

Team RavBhar






IIT TIRUPATI

.....

Data Analytics



Table of Contents

Acknowledgments.....	3
♥ Our Team ♥	4
Google Colab Link 	5
Libraries Used 	5
1. Visualization and Analysis of Data 	6
2. Interpretation and calculation of physical parameters 	17
3. Feature Engineering.....	28
4. Habitability Classification 	35
5. Additional Notes.....	40

Acknowledgments

It is essential to acknowledge the support of our college's astronomy club (Gagan Vedhi), who informed us of this competition in the first place.

We also thank our college, IIT Tirupati, for extending out their support and offering us the computer lab for our late night efforts.

Next, we offer our thankfulness and respect to our parents for blessing us with good luck and knowledge for our ventures and exploration in such competitions, programs, etc.

Thank you.

♥ Our Team ♥

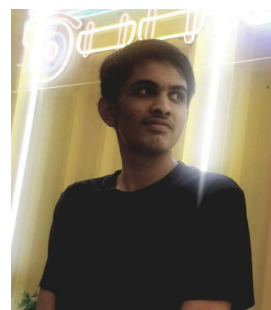


Ritvik Ravi 🧠 (Team Lead & Business Logic Developer)

I am deeply interested in aeronautics, aerospace, and in realizing how modern IT and technological advancements can be applied in space technology.

💻 Suriyaa MM (Lead Developer)

Passionate about connecting silicon to software.
I am specifically interested in hardware-accelerated Deep Learning Techniques and Algorithms



Sudhanva Bharadwaj BM 💻 (Lead Data Analyst & Developer)



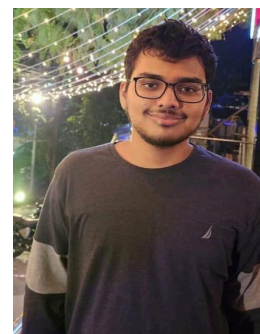
I am interested in solving machine learning and deep learning problems.

I am fascinated by how algorithms can learn patterns in data and their applications in real-world problems like object recognition and Generative AI.

Adithya Ananth 💻 (Data Analyst and Developer)

I am interested in algorithmic problem solving using AI and ML.

As a competitive programmer, my main interest lies in applying efficient data structures and algorithms to solve complex problems.



Google Colab Link

 NSSEC.ipynb

(<https://colab.research.google.com/drive/1BocCrRLRRqEkfEgkJaaEVobdGqd2YkDF?usp=sharing>)

Libraries Used

We have used Python as our programming language due to its versatility and better visualization of the data.

The external libraries we have used-

[Pandas](#)

[Seaborn](#)

[NumPy](#)

[Matplotlib](#)

[Scikit-learn](#)

1. Visualization and Analysis of Data

What is Data Visualization?

Data visualization is the graphical representation of data. It transforms data into a visual format that can be easily understood and interpreted. Data visualization can be used to communicate data to others, to identify patterns and trends in data, and to make better decisions.

1.1 Inspecting the Data and Extracting Basic Information

1.1.1 Range, Mean, Median, and Standard deviation of the dataset

Python

```
# Creating a directory for storing different files used to store data
if not os.path.exists("./Question1/"):
    os.mkdir("./Question1/")
# Writes the median to Median.csv file
DataSet.median(numeric_only=True).dropna().to_csv("Question1/Median.csv")
# Writes the standard deviation to StandardDeviation.csv file
DataSet.std(numeric_only=True).dropna().to_csv("Question1/StandardDeviation.csv")
# Writes the mean to Mean.csv file
DataSet.mean(numeric_only=True).dropna().to_csv("Question1/Mean.csv")
# Dictionary for storing range values of respective columns
RangeDataSet = {}
# Logic for finding range and writing it to above given dictionary
for Columns in DataSet.select_dtypes(include=np.number).columns.tolist():
    RangeDataSet[Columns] = DataSet[Columns].max() - DataSet[Columns].min()
# Converting the above dictionary into a panda's dataframe for visualizing as CSV file
RangeDataFrame = pd.DataFrame.from_dict(RangeDataSet, orient="index")
# Writes the range to Range.csv file
RangeDataFrame.dropna().to_csv("Question1/Range.csv")
```

The above code is intended to extract the mean, range, median, and standard deviation from the given dataset using the functions `DataSet.mean`, `DataSet.Median`, `DataSet.std`, and then putting the result into separate CSV files for complete output and effortless ability for in-depth analysis.

 [Question_1.1.1](#) [Link To The CSV Files mentioned above]

1.1.2 Does the Dataset require normalization?

YES ✓

Normalization refers to the simplification of given data such that it is easier for a machine learning mechanism to understand and apply the data more effortlessly.

One such method for normalization is **common scaling** - which involves ranging all the attributes between -1 and 1. So that while these are all on a relative scale, it is easier for the model to compare between values.

Implementation:

Python

```
# Using the Scikit-Learn MinMax Scaler for Normalizing
Normalizer = sk.preprocessing.MinMaxScaler()
# Converting into Pandas DataFrame
NormalizedDataFrame = pd.DataFrame(Normalizer.fit_transform(DataSet[DataSet.select_dtypes(include =
np.number).columns.tolist()]), columns = DataSet.select_dtypes(include = np.number).columns.tolist())
# Exporting to CSV File
NormalizedDataFrame.to_csv("Normalized_Exoplanet_Catalog.csv")
```

 [Question_1.1.2](#) [Link To The CSV Files Mentioned Above]

In the above code, we have normalized the values by using common scaling on all the values.

1.1.3 What are your inferences based on the above results? 🤖

Looking at the CSV file that displays the output of the normalized dataset, we can see that the values seem more relatable and on a common scale.

For example:

if the ML model were to compare the relation between **S_MAG**, **S_AGE**, and **S_TEMPERATURE** to see if they are somehow related to each other,

it would be best to put all these three into a common scale of -1 to 1 (where -1 is the least, 1 is the maximum) to get a correlation value that is also numerical for a better analysis. Also, it is computationally inexpensive.

Additionally, another case where the normalization of data would help is in the case of comparison of errors. A parameter 'A' could have an extensive range of values (say between 10-10000) and have a slight error of 2% compared to the range, which would give 200. However, parameter 'B' could have values in the range of 2-80, have a significant error % compared to the range (say 20%), and still have a lesser error, about 16. A scale of 0-1 solves the problems of relative comparison, like the errors and measures of central tendency.

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

Python

```
# Using matplotlib to set the Figure Size (In This Case Image Size)
plt.figure(figsize=(60,30))
sns.set_context(font_scale = 6.0)

# Tick values for SideMap
TickArray = []
# TickArray has 0, 100, 200 ..... 1400
for i in range(0,1400, 100):
    TickArray.append(i)

# CNT feature is added to DataSet and values for each row is set has 1.
# We need to sum up the number of planets detected by a method in each year
# so we need a sum of the number of planets
# to sum up we assigned each planet's count as 1 and then add accordingly
DataSet['CNT'] = 1

# Trimming the dataset such that only these three columns are there
TrimmedDataSet = DataSet[['P_YEAR', 'P_DETECTION', 'CNT']]

# Customization for sidemap of heatmap
CBARCustomizationDict = {"shrink" : 1, "ticks" : TickArray}

# A pivot table is created and stored in the PivotedDataSet
# Pivot table accepts the data set and reorganizes selected columns and rows of the
# data accepting desired columns,rows,values and filters
# all null values are filled with zero
PivotedDataSet = pd.pivot_table(TrimmedDataSet, index='P_DETECTION', columns='P_YEAR',
values='CNT', aggfunc="sum").fillna(0)

# Sets the plot title
plt.title("Various Detection Methods Used Over Years")

# Plots the actual heatmap.
# cbar_kws is the kwarg for Customizing sidemap
```

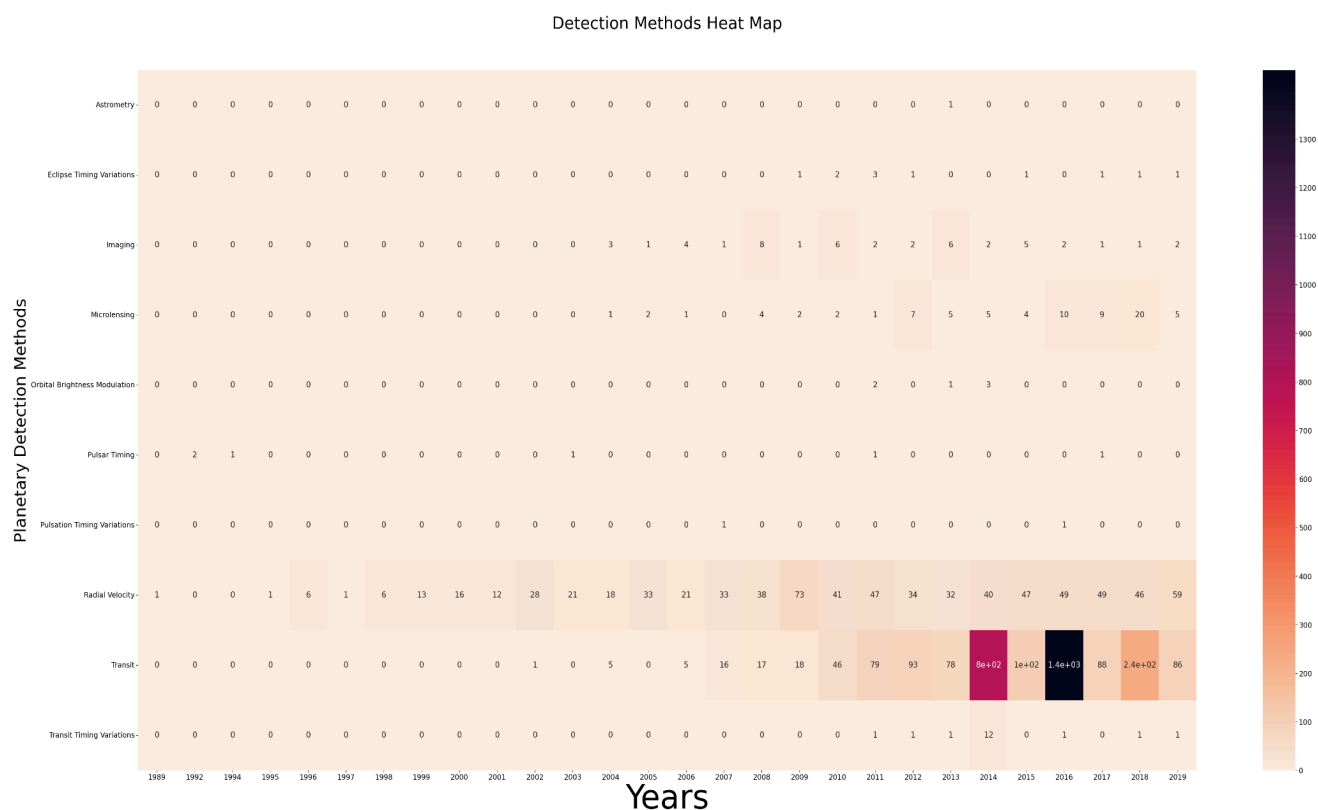


```
# cmap is kwarg for Colour Map
# annot is kwarg for setting values on the each square in heatmap
sns.heatmap(PivotedDataSet, cmap="rocket_r", cbar_kws= CBARCustomizationDict,annot=True)
# Y-Axis Label
plt.ylabel("Planetary Detection Methods")
# X-Axis Label
plt.xlabel("Years")
# Saving the plot to DetectionMethodsHeatMap.png
plt.savefig("DetectionMethodsHeatMap.pdf")
```

[Question 1.2](#) [Link To The PDF Mentioned Above]

Here, we use the libraries of matplotlib and seaborn to make an interactive graph that shows a heatmap of the various planetary detection methods used over the years

Check Below For Output



1.2.1 What do you infer from the above heatmap?

We see that most of the exoplanets have been found after 2011, and very few exoplanets were found earlier. Additionally, there is a large density of planets found in 2014-18.

Regarding the planetary detection methods that have been used, the **transit method** is the most used method to find exoplanets. This is because the transit method has several **advantages**, such as

- i) It does not depend on the gravitational influence of the star, so planets with extreme eccentricities are not affected, unlike methods like radial velocity methods that are more accurate for planets that have low eccentricities.
- ii) It is a precise method that allows for measurements of properties of the planet, such as the size of the planet, orbital period, and sometimes even its atmospheric properties.

1.3 Identify the planetary detection methods that have identified the most:

- 1) Uninhabitable planets (0)
- 2) Conservatively habitable planets (1)
- 3) Optimistically habitable planets (2)

Implementation:


```
Python
# Trimming the dataset such that only these three columns are there
TrimmedDataSet2 = DataSet[['P_HABITABLE', 'P_DETECTION', 'CNT']]
# Values used for sidemap
TickArray = []
# TickArray holds the value 0, 250, 500 ..... 3000
for i in range(0, 3001, 250):
    TickArray.append(i)
# Sidemap customization
CBARCustomizationDict = {"shrink" : 1, "ticks" : TickArray}
```

```

PivotedDataSet2 = pd.pivot_table(TrimmedDataSet2, index='P_DETECTION',
columns='P_HABITABLE', values='CNT', aggfunc="sum").fillna(0)
# Plots the actual heatmap
# cmap kwarg is the colour map (palette used)
# cbar_kws kwarg is the customization of sidemap
# annot kwarg is used to set values on each square used in heatmap
sns.heatmap(PivotedDataSet2, cmap="rocket_r", cbar_kws = CBARCustomizationDict,
annot=True)
# Y-Axis Label
plt.ylabel("Planetary Detection Methods")
# X-Axis Label
plt.xlabel("Habitability")
# Writes the PivotedDataSet2 into Habitability.csv file
PivotedDataSet2.to_csv("Habitability.csv")

```

We make a heatmap that utilizes the seaborn library of Python to make data pleasing to the eye and show the data more effectively.

 [Question_1.3](#) [Link To The Heatmap Mentioned Above]

Check Below For Output Image



We see that a lot more planets are in the **uninhabitable range** as compared to the **habitable range**.

Techniques other than Radial Velocity and Transit have been completely ineffective when it comes to scanning the sky out for habitable planets.

The Transit method has found the most number of planets in the **habitable range** (38) followed by the radial velocity method (17).

Techniques other than the transit and the radial velocity method have been effective only when finding **uninhabitable planets**.

This could be so because the planets found by these methods are large gas giants that are far away from the star, away from the **habitable zone**, making it easy for these less precise methods to detect such planets.

1.4 Determine the Interquartile range and the skewness of the dataset

Skewness is a measure of **inaccuracy** and **variance** of the dataset. It is given by the formula:

$$\text{Skewness} = 3(\text{Mean} - \text{Median})/(\text{Standard Deviation})$$

Interquartile range is one of the best measures for variability for skewed and extreme datasets, and ones with the presence of several outliers (observations that lie a large distance away from the other values in a random sample).

$$\text{Interquartile range} = Q3 - Q1$$

Where Q3 and Q1 are the values of the third and first quartiles of the data, respectively.

Implementation:

```
Python
# Gets the skewness value of each row and writes it as CSV file to Skewness.csv file
DataSet.skew(numeric_only=True).dropna().to_csv("Question1/Skewness.csv")
# Inter Quartile Range = Third Quartile - First Quartile
IQRDataSet = DataSet.quantile(0.75, numeric_only=True) - DataSet.quantile(0.25, numeric_only=True)
# Writes the IQRDataSet to IQR.csv file
IQRDataSet.dropna().to_csv("Question1/IQR.csv")
```

 [Question_1.4](#) [Link To The CSV Mentioned Above]

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

Often when we have a skewed data distribution, we face issues regarding lower recognition rates for minority classes. This makes the algorithms to favor or be biased towards the majority class leading to class imbalance

To tackle this bias, we have two techniques as follows:

1. **Oversampling:**

- In this approach we synthesize newer examples from the minority class. The technique used is **SMOTE** (Synthetic Minority Oversampling Technique).
 - **SMOTE** randomly selects a minority class example and finds its nearest minority class neighbors at random. A synthetic example is generated at a point in the line connecting the two examples.
 - When created samples from **SMOTE** are added to our dataset, then the class distributions are balanced and help the classifier to recognize the minority class better with lesser overfitting to the majority class.

2. **Undersampling:**

- Here, as the name suggests, we decrease the number of samples from the majority class enough to match the number of samples from the minority class.
- The method followed to achieve Undersampling is a *Random sampler*. We select randomly a desired number of samples from the majority class and remove them.

1.6) Ratio of the Number of Fe atoms to the Number of H atoms for the Exoplanets in the Dataset

The following ratio $[\text{Fe}/\text{H}]$ refers to the star's metallicity, the fraction of heavier metals in a star. The metallicity of a star can be a function that directly describes its age- a star with a larger metallicity is a younger star. In contrast, one with low metallicity is an older star. This is because the newer stars formed are made from the nebulae leftovers from supernovas strewn across the cosmic landscape and are filled with heavy metals that are made from the stars that failed to make effective fusion processes and in turn, made far less energy-giving heavier element fusion processes.

Implementation:

Python

```
# SUN_X - mass fraction of hydrogen in the sun
# SUN_Z - metallicity of sun
# FE_M - atomic mass of iron
# HY_M - atomic mass of hydrogen
# SUN_N_BY_H - the ratio of number of iron atoms to number of hydrogen atoms of
sun
SUN_X = 0.7381
SUN_Z = 0.0134
FE_M = 55.845
HY_M = 1.00784
StarMetallicity = DataSet['S_METALLICITY']
StarMetallicity = 10**(StarMetallicity)
SUN_N_BY_H = (SUN_Z / SUN_X) * (FE_M / HY_M)
StarMetallicity = StarMetallicity * SUN_N_BY_H
DataSet['P_FE_TO_H_RATIO'] = StarMetallicity
# Writes the Columns P_NAME and P_FE_TO_H_RATIO to FE_TO_H_RATIO.csv file
DataSet[["P_NAME", "P_FE_TO_H_RATIO"]].to_csv("Question1/FE_TO_H_RATIO.csv")
```

 [Question 1.6](#) [Link To The CSV Mentioned Above]

Here is the explanation of the mathematics of the code written above:

$$\text{Metallicity of a star} = \log_{10} \left(\frac{N_{\text{Fe}}}{N_{\text{H}}} \right)_{\star} - \log_{10} \left(\frac{N_{\text{Fe}}}{N_{\text{H}}} \right)_{\odot} \rightarrow \textcircled{1}$$

where N_{Fe} and N_{H} are the number of iron & hydrogen atoms per unit of volume. \star represents the quantity for a star, \odot represents the quantity for our own Sun.

$$\left[\left(\frac{N_{\text{Fe}}}{N_{\text{H}}} \right)_{\odot} = \left(\frac{\text{no of atoms of Fe}}{\text{no of atoms of H}} \right)_{\odot} \times \left(\frac{m_{\text{Fe}}}{m_{\text{H}}} \right)_{\odot} \right]$$

$$\begin{aligned} \text{C: where } m_{\text{Fe}} \text{ \& } m_{\text{H}} \text{ are the atomic masses of Fe \& H} \\ &= \left(\frac{\text{total mass of Fe}}{\text{total mass of H}} \right)_{\odot} \times \frac{m_{\text{H}}}{m_{\text{Fe}}} = \left(\frac{\text{total mass of Fe} \times \text{mass of whole system}}{\text{total mass of H} \times \text{mass of whole system}} \right)_{\odot} \times \frac{m_{\text{H}}}{m_{\text{Fe}}} \\ &= \left(\frac{\text{mass fraction of Fe}}{\text{mass fraction of H}} \right)_{\odot} \times \frac{m_{\text{H}}}{m_{\text{Fe}}} \end{aligned} \rightarrow \textcircled{2}$$

$$\therefore \text{Metallicity of a star} = \log_{10} \left(\frac{\left(\frac{N_{\text{Fe}}}{N_{\text{H}}} \right)_{\star}}{\left(\frac{N_{\text{Fe}}}{N_{\text{H}}} \right)_{\odot}} \right)$$

$$\textcircled{3} \leftarrow 10^{\text{metallicity}} = \frac{\left(\frac{N_{\text{Fe}}}{N_{\text{H}}} \right)_{\star}}{\left(\frac{N_{\text{Fe}}}{N_{\text{H}}} \right)_{\odot}}$$

$$10^{\text{metallicity}} \left(\frac{N_{\text{Fe}}}{N_{\text{H}}} \right)_{\odot} = \left(\frac{N_{\text{Fe}}}{N_{\text{H}}} \right)_{\star}$$

from ②,

$$\textcircled{4} \leftarrow 10^{\text{metallicity}} \left(\frac{\text{mass fraction of Fe}}{\text{mass fraction of H}} \right)_{\odot} \times \left(\frac{m_{\text{H}}}{m_{\text{Fe}}} \right)_{\odot} = \left(\frac{N_{\text{Fe}}}{N_{\text{H}}} \right)_{\star}$$

For the sun, mass fraction of Fe = 0.0134,
mass fraction of H = 0.7381,
 $m_{\text{H}} = 1.00784$
 $m_{\text{Fe}} = 55.845$

Formula ② is in line 12 in code box
Step ③ is in line 11 in code box
Formula ④ is in line 13 in code box.

2. Interpretation and calculation of physical parameters

2.1 Calculate the escape velocities of exoplanets and compare them to their estimated temperatures.

Implementation:

```
Python
# Dictionary to store the Escape Velocities in the format (P_NAME)(str) :
# (EscapeVelocity)(float)
EscapeVelocityDict = {}
# Newtons Gravitational Constant
G = 6.67430 * (10 ** (-11))
# Logic for finding Escape Velocity iteratively
# DataSet.shape[0] gives the number of rows in DataSet
for i in range(0, DataSet.shape[0]):
    # If Either P_MASS or P_RADIUS is NAN, Then we do not calculate Escape Velocity
    if not ((math.isnan(DataSet["P_MASS"][i])) or
            (math.isnan(DataSet["P_RADIUS"][i]))):
        # Escape Velocity = sqrt(2GM/r)
        EscapeVelocityDict[DataSet["P_NAME"][i]] = ((2 * G *
            DataSet["P_MASS"][i])/DataSet["P_RADIUS"][i]) ** (1/2)
plt.scatter(DataSet["P_ESCAPE"].fillna(0), DataSet["P_TEMP_EQUIL"].fillna(0),
            color = "crimson")
plt.savefig("EscapeVelocity_VS_Temperature_Plot.pdf")
```

 [Question 2.1](#) [Link To The CSV Mentioned Above]

Note that the x-axis, P_TEMP_EQUIL has been compressed on the range of 0 to 48 because the graph looks extremely congested around the lower left part otherwise.

2.1.1 Analyze if velocities are sufficient to retain the atmospheres (keeping in mind of atmospheric escape processes)

Atmospheric escape processes refer to the process of a planetary body's atmosphere slowly getting decayed away due to external/internal factors.

They can be because of reasons like:

- i) Solar wind processes- It is believed that Mars once had flowing water rivers and a healthy atmosphere. Due to a poor magnetosphere, the solar wind slowly blew the Martian atmosphere away. The closest exoplanet to us, about 4.2 light years away, Proxima Centauri-b, is in the habitable zone around the dwarf star it revolves around. Still, it is speculated to be uninhabitable because of the absence of a thick atmosphere due to the constant bombardment of intense solar radiation.
- ii) Low escape velocity, high planet temperature- A high planet temperature would mean that the gaseous particles in the atmosphere have incredibly high kinetic energies. A low escape velocity would help the gaseous particles escape quite easily, leading to the loss of an atmosphere incredibly quickly.
- iii) Other processes include Photochemical escape, sputtering escape, and charge exchange escape.

Analysis of the graph shows that there is a lot of congestion in the graph around the lower x and y-axes, indicating that a stable escape velocity is present for lower to medium ranges of mean planet temperature. A surprising revelation that arises for extreme cases is that there are planets with extremely high surface temperatures with low escape velocities. This would lead to steady deterioration of the atmosphere of the planet as we know it; such a planet could exist only in the most tiniest of chances under the influence of extreme greenhouse effects.

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

In the below code, we've made a heatmap on Seaborn that maps the metallicity of a star to its age. The metallicity is negative in some cases as it is compared to the metallicity of our Sun in the logarithmic scale- hence a negative metallicity would mean lesser metallicity than our sun.

Implementation:

```
Python
# Styling the plot
sns.set_context('paper', font_scale=4.0)
DataSet['CNT'] = 1
# Extract required data fields
star_data = DataSet[['S_AGE', 'S_METALLICITY', 'CNT']]
# Sort metallicity values
star_data.sort_values(by=['S_METALLICITY'])
pivoted_star_data = pd.pivot_table(star_data, index = 'S_AGE', columns =
'S_METALLICITY', values = 'CNT')
# Adjusting plot attributes
plt.figure(figsize = (50, 40))
sns.heatmap(pivoted_star_data, vmin = -1, vmax = 1, cmap = 'Blues')
# Labelling plot axes
plt.xlabel("Star Metallicity")
plt.ylabel("Star Age")
# Plot title
plt.title("Star Ages vs Star Metallicity\n\n")
# Exports as pdf
plt.savefig("StarAges_VS_StarMetallicity_Plot.pdf")
```

[🔗Question_2.2](#) [Link To The PDF Mentioned Above]

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

As shown in the data visual in the PDF above, we can see that there seems to be a lot of crowding in the metallicity ranges of -0.11 to 0.108.

An important analysis that we can make from the graph is that the older the star, the lesser the metallicity it has. This is because the presence of elements other than H and He (the very definition of metallicity is the fraction of elements that are non-H and He) is incredibly low in older stars. After all, while collapsing on the gas clouds used to make the star, the more primitive cloud lacked any heavier elements. Only as time passed, due to supernovas and other star material ejecting processes, gas clouds began to get rich in heavier elements due to ineffective fusion processes as some stars began to run out of fuel and underwent such events.

As expected, younger stars have more metallicity because of longer exposure to several generations of heavier element-rich supernovae processes.

[An indirect observation that can be made is that stars that have more metallicity tend to have more planets. Refer:

(<https://iopscience.iop.org/article/10.1086/428383/pdf>)]

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

In the following code, we have linked the magnetic field of the star (given by S_MAG) with various parameters of the planet's atmosphere, such as the equilibrium temperature, and secondary parameters such as the density and escape velocity of the planet.

The temperature of a planet can be directly attributed to the properties of its atmosphere. A planet with a large temperature and sustains it can be likened to an atmosphere that is thick and has a large amount of greenhouse gasses. A planet with a very low temperature, on the other hand, goes along with a thinner atmosphere, or possibly being far away from the parent star.

Secondary factors like density can be remotely linked to atmospheric properties. A planet with a large density can be deemed to be a rocky planet, and the more density it has, it can *possibly* be linked to a greater atmospheric pressure. On the other hand, a planet with really low density can be straightaway deemed to be a gas giant, and the properties of the atmosphere of a gas giant are more or less the same-layers and layers of gasses like Hydrogen, Helium, and a possible trace of Ammonia.

Implementation:

```
Python
plt.figure(figsize = (15,8), tight_layout = True)
# Plots a 2D Histogram
plt.hist2d(DataSet["S_MAG"].fillna(0), DataSet["P_TEMP_EQUIL"].fillna(0), bins =
40, cmap = "rocket_r")
plt.title("Magnetic Field vs Temperature", fontsize = 20)
# X and Y Labels
plt.xlabel("Star's Magnetic Field (Tesla)", fontsize = 10)
plt.ylabel("Atmospheric Temperature of planet (Kelvin)", fontsize = 10)
plt.colorbar()
# Exports as PDF
plt.savefig("StarMagneticField_VS_Temperature_Plot.pdf")
```

Python

```
plt.figure(figsize = (15,8), tight_layout = True)
# Plots a 2D Histogram
plt.hist2d(DataSet["S_MAG"].fillna(0), DataSet["P_DENSITY"].fillna(0), bins = 40)
plt.title("Magnetic Field vs Density of Planet ", fontsize = 20)
# X and Y Labels
plt.xlabel("Star's Magnetic Field (Tesla)", fontsize = 10)
plt.ylabel("Density of Planet ", fontsize = 10)
plt.yticks(np.arange(0,20,2))
plt.ylim(0,20)
plt.colorbar()
# Exports as PDF
plt.savefig("StarMagneticField_VS_Density_Plot.pdf")
```

Python

```
plt.figure(figsize = (15,8), tight_layout = True)
# Plots a 2D Histogram
plt.hist2d(DataSet["S_MAG"].fillna(0), DataSet["P_ESCAPE"].fillna(0), bins = 40)
plt.title("Magnetic Field vs Escape Velocity", fontsize = 20)
# X and Y Labels
plt.xlabel("Star's Magnetic Field (Tesla)", fontsize = 10)
plt.ylabel("Escape Velocity", fontsize = 10)
plt.yticks(np.arange(0,10,1))
plt.ylim(0, 10)
plt.colorbar()
# Exports as PDF
plt.savefig("StarMagneticField_VS_EscapeVelocity.pdf")
```

 [Question 2.3](#) [Link To The PDF's Mentioned Above]

(Note that the values of escape velocity have been normalized to a scale of 0-10 for easier visualization and applicability.

Another way to tackle this problem by relating the S_MAG and the 'atmospheric properties of the planet' is to directly find out the

P_ATMOSPHERE of the planet by applying the formula of P_ESI. That is, P_ESI can be given by:

$$ES = \prod_{i=1}^n \left(1 - \left| \frac{o_i - r_i}{o_i + r_i} \right| \right)^{\frac{w_i}{n}}$$

Where ‘i’ refers to the i'th parameter that influences the ESI, such as escape velocity, density, or surface temperature of the planet. O_i is the value of the parameter for the planet we have taken and R_i is the value of the parameter on earth. ‘N’ is the total number of parameters taken into consideration, and W_i is the weightage of each of the parameters in the formula.

However, the exact weightage varies from the exact set of parameters taken, and hence we have not applied this formula here. This method however is the best way to get the value of atmospheric pressure when there is no direct parameter of P_ATMOSPHERE that has been provided.

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

The stability of any planet's atmosphere is strongly dependent on its magnetosphere, i.e. the magnetic field around the planet generated by the moving mantle made of metals like Iron and Nickel.

Planets like Mars had a very weak magnetosphere, due to which its atmosphere slowly faded away due to the influence of the sun's strong solar wind.

Volatile compounds like Methane, Carbon Dioxide, and Water can be blown away by photochemical degradation of the Star's strong magnetic field. Additionally, Ozone, which relies on the stability of free ions of oxygen broken by sunlight in an appropriate amount, can cease to exist under the influence of too many UV rays that would not be controlled due to a poor Magnetosphere.

All in all, the presence of a good and stable Magnetosphere is essential for life to sustain on any planet, regardless of its presence around the habitable zone around its star.

2.4 What are spectral types? How many major spectral types exist?

The spectral type of a star gives several characteristics of a star- Its age, the color of its outer photosphere, the events leading to its formation and death, and so on.

There are 7 major types (O, B, A, F, G, K, M) of spectral stars, of which O is the brightest, hottest, and lives for the shortest. In contrast, M is the dimmest, relatively cooler, and long-living type of star.

As the name suggests, 'spectral' types are found for a star using its 'spectrum.' Due to the Doppler effect arising due to the rapid expansion of spacetime,, initially, there were errors in the correct attribution of spectral type for a star, but methods that negligisized this error came up.

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

Implementation:

Python

```
# Creating a set to store star types
type_temp = set()
for i in DataSet['S_TYPE_TEMP']:
    # Eliminating any 'nan' values
    if i in ['A', 'B', 'F', 'G', 'K', 'M', 'O']:
        type_temp.add(i)

# Styling the plot
sns.set_context('paper', font_scale=2.0)

# Creating a column 'CNT' and equating it to indexes in 'P_HABITABLE'
DataSet['CNT'] = DataSet['P_HABITABLE'] + 1

# Extract required data columns
req_data = DataSet[['S_TYPE_TEMP', 'P_HABITABLE', 'CNT']]

# Creating a pivot table
pivoted_req_data = pd.pivot_table(req_data, index = 'S_TYPE_TEMP', columns =
    'P_HABITABLE', values = 'CNT', aggfunc="sum").fillna(0)
```



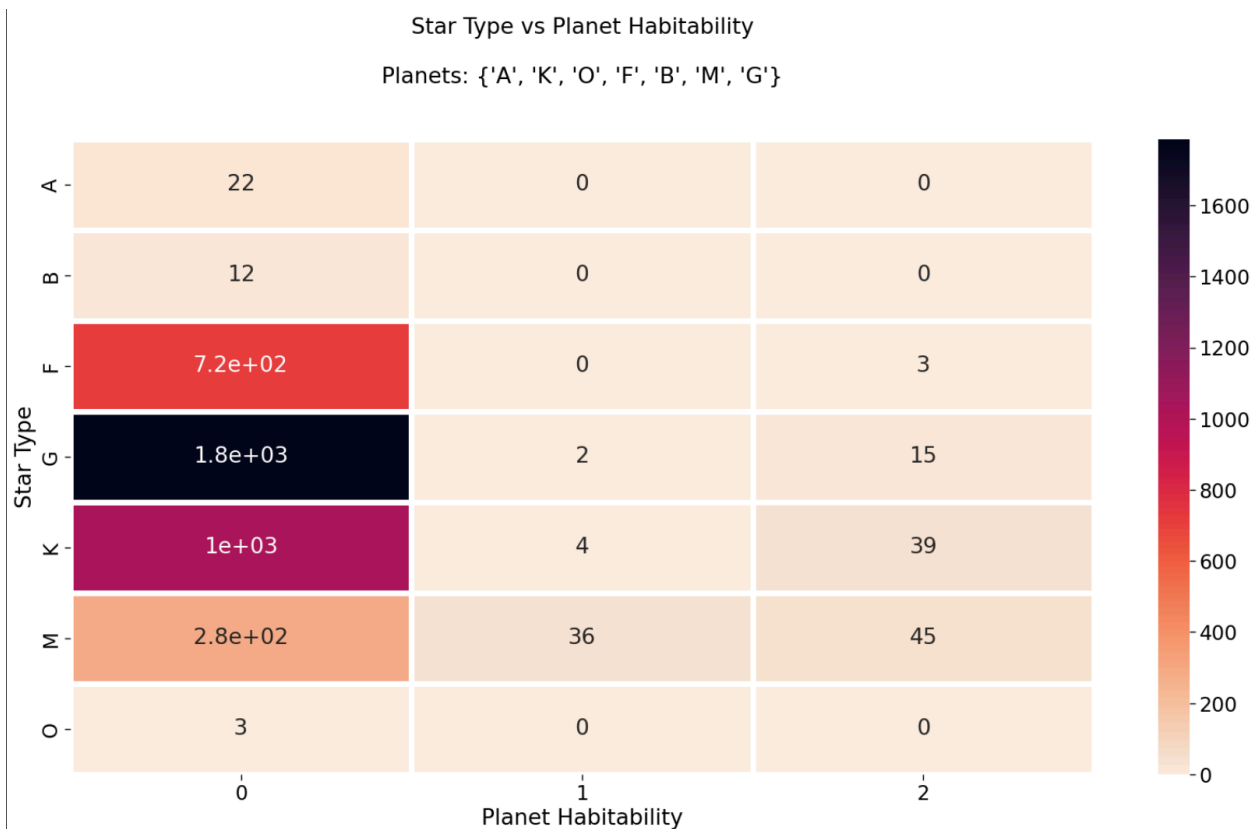
```
# Adjusting plot attributes
plt.figure(figsize = (20, 10))
sns.heatmap(pivoted_req_data, cmap='rocket_r', linewidths = 5.0, annot=True)

# Labelling the axes
plt.xlabel("Planet Habitability")
plt.ylabel("Star Type")

# Title of the Plot
plt.title(f'Star Type vs Planet Habitability\n\nPlanets: {type_temp}\n\n')

# Exports as pdf
plt.savefig("StarType_VS_PlanetHabitability_Plot.pdf")
```

[Question_2.4.1](#)[Link To The PDF Mentioned Above]



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

We can see that a majority of the planets are in the spectral types between M, K, G and F; with the majority of the potentially habitable planets of the spectral class M ($36+45=81$ planets) and followed by the K class of spectral type of stars.

There are a few major reasons why M and K spectral types show a large amount of habitable planets-

- i) The stars that come under M and K are cooler than our own sun (most frequently red dwarf stars) and they are quite numerous in nature as compared to the presence of other spectral types.
- ii) The habitable zone of a planet in the M and K types, as they are smaller than our own sun's, are much narrower and closer to the star, and hence easier to detect with the major mode of planetary detection: the **transit** method.

Another analysis we can make is that the majority of the 'uninhabitable' planets found have been under the G spectral type. This is just a coincidence. Most of the planets that are easy to find using the transit method around the G spectral type seem to be far away from the star.

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

Upon analyzing the provided CSV file and going through the data, we gather the following points:

- i) More massive stars that came under the spectral types like O, B, A tend to have more massive planets, like gas giants. Smaller stars tend to have smaller, rocky, earth-like planets.
- ii) The closer the planet is to a specific spectral type, the more it might determine its atmospheric properties and its size. For example, a planet

close to an O type (hot) star can be blown into smithereens and left into nothing but a ball of rock, can be an earth-like planet at the same distance from a dwarf sun can be habitable.

3. Feature Engineering

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

Implementation:

Python

```
# Fetching the numeric columns in DataSet as list
NumericColumns = DataSet.select_dtypes(include=np.number).columns.tolist()
# Dictionary for storing NULL_VALUE_PERCENTAGE of each column in the format
# (COLUMN_NAME)(str) : (NULL_VALUE_PERCENTAGE)(float)
NullRowsDict = {}
# Logic for finding NULL_VALUE_PERCENTAGE iteratively
for NumericColumn in NumericColumns:
    # Initialization of Variables
    NullRowCount = 0
    # Fetches data available in each row in respective column
    for Data in DataSet[NumericColumn]:
        # We are checking whether the given data is NAN(Not Any Number)
        if math.isnan(Data):
            # If data is nan then we are incrementing NullRowCount
            NullRowCount += 1
    # NULL_VALUE_PERCENTAGE = ((Rows Which are Null)/(Total Number of Rows)) * 100
    NullRowsDict[NumericColumn] = (NullRowCount/DataSet.shape[0]) * 100
# Defines the figure size (in this case figure is image)
plt.figure(figsize=(40,40))
# Keys and values as list
Values = list(NullRowsDict.values())
Keys = list(NullRowsDict.keys())
# Plots the Bar Graph of Null Value Percentage of Each Column (Horizontally)
plt.barh(Keys, Values)
# Title
plt.title("Percentage of Null Values", fontsize=25)
# Y-Axis Label
plt.xlabel("Percentage of Null Values in Each Column", fontsize=15)
# X-Axis Label
plt.ylabel("Columns ", fontsize=10)
# Saves the plot in NullValuesPercentagePlot.png
plt.savefig("NullValuesPercentagePlot.pdf")
```

From the output chart, it is apparent that several data values are missing and require some sort of imputation technique to fill out the null values in the dataset for better analysis of the exoplanet data if we were to train a statistical ML model on the given data.

 [Question_3.1](#) [Link To The PDF Mentioned Above]

3.2 Feature Reduction

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

The following features were identified to be redundant or highly correlated :

1. P_MASS_ERROR_MIN & P_MASS_ERROR_MAX
2. P_RADIUS_ERROR_MIN & P_RADIUS_ERROR_MAX
3. P_SEMI_MAJOR_AXIS_ERROR_MIN & P_SEMI_MAJOR_AXIS_ERROR_MAX
4. P_ECCENTRICITY_ERROR_MIN & P_ECCENTRICITY_ERROR_MAX
5. P_PERIOD_ERROR_MIN & P_PERIOD_ERROR_MAX
6. P_INCLINATION_ERROR_MIN & P_INCLINATION_ERROR_MAX
7. P_DISTANCE_ERROR_MIN & P_DISTANCE_ERROR_MAX
8. P_METALLICITY_ERROR_MIN & P_METALLICITY_ERROR_MAX
9. P_TEMP_ERROR_MIN & P_TEMP_ERROR_MAX
10. P_IMPACT_PARAMETER

For the first 9 features, we realized having the absolute error was more efficient when it came to analyzing and approximating the data. The absolute error is an unambiguous measure of error magnitude.

For feature number 10, the impact parameter can be directly derived from the formula:

$$b = \frac{\cos(i) \times a}{R_s}$$

Where 'i' is the inclination and 'a/Rs' is the semi-major axis in the units of 'Stellar radii.'

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

To reduce the features identified as redundant and highly correlated, the following methods were adopted:

1. Features with *more than 50%* of the cells having NULL value were removed from the dataset.
2. For features involving error_max and error_min
 - a. If any one feature is NULL then the other feature was removed.
 - b. If both the features are not NULL, then that having the maximum absolute error was removed.
 - c. Few other features were classified under Dependent and Useless Columns -

```
DependentColumns = ["P_DENSITY", "P_GRAVITY",
                    "P_IMPACT_PARAMETER", "P_TEMP_EQUIL", "P_PERIOD",
                    "P_FLUX"]

UselessColumns = ["S_ALT_NAMES", "S_CONSTELLATION",
                  "S_CONSTELLATION_ABR", "S_CONSTELLATION_ENG",
                  "P_YEAR", "P_UPDATED"]
```

This way we were able to reduce the feature count from 112 to 70.

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 cluster

Python

```
# Function used to reduce Error Columns Alone

def ReduceErrorColumn(ColumnName1: str, ColumnName2: str):

    Array = [ ]

    # Logic to Get the Absolute Maximum of Errors in Each of these Columns

    for i in range(0, DataSet.shape[0]):

        if not math.isnan(DataSet[ColumnName1][i]) and not
        math.isnan(DataSet[ColumnName2][i]):

            MaxError = max(abs())

            # If Either of them is NAN, then Absolute Value of Other will be the Greatest
            Possible Error

            elif math.isnan(DataSet[ColumnName1][i]) and not
            math.isnan(DataSet[ColumnName2][i]):

                MaxError = abs(DataSet[ColumnName2][i])

                Array.append(MaxError)

            elif math.isnan(DataSet[ColumnName2][i]) and not
            math.isnan(DataSet[ColumnName1][i]):

                MaxError = abs(DataSet[ColumnName1][i])

                Array.append(MaxError)

            # If both are NAN, then we fill this also with NAN

            else:

                Array.append(np.nan)

    # Converts This Array into Pandas Dataframe for Converting it into CSV file

    ReducedDataFrame = pd.DataFrame(Array)

    return ReducedDataFrame

# Catching Half Empty Columns and Removing Insignificant Ones
```

```

HalfEmptyColumns = []

# Determining Significant Columns in those HalfEmptyColumns
SignificantButHalfEmptyColumns = ["P_MASS"]

# Appending Columns that have more than 50% of NULL values
for Column in DataSet:
    if DataSet[Column].isnull().mean() >= 0.5:
        HalfEmptyColumns.append(Column)

# Removing Significant Columns if Any have more than 50% of null Values from
the HalfEmptyColumns List
for Elements in SignificantButHalfEmptyColumns:
    HalfEmptyColumns.remove(Elements)

# Getting Error Column Names
RedundantErrorColumnNames = []

# Finding Column Names Ending With ERROR_MAX or ERROR_MIN, because they are
ones that are redundant
for Column in DataSet:
    if (Column[-9:] == "ERROR_MAX" or Column[-9:] == "ERROR_MIN"):
        RedundantErrorColumnNames.append(Column)

# Removing Dependent Columns and Useless Columns
DependentColumns = ["P_DENSITY", "P_GRAVITY", "P_IMPACT_PARAMETER",
"P_TEMP_EQUIL", "P_PERIOD", "P_FLUX"]

UselessColumns = ["S_ALT_NAMES", "S_CONSTELLATION", "S_CONSTELLATION_ABR",
"S_CONSTELLATION_ENG", "P_YEAR", "P_UPDATED"]

# Columns to Remove
ColumnsToRemove = DependentColumns + UselessColumns + RedundantErrorColumnNames +
HalfEmptyColumns

# Copying the reduced dataset into Another DataSet
ModifiedDataSet = DataSet.copy().drop(columns = ColumnsToRemove)

# Logic for iteratively removing the Redundant Error Columns
for i in range(0, len(RedundantErrorColumnNames), 2):

```



```

Reduced = ReduceErrorColumn(RedundantErrorColumnNames[i],
RedundantErrorColumnNames[i+1])

ModifiedDataSet[RedundantErrorColumnNames[i][: -4] + "_MAX"] = Reduced

# Writes the ModifiedDataSet to FeatureReduced_ExoplanetCatalog.csv

ModifiedDataSet.to_csv("FeatureReduced_ExoplanetCatalog.csv")

```

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

Python

```

# Gets the already implemented K-Nearest Neighbours Imputer from Scikit-Learn
Library

# We are using K = 4

from sklearn.impute import KNNImputer

Imputer = KNNImputer(n_neighbors=4)

# Gets names of numeric columns as list

NumericColumns = DataSet.select_dtypes(include = np.number).columns.tolist()

# Algorithm Writes the values to missing columns in the DataSet

DataSet_Imputed = pd.DataFrame(Imputer.fit_transform(DataSet[NumericColumns]))

# Writes DataSet_Imputed to Imputed_Exoplanet_2019.csv file

DataSet_Imputed.to_csv("Imputed_ExoPlanet_2019.csv")

```

[Question_3.3](#) - [Link To The PDF Mentioned Above]

The following method uses the KKN Imputer which uses the Scikit-Learn Library.

To elaborate, this is whatever has been done above:

i) We have taken the KKN Imputer from the SciKit library to process the given data by taking the nearest 4 neighbors (that have a value) of the missing data value.

ii) Next, the imputer takes the mean of the 4 nearest values and plugs it into the missing value.

iii) In this manner, the missing values in the dataset are filled. It is important to note that the values, when substituted, may not give the exact value for *that* specific null value, but would definitely give a distributed dataset that would have very similar measures of central tendency as the actual dataset would have if all the values were filled.

4. Habitability Classification



4.1) 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.

Implementation:

Python

```
# Take copy for working
DataSet_31 = DataSet[DataSet.select_dtypes(include
np.number).columns.tolist()].fillna(0)
# Drop derived fields, that are unnecessary for training
DataSet_31 = DataSet_31.drop(['P_MASS_ERROR_MAX', 'P_RADIUS_ERROR_MAX',
'P_PERIOD_ERROR_MAX', 'P_SEMI_MAJOR_AXIS_ERROR_MAX', 'P_ECCENTRICITY_ERROR_MAX',
'P_INCLINATION_ERROR_MAX', 'P_OMEGA_ERROR_MAX', 'P_TPERI_ERROR_MAX',
'P_IMPACT_PARAMETER_ERROR_MAX', 'P_GEO_ALBEDO_ERROR_MAX', 'S_DISTANCE_ERROR_MAX',
'S_METALLICITY_ERROR_MAX', 'S_MASS_ERROR_MAX', 'S_RADIUS_ERROR_MAX',
'S_AGE_ERROR_MAX', 'S_TEMPERATURE_ERROR_MAX'], axis=1)
# Split train data and target data
TrainDataSet = DataSet_31
TrainLabels = np.array(TrainDataSet.pop('P_HABITABLE'))
TrainFeatures = np.array(TrainDataSet)
# Data normalization - limit values for all columns between -5 and 5
scaler = StandardScaler()
TrainFeatures = scaler.fit_transform(TrainFeatures)
TrainFeatures = np.clip(TrainFeatures, -1, 1)
# Define a function to provide Decision Tree model
def model(depth):
    return tree.DecisionTreeClassifier(random_state=42, max_depth= depth)
# Function to train using kfold cross validation
def kfold_train(train_features, train_labels, model):
    X = TrainFeatures
    y = TrainLabels
    # Return arrays to store result
    accuracy_scores = []
    k_splits = []
    # Run KFold for values from 2 to 10
    for i in range(2, 11):
```

```

# Run the model
# Get model scores
kf = StratifiedKFold(n_splits=i, shuffle=True, random_state=42)
score = cross_val_score(model, X, y, cv=kf, scoring="accuracy")
# Store accuracy scores
accuracy_scores.append(score.mean())
k_splits.append(i)
return [k_splits, accuracy_scores]
# Utility to train different models
def fine_tune_decision_tree():
    res = []
    for depth in range(1, 11):
        # Create a Decision tree model with given depth
        tmodel = model(depth)
        # Train and get accuracy scores for the above model
        k_splits, accuracy_scores = kfold_train(TrainFeatures, TrainLabels, tmodel)
        # Store the results
        res.append([depth, k_splits, accuracy_scores])
    return res
# Start training for all combinations of decision trees and KFold splits
res = fine_tune_decision_tree()
# For each score obtained from training, plot the results in a line chart
LineColors = ['black', 'tab:blue', 'tab:orange', 'tab:green', 'tab:red',
'tab:purple', 'tab:brown', 'tab:pink', 'tab:gray', 'tab:olive', 'tab:cyan']
plt.figure(figsize=(50,20))
for row in res:
    plt.plot(row[1], row[2], color=LineColors[row[0]], label=f'depth={row[0]}')
plt.legend(loc = "lower right")
plt.title("K-Fold Cross Validation Accuray Graph", fontsize = 40)
plt.xlabel('K-Fold value', fontsize = 40)
plt.ylabel('Accuracy', fontsize = 40)
plt.savefig("KFold_Accuracy_Plot.pdf")

```

 [Question 4.1](#) [Link To The PDF Mentioned Above]

The following are the steps taken to implement 4.1:

To a new data frame - DataSet_31 all features / columns having numerical values were copied to DataSet_31 . The cells having null values in those columns were set to 0.

From this data set, we have dropped the unnecessary columns in the training process like 'P_MASS_ERROR_MAX', 'P_RADIUS_ERROR_MAX', 'P_PERIOD_ERROR_MAX', 'P_SEMI_MAJOR_AXIS_ERROR_MAX', 'P_ECCENTRICITY_ERROR_MAX', 'P_INCLINATION_ERROR_MAX', 'P_OMEGA_ERROR_MAX', 'P_TPERI_ERROR_MAX', 'P_IMPACT_PARAMETER_ERROR_MAX', 'P_GEO_ALBEDO_ERROR_MAX', 'S_DISTANCE_ERROR_MAX', 'S_METALLICITY_ERROR_MAX', 'S_MASS_ERROR_MAX', 'S_RADIUS_ERROR_MAX', 'S_AGE_ERROR_MAX', 'S_TEMPERATURE_ERROR_MAX'

We copied the data to another frame, TrainDataSet, and from the TrainDataSet we popped out the P_Habitable feature and then stored it into the TrainLabels class .

Q1 - What is **popping** ?

We removed the P_Habitable feature and added the feature to TrainLabels - it's like cut-and-paste work

Q2 - Why did we do it ?

Our goal is to train a model to predict and classify a given exoplanet into 0 / 1 / 2 class . So for that we should not train it with the already existing Habitability feature.

Q3 - Then what's the use of **TrainLabels** ?

Yes, so now we use this Habitable feature of TrainLabels for VALIDATING the predictions made by our model after training. So TrainLabels acts as a Target Feature in Data Analytics terms.

Now the remaining features are added to the **TrainFeatures** class.

Normalizing - “scaler” in the code is an object of StandardScaler class - scaler is a specific object which normalizes data to default -1 to 1.

The main idea of training our model:

First, we have defined the “model” function which accepts depth as the parameter . This function is returning a classifier using an algorithm - DecisionTreeClassifier

Now we Use the K-fold validation:

Q4 - What is **K-fold validation**, and why is it used?

It was seen that if the entire dataset is transferred for training at once then the classifier works very inefficiently. So to counter this inefficiency **K-Cross Validation** is used.

X = TrainFeatures

Y = TrainLabels

So the process is - We pass X and Y, then the model must get trained with X and get validated with Y.

So instead of all at once - We **split** the X and Y simultaneously into **K** parts.

- Now we start training our model by feeding in the features of X of first split and making it learn the corresponding 0/1/2 of Y of the same 1st split. Once the model is trained for the 1st split from each X and Y, now the model is tested or validated with the X and Y parts of Kth split i.e, the model is given every other information regarding few sets of exoplanets of Kth split, and then the model predicts the classification based on previous training.
- Just like a test, we validate its predictions and get scores of it and store it in an array.
- This procedure of training and validating with the Kth split continues for all (K-1) splits and corresponding scores are stored.

This value K is supposed to vary from 2 to 10, which is what is asked in the question so we train the model starting with 2 splits up till 10 splits

Now we have scores stored in an array, we take the mean of scores, that is for example - in the 5 fold validation, we had split data to 5 parts and got 4 scores - for 1st, 2nd, 3rd, and 4th parts, we take the mean of this score and store it in another array - accuracy_scores

- Now our model is trained and we check that- at WHAT DEPTH, at WHICH SPLIT, WHAT ACCURACY(from scores) is observed, this is plotted with accuracy in Y-axis and K-Fold Value in X axis and color of plots represent the Depths.

5. Additional Notes

It is with our best efforts, we've put together this project. We thoroughly enjoyed being part of it, learning new things as we went through the coding process, and were left in admiration when it came to analyzing the CSV file of the exoplanet data, realizing how small of a speck we are in a gigantic universe.

We give our heartfelt thanks to the organizers of the competition. Some of our team members being really into space, the others being really into coding, this was a great opportunity to come together and work on this amazing competition.