Project: Data Acquisition

Name: Melody Goldanloo

This Jupyter Notebook Starter File provides a basic outline for your solutions. For detailed instructions, please refer to the assignment on Canvas. Complete all your work for this project in this same Jupyter Notebook file, which you will submit:

- Code:
  - Insert your code where you see #Insert Code Here.
  - Ensure all code is well-commented and easy to understand.
  - Use clear and descriptive variable names.
- Questions:
  - Provide your answers to the guided questions in the same markdown cell as the questions.
  - Demonstrate a deep understanding of the concepts through thorough explanations and critical thinking.

In [1]:
```
import pandas as pd
import sqlite3
import requests
import os
```

Part 1: Structured Data

Files

CSV

Steps:

1. Load csv data from the provided data URL
2. Print the dataframe's first few elements using the `.head()` command to see what data you have
3. Complete the remaining tasks outlined in Canvas
4. Answer the questions

In [2]:
```
data_url = "https://archive.ics.uci.edu/ml/machine-learning-databases/wine-c
df = pd.read_csv(data_url, delimiter=';')
df.head()
```

Out[2]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphat
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.

In [4]:
```
df.rename(str.title, axis="columns", inplace=True)
```

In [9]:
```
df = df.rename(str.title, axis="columns")
df
```

Out[9]:

	Fixed Acidity	Volatile Acidity	Citric Acid	Residual Sugar	Chlorides	Free Sulfur Dioxide	Total Sulfur Dioxide	Density	Ph	S
0	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	
1	7.8	0.880	0.00	2.6	0.098	25.0	67.0	0.99680	3.20	
2	7.8	0.760	0.04	2.3	0.092	15.0	54.0	0.99700	3.26	
3	11.2	0.280	0.56	1.9	0.075	17.0	60.0	0.99800	3.16	
4	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	
...	...	...	...	...	...	...	...	...	...	
1594	6.2	0.600	0.08	2.0	0.090	32.0	44.0	0.99490	3.45	
1595	5.9	0.550	0.10	2.2	0.062	39.0	51.0	0.99512	3.52	
1596	6.3	0.510	0.13	2.3	0.076	29.0	40.0	0.99574	3.42	
1597	5.9	0.645	0.12	2.0	0.075	32.0	44.0	0.99547	3.57	
1598	6.0	0.310	0.47	3.6	0.067	18.0	42.0	0.99549	3.39	

1599 rows × 12 columns

In [10]:
```
df.tail()
```

Out[10]:

	Fixed Acidity	Volatile Acidity	Citric Acid	Residual Sugar	Chlorides	Free Sulfur Dioxide	Total Sulfur Dioxide	Density	Ph	S
1594	6.2	0.600	0.08	2.0	0.090	32.0	44.0	0.99490	3.45	
1595	5.9	0.550	0.10	2.2	0.062	39.0	51.0	0.99512	3.52	
1596	6.3	0.510	0.13	2.3	0.076	29.0	40.0	0.99574	3.42	
1597	5.9	0.645	0.12	2.0	0.075	32.0	44.0	0.99547	3.57	
1598	6.0	0.310	0.47	3.6	0.067	18.0	42.0	0.99549	3.39	

CSV Questions

1. Please explain the difference between using .tail() and .head(), when might you use one over the other?

The difference between .tail() and .head() is .tail() returns the last *n* rows and .head() returns the first *n* rows. Where 5 is the default value for *n* if there is no argument given. Using .head() would be useful for getting an idea of the dataset and values, especially if it is a large dataset where you cannot reasonably print all of the values. Similarly, .tail() might be useful for the same purposes, but also seeing the last entries in a dataset and/or quickly seeing how many rows there are.

2. Please explain the difference between using inplace and not, what is the impact?

The difference between using inplace=True and not is that inplace=True modifies the dataframe itself without creating a copy, whereas nothing for that argument defaults it to inplace=False, which copies the DataFrame and prints it.

3. In relation to the above, what is an important consideration when working with Pandas?

Using inplace=True can save memory since it edits the dataframe and doesn't return it. Not using inplace=True creates and returns a new dataframe, leaving the original one unchanged and returns a new dataframe which uses more memory. If you plan on using the edited version of the DataFrame in the future and NOT the original version, then the best thing to do would be using inplace=True.

Text (TXT)

Steps:

1. Load the employees.txt file in using a pandas dataframe
2. Print the dataframe's first few elements using the `.head()` command to see what data you have
3. Complete the remaining tasks outlined in Canvas

4. Answer the questions

In [11]:
```
fileName = 'employees.txt'
employees = pd.read_table(fileName, delimiter="|")
employees.head()
```

Out[11]:

	Name	Age	Gender	Occupation	Salary
0	John Doe	28	Male	Software Engineer	85000
1	Jane Smith	34	Female	Data Scientist	95000
2	Sam Brown	45	Male	Project Manager	105000
3	Lisa White	29	Female	Designer	72000
4	Tom Hanks	38	Male	Product Manager	98000

In [12]:
```
employees.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10 entries, 0 to 9
Data columns (total 5 columns):
 #   Column      Non-Null Count  Dtype 
---  ------      --------------  ----- 
 0   Name        10 non-null     object
 1   Age         10 non-null     int64 
 2   Gender      10 non-null     object
 3   Occupation  10 non-null     object
 4   Salary      10 non-null     int64 
dtypes: int64(2), object(3)
memory usage: 528.0+ bytes
```

In [13]:
```
employees.describe()
```

Out[13]:

	Age	Salary
count	10.000000	10.000000
mean	36.000000	92500.000000
std	7.745967	11578.236289
min	27.000000	72000.000000
25%	29.500000	85500.000000
50%	35.000000	92500.000000
75%	41.000000	98000.000000
max	50.000000	112000.000000

In [14]:
```
employees['Occupation'].value_counts()
```

Out[14]:
```
Occupation
Software Engineer     2
Data Scientist        1
Project Manager       1
Designer              1
Product Manager       1
Marketing Director    1
Business Analyst      1
HR Manager            1
Sales Manager         1
Name: count, dtype: int64
```

In [15]:
```
employees.groupby(['Occupation'])[['Salary']].mean()
```

Out[15]:

Occupation	Salary
Business Analyst	87000.0
Data Scientist	95000.0
Designer	72000.0
HR Manager	98000.0
Marketing Director	112000.0
Product Manager	98000.0
Project Manager	105000.0
Sales Manager	90000.0
Software Engineer	84000.0

In [16]:
```
employees.groupby('Occupation').describe()
```

Out[16]:

	Age								Salary	
	count	mean	std	min	25%	50%	75%	max	count	mean
Occupation										
Business Analyst	1.0	31.0	NaN	31.0	31.00	31.0	31.00	31.0	1.0	87000.0
Data Scientist	1.0	34.0	NaN	34.0	34.00	34.0	34.00	34.0	1.0	95000.0
Designer	1.0	29.0	NaN	29.0	29.00	29.0	29.00	29.0	1.0	72000.0
HR Manager	1.0	50.0	NaN	50.0	50.00	50.0	50.00	50.0	1.0	98000.0
Marketing Director	1.0	42.0	NaN	42.0	42.00	42.0	42.00	42.0	1.0	112000.0
Product Manager	1.0	38.0	NaN	38.0	38.00	38.0	38.00	38.0	1.0	98000.0
Project Manager	1.0	45.0	NaN	45.0	45.00	45.0	45.00	45.0	1.0	105000.0
Sales Manager	1.0	36.0	NaN	36.0	36.00	36.0	36.00	36.0	1.0	90000.0
Software Engineer	2.0	27.5	0.707107	27.0	27.25	27.5	27.75	28.0	2.0	84000.0

Text Questions

1. How many entries and columns are there in the dataset?
2. What is the average age and average salary of the employees in the dataset?
3. Which occupation has the highest number of employees?
4. What is the average salary for each occupation, and which occupation has the highest average salary?
5. Please explain the difference between using .info() and .describe(), when might you use one over the other?
6. Please explain the significance of value_counts(), and how it might be used in data analysis?
7. In relation to the above, what is an important consideration when grouping data using groupby() in Pandas?

JSON

Steps:

1. Load the movies.json file using pandas
2. Print the dataframe's first few elements using the `.head()` command to see what data you have

3. Complete the remaining tasks outlined in Canvas
4. Answer the questions

In [17]:
```
movies = pd.read_json('movies.json')
movies.head()
```

Out[17]:

	Title	Year	Genre	Rating
0	The Shawshank Redemption	1994	Drama	9.3
1	The Godfather	1972	Crime	9.2
2	The Godfather: Part II	1974	Crime	9.0
3	The Dark Knight	2008	Action	9.0
4	12 Angry Men	1957	Drama	8.9

In [18]:
```
movies.isna()
```

Out[18]:

	Title	Year	Genre	Rating
0	False	False	False	False
1	False	False	False	False
2	False	False	False	False
3	False	False	False	False
4	False	False	False	False
5	False	False	False	False
6	False	False	False	False
7	False	False	False	False
8	False	False	False	False
9	False	False	False	False

In [27]:
```
movies.sample()
```

Out[27]:

	Title	Year	Genre	Rating
1	The Godfather	1972	Crime	9.2

JSON Questions

1. Are there any missing values in the dataset?

There are no missing values in the dataset.

2. What did you get in your random sample of the data?

I got the Godfather from 1972 with a rating of 9.2.

3. What is the purpose of taking a random sample of the DataFrame?

The purpose of taking a random sample of a DataFrame is it allows you to quickly inspect or analyze a smaller, representative subset of your data. It's useful for exploratory data analysis, model validation, and testing code on large datasets without processing the entire dataset.

Database(s)

SQLITE (this piece is a step beyond what we discussed, it will be an OPTIONAL exercise - or you can try it)

Steps:

1. Load the SQLite database.
    - Use read_sql() to load data from the database into a Pandas DataFrame.
    - Database Schema:
        - employees table:
            - Contains information about employees.
            - Columns: id, name, age, department, salary
        - departments table:
            - Contains information about departments.
            - Columns: id, name
2. Print the dataframe's first few elements using the `.head()` command to see what data you have
3. Complete the remaining tasks outlined in Canvas
4. Answer the questions

In [28]:
```
#from sqlite3 import connect
#conn = connect(':memory:')
#employees_db = pd.read_sql('employees.db', conn)
```

Database Questions

1. Provide the number of unique departments
2. Provide the average salary
3. Provide the total number of employees
4. Who are all the names of the employees in the engineering department?

Part 2: Unstructured Data

SpaceX API

Steps:

1. There will be some given code for the API URL and Connection.
2. Print the dataframe's first few elements using the `.head()` command to see what data you have
3. Complete the remaining tasks outlined in Canvas
4. Answer the questions

In [29]:
```
# SpaceX API URL for past launches
spacex_api_url = "https://api.spacexdata.com/v4/launches/past"
```

In [30]:
```
# Fetch data from the SpaceX API
response = requests.get(spacex_api_url)
launches_data = response.json()
```

In [31]:
```
# Load data into a Pandas DataFrame
df_launches = pd.DataFrame(launches_data)
```

In [32]:
```
df_launches.head()
```

Out[32]:

	fairings	links	static_fire_date_utc	static_fire_date_
0	{'reused': False, 'recovery_attempt': False, '...	{'patch': {'small': 'https://images2.imgbox.co...	2006-03-17T00:00:00.000Z	1.142554
1	{'reused': False, 'recovery_attempt': False, '...	{'patch': {'small': 'https://images2.imgbox.co...	None	
2	{'reused': False, 'recovery_attempt': False, '...	{'patch': {'small': 'https://images2.imgbox.co...	None	
3	{'reused': False, 'recovery_attempt': False, '...	{'patch': {'small': 'https://images2.imgbox.co...	2008-09-20T00:00:00.000Z	1.221869
4	{'reused': False, 'recovery_attempt': False, '...	{'patch': {'small': 'https://images2.imgbox.co...	None	

5 rows × 27 columns

In [34]:
```
df_launches.describe()
```

Out[34]:

	static_fire_date_unix	window	flight_number	date_unix
count	1.210000e+02	117.000000	187.000000	1.870000e+02
mean	1.520206e+09	2568.974359	94.000000	1.556247e+09
std	9.080036e+07	4389.948430	54.126395	1.034178e+08
min	1.142554e+09	0.000000	1.000000	1.143239e+09
25%	1.475417e+09	0.000000	47.500000	1.506172e+09
50%	1.529789e+09	0.000000	94.000000	1.590867e+09
75%	1.589368e+09	5280.000000	140.500000	1.639430e+09
max	1.650464e+09	18300.000000	187.000000	1.664986e+09

In [35]:
```
df_launches.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 187 entries, 0 to 186
Data columns (total 27 columns):
 #   Column                 Non-Null Count  Dtype  
---  ------                 --------------  -----  
 0   fairings               152 non-null    object 
 1   links                  187 non-null    object 
 2   static_fire_date_utc   121 non-null    object 
 3   static_fire_date_unix  121 non-null    float64
 4   net                    187 non-null    bool   
 5   window                 117 non-null    float64
 6   rocket                 187 non-null    object 
 7   success                186 non-null    object 
 8   failures               187 non-null    object 
 9   details                134 non-null    object 
 10  crew                   187 non-null    object 
 11  ships                  187 non-null    object 
 12  capsules               187 non-null    object 
 13  payloads               187 non-null    object 
 14  launchpad              187 non-null    object 
 15  flight_number          187 non-null    int64  
 16  name                   187 non-null    object 
 17  date_utc               187 non-null    object 
 18  date_unix              187 non-null    int64  
 19  date_local             187 non-null    object 
 20  date_precision         187 non-null    object 
 21  upcoming               187 non-null    bool   
 22  cores                  187 non-null    object 
 23  auto_update            187 non-null    bool   
 24  tbd                    187 non-null    bool   
 25  launch_library_id      69 non-null     object 
 26  id                     187 non-null    object 
dtypes: bool(4), float64(2), int64(2), object(19)
memory usage: 34.5+ KB
```

In [47]:
```
# count the number of launches per site
launch_site_counts = df_launches['launchpad'].value_counts()
print("Number of Launches per Launch Site:")
print(launch_site_counts)
```

```
Number of Launches per Launch Site:
launchpad
5e9e4501f509094ba4566f84    99
5e9e4502f509094188566f88    55
5e9e4502f509092b78566f87    28
5e9e4502f5090995de566f86     5
Name: count, dtype: int64
```

In [48]:
```
# Identify the most common launch site
most_common_launch_site = launch_site_counts.idxmax()
print("Most Common Launch Site:", most_common_launch_site)
```

```
Most Common Launch Site: 5e9e4501f509094ba4566f84
```

In [49]:
```
df_launches['launchpad'].value_counts()
```

Out[49]:
```
launchpad
5e9e4501f509094ba4566f84    99
5e9e4502f509094188566f88    55
5e9e4502f509092b78566f87    28
5e9e4502f5090995de566f86     5
Name: count, dtype: int64
```

In [53]:
```
df_launches['date_utc'] = pd.to_datetime(df_launches['date_utc'])

# Extract the year from the 'date_utc' column
df_launches['year'] = df_launches['date_utc'].dt.year

# Group by year and count the number of launches
launch_counts = df_launches['year'].value_counts().sort_index()

most_launches_year = launch_counts.idxmax()
most_launches_count = launch_counts.max()

print(f"The year with the most launches is {most_launches_year} with {most_l
```

```
The year with the most launches is 2022 with 44 launches.
```

In [56]:
```
launch_counts = df_launches['launchpad'].value_counts()

most_launches= launch_counts.idxmax()
most_launches_count = launch_counts.max()

print(f"The most common launch site: {most_common_launchsite} with {most_com
```

```
The most common launch site: 5e9e4501f509094ba4566f84 with 99 launches.
```

In [57]:
```
df_launches['date_utc'] = pd.to_datetime(df_launches['date_utc'])

# Find the most recent date
most_recent_date = df_launches['date_utc'].max()

# Retrieve the launch ID for the most recent date
most_recent_launch = df_launches[df_launches['date_utc'] == most_recent_date
most_recent_launch_id = most_recent_launch['id'].values[0]

print(f"The most recent launch ID is {most_recent_launch_id} and the date is
```

```
The most recent launch ID is 62dd70d5202306255024d139 and the date is 2022-1
0-05 16:00:00+00:00.
```

API Questions

1. What year has the most launches and how many total?

The year with the most launches i 2022 with 44 launches.

2. What is the most common launch site with how many launches?

The most common launch site is'5e9e4501f509094ba4566f84' with 99 launches,

3. What is the most recent launch ID AND DATE (per the data)?

The most launch ID is '62dd70d5202306255024d139' and the date is October 5, 2022 at 4pm.

Reading a Directory of Files

Steps:

1. Since we did not cover this, the below code is given
2. See if you can setup the directory and read the files
3. As a stretch goal, modify the code and see if you can read other types of files
4. Aswer the questions

In [64]:
```
#Path to the folder containing the reviews
reviews_path = 'reviews'

#Load all text files from the folder
reviews = []
csv_data = []
for filename in os.listdir(reviews_path):
    if filename.endswith('.txt'):
        with open(os.path.join(reviews_path, filename), 'r', encoding='utf-8
            reviews.append(file.read())
        print("Loaded Reviews:")
        print(reviews)
    if filename.endswith('.csv'):
        file_path = os.path.join(reviews_path, filename)
        data = pd.read_csv(file_path)
        csv_data.append(data)
        print("Loaded CSV Data:")
        for data in csv_data:
            print(data.head())
    if filename.endswith('.json'):
        file_path = os.path.join(reviews_path, filename)
        with open(file_path, 'r', encoding='utf-8') as file:
            data = json.load(file)
            json_data.append(data)
            print("Loaded JSON Data:")
            for data in json_data:
                print(data)
```

```
Loaded Reviews:
['This movie was fantastic! The plot was engaging and the characters were we
ll-developed.\n\n']
Loaded Reviews:
['This movie was fantastic! The plot was engaging and the characters were we
ll-developed.\n\n', 'I did not enjoy this movie. The storyline was boring an
d the acting was subpar.\n\n']
Loaded Reviews:
['This movie was fantastic! The plot was engaging and the characters were we
ll-developed.\n\n', 'I did not enjoy this movie. The storyline was boring an
d the acting was subpar.\n\n', 'An average film with some good moments. Not
the best, but worth watching.\n\n']
```

Directory of Files Questions

1. How does the loading of these files compare to the loading of structured data with pandas?

The loading of these files from a directory involves reading file contennt line by line or in chunks, whereas loading structured data with Pandas uses dataframes for easy manipulation and analysis- many more tools and generaly easier. Pandas also support various file formats and provides functions for cleaning and analyzing data.

3. Which do you prefer and why?

I prefer using Pandas for structured data because it simplifies data manipulation and analysis with built-in functions for handling missing values, filtering, and summarizing data, making it more efficient and user-friendly for data analysis tasks. I don't need to write for-loops to read it in/save it at the very minimum- one line of code vs multiple. I can do a lot more with the data after reading it in, too.

In [ ]: