1. How would you load a CSV dataset using Pandas in Python? Provide a step-by-step code example.

Need to follow below steps to load a csv file to the pandas data frame

1. Import the Pandas Library
    - Need to import the Pandas library.
    - you can install it using pip install pandas on the jupyter notebook cell
2. Loading the CSV File
    - Need to use the pd.read_csv() function to load the CSV file into a Pandas DataFrame.
3. Visualise the Data
    - After loading the data, you can print/visualise it by printing the first few rows or checking its structure by using different methods

Below are the methods.

- print() : This method will be used to print all the rows
- head() : This method will be uses to return first 5 rows of the dataframe
- tail() : This method will used to print the last 5 rows of the dataframe
- info() : This method will be used to get the summary of the DataFrame, including the number of non-null entries, data types of each column, and memory usage.

In [212…

```python
# 1. Import the Pandas library

import pandas as pd

# 2. Load the data.csv file
df = pd.read_csv('tips.csv')

# 3. Visualise the data
# Display the first few rows of the DataFrame
print("First few rows")
print("--------------")
print(df.head())

# Display the last few rows of the DataFrame
print("\n")
print("Last few rows")
print("--------------")
print(df.tail())

#  Display the DataFrame's structure
print("\n")
print("Info for the dataframe")
print("--------------")
print(df.info())
```

```
First few rows
--------------
   total_bill   tip     sex smoker  day    time  size
0       16.99  1.01  Female     No  Sun  Dinner     2
1       10.34  1.66    Male     No  Sun  Dinner     3
2       21.01  3.50    Male     No  Sun  Dinner     3
3       23.68  3.31    Male     No  Sun  Dinner     2
```

```
4       24.59  3.61  Female     No  Sun  Dinner     4


Last few rows
--------------
     total_bill   tip     sex smoker   day    time  size
239       29.03  5.92    Male     No   Sat  Dinner     3
240       27.18  2.00  Female    Yes   Sat  Dinner     2
241       22.67  2.00    Male    Yes   Sat  Dinner     2
242       17.82  1.75    Male     No   Sat  Dinner     2
243       18.78  3.00  Female     No  Thur  Dinner     2


Info for the dataframe
--------------
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 244 entries, 0 to 243
Data columns (total 7 columns):
 #   Column      Non-Null Count  Dtype  
---  ------      --------------  -----  
 0   total_bill  244 non-null    float64
 1   tip         244 non-null    float64
 2   sex         244 non-null    object 
 3   smoker      244 non-null    object 
 4   day         244 non-null    object 
 5   time        244 non-null    object 
 6   size        244 non-null    int64  
dtypes: float64(2), int64(1), object(4)
memory usage: 13.5+ KB
None
```

2. Explain the role and importance of exploratory data analysis in the context of data preprocessing.

Give examples of techniques used in EDA.

1. Overview of Exploratory Data Analysis In the vast realm of data science and analytics, understanding and making sense of data is crucial. Before delving into sophisticated modeling or forecasting, it's pivotal to grasp the basic nature of the data we're dealing with. This foundational stage of analysis is called Exploratory Data Analysis, often abbreviated as EDA.

EDA is the initial step in your data analysis process. Here, the focus is on understanding the patterns, spotting anomalies, testing a hypothesis, or checking assumptions related to a specific dataset. It's about being a detective, exploring the data to uncover its secrets and nuances. Often, it is during this process that the data speaks, revealing its essential stories and potentially guiding subsequent analysis or modeling. Various graphical representations, such as histograms, box plots, scatter plots, and more, aid in this exploration. Besides visual methods, EDA also involves statistical methods. For instance, understanding the distribution of a dataset, its central tendencies, or variance provides a comprehensive view of the data.

1. Roles and importance of Exploratory Data Analysis

- Visualisation Tool: EDA employs a variety of visualization techniques to provide a clear view of the data. Visual methods are an intuitive way to understand the intricacies of the dataset and help in revealing hidden patterns, relationships, or anomalies.

Histograms: Show the distribution of a single continuous variable. To visualize the distribution of age. Example:

```
sns.histplot(df['age'])
```

Boxplots: Used to detect outliers and understand the spread of data. Can show age distribution across different sex categories. Example:

```
sns.boxplot(x='sex', y='age', data=df)
```

Scatter plots: Help in understanding relationships between two continuous variables. can reveal correlations between height and weight. Example:

```
sns.scatterplot(x='height', y='weight', data=df)
```

Pairplots: Show relationships between multiple features. plots pairwise relationships for all numeric features. Example:

```
sns.pairplot(df)
```

- Statistical Analysis: Description: Beyond visuals, EDA delves into statistical measures to quantify the characteristics of the dataset. These metrics provide a foundational understanding of the data's central tendencies, spread, and relationships.

Example: describe() function in Pandas provides summary statistics for each numeric column

```
df.describe()
```

- Handling Missing Data: Description: During EDA, it's crucial to detect and manage missing or null values. Missing data can skew results, reduce the statistical power of tests, and lead to biased estimates. EDA provides methods to either impute these values or make informed decisions about removing them.

Filling Missing Data:¶ The missing values can be replaced by meaningful data, such as .

Mean Median Mode

Other Methods Forward filling Backward filling Predictive modeling

```
# Fill missing values in column 'Age' with the mean of the column
df['Age'].fillna(df['Age'].mean(), inplace=True)
# Fill missing values in the entire DataFrame with 0
df.fillna(0, inplace=True)
#Forward fill missing values
df.fillna(method='ffill', inplace=True)
#Backward fill missing values
df.fillna(method='bfill', inplace=True)
```

- Identifying Outliers: Description: Outliers are data points that significantly deviate from the other observations. While some outliers are genuine and provide valuable information, others might be due to errors and can distort analysis results. EDA aids in spotting and, if necessary, addressing these outliers.

Example: Boxplots and scatter plots can help spot outliers in continuous variables.

- Correlation Analysis: Description: Understanding how different variables relate to each other is essential. Correlation analysis in EDA assesses the linear relationship between two quantitative variables, helping in feature selection and understanding multicollinearity.

creates a heatmap of correlations between numeric features Example:

```
sns.heatmap(df.corr(), annot=True, cmap='coolwarm')
```

- Box-Cox Transformation: Description: Used to stabilize variance and make data more normally distributed, especially for skewed data. Example: Applying the transformation to reduce skewness:

  ```
  df['new_column'] = stats.boxcox(df['skewed_column'])[0]
  ```

1. Loading the dataset using pandas

- Loading the CSV file: Pandas is a widely-used Python library for data manipulation and analysis. One of its core functionalities is reading and writing data to various formats. When working with tabular data, the CSV (Comma-Separated Values) format is commonly encountered. To load a CSV file into a Pandas DataFrame, the read_csv() function is utilized. e.g.

```
import pandas as pd
# Load the CSV file into a DataFrame
df = pd.read_csv('path_to_file.csv')
# Display the first few rows of the DataFrame
print(df.head())
```

- Loading the excel file: Pandas, a popular Python library for data analysis, offers comprehensive tools to read and write data from diverse file formats. For Excel files, which are commonly used in business analytics and data reporting, Pandas provides the read_excel() function. e.g.

```
import pandas as pd
# Load the Excel file into a DataFrame
df = pd.read_excel('path_to_file.xlsx')
# Display the first few rows of the DataFrame
print(df.head())
```

1. Concept of DataFrame and Series Pandas Series: A Series in Pandas is one of the core data structures in the library. It represents a one-dimensional labeled array capable of holding data of any type (integer, string, float, python objects, etc.). e.g. ```python import pandas as pd data = { 'Name': ['John', 'Anna', 'Mike'], 'Age': [28, 22, 32], 'City': ['New York', 'Paris', 'London'] } df = pd.DataFrame(data) print(df)

   ```
   Pandas DataFrame:
   A DataFrame in Pandas is a two-dimensional, size-mutable, and
   heterogeneous tabular data structure with labeled axes (rows and
   columns). It can be thought of as a combination of multiple Pandas
   Series objects, where each column in the DataFrame is essentially a
   Series.
   e.g.

   ```python
   import pandas as pd
   data = {
       'Name': ['John', 'Anna', 'Mike'],
   ```
   ```

```
    'Age': [28, 22, 32],
    'City': ['New York', 'Paris', 'London']
}
df = pd.DataFrame(data)
print(df)
```

3. Discuss the process of handling missing values in a dataset. Provide Python code snippets demonstrating the

use of `fillna()` and `dropna()` functions in Pandas.

Handling missing values: For any dataset data preprocessing if there are presence of missing values it can lead to inaccurate analyses or model predictions. There are different approaches for handling missing values based on the dataset based on the missing values scenarios

Different approches

Remove Missing Data:

If the volume of missing data is minimal and if it is not affecting any changes in the model you can remove the rows or columns containing missing values.

Filling Missing Data:

The missing values can be replaced by meaningful data, such as the

1. Mean
2. Median
3. Mode

Other Methods

1. Forward filling
2. Backward filling
3. Predictive modeling.

Using fillna() and dropna() in Pandas

- Pandas provides two primary functions for handling missing data: fillna() and dropna().

1. fillna() The fillna() function is used to replace missing values (NaN) with a specified value or a method(mean,mode, forward or backward filling).

In [220…

```python
import pandas as pd

# Student DataFrame
student_df = pd.DataFrame({
    'Name': ['Jo', 'Mark', 'Nav', 'Kiran', 'Sam'],
    'Age': [None, 30, 32, None, 24],
    'Credits': [1, None, 3, 4, 5]
})

print("Original student Dataframe")
```

```
print("____________________________\n")
print(student_df)
```

```
Original student Dataframe
____________________________

    Name   Age  Credits
0     Jo   NaN      1.0
1   Mark  30.0      NaN
2    Nav  32.0      3.0
3  Kiran   NaN      4.0
4    Sam  24.0      5.0
```

In [222…

```
# Fill missing values in column 'Age' with the mean of the column
student_df['Age'].fillna(student_df['Age'].mean(), inplace=True)

print(" Student Dataframe replacing missing values in column Age with Mean value of colu
print("___________________________________________________________________________________\n")
print(student_df)
```

```
 Student Dataframe replacing missing values in column Age with Mean value of column
___________________________________________________________________________________

    Name        Age  Credits
0     Jo  28.666667      1.0
1   Mark  30.000000      NaN
2    Nav  32.000000      3.0
3  Kiran  28.666667      4.0
4    Sam  24.000000      5.0
```

In [224…

```
# Fill missing values in the entire DataFrame with 0
student_df.fillna(0, inplace=True)

print(" Student Dataframe replacing missing values with Zero's")
print("___________________________________________________________________________________\n")
print(student_df)
```

```
 Student Dataframe replacing missing values with Zero's
___________________________________________________________________________________

    Name        Age  Credits
0     Jo  28.666667      1.0
1   Mark  30.000000      0.0
2    Nav  32.000000      3.0
3  Kiran  28.666667      4.0
4    Sam  24.000000      5.0
```

In [226…

```
# Sample DataFrame
marks_df = pd.DataFrame({
    'Id': [1, 2, 3, 4, 5],
    'Marks': [None, 30, 32, None, 24]
})

print(marks_df)
```

```
   Id  Marks
0   1    NaN
1   2   30.0
2   3   32.0
3   4    NaN
4   5   24.0
```

In [228…

```
# Fill with Forward or Backward Filling:

# Replace missing values with the preceding (forward fill) or following (backward fill)
```

```
# Forward fill missing values
marks_df.fillna(method='ffill', inplace=True)
```

In [230…

```
# Marks df after forward filling
print("Marks Dataframe after forward filling")
print('______________________________________\n')
print(marks_df)
```

```
Marks Dataframe after forward filling
______________________________________

   Id  Marks
0   1    NaN
1   2   30.0
2   3   32.0
3   4   32.0
4   5   24.0
```

In [232…

```
# Backward fill missing values
marks_df.fillna(method='bfill', inplace=True)
```

In [234…

```
# Marks df after Backward filling
print("Marks Dataframe after Backward filling")
print('______________________________________\n')
print(marks_df)
```

```
Marks Dataframe after Backward filling
______________________________________

   Id  Marks
0   1   30.0
1   2   30.0
2   3   32.0
3   4   32.0
4   5   24.0
```

1. dropna() The dropna() function is used to remove rows or columns with missing values.

In [237…

```
# Sample DataFrame
df = pd.DataFrame({
    'A': [1, 2, 3, 4, 5],
    'B': [None, 30, 32, None, 24]
})
# Drop Rows with Missing Values:
# Remove any rows containing at least one missing value.

# Drop rows with any missing values
df_cleaned = df.dropna()

print("Removed rows having at least a missing value")
print('______________________________________\n')

print(df_cleaned)
```

```
Removed rows having at least a missing value
______________________________________

   A     B
1  2  30.0
2  3  32.0
4  5  24.0
```

In [239…

```
# Drop Columns with Missing Values:
# Remove any columns containing at least one missing value.

# Drop columns with any missing values
df_cleaned = df.dropna(axis=1)

print("Removed columns having at least a missing value")
print('______________________________________\n')
print(df_cleaned)
```

```
Removed columns having at least a missing value
______________________________________

   A
0  1
1  2
2  3
3  4
4  5
```

In [241…

```
# Drop Rows or Columns Only If All Values Are Missing:
# Drop rows only if all values are missing
df_cleaned = df.dropna(how='all')
# Remove rows or columns only if all values are missing.

# Drop columns only if all values are missing
df_cleaned = df.dropna(axis=1, how='all')
print("Drop columns if all values are missing")
print("_____________________\n")
print(df_cleaned)
```

```
Drop columns if all values are missing
_____________________

   A     B
0  1   NaN
1  2  30.0
2  3  32.0
3  4   NaN
4  5  24.0
```

4. Write Python code to perform data type conversion for a given dataset. Include examples of converting continuous and categorical data types.

In [244…

```
import pandas as pd
import numpy as np

# Load the dataset
mart_df = pd.read_csv('bigmart.csv')
```

In [246…

```
# Print first 5 rows
mart_df.head()
```

Out[246]:

	Item_Identifier	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP	Outlet_Identifier
0	FDA15	9.30	Low Fat	0.016047	Dairy	249.8092	OUT049
1	DRC01	5.92	Regular	0.019278	Soft Drinks	48.2692	OUT018
2	FDN15	17.50	Low Fat	0.016760	Meat	141.6180	OUT049

3	FDX07	19.20	Regular	0.000000	Fruits and Vegetables	182.0950	OUT010
4	NCD19	8.93	Low Fat	0.000000	Household	53.8614	OUT013

In [248…

```
# Info function output for the dataset
mart_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8523 entries, 0 to 8522
Data columns (total 12 columns):
 #   Column                     Non-Null Count  Dtype  
---  ------                     --------------  -----  
 0   Item_Identifier            8523 non-null   object 
 1   Item_Weight                7060 non-null   float64
 2   Item_Fat_Content           8523 non-null   object 
 3   Item_Visibility            8523 non-null   float64
 4   Item_Type                  8523 non-null   object 
 5   Item_MRP                   8523 non-null   float64
 6   Outlet_Identifier          8523 non-null   object 
 7   Outlet_Establishment_Year  8523 non-null   int64  
 8   Outlet_Size                6113 non-null   object 
 9   Outlet_Location_Type       8523 non-null   object 
 10  Outlet_Type                8523 non-null   object 
 11  Item_Outlet_Sales          8523 non-null   float64
dtypes: float64(4), int64(1), object(7)
memory usage: 799.2+ KB
```

In [250…

```
# Dropping all null values
mart_df = mart_df.dropna()
```

In [252…

```
print("Original df after removing null values")
print("______________________\n")
mart_df.info()
```

```
Original df after removing null values
______________________

<class 'pandas.core.frame.DataFrame'>
Index: 4650 entries, 0 to 8522
Data columns (total 12 columns):
 #   Column                     Non-Null Count  Dtype  
---  ------                     --------------  -----  
 0   Item_Identifier            4650 non-null   object 
 1   Item_Weight                4650 non-null   float64
 2   Item_Fat_Content           4650 non-null   object 
 3   Item_Visibility            4650 non-null   float64
 4   Item_Type                  4650 non-null   object 
 5   Item_MRP                   4650 non-null   float64
 6   Outlet_Identifier          4650 non-null   object 
 7   Outlet_Establishment_Year  4650 non-null   int64  
 8   Outlet_Size                4650 non-null   object 
 9   Outlet_Location_Type       4650 non-null   object 
 10  Outlet_Type                4650 non-null   object 
 11  Item_Outlet_Sales          4650 non-null   float64
dtypes: float64(4), int64(1), object(7)
memory usage: 472.3+ KB
```

In [254…

```
# ---- Continuous Data Type Conversion ---- #
# Converting 'Age' (float) to int
mart_df['Item_Weight'] = mart_df['Item_Weight'].astype(int)
```

```
# Converting 'target' (int) to float
mart_df['Outlet_Establishment_Year'] = mart_df['Outlet_Establishment_Year'].astype(float
```

In [256…

```
# Checking info after converting
print("After conversion of data type")
print("______________________\n")
mart_df.info()
```

```
After conversion of data type
______________________

<class 'pandas.core.frame.DataFrame'>
Index: 4650 entries, 0 to 8522
Data columns (total 12 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   Item_Identifier            4650 non-null   object
 1   Item_Weight                4650 non-null   int64
 2   Item_Fat_Content           4650 non-null   object
 3   Item_Visibility            4650 non-null   float64
 4   Item_Type                  4650 non-null   object
 5   Item_MRP                   4650 non-null   float64
 6   Outlet_Identifier          4650 non-null   object
 7   Outlet_Establishment_Year  4650 non-null   float64
 8   Outlet_Size                4650 non-null   object
 9   Outlet_Location_Type       4650 non-null   object
 10  Outlet_Type                4650 non-null   object
 11  Item_Outlet_Sales          4650 non-null   float64
dtypes: float64(4), int64(1), object(7)
memory usage: 472.3+ KB
```

In [258…

```
# Checking unique values for Outlet_Size column
mart_df['Outlet_Size'].unique()
```

Out[258]:

```
array(['Medium', 'High', 'Small'], dtype=object)
```

In [260…

```
# Checking unique values for Item_Fat_Content column
mart_df['Item_Fat_Content'].unique()
```

Out[260]:

```
array(['Low Fat', 'Regular', 'low fat', 'reg', 'LF'], dtype=object)
```

In [262…

```
# Mapping correct values for Item_Fat_Content column which are with different naming
mart_df['Item_Fat_Content'] = mart_df['Item_Fat_Content'].map({'low fat': 'Low Fat', 're
                                                                'Low Fat':'Low Fat', 'Regu
```

In [264…

```
# Checking unique values for Item_Fat_Content column after mapping with correct name
mart_df['Item_Fat_Content'].unique()
```

Out[264]:

```
array(['Low Fat', 'Regular'], dtype=object)
```

2. Convert categorical 'Item_Fat_Content' and 'Outlet_Size' to numerical values (Label Encoding)

Label encoding for 'Item_Fat_Content' and 'Outlet_Size' columns using map

In [267…

```
mart_df['Item_Fat_Content_Code'] = mart_df['Item_Fat_Content'].map({'Low Fat': 0, 'Regul
mart_df['Outlet_Size_Code'] = mart_df['Outlet_Size'].map({'Small': 0, 'Medium': 1,'High'
```

In [269…

```
print("After mapping")
print("_____________")
mart_df.head()
```

After mapping
____

Out[269]:

	Item_Identifier	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP	Outlet_Identifier
0	FDA15	9	Low Fat	0.016047	Dairy	249.8092	OUT049
1	DRC01	5	Regular	0.019278	Soft Drinks	48.2692	OUT018
2	FDN15	17	Low Fat	0.016760	Meat	141.6180	OUT049
4	NCD19	8	Low Fat	0.000000	Household	53.8614	OUT013
5	FDP36	10	Regular	0.000000	Baking Goods	51.4008	OUT018

5. Implement a Python function to merge two datasets based on a single key column using Pandas. Provide a code example and explain the result.

In [272…

```
import pandas as pd

# Function to merge two datasets based on a key column
def merge_datasets(df1, df2, key_column, how='inner'):
    """
    Merge two dataframes on a specified key column.

    :param df1: First dataframe
    :param df2: Second dataframe
    :param key_column: The column name on which to merge
    :param how: Type of merge - 'left', 'right', 'outer', 'inner' (default is 'inner')
    :return: Merged dataframe
    """
    merged_df = pd.merge(df1, df2, on=key_column, how=how)
    return merged_df

# Example dataframes with Enrollments and New grades data
df_enrollments = pd.DataFrame({
    'student_id':[101,102,103,105],
    'course_id':[301,302,301,304],
    'semester':['2024 Spring', '2024 Spring', '2024 Fall','2024 Spring'],
    'grades': ['A','B','A+','C']
})

df_new_grades = pd.DataFrame({
    'student_id':[101,102,103,105,106],
    'course_id':[301,302, 301,304,300],
    'semester':['2024 Spring', '2024 Spring', '2024 Fall','2024 Spring','2024 Spring'],
    'grades': ['A-','B+','A','B-', 'A']
} )


# Merge the datasets on the 'student_id' column
merged_df = merge_datasets(df_enrollments, df_new_grades, 'student_id', how='inner')

print("Merged DataFrame (Inner Join):")
print("______________________________")
print(merged_df)

# Try different types of merge (e.g., outer join)
outer_merged_df = merge_datasets(df_enrollments, df_new_grades, 'student_id', how='outer
```

```
print("\nMerged DataFrame (Outer Join):")
print("______________________________")
print(outer_merged_df)

# Try different types of merge (e.g., right join)
right_merged_df = merge_datasets(df_enrollments, df_new_grades, 'student_id', how='right

print("\nMerged DataFrame (Right Join):")
print("______________________________")
print(right_merged_df)
```

```
Merged DataFrame (Inner Join):
______________________________
   student_id  course_id_x   semester_x grades_x  course_id_y   semester_y  \
0         101          301  2024 Spring        A          301  2024 Spring
1         102          302  2024 Spring        B          302  2024 Spring
2         103          301    2024 Fall       A+          301    2024 Fall
3         105          304  2024 Spring        C          304  2024 Spring

  grades_y
0       A-
1       B+
2        A
3       B-

Merged DataFrame (Outer Join):
______________________________
   student_id  course_id_x   semester_x grades_x  course_id_y   semester_y  \
0         101        301.0  2024 Spring        A          301  2024 Spring
1         102        302.0  2024 Spring        B          302  2024 Spring
2         103        301.0    2024 Fall       A+          301    2024 Fall
3         105        304.0  2024 Spring        C          304  2024 Spring
4         106          NaN          NaN      NaN          300  2024 Spring

  grades_y
0       A-
1       B+
2        A
3       B-
4        A

Merged DataFrame (Right Join):
______________________________
   student_id  course_id_x   semester_x grades_x  course_id_y   semester_y  \
0         101        301.0  2024 Spring        A          301  2024 Spring
1         102        302.0  2024 Spring        B          302  2024 Spring
2         103        301.0    2024 Fall       A+          301    2024 Fall
3         105        304.0  2024 Spring        C          304  2024 Spring
4         106          NaN          NaN      NaN          300  2024 Spring

  grades_y
0       A-
1       B+
2        A
3       B-
4        A
```

Explanation: Function Definition:

The function merge_datasets(df1, df2, key_column,how='inner') takes two DataFrames (df1, df2) and the name of the key column (key_column) as input. It merges the two DataFrames using pd.merge(df1, df2, on=key_column, how='join type'), which merges on the specified key column. Sample DataFrames:

- df_enrollments contains student information (student ID, courseId, Semester and Grades).

- df_new_grades contains student information (student ID, courseId, Semester and Grades). Both datasets share the student_id column, which is used as the key for merging. Merging:

The pd.merge() function merges the two DataFrames based on the student_id column, keeping only the rows where student_id is present in both DataFrames (this is the default "inner join" behavior).

6. How would you use the `groupby()` functionality in Pandas to perform aggregate functions like sum, average, max, and min on a dataset? Provide Python code demonstrating each aggregate function.

Grouping

In general, grouping data in Pandas works as follows:

```
df.groupby(by=grouping_columns)[columns_to_show].function()
```

1. First, the `groupby` method divides the `grouping_columns` by their values. They become a new index in the resulting dataframe.
2. Then, columns of interest are selected (`columns_to_show`). If `columns_to_show` is not included, all non groupby clauses will be included.
3. Finally, one or several functions are applied to the obtained groups per selected columns.

Here is an example where we group the data according to the values of the `Churn` variable and display statistics of three columns in each group:

Loading MentalHealth dataset MentalHealthSurvey.csv

Kaggle https://www.kaggle.com/datasets/abdullahashfaqvirk/student-mental-health-survey?resource=download&select=MentalHealthSurvey.csv

In [278…

```
import pandas as pd
health_df = pd.read_csv("MentalHealthSurvey.csv")
```

In [280…

```
# print top 5 rows
health_df.head()
```

Out[280]:

	gender	age	university	degree_level	degree_major	academic_year	cgpa	residential_status	campu
0	Male	20	PU	Undergraduate	Data Science	2nd year	3.0-3.5	Off-Campus	
1	Male	20	UET	Postgraduate	Computer Science	3rd year	3.0-3.5	Off-Campus	
2	Male	20	FAST	Undergraduate	Computer Science	3rd year	2.5-3.0	Off-Campus	
3	Male	20	UET	Undergraduate	Computer Science	3rd year	2.5-3.0	On-Campus	
4	Female	20	UET	Undergraduate	Computer Science	3rd year	3.0-3.5	Off-Campus	

5 rows × 21 columns

In [282…
```
# print info
health_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 87 entries, 0 to 86
Data columns (total 21 columns):
 #   Column                    Non-Null Count  Dtype 
---  ------                    --------------  ----- 
 0   gender                    87 non-null     object
 1   age                       87 non-null     int64 
 2   university                87 non-null     object
 3   degree_level              87 non-null     object
 4   degree_major              87 non-null     object
 5   academic_year             87 non-null     object
 6   cgpa                      87 non-null     object
 7   residential_status        87 non-null     object
 8   campus_discrimination     87 non-null     object
 9   sports_engagement         87 non-null     object
 10  average_sleep             87 non-null     object
 11  study_satisfaction        87 non-null     int64 
 12  academic_workload         87 non-null     int64 
 13  academic_pressure         87 non-null     int64 
 14  financial_concerns        87 non-null     int64 
 15  social_relationships      87 non-null     int64 
 16  depression                87 non-null     int64 
 17  anxiety                   87 non-null     int64 
 18  isolation                 87 non-null     int64 
 19  future_insecurity         87 non-null     int64 
 20  stress_relief_activities  87 non-null     object
dtypes: int64(10), object(11)
memory usage: 14.4+ KB
```

In [284…
```
# Getting unique values for degree_major column
health_df['degree_major'].unique()
```

Out[284]:
```
array(['Data Science', 'Computer Science', 'Software Engineering',
       'Information Technology'], dtype=object)
```

In [286…
```
# display statistics of residential_status columns in each degree_major group:
health_df.groupby(['degree_major'])['residential_status'].describe(percentiles=[])
```

Out[286]:

degree_major	count	unique	top	freq
Computer Science	34	2	Off-Campus	23
Data Science	41	2	Off-Campus	31
Information Technology	9	2	Off-Campus	8
Software Engineering	3	1	Off-Campus	3

By passing a list of functions to `agg()`:

Below code snippet uses the groupby() method to group the data in health_df by the "academic_year" column and then applies aggregate functions (mean, std, min, and max) to the "study_satisfaction" column. This operation helps summarize the satisfaction levels for each academic year by computing key statistical metrics.

With agg(), you can apply multiple aggregate functions on different columns at once. In this example, we calculated the std,mean, max, and min for the study_satisfaction column using academic_year as the group by .

In [290…

```
health_df.groupby(["academic_year"])["study_satisfaction"].agg(['mean', 'std', 'min', 'm
```

Out[290]:

academic_year	mean	std	min	max
1st year	4.058824	1.179141	1	5
2nd year	3.933333	1.032796	2	5
3rd year	3.821429	0.772374	3	5
4th year	3.800000	1.316561	1	5

Sum (sum): Calculates the total salary and total age for each department.

Mean (mean): Finds the average salary and average age for each department.

Max (max): Determines the maximum salary and age in each department.

Min (min): Finds the minimum salary and age in each department.

In [293…

```
# Group by 'degree_level' and calculate the sum of 'study_satisfaction' and 'social_rela
grouped_sum = health_df.groupby(['degree_level'])[['study_satisfaction','social_relation

print("\nSum of study_satisfaction and social_relationships by degree_level:")
print("____________")
print(grouped_sum)
```

```
Sum of study_satisfaction and social_relationships by degree_level:
____________
               study_satisfaction  social_relationships
degree_level
Postgraduate                    9                     5
Undergraduate                 333                   237
```

In [295…

```
# Group by 'residential_status' and calculate the average of 'depression'
grouped_avg = health_df.groupby(['residential_status'])['depression'].mean()

print("\n Average of depression by residential_status:")
print("____________")
print(grouped_avg)
```

```
 Average of depression by residential_status:
____________
residential_status
Off-Campus    3.230769
On-Campus     3.181818
Name: depression, dtype: float64
```

In [297…

```
# Group by 'academic_year' and calculate the max of 'social_relationships'
grouped_max = health_df.groupby(['academic_year'])['social_relationships'].max()

print("\n Max of social_relationships by academic_year:")
print("____________")
print(grouped_max)
```

```
 Max of social_relationships by academic_year:
____________
academic_year
1st year    5
2nd year    5
```

```
3rd year    5
4th year    4
Name: social_relationships, dtype: int64
```

In [299…

```python
# Group by 'academic_year' and calculate the min of 'financial_concerns'
grouped_min = health_df.groupby(['academic_year'])['financial_concerns'].min()

print("\n Min of financial_concerns by academic_year:")
print("____________")
print(grouped_min)
```

```
 Min of financial_concerns by academic_year:
____________
academic_year
1st year    1
2nd year    1
3rd year    1
4th year    1
Name: financial_concerns, dtype: int64
```

In [ ]: