001, 'hidden_layer_sizes': (100, 50)} | 0.776 |

The results show that SVM has the highest average accuracy score of 85.2%, for the best hyperparameters C=10, gamma=1, and kernel='rbf'. The Decision Tree comes second with a lower accuracy score of 76.5%, with the best hyperparameters being max_depth=15, min_samples_leaf=1, and min_samples_split=2. The Neural Network model had an accuracy score of 77.6%, with the best hyperparameters being activation='tanh', alpha=0.0001, and hidden_layer_sizes=(100, 50).^

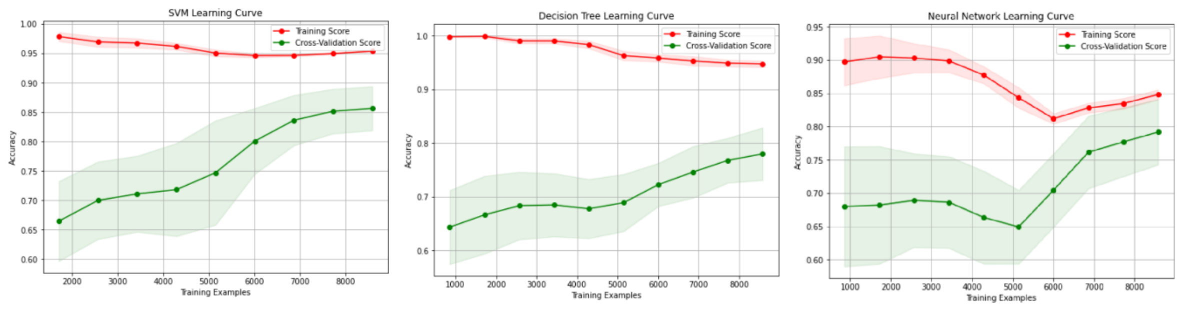
To investigate further on the learning performance, we can employ the learning curve to check how each model is preforming when the training samples increase and asses any overfitting per each model. We generate learning curves for the three models with their best hyperparameters obtained from GridSearch by plotting the model's performance on the training set and the cross-validation set as a function of the number of training examples using the following code:

**Figure 29: Learning curve code snippet**



```python
#Define models with best hyperparameters obtained from GridSearch
svm_model_ = SVC(C=10, gamma=1, kernel="rbf")
dt_model_ = DecisionTreeClassifier(max_depth=15, min_samples_leaf=1, min_samples_split=2)
nn_model_ = MLPClassifier(activation="tanh", alpha=0.0001, hidden_layer_sizes=(100, 50))

# Create a list of models
models = [("SVM", svm_model_), ("Decision Tree", dt_model_), ("Neural Network", nn_model_)]

# plot the learning curve for each model
for model_name, model in models:
    # compute the learning curve
    train_sizes, train_scores, test_scores = learning_curve(model, X_pr, y_pr, cv=10, n_jobs=-1,
                                                            train_sizes=np.linspace(0.1, 1.0, 10),
                                                            scoring="accuracy")
    # calculate the mean and standard deviation of the training and test scores
    train_mean = np.mean(train_scores, axis=1)
    train_std = np.std(train_scores, axis=1)
    test_mean = np.mean(test_scores, axis=1)
    test_std = np.std(test_scores, axis=1)

    # plot the learning curve
    plt.figure(figsize=(8, 6))
    plt.title(f"{model_name} Learning Curve")
    plt.xlabel("Training Examples")
    plt.ylabel("Accuracy")
    plt.grid(True)
    plt.fill_between(train_sizes, train_mean - train_std, train_mean + train_std, alpha=0.1, color="r")
    plt.fill_between(train_sizes, test_mean - test_std, test_mean + test_std, alpha=0.1, color="g")
    plt.plot(train_sizes, train_mean, "o-", color="r", label="Training Score")
    plt.plot(train_sizes, test_mean, "o-", color="g", label="Cross-Validation Score")
    plt.legend(loc="best")
    plt.show()
```

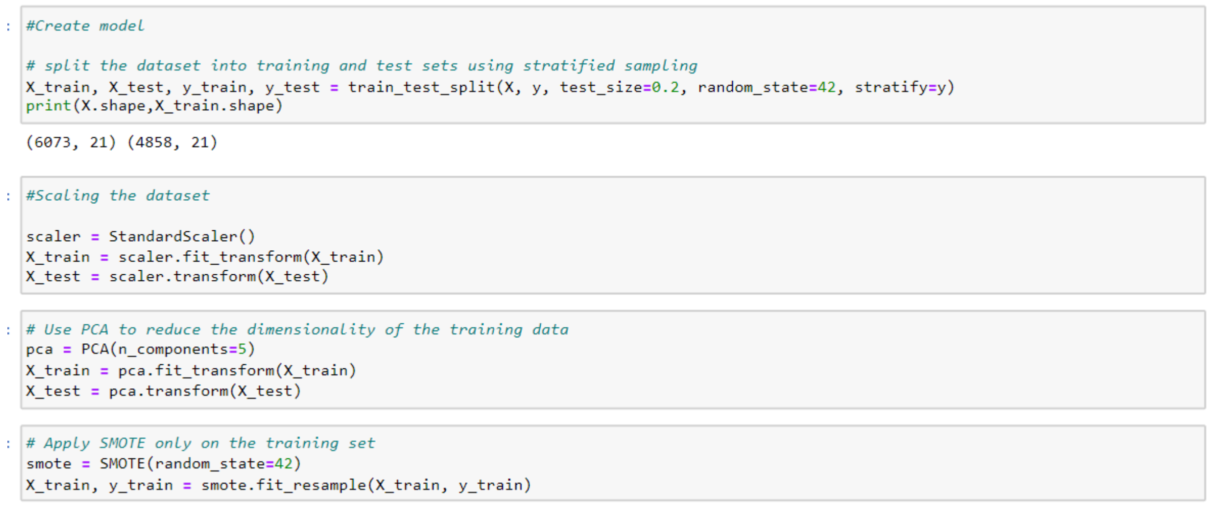
**Figure 30: Learning curve plot**



The learning curves generated by the code above show that all three models are converging to a high accuracy value as the number of training examples increase. Hence, the models have learned the important patterns in the data and are not overfitting. However, we can notice a small gap between the training score and cross-validation score for SVM and Decision Tree. This implies that both models can improve accuracy performance with more training samples / records.

After evaluating the training process and how our selected models preform as well as the optimal hyperparameters to use, we may proceed with model build and split our saved data ***'X'*** and ***'y'*** into *training and test.* Since ***X'*** and ***y'*** represent data before implementing any pre-processing, we will need to apply the same pre-processing steps to retain the information gained from GridSearch. Re-applying the pre-processing steps at this stage is intended to avoid information leakage and prevent overfitting. Hence, we will applying the pre-processing with the same setting and component numbers by fitting on the training and transforming both sets as follows:

**Figure 31: Splitting the dataset and fitting pre-processing steps**



```
#Create model

# split the dataset into training and test sets using stratified sampling
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)
print(X.shape,X_train.shape)
```

```
(6073, 21) (4858, 21)
```

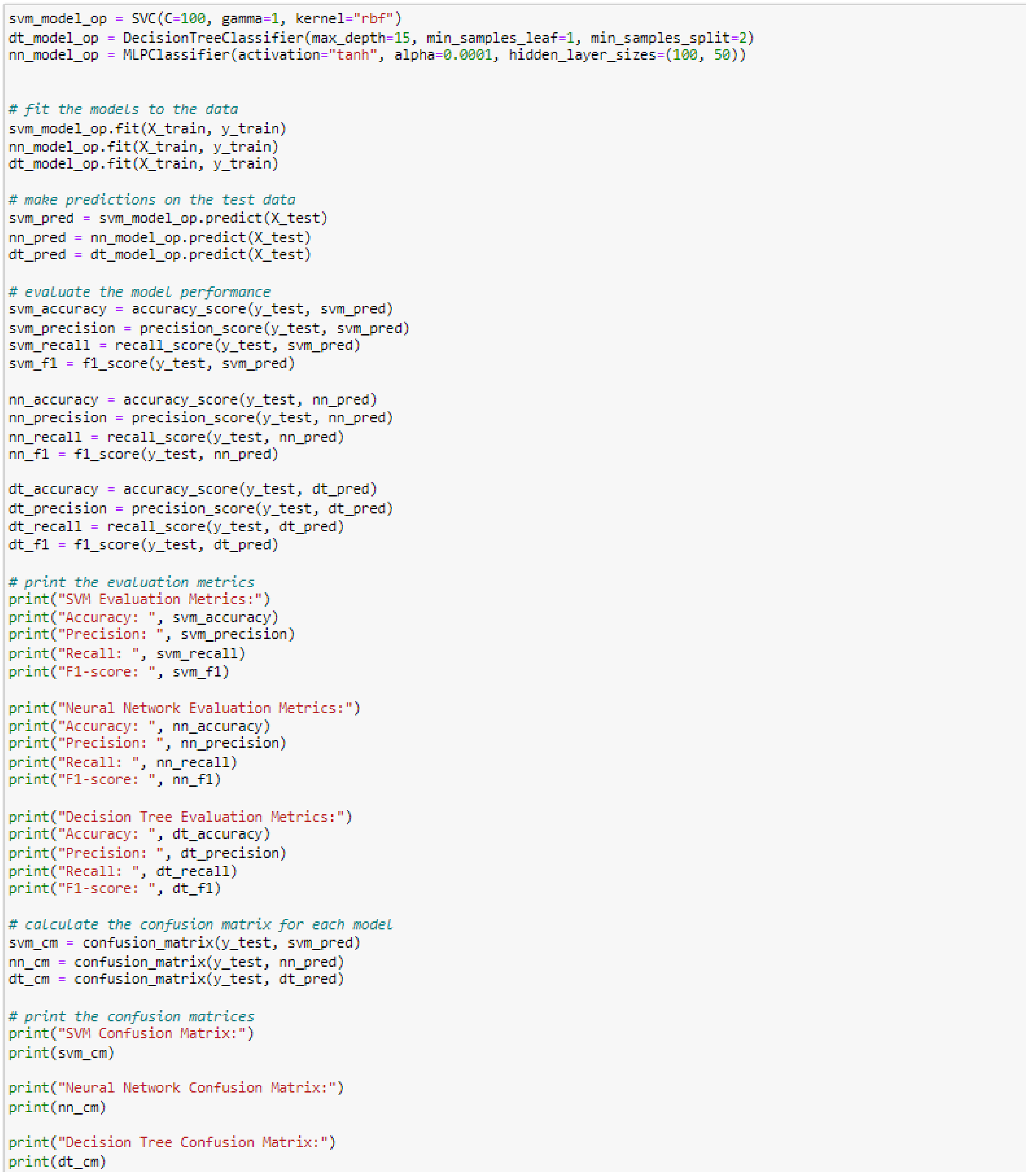
```
#Scaling the dataset

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

```
# Use PCA to reduce the dimensionality of the training data
pca = PCA(n_components=5)
X_train = pca.fit_transform(X_train)
X_test = pca.transform(X_test)
```

```
# Apply SMOTE only on the training set
smote = SMOTE(random_state=42)
X_train, y_train = smote.fit_resample(X_train, y_train)
```

As shown in the code snippet in Figure 31, we implemented the same pre-processing steps, however when we split the data into 80% training and 20% test, we set the stratify to yes in order to retain the portion of the class label. We also used SMOTE only on training in order for our model to obtain enough samples in both classes to learn efficiently, then test it on the unseen data which would be the test set. After this step, will be creating the model using the same hyperparameters obtained from GridSearchCV for our three selected models as follows:

**Figure 32: models creation using optimized hyperparameters**



```
svm_model_op = SVC(C=100, gamma=1, kernel="rbf")
dt_model_op = DecisionTreeClassifier(max_depth=15, min_samples_leaf=1, min_samples_split=2)
nn_model_op = MLPClassifier(activation="tanh", alpha=0.0001, hidden_layer_sizes=(100, 50))


# fit the models to the data
svm_model_op.fit(X_train, y_train)
nn_model_op.fit(X_train, y_train)
dt_model_op.fit(X_train, y_train)

# make predictions on the test data
svm_pred = svm_model_op.predict(X_test)
nn_pred = nn_model_op.predict(X_test)
dt_pred = dt_model_op.predict(X_test)

# evaluate the model performance
svm_accuracy = accuracy_score(y_test, svm_pred)
svm_precision = precision_score(y_test, svm_pred)
svm_recall = recall_score(y_test, svm_pred)
svm_f1 = f1_score(y_test, svm_pred)

nn_accuracy = accuracy_score(y_test, nn_pred)
nn_precision = precision_score(y_test, nn_pred)
nn_recall = recall_score(y_test, nn_pred)
nn_f1 = f1_score(y_test, nn_pred)

dt_accuracy = accuracy_score(y_test, dt_pred)
dt_precision = precision_score(y_test, dt_pred)
dt_recall = recall_score(y_test, dt_pred)
dt_f1 = f1_score(y_test, dt_pred)

# print the evaluation metrics
print("SVM Evaluation Metrics:")
print("Accuracy: ", svm_accuracy)
print("Precision: ", svm_precision)
print("Recall: ", svm_recall)
print("F1-score: ", svm_f1)

print("Neural Network Evaluation Metrics:")
print("Accuracy: ", nn_accuracy)
print("Precision: ", nn_precision)
print("Recall: ", nn_recall)
print("F1-score: ", nn_f1)

print("Decision Tree Evaluation Metrics:")
print("Accuracy: ", dt_accuracy)
print("Precision: ", dt_precision)
print("Recall: ", dt_recall)
print("F1-score: ", dt_f1)

# calculate the confusion matrix for each model
svm_cm = confusion_matrix(y_test, svm_pred)
nn_cm = confusion_matrix(y_test, nn_pred)
dt_cm = confusion_matrix(y_test, dt_pred)

# print the confusion matrices
print("SVM Confusion Matrix:")
print(svm_cm)

print("Neural Network Confusion Matrix:")
print(nn_cm)

print("Decision Tree Confusion Matrix:")
print(dt_cm)
```

As per the above, we obtained the following results when testing on the test set:

**Table 03: Performance metrics**

| **Model** | **Accuracy** | **Precision** | **Recall** | **F1-score** |
|---|---|---|---|---|
| **SVM** | 0.752 | 0.430 | 0.471 | 0.450 |
| **Neural Network** | 0.735 | 0.425 | 0.659 | 0.517 |
| **Decision Tree** | 0.699 | 0.366 | 0.548 | 0.439 |

**Table 04: SVM Confusion Matrix**

| | **Predicted Negative** | **Predicted Positive** |
|---|---|---|
| **Actual Negative** | 791 | 163 |
| **Actual Positive** | 138 | 123 |

**Table 05: Neural Network Confusion Matrix**

| | **Predicted Negative** | **Predicted Positive** |
|---|---|---|
| **Actual Negative** | 721 | 233 |
| **Actual Positive** | 89 | 172 |

**Table 06: Decision Tree Confusion Matrix**

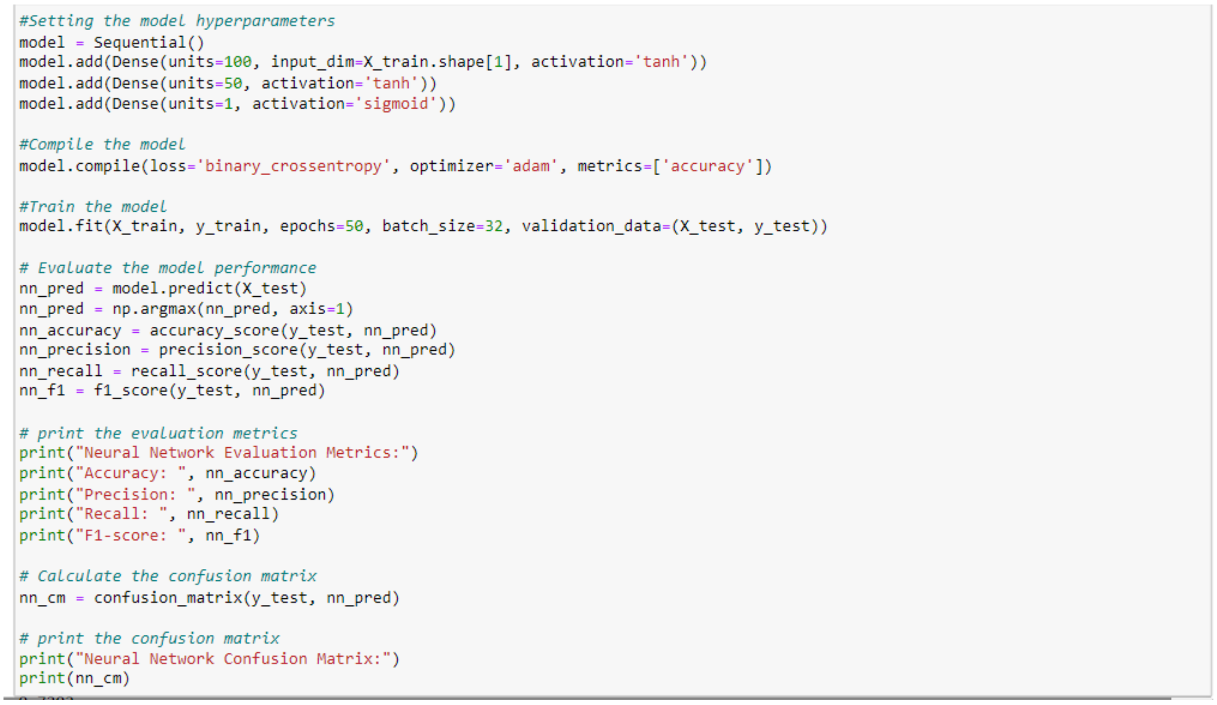
| | **Predicted Negative** | **Predicted Positive** |
|---|---|---|
| **Actual Negative** | 706 | 248 |
| **Actual Positive** | 118 | 143 |

As outputted in the evaluation metrics, the SVM model performs the best with an accuracy of 75.2%, while the Decision Tree model preforms with the lowest accuracy of 69.9%.  As the precision, recall, and F1-score vary across the models and across the classes, we should note the relevance which may depending on the specific problem. Since this represents the classification of exoplanet candidate, we would more interested in evaluating the true positives and true negatives.

Looking at the confusion matrices, we can see that the SVM model has the highest true negatives (791) and true positives (123), indicating that it correctly predicted the negative and positive cases more often than the other models. On the other hand, the Decision Tree model has the highest false positives (248), indicating that it incorrectly predicted positive cases more often than the other models. However, our results are not entirely compatible with those obtained by GridSearchCV for evaluation. While SVM performed better than Decision Tree and Neural Network on the test set, the Decision Tree model preformed the lowest in comparison to the Neural Network model.

At this stage, we may also create a deep learning architecture using the Keras library using the same dataset to compare the Sklearn Neural Network and Keras Neural Network. To do so, we the used same hyperparameters outputted from GridSearch to define the architecture of the Keras neural network model. The Keras model has the same number of hidden layers and neurons as the MLPClassifier, with the same activation function. The binary_crossentropy loss function and the Adam optimizer were used to compile the model. The epochs was set to 50 to determine the number of iterations used on the training data to update the weights of the model. Afterwards, the model was trained on the training data, and its performance was evaluated on the test data as follows:

**Figure 33: Neural Network Architecture using Keras**



```
#Setting the model hyperparameters
model = Sequential()
model.add(Dense(units=100, input_dim=X_train.shape[1], activation='tanh'))
model.add(Dense(units=50, activation='tanh'))
model.add(Dense(units=1, activation='sigmoid'))

#Compile the model
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

#Train the model
model.fit(X_train, y_train, epochs=50, batch_size=32, validation_data=(X_test, y_test))

# Evaluate the model performance
nn_pred = model.predict(X_test)
nn_pred = np.argmax(nn_pred, axis=1)
nn_accuracy = accuracy_score(y_test, nn_pred)
nn_precision = precision_score(y_test, nn_pred)
nn_recall = recall_score(y_test, nn_pred)
nn_f1 = f1_score(y_test, nn_pred)

# print the evaluation metrics
print("Neural Network Evaluation Metrics:")
print("Accuracy: ", nn_accuracy)
print("Precision: ", nn_precision)
print("Recall: ", nn_recall)
print("F1-score: ", nn_f1)

# Calculate the confusion matrix
nn_cm = confusion_matrix(y_test, nn_pred)

# print the confusion matrix
print("Neural Network Confusion Matrix:")
print(nn_cm)
```

**Table 07: Comparison between Keras NN Score and Sklearn NN score**

| Evaluation Metric | Score Keras NN | Sklearn NN |
|---|---|---|
| Accuracy | 0.7852 | 0.735 |
| Precision | 0.0 | 0.425 |
| Recall | 0.0 | 0.659 |
| F1-score | 0.0 | 0.517 |

**Table 08: Confusion Matrix**

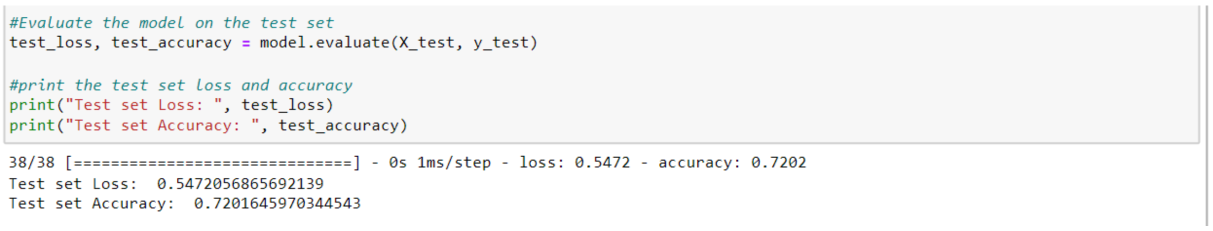
| | **Predicted Negative** | **Predicted Positive** |
|---|---|---|
| **Actual Negative** | 954 | 0 |
| **Actual Positive** | 261 | 0 |

The accuracy of the model on the test data is 0.785, which means that it classified approximately 78.5% of the test samples correctly. However, the precision and recall scores are both 0 which indicates that the model was not able to correctly classify the positive samples, as both the true positive and false positive account for zero. In addition, the F1-score is also 0 establishing that the model's performance is very low. The confusion matrix confirms that the model has predicted all the samples as negative class, and none of the positive samples have been correctly identified. In comparison with the Sklearn NN model, accuracy-wise the

This could be because the model is biased towards the negative class or may not have been trained properly. Hence, further adjustment and fine tuning will be required in order to enhance the overall performance of the model.

We may also evaluate the performance of a trained model on a test dataset using the evaluate method in Keras as follows:

**Figure 34: Test set evaluation using loss and accuracy**



```
#Evaluate the model on the test set
test_loss, test_accuracy = model.evaluate(X_test, y_test)

#print the test set loss and accuracy
print("Test set Loss: ", test_loss)
print("Test set Accuracy: ", test_accuracy)
```

```
38/38 [==============================] - 0s 1ms/step - loss: 0.5472 - accuracy: 0.7202
Test set Loss:  0.5472056865692139
Test set Accuracy:  0.7201645970344543
```

As per Figure 34, the model has a loss of 0.5472 and an accuracy of 0.7202. This would imply that the model correctly classified 72.02% of the samples in the unseen dataset, while the remaining 27.98% were misclassified.

## 6. References.

[1] R. Tang, "Machine Learning Meets Astronomy," 2020 International Conference on Computing and Data Science (CDS), Stanford, CA, USA, 2020, pp. 3-6, doi: 10.1109/CDS49703.2020.00008.

[2] V. Bahel and M. Gaikwad, "A Study of Light Intensity of Stars for Exoplanet Detection using Machine Learning," 2022 IEEE Region 10 Symposium (TENSYMP), Mumbai, India, 2022, pp. 1-5, doi: 10.1109/TENSYMP54529.2022.9864366.

[3]. Kepler Mission Overview. (n.d.). Retrieved March 09, 2023, from https://www.nasa.gov/mission_pages/kepler/overview/index.html

[4] A. R. Bhamare, A. Baral and S. Agarwal, "Analysis of Kepler Objects of Interest using Machine Learning for Exoplanet Identification," 2021 International Conference on Intelligent Technologies (CONIT), Hubli, India, 2021, pp. 1-8, doi: 10.1109/CONIT51480.2021.9498407.

[5] R. Jagtap, U. Inamdar, S. Dere, M. Fatima and N. B. Shardoor, "Habitability of Exoplanets using Deep Learning," *2021 IEEE International IOT, Electronics and Mechatronics Conference (IEMTRONICS)*, Toronto, ON, Canada, 2021, pp. 1-6, doi: 10.1109/IEMTRONICS52119.2021.9422571.