Cutting Data with the Pandas Library: Exoplanet Statistics

Background:

Exoplanets are planets orbiting a star other than the Sun and can range from small rocky worlds to large gas giants just like in our solar system. However, there are plenty of exoplanets for which there is no direct analog in our solar system, including worlds that are thought to be covered in molten lava, "snowball" worlds, and super-earths. We will be exploring a dataset that includes information on some of the known exoplanets and their host stars.

The exoplanets in this dataset were discovered via the transit method, where a periodic dip in a star's brightness indicates that a planet is blocking a small amount of the star's light when the planet's orbit brings it between the star and the observer. Various other methods have been employed to fully characterize the orbit of each exoplanet and estimate the mass and size of the host star and the planet.

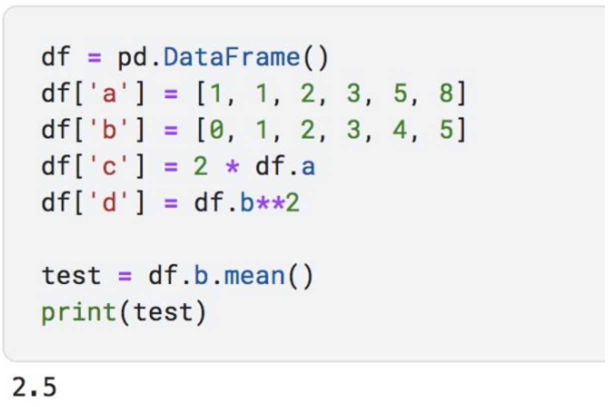
The data has the following columns:

Quantity	**Description (and Units)**
System	The name of the exoplanet
Teff	The temperature of the host star (Kelvin)
M_A	The mass of the host star ($M_\odot$)
R_A	The radius of the host star ($R_\odot$)
Period	The orbital period of the exoplanet (days)
aAU	The semi-major axis of the exoplanet's orbit (AU)
M_b	The mass of the exoplanet ($M_\oplus$)
R_b	The radius of the exoplanet ($R_\oplus$)
Teq	The theoretical equilibrium temperature of the surface of the exoplanet (Kelvin)

Skills:

Today, we'll be practicing the skills below:

- In order to determine the mean value of a column stored in a DataFrame (named df in this code), use the .mean() method for that column. In this example we are computing the mean of the b column:

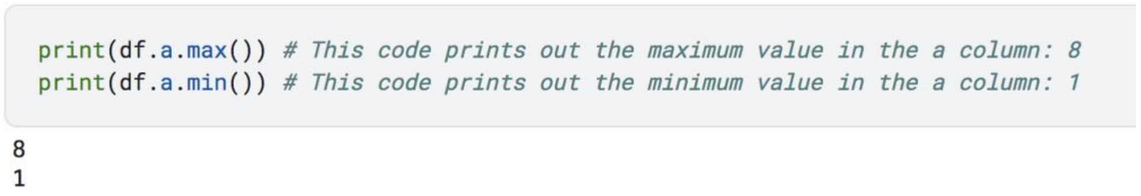


```python
df = pd.DataFrame()
df['a'] = [1, 1, 2, 3, 5, 8]
df['b'] = [0, 1, 2, 3, 4, 5]
df['c'] = 2 * df.a
df['d'] = df.b**2

test = df.b.mean()
print(test)
```

```
2.5
```

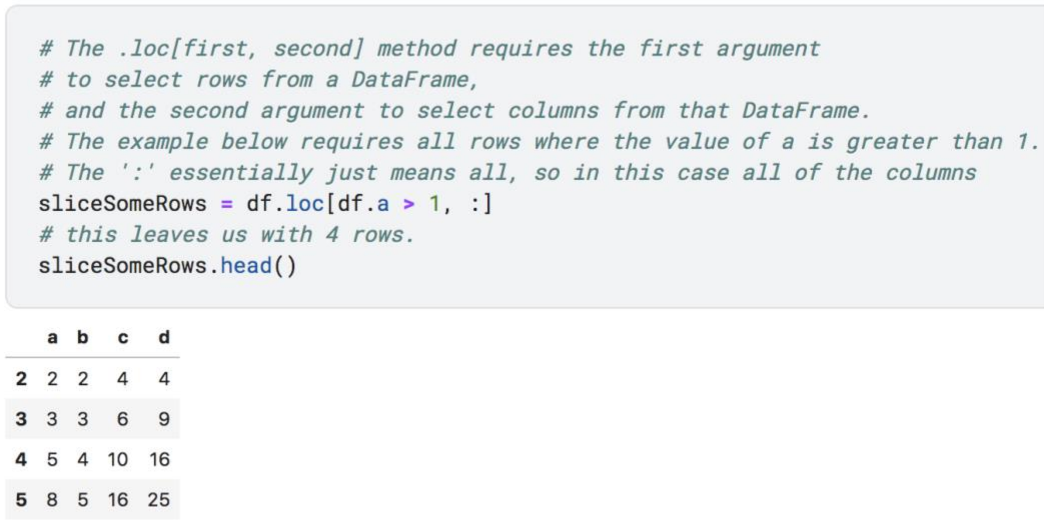
- In order to determine the maximum value of a column stored in a DataFrame, use the .max() method for that column. In order to determine the minimum value of a column stored in a DataFrame, use the .min() method for that column:



```python
print(df.a.max()) # This code prints out the maximum value in the a column: 8
print(df.a.min()) # This code prints out the minimum value in the a column: 1
```

```
8
1
```

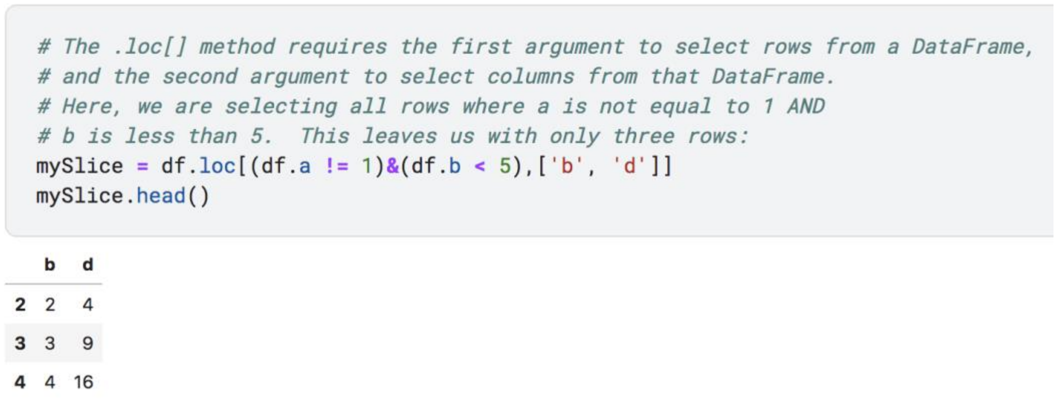
- To cut the data according to some criteria, you can use the .loc[] command. We are essentially "slicing" the data set to keep only what we are interested in. There are two different examples below:
  - First, let's try a simpler example. If we want only rows where a > 1 and we would like all of the columns in the original DataFrame, the following use of .loc can accomplish this:



```python
# The .loc[first, second] method requires the first argument
# to select rows from a DataFrame,
# and the second argument to select columns from that DataFrame.
# The example below requires all rows where the value of a is greater than 1.
# The ':' essentially just means all, so in this case all of the columns
sliceSomeRows = df.loc[df.a > 1, :]
# this leaves us with 4 rows.
sliceSomeRows.head()
```

	a	b	c	d
2	2	2	4	4
3	3	3	6	9
4	5	4	10	16
5	8	5	16	25

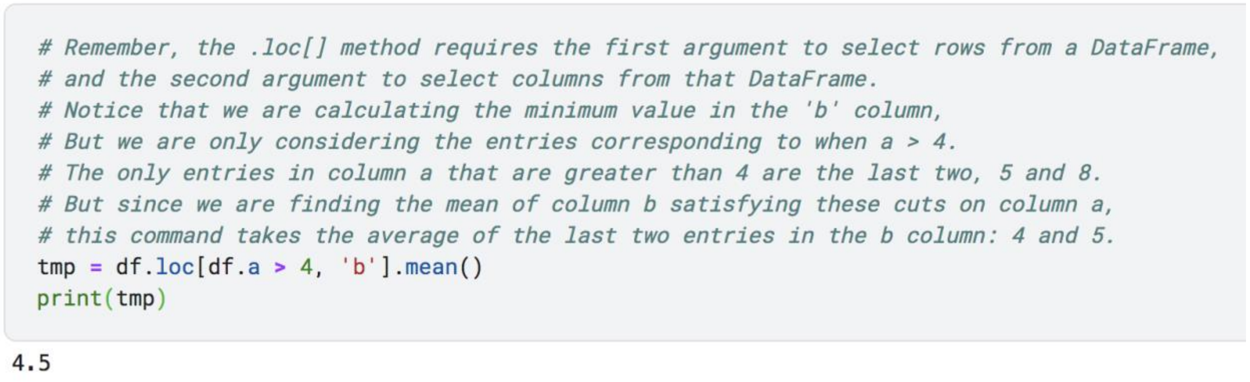
  - We can also combine multiple cuts into one command. This .loc command requires all rows where a is not equal to 1 AND b is less than five. We have also only selected columns b and d. The result is stored in a new DataFrame named mySlice.



```python
# The .loc[] method requires the first argument to select rows from a DataFrame,
# and the second argument to select columns from that DataFrame.
# Here, we are selecting all rows where a is not equal to 1 AND
# b is less than 5.  This leaves us with only three rows:
mySlice = df.loc[(df.a != 1)&(df.b < 5),['b', 'd']]
mySlice.head()
```

```
   b   d
2  2   4
3  3   9
4  4  16
```

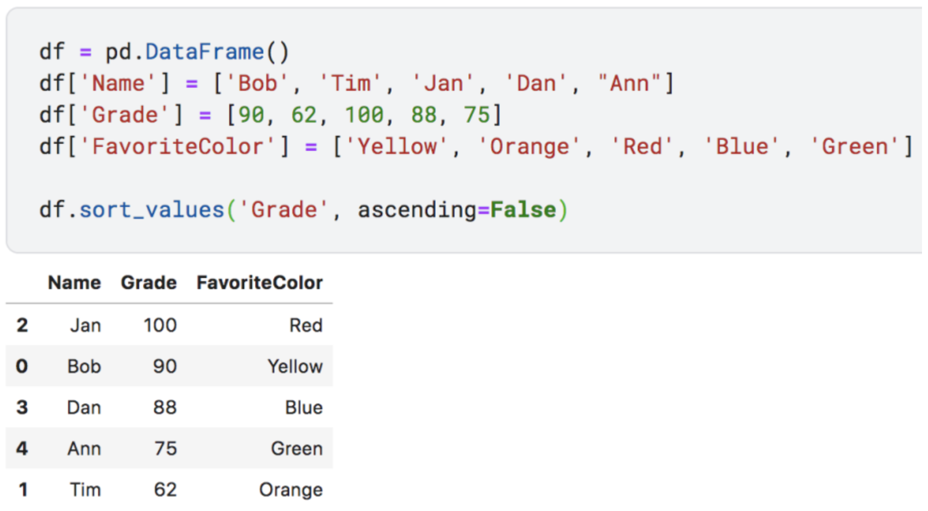
- You can combine cuts and the min, max, and mean methods above by method chaining:



```python
# Remember, the .loc[] method requires the first argument to select rows from a DataFrame,
# and the second argument to select columns from that DataFrame.
# Notice that we are calculating the minimum value in the 'b' column,
# But we are only considering the entries corresponding to when a > 4.
# The only entries in column a that are greater than 4 are the last two, 5 and 8.
# But since we are finding the mean of column b satisfying these cuts on column a,
# this command takes the average of the last two entries in the b column: 4 and 5.
tmp = df.loc[df.a > 4, 'b'].mean()
print(tmp)
```

```
4.5
```

- Finally, you can sort a DataFrame on a particular column by using the .sort_values() method. This is another way of finding the minimum and maximum values for a column along with other context. (NOTE: sorting DOES NOT change the order of the original dataframe in the first example below – it merely displays the sorted data.)
    - You can sort data by the values in a particular column via the following syntax. Note the ascending key word determines if data is sorted from least to greatest or vice versa.

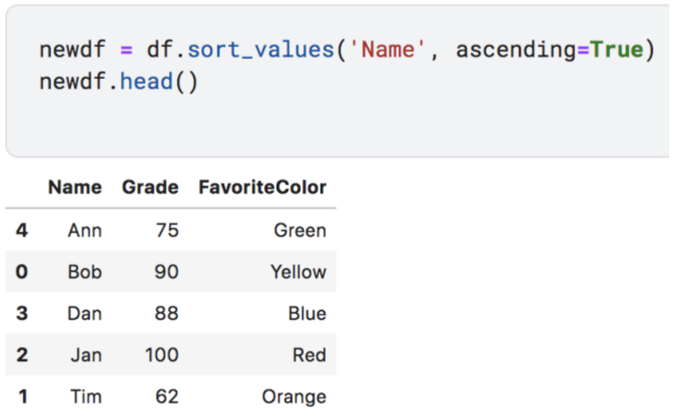


```python
df = pd.DataFrame()
df['Name'] = ['Bob', 'Tim', 'Jan', 'Dan', "Ann"]
df['Grade'] = [90, 62, 100, 88, 75]
df['FavoriteColor'] = ['Yellow', 'Orange', 'Red', 'Blue', 'Green']

df.sort_values('Grade', ascending=False)
```

	Name	Grade	FavoriteColor
2	Jan	100	Red
0	Bob	90	Yellow
3	Dan	88	Blue
4	Ann	75	Green
1	Tim	62	Orange

- Sorting even helps alphabetize string-type data. Notice the sorted data is assigned to a new DataFrame in this example, so the sorted data is stored in the newly sorted order.



```python
newdf = df.sort_values('Name', ascending=True)
newdf.head()
```

	Name	Grade	FavoriteColor
4	Ann	75	Green
0	Bob	90	Yellow
3	Dan	88	Blue
2	Jan	100	Red
1	Tim	62	Orange

Problems:

1. Go to the dataset linked below and create a new notebook: https://www.kaggle.com/datasets/austinhinkel/transiting-exoplanets-selected-system-properties/data

2. Read in the data into a pandas DataFrame named data and use the .head() method to peak at the data. Use the len() function to check how many exoplanets are in the DataFrame. How many exoplanets are in the dataset?

3. Use the .min(), .max(), .mean(), and .sort_values() methods to answer the following questions:

a. What is the highest planetary mass in the dataset?

b. What is the minimum exoplanet radius in the dataset? Which system is this?

c. What is the lowest Stellar temperature in the dataset? Which star system has this temperature?

d. Which planet has the highest equilibrium temperature?

e. What is the mean (average) orbital period for all exoplanets in the dataset? How does this compare to Earth's orbital period of one year?

4. Use the .loc[] method to create slices of the original DataFrame (slice1 = data.loc[(cut1)&(cut2)&…,:]) corresponding to the questions below. Then, calculate the number of exoplanets that satisfy each set of conditions by using the len() function on the slice you have created.
   a. How many exoplanets have surface temperatures (Teq) between 273 and 373 Kelvin? (i.e., liquid water could theoretically exist).

   b. How many exoplanets in the dataset orbit a sun-like star (M_A between 0.95 and 1.05) AND have surface temperatures (Teq) between 273 and 373 Kelvin?

   c. How many exoplanets have a radius larger than Jupiter?

d. How many planets have at least half the mass of Earth but no more than twice the mass of Earth?

e. What is the mean orbital period for exoplanets orbiting a star with a mass of less than half of the Sun’s mass?

f. How many exoplanets have an orbital period greater than 365 days?

5. Can you think of any reasons why there are not many exoplanets in the dataset with periods longer than one year?