DATA ANALYTICS

Team Error_404

Sheetali Maity

Saptarshi Maity

Rishav Ray Chaudhury

Ritabrata Roy

1. Visualization and Analysis of Dataset:

✨1.1 Inspecting the data type and extracting basic statistics:

Data Type:

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4048 entries, 0 to 4047
Columns: 112 entries, P_NAME to P_SEMI_MAJOR_AXIS_EST
dtypes: float64(94), int64(4), object(14)
memory usage: 3.5+ MB
```

We employed the **data.info()** method to retrieve the following information.

No of Rows: 4048

No of COlumns: 112

float64: 94 features

int64: 4

object: 14

Data Shape:

```
data.shape
```

```
(4048, 112)
```

The dataset shape is presented here (4048,112)

1.2.1 Printing Range, Mean, Median, and Standard Deviation of the dataset.

- Used data.describe for the above question for all the numerical columns.
  Here Mean is given by the mean, Median by 50 th percentile, Standard Deviation by std.

```
print(data.describe())
```

```
       P_STATUS        P_MASS  P_MASS_ERROR_MIN  P_MASS_ERROR_MAX  \
count    4048.0   1598.000000       1467.000000       1467.000000
mean        3.0    798.384920       -152.292232        190.289692
std         0.0   1406.808654        783.366353       1082.061976
min         3.0      0.019070     -24965.390000          0.000000
25%         3.0     26.548968        -79.457001          4.449592
50%         3.0    273.332080        -24.154928         25.108412
75%         3.0    806.488560         -4.392383         85.813561
max         3.0  17668.059000          0.270000      26630.808000

          P_RADIUS  P_RADIUS_ERROR_MIN  P_RADIUS_ERROR_MAX       P_YEAR  \
count  3139.000000         3105.000000         3105.000000  4048.000000
mean      4.191426           -0.483990            0.621867  2014.212945
std       4.776830            1.409048            2.007592     3.704839
min       0.336300          -54.592700            0.000000  1989.000000
25%       1.569400           -0.526870            0.145730  2014.000000
50%       2.331680           -0.235410            0.325090  2016.000000
75%       3.553570           -0.134520            0.661390  2016.000000
max      77.349000            0.450000           68.919080  2019.000000

           P_PERIOD  P_PERIOD_ERROR_MIN  ...  S_SNOW_LINE   S_ABIO_ZONE  \
count  3.938000e+03        3.807000e+03  ...  3786.000000  3.083000e+03
mean   2.309342e+03       -1.073631e+03  ...     3.513348  1.768991e+35
std    1.167012e+05        5.943181e+04  ...     5.463171  6.944274e+36
min    9.070629e-02       -3.650000e+06  ...     0.002405  7.293660e-05
25%    4.497336e+00       -1.129000e-03  ...     1.740762  5.264169e-01
50%    1.187053e+01       -9.392000e-05  ...     2.568600  1.429118e+00
75%    4.186661e+01       -1.594000e-05  ...     3.661581  2.641037e+00
max    7.300000e+06        3.200000e-02  ...   104.112780  2.726899e+38
```

Range:

```python
# Dropping object type columns (string columns)
data_numeric = data.select_dtypes(exclude=['object'])

data_range = data_numeric.max() - data_numeric.min()
print("Range:")
print(data_range)
```

```
                            Range
P_STATUS                 0.000000
P_MASS                17668.039930
P_MASS_ERROR_MIN      24965.660000
P_MASS_ERROR_MAX      26630.808000
P_RADIUS                 77.012700
...                            ...
P_HABITABLE               2.000000
P_ESI                     0.924439
P_RADIUS_EST             77.349000
P_MASS_EST            17668.059000
P_SEMI_MAJOR_AXIS_EST  2499.995600

[98 rows x 1 columns]
```

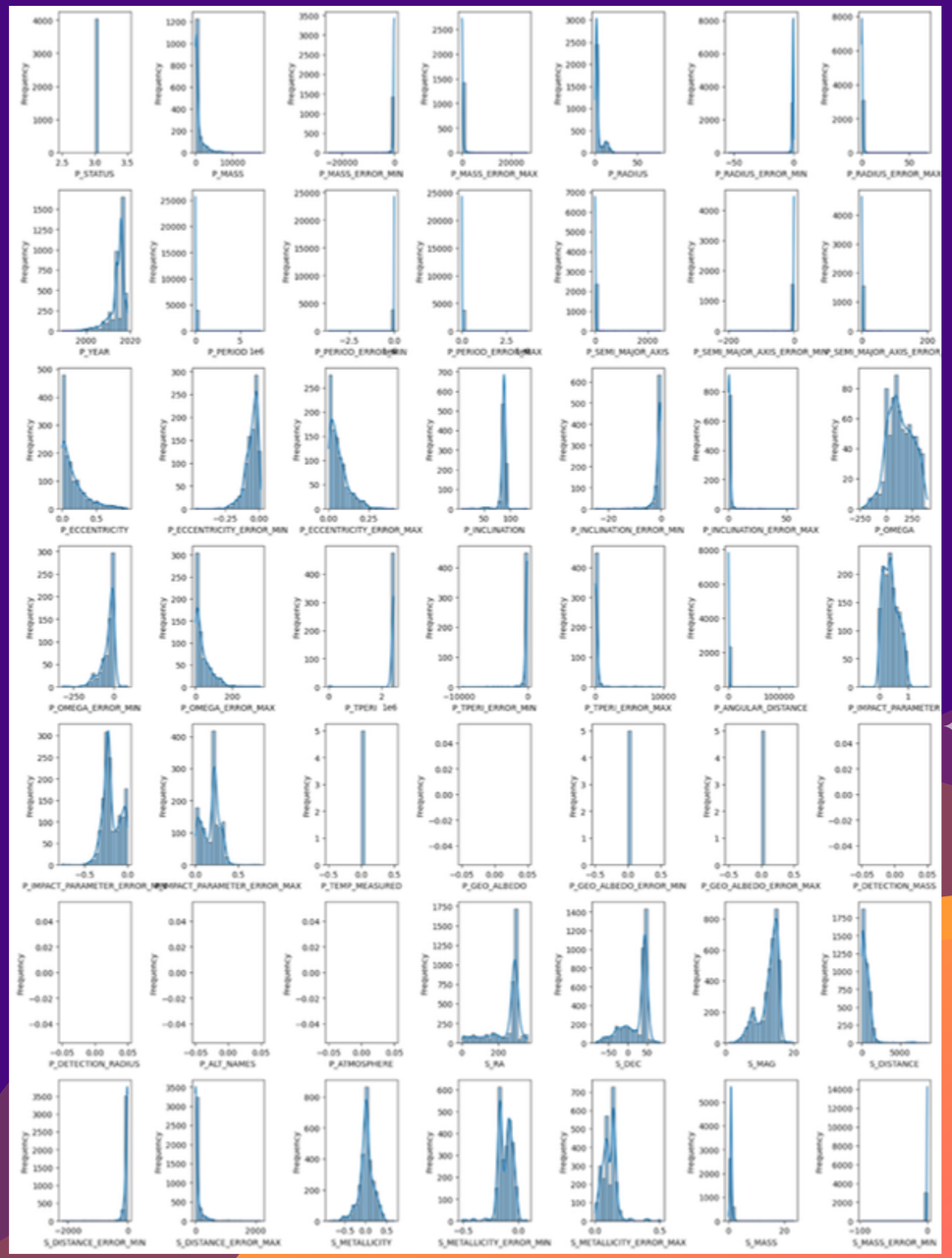
Using the formula max minus min, we determined the dataset's range.

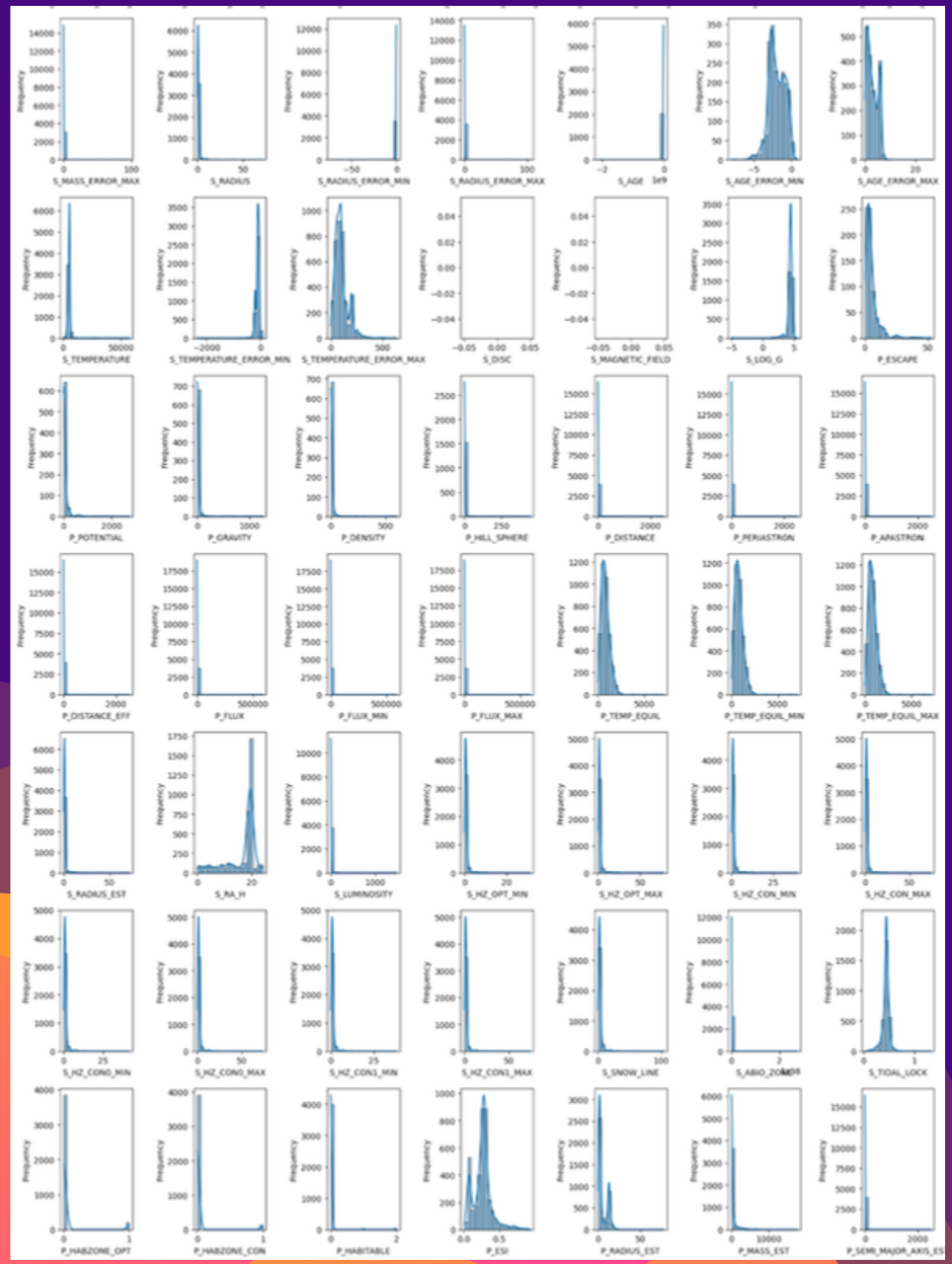
1.1.2 DOES THE DATASET REQUIRE NORMALISATION?

```
import seaborn as sns
import matplotlib.pyplot as plt

numeric_columns = df.select_dtypes(include=['number'])
columns_to_plot = numeric_columns.columns
fig, axes = plt.subplots(16, 7, figsize=(16, 48))
plt.subplots_adjust(hspace=0.5)
axes = axes.flatten()

for i, col in enumerate(columns_to_plot):
    ax = axes[i]
    sns.histplot(data=df, x=col, bins=20, ax=ax, kde=True)
    ax.set_xlabel(col)
    ax.set_ylabel('Frequency')

for i in range(len(columns_to_plot), 16*7):
    fig.delaxes(axes[i])

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

Yes, The dataset does require Normalisation.While plotting the numerical features with hisplot (kde=True), we observe that different features follow different distributions (like some are normally distributed, while others are skewed).Normalization will help to scale and standardize the numerical features specially for scale-sensitive ML algorithms like SVMs,etc.

Output Graphs:



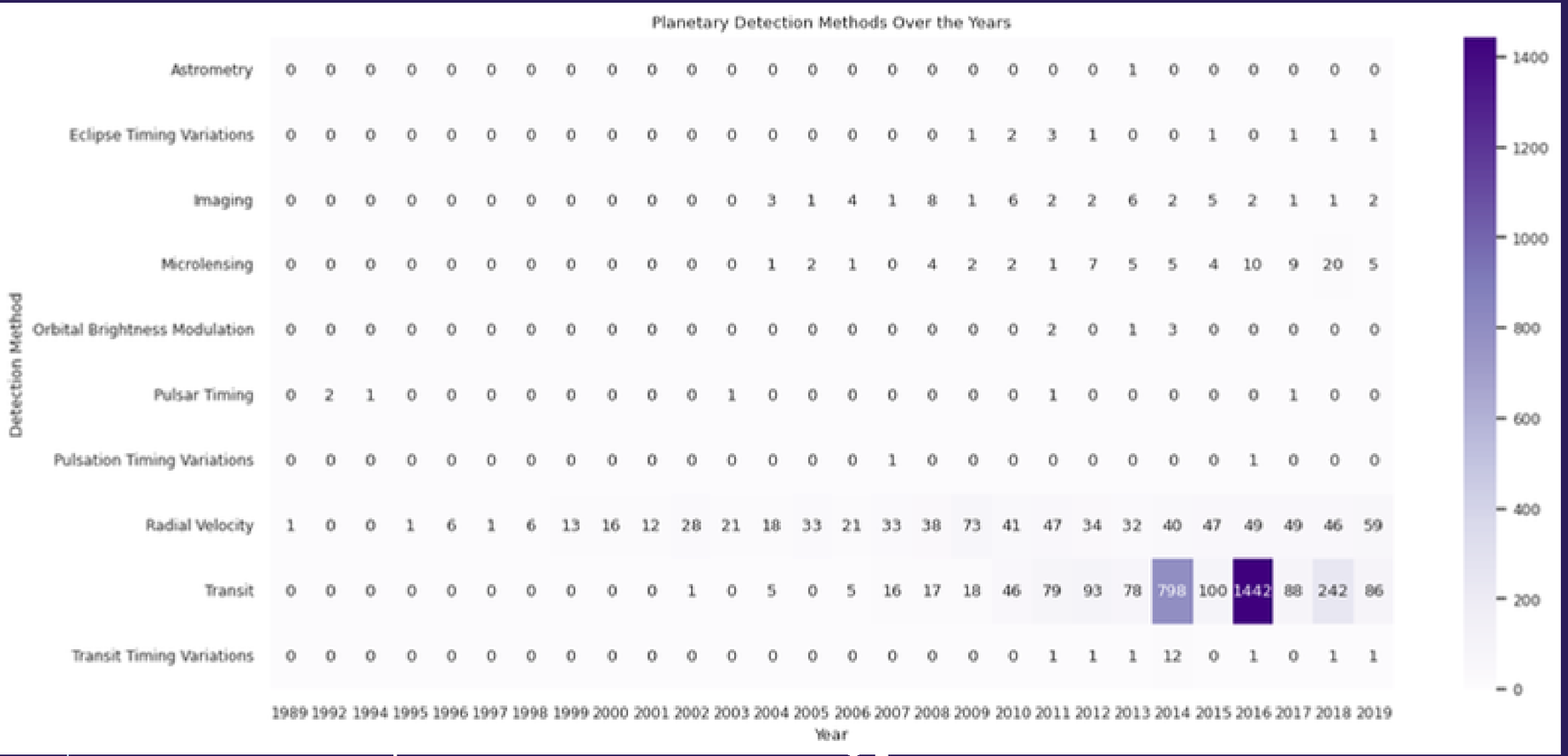


1.1.3 What are your inferences based on the above results?

The range of the result varies vastly from column to column and hence we need to normalize the data. We can also find a **high standard deviation in many columns** which suggests that there is a **substantial amount of variability or dispersion in the data**. **The data points are not closely clustered around the mean but are instead scattered over a wide range of values in many columns like P_MASS, P_PERIOD.** The difference between the mean and median signifies the presence of skewness or asymmetry in the distribution of data.

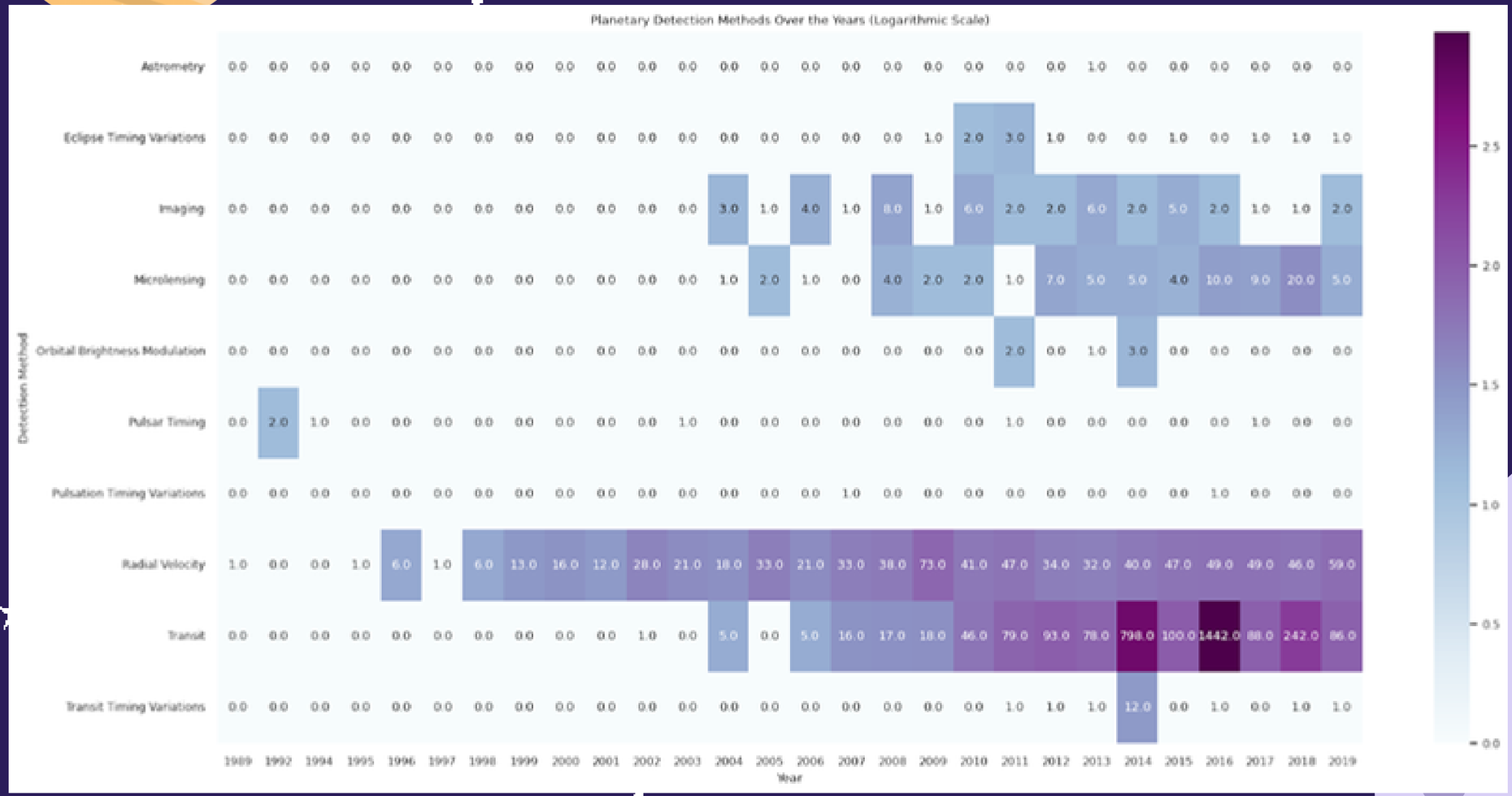
1.2 Using the Seaborn module, plot a heatmap to explore the various planetary detection methods used over the years

```
[8]  plt.figure(figsize = (15, 7))
     sns.set(font_scale=0.8)
     sns.heatmap(heatmap_data,  cmap="Purples",annot=True, fmt='d',cbar=True)
     plt.title('Planetary Detection Methods Over the Years')
     plt.xlabel('Year')
     plt.ylabel('Detection Method')
     plt.show()
```



Log scale Heatmap

```
heatmap_data_log = heatmap_data.applymap(lambda x: 1 if x == 0 else x)
heatmap_data_log = heatmap_data_log.applymap(lambda x: 0 if x == 1 else x)
heatmap_data_log = heatmap_data_log.applymap(lambda x: abs(x) ** 0.15)
```



1.2.1 Inference from the above heatmap:

MOST COMMONLY USED DETECTION METHODS:

1.Transit
2.Radial velocity
3.Microlensing
4.Imaging

RARELY USED DETECTION METHODS:

1.Astrometry
2.Eclipse timing variations
3.Orbital Brightness Modulation
4.Pulsar Timing
5.Pulsation Timing Variations
6.Transit Timing Variations

1.3 Identify the planetary detection methods that have identified the most: i) Uninhabitable planets (0) ii) Conservatively habitable planets (1) iii) Optimistically habitable planets (2)

```
Uninhabitable_df = df[df['P_HABITABLE'] == 0]
Conservatively_habitable_df = df[df['P_HABITABLE'] == 1]
Optimistically_habitable_df = df[df['P_HABITABLE'] == 2]
```

Here is the code to make separate datasets based on exoplanet habitability types

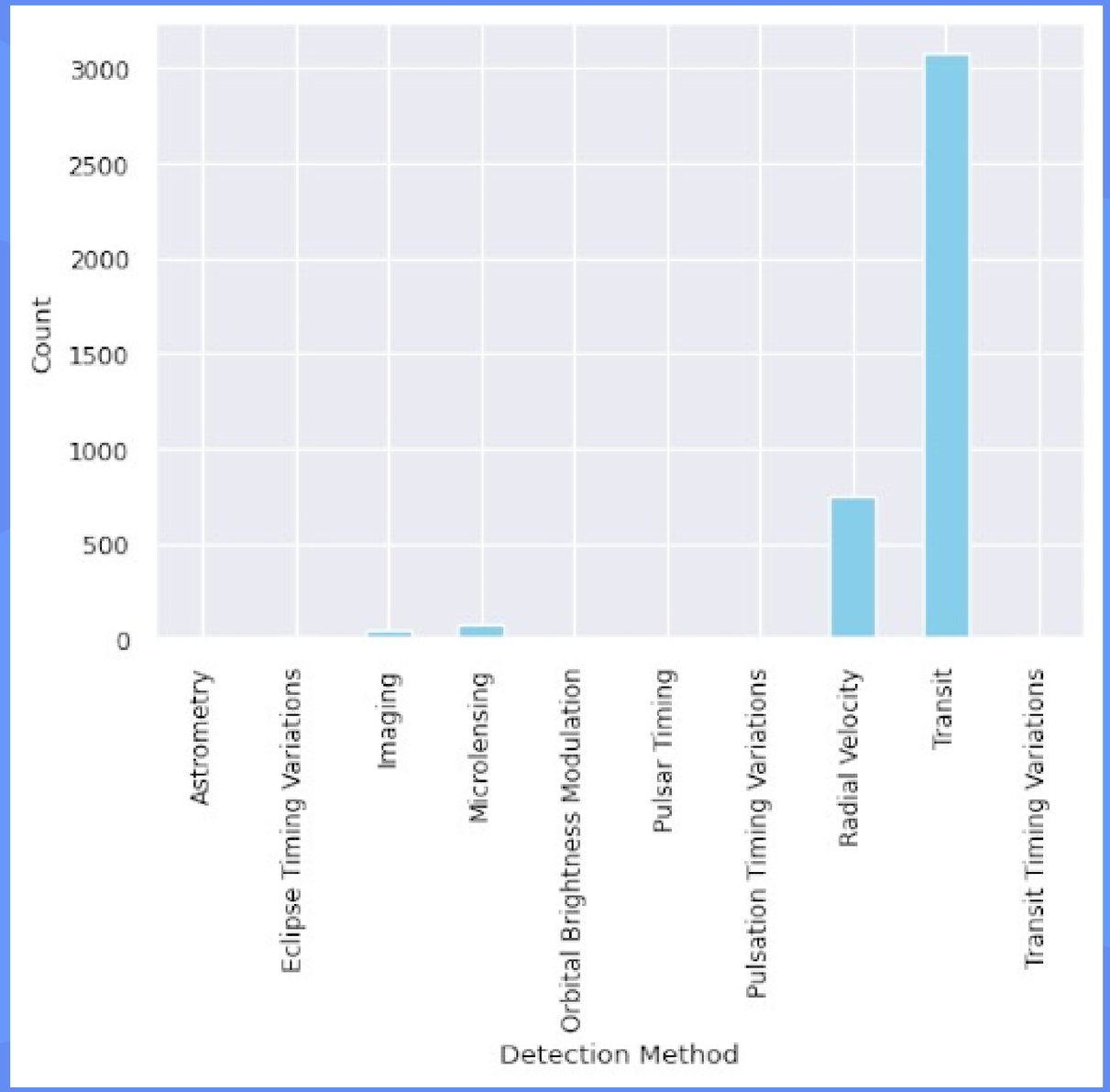
```
# Calculating the count of uninhabitable planets for each detection method.
Uninhabitable_count= Uninhabitable_df.groupby('P_DETECTION')['P_NAME'].count()
plt.figure(figsize=(6, 4))
Uninhabitable_count.plot(kind='bar', color='skyblue') # Plotting bar chart
plt.xlabel('Detection Method')
plt.ylabel('Count')
plt.xticks(rotation=90)
plt.show()
```

Here we are ploting bar plots of Count vs Detection Method

Output:

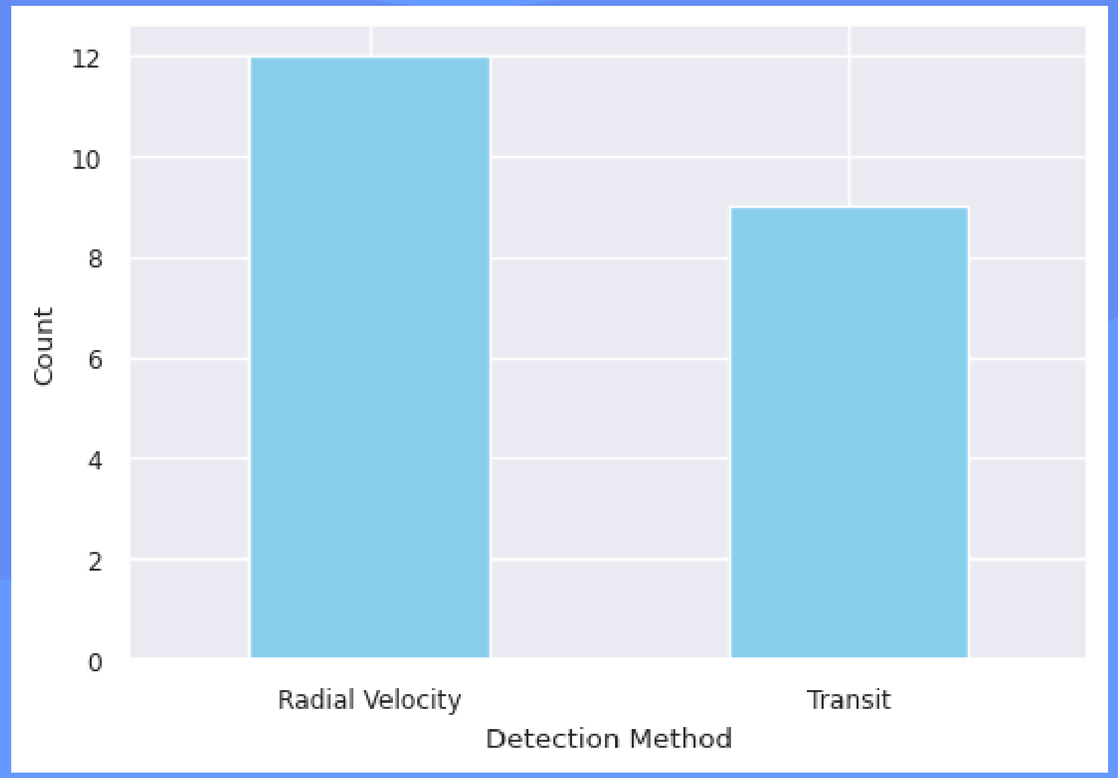
Uninhabitable Planets

We've generated bar plots illustrating the relationship between Detection Method and Count.

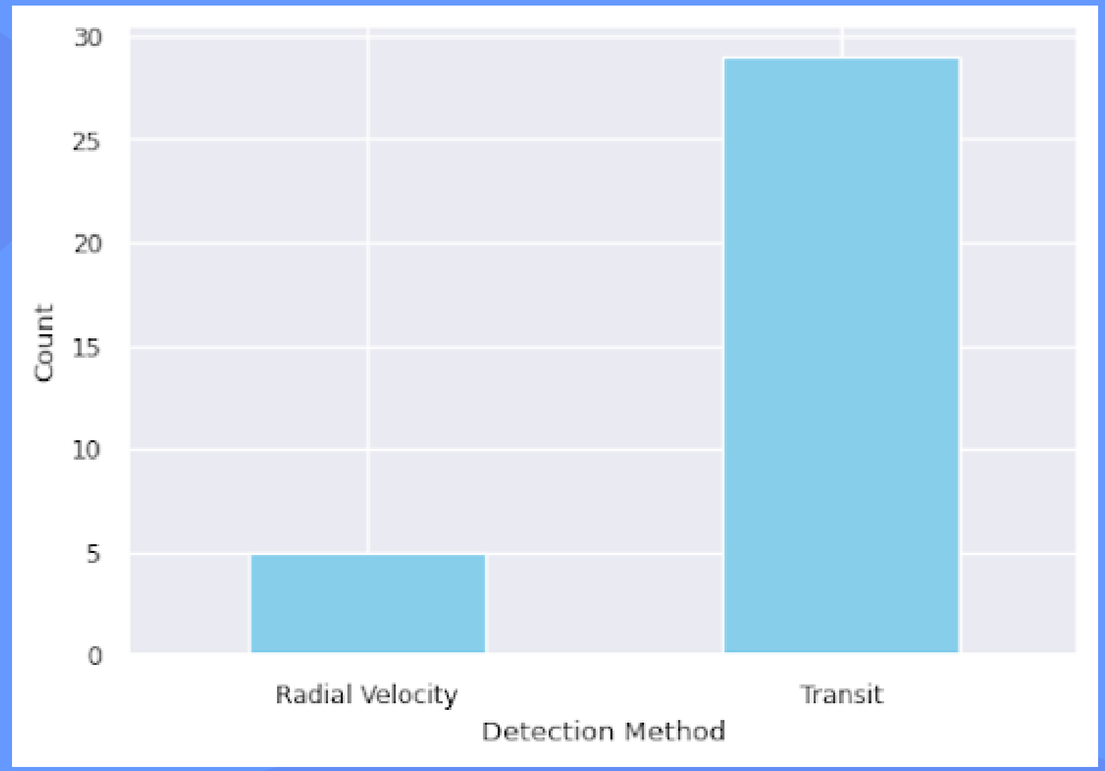


Output

Conservatively habitable



Optimistically habitable



Here are the bar plots for
Conservatively Habitable and
Optimistically Habitable exoplantes

1.4 Determine the Interquartile Range and the Skewness of the Dataset

IQR

```
iqr_df = pd.DataFrame({'IQR': df.describe(percentiles=[.25, .75]).loc['75%'] - df.describe(percentiles=[.25, .75]).loc['25%']})
iqr_df
```

Output

	IQR
P_STATUS	0.000000
P_MASS	779.939592
P_MASS_ERROR_MIN	75.064618
P_MASS_ERROR_MAX	81.363969
P_RADIUS	1.984170
P_HABITABLE	0.000000
P_ESI	0.106363
P_RADIUS_EST	10.066580
P_MASS_EST	145.751059
P_SEMI_MAJOR_AXIS_EST	0.207507

We utilized the df.describe function to obtain the 75% and 25% values to calculate the interquartile range.

Skewness

Code:

```
from scipy.stats import skew
numeric_columns = df.select_dtypes(include=['number'])
skewness = numeric_columns.skew()


skewness_df = pd.DataFrame(skewness)
```

We utilized the scipy library to compute the skewness.

Output:

P_STATUS	0.000000
P_MASS	3.709352
P_MASS_ERROR_MIN	-23.147053
P_MASS_ERROR_MAX	19.064529
P_RADIUS	2.957998
P_HABITABLE	9.321263
P_ESI	1.039325
P_RADIUS_EST	1.545462
P_MASS_EST	5.797200
P_SEMI_MAJOR_AXIS_EST	28.395487

1.5 How would you tackle the classification bias (class imbalance) of the Dataset?

Code+Output(Target Class Imbalance)

```
counts = df.groupby('P_HABITABLE')['P_NAME'].count()

print("Uninhabitable Planets:",counts[0])
print("Conservatively Habitable Planets:",counts[1])
print("Optimistically Habitable Planets:",counts[2])
```

```
Uninhabitable Planets: 3993
Conservatively Habitable Planets: 21
Optimistically Habitable Planets: 34
```

Ways to tackle class imbalance

01 Oversampling with SMOTE

```python
from sklearn.preprocessing import LabelEncoder
from sklearn.impute import SimpleImputer
from imblearn.over_sampling import SMOTE
X = df.drop('P_HABITABLE', axis=1)
y = df['P_HABITABLE']

# Identifying and label encoding string-type columns
string_columns = X.select_dtypes(include=['object']).columns
label_encoders = {}

for col in string_columns:
    label_encoder = LabelEncoder()
    X[col] = label_encoder.fit_transform(X[col])
    label_encoders[col] = label_encoder

# Imputing missing values in X with the mean
imputer = SimpleImputer(strategy='mean')
X = imputer.fit_transform(X)

# Applying SMOTE to the entire dataset for the target 'P_HABITABLE'
smote = SMOTE(random_state=42)
X_resampled, y_resampled = smote.fit_resample(X, y)

class_counts = pd.Series(y_resampled).value_counts()
print("Balanced Target Classes After SMOTE:")
print(class_counts)
```

Output:

```
Balanced Target Classes After SMOTE:
0    3993
2    3993
1    3993
Name: P_HABITABLE, dtype: int64
```

2. SMOTE ENN

SMOTE generates synthetic samples to balance class distribution, while ENN removes redundant samples from the majority class. Together, they help create a more balanced and informative dataset for machine learning models, improving classification performance.

```
#SMOTE to oversample the minority class
smote = SMOTE(sampling_strategy='auto', random_state=42)
X_resampled, y_resampled = smote.fit_resample(X, y)

#ENN to clean the majority class samples
enn = EditedNearestNeighbours(sampling_strategy='auto')
X_resampled, y_resampled = enn.fit_resample(X_resampled, y_resampled)

print("Class distribution after resampling:", Counter(y_resampled))
```

```
Class distribution before resampling: Counter({0: 3993, 2: 34, 1: 21})
Class distribution after resampling: Counter({0: 3993, 1: 3988, 2: 3987})
```

Here, we can see the target class distribution

1.6 Report the ratio of the Number of Iron atoms to the Number of Hydrogen atoms for the exoplanets in the dataset by creating a separate feature titled ‘Fe to H Ratio’

```
[8]  #50th percentile value (median) of the "S_METALLICITY" column
     met_fill = data["S_METALLICITY"].describe()[5]

[9]  #fillna method is used to replace NaN (missing) values in the column with median
     data["S_METALLICITY"].fillna(met_fill,inplace=True)
```

Metallicity = log(Fe/H)(star) - log(Fe/H)(sun)

The Metallicity is the ratio of the available heavy metals (here, Fe) with respect to the available H and He in the star.

```
     log_sun = -4.3
0s

[12] data["Fe to H Ratio"] = np.power(10,data["S_METALLICITY"]-4.3)
0s
```

We calculated the "Fe to H Ratio" for exoplanets in a dataset by using the "S_METALLICITY" column and a fixed value for log_sun. This ratio represents the abundance of iron (Fe) relative to hydrogen (H) atoms. We filled missing values with median in the "S_METALLICITY" column, computed the ratio, and stored it as a new feature.

Output

```
print(data["Fe to H Ratio"])
```

```
0       0.000022
1       0.000048
2       0.000029
3       0.000129
4       0.000058
          ...
4041    0.000052
4042    0.000052
4045    0.000042
4046    0.000042
4047    0.000042
Name: Fe to H Ratio, Length: 4046, dtype: float64
```

2. Interpretation and calculation of physical parameters:

2.1 Calculate the escape velocities of exoplanets and compare them to their estimated temperatures. Present a plot of estimated temperature with escape velocity. Explain the nature of the plot obtained

```
df['Escape_Velocity'] = np.sqrt(2*6.674e-11*(df['P_MASS_EST']*5.972e+24)/(df['P_RADIUS_EST']*6.378e+6))
df['Escape_Velocity']
```

```
0       252546.020228
1       218808.258591
2       121819.258711
3       119951.443659
4        72581.721923
            ...
4043     16665.343168
4044     19089.796499
4045     12506.099720
4046     13624.189946
4047     13425.159978
Name: Escape_Velocity, Length: 4048, dtype: float64
```

We calculated it by utilizing the escape velocity formula, which depends on the gravitational constant (G), the estimated mass of each exoplanet (converted from Earth masses to kilograms), and the estimated radius of each exoplanet (converted from Earth radii to meters).

Estimated Temperatures

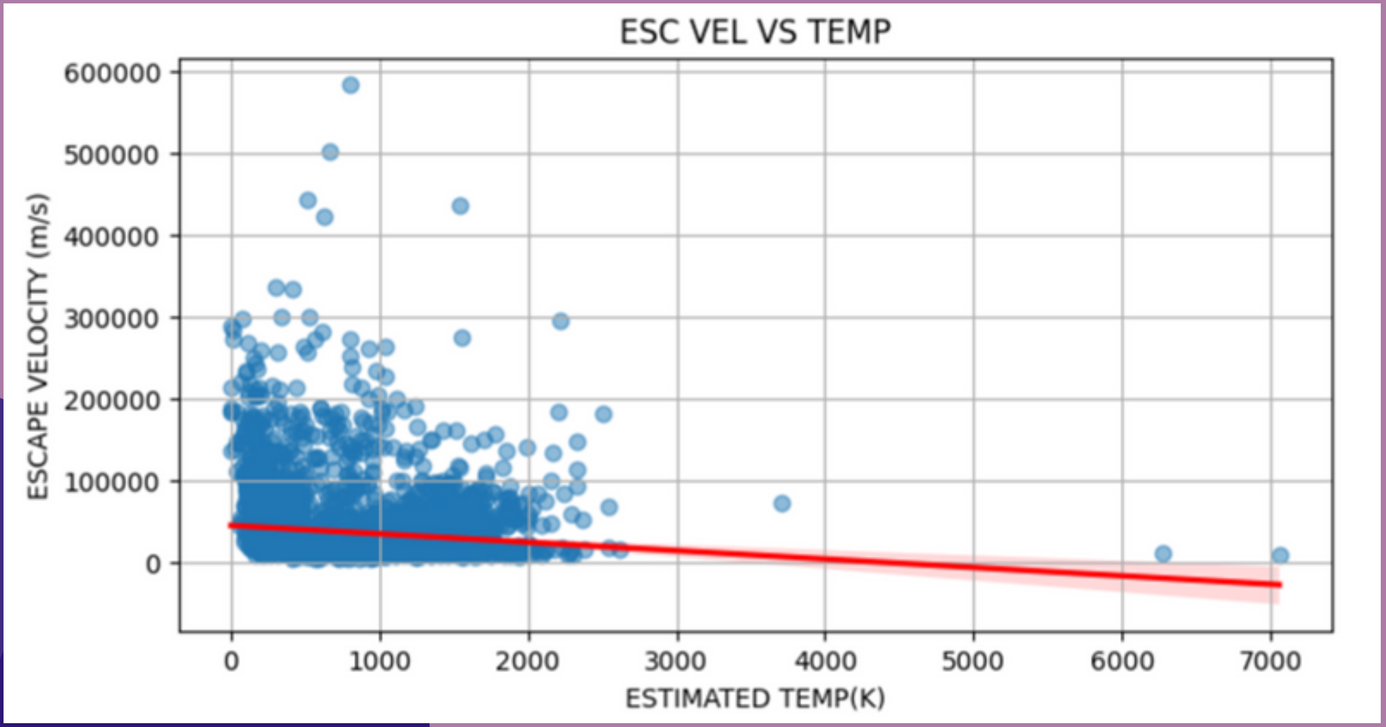
```
df['P_TEMP_EQUIL']
```

```
0        799.44963
1        819.07569
2        772.30746
3        131.38424
4        202.22783
           ...
4043     277.27227
4044     423.46173
4045     354.75879
4046     274.79498
4047     221.23098
Name: P_TEMP_EQUIL, Length: 4048, dtype: float64
```

Here we presented the Planet's equilibrium temperature from P_TEMP_EQUIL column

Scatterplot of estimated temperature vs escape velocity.

```
plt.figure(figsize=(8, 4))
plt.scatter(df['P_TEMP_EQUIL'],df['Escape_Velocity'],alpha=0.5) #scatterplot
sns.regplot(data=df,x='P_TEMP_EQUIL', y='Escape_Velocity', scatter= False, line_kws={'color':'red'}) #regression line
plt.xlabel('ESTIMATED TEMP(K)')
plt.ylabel('ESCAPE VELOCITY (m/s)')
plt.title('ESC VEL VS TEMP')
plt.grid(True)
plt.show()
```



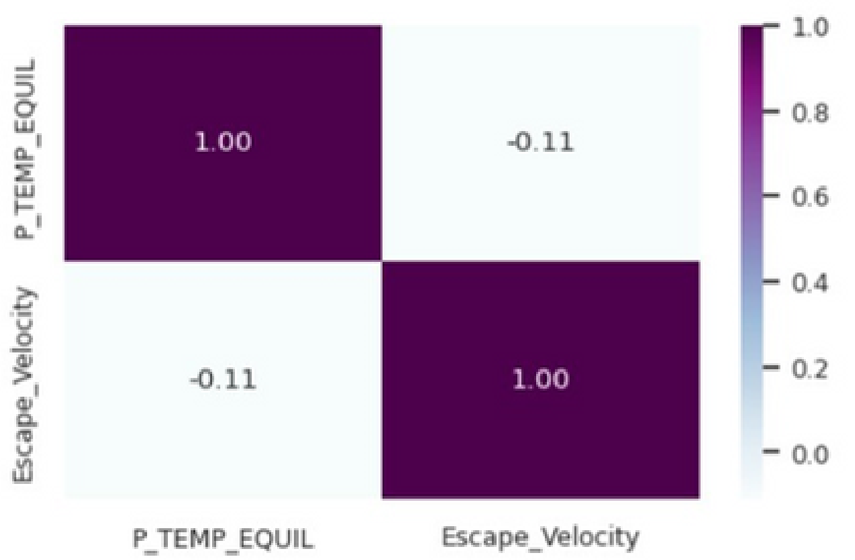
The nature of the plot obtained:

This is case of inverse or negative correlation.The red regression line is added to indicate the direction of the inverse correlation. As estimated temperature decreases, escape velocity tends to increase, demonstrating the inverse relationship.

This -ve correlation(-0.11) is also shown through the heatmap of df.corr().

This -ve correlation(-0.11) is also shown through the heatmap of df.corr().

```
plt.figure(figsize=(5, 3))
sns.heatmap(df[['P_TEMP_EQUIL', 'Escape_Velocity']].corr(), annot=True, cmap="BuPu", fmt=".2f")
plt.show()
```



2.1.1 Analyze whether these velocities are sufficient to retain their atmospheres, drawing connections to atmospheric escape processes.

$$F_{Jeans} = S_c \frac{n_c U}{2\sqrt{\pi}} e^{-V_{esc}^2/U^2} \left(1 + V_{esc}^2/U^2\right)$$

We calculated Fjeans from this formula and made a seperate feature in the df

```
df['Jeans Esc']=(44.6*df['Thermal Vel']/2*np.sqrt(3.14)) * (1+ (df['Escape_Velocity']/df['Thermal Vel'])**2)
```

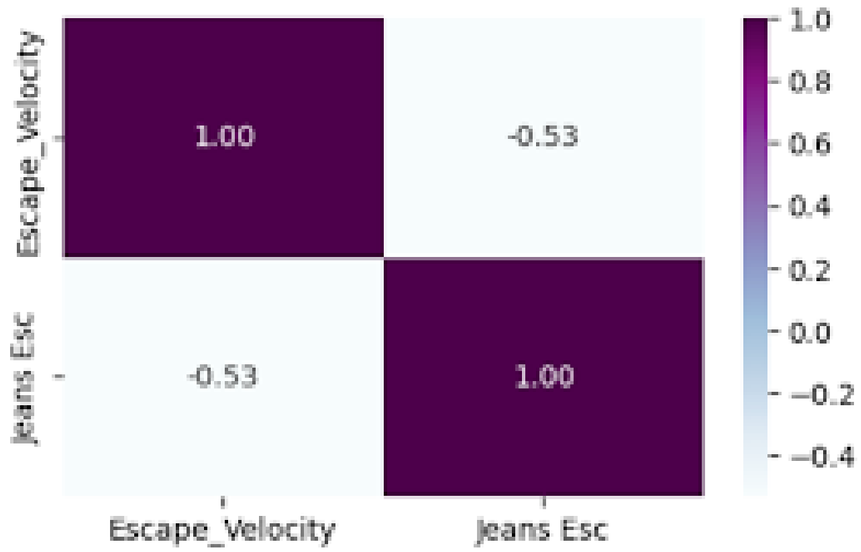
```
)*np.exp(-(df['Escape_Velocity']/df['Thermal Vel'])**2)
```

```
df['Jeans Esc']
```

```
0          337.888672
1         1429.178051
2        13987.032334
3           17.101186
4         5054.597374
```

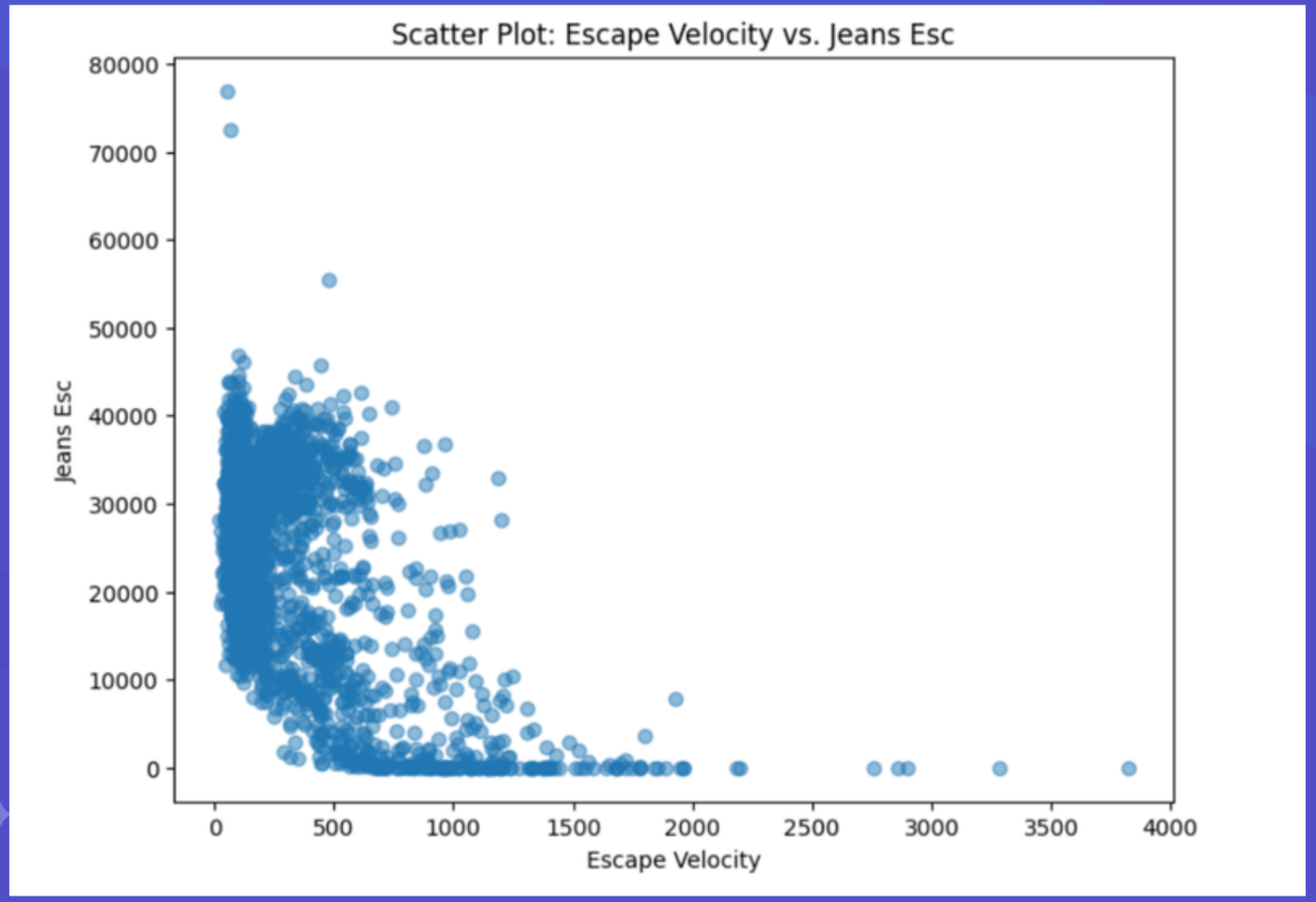
[15]:

```
plt.figure(figsize=(5, 3))
sns.heatmap(df[['Escape_Velocity','Jeans Esc']].corr(), annot=True, cmap="BuPu", fmt=".2f")
plt.show()
```



There is a negative correlation between Jeans Escape (a measure of loss of a part of the atmosphere due to thermal velocity of atmospheric molecules) and Escape Velocity.

We further plotted a scatterplot



Based on the scatter plot analysis, it appears that there is an inverse relationship between Jeans Escape and the escape velocity of exoplanets. Specifically, when the Jeans Escape value is high, it suggests that the exoplanet may struggle to retain its atmosphere. Conversely, when an exoplanet has a low escape velocity, it becomes more susceptible to atmospheric loss, resulting in a higher Jeans Escape value.

2.2 How does the distribution of host star ages correlate with the metallicity of their associated exoplanets?

```
#handling na data with median
data["S_AGE"].fillna(data["S_AGE"].describe()[5],inplace=True)
```

+ Code + Text

```
[15] data["S_AGE"]
```

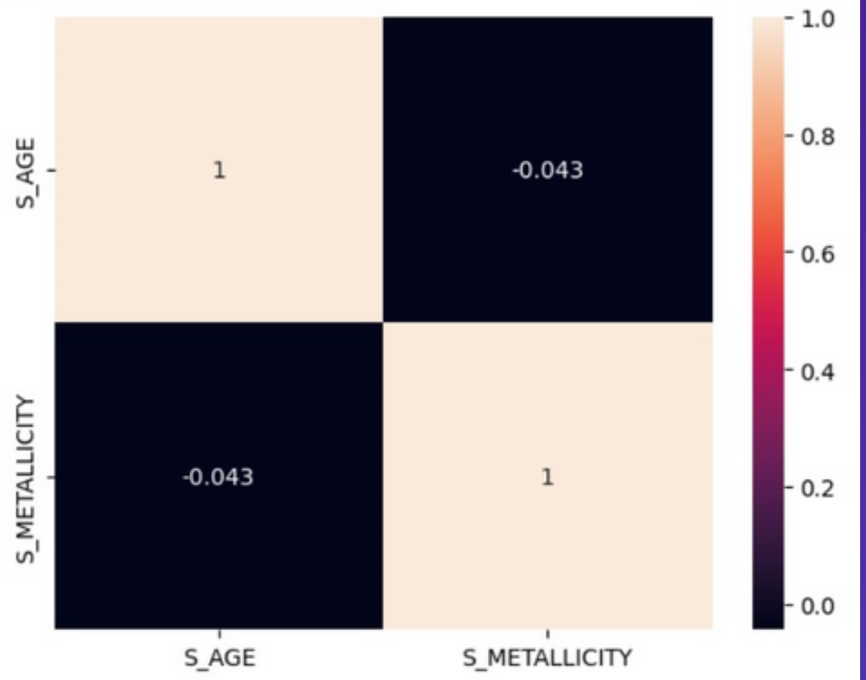
```
0       4.070000e+00
1       4.070000e+00
2       4.070000e+00
3       4.070000e+00
4       4.070000e+00
              ...
4043   -2.147484e+09
4044   -2.147484e+09
4045    7.000000e+00
4046    7.000000e+00
4047    7.000000e+00
Name: S_AGE, Length: 4048, dtype: float64
```

Imputed the na values with median

plotted both heatmap and scatterplot

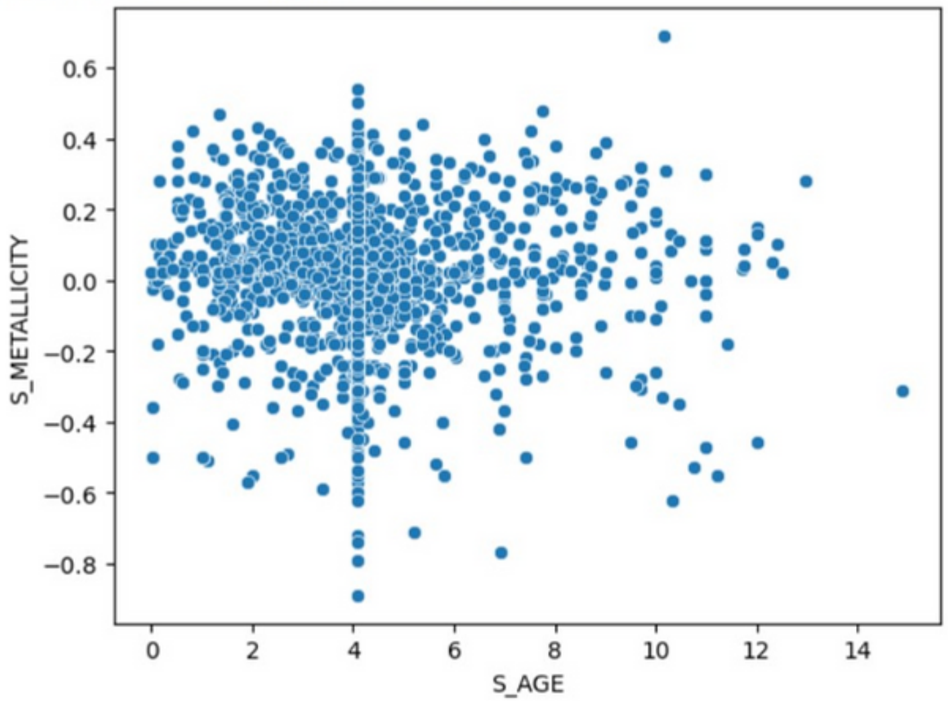
```
sns.heatmap(data[["S_AGE","S_METALLICITY"]].corr(),annot=True)
```

```
<Axes: >
```




```
[18] sns.scatterplot(x=data["S_AGE"],y=data["S_METALLICITY"])
```

```
<Axes: xlabel='S_AGE', ylabel='S_METALLICITY'>
```



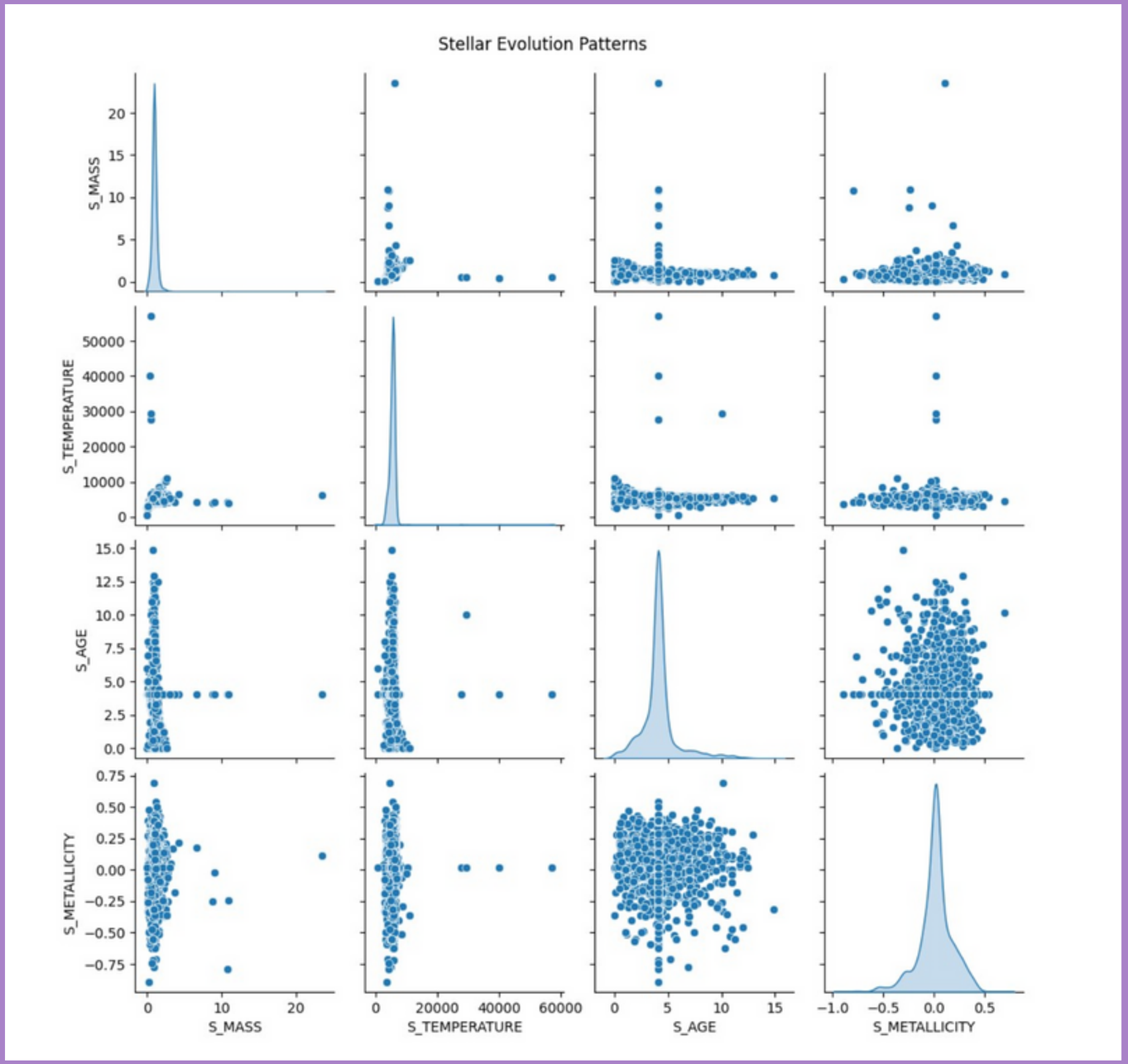
Star age and metallicity have little to no correlation. Over a large Gyr span, the age of a star has negligible bearing on its metallicity. There is no conceivable pattern as is apparent from the scatterplot above.

2.2.1 Identify any patterns that align with our understanding of stellar evolution and planet formation

```python
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt


star_columns = ['S_MASS', 'S_TEMPERATURE', 'S_AGE', 'S_METALLICITY']

# Droping rows with missing values in star columns
data.dropna(subset=star_columns, inplace=True)

#pairplot to visualize relationships between star-related variables
sns.pairplot(data[star_columns], diag_kind='kde')
plt.suptitle('Stellar Evolution Patterns', y=1.02)
plt.show()
```

We used the following four features for our analysist

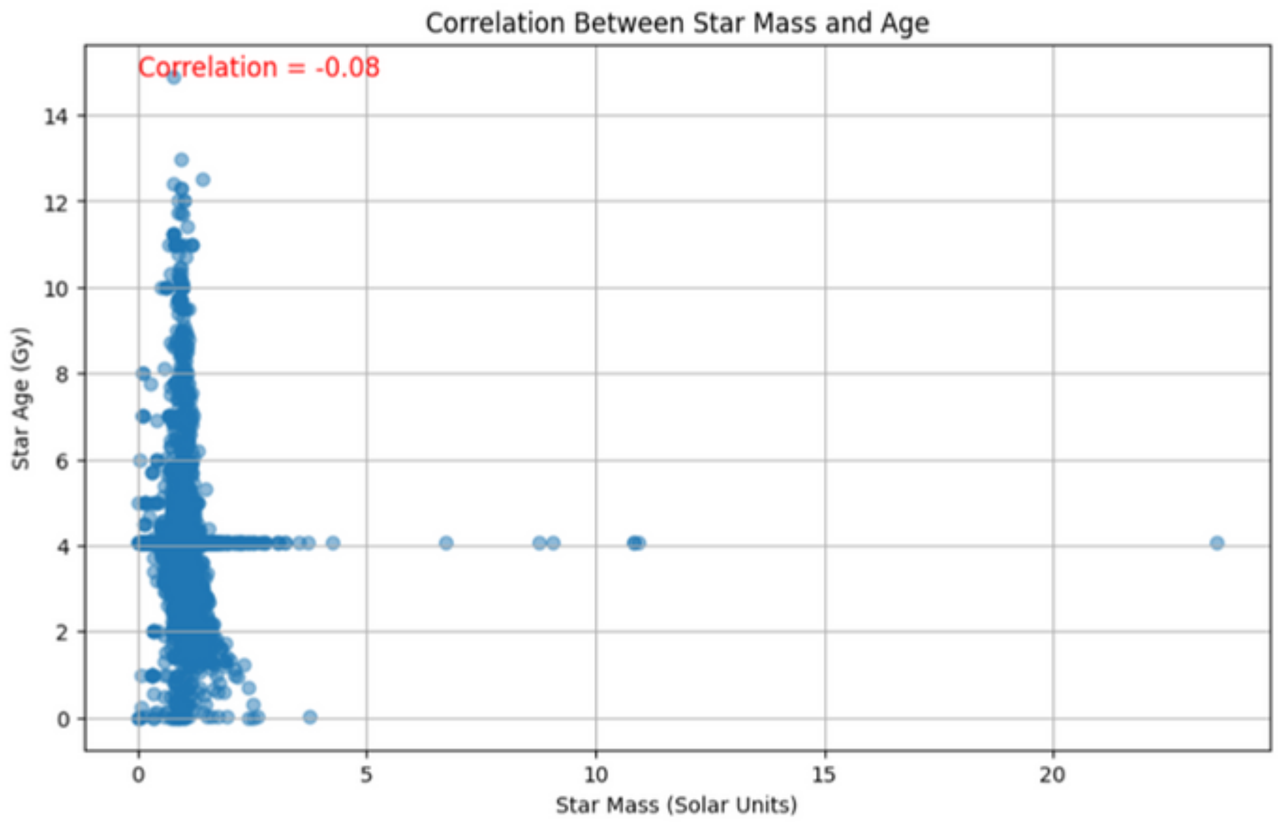
- Star mass
- Star Temperature
- Star Age
- Star Metallicity



1. Younger stars are hotter and more massive (indicating the main-sequence phase) or if older stars are cooler and may have transitioned into later phases such as red giants.
2. Age-metallicity relation (AMR) in the disk is used for developing chemical evolution models. (homogeneity and inhomogeneity)
3. A concentration of data points/clusters in a specific regions of the scatter plot indicate that many stars share similar characteristics and are at a similar phase in their evolution.

Correlation between Star Mass and Age

```
data.dropna(subset=['S_MASS', 'S_AGE'], inplace=True)
correlation = data['S_MASS'].corr(data['S_AGE']) #correlation
plt.scatter(data['S_MASS'], data['S_AGE'], alpha=0.5)#scatterplot
plt.title('Correlation Between Star Mass and Age')
plt.xlabel('Star Mass (Solar Units)')
plt.ylabel('Star Age (Gy)')
plt.grid(True)
```



A weaker or negative correlation indicates the presence of more massive, shorter-lived stars still in the main sequence phase.
More massive stars, which have shorter lifespans due to their high rates of hydrogen fusion, may still be in the main sequence phase. They burn through their hydrogen fuel relatively quickly.

Planet Formation Patterns

Code to create a pair plot

```
planet_variables = ['P_MASS', 'P_RADIUS', 'P_SEMI_MAJOR_AXIS']

# host star's mass
star_variables = ['S_MASS']
subset_data = data[planet_variables + star_variables]
subset_data.dropna(subset=planet_variables + star_variables, inplace=True)#dropping na values
# pair plot
sns.set(style="ticks")
g = sns.PairGrid(subset_data, diag_sharey=False)
g.map_lower(sns.scatterplot, palette='viridis', markers='o')
g.map_diag(sns.kdeplot)
g.fig.suptitle('Planet Formation Patterns', y=1.02)

correlation_coefficients = subset_data.corr()
for i, j in zip(*plt.np.triu_indices_from(g.axes, 1)):
    g.axes[i, j].annotate(f"Corr: {correlation_coefficients.values[i, j]:.2f}", (0.5, 0.9), xycoords='axes fraction', ha='center', fontsize=10, color='r')

plt.show()
```

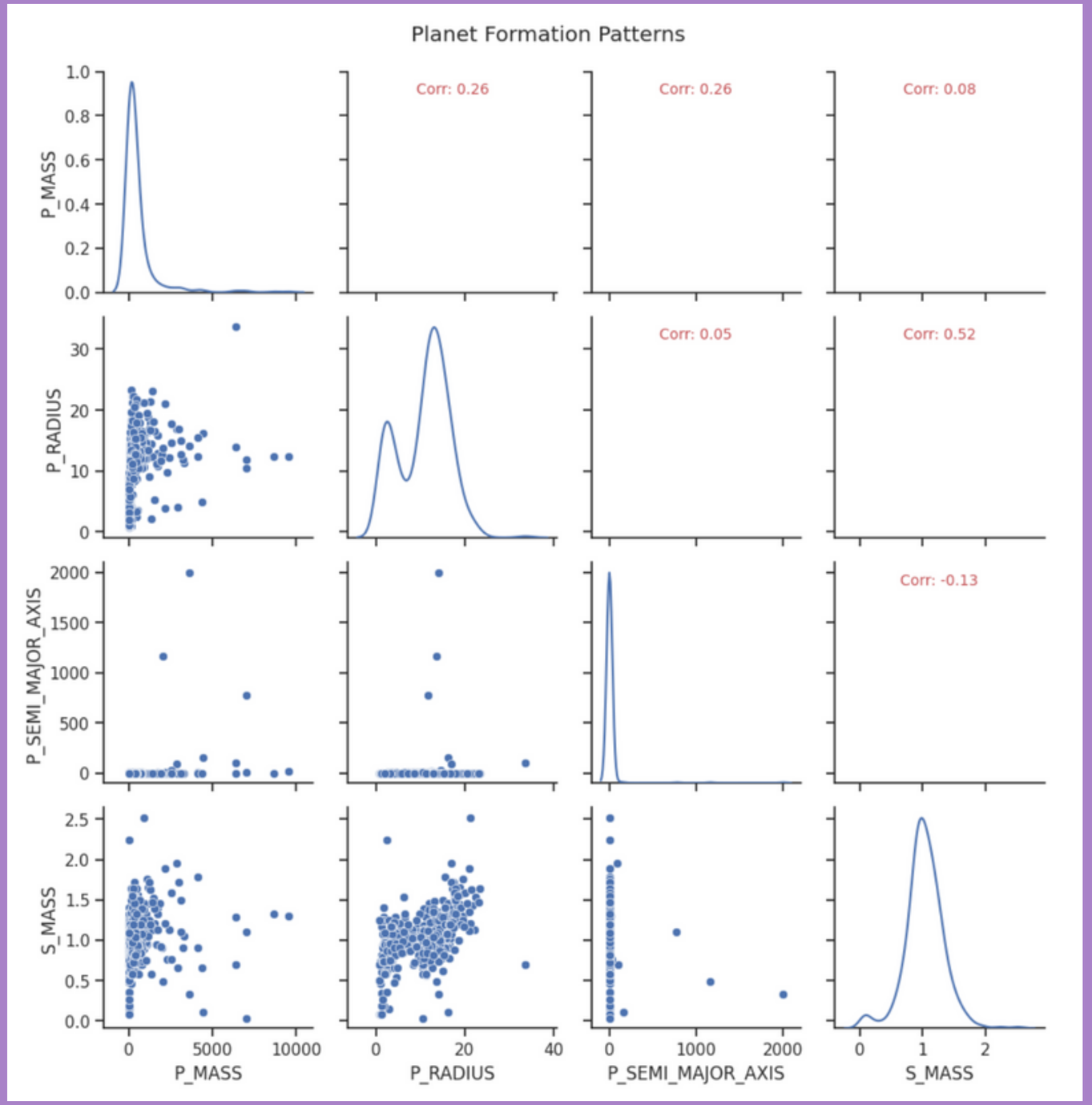
Planet Formation Patterns

Mass-Orbit Interaction (P_MASS vs. P_SEMI_MAJOR_AXIS):

- The correlation between planet mass and semi-major axis **reveals information about planetary migration** or interactions with the protoplanetary disk during formation.
- **A positive correlation says that more massive planets tend to migrate inward or outward during their evolution,** impacting their final orbital distances.

Mass-Radius Relationship (P_MASS vs. P_RADIUS):

- **A positive correlation between planet mass and radius suggests that more massive planets tend to be larger,** resembling gas giants like Jupiter or Saturn.
- This correlation aligns with the idea that massive planets can accumulate significant gas envelopes during their formation, a process linked to the early stages of planetary evolution.



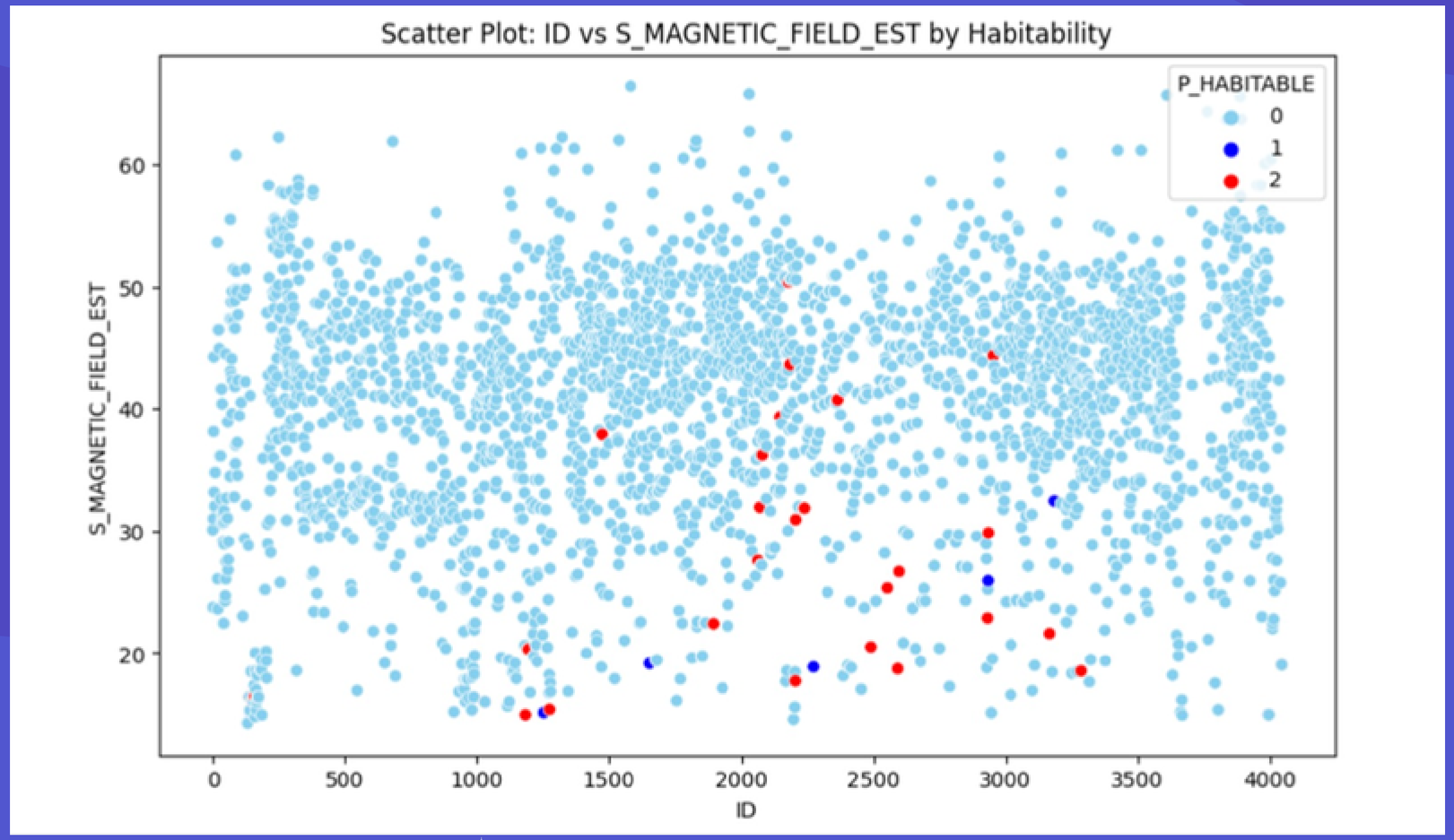
2.3 Examine the correlation between host star magnetic field strength and exoplanet atmospheric properties.

```
mu_0 = 4 * np.pi * 1e-7
df['S_MAGNETIC_FIELD_EST'] = np.sqrt((2 * mu_0 * df['S_LUMINOSITY']*3.846e26) / ((df['S_RADIUS']* 6.95700e+8)**2))
```

```
Q1 = df['S_MAGNETIC_FIELD_EST'].quantile(0.25)
Q3 = df['S_MAGNETIC_FIELD_EST'].quantile(0.75)
IQR = Q3 - Q1

lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

filtered_df = df[(df['S_MAGNETIC_FIELD_EST'] >= lower_bound) & (df['S_MAGNETIC_FIELD_EST'] <= upper_bound)]

plt.figure(figsize=(10, 6))
sns.scatterplot(data=filtered_df, x=filtered_df.index, y='S_MAGNETIC_FIELD_EST', hue='P_HABITABLE', palette=['skyblue','blue','red'])
plt.xlabel('ID')
plt.ylabel('S_MAGNETIC_FIELD_EST')
plt.title('Scatter Plot: ID vs S_MAGNETIC_FIELD_EST by Habitability')

plt.show()
```

Approach

2.3 Examine the correlation between host star magnetic field strength and exoplanet atmospheric properties.



It is evident that planets with low to medium magnetic fields are likely to be habitable because high magnetic fields can introduce stellar winds to the planet and too low magnetic fields cannot offer some protection from the charged particles in the stellar wind, reducing their impact on the planet's atmosphere to nearby exoplanets.

2.3.1 Explore how magnetospheric interactions might impact the composition and stability of exoplanet atmospheres

Atmospheric Erosion: Stellar winds can strip the atmosphere of planets with weak or no magnetic fields, leading to the loss of vital gases

Ionization and Heating: Stellar wind-ionosphere interactions heat and ionize the upper atmosphere, potentially causing expansion and altering atmospheric chemistry.

Magnetospheric Protection: Planets with strong magnetic fields deflect harmful charged particles, preserving atmospheres and shielding from radiation.

Auroras: Stellar wind interactions produce auroras, revealing magnetic fields and atmospheric composition.

Stability and Climate: Magnetic fields impact long-term atmospheric stability, influencing planetary habitability.

2.4 What are Spectral Types? How many major star Spectral Types exist?

Spectral types refer to a system of classifying stars based on the detailed analysis of their electromagnetic spectra. This classification is fundamental in astrophysics and provides crucial insights into the physical characteristics and evolutionary stages of stars.

01

O-Type Stars : hottest and most massive in the classification and surface temperatures exceeding 30,000 Kelvin

02

B-Type Stars : B-type stars are also hot and massive, with temperatures ranging from 10,000 to 30,000 Kelvin.

03

A-Type Stars: A-type stars have surface temperatures between 7,500 and 10,000 Kelvin. They appear white or bluish-white and exhibit strong hydrogen lines in their spectra.

04

F-Type Stars: With temperatures between 6,000 and 7,500 Kelvin,they have a yellowish-white appearance and display significant hydrogen lines.

05

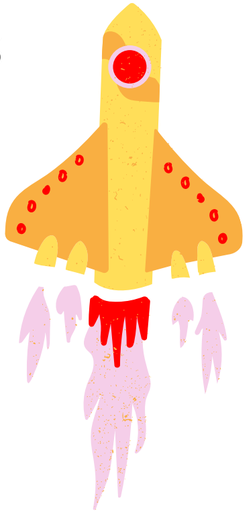
G-Type Stars: G-type stars, like Sun, have temperatures ranging from 5,000 to 6,000 Kelvin. They appear yellow.

06

K-Type Stars : With temperatures between 3,500 and 5,000 Kelvin. They appear orange-red and are known for strong neutral metal lines in their spectra.

07

M-Type Stars : With temperatures ranging from 2,400 to 3,500 Kelvin,they appear red and display molecular absorption bands in their spectra.



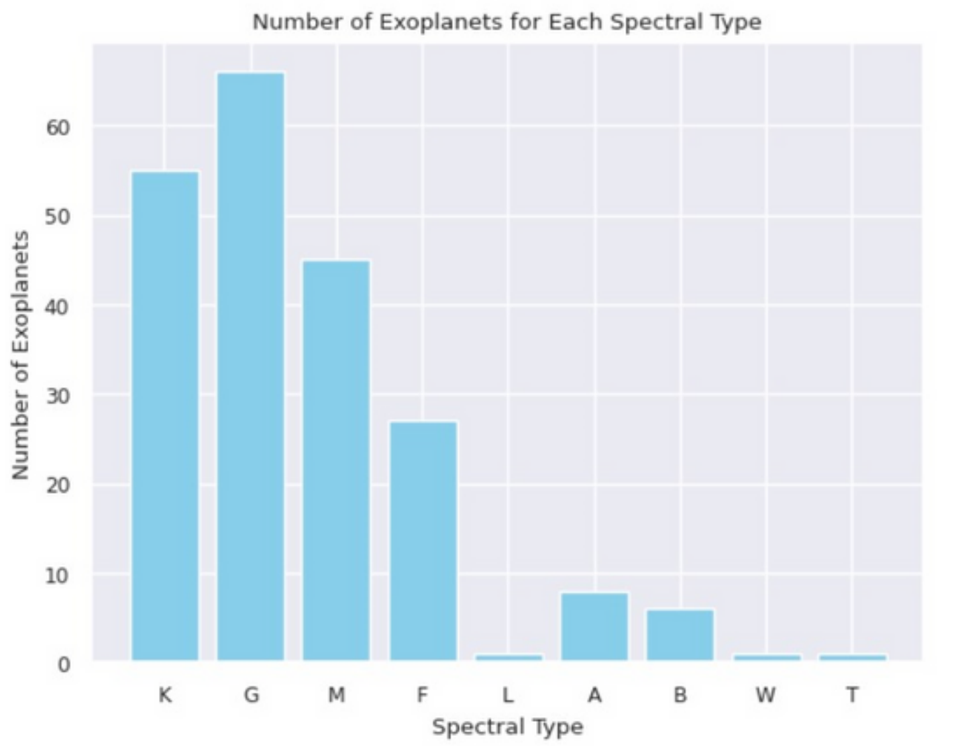
2.4.1 Present a categorical plot representing various Spectral Types and the habitability associated with it.

No of exoplanets for each spectral types:

```
#a dictionary to count the occurrences of each spectral type
spectral_counts = {'K':0, 'G':0, 'M':0, 'F':0, 'L':0, 'A':0, 'B':1, 'W':0, 'T':0}
#3829 sdBV start with letter s but is actually B type so B is initialised to 1
spectral_types = df['S_TYPE'].unique()
for i in spectral_types:
    if pd.notna(i): # Checking if the spectral type is not NaN
        a = i[0].upper()
        if a in spectral_counts:
            spectral_counts[a]+=1 #incrementing the counter

print(spectral_counts)
# empty lists to store spectral types and their counts
spectral_counts_list=[]
spectral_type_list=[]
for i in spectral_counts:
  spectral_type_list.append(i)

print(spectral_type_list)
# Extracting spectral counts from the spectral_counts dictionary
for i in spectral_counts:
  spectral_counts_list.append(spectral_counts[i])
print(spectral_counts_list)

#bar plot to visualize the number of exoplanets for each spectral type
plt.bar(spectral_type_list,spectral_counts_list,color='skyblue')
plt.xlabel('Spectral Type')
plt.ylabel('Number of Exoplanets')
plt.title('Number of Exoplanets for Each Spectral Type')
```

```
plt.title('Number of Exoplanets for Each Spectral Type')
```

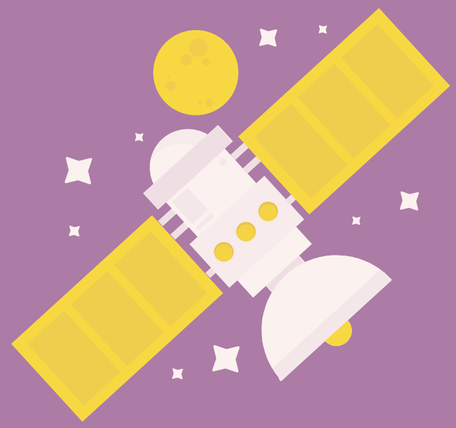
```
{'K': 55, 'G': 66, 'M': 45, 'F': 27, 'L': 1, 'A': 8, 'B': 6, 'W': 1, 'T': 1}
['K', 'G', 'M', 'F', 'L', 'A', 'B', 'W', 'T']
[55, 66, 45, 27, 1, 8, 6, 1, 1]
Text(0.5, 1.0, 'Number of Exoplanets for Each Spectral Type')
```



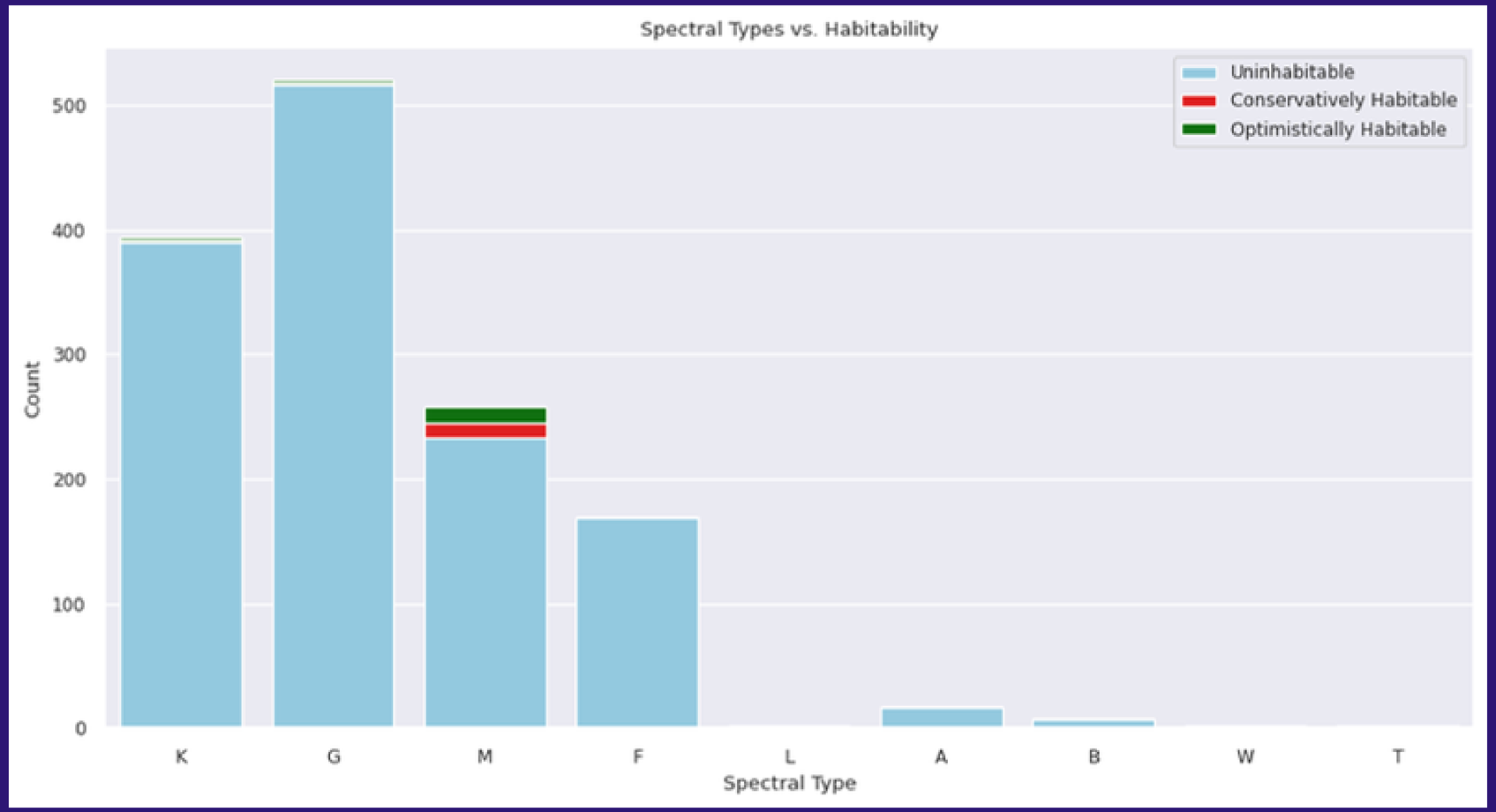
Categorical plot representing various Spectral Types and the habitability

Code

```
spectral_types = ['K', 'G', 'M', 'F', 'L', 'A', 'B', 'W', 'T']#a list of spectral types to analyze
#empty lists to store counts for different habitability categories
uninhabitable_counts = []
conservatively_habitable_counts = []
uninhabitable_counts = []
conservatively_habitable_counts = []
optimistically_habitable_counts = []
# Looping through each spectral type to count habitability categories
for spectral_type in spectral_types:
  spectral_type_df = df[df['S_TYPE_CAT'] == spectral_type]
  uninhabitable_count = len(spectral_type_df[spectral_type_df['P_HABITABLE'] == 0])
  conservatively_habitable_count = len(spectral_type_df[spectral_type_df['P_HABITABLE'] == 1])
  optimistically_habitable_count = len(spectral_type_df[spectral_type_df['P_HABITABLE'] == 2])
  uninhabitable_counts.append(uninhabitable_count)
  conservatively_habitable_counts.append(conservatively_habitable_count)
  optimistically_habitable_counts.append(optimistically_habitable_count)

plt.figure(figsize=(12, 6))
#bar plot for uninhabitable planets
sns.barplot(x=spectral_types, y=uninhabitable_counts, label='Uninhabitable', color='skyblue')
#bar plot for conservatively habitable planets and stacked it on top of uninhabitable planets
sns.barplot(x=spectral_types, y=conservatively_habitable_counts, label='Conservatively Habitable', bottom=uninhabitable_counts, color='red')
#plot for optimistically habitable planets and stack it on top of both uninhabitable and conservatively habitable planets
sns.barplot(x=spectral_types, y=optimistically_habitable_counts, label='Optimistically Habitable',
            bottom=[i + j for i, j in zip(uninhabitable_counts, conservatively_habitable_counts)], color='green')
plt.xlabel('Spectral Type')
plt.ylabel('Count')
plt.title('Spectral Types vs. Habitability')
plt.legend()
```



Graph:



2.4.2 Give reasons for the plot obtained by using the knowledge gained regarding star Spectral Types

M-type stars are red dwarfs, which are smaller and cooler than other types of stars like Sun (G-type stars) and have a longer lifespan. As a result, they have been a subject of interest in the search for habitable exoplanets, particularly in the context of potentially habitable zones. They are known to have some potential benefits for habitability, such as longer lifetimes, but they also have some challenges,like the need for planets to be much closer to the star to be in the habitable zone.

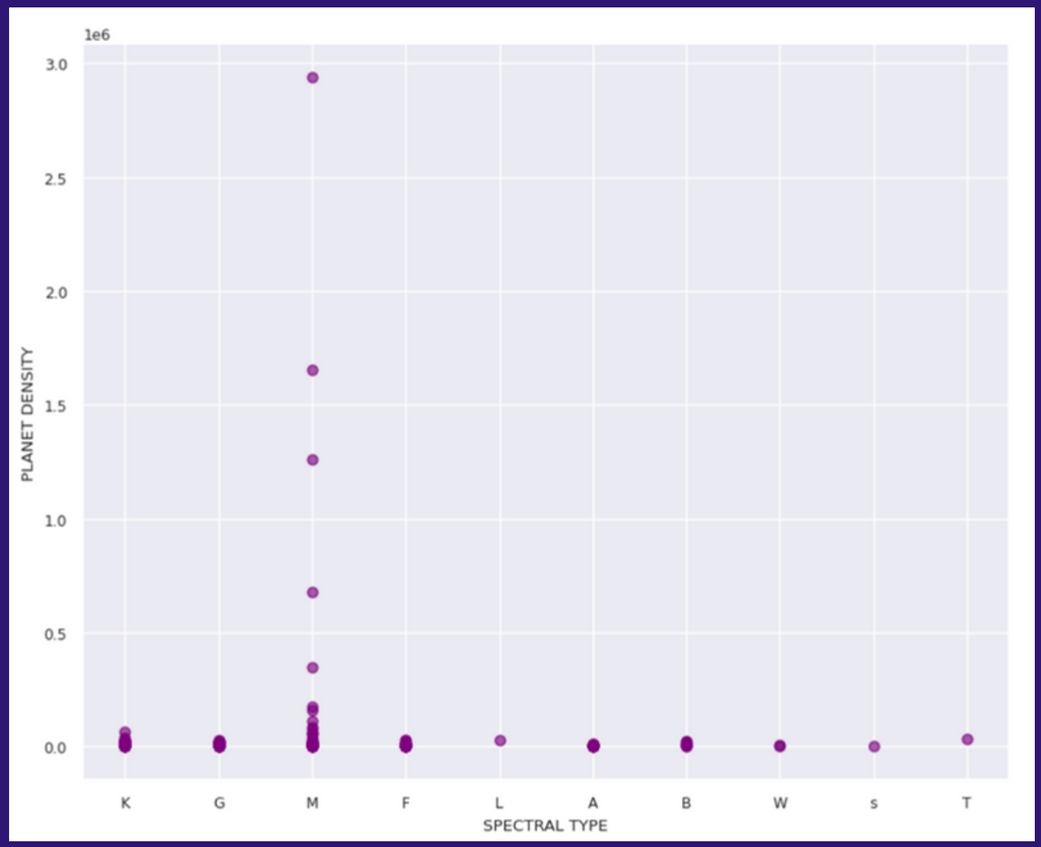
Additionally, they tend to be more active, which can lead to increased radiation and stellar flares that could impact habitability.

2.4.3 Is there a relationship between the size and density of exoplanets and the specific spectral characteristics of their host stars?

Spectral Type vs Density

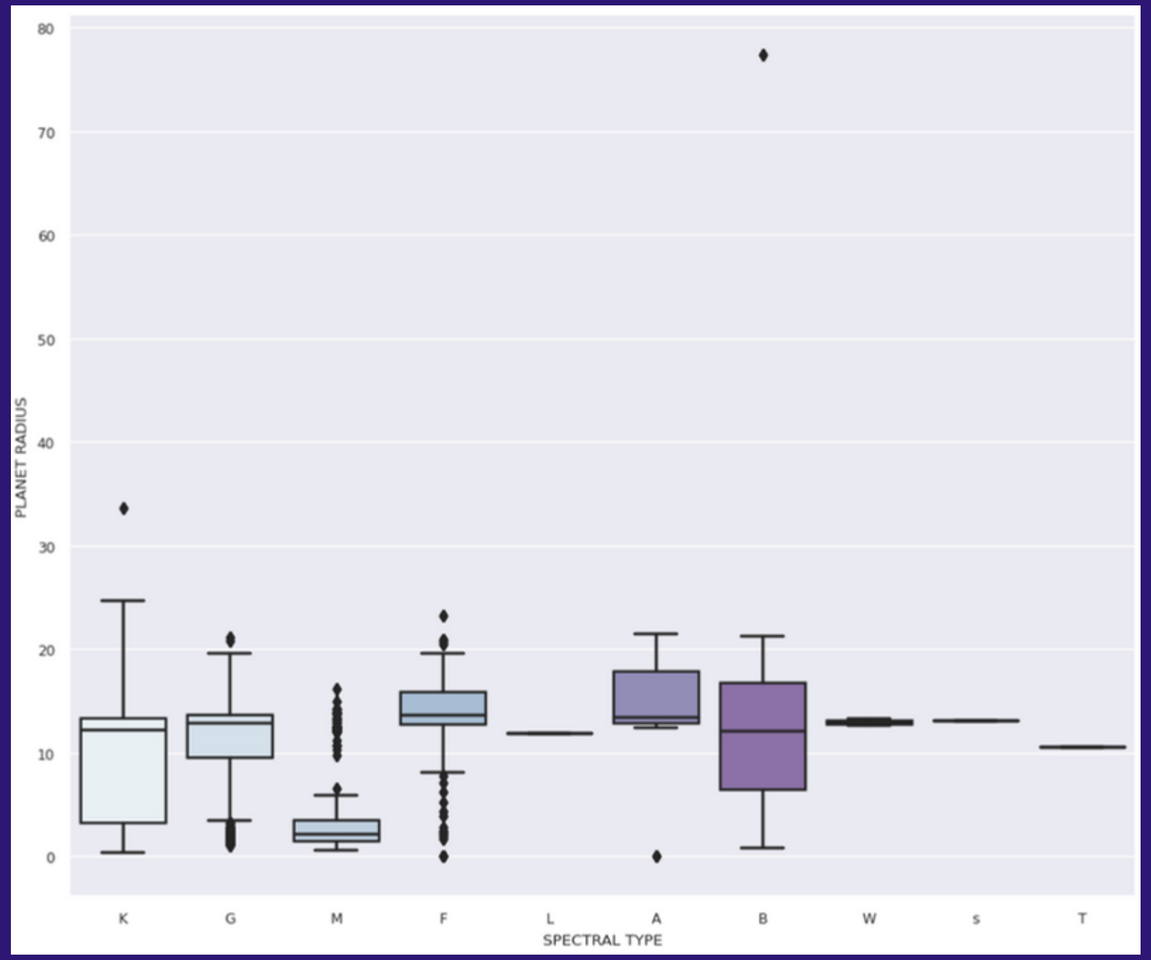
```
[ ]  # Calculated the density of exoplanets using estimated mass and radius
     # This formula assumes planets have a roughly spherical shape
     df['P_DENSITY_EST'] = (df['P_MASS_EST']*5.972e+24 ) / ((4/3) * 3.14 * ((df['P_RADIUS_EST']*6.371e+6) ** 3))
     #5.972e+24 is the conversion factor from Earth masses to kilograms.
     #((4/3) * 3.14 * ((df['P_RADIUS_EST'] * 6.371e+6) ** 3)) This calculates the volume of the exoplanets.
```

```
     density_df=df.dropna(subset=['S_TYPE_CAT', 'P_DENSITY_EST'])
     spectral_types = ['K', 'G', 'M', 'F', 'L', 'A', 'B', 'W', 'T']
     plt.figure(figsize=(10, 8))
     #a scatter plot for planet density vs. spectral type
     for i,j in enumerate(spectral_types):
       L=size_df[size_df['S_TYPE_CAT']==j]
       plt.scatter(x='S_TYPE_CAT', y='P_DENSITY_EST', data=density_df, color='purple',alpha=0.1)

     plt.xlabel('SPECTRAL TYPE')
     plt.ylabel('PLANET DENSITY')
     plt.show()
```



M dwarf stars have exoplanets with small planet radius and high density. M dwarf stars are dimmer and cooler than larger stars like the Sun. As a result, the habitable zone (the region where liquid water could exist on the surface of an orbiting planet) for M dwarf stars is much closer in.

Box Plot of Planet Radius vs Spectral Type




```
size_df=df.dropna(subset=['S_TYPE_CAT', 'P_RADIUS_EST'])
spectral_types = ['K', 'G', 'M', 'F', 'L', 'A', 'B', 'W', 'T']
plt.figure(figsize=(12, 10))
for i,j in enumerate(spectral_types):
  L=size_df[size_df['S_TYPE_CAT']==j]
  sns.boxplot(x='S_TYPE_CAT', y='P_RADIUS_EST', data=size_df, palette='BuPu')

plt.xlabel('SPECTRAL TYPE')
plt.ylabel('PLANET RADIUS')
plt.show()
```

The smaller radii of exoplanets around M dwarf stars often lead to higher densities.

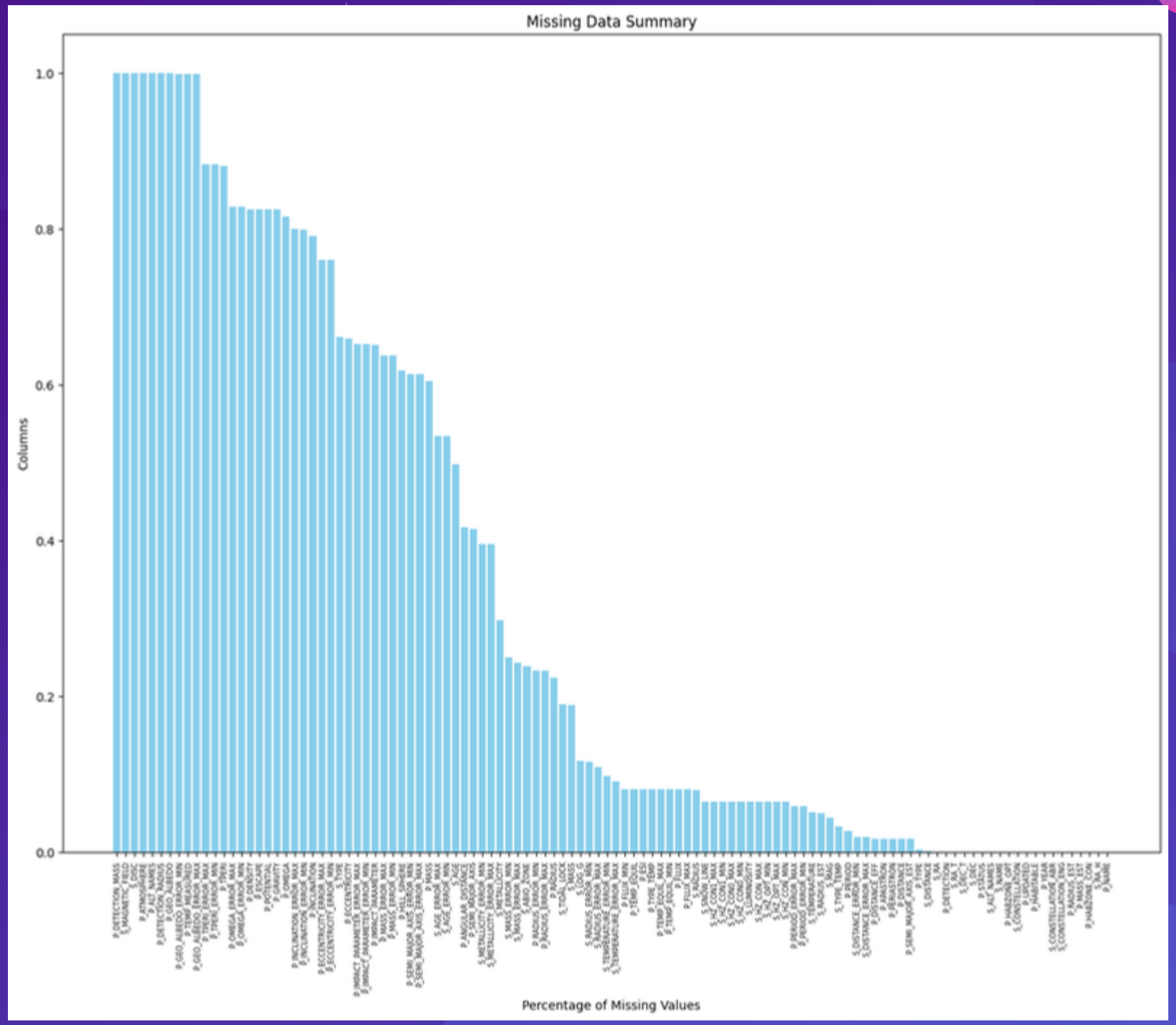
3.1 What percentage of null values exists in each feature? Visualize these percentages to identify which features have the most missing data

```
Total = df.isnull().sum()
Percent = (Total/df.isnull().count())
Null_vals = pd.concat([Total.sort_values(ascending=False), Percent.sort_values(ascending=False)], axis=1, keys=['Total', 'Percent'])
Null_vals
```

	Total	Percent
P_DETECTION_MASS	4048	1.0
P_GEO_ALBEDO	4048	1.0
S_MAGNETIC_FIELD	4048	1.0
S_DISC	4048	1.0
P_ATMOSPHERE	4048	1.0
...	...	...
S_DEC	0	0.0
P_STATUS	0	0.0
S_ALT_NAMES	0	0.0
S_DEC_T	0	0.0
P_NAME	0	0.0

112 rows × 2 columns

```
Null_vals_sorted = Null_vals.sort_values(by='Percent', ascending=False)

plt.figure(figsize=(16, 12))
plt.bar(Null_vals_sorted.index, Null_vals_sorted['Percent'], color='skyblue')
plt.xlabel('Percentage of Missing Values')
plt.ylabel('Columns')
plt.xticks(rotation=90,fontsize=6)
plt.title('Missing Data Summary')
plt.show()
```



This is a bar chart representing the percentage of missing (null) values.

3.2 Feature Reduction

```
# new df with columns having less than 60% of missing values
selected_columns = missing_percentages[missing_percentages < 60.0].index
new_df = df[selected_columns]
new_df
```

	P_NAME	P_STATUS	P_RADIUS	P_RADIUS_ERROR_MIN	P_RADIUS_ERROR_MAX	P_YEAR	P_UPDATED	P_PERIOD	P_PERIOD_ERROR_MIN	P_P
0	11 Com b	3.0	NaN	NaN	NaN	2007	2014-05-14	326.030000	-0.3200	
1	11 UMi b	3.0	NaN	NaN	NaN	2009	2018-09-06	516.219970	-3.2000	
2	14 And b	3.0	NaN	NaN	NaN	2008	2014-05-14	185.840000	-0.2300	
3	14 Her b	3.0	NaN	NaN	NaN	2002	2018-09-06	1773.400000	-2.5000	
4	16 Cyg B b	3.0	NaN	NaN	NaN	1996	2018-09-06	798.500000	-1.0000	

As seen previously , there were a lot of features with even 100% null values...to reduced dimensionality we dropped features with more than 60% null values to make reduce chance of overfitting in the classifier model

3.2.1 Given a large number of features, identify redundant or highly correlated features

```
[45] corr_threshold = 0.99
```

```
corr_matrix = new_df.corr().astype(float)

#a list to store the feature pairs with high correlation
high_corr_pairs = []
for i in range(len(corr_matrix.columns)):
    for j in range(i + 1, len(corr_matrix.columns)):
        if abs(corr_matrix.iloc[i, j]) >= corr_threshold:
            high_corr_pairs.append((corr_matrix.columns[i], corr_matrix.columns[j]))

high_corr_pairs
```

```
('S_LUMINOSITY', 'S_ABIO_ZONE'),
('S_LUMINOSITY', 'P_ESI'),
('S_HZ_OPT_MIN', 'S_HZ_OPT_MAX'),
('S_HZ_OPT_MIN', 'S_HZ_CON_MIN'),
('S_HZ_OPT_MIN', 'S_HZ_CON_MAX'),
('S_HZ_OPT_MIN', 'S_HZ_CON0_MIN'),
('S_HZ_OPT_MIN', 'S_HZ_CON0_MAX'),
('S_HZ_OPT_MIN', 'S_HZ_CON1_MIN'),
('S_HZ_OPT_MIN', 'S_HZ_CON1_MAX'),
```

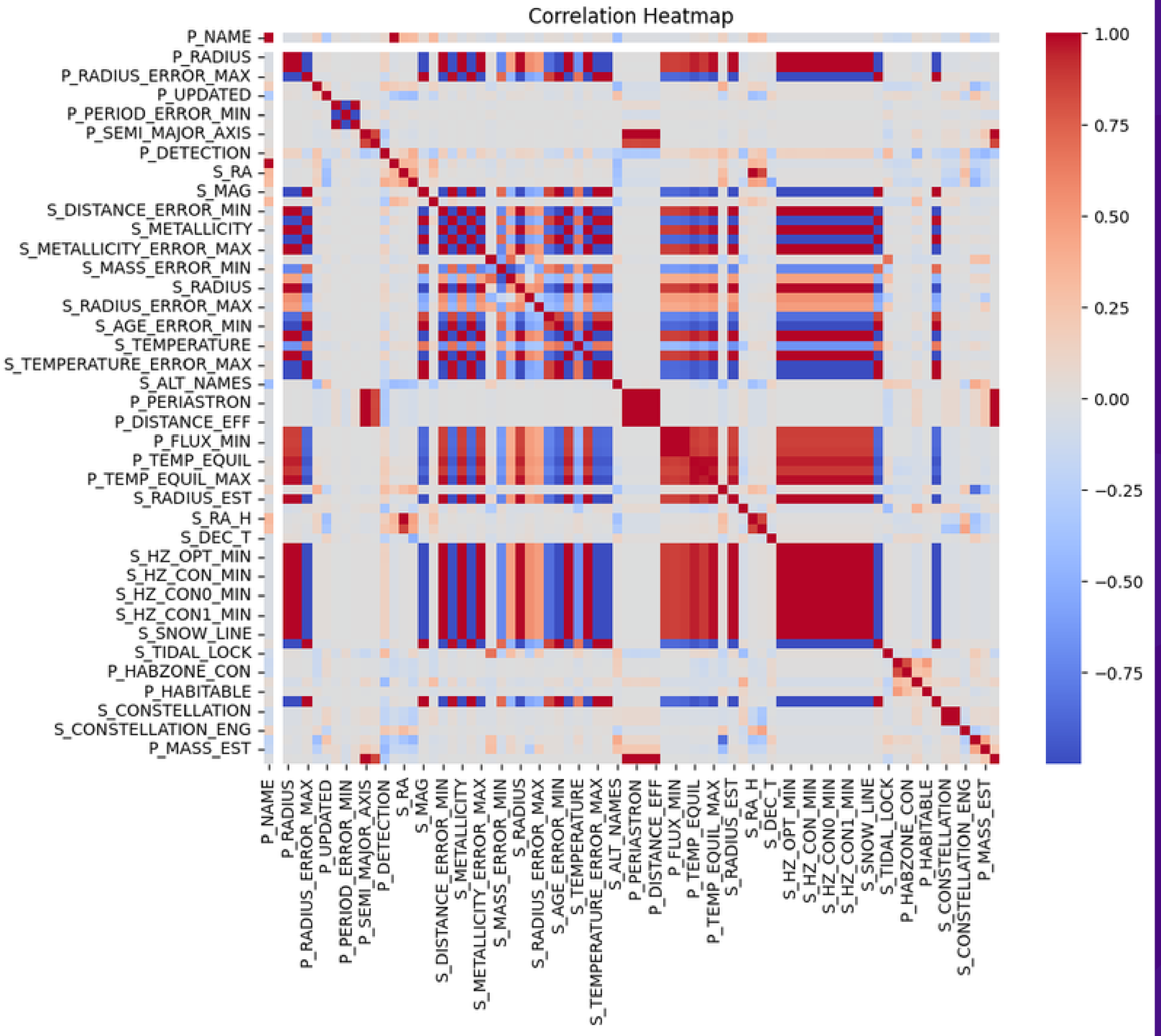
Then, We set the correlation thresold to 0.99 and calculated highly correlated features (columns) in a DataFrame to identify multicollinearity in the data.

We calculated the correlation between columns using VIF

```python
import statsmodels.api as sm
from statsmodels.stats.outliers_influence import variance_inflation_factor

numeric_columns = zahlen.select_dtypes(include=[np.number])
numeric_columns = numeric_columns.fillna(numeric_columns.mean())

numeric_columns = numeric_columns.replace([np.inf, -np.inf], np.nan)
numeric_columns = numeric_columns.dropna(axis=1, how='all')

vif_data = pd.DataFrame()

vif_data["Variable"] = numeric_columns.columns
vif_data["VIF"] = [variance_inflation_factor(numeric_columns.values, i) for i in range(numeric_columns.shape[1])]
vif_data = vif_data.sort_values(by='VIF', ascending=False)
vif_data
```

VIF Scores

index	Variable	VIF
82	S_ABIO_ZONE	3.079156163711413
10	P_PERIOD_ERROR_MAX	0.9996699949576582
9	P_PERIOD_ERROR_MIN	0.9996528505073621
8	P_PERIOD	0.9995981190300403
65	P_FLUX_MIN	0.9969600198508157
64	P_FLUX	0.9968334368122473
66	P_FLUX_MAX	0.9965498810789751
61	P_PERIASTRON	0.9960668011525058
90	P_SEMI_MAJOR_AXIS_EST	0.9958141537879923
63	P_DISTANCE_EFF	0.9957920801353303



We also
plotted a correlation heatmap to
to identify redundant features.

And dropped those highly correlated features to form a cleaner dataset consisting of 36 features

```
[51] new_df = new_df.drop(['S_NAME', 'P_RADIUS', 'P_RADIUS_ERROR_MIN', 'P_RADIUS_ERROR_MAX', 'P_DISTANCE', 'P_PERIASTRON', 'P_APASTRON',
                           'P_DISTANCE_EFF', 'P_FLUX_MIN', 'P_FLUX_MAX', 'P_TEMP_EQUIL', 'P_TEMP_EQUIL_MIN', 'P_TEMP_EQUIL_MAX',
                           'S_RADIUS_EST', 'S_RA_H', 'S_RA_T', 'S_LUMINOSITY', 'S_HZ_OPT_MIN', 'S_HZ_OPT_MAX', 'S_HZ_CON_MIN',
                           'S_HZ_CON_MAX', 'S_HZ_CON0_MIN', 'S_HZ_CON0_MAX', 'S_HZ_CON1_MIN', 'S_HZ_CON1_MAX', 'S_SNOW_LINE',
                          'P_PERIOD_ERROR_MIN', 'P_PERIOD_ERROR_MAX', 'S_MAG', 'S_DISTANCE_ERROR_MIN', 'S_DISTANCE_ERROR_MAX',
                           'S_METALLICITY', 'S_METALLICITY_ERROR_MIN', 'S_METALLICITY_ERROR_MAX', 'S_AGE', 'S_TEMPERATURE_ERROR_MIN',
                           'S_TEMPERATURE_ERROR_MAX', 'S_ABIO_ZONE', 'P_ESI', 'S_CONSTELLATION_ABR', 'P_SEMI_MAJOR_AXIS_EST'], axis=1)
```

```
[52] new_df.columns
```

```
Index(['P_NAME', 'P_STATUS', 'P_YEAR', 'P_UPDATED', 'P_PERIOD',
       'P_SEMI_MAJOR_AXIS', 'P_ANGULAR_DISTANCE', 'P_DETECTION', 'S_RA',
       'S_DEC', 'S_DISTANCE', 'S_MASS', 'S_MASS_ERROR_MIN', 'S_MASS_ERROR_MAX',
       'S_RADIUS', 'S_RADIUS_ERROR_MIN', 'S_RADIUS_ERROR_MAX',
       'S_AGE_ERROR_MIN', 'S_AGE_ERROR_MAX', 'S_TEMPERATURE', 'S_LOG_G',
       'S_ALT_NAMES', 'P_FLUX', 'P_TYPE', 'S_TYPE_TEMP', 'S_DEC_T',
       'S_TIDAL_LOCK', 'P_HABZONE_OPT', 'P_HABZONE_CON', 'P_TYPE_TEMP',
       'P_HABITABLE', 'S_CONSTELLATION', 'S_CONSTELLATION_ENG', 'P_RADIUS_EST',
       'P_MASS_EST'],
      dtype='object')
```

3.2.2 Choose an appropriate feature reduction method for this dataset. Expatiate the premise behind choosing that method

PCA can be used for dimensionality reduction.By focusing on the principal components (linear combinations of the original features) that capture the most variance, PCA can effectively reduce the impact of noise or irrelevant information in the data.

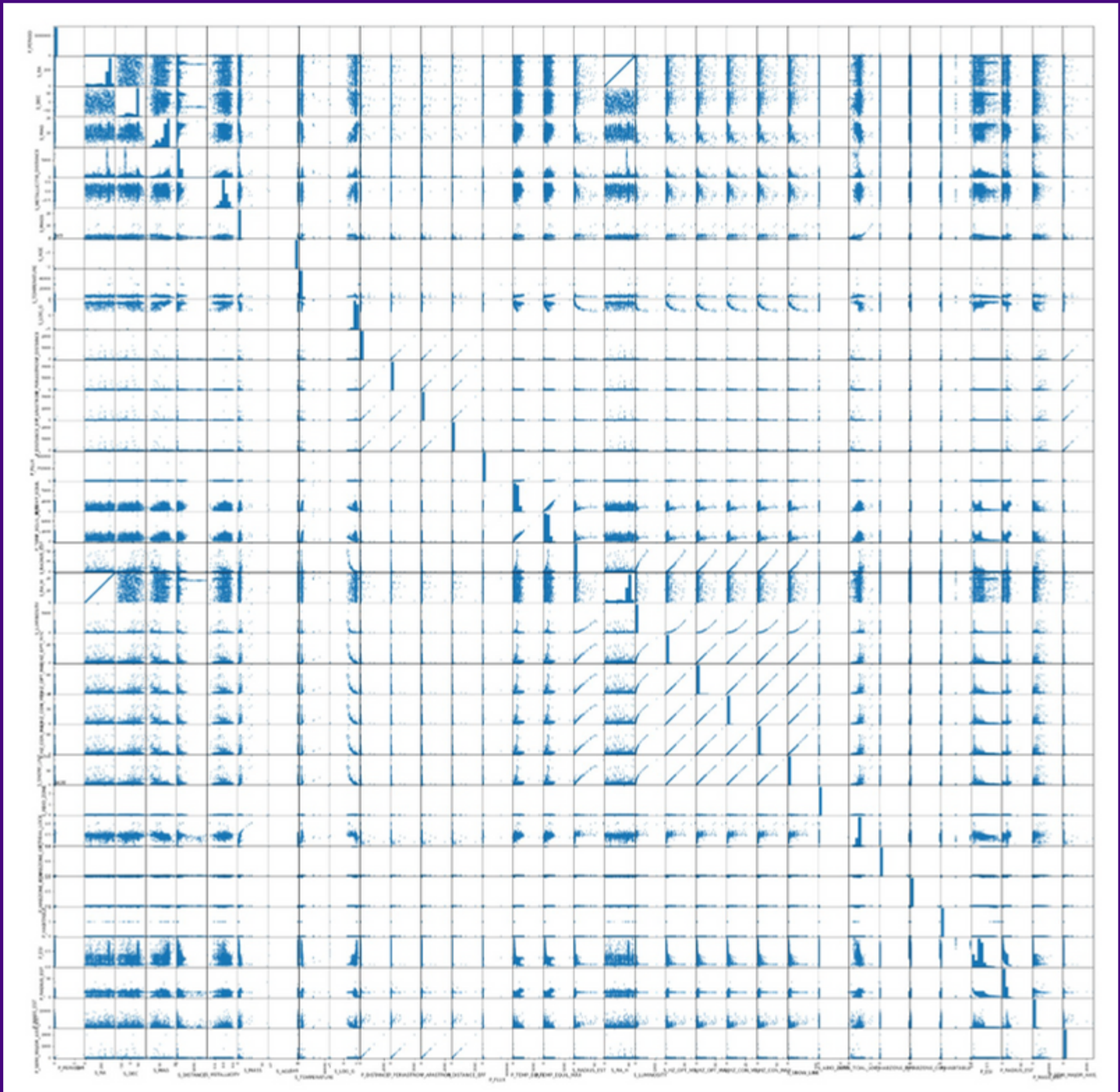
Alternatively,Mutual Information can be used to assess the relevance of individual features (independent variables) with respect to a target variable (dependent variable).In the context of feature selection, it measures how much knowing the value of one feature reduces uncertainty about the target variable.High Mutual Information between a feature and the target variable suggests that the feature is informative and relevant for predicting the target. Features with low MI may be less relevant.

3.2.3 Describe the relationship between various features before and after feature reduction by creating scatter plots and identifying changes in the distribution of data, patterns and clusters

We plotted pairplot plots of the features after cleaning the data

Code:

```
columns=['P_PERIOD', 'S_RA', 'S_DEC', 'S_MAG', 'S_DISTANCE', 'S_METALLICITY',
       'S_MASS', 'S_AGE', 'S_TEMPERATURE', 'S_LOG_G', 'P_DISTANCE',
       'P_PERIASTRON', 'P_APASTRON', 'P_DISTANCE_EFF', 'P_FLUX',
       'P_TEMP_EQUIL', 'P_TEMP_EQUIL_MAX', 'P_TYPE', 'S_RADIUS_EST', 'S_RA_H',
       'S_LUMINOSITY', 'S_HZ_OPT_MIN', 'S_HZ_OPT_MAX', 'S_HZ_CON_MIN',
       'S_HZ_CON_MAX', 'S_SNOW_LINE', 'S_ABIO_ZONE', 'S_TIDAL_LOCK',
       'P_HABZONE_OPT', 'P_HABZONE_CON', 'P_TYPE_TEMP', 'P_HABITABLE', 'P_ESI',
       'P_RADIUS_EST', 'P_MASS_EST', 'P_SEMI_MAJOR_AXIS_EST']
pd.plotting.scatter_matrix(df[columns], alpha=0.5, figsize=(40, 40), diagonal='hist')
plt.show()
```



We can find linear dependence between columns like P_DISTANCE, P_PERIASTRON, P_APASTRON, P_DISTANCE_EFF, S_HZ_OPT_MIN, S_HZ_OPT_MAX, S_HZ_CON_MIN, S_HZ_CON_MAX, S_SNOW_LINE

The presence of small clusters at the bottom signify outliers in the dataset.

We find inverse correlation between P_ESI and S_HZ_OPT_MIN, S_HZ_OPT_MAX, S_HZ_CON_MIN, S_HZ_CON_MAX, S_SNOW_LINE

Majority of columns show non-linear relationships between each other which is good for our dataset specially for classification.

3.3 Apply a suitable imputation technique to fill the null values of the dataset. Clarify your choice of technique used

```
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer

X = data_clean.drop(["P_HABITABLE",],axis=1)
Y = data_clean["P_HABITABLE"]

imputer = IterativeImputer()
imputer.fit(X)
new = imputer.transform(X)
X.replace(X.values,new,inplace=True)
```

- We have used the Iterative Imputer to fill the large number of missing values with intricate correlations to other features.Iterative imputer is a multivariate imputation technique, which means it considers relationships between variables when imputing missing values. This can be especially useful when the missingness in one variable is related to the values of other variables in the dataset.

4. Habitability classification:

4.1 Build a robust and efficient classifier for classifying a new exoplanet into the three classes of habitability namely, i) Uninhabitable planets (0) ii) Conservatively habitable planets (1) iii) Optimistically habitable planets (2) by utilizing the target features. Implement K-Fold Cross Validation for training. Train the dataset for all values of K from 2-10. Plot the loss and accuracy versus epochs for these K values.

Reason for our final choice of model:

We have used the LightGBMClassifier to build the exoplanet classifier. We did this as the LGBM model is robust to unencoded categorical features.It uses a histogram-based approach for tree construction, which reduces memory usage and speeds up training considerably.

Feature selection for our model

- 10 best features for our model

Selected Features:
Index(['P_DETECTION', 'S_MASS', 'S_LOG_G',
'P_TYPE', 'S_TYPE_TEMP',
'P_HABZONE_OPT', 'P_HABZONE_CON',
'P_TYPE_TEMP', 'P_RADIUS_EST',
'P_MASS_EST'],
dtype='object')

We used Recursive Feature Elimination (RFE) with Linear Regression as the base estimator for feature selection.
RFE is a backward selection method, meaning it starts with all features and progressively removes the least important ones until the desired number of features is retained.

```
from sklearn.feature_selection import RFE
from sklearn.linear_model import LogisticRegression
```

```
X = new_df.drop("P_HABITABLE", axis=1)
y = new_df['P_HABITABLE']
```

```
estimator = LogisticRegression(max_iter=10000)
rfe = RFE(estimator, n_features_to_select=10)

rfe.fit(X, y)
selected_features = X.columns[rfe.support_]

print(selected_features)
```

Data Preprocessing for the model

As mentioned before, we performed Null value processing , Dropped redundant features and performed feature selection for 10 best features.

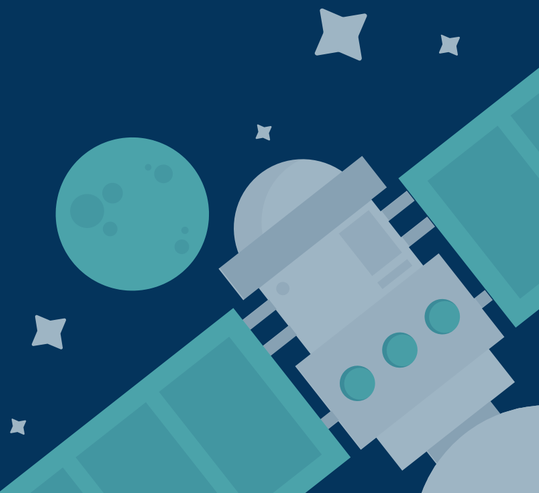
Performed Label Encoding for Categorical Columns

```
#label encoding for categorical columns
categorical_columns = features.select_dtypes(include=['object']).columns.tolist()
label_encoders = {}

for col in categorical_columns:
    le = LabelEncoder()
    features[col] = le.fit_transform(features[col])
    label_encoders[col] = le
```

Did iterative imputing for Null value handling and Used min max scaler to normalize the data

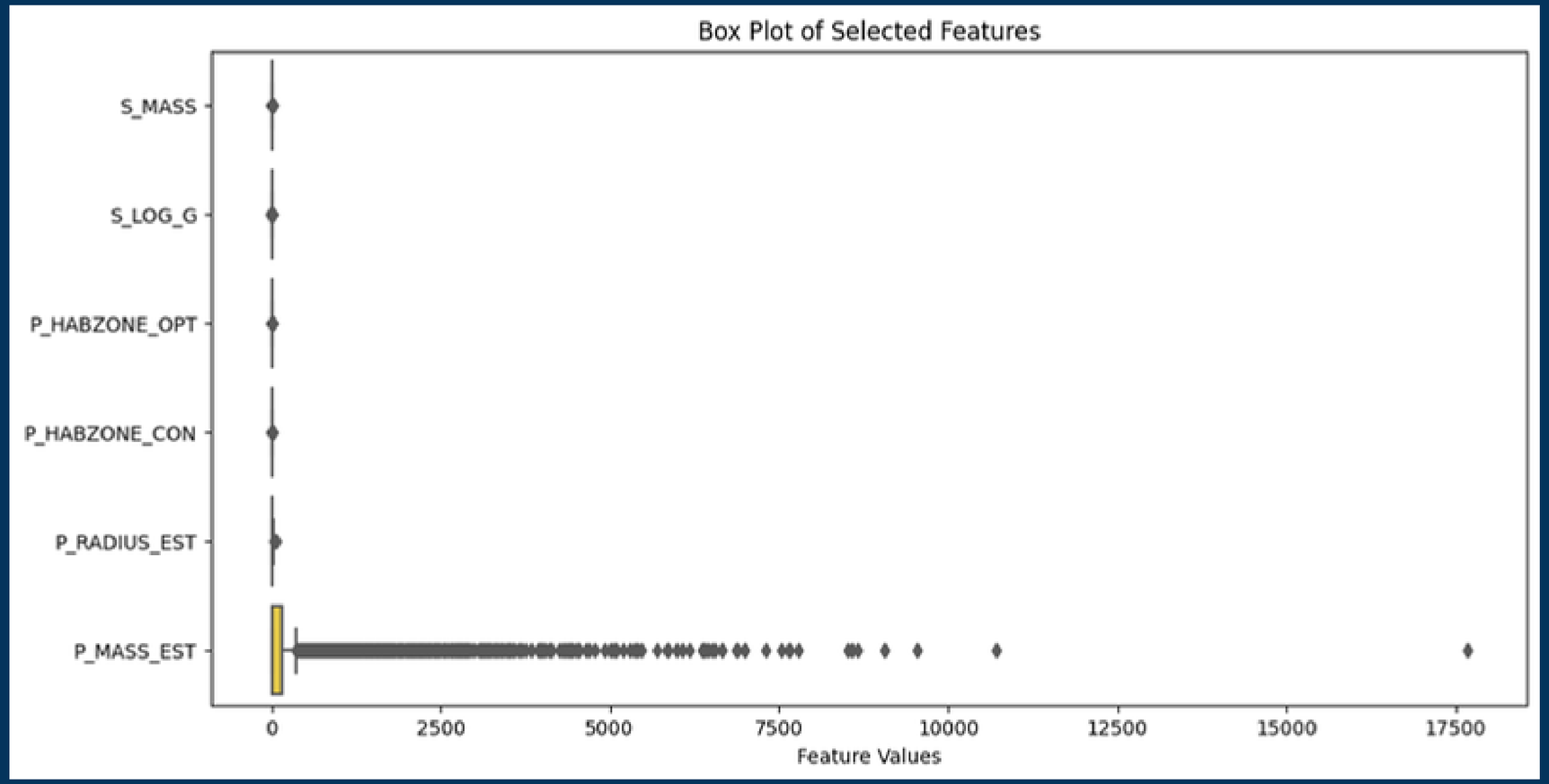
```
#Iterative Imputer
imputer = IterativeImputer(max_iter=10, random_state=0)
imputed_features = imputer.fit_transform(features)
```

```
# Scale the features
scaler = MinMaxScaler()
scaled_features = scaler.fit_transform(imputed_features)
```



Data Preprocessing for the model

Box plots for Outlier Detection and Removal for the selected features

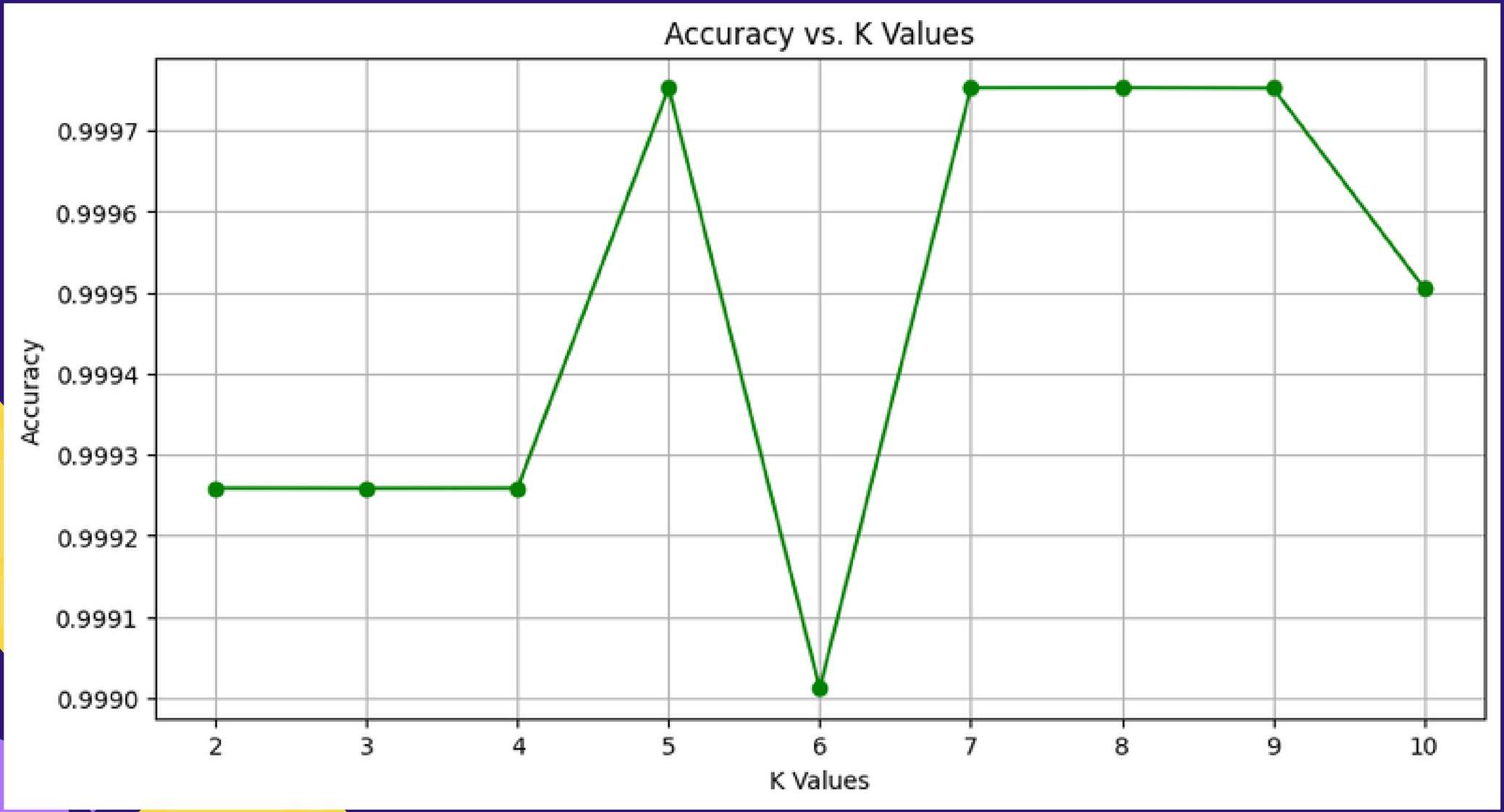


Exoplanet Classifier (LGBM + K-Fold Cross Validation)

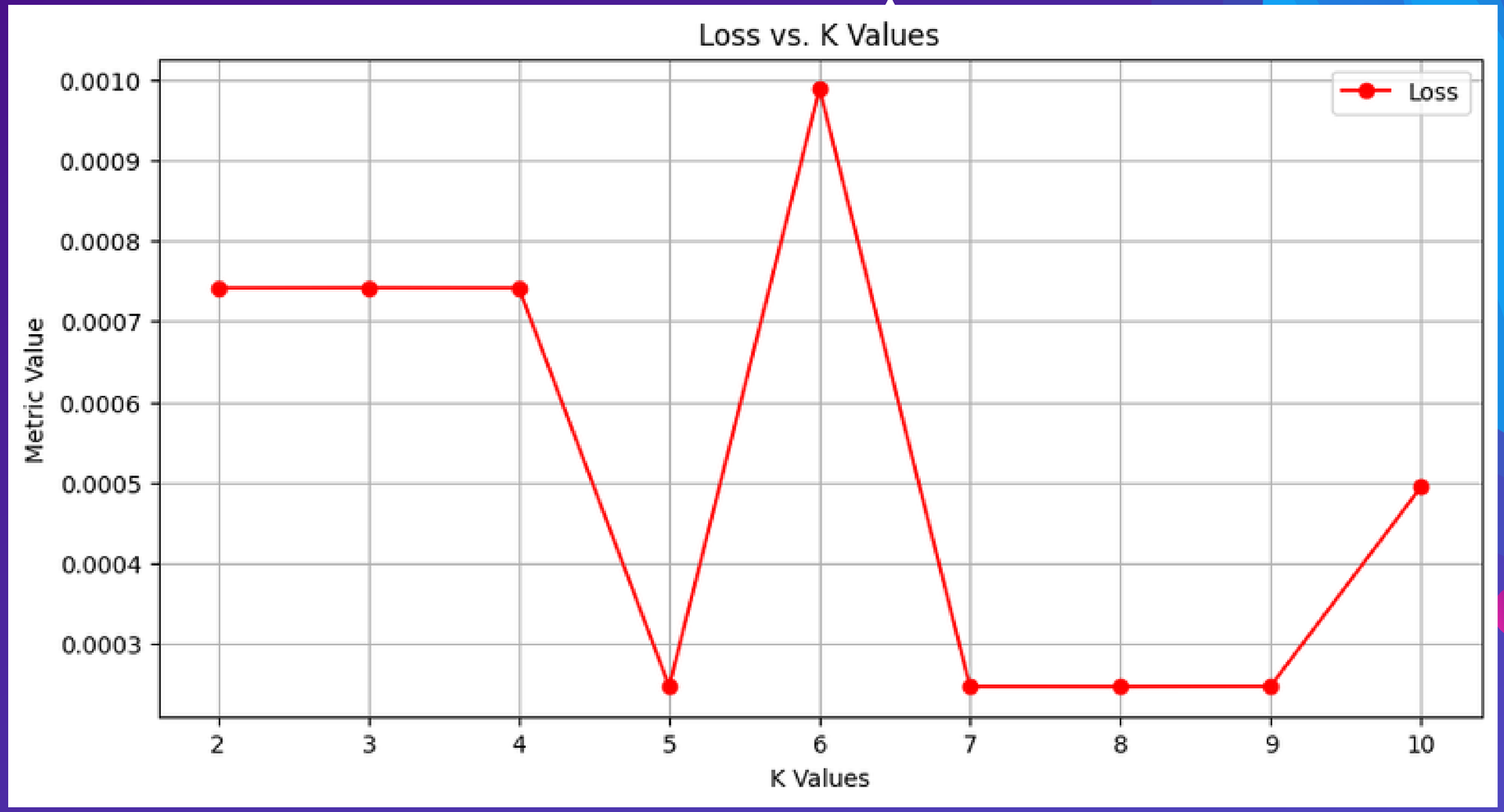
```
[73] k_values = range(2, 11)
     losses = []
     accuracies = []

     #K-Fold Cross Validation
     for K in k_values:
         skf = StratifiedKFold(n_splits=K, shuffle=True, random_state=42)
         fold_losses = []
         fold_accuracies = []

         for train_index, val_index in skf.split(X, y):
             X_train, X_val = X[train_index], X[val_index]
             y_train, y_val = y.iloc[train_index], y.iloc[val_index]

             #SMOTE
             smote = SMOTE(sampling_strategy='auto', random_state=42)
             X_resampled, y_resampled = smote.fit_resample(X_train, y_train)

             #ENN
             enn = EditedNearestNeighbours(sampling_strategy='auto')
             X_resampled, y_resampled = enn.fit_resample(X_resampled, y_resampled)

             #LightGBM model
             lgbm_model = LGBMClassifier(random_state=42)
             lgbm_model.fit(X_resampled, y_resampled)
             y_pred = lgbm_model.predict(X_val)
             accuracy = accuracy_score(y_val, y_pred)
             fold_accuracies.append(accuracy)
             #mse
             mse = mean_squared_error(y_val, y_pred)
             fold_losses.append(mse)

         accuracies.append(np.mean(fold_accuracies))
         losses.append(np.mean(fold_losses))
```

We did K-Fold Cross Validation with varying K values (2 to 10) on our dataset. For each fold, we used Synthetic Minority Over-sampling Technique (SMOTE) to counter class imbalance and Edited Nearest Neighbors (ENN) to enhance majority class samples. A LightGBM model was trained in each fold, and we evaluated its performance using accuracy and mean squared error (MSE). This iterative process allowed us to assess model consistency across different data splits and refine it further to improve overall predictive performance.

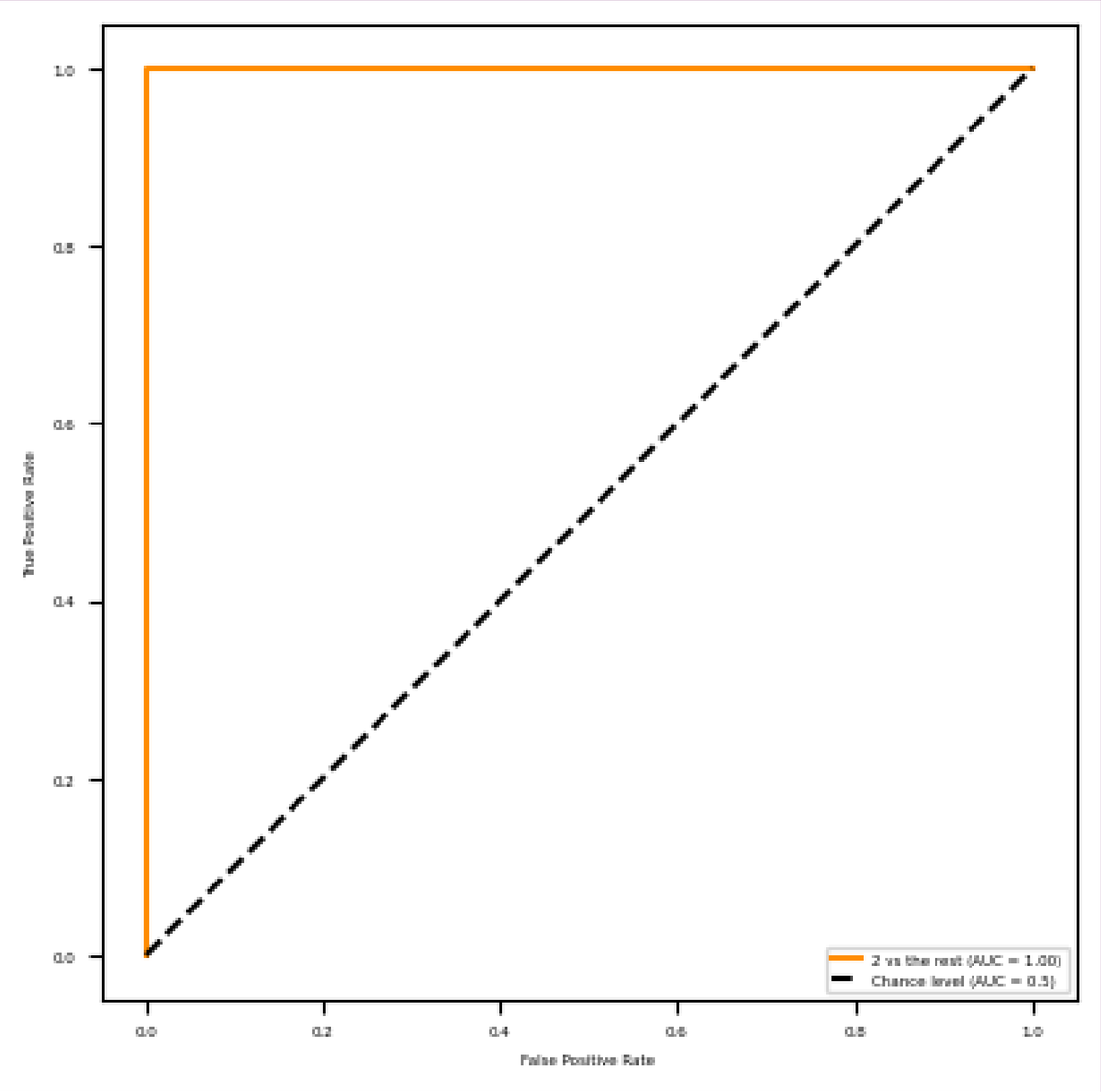
Accuracy vs Epochs Graph:



Loss Vs Epochs:



4.2 Plot the ROC curve and Confusion Matrix to quantify the performance of your classifier.



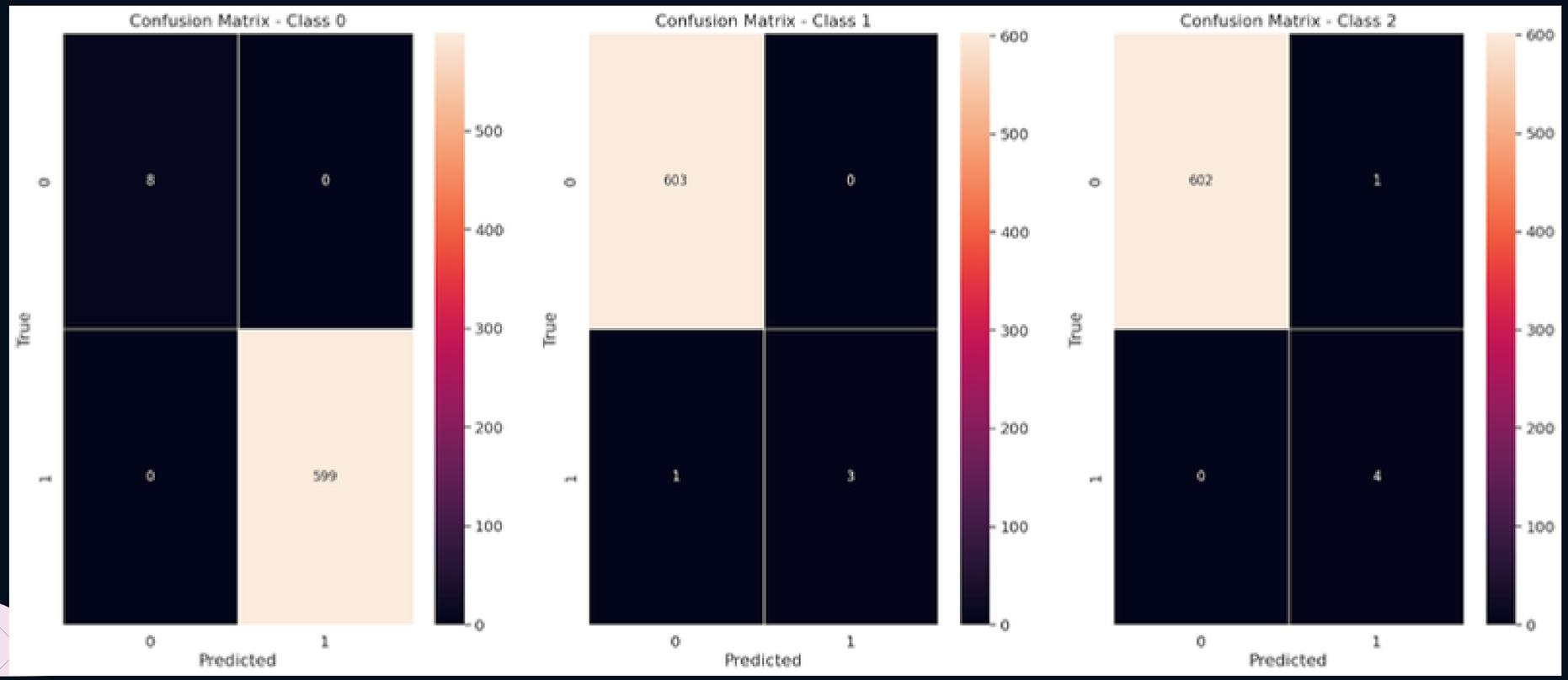

```
RocCurveDisplay.from_predictions(
    Y_onehot_test[:, class_id],
    model.predict_proba(X_test)[:, class_id],
    name=f"{class_of_interest} vs the rest",
    color="darkorange",
    plot_chance_level=True,
)
plt.axis("square")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
```

The Receiver Operating Characteristic (ROC) curve is a graphical representation of a binary classifier's performance, displaying the trade-off between true positive rate and false positive rate across various thresholds. Higher AUROC meaning better model

Confusion Matrix

A confusion matrix in classification is a 2x2 table that represents the performance of a machine learning model. It quantifies the number of true positives (correctly predicted positive class), true negatives (correctly predicted negative class), false positives (actual negative but predicted positive), and false negatives (actual positive but predicted negative).

```python
from sklearn.metrics import multilabel_confusion_matrix
class_labels = ["Class 0", "Class 1", "Class 2"]
cm=multilabel_confusion_matrix(Y_test,Ypre)
sns.set(font_scale=1.3)
fig, axes = plt.subplots(1, 3, figsize=(25, 10))
for i, (label, matrix) in enumerate(zip(class_labels, cm)):
    sns.heatmap(matrix, annot=True, linewidths=0.5, linecolor="white", fmt=".0f", ax=axes[i],annot_kws={"size": 12})
    axes[i].set_title(f"Confusion Matrix - {label}")
    axes[i].set_xlabel("Predicted")
    axes[i].set_ylabel("True")
```



4.3 Optimize all the hyperparameters used in the classifier by choosing an appropriate optimization method. Also, explain the premise behind choosing the optimization method.

We have used RandomizedSearchCV here to find the optimal hyperparameters from the given choices. RandomizedSearchCV is a technique for hyperparameter tuning in machine learning using random search instead of exhaustive grid search. The primary goal is to find a set of hyperparameters that optimize the performance of your model without trying every possible combination, which can be computationally expensive.

Random Search CV is often computationally more efficient than Grid Search CV.

Random Search handles high-dimensional spaces more effectively since it doesn't require evaluating all possible combinations.

Randomized Search CV

```python
param_dist = {
    'n_estimators': [100, 200, 300],
    'max_depth': [5, 10, 15],
    'learning_rate': [0.01, 0.1, 0.2],
    'subsample': [0.8, 0.9, 1.0],
}

# Randomized Search CV
random_search = RandomizedSearchCV(
    LGBMClassifier(random_state=42),
    param_distributions=param_dist,
    n_iter=10,
    scoring='roc_auc',
    cv=3,
    n_jobs=-1,
    random_state=42,
)
```

BEST PARAMETERS

```
[77] random_search.fit(X_resampled, y_resampled)

     #best parameters
     best_params = random_search.best_params_
     print("Best Parameters:", best_params)
```

Best Parameters:
{'subsample': 0.8,
'n_estimators': 200,
'max_depth': 5,
'learning_rate': 0.1}


THANK
YOU!