Predicting Exoplanets Using Big Data Techniques: A Classification Task

1. Introduction

The discovery and classification of exoplanets—planets orbiting stars outside our solar system—has become a major focus in modern astronomy. With the advent of advanced telescopes and space missions like NASA's Kepler Space Telescope, large amounts of data have been collected, leading to the identification of thousands of potential exoplanets. However, classifying these candidates as confirmed exoplanets or false positives requires considerable manual analysis and verification, which can be time-consuming and resource-intensive. In this context, machine learning has emerged as a powerful tool for automating the classification process, improving both efficiency and accuracy.

This project aims to apply big data tools such as Hadoop Distributed File System (HDFS), Apache Pyspark and machine learning techniques to the classification of exoplanet candidates. Using a dataset from NASA's Exoplanet Archive, various models are trained to predict whether a candidate exoplanet is a confirmed exoplanet or a false positive. The project explores multiple machine learning algorithms, including Naive Bayes, Logistic Regression, Support Vector Machines, Random Forests, and Multilayer Perceptrons, each offering different strengths in handling the complexity and variability of the data. By comparing the performance of these models, the project seeks to identify the most effective approach for exoplanet classification.

Beyond model development, the project also aims to apply the selected model to real-world data to predict the classification of currently unconfirmed exoplanet candidates. This approach showcases the potential for machine learning to accelerate the process of exoplanet discovery, allowing astronomers to focus on the most promising candidates for further investigation. Through this work, big data analysis and machine learning is demonstrated as a valuable asset in the growing field of exoplanet research.

1.1 Research Questions

These relevant research questions can guide the objectives and outcomes of this project, helping to focus on both the scientific and methodological aspects of exoplanet classification using machine learning:

1. How accurately can machine learning models classify exoplanet candidates as confirmed exoplanets or false positives?
2. Which machine learning algorithms perform best for exoplanet classification, and why?
3. How can machine learning models be used to predict and validate current exoplanet candidates that have yet to be confirmed or classified?

1.2 Aims

The aim of this project is to develop and evaluate machine learning models for the classification of exoplanet candidates, distinguishing between confirmed exoplanets and false positives. By leveraging various machine learning algorithms, the project seeks to optimise the classification process, reduce manual analysis, and enhance the efficiency and accuracy of exoplanet discovery.

Additionally, the project aims to apply the best-performing model to real-world data to predict the classification of unconfirmed exoplanet candidates, demonstrating the practical utility of machine learning in advancing exoplanet research.

1.3 Objectives

To achieve the aim of the project, the following objectives have been set:

1. **Data Preprocessing**: Clean and preprocess the exoplanet dataset, handling missing values, normalising features, and addressing outliers to prepare the data for machine learning models.
2. **Exploratory Data Analysis (EDA) and Statistical Tests**: Perform EDA and statistical tests to gain insights into the data, identify key features, and understand relationships between variables that influence exoplanet classification.
3. **Feature Selection and Handling Class Imbalance**: Identify the most important features that contribute to accurate classification and refine the dataset by removing irrelevant or low-importance features. Also address any6 class imbalance in the dataset using techniques such as oversampling and synthetic data generation to improve model performance.
4. **Model Selection**: Train and evaluate a variety of machine learning models, including Naive Bayes, Logistic Regression, Random Forest, Support Vector Machines, and Multilayer Perceptrons, to compare their performance.
5. **Model Evaluation**: Evaluate the models using performance metrics such as accuracy, precision, recall, F1 score, and ROC-AUC to determine the best-performing algorithm.
6. **Prediction on Real-World Data**: Apply the selected model to predict the classification of current exoplanet candidates in the dataset as either confirmed exoplanets or false positives.

2. Background literature

The application of machine learning (ML) techniques in the field of exoplanet detection and classification has gained significant traction in recent years. This surge is largely attributed to the increasing volume of astronomical data generated by missions such as Kepler and TESS, which necessitate efficient and automated methods for data analysis. Traditional methods of exoplanet detection, such as the transit method and radial velocity techniques, often require extensive manual intervention and are limited by observational constraints (Singh, 2024; Jin et al., 2022).

In contrast, ML offers a promising avenue for enhancing the accuracy and efficiency of exoplanet identification. One of the primary advantages of ML in exoplanet research is its ability to classify and predict exoplanet characteristics based on large datasets. For instance, Nayak (2024) highlights the use of classification models to analyse Kepler data, demonstrating that machine learning can effectively categorise exoplanets based on their features. Similarly, Basak et al. (2018) emphasises the need for automated annotation methods to streamline the classification process, thereby reducing the time and effort required for manual analysis. This automation is crucial as the volume of data continues to grow, making manual classification increasingly impractical.

Moreover, various ML algorithms have been employed to model specific relationships within exoplanet data. Ulmer-Moll et al. (2019) explore the mass-radius relationship of exoplanets using random forest models, showcasing how ML can be utilised to predict exoplanet radii from observable parameters. This predictive capability is further supported by the work of (Shallue & Vanderburg, 2018), who applied deep learning techniques to classify potential planet signals from Kepler data, achieving high accuracy rates. Such advancements underscore the transformative potential of ML in refining our understanding of exoplanet characteristics and their distributions.

In addition to classification and prediction, ML has also been instrumental in anomaly detection within exoplanetary atmospheres. Forestano (2023) discusses the use of unsupervised learning to identify unusual chemical compositions in exoplanet transit spectra, which can provide insights into the atmospheric conditions of these distant worlds. This approach aligns with the findings of (Pham, 2022), who utilised ML to assess the presence of water and other critical elements on terrestrial exoplanets, further illustrating the diverse applications of ML in exoplanet research. The integration of scientific domain knowledge into ML models has also proven beneficial. Ansdell et al. (2018) demonstrate that incorporating domain-specific insights can significantly enhance model performance, achieving remarkable accuracy in classifying transit signals. This highlights the importance of interdisciplinary collaboration in optimising ML applications for exoplanet detection.

However, due to the current smaller scale of space data and the relatively limited number of exoplanet candidates discovered so far, there has been little focus on applying big data techniques to publicly available exoplanet datasets, particularly for exoplanet classification. Most existing methods rely on traditional statistical models or manual classification, which, while effective, may not be scalable as the volume of data continues to grow with ongoing and future missions like TESS and the James Webb Space Telescope. This represents a gap in the literature that this project aims to address by exploring the potential of big data tools, such as Hadoop and PySpark, for efficient processing and classification. By building a scalable framework now, this project prepares for a future where these big data techniques will be essential to handle the exponentially growing datasets expected in exoplanet discovery.

3. Project Impact/ Justification of Relevance

Current techniques for classifying exoplanet candidates—such as those employed by NASA's Kepler mission and other observational studies—primarily rely on human-driven analysis of light curves and radial velocity data. These methods involve extensive manual examination of transit signals, where dips in star brightness suggest the presence of orbiting planets. In addition, sophisticated statistical models like Bayesian inference, coupled with astrophysical knowledge, are often applied to distinguish between true exoplanets and false positives, such as stellar activity or binary star systems. While effective, these approaches are time-consuming and labor-intensive.

This project seeks to address these limitations by implementing machine learning models that can automate the classification process, offering significant enhancements in efficiency, scalability, and accuracy. By leveraging a dataset from NASA's Exoplanet Archive, the project applies machine learning algorithms to classify exoplanet candidates based on multiple astrophysical features, such as orbital period, planet radius, equilibrium temperature, and star-planet distance. These algorithms are capable of handling complex, high-dimensional data more efficiently than traditional methods.

One of the primary advantages of this approach is its ability to reduce human intervention in the preliminary stages of classification. Machine learning models, once trained, can process thousands of candidates quickly, flagging high-probability exoplanets for further investigation. This significantly speeds up the workflow compared to manual classification, where astronomers may spend considerable time on each individual candidate.

Furthermore, machine learning models excel in pattern recognition and can detect subtle relationships between variables that might be overlooked by manual techniques. For instance, the Random Forest model can capture non-linear interactions between features, while Support Vector Machines are adept at finding optimal decision boundaries between confirmed exoplanets and false positives. Neural networks, such as the Multilayer Perceptron, further enhance this capability by learning deeper, more intricate patterns in the data. These models can complement existing statistical techniques, potentially improving classification accuracy and reducing false positive rates.

Another significant impact of this approach is its scalability. Since it run using big data data structures and techniques that are capable of running large scale datasets, it can be scaled to datasets of several magnitutes more. With upcoming space missions like NASA’s TESS (Transiting Exoplanet Survey Satellite) and the James Webb Space Telescope expected to generate even larger volumes of data, the need for automated, scalable classification techniques is more urgent than ever. Machine learning offers a solution that can keep pace with the

increasing data flow, enabling continuous, real-time classification of exoplanet candidates.

Finally, the practical application of this project to current, unclassified exoplanet candidates demonstrates its direct relevance to ongoing astronomical research. By predicting whether current candidates are likely to be confirmed exoplanets or false positives, this project provides a valuable tool for astronomers to prioritise further investigations, potentially accelerating the discovery of new exoplanets.

4. Methodology

4.1. Dataset

The dataset used for this project is sourced from NASA's Exoplanet Archive, specifically the Kepler Candidates Table. It is comprised of identification columns, exoplanet archive information, project disposition columns, transit properties, threshold-crossing event (TCE) information, stellar parameters, KIC parameters and pixel-based KOI vetting statistics. Key features include orbital period, planet radius, equilibrium temperature, and the disposition (classification) of the planet—whether it is a confirmed exoplanet, candidate, or false positive. These features provide essential data for the machine learning models to classify exoplanet candidates accurately. Although the dataset has fewer than 10,000 entries, its complexity and the scientific significance of its content make it highly relevant for advanced analysis.

Despite its relatively small size, this dataset is appropriate for big data analysis due to the high dimensionality of the features and the need for distributed processing. Machine learning models that deal with multidimensional, complex data can benefit from big data tools like Hadoop and PySpark, which allow for efficient data preprocessing, training, and evaluation on multiple nodes. Moreover, as future space missions produce increasing amounts of data, using a scalable big data framework ensures that the project can grow to meet future demands.

Additionally, the dataset's relevance lies in its role in exoplanet discovery—an area of great importance in modern astronomy. Automating the classification of exoplanet candidates with machine learning models, enhanced by big data tools, allows researchers to prioritise candidates for further investigation more efficiently. The use of distributed frameworks also provides immediate value by handling complex astrophysical data in a robust, distributed manner. This makes the Kepler Candidates dataset an ideal choice for big data analysis, despite its current size.

4.2. Choosing a Measure of Success

The performance metrics used in this project—accuracy, precision, recall, F1 score, and area under the ROC curve (AUC)—were carefully selected to provide a comprehensive evaluation of the machine learning models' ability to classify exoplanet candidates. Accuracy was chosen to give a general sense of how well the model correctly classifies both confirmed exoplanets and false positives. However, accuracy alone can be misleading in cases where the dataset is imbalanced, making additional metrics necessary to assess the model's true effectiveness.

Precision and recall are particularly important in this context. Precision measures how often the model's positive predictions (confirmed exoplanets) are correct, reducing the number of false positives, which is critical for preventing unnecessary follow-up investigations. Recall, on the other hand, focuses on minimising false negatives, ensuring that the model correctly identifies as many true exoplanets as possible, which is vital for prioritizing potential discoveries.

The F1 score was chosen because it balances precision and recall, making it an essential metric when dealing with imbalanced datasets, such as those in exoplanet classification, where false positives often outnumber confirmed exoplanets. Finally, the AUC was included to assess the model's ability to distinguish between the two classes across different thresholds, ensuring that the model can effectively separate confirmed exoplanets from false positives, even as decision thresholds are adjusted. These metrics collectively ensure that the model is evaluated thoroughly in terms of both accuracy and the scientific implications of misclassifications.

4.3. Deciding on an Evaluation Protocol

The evaluation protocol chosen for this project was the hold-out validation set method, where the dataset was split into separate training and test sets. This approach was selected over other commonly used methods such as K-fold cross-validation and iterated K-fold cross-validation. The hold-out method was chosen initially following the principle of Occam's razor, which suggests that the simplest solution should be tried first. If the model's performance had been suboptimal, a more robust technique like K-fold cross-validation would have been employed to ensure more reliable evaluation across different data splits. However, since the model performed exceptionally well using the hold-out validation set, this simpler evaluation protocol proved to be sufficient for the project, validating the model's effectiveness without the need for more complex validation techniques.

---

5. Data Loading

The first step to implementing this big data project was to connect to the Lena cluster and transfer the dataset from a local machine to the Hadoop Distributed File System (HDFS), which is capable of distributing and managing large volumes of data across multiple nodes in a cluster. Once this was done, a PySpark Jupyter notebook was opened on the cluster, and a Spark session was initialised. The data was then loaded into a PySpark DataFrame, which allows for efficient processing of large datasets in a distributed environment.

This DataFrame was subsequently converted to a Pandas DataFrame for data preprocessing and exploratory data analysis (EDA) because Pandas provides a more user-friendly and flexible interface for manipulating and analysing smaller portions of data. While PySpark excels in handling large datasets across distributed systems, Pandas is better suited for detailed operations such as data cleaning, transformation, and statistical exploration on smaller subsets of the data that fit in memory. Since this dataset is relatively small for big data analysis, a Pandas dataframe is preffered for these operations. The dataframe would be later converted to a Pyspark dataframe for applying machine learning techniques.

5.1 Data Loading from Local Machine to HDFS

To begin the analysis, the following steps were taken to ensure proper connection to the hadoop server and uploading of files from the local machine to the nodes. These steps were obtained from the programming activity of the second week of the Big Data Analysis course:

1. Connecting to the cluster
   ssh obanj001@lena.doc.gold.ac.uk
2. Creating a directory called cw1
   mkdir cw2
3. Changing directories to cw
   cd cw2
4. Transferring the exoplanet data from my local machine (in another terminal)
   scp /Users/fisayo/Downloads/exoplanet_data2.csv obanj001@lena.doc.gold.ac.uk:cw2
5. Copy the data to Hadoop
   hadoop fs -mkdir coursework2
   hadoop fs -copyFromLocal exoplanet_data2.csv ./coursework2

5.2 Data Loading on the Pyspark Jupyter Notebook

In [1]:
```
# Importing necessary libraries
import math
import numpy as np
import pandas as pd # Use pyspark.pandas as ps if running locally
import seaborn as sns
from scipy import stats
from functools import reduce
from pyspark.ml import Pipeline
import matplotlib.pyplot as plt
from pyspark.sql.window import Window
from pyspark.mllib.util import MLUtils
from pyspark.sql import functions as F
from pyspark.sql.types import DoubleType
from pyspark import SparkContext, SparkConf
from sklearn.preprocessing import RobustScaler
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.evaluation import RegressionEvaluator
from pyspark.sql import SparkSession, SQLContext, DataFrame
from scipy.stats import chi2_contingency, shapiro, f_oneway
from sklearn.feature_selection import SelectKBest, f_classif
from sklearn.metrics import confusion_matrix, roc_curve, auc
from pyspark.mllib.tree import RandomForest, RandomForestModel
from pyspark.sql.functions import row_number, sqrt, col, reduce, sum as F_sum
from pyspark.ml.evaluation import BinaryClassificationEvaluator, MulticlassClassificationEvaluator
from pyspark.ml.classification import LogisticRegression, RandomForestClassifier, NaiveBayes, LinearSVC, Multilay
```

In [2]:
```
# Starting a spark session
spark = SparkSession.builder.getOrCreate()
```

In [3]:
```
# Sanity check that a spark session is running
spark
```

Out[3]:

SparkSession - in-memory

SparkContext

Spark UI

Version	`v3.5.1`
Master	`yarn`
AppName	`pyspark-shell`

In [4]:
```
# Load the dataset from the HDFS directory and create a pyspark dataframe
```

```
df = spark.read.csv("coursework2/exoplanet_data2.csv")

# View the first row of the dataframe
df.head(1)
```

Out[4]:
```
[Row(_c0='rowid', _c1='kepid', _c2='kepoi_name', _c3='kepler_name', _c4='koi_disposition', _c5='koi_vet_stat', _c6='koi_vet_date', _c7='koi_pdisposition', _c8='koi_score', _c9='koi_fpflag_nt', _c10='koi_fpflag_ss', _c11='koi_fpflag_co', _c12='koi_fpflag_ec', _c13='koi_disp_prov', _c14='koi_comment', _c15='koi_period', _c16='koi_time0bk', _c17='koi_time0', _c18='koi_eccen', _c19='koi_longp', _c20='koi_impact', _c21='koi_duration', _c22='koi_ingress', _c23='koi_depth', _c24='koi_ror', _c25='koi_srho', _c26='koi_fittype', _c27='koi_prad', _c28='koi_sma', _c29='koi_incl', _c30='koi_teq', _c31='koi_insol', _c32='koi_dor', _c33='koi_limbdark_mod', _c34='koi_ldm_coeff4', _c35='koi_ldm_coeff3', _c36='koi_ldm_coeff2', _c37='koi_ldm_coeff1', _c38='koi_parm_prov', _c39='koi_max_sngle_ev', _c40='koi_max_mult_ev', _c41='koi_model_snr', _c42='koi_count', _c43='koi_num_transits', _c44='koi_tce_plnt_num', _c45='koi_tce_delivname', _c46='koi_quarters', _c47='koi_bin_oedp_sig', _c48='koi_trans_mod', _c49='koi_model_dof', _c50='koi_model_chisq', _c51='koi_datalink_dvr', _c52='koi_datalink_dvs', _c53='koi_steff', _c54='koi_slogg', _c55='koi_smet', _c56='koi_srad', _c57='koi_smass', _c58='koi_sage', _c59='koi_sparprov', _c60='ra', _c61='dec', _c62='koi_kepmag', _c63='koi_gmag', _c64='koi_rmag', _c65='koi_imag', _c66='koi_zmag', _c67='koi_jmag', _c68='koi_hmag', _c69='koi_kmag', _c70='koi_fwm_stat_sig', _c71='koi_fwm_sra', _c72='koi_fwm_sdec', _c73='koi_fwm_srao', _c74='koi_fwm_sdeco', _c75='koi_fwm_prao', _c76='koi_fwm_pdeco', _c77='koi_dicco_mra', _c78='koi_dicco_mdec', _c79='koi_dicco_msky', _c80='koi_dikco_mra', _c81='koi_dikco_mdec', _c82='koi_dikco_msky')]
```

In [5]:
```
# Changing the pyspark dataframe to a pandas dataframe because its easier to work
pd_df = df.toPandas()

# Now you can work with the pandas DataFrame
pd_df.head(5)
```

Out[5]:

	_c0	_c1	_c2	_c3	_c4	_c5	_c6	_c7	_c8	_c9	...	_c73	
0	rowid	kepid	kepoi_name	kepler_name	koi_disposition	koi_vet_stat	koi_vet_date	koi_pdisposition	koi_score	koi_fpflag_nt	...	koi_fwm_srao	k
1	1	10797460	K00752.01	Kepler-227 b	CONFIRMED	Done	2018-08-16	CANDIDATE	1.0000	0	...	0.43000	
2	2	10797460	K00752.02	Kepler-227 c	CONFIRMED	Done	2018-08-16	CANDIDATE	0.9690	0	...	-0.63000	
3	3	10811496	K00753.01	None	CANDIDATE	Done	2018-08-16	CANDIDATE	0.0000	0	...	-0.02100	
4	4	10848459	K00754.01	None	FALSE POSITIVE	Done	2018-08-16	FALSE POSITIVE	0.0000	0	...	-0.11100	

5 rows × 83 columns

---

6. Data Preprocessing

Data Preprocessing was done to ensure that the data was clean, consistent, and organised in a way that would prepare it for further analysis. his dataset had some issues, such as empty columns and the header being stored in the first row. The first preprocessing step involved setting the first row as the header and then removing that row. Next, the rowid column was set as the index, followed by resetting the index to maintain an organised structure. Afterward, the data was subsetted to include the most relevant columns. These columns were selected based on extensive research on factors that determine an exoplanet, obtained from Google searches and NASA's Exoplanet Archive.

Once the data was subset, it was checked for duplicates, but no duplicates were found. Null values were then identified and handled. Mean imputation was applied for numeric values because it helps preserve the central tendency of the data without introducing significant bias, especially when the missing values are random. Mode imputation was used for categorical values because the mode represents the most frequent category and is generally the least disruptive method for handling missing categorical data.

After the data was cleared of null values that could introduce bias or errors in the analysis, it was checked for outliers. Outliers were present, but they were not removed because they could represent valid, extreme observations, and removing them could lead to the loss of important information. It was decided to retain the outliers and assess how well the machine learning models performed with them included. If the models performed poorly, further steps could be taken to handle the outliers. The presence of outliers, which indicated skewness and non-normality in the data, was considered during subsequent analysis to ensure the models were properly evaluated with these characteristics in mind.

6.1 Adding Data Header

In [6]:
```
# Make the first row the header
new_header = pd_df.iloc[0]  # Extract the third row (index 2)
pd_df.columns = new_header

# Drop the first row
pd_df = pd_df.iloc[1:]

# Set column 'rowid' as the index
pd_df = pd_df.set_index('rowid')
```

```
# Reset the index
pd_df = pd_df.reset_index(drop=True)
pd_df.head(5)
```

Out[6]:

	kepid	kepoi_name	kepler_name	koi_disposition	koi_vet_stat	koi_vet_date	koi_pdisposition	koi_score	koi_fpflag_nt	koi_fpflag_ss	...
0	10797460	K00752.01	Kepler-227 b	CONFIRMED	Done	2018-08-16	CANDIDATE	1.0000	0	0	...
1	10797460	K00752.02	Kepler-227 c	CONFIRMED	Done	2018-08-16	CANDIDATE	0.9690	0	0	...
2	10811496	K00753.01	None	CANDIDATE	Done	2018-08-16	CANDIDATE	0.0000	0	0	...
3	10848459	K00754.01	None	FALSE POSITIVE	Done	2018-08-16	FALSE POSITIVE	0.0000	0	1	...
4	10854555	K00755.01	Kepler-664 b	CONFIRMED	Done	2018-08-16	CANDIDATE	1.0000	0	0	...

5 rows × 82 columns

6.2 Subsetting Data

In [7]:

```
# Subsetting the columns of the dataset relevant to the analysis according to literature
# Subsetting based on the 24 relevant columns

columns_to_keep = [
    'koi_score',           # Disposition score
    'koi_fpflag_nt',       # Not Transit-Like False Positive Flag
    'koi_fpflag_ss',       # Stellar Eclipse False Positive Flag
    'koi_fpflag_co',       # Centroid Offset False Positive Flag
    'koi_fpflag_ec',       # Ephemeris Match False Positive Flag
    'koi_prad',            # Planetary Radius [Earth radii]
    'koi_period',          # Orbital Period [days]
    'koi_duration',        # Transit Duration [hrs]
    'koi_depth',           # Transit Depth [ppm]
    'koi_srho',            # Fitted Stellar Density [g/cm³]
    'koi_model_snr',       # Transit Signal-to-Noise Ratio
    'koi_impact',          # Impact Parameter
    'koi_insol',           # Insolation Flux [Earth flux]
    'koi_teq',             # Equilibrium Temperature [K]
    'koi_slogg',           # Stellar Surface Gravity [log10(cm/s²)]
    'koi_steff',           # Stellar Effective Temperature [K]
    'koi_srad',            # Stellar Radius [Solar radii]
    'koi_smass',           # Stellar Mass [Solar mass]
    'koi_ror',             # Planet-Star Radius Ratio
    'koi_num_transits',    # Number of Transits
    'koi_tce_plnt_num',    # TCE Planet Number
    'koi_dor',             # Planet-Star Distance over Star Radius
    'koi_disposition'     # Target: Confirmed, Candidate or False Positive
]

# Creating the new subset DataFrame
pd_df2 = pd_df[columns_to_keep]

# Display the first few rows of the new subset DataFrame
pd_df2.head()
```

Out[7]:

	koi_score	koi_fpflag_nt	koi_fpflag_ss	koi_fpflag_co	koi_fpflag_ec	koi_prad	koi_period	koi_duration	koi_depth	koi_srho	...	koi_teq
0	1.0000	0	0	0	0	2.26	9.488035570	2.95750	6.158E+02	3.20796	...	793.0
1	0.9690	0	0	0	0	2.83	54.418382700	4.50700	8.748E+02	3.02368	...	443.0
2	0.0000	0	0	0	0	14.60	19.899139950	1.78220	1.0829E+04	7.29555	...	638.0
3	0.0000	0	1	0	0	33.46	1.736952453	2.40641	8.0792E+03	0.22080	...	1395.0
4	1.0000	0	0	0	0	2.75	2.525591777	1.65450	6.033E+02	1.98635	...	1406.0

5 rows × 23 columns

6.3 Removing duplicates

In [8]:

```
# Check for duplicates
duplicates = pd_df2.duplicated()

# Print number of duplicated rows
print("Number of duplicate rows:", duplicates.sum())

# Remove duplicate rows (by default, keeps the first occurrence)
pd_df2 = pd_df2.drop_duplicates()
```

```
# Print the DataFrame after removing duplicates
print("\nDataFrame after removing duplicates:" )
pd_df2.head(5)
```

```
Number of duplicate rows: 0

DataFrame after removing duplicates:
```

Out[8]:

	koi_score	koi_fpflag_nt	koi_fpflag_ss	koi_fpflag_co	koi_fpflag_ec	koi_prad	koi_period	koi_duration	koi_depth	koi_srho	...	koi_teq
0	1.0000	0	0	0	0	2.26	9.488035570	2.95750	6.158E+02	3.20796	...	793.0
1	0.9690	0	0	0	0	2.83	54.418382700	4.50700	8.748E+02	3.02368	...	443.0
2	0.0000	0	0	0	0	14.60	19.899139950	1.78220	1.0829E+04	7.29555	...	638.0
3	0.0000	0	1	0	0	33.46	1.736952453	2.40641	8.0792E+03	0.22080	...	1395.0
4	1.0000	0	0	0	0	2.75	2.525591777	1.65450	6.033E+02	1.98635	...	1406.0

5 rows × 23 columns

6.4 Handling null values

In [9]:
```
# Checking for the count of null values by column
print(pd_df2.isnull().sum())
```

```
0
koi_score           1510
koi_fpflag_nt          0
koi_fpflag_ss          0
koi_fpflag_co          0
koi_fpflag_ec          0
koi_prad             363
koi_period             0
koi_duration           0
koi_depth            363
koi_srho             321
koi_model_snr        363
koi_impact           363
koi_insol            321
koi_teq              363
koi_slogg            363
koi_steff            363
koi_srad             363
koi_smass            363
koi_ror              363
koi_num_transits    1142
koi_tce_plnt_num     346
koi_dor              363
koi_disposition        0
dtype: int64
```

In [10]:
```
# Get a list of columns that contain null values
columns_with_null = pd_df2.columns[pd_df2.isnull().any()].tolist()

# Print the list of columns with null values
print(len(columns_with_null))
```

```
16
```

In [11]:
```
# Convert columns to numeric where possible, but don't change the non-numeric columns
for col in pd_df2.columns:
    try:
        pd_df2[col] = pd.to_numeric(pd_df2[col], errors='raise')  # Try to convert to numeric
    except ValueError:
        # If it raises a ValueError, the column remains as it is
        pass
```

In [12]:
```
def mean_imputation(df):
    """
    This function imputes missing values in numeric columns of a DataFrame using the mean of the column.

    Parameters:
    df (pd.DataFrame): The input DataFrame with potential missing values.

    Returns:
```

```
    pd.DataFrame: A DataFrame with missing values in numeric columns imputed with the mean.
    """
    # Identify numeric columns
    numeric_cols = df.select_dtypes(include=[np.number]).columns

    # Loop through each numeric column and impute missing values with the mean
    for col in numeric_cols:
        if df[col].isnull().sum() > 0:  # Check if the column has missing values
            mean_value = df[col].mean()  # Compute the mean of the column
            df[col].fillna(mean_value, inplace=True)  # Impute missing values with the mean

    return df

pd_df2 = mean_imputation(pd_df2)
```

In [13]:
```
def mode_imputation(df):
    """
    This function imputes missing values in non-numeric columns of a DataFrame using the mode (most common value)

    Parameters:
    df (pd.DataFrame): The input DataFrame with potential missing values.

    Returns:
    pd.DataFrame: A DataFrame with missing values in non-numeric columns imputed with the mode.
    """
    # Identify non-numeric columns
    non_numeric_cols = df.select_dtypes(exclude=[float, int]).columns

    # Loop through each non-numeric column and impute missing values with the mode
    for col in non_numeric_cols:
        if df[col].isnull().sum() > 0:  # Check if the column has missing values
            mode_value = df[col].mode()[0]  # Compute the mode (most frequent value)
            df[col].fillna(mode_value, inplace=True)  # Impute missing values with the mode

    return df

pd_df2 = mode_imputation(pd_df2)
```

In [14]:
```
# Sanity check that all the missing values were handled
print(pd_df2.isnull().sum())
```

```
0
koi_score           0
koi_fpflag_nt       0
koi_fpflag_ss       0
koi_fpflag_co       0
koi_fpflag_ec       0
koi_prad            0
koi_period          0
koi_duration        0
koi_depth           0
koi_srho            0
koi_model_snr       0
koi_impact          0
koi_insol           0
koi_teq             0
koi_slogg           0
koi_steff           0
koi_srad            0
koi_smass           0
koi_ror             0
koi_num_transits    0
koi_tce_plnt_num    0
koi_dor             0
koi_disposition     0
dtype: int64
```

6.5 Handling outliers

In [15]:
```
def detect_outliers_zscore(df, threshold=3):
    """
    Detects outliers in a given DataFrame using the Z-score method.

    This function computes the Z-scores for each numerical column in the DataFrame
    and counts the number of outliers based on a specified threshold. An outlier
    is defined as a data point whose Z-score is greater than the absolute value
    of the threshold.

    Parameters:
    df : pandas.DataFrame
    threshold : int or float, optional
    The Z-score threshold for identifying outliers. The default is 3, which means any data point with a Z-score g
```

```
    Returns: None
    Prints the column name and the number of outliers in that column
    """
    outlier_counts = {}

    for column in df.select_dtypes(include=['float64', 'int64']).columns:
        z_scores = stats.zscore(df[column].dropna())  # Compute Z-scores, ignoring NaNs
        outliers = (abs(z_scores) > threshold)
        outlier_counts[column] = outliers.sum()  # Count number of outliers

    for column, count in outlier_counts.items():
        print(f"{column}: {count}")

outliers_count = detect_outliers_zscore(pd_df2, threshold=3)
```

```
koi_score: 0
koi_fpflag_nt: 1
koi_fpflag_ss: 0
koi_fpflag_co: 0
koi_fpflag_ec: 0
koi_prad: 12
koi_period: 1
koi_duration: 149
koi_depth: 285
koi_srho: 86
koi_model_snr: 234
koi_impact: 31
koi_insol: 19
koi_teq: 137
koi_slogg: 218
koi_steff: 135
koi_srad: 46
koi_smass: 191
koi_ror: 31
koi_num_transits: 282
koi_tce_plnt_num: 175
koi_dor: 1
```

---

7. Exploratory Data Analysis

A few non-trivial exploratory data analysis (EDA) steps were performed on the data to obtain insights and better understand the patterns and relationships within the dataset. First, the describe() function was called to view descriptive statistics, which provided key summary metrics like mean, median, standard deviation, and percentiles for all numeric variables, offering a quick overview of the data's central tendencies and variability. The DataFrame was then grouped by the Exoplanet Archive category/disposition (koi_disposition), which would be the target variable for this classification task, to explore the distribution and characteristics of each exoplanet class (false positive, candidate, and confirmed) and observe how other variables behaved across these categories.

Next, histograms were plotted for all the variables to visualise their distributions. These plots revealed that none of the variables followed a normal distribution, although further statistical tests (such as the Shapiro-Wilk test) would be necessary to confirm the degree of non-normality. A heatmap was then generated to visualise the correlation matrix, showing the relationships between variables. This revealed several interesting positive and negative correlations, such as the strong correlation between orbital period (koi_period) and the planet-star distance over star radius (koi_dor). This is not surprising because a planet's orbital period tends to increase with its distance from the star due to the laws of planetary motion.

Additionally, a bivariate analysis was conducted by plotting the equilibrium temperature (koi_teq) against the disposition score (koi_score) using a scatterplot with the disposition as the hue. This plot revealed distinct clustering patterns: many false positive exoplanets were concentrated at the lower end of the disposition score, confirmed exoplanets were clustered at the higher end, and candidate exoplanets, which had not been definitively classified, were scattered across the full range of the disposition score. This suggests that while the disposition score is an important indicator for determining an exoplanet's classification, there is variability in how candidate exoplanets are distributed, which may require further analysis and additional features to refine their classification.

7.1 Descriptive Statistics

In [16]:
```
# Summary statistics of the dataset
pd_df2.describe()
```

Out[16]:

	koi_score	koi_fpflag_nt	koi_fpflag_ss	koi_fpflag_co	koi_fpflag_ec	koi_prad	koi_period	koi_duration	koi_depth	koi_
count	9564.000000	9564.000000	9564.000000	9564.000000	9564.000000	9564.000000	9564.000000	9564.000000	9.564000e+03	9564.00
mean	0.480829	0.208595	0.232748	0.197512	0.120033	102.891778	75.671358	5.621606	2.379134e+04	9.16
std	0.437658	4.767290	0.422605	0.398142	0.325018	3018.662296	1334.744046	6.471554	8.066667e+04	52.89

min	0.000000	0.000000	0.000000	0.000000	0.000000	0.080000	0.241843	0.052000	0.000000e+00	0.00
25%	0.000000	0.000000	0.000000	0.000000	0.000000	1.430000	2.733684	2.437750	1.668000e+02	0.24
50%	0.480829	0.000000	0.000000	0.000000	0.000000	2.490000	9.752831	3.792600	4.537000e+02	1.04
75%	0.995000	0.000000	0.000000	0.000000	0.000000	21.712500	40.715178	6.276500	2.125325e+03	3.32
max	1.000000	465.000000	1.000000	1.000000	1.000000	200346.000000	129995.778400	138.540000	1.541400e+06	980.85

8 rows × 22 columns

In [17]:
```
# Statistical summary by group
pd_df2.groupby('koi_disposition').mean()
```

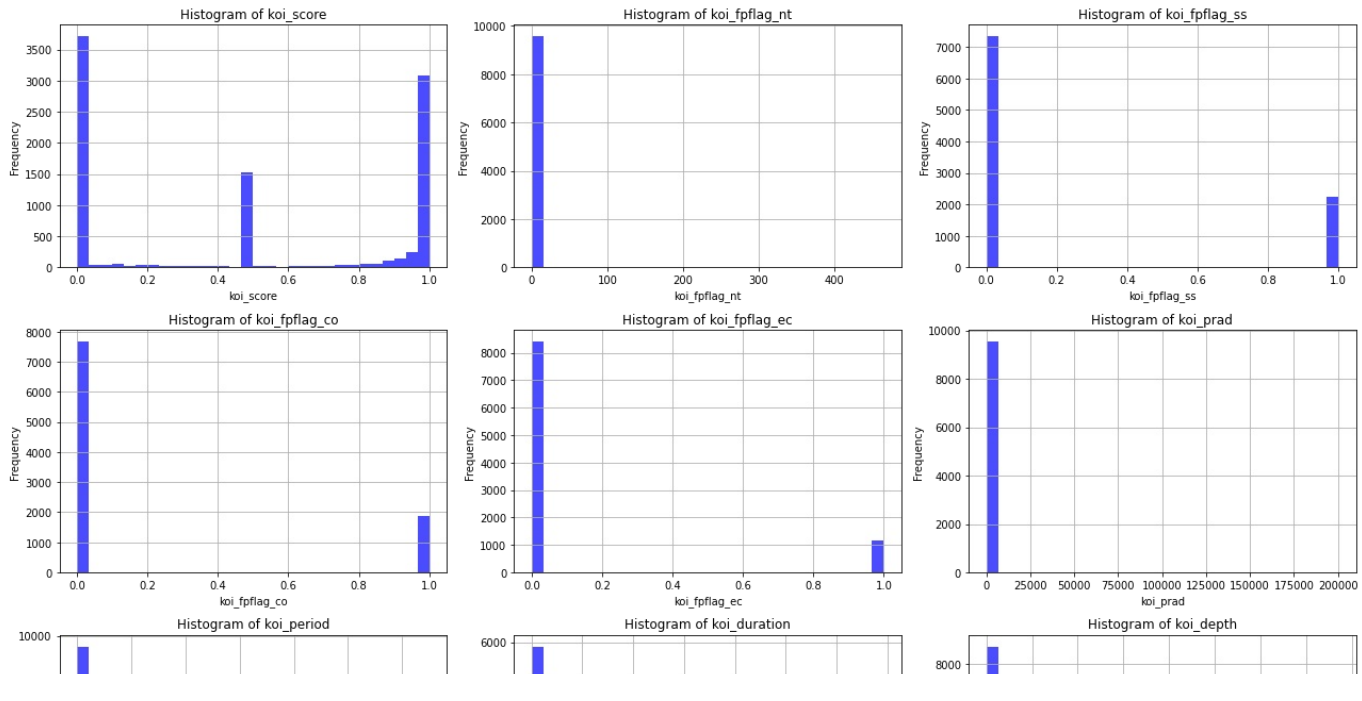
Out[17]:

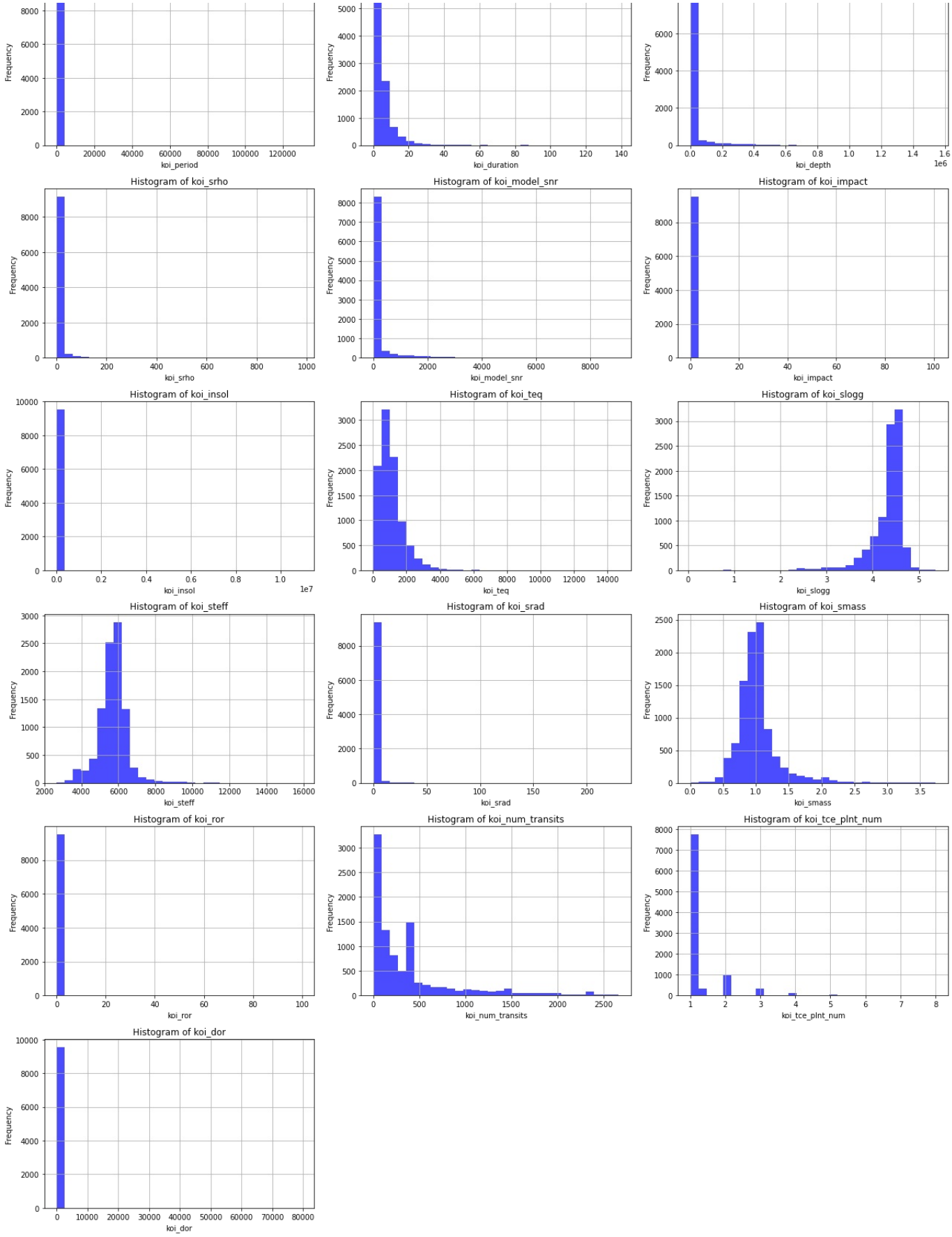
koi_disposition	koi_score	koi_fpflag_nt	koi_fpflag_ss	koi_fpflag_co	koi_fpflag_ec	koi_prad	koi_period	koi_duration	koi_depth	koi_sr
CANDIDATE	0.701644	0.000505	0.001009	0.00000	0.000000	98.023358	167.693452	5.283906	2547.757837	17.3567
CONFIRMED	0.958167	0.170616	0.005104	0.00000	0.000000	2.926447	27.759807	4.235488	1067.655002	2.7414
FALSE POSITIVE	0.119806	0.315354	0.456706	0.39037	0.237239	161.551442	65.138933	6.545650	45373.445511	9.4497

3 rows × 22 columns

7.2 Histogram

In [18]:
```
# Function to plot histograms for all numerical columns to view their distribution
def facet_histograms(df, columns_per_row=3):
    numeric_columns = df.select_dtypes(include=['float64', 'int64']).columns
    num_columns = len(numeric_columns)
    num_rows = math.ceil(num_columns / columns_per_row)  # Calculate the number of rows required

    # Create subplots
    fig, axes = plt.subplots(num_rows, columns_per_row, figsize=(columns_per_row * 6, num_rows * 4))
    axes = axes.flatten()  # Flatten the axes array for easier iteration

    # Loop through each column and plot
    for i, column in enumerate(numeric_columns):
        axes[i].hist(df[column].dropna(), bins=30, color='blue', alpha=0.7)
        axes[i].set_title(f"Histogram of {column}")
        axes[i].set_xlabel(column)
        axes[i].set_ylabel('Frequency')
        axes[i].grid(True)

    # Remove unused subplots
    for j in range(i + 1, len(axes)):
        fig.delaxes(axes[j])

    plt.tight_layout()  # Adjust layout to avoid overlap
    plt.show()

facet_histograms(pd_df2, columns_per_row=3)
```



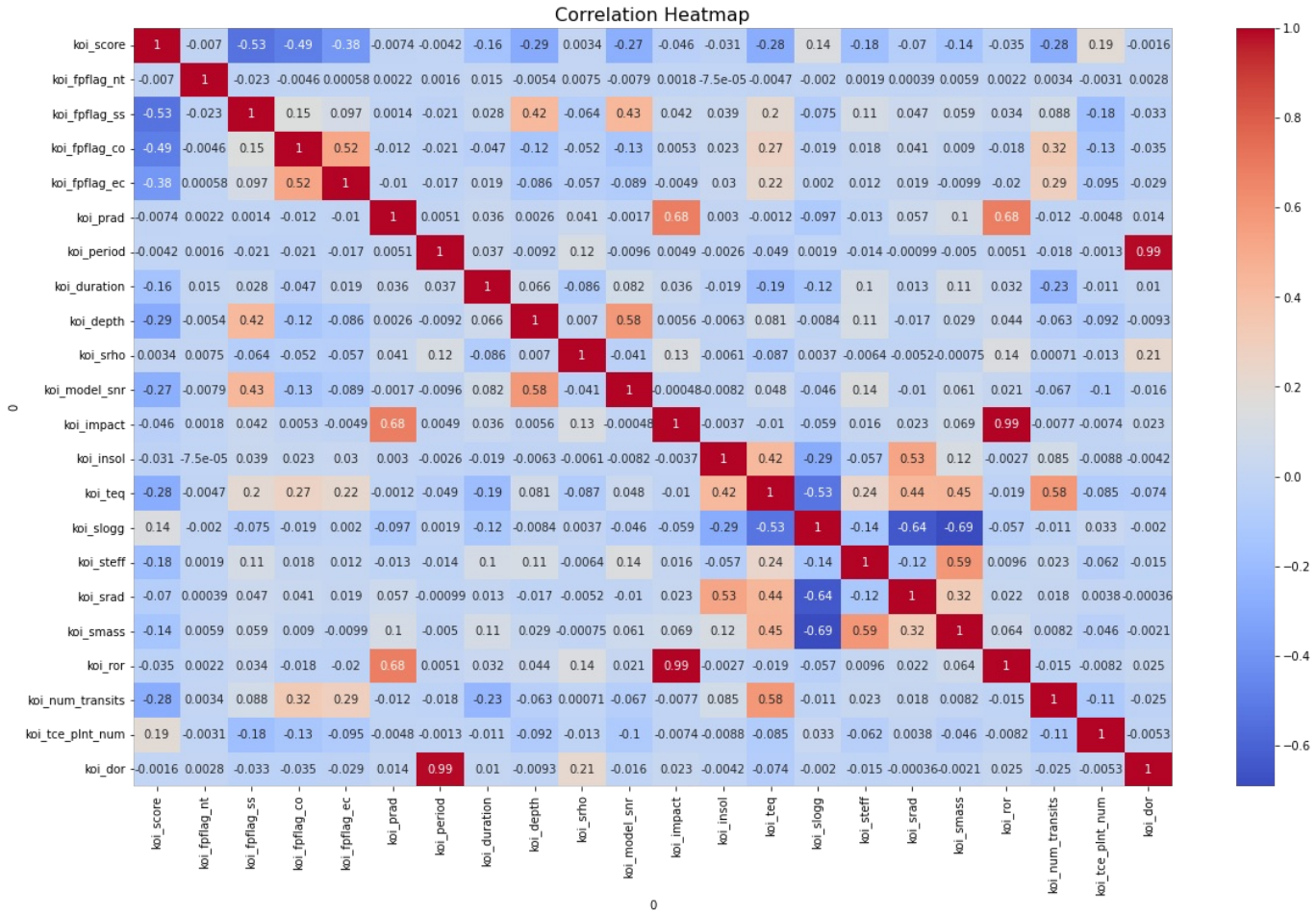


7.3 Correlation Analysis

In [19]:

```
# Compute the correlation matrix
corr_matrix = pd_df2.corr()

# Set the figure size before plotting the heatmap
plt.figure(figsize=(20, 12))  # You can adjust the width and height as needed

# Create the heatmap with annotations
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm')

# Add title and display the plot
```

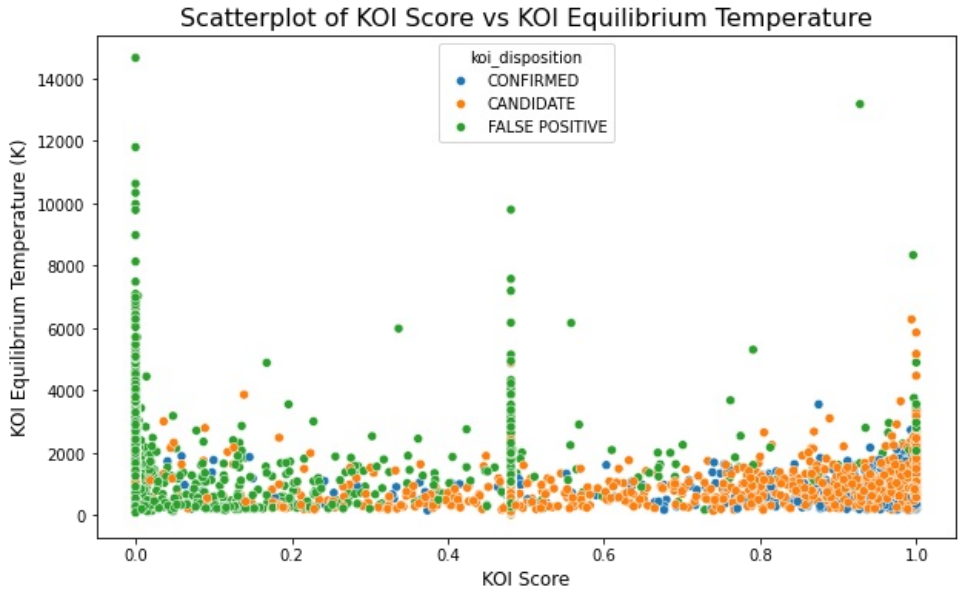
```
plt.title('Correlation Heatmap', fontsize=16)  # Adjust the font size if needed
plt.show()
```



7.4 Bivariate Analysis

In [20]:

```
# Create the scatterplot
plt.figure(figsize=(10, 6))  # Set the figure size
sns.scatterplot(data=pd_df2, x='koi_score', y='koi_teq', hue='koi_disposition') #, palette='viridis')

# Add titles and labels
plt.title('Scatterplot of KOI Score vs KOI Equilibrium Temperature', fontsize=16)
plt.xlabel('KOI Score', fontsize=12)
plt.ylabel('KOI Equilibrium Temperature (K)', fontsize=12)

# Show the plot
plt.show()
```



8. Statistical Tests

A few statistical tests were also performed to assess the distribution and relationships within the data, validate assumptions for modeling, and determine the presence of any significant patterns or correlations that could influence the accuracy of the machine learning models. A Shapiro-Wilk normality test was first performed and the results indicate that all the variables listed are not normally distributed. The test statistic for each variable is far from 1 (which would suggest normality), and the p-values are all 0.0, meaning that the null hypothesis of normality is rejected for every variable at any reasonable significance level. This suggests that none of the variables follow a normal distribution, which is important to consider when selecting appropriate statistical techniques and models, as many standard statistical methods assume normality. Alternative approaches, such as non-parametric methods or data transformations, might be needed for analysis.

Then a Chi-square test was performed and the results show a very large test statistic (10183.11) and a p-value of 0.0, indicating that there is a significant association between the categorical variable koi_score and koi_disposition. Since the p-value is 0.0, we reject the null hypothesis, which states that there is no relationship between these two variables. This means that koi_score and koi_disposition are not independent, and there is a strong relationship between them. This relationship suggests that the koi_score is likely a significant factor in determining or influencing the disposition (classification) of exoplanets.

An ANOVA test was conducted and it revealed that the variable koi_prad (planetary radius) across two categories of koi_disposition (CONFIRMED and FALSE POSITIVE) resulted in an F-statistic of 4.51 and a p-value of 0.0337. The p-value is less than the typical significance level of 0.05, which means we reject the null hypothesis that the mean planetary radius (koi_prad) is the same across the two disposition categories. This indicates that there is a statistically significant difference in the planetary radius between confirmed exoplanets and false positives. However, the effect size and practical significance should be further explored to understand the magnitude of this difference.

8.1 Normality Test (Shapiro Wilk)

In [23]:
```
# List of all variables to test for normality
all_vars = pd_df2.columns

# Prepare a list to hold the results
results = []

# Loop through each variable and perform the Shapiro-Wilk test if it's numerical
for var in all_vars:
    if pd.api.types.is_numeric_dtype(pd_df2[var]):
        # Drop missing values and perform the Shapiro-Wilk test
        stat, p_value = shapiro(pd_df2[var].dropna())

        # Append results to the list
        results.append({'Variable': var, 'Test Statistic': stat, 'p-value': p_value,
                        'Normality': 'Not Normally Distributed' if p_value < 0.05 else 'Normally Distributed'})

# Convert the results to a DataFrame
normality_results_df = pd.DataFrame(results)

# Display the table of results
print(normality_results_df)
```

```
            Variable  Test Statistic  p-value                Normality
0          koi_score        0.769627      0.0  Not Normally Distributed
1      koi_fpflag_nt        0.008506      0.0  Not Normally Distributed
2      koi_fpflag_ss        0.523219      0.0  Not Normally Distributed
3      koi_fpflag_co        0.487069      0.0  Not Normally Distributed
4      koi_fpflag_ec        0.378981      0.0  Not Normally Distributed
5           koi_prad        0.011047      0.0  Not Normally Distributed
6         koi_period        0.012209      0.0  Not Normally Distributed
7       koi_duration        0.560808      0.0  Not Normally Distributed
8          koi_depth        0.327407      0.0  Not Normally Distributed
9           koi_srho        0.136822      0.0  Not Normally Distributed
10     koi_model_snr        0.349436      0.0  Not Normally Distributed
11        koi_impact        0.076234      0.0  Not Normally Distributed
12         koi_insol        0.020295      0.0  Not Normally Distributed
13           koi_teq        0.758386      0.0  Not Normally Distributed
14         koi_slogg        0.701088      0.0  Not Normally Distributed
15         koi_steff        0.913332      0.0  Not Normally Distributed
16          koi_srad        0.105345      0.0  Not Normally Distributed
17         koi_smass        0.793182      0.0  Not Normally Distributed
18           koi_ror        0.042601      0.0  Not Normally Distributed
19  koi_num_transits        0.715726      0.0  Not Normally Distributed
20  koi_tce_plnt_num        0.429124      0.0  Not Normally Distributed
21           koi_dor        0.027605      0.0  Not Normally Distributed
```

```
/opt/jupyterhub/lib/python3.8/site-packages/scipy/stats/_morestats.py:1761: UserWarning: p-value may not be accur
ate for N > 5000.
  warnings.warn("p-value may not be accurate for N > 5000.")
```

8.2 Chi-Square Test

In [24]:
```
# Creating a contingency table for a categorical variable and koi_pdisposition
contingency_table = pd.crosstab(pd_df2['koi_score'], pd_df2['koi_disposition'])

# Perform the Chi-Square test
chi2, p, dof, expected = chi2_contingency(contingency_table)

print(f'Chi-Square Test Statistic: {chi2}')
print(f'p-value: {p}')
```

```
Chi-Square Test Statistic: 10183.110202926626
p-value: 0.0
```

8.3 ANOVA

In [25]:
```
# Perform ANOVA on koi_prad across two/ three categories of koi_pdisposition
anova_result = f_oneway(
    pd_df2[pd_df2['koi_disposition'] == 'CONFIRMED']['koi_prad'],
#     pd_df3[pd_df3['koi_disposition'] == 'CANDIDATE']['koi_prad'],
    pd_df2[pd_df2['koi_disposition'] == 'FALSE POSITIVE']['koi_prad']
)

print(f'ANOVA F-statistic: {anova_result.statistic}')
print(f'p-value: {anova_result.pvalue}')
```

```
ANOVA F-statistic: 4.508941537109499
p-value: 0.033750404387049564
```

---

9. Normalising, Oversampling and Feature Selection

To ensure that all features contributed equally to the analysis or machine learning model, especially when features have different scales or units, the dataset was normalised. Due to the nature of the data, which contained outliers, the dataset was normalised using the sklearn RobustScaler. This scaler is well-suited for datasets with outliers because it scales the data according to the interquartile range (IQR), reducing the influence of extreme values. Next, the balance between classes in the target variable (koi_disposition) was examined. This revealed a class imbalance, with the false positive class being the majority, and the confirmed and candidate classes being the minority. Since the candidate class represents exoplanets that have not yet been definitively classified as either confirmed exoplanets or false positives, it was removed from the training set. This data was kept aside for future predictions after the model is built. The remaining dataset contained only the confirmed and false positive classes, which would be used for binary classification.

To address the class imbalance between confirmed and false positive exoplanets, the confirmed class was oversampled using Manual Synthetic Sample Generation (Random Interpolation) since the imbalanced-learn (imblearn) library for SMOTE (Synthetic Minority Over-sampling Technique) could not be installed. This manual method is still effective because it creates synthetic data points by randomly interpolating between existing data points, helping balance the dataset without introducing noise or unrealistic data. Finally, feature selection was performed to identify the most important variables for classification. This revealed that the disposition score (koi_score) was the most predictive variable, with a 75.63% feature importance value. Features with importance values below 1% were removed because they contribute very little to the model's predictive power and can introduce noise, slowing down model performance. After this, the false positive values in the disposition variable (koi_disposition) were changed to 0, and the confirmed values were changed to 1, converting the problem into a binary classification task.

9.1 Normalising the data

In [26]:
```
# Using robust scaler since the data has outliers
features = pd_df2.drop(columns=['koi_disposition'])
target = pd_df2['koi_disposition']

# Initialize RobustScaler
scaler = RobustScaler()

# Fit and transform the features data
scaled_features = scaler.fit_transform(features)

# Create a new DataFrame with the scaled data
pd_df3 = pd.DataFrame(scaled_features, columns=features.columns)

# Add back the target column (which doesn't need to be normalized)
pd_df3['koi_disposition'] = target.values
```

```
# Display the first few rows of the scaled data
pd_df3.head()
```

Out[26]:

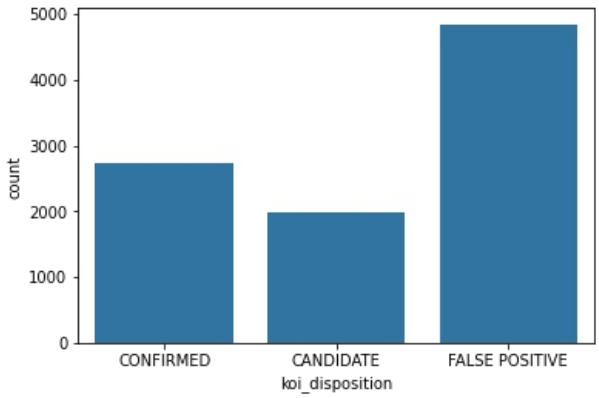
	koi_score	koi_fpflag_nt	koi_fpflag_ss	koi_fpflag_co	koi_fpflag_ec	koi_prad	koi_period	koi_duration	koi_depth	koi_srho	...	koi_teq
0	0.521779	0.0	0.0	0.0	0.0	-0.011340	-0.006972	-0.217545	0.082766	0.704161	...	-0.141338
1	0.490624	0.0	0.0	0.0	0.0	0.016763	1.175982	0.186102	0.215009	0.644285	...	-0.579112
2	-0.483246	0.0	0.0	0.0	0.0	0.597066	0.267138	-0.523712	5.297507	2.032300	...	-0.335210
3	-0.483246	0.0	1.0	0.0	0.0	1.526932	-0.211047	-0.361105	3.893491	-0.266426	...	0.611632
4	0.521779	0.0	0.0	0.0	0.0	0.012819	-0.190283	-0.556978	0.076384	0.307236	...	0.625391

5 rows × 23 columns

9.2 Oversampling

In [27]:
```
# Checking for the distribution of each class in the koi_disposition using a countplot
sns.countplot(x=target)
```

Out[27]:
```
<AxesSubplot:xlabel='koi_disposition', ylabel='count'>
```



In [28]:
```
# Subsetting the data, so the candidate class is kept for prediction
# Subset for 'confirmed' and 'false positive' classes
df_cf_fp = pd_df3[pd_df3['koi_disposition'].isin(['CONFIRMED', 'FALSE POSITIVE'])]

# Subset for the 'candidate' class
df_cd = pd_df3[pd_df3['koi_disposition'] == 'CANDIDATE']

# Check the number of rows in each subset
print(f"Confirmed and False Positive: {df_cf_fp.shape[0]} rows")
print(f"Candidate: {df_cd.shape[0]} rows")
```

```
Confirmed and False Positive: 7582 rows
Candidate: 1982 rows
```

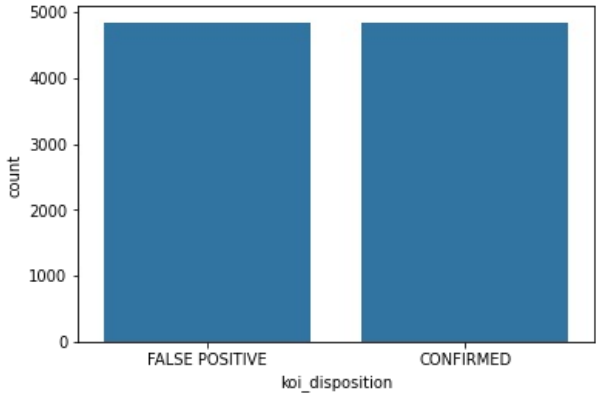
In [29]:
```
# Oversampling the 'confirmed' class using Manual Synthetic Sample Generation (Random Interpolation) to balance t
# Separate the majority and minority classes
df_majority = df_cf_fp[df_cf_fp['koi_disposition'] == 'FALSE POSITIVE']
df_minority = df_cf_fp[df_cf_fp['koi_disposition'] == 'CONFIRMED']

# Number of samples to generate
n_samples = len(df_majority) - len(df_minority)

# Select only numeric columns for interpolation
numeric_columns = df_minority.select_dtypes(include=[np.number]).columns

# Generate synthetic samples
synthetic_samples = []

for _ in range(n_samples):
    # Randomly select two samples from the minority class, using only numeric columns
    sample1, sample2 = df_minority.sample(2, random_state=np.random.randint(1000))[numeric_columns].values
    # Linearly interpolate between the two samples
    new_sample = sample1 + np.random.rand() * (sample2 - sample1)
    synthetic_samples.append(new_sample)

# Convert the synthetic samples to a DataFrame with the same numeric columns
synthetic_df_numeric = pd.DataFrame(synthetic_samples, columns=numeric_columns)
```

```
# For non-numeric columns, just duplicate the values randomly from the minority class
non_numeric_columns = df_minority.select_dtypes(exclude=[np.number]).columns
synthetic_df_non_numeric = df_minority[non_numeric_columns].sample(n=n_samples, replace=True, random_state=42).re

# Combine the numeric and non-numeric synthetic data
synthetic_df = pd.concat([synthetic_df_numeric, synthetic_df_non_numeric], axis=1)

# Combine the original data with the synthetic samples
df_oversampled = pd.concat([df_majority, df_minority, synthetic_df])

# Check the new class distribution
print(df_oversampled['koi_disposition'].value_counts())
```

```
FALSE POSITIVE    4839
CONFIRMED         4839
Name: koi_disposition, dtype: int64
```

In [30]:
```
# Sanity check that the classes are balanced
# Update the features and target
X = df_oversampled.drop(columns=['koi_disposition'])
y = df_oversampled['koi_disposition']

# Show the countplot
sns.countplot(x=y)
```

Out[30]: <AxesSubplot:xlabel='koi_disposition', ylabel='count'>

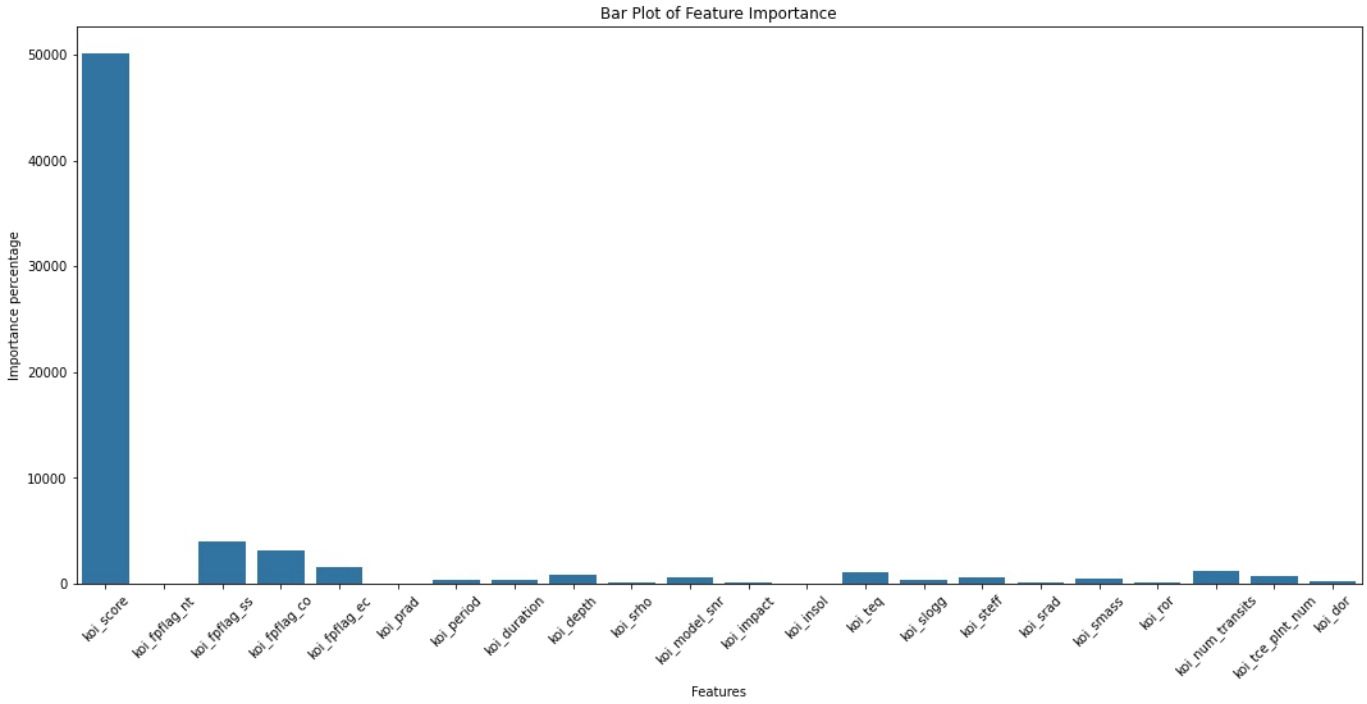


9.3 Feature Selection

In [31]:
```
# Get the feature importance using sklearn's SelectKBest
# Fit SelectKBest with f_classif
fs = SelectKBest(score_func=f_classif, k='all')
fs.fit(X, y)
SelectKBest(k='all')

# Calculate the percentage contribution of each feature
feature_contribution=(fs.scores_/sum(fs.scores_))*100
for i,j in enumerate(X.columns):
    print(f'{j} : {feature_contribution[i]:.2f}%')
plt.figure(figsize=(18,8))
plt.xticks(rotation=45)
sns.barplot(x=X.columns,y=fs.scores_)
plt.xlabel('Features')
plt.ylabel('Importance percentage')
plt.title('Bar Plot of Feature Importance')
plt.show()
```

```
koi_score : 76.39%
koi_fpflag_nt : 0.01%
koi_fpflag_ss : 6.00%
koi_fpflag_co : 4.72%
koi_fpflag_ec : 2.29%
koi_prad : 0.01%
koi_period : 0.49%
koi_duration : 0.52%
koi_depth : 1.22%
koi_srho : 0.10%
koi_model_snr : 0.89%
koi_impact : 0.12%
koi_insol : 0.03%
koi_teq : 1.59%
koi_slogg : 0.63%
koi_steff : 0.94%
koi_srad : 0.14%
```

```
koi_smass : 0.69%
koi_ror : 0.09%
koi_num_transits : 1.83%
koi_tce_plnt_num : 1.03%
koi_dor : 0.29%
```



In [32]:
```python
# Removing features that have an importance < 1%
# Create a dictionary to map feature names to their contribution
feature_importance = {X.columns[i]: feature_contribution[i] for i in range(len(X.columns))}

# Set cutoff threshold
cutoff = 1.0

# Filter features based on cutoff
important_features = [feature for feature, importance in feature_importance.items() if importance >= cutoff]
print(f'Selected features: {important_features}')

# Subset the original data to include only the selected features
X_selected = X[important_features]
```

```
Selected features: ['koi_score', 'koi_fpflag_ss', 'koi_fpflag_co', 'koi_fpflag_ec', 'koi_depth', 'koi_teq', 'koi_
num_transits', 'koi_tce_plnt_num']
```

In [33]:
```python
# Join the selected features (X_selected) with the target (y)
df_final = pd.concat([X_selected, y], axis=1)

# Reset the index
df_final = df_final.reset_index(drop=True)
```

In [34]:
```python
# Replace 'FALSE POSITIVE' with 0 and 'CONFIRMED' with 1 in the 'koi_disposition' column
df_final['koi_disposition'] = df_final['koi_disposition'].replace({'FALSE POSITIVE': 0, 'CONFIRMED': 1})

# Display the updated DataFrame
# df_final.head()
```

In [35]:
```python
df_final.head()
```

Out[35]:

	koi_score	koi_fpflag_ss	koi_fpflag_co	koi_fpflag_ec	koi_depth	koi_teq	koi_num_transits	koi_tce_plnt_num	koi_disposition
0	-0.483246	1.0	0.0	0.0	3.893491	0.611632	1.282256	0.0	0
1	-0.483246	1.0	1.0	0.0	-0.112329	0.545341	-0.027026	0.0	0
2	-0.483246	1.0	0.0	0.0	8.950767	-0.191370	-0.294288	0.0	0
3	-0.483246	1.0	0.0	0.0	4.322130	-0.479049	-0.369362	0.0	0
4	-0.483246	1.0	0.0	0.0	37.696889	-0.260163	-0.417409	0.0	0

---

10. Machine Learning Model

10. Machine Learning Models

Prior to building the machine learning models using various algorithms, the Pandas DataFrame was converted to a Spark DataFrame to leverage the distributed computing capabilities of Spark. An attempt was initially made to use a Spark Resilient Distributed Dataset (RDD), but this proved somewhat difficult to work with, possibly due to RDDs being lower-level abstractions that require more manual handling for data transformations. Since the dataset is manageable in size and structure, a Spark DataFrame was deemed sufficient for building machine learning models without any performance issues. The data was then split into training and test sets to ensure that model evaluation could be performed on unseen data. A Spark vector assembler was created to combine the relevant features into a single vector, which is necessary for Spark's machine learning algorithms, as they expect input features to be in vectorised form.

To ensure the code was modular and reusable, functions were created for key tasks such as model evaluation and performance visualisation. These functions included generating and plotting the confusion matrix to assess classification accuracy and plotting the ROC (Receiver Operating Characteristic) curve to evaluate the model's ability to distinguish between classes. This modular approach not only streamlined the machine learning workflow but also made it easier to test and compare different models efficiently.

Finally, a variety of machine learning models were trained to evaluate different types of algorithms and their performance on the exoplanet classification task. These algorithms were chosen based on their distinct characteristics and learning paradigms such as statistical models, linear models, tree-based models, kernel-based models and neural networks. Naive Bayes was selected as a representative of statistical models, particularly well-suited for handling probabilistic classification tasks. Logistic regression was chosen as a linear model, known for its simplicity and effectiveness in binary classification.

A random forest model was used to represent tree-based models, which excel at handling complex datasets and capturing non-linear relationships through ensemble learning. The support vector machine (SVM) was chosen for its kernel-based approach, which is effective in finding optimal decision boundaries, especially when the classes are not linearly separable. Lastly, a multilayer perceptron (MLP) was included as a neural network model, capable of learning intricate patterns and capturing non-linear relationships through its multi-layer structure.

By training these diverse models, a comprehensive understanding of how different algorithms perform on this classification problem was obtained, allowing for comparison and selection of the best-performing model based on metrics such as accuracy, precision, recall, F1 score, and AUC. This approach ensures that the strengths of different model types are fully leveraged for the exoplanet classification task.

10.1 Creating an Pyspark DF

In [36]:
```
# Currently, data is a pandas dataframe, not a spark dataframe
# Create a pyspark dataframe from the dataframe (using the Topic 9 programming exercise as a guide)
conf = SparkConf().setAppName('test')
sc = SparkContext.getOrCreate(conf=conf)
sq = SQLContext(sc)

# Conversion
spark_data = sq.createDataFrame(df_final)
```

```
/opt/spark/3.5/python/lib/pyspark.zip/pyspark/sql/context.py:113: FutureWarning: Deprecated in 3.0.0. Use SparkSe
ssion.builder.getOrCreate() instead.
```

In [37]:
```
spark_data.describe()
```

Out[37]:
```
DataFrame[summary: string, koi_score: string, koi_fpflag_ss: string, koi_fpflag_co: string, koi_fpflag_ec: string
, koi_depth: string, koi_teq: string, koi_num_transits: string, koi_tce_plnt_num: string, koi_disposition: string
]
```

10.2 Splitting the Data

In [38]:
```
# Weights for the split
training_ratio = 0.8
testing_ratio = 0.2

# Split the RDD
train_df, test_df = spark_data.randomSplit([training_ratio, testing_ratio], seed=42)
```

10.3 Creating a Spark Vector

In [39]:
```
# Creating a spark vector using the 'important features' variable previously defined
assembler = VectorAssembler(inputCols=important_features, outputCol="features")
```

10.4 Relevant Functions

10.4.1 Model Evaluation Function

In [40]:
```
def evaluate_classification_model(predictions, label_col, pred_col):
    """
    Evaluates the classification model using accuracy, precision, recall, and F1 score.

    Parameters:
    predictions (DataFrame): Spark DataFrame containing the true and predicted labels.
    label_col (str): The name of the column containing the true labels.
    pred_col (str): The name of the column containing the predicted labels.

    Returns:
    dict: A dictionary containing the calculated metrics (accuracy, precision, recall, and F1 score).
    """
    metrics = {}

    # Accuracy
    accuracy_evaluator = MulticlassClassificationEvaluator(labelCol=label_col, predictionCol=pred_col, metricName
    metrics['accuracy'] = accuracy_evaluator.evaluate(predictions)

    # Precision (weighted)
    precision_evaluator = MulticlassClassificationEvaluator(labelCol=label_col, predictionCol=pred_col, metricNam
    metrics['precision'] = precision_evaluator.evaluate(predictions)

    # Recall (weighted)
    recall_evaluator = MulticlassClassificationEvaluator(labelCol=label_col, predictionCol=pred_col, metricName='
    metrics['recall'] = recall_evaluator.evaluate(predictions)

    # F1 Score (weighted)
    f1_evaluator = MulticlassClassificationEvaluator(labelCol=label_col, predictionCol=pred_col, metricName="f1")
    metrics['f1_score'] = f1_evaluator.evaluate(predictions)

    # Print the metrics
    print(f"Test Accuracy: {metrics['accuracy']}")
    print(f"Test Precision: {metrics['precision']}")
    print(f"Test Recall: {metrics['recall']}")
    print(f"Test F1 Score: {metrics['f1_score']}")

    return metrics
```

10.4.2 Confusion Matrix Function

In [41]:
```
def plot_confusion_matrix(model_predictions, label_col, pred_col, labels=["False Positive", "Confirmed"]):
    """
    Plots the confusion matrix for the given model predictions.

    Parameters:
    model_predictions (DataFrame): Spark DataFrame containing the true and predicted labels.
    label_col (str): The name of the column containing the true labels.
    pred_col (str): The name of the column containing the predicted labels.
    labels (list): The list of class labels to use in the plot.

    Returns:
    None
    """
    # Collect the true labels and predicted labels into Pandas DataFrames
    preds_and_labels = model_predictions.select(pred_col, label_col).toPandas()

    # Extract the actual labels and predictions
    y_true = preds_and_labels[label_col]
    y_pred = preds_and_labels[pred_col]

    # Compute the confusion matrix
    cm = confusion_matrix(y_true, y_pred)

    # Create a heatmap for the confusion matrix using Seaborn
    plt.figure(figsize=(8, 6))
    sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=labels, yticklabels=labels)
    plt.xlabel('Predicted Label')
    plt.ylabel('True Label')
    plt.title('Confusion Matrix')
    plt.show()
```

10.4.3 ROC Curve Function

In [48]:
```
def plot_roc_curve(predictions, label_col, raw_pred_col):
    """
    Plots the ROC curve for the given SVM model predictions.

    Parameters:
    svm_predictions (DataFrame): Spark DataFrame containing the true labels and raw prediction scores.
    label_col (str): The name of the column containing the true labels.
    raw_pred_col (str): The name of the column containing the raw prediction scores.

    Returns:
```

```
    None
    """
    # Convert the true labels and decision scores into Pandas DataFrame
    y_true = predictions.select(label_col).toPandas().values
    decision_scores = predictions.select(raw_pred_col).toPandas()[raw_pred_col].apply(lambda x: x[1])

    # Compute ROC curve and ROC area
    fpr, tpr, _ = roc_curve(y_true, decision_scores)
    roc_auc = auc(fpr, tpr)

    # Plot ROC curve
    plt.figure()
    plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' % roc_auc)
    plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver Operating Characteristic')
    plt.legend(loc="lower right")
    plt.show()
```

10.5 Naive Bayes Model (Benchmark)

The Naive Bayes Model, which is a probabilistic classifier based on Bayes' Theorem and assumes independence between predictors, was chosen as the benchmark model. This is because Naive Bayes is computationally efficient and often performs well as a baseline model, especially for classification tasks with categorical or binary target variables. First, the training and test data had to be transformed so that all feature values were non-negative. This adjustment was necessary because the Naive Bayes algorithm in Spark cannot handle negative feature values, as it relies on probabilistic calculations that assume non-negative inputs. Using the NaiveBayes function and the vector assembler , a Naive Bayes pipeline was created. The model was then trained (fit) on the absolute value-transformed training dataset, and predictions were made on the transformed test set.

The model's performance was evaluated using several key metrics: accuracy, precision, recall, and F1 score. The results revealed an accuracy of 80%, a precision of 83%, a recall of 80%, and an F1 score of 80%. These metrics indicate that the model performs reasonably well, with precision showing a slightly better ability to correctly identify true positives (confirmed exoplanets) than recall, which measures the model's ability to identify all true positives. The ROC curve was also plotted, and the area under the ROC curve (AUC) was found to be 0.91. This indicates that the Naive Bayes model has a strong ability to distinguish between the two classes (confirmed exoplanets and false positives), with a high level of discriminative power. Overall, the model's performance, particularly the AUC, suggests it provides a solid starting point for further optimsation and comparison with more complex models.

10.5.1 Modifying the dataframe

In [43]:
```
# Make PySpark DataFrame absolute columns so there are no negative values
train_df_abs = train_df.select([F.abs(F.col(c)).alias(c) for c in train_df.columns])

test_df_abs = test_df.select([F.abs(F.col(c)).alias(c) for c in test_df.columns])

# Show the result
# train_df_abs.show()
# test_df_abs.show()
```

10.5.2 Model Fitting

In [44]:
```
# Create the NaiveBayes model
nb = NaiveBayes(featuresCol="features", labelCol="koi_disposition", predictionCol="predicted_label", modelType='m

# Create a Pipeline with the VectorAssembler and NaiveBayes classifier
nb_pipeline = Pipeline(stages=[assembler, nb])

# Train the Naive Bayes model
nb_model = nb_pipeline.fit(train_df_abs)

# Make predictions using the test set
nb_predictions = nb_model.transform(test_df_abs)
```

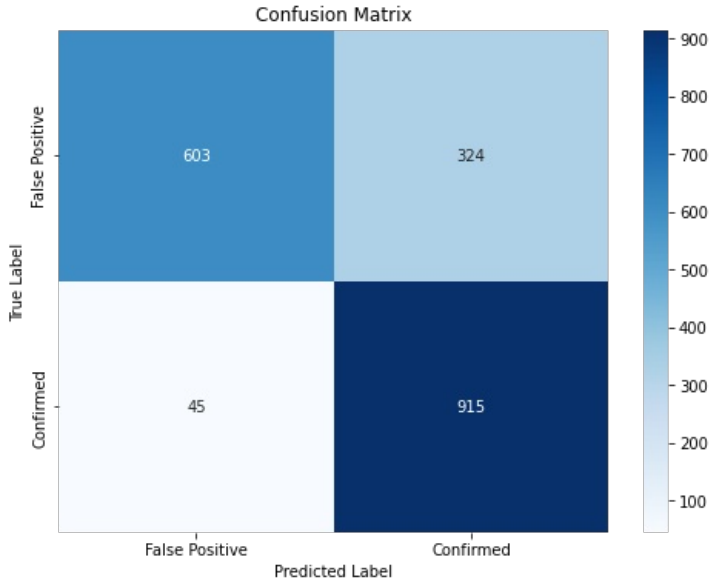
10.5.3 Model Evaluation

In [45]:
```
# Evaluate the Naive Bayes model
nb_metrics = evaluate_classification_model(nb_predictions, label_col="koi_disposition", pred_col="predicted_label
```

```
Test Accuracy: 0.8044515103338633
Test Precision: 0.832847820759087
Test Recall: 0.8044515103338633
Test F1 Score: 0.7995366919305609
```

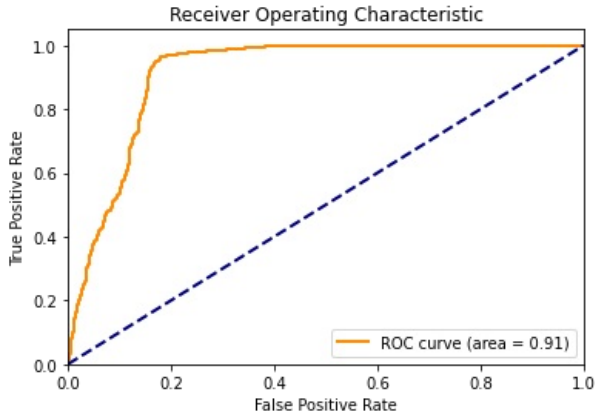
10.5.4 Confusion Matrix

In [46]:
```
# Confuson matrix for Naive Bayes model
plot_confusion_matrix(nb_predictions, label_col="koi_disposition", pred_col="predicted_label")
```



10.5.5 ROC Curve

In [49]:
```
# Plot roc curve
plot_roc_curve(nb_predictions, label_col="koi_disposition", raw_pred_col="rawPrediction")
```



10.6 Logistic Regression Model

Similar to the Naive Bayes model, a logistic regression model was fit to the original training data using the vector assembler and the logistic regression function within a pipeline. Note that, unlike the Naive Bayes model, this data did not have negative values removed, as logistic regression can handle both positive and negative feature values. Although the primary focus was on the performance metrics of the models, additional analyses were performed to gain insight into the logistic regression model, such as viewing the coefficients, intercepts, objective history, and the ROC training summary.

Since this is a binary classification problem, the logistic regression model's family was set to "multinomial," allowing for the calculation of multinomial coefficients for each feature across the two outcome classes (confirmed exoplanet vs. false positive). The coefficients indicate how each feature contributes to the probability of each class, with some features having stronger effects than others. The intercepts represent the baseline probabilities for the classes, slightly favoring the first class (false positives) in this case.

Additionally, the objective history, which tracks the optimisation process during training, was printed, showing the objective value per iteration. This confirmed that the model successfully learned from the data, as the objective converged, meaning further training would not have resulted in meaningful improvements. A DataFrame of the receiver-operating characteristic (ROC) curve was also obtained, providing a deeper look into the model's ability to distinguish between the two classes.

These additional analyses were performed only for the logistic regression model and not for other models, as they provide specific insights into the workings of the logistic regression algorithm, such as how features influence predictions and the optimisation process over time.

The logistic regression model performed exceptionally well and significantly outperformed the Naive Bayes model. It achieved an accuracy,

precision, recall, and F1 score of approximately 98.4%, reflecting its strong predictive power. The ROC curve was also plotted, with the area under the curve (AUC) reaching 1. This perfect AUC value suggests that the model was able to perfectly distinguish between the two classes, making it an excellent choice for this binary classification task.

10.6.1 Model Fitting

In [67]:
```
# Fit a logistic regression to the model
# Using the multinomial family since it is a binary classification task
lr = LogisticRegression(featuresCol="features", labelCol="koi_disposition", predictionCol="predicted_label", fami

# Create the Pipeline with the VectorAssembler and LogisticRegression stages
lrPipe = Pipeline(stages=[assembler, lr])

# Fit the Pipeline to the DataFrame
lrModel = lrPipe.fit(train_df)

# Make predictions using the held-out test data
lr_predictions = lrModel.transform(test_df)

# Since lrModel is a PipelineModel, you need to extract the LogisticRegressionModel
log_reg_model = lrModel.stages[-1]  # Assuming LogisticRegression is the last stage in the pipeline

# Print the coefficients and intercepts for logistic regression with multinomial family
print("Multinomial coefficients: " + str(log_reg_model.coefficientMatrix))
print("Multinomial intercepts: " + str(log_reg_model.interceptVector))

# Extract the summary from the returned Logistic Regression model instance trained
training_summary = log_reg_model.summary

# Obtain the objective per iteration
obj_history = training_summary.objectiveHistory
print("Objective history:", obj_history)
# for objective in obj_history:
#     print(objective)
```

```
Multinomial coefficients: DenseMatrix([[ -4.99051636,   2.51034482,  10.39097109,   5.35787606,
                0.06808114,  -0.09925888,   0.3064918 ,  -0.27336181],
             [  4.99051636,  -2.51034482, -10.39097109,  -5.35787606,
               -0.06808114,   0.09925888,  -0.3064918 ,   0.27336181]])
Multinomial intercepts: [0.27649295063628987,-0.27649295063628987]
Objective history: [0.6931382101405943, 0.2374391474744066, 0.1976610634757886, 0.13090051882480552, 0.1082436893
6979967, 0.09568770825183577, 0.08894955334129269, 0.08574239019128671, 0.08458947094085252, 0.08409827246270794,
0.08368437348395044, 0.0829793515990978, 0.0827020445086764, 0.08193085943016241, 0.08123376372884049, 0.08073680
288109303, 0.08061210792842484, 0.08028633637948185, 0.08027410029515557, 0.08027130854195066, 0.0802711519057352
6, 0.08027113408099647, 0.08027113080488764, 0.0802711305448355, 0.08027113028537762, 0.08027113027900178, 0.0802
7113027485085, 0.08027113027076344, 0.08027113026414764, 0.08027113024752598, 0.08027113021037859, 0.080271130117
99958, 0.08027112992372791, 0.0802711297846216, 0.08027112940670827, 0.0802711285988861, 0.08027112766841013, 0.0
8027112620929823, 0.08027112333469458]
```

10.6.2 Training ROC

In [68]:
```
# Obtain the receiver-operating characteristic as a dataframe and area under ROC
training_summary.roc.show()
print("areaUnderROC " + str(training_summary.areaUnderROC))
```

```
+--------------------+--------------------+
|                 FPR|                 TPR|
+--------------------+--------------------+
|                 0.0|                 0.0|
|2.556237218813906E-4|0.001546790409899...|
|2.556237218813906E-4|0.003351379221448827|
|2.556237218813906E-4|0.005155968032998196|
|2.556237218813906E-4|0.006960556844547...|
|2.556237218813906E-4|0.008765145656096932|
|2.556237218813906E-4|0.010569734467646301|
|2.556237218813906E-4| 0.01237432327919567|
|2.556237218813906E-4|0.014178912090745037|
|2.556237218813906E-4|0.015983500902294407|
|2.556237218813906E-4|0.017788089713843776|
|2.556237218813906E-4| 0.01959267852539314|
|2.556237218813906E-4| 0.02139726733694251|
|2.556237218813906E-4| 0.02320185614849188|
|2.556237218813906E-4|0.025006444960041247|
|2.556237218813906E-4|0.026811033771590616|
|2.556237218813906E-4|0.028615622583139984|
|2.556237218813906E-4|0.030420211394689353|
|2.556237218813906E-4| 0.03222480020623872|
|2.556237218813906E-4| 0.03402938901778809|
+--------------------+--------------------+
only showing top 20 rows
```

```
areaUnderROC 0.9953147512878058
```

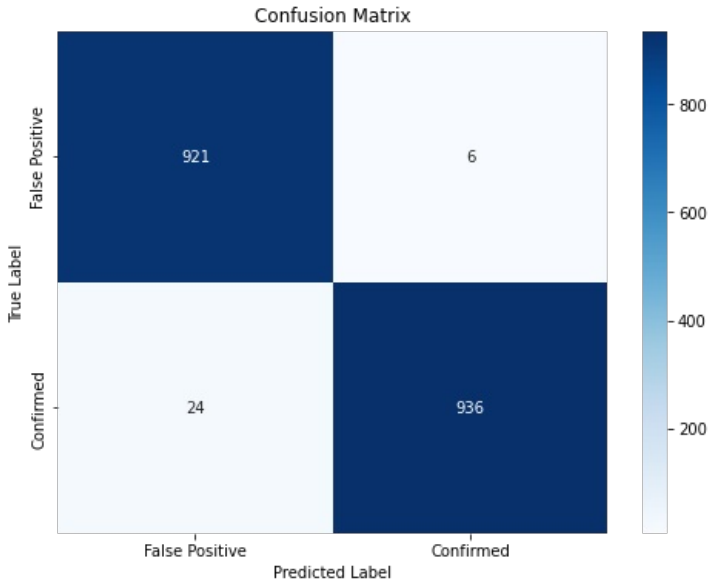
10.6.3 Model Evaluation

In [69]:
```
# Calling the function for the logistic regression evaluation
lr_metrics = evaluate_classification_model(lr_predictions, label_col="koi_disposition", pred_col="predicted_label
```

```
Test Accuracy: 0.9841017488076311
Test Precision: 0.9842832502255506
Test Recall: 0.9841017488076311
Test F1 Score: 0.9841029543896496
```

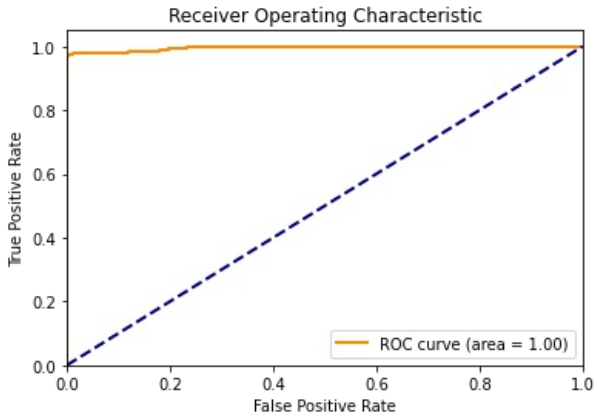
10.6.4 Confusion matrix

In [70]:
```
# Confusion matrix for Logistic Regression
plot_confusion_matrix(lr_predictions, label_col="koi_disposition", pred_col="predicted_label")
```



10.6.5 ROC Curve

In [71]:
```
# Plot roc curve
plot_roc_curve(lr_predictions, label_col="koi_disposition", raw_pred_col="rawPrediction")
```



10.7 Random Forest

A Random Forest model was also trained on the dataset. Similar to the logistic regression model, the Random Forest performed exceptionally well, achieving an accuracy, precision, recall, and F1 score of approximately 98.8%. This strong performance could be attributed to the model's ability to handle complex, non-linear relationships and its robustness to overfitting due to the ensemble nature of decision trees. This model has the best performance so far.

The ROC curve was also plotted, and the area under the curve (AUC) was 1.0, indicating perfect model performance. An AUC of 1.0 means that the model was able to perfectly distinguish between the two classes (false positives and confirmed exoplanets) across all thresholds. This suggests that the Random Forest model is highly effective at classifying exoplanet data with very few, if any, misclassifications, making it a reliable model for this task.

10.7.1 Model Fitting

In [62]:
```
# Create random forest classifier
rf_model = RandomForestClassifier(featuresCol="features", labelCol="koi_disposition", predictionCol="predicted_la

# Create a Pipeline with the VectorAssembler and RandomForestClassifier
rf_pipeline = Pipeline(stages=[assembler, rf_model])

# Train the Random Forest model
rf_model = rf_pipeline.fit(train_df)

# Make predictions using the test set
rf_predictions = rf_model.transform(test_df)
```

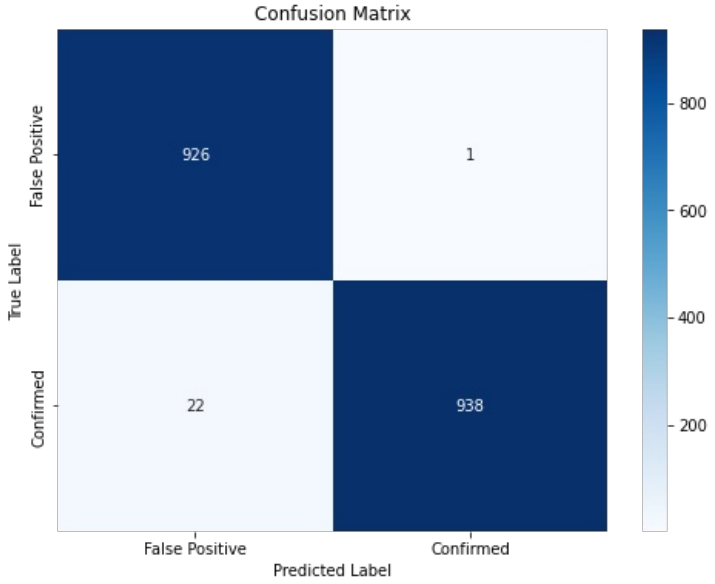
10.7.2 Model Evaluation

In [63]:
```
# Evaluating the Random Forest model
rf_metrics = evaluate_classification_model(rf_predictions, label_col="koi_disposition", pred_col="predicted_label
```

```
Test Accuracy: 0.9878113407525172
Test Precision: 0.9880577517514041
Test Recall: 0.9878113407525172
Test F1 Score: 0.9878122033938036
```

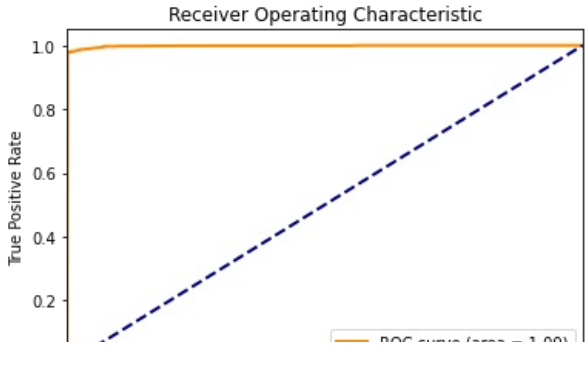
10.7.3 Confusion Matrix

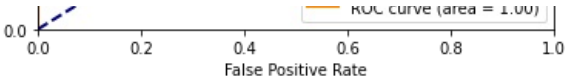
In [64]:
```
# Confusion matrix for Random Forest
plot_confusion_matrix(rf_predictions, label_col="koi_disposition", pred_col="predicted_label")
```



10.7.4 ROC Curve

In [72]:
```
# Plot roc curve
plot_roc_curve(rf_predictions, label_col="koi_disposition", raw_pred_col="rawPrediction")
```





10.7.5 Random Forest Model Structure

In [ ]:
```
# Uncomment if you would like to view the structure of the random forest model

# Extract the RandomForest model from the pipeline stages
# rf_model_extract = rf_model.stages[1]  # The RandomForestClassifier is the second stage in the pipeline

# # Access individual trees from the RandomForest model
# for idx, tree in enumerate(rf_model_extract.trees):
#     print(f"Tree {idx} structure:\n")
#     print(tree.toDebugString)
#     print("="*60)
```

10.8 SVM Model

A Support Vector Machine (SVM), a kernel-based model, was built for this classification task to evaluate its ability to create optimal decision boundaries between classes, especially when the data is not linearly separable. SVMs are known for their effectiveness in high-dimensional spaces and their flexibility, thanks to the use of different kernel functions such as linear, polynomial, or radial basis function (RBF) kernels, which can capture complex relationships in the data.

The model performed well, achieving an accuracy, precision, recall, and F1 score of approximately 97.7%, demonstrating that the SVM was able to classify exoplanet candidates with high accuracy. As with the other models, the ROC curve was plotted, and the area under the curve (AUC) was 0.99. This high AUC value indicates that the SVM model is highly capable of distinguishing between the two classes (false positives and confirmed exoplanets), reflecting its strong classification performance in this task.

10.8.1 Model Fitting

In [78]:
```
# Create the LinearSVC model
svm = LinearSVC(featuresCol="features", labelCol="koi_disposition", predictionCol="predicted_label", maxIter=10)

# Create a Pipeline with the VectorAssembler and LinearSVC classifier
svm_pipeline = Pipeline(stages=[assembler, svm])

# Train the SVM model
svm_model = svm_pipeline.fit(train_df)

# Make predictions using the test set
svm_predictions = svm_model.transform(test_df)
```

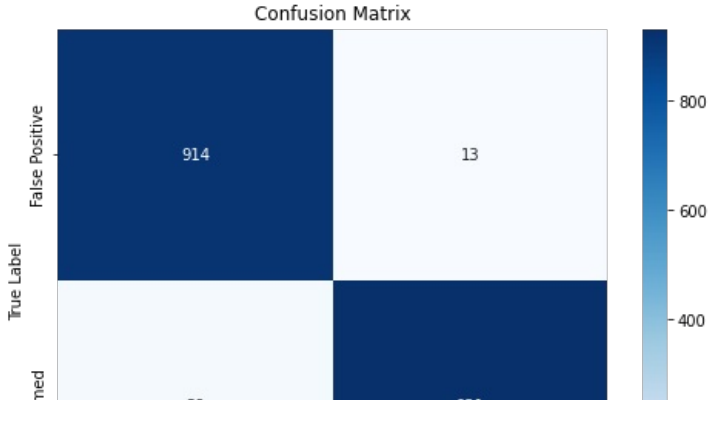
10.8.2 Model Evaluation

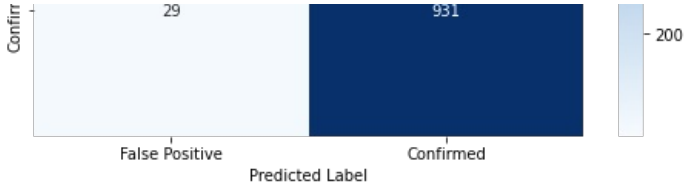
In [79]:
```
# Evaluating the Random Forest model
svm_metrics = evaluate_classification_model(svm_predictions, label_col="koi_disposition", pred_col="predicted_lab
```

```
Test Accuracy: 0.9777424483306836
Test Precision: 0.9778864375604378
Test Recall: 0.9777424483306836
Test F1 Score: 0.9777441486784046
```

10.8.3 Confusion Matrix

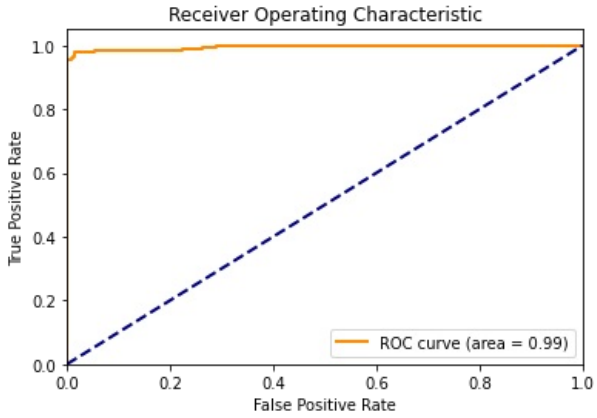
In [80]:
```
# Confusion matrix for the SVM model
plot_confusion_matrix(svm_predictions, label_col="koi_disposition", pred_col="predicted_label")
```





10.8.4 ROC Curve

In [81]:
```
# Plot svm roc curve
plot_roc_curve(svm_predictions, label_col="koi_disposition", raw_pred_col="rawPrediction")
```



10.9 Neural Network: Multilayer Perceptron

A Multilayer Perceptron (MLP) was chosen for this task as a neural network model due to its ability to capture non-linear relationships in the data through its multi-layer structure. Compared to other neural networks, such as convolutional or recurrent networks, MLPs are simpler and well-suited for structured tabular data where spatial or sequential dependencies (like images or time series) are not present. The MLP's fully connected layers allow it to learn complex patterns from the input features, making it a suitable choice for this binary classification task.

The input layer size of 8 was chosen to match the number of input features (after preprocessing and feature selection), ensuring that the model receives all relevant information about the exoplanets. The three hidden layers were designed with 5 and 4 neurons, striking a balance between model complexity and computational efficiency. Too many neurons could lead to overfitting, while too few might result in underfitting. This architecture is sufficient given the size and complexity of the dataset. The output layer size of 2 corresponds to the two possible classes: confirmed exoplanets and false positives. Since this is a binary classification task, having 2 neurons in the output layer is appropriate, with each representing one of the classes.

The code implemented a pipeline that first used the VectorAssembler to combine the features into a vector and then passed them into the MultilayerPerceptronClassifier. The MLP model was trained on the training set and then evaluated on the test set.

The performance metrics revealed that the MLP model performed exceptionally well, with a test accuracy, precision, recall and an F1 score of approximately 98.5%. These metrics indicate that the model is highly accurate and balances both precision and recall effectively, making it reliable for predicting both confirmed exoplanets and false positives. Additionally, the area under the ROC curve (AUC) was 1.00, indicating perfect classification performance, with the model being able to distinguish between the two classes without any errors.

This strong performance suggests that the architecture chosen, including the number of layers and neurons, was sufficient for the task and provided a robust classification model.

10.9.1 Model Fitting

In [94]:
```
# Defining the layers
layers = [8, 5, 4, 2]  # Input size 8, three hidden layers with 5 and 4 neurons, and output size 2 (classes)

# Create the MultilayerPerceptronClassifier (Neural Network model)
mlpc = MultilayerPerceptronClassifier(featuresCol="features", labelCol="koi_disposition", predictionCol="predicte

# Create a Pipeline with the VectorAssembler and MultilayerPerceptronClassifier
mlp_pipeline = Pipeline(stages=[assembler, mlpc])

# Train the Neural Network model
mlpc_model = mlp_pipeline.fit(train_df)

# Make predictions using the test set
mlp_predictions = mlpc_model.transform(test_df)
```

10.9.2 Model Evaluation

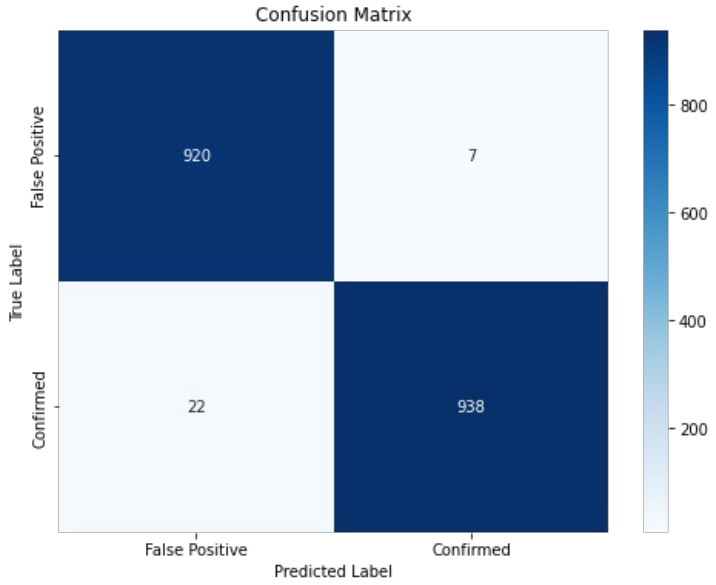
10.9.2 Model Evaluation

In [95]:
```
# Evaluating the multilayer perceptron model
mlp_metrics = evaluate_classification_model(mlp_predictions, label_col="koi_disposition", pred_col="predicted_lab
```

```
Test Accuracy: 0.9846316905140434
Test Precision: 0.9847584564707084
Test Recall: 0.9846316905140434
Test F1 Score: 0.984632855942399
```

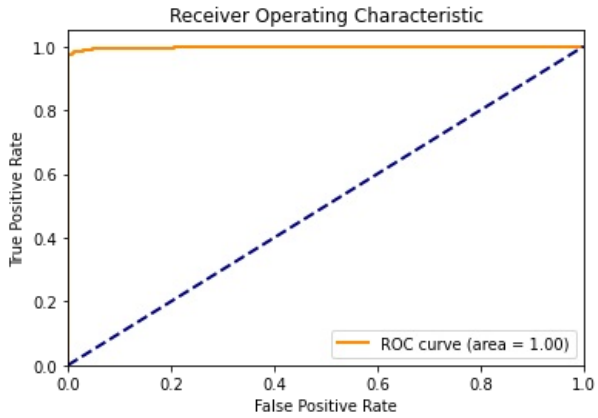
10.9.3 Confusion Matrix

In [96]:
```
# Plot the multilayer perceptron confusion matrix
plot_confusion_matrix(mlp_predictions, label_col="koi_disposition", pred_col="predicted_label")
```



10.9.4 ROC Curve

In [97]:
```
# Plot neural network roc curve
plot_roc_curve(mlp_predictions, label_col="koi_disposition", raw_pred_col="rawPrediction")
```



---

11. Exoplanet Prediction on Current Exoplanet Candidates

The final step in the implementation of this project was to apply the best machine learning model, the Random Forest, to real-world data. Specifically, this involved predicting the classification of current candidate exoplanets—whether they are false positives or confirmed exoplanets. This implementation underscores the importance of the project by demonstrating its practical applicability to real scientific data, potentially aiding in the discovery and validation of new exoplanets. The predictive power of machine learning in this context can significantly streamline the classification process, reducing the need for manual and time-consuming investigations.

The predictions were added to the dataframe, and a bar plot was generated to visualise the results. According to the predictions from the Random Forest model, out of a total of 1,928 candidate exoplanets, 1,151 were predicted to be confirmed exoplanets, while 831 were

classified as false positives. This indicates that the model was able to provide clear predictions, helping astronomers prioritise further investigation into these candidates. By leveraging machine learning, this project demonstrates its potential to accelerate the process of exoplanet discovery and improve the accuracy of classifications.

However, it is important to note that while the Random Forest model performed well on the training and test datasets, its predictions on the real-world data should be interpreted with caution. Machine learning models, including Random Forests, are only as good as the data they are trained on, and any biases or limitations in the training data may carry over to the predictions. For instance, the model might not account for nuances or unseen patterns in the candidate exoplanet data that were not present in the training set.

Moreover, real-world predictions of exoplanet classification rely on a range of observational and astrophysical factors, and while machine learning provides an efficient and accurate method for classification, it cannot replace the need for subsequent scientific validation through further observations. Thus, these predictions serve as a helpful tool to prioritise candidates for follow-up but should not be taken as definitive without additional investigation with external data sources and expert review to ensure their reliability and accuracy.

In [99]:
```
def make_predictions(model: PipelineModel, data: DataFrame, features_col: str = "features", prediction_col: str =
    """
    Function to pass data into an already built machine learning model for prediction.

    Parameters:
    ----------
    model : PipelineModel
        A pre-trained machine learning model in PySpark

    data : DataFrame
        The input data on which predictions are to be made. The DataFrame must have a column for features, typica

    features_col : str, optional, default="features"
        The name of the column in the input DataFrame containing the feature vectors. This column will be used by

    prediction_col : str, optional, default="prediction"
        The name of the column in the output DataFrame that will store the prediction results.

    Returns:
    -------
    DataFrame
        A new DataFrame with predictions, containing all the original columns along with a new column for predict
    """
    # Check if the model and data are valid inputs
    if not isinstance(model, PipelineModel):
        raise TypeError("The 'model' must be a PySpark PipelineModel.")

    if not isinstance(data, DataFrame):
        raise TypeError("The 'data' must be a PySpark DataFrame.")

    # Ensure that the features column exists in the input DataFrame
    if features_col not in data.columns:
        raise ValueError(f"Column '{features_col}' not found in the input DataFrame.")

    # Apply the model to the data to make predictions
    predictions = model.transform(data)

    # Select the relevant columns: original data and prediction column
    return predictions.select("*", prediction_col)
```

In [120…
```
# Converting the candidate pandas dataframe to a Pyspark dataframe
candidate_data = sq.createDataFrame(df_cd)

# Combining raw feature columns into a 'features' column
assembler = VectorAssembler(inputCols=important_features, outputCol="assembled_features")
prediction_data = assembler.transform(candidate_data)

# Apply the model and get predictions
predictions = rf_model.transform(prediction_data)

# Print the schema to verify the output columns
# predictions.printSchema()

# If the 'prediction' column exists, use it. Otherwise, adjust accordingly.
if "prediction" in predictions.columns:
    random_forest_predictions = make_predictions(rf_model, prediction_data, features_col="assembled_features", pr
else:
    # Use the available prediction-related column like 'rawPrediction' or 'probability'
    random_forest_predictions = make_predictions(rf_model, prediction_data, features_col="assembled_features", pr
```

In [125…
```
# Convert to pandas dataframe
rf_pred = random_forest_predictions.toPandas()

# Drop the added features column
rf_pred = rf_pred.drop(['assembled_features', 'rawPrediction'], axis=1)

# Replace 0 with 'FALSE POSITIVE' and 1 with 'CONFIRMED' in the 'predicted_label' column
rf_pred['predicted_label'] = rf_pred['predicted_label'].replace({0 : 'FALSE POSITIVE', 1 : 'CONFIRMED'})
```

```
# View the dataframe
rf_pred.head()
```

Out[125…

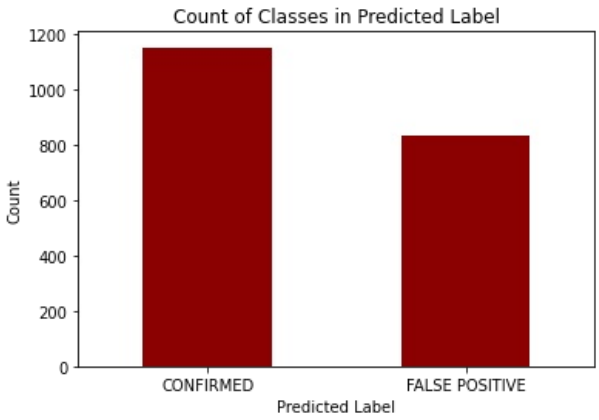
	koi_score	koi_fpflag_nt	koi_fpflag_ss	koi_fpflag_co	koi_fpflag_ec	koi_prad	koi_period	koi_duration	koi_depth	koi_srho	...	koi_srad
0	-0.483246	0.0	0.0	0.0	0.0	0.597066	0.267138	-0.523712	5.297507	2.032300	...	-0.231026
1	0.520774	0.0	0.0	0.0	0.0	0.247504	0.807411	-0.112172	2.962587	0.277714	...	-0.376147
2	0.514744	0.0	0.0	0.0	0.0	0.836189	-0.066142	-0.842618	0.052437	2.476976	...	-0.402836
3	0.392131	0.0	0.0	0.0	0.0	-0.095649	-0.166316	-0.171827	-0.219808	-0.185378	...	0.134279
4	0.521779	0.0	0.0	0.0	0.0	0.258351	0.011288	-0.073904	2.699685	-0.214507	...	-0.284404

5 rows × 26 columns

In [134…

```
# Count the occurrences of each class in the 'predicted_label' column
class_counts = rf_pred['predicted_label'].value_counts()

print(class_counts)

# Create a bar plot
class_counts.plot(kind='bar', color='darkred')

# Add title and labels
plt.title('Count of Classes in Predicted Label')
plt.xticks(rotation=0)
plt.xlabel('Predicted Label')
plt.ylabel('Count')

# Show the plot
plt.show()
```

```
CONFIRMED         1151
FALSE POSITIVE     831
Name: predicted_label, dtype: int64
```



---

12. Summary and Conclusion

12.1 Summary

This project aimed to classify exoplanet candidates using a variety of machine learning models. The dataset was preprocessed, normalised, and key features were selected based on domain knowledge and exploratory data analysis. A variety of models were implemented, including Naive Bayes, Logistic Regression, Support Vector Machine, Random Forest, and a Multilayer Perceptron. These models were evaluated on metrics such as accuracy, precision, recall, F1 score, and area under the ROC curve. Among the models, Random Forest performed the best with an accuracy of 98.8% and an AUC of 1.0, making it the selected model for further use.

The final step involved applying the Random Forest model to real-world data consisting of exoplanet candidates. Predictions were made to classify these candidates as either confirmed exoplanets or false positives. Out of 1,928 candidates, the model predicted that 1,151 were confirmed exoplanets, while 831 were classified as false positives. This outcome underscores the practical application of machine learning in

exoplanet discovery, providing a faster and more efficient way to classify large datasets and prioritize candidates for further investigation.

12.2 Conclusion

In conclusion, the implementation of machine learning in exoplanet classification has demonstrated strong potential, with the Random Forest model emerging as a highly accurate tool for classifying exoplanet candidates. The project highlights the power of machine learning algorithms in analyzing complex astronomical datasets and making predictions that can guide further scientific investigation. However, while the model's performance on real-world data is promising, the predictions should be validated through additional scientific methods and observations to ensure their accuracy and reliability.

Ultimately, this project illustrates how machine learning can be a valuable tool in the field of astronomy, particularly in the classification of exoplanet candidates. By improving the efficiency and accuracy of this process, machine learning models can significantly contribute to the discovery of new exoplanets, enhancing our understanding of distant worlds.

Kindly see the third component of this project for more in-depth analysis of the results, discussion and conclusion.

---

13. References

1. Ansdell, M., Ioannou, Y., Osborn, H. P., Sasdelli, M., Smith, J. C., Caldwell, D. A., … & Angerhausen, D. (2018). Scientific domain knowledge improves exoplanet transit classification with deep learning. The Astrophysical Journal Letters, 869(1), L7. https://doi.org/10.3847/2041-8213/aaf23b.
2. Basak, S., Agrawal, S., Saha, S., Theophilus, A., Bora, K., Deshpande, G., … & Murthy, J. (2018). Habitability classification of exoplanets: a machine learning insight.. https://doi.org/10.48550/arxiv.1805.08810
3. Forestano, R. T., Matchev, K. T., Matcheva, K., & Unlu, E. B. (2023). Searching for novel chemistry in exoplanetary atmospheres using machine learning for anomaly detection. The Astrophysical Journal, 958(2), 106. https://doi.org/10.3847/1538-4357/ad0047.
4. Jin, Y., Yang, L., & Chiang, C. (2022). Identifying exoplanets with machine learning methods: a preliminary study. International Journal on Cybernetics &Amp; Informatics, 11(2), 31-42. https://doi.org/10.5121/ijci.2022.110203.
5. Pham, D. and Kaltenegger, L. (2022). Follow the water: finding water, snow and clouds on terrestrial exoplanets with photometry and machine learning.. https://doi.org/10.48550/arxiv.2203.04201. Sasmita Kumari Nayak (2024). Classification of kepler exoplanet searching from galaxy using machine learning model. World Journal of Advanced Research and Reviews, 21(1), 227-230. https://doi.org/10.30574/wjarr.2024.21.1.0012.
6. Shallue, C. J. and Vanderburg, A. (2018). Identifying exoplanets with deep learning: a five-planet resonant chain around kepler-80 and an eighth planet around kepler-90. The Astronomical Journal, 155(2), 94. https://doi.org/10.3847/1538-3881/aa9e09
7. Singh, H. V., Agarwal, N., & Yadav, A. (2024). A novel methodology for hunting exoplanets in space using machine learning. EAI Endorsed Transactions on Internet of Things, 10. https://doi.org/10.4108/eetiot.5331.
8. Ulmer-Moll, S., Santos, N. C., Figueira, P., Brinchmann, J., & Faria, J. P. (2019). Beyond the exoplanet mass-radius relation. Astronomy &Amp; Astrophysics, 630, A135. https://doi.org/10.1051/0004-6361/201936049.

Non-academic Referencing:

1. ChatGPT, OpenAI, 2024: For assistance with restructuring paragraphs for better coherence, more scientific tone and grammatical correctness.

Thank you for reading this notebook :)