5CS037 - Concepts and Technologies of AI.
Worksheet-2: Exploratory Data Analysis with Pandas -Part-1.

Prepared By: Siman Giri {Module Leader - 5CS037}

November 19, 2025

1 Instructions

{Disclaimer:Exploratory Data Analysis is designed to be two part exercise and this is Part 1, which mostly focuses on use of Pandas for efficient data cleaning and data transformation operation.}

This worksheet contains programming exercises on Data cleaning and Data Transformation with pandas based on the material discussed from the slides.This is a graded exercise and are to be completed on your own and is compulsory to submit.
Please answer the questions below using python in the Jupyter Notebook and follow the guidelines below:

• This worksheet must be completed individually.

• All the solutions must be written in Jupyter Notebook.

• Our Recommendation - Google Colaboratory.

• Dataset used for this session can be downloaded from shared drive.

Figure 1: Getting Started with Pandas.

2 Getting Started with Pandas.

This Section contains all the sample code from the slides and are here for your reference, you are highly recommended to run all the code with some of the input changed in order to understand the meaning of the operations and also to be able to solve all the exercises from further sections.

- Cautions!!!:
  - This Guide may not contain sample output, as we expect you to re-write the code and observe the output.
  - If found: any error or bugs, please report to your instructor and Module leader. *{Will hugely appreciate your effort.}*

2.1 Building Blocks of Pandas:

1. Data Structure - Series:

Sample Code from Slide - 15 - Creating a Simple Series.

```
import pandas as pd
# Creating a simple Series
data = [10, 20, 30, 40]
series = pd.Series(data)
print(series)
```

2. Data Structure - Index:

Sample Code from Slide - 16 to 18 - Types of Index.

```
# Default Index
import pandas as pd
series = pd.Series([10, 20, 30])
print(series.index)
# Output: RangeIndex(start=0, stop=3, step=1)
# User Defined:
series = pd.Series([10, 20, 30],index=['a','b','c'])
print(series)
# Output:
# a 10
# b 20
# c 30
#datestime index
dates = pd.date_range('2023-01-01', periods=3)
series = pd.Series([10, 20, 30], index=dates)
print(series)
# Output:
# 2023-01-01 10
# 2023-01-02 20
# 2023-01-03 30
```

Sample Code from Slide - 19 - Acess and Reset Index.

```
#Access
print(series.index)
# Set or Reset Index
series.index = ['x', 'y', 'z'] # Series
# For DataFrame
df = pd.DataFrame({'A': [1, 2]}, index=['row1', 'row2'])
df.reset_index(inplace=True)
# Converts the index into a column
```

3. Data Structure - DataFrames.

Sample Code from Slide - 22 - Creating DataFrames.

```
#Transforming in-built data structures-DataFrame
#Style-1
import pandas as pd
pd.DataFrame({'Bob': ['I liked it.','It was awful'], 'Sue': ['Pretty good.', 'Bland.']}) #Style-2
pd.DataFrame({'Bob': ['I liked it.', 'It was awful.'], 'Sue': ['Pretty good.', 'Bland.']}, index=['Product A',
            'Product B'])
```

4. DataFrames - Loading Data to DataFrames.

Sample Code from Slide - 23 - Loading Data To DataFrames.

```
#Importing Data from file
import pandas as pd
# path to your dataset must be given to built in read_csv("Your path") function. dataset =
pd.read_csv("/data/Week02/bank.csv")
dataset.head()
dataset.tail()
dataset.info()
# Run the above code and observe the output.
```

3. Data Structure - DataFrames.

Sample Code from Slide - 22 - Creating DataFrames.

```
#Transforming in-built data structures-DataFrame
#Style-1
import pandas as pd
pd.DataFrame({'Bob': ['I liked it.','It was awful'], 'Sue': ['Pretty good.', 'Bland.']}) #Style-2
pd.DataFrame({'Bob': ['I liked it.', 'It was awful.'], 'Sue': ['Pretty good.', 'Bland.']}, index=['Product A',
            'Product B'])
```

5. DataFrames - Writing DataFrames to CSV.

Sample Code from Slide - 24 - Writing DataFrames to CSV.

```
#Importing Data from file
import pandas as pd
data = {'Name': ['Alice', 'Bob', 'Charlie'],'City': ['New York', 'San Francisco', 'Los Angeles']} df =
pd.DataFrame(data) # creating a DataFrame
#Writing DataFrame to csv.
df.to_csv('output.csv', index=False)
# Run the above code and observe the output.
```

2.2 Basic Operation on Data: Data Inspection and Exploration: 1. First

Data Inspection and Exploration:

Sample Code from Slide - 27 to 28 - First Data Inspection and Exploration.

```
import pandas as pd
# Sample DataFrame
data = {
    'Name': ['Alice', 'Bob', 'Charlie'],
    'Age': [25, 30, 35],
    'Salary': [50000, 60000, 70000]
}
df = pd.DataFrame(data)
# View the first two rows
print(df.head(2))
# View the last row
print(df.tail(1))
# DataFrame information
print(df.info())
# Summary statistics
print(df.describe())
# Check dimensions of the DataFrame
print(f"The DataFrame has {df.shape[0]} rows and {df.shape[1]} columns.")
# Access the 'Age' column
print(df['Age'])
# Select rows by numerical index
print(df.iloc[0]) # First row
# Select rows by condition
print(df.loc[df['Age'] > 30]) # Rows where Age > 30
```

Understanding DataFrame.info()

The DataFrame.info() method provides a concise summary of a DataFrame, which is particularly useful for getting an overview of its structure.
Output Components:

1. Class Type: Shows that the object is a pandas DataFrame.
2. RangeIndex: Indicates the number of rows in the DataFrame.
3. Column Information:
    - Column names
    - Data types (int64, float64, object, etc.).
    - Non-null counts (useful for spotting missing values).
4. Memory Usage: Displays the approximate memory usage of the DataFrame.

Sample Code and Output Explaination for df.info().

```
import pandas as pd
# Sample DataFrame
data = {
    'Name': ['Alice', 'Bob', 'Charlie'],
    'Age': [25, 30, None],
    'Salary': [50000, 60000, 55000]
}
df = pd.DataFrame(data)
# Check info
df.info()
#---------------------------------------------------------------------
#*************************OUTPUT Explaination**************************
#---------------------------------------------------------------------
<class 'pandas.core.frame.DataFrame'> # The object type
RangeIndex: 3 entries, 0 to 2 # Number of rows
Data columns (total 3 columns): # Number of columns
 # Column Non-Null Count Dtype
--- ------ -------------- -----
 0 Name 3 non-null object # All rows have values
 1 Age 2 non-null float64 # One missing value
 2 Salary 3 non-null int64 # All rows have values
dtypes: float64(1), int64(1), object(1) # Data types
memory usage: 200.0+ bytes # Memory used
```

Understanding DataFrame.describe()

The DataFrame.describe() method provides summary statistics for numerical columns in a DataFrame by default.
Output Components:

1. ount: Number of non-null values.
2. Mean: The average value.
3. Standard Deviation (std): Measure of data dispersion.
4. Minimum (min): The smallest value.
5. 25%, 50%, 75%: Percentile values (25th, 50th/median, and 75th).
6. Maximum (max): The largest value.

Sample Code and Output Explaination for df.describe().

```
# Generate descriptive statistics
import pandas as pd
# Sample DataFrame
data = {
    'Name': ['Alice', 'Bob', 'Charlie'],
    'Age': [25, 30, None],
    'Salary': [50000, 60000, 55000]
}
df = pd.DataFrame(data)
# Check summary statistics
df.describe() # Generate descriptive statistics
#-----------------------------------------------------------------------
#*************************OUTPUT Explaination*************************
#-----------------------------------------------------------------------
              Age Salary
count 2.000000 3.000000 # Non-null values
mean 27.500000 55000.000000 # Average values
std 3.535534 5000.000000 # Dispersion of values
min 25.000000 50000.000000 # Minimum values
25% 26.250000 52500.000000 # 25th percentile
50% 27.500000 55000.000000 # Median
75% 28.750000 57500.000000 # 75th percentile
max 30.000000 60000.000000 # Maximum values
```

2. Filtering and Modifying Data:

Sample Code from Slide - 29 - Filtering Rows and Columns.

```
import pandas as pd
df = pd.DataFrame({
    'Name': ['Alice', 'Bob', 'Charlie'],
    'Age': [25, 30, 35],
    'Salary': [50000, 60000, 70000]
})
# Filter rows where Age > 28
filtered_rows = df[df['Age'] > 28]
print(filtered_rows)
#Select Specific Columns
# Select only 'Name' and 'Salary' columns
selected_columns = df[['Name', 'Salary']]
print(selected_columns)
```

Sample Code from Slide - 30 - Droping and Adding a Columns.

```
# Drop the 'Salary' column
df_without_salary = df.drop(columns=['Salary'])
print(df_without_salary)
# Drop the row with index 1 (Bob)
df_without_row = df.drop(index=1)
print(df_without_row)
# Add a new column for Bonus
df['Bonus'] = df['Salary'] * 0.1
print(df)
```

2.3 Basic Operation on Data - Data Wrangling - Common Data Cleaning operations:

1. Handling Missing Values:

Sample Code from Slide - 34 - Handling Missing values - Adding Missing Values.

```
# Adding Some Missing Values
import pandas as pd
from sklearn.datasets import load_iris
import numpy as np
iris = load_iris() # Load the Iris dataset
iris_df = pd.DataFrame(data=np.c_[iris['data'], iris['target']], columns=iris['feature_names'] + [' target'])
np.random.seed(42) # Introduce missing values randomly
mask = np.random.rand(*iris_df.shape) < 0.1 # 10%
iris_df[mask] = np.nan
print("Missing Values in Iris Dataset:")
print(iris_df.isnull().sum())
```

Sample Code from Slide - 35 - Handling Missing values - Techniques for Filling Missing Values.

```
# Filling missing values with forward fill (ffill), mean, median, and 0
iris_df_ffill = iris_df.ffill()
iris_df_mean = iris_df.fillna(iris_df.mean())
iris_df_median = iris_df.fillna(iris_df.median())
iris_df_zero = iris_df.fillna(0)
# Expand iris_df with filled columns
iris_df_expanded = pd.concat([iris_df, iris_df_ffill.add_suffix('_ffill'), iris_df_mean.add_suffix('
```

```
_mean'),iris_df_median.add_suffix('_median'),iris_df_zero.add_suffix('_zero')], axis=1) # Display the head of the expanded DataFrame
print("\nDataset after Filling Missing Values:")
print(iris_df_expanded.head())
```

2. Some Common Operation performed for cleaning data.

Sample Code from Slide - 36 to 37 - Some Common Operations on Data Cleaning.

```
#-------------------------------------------------------------
# ----------Trimming Whitespaces:-------------------------------
#-------------------------------------------------------------
df = pd.DataFrame({'Name': ['Alice ', 'Bob '], 'Age': [25, 30]})
df['Name'] = df['Name'].str.strip()
#-------------------------------------------------------------
#------------Changing Datatype:-------------------------------
#-------------------------------------------------------------
df = pd.DataFrame({'Age': ['25', '30', '35']})
# Change 'Age' column data type to integer
df['Age'] = df['Age'].astype(int)
print(df)
#-------------------------------------------------------------
#------------Renaming Columns:--------------------------------
#-------------------------------------------------------------
# Rename columns
df = pd.DataFrame({'Name': ['Alice', 'Bob'], 'Age': [25, 30]})
df = df.rename(columns={'Name': 'Full Name', 'Age': 'Years'})
print(df)
#-------------------------------------------------------------
#-------------Removing Duplicates:----------------------------
#-------------------------------------------------------------
# Remove duplicate rows
df = pd.DataFrame({'Name': ['Alice', 'Bob', 'Alice'], 'Age': [25, 30, 25]})
df = df.drop_duplicates()
print(df)
```

3. Data Transformation - DataFrame Reshaping.

Sample Code from Slide - 39 to 40 - DataFrame Reshaping - Pivot and Melt.

```
#-----------------Pivoting-----------------------------------
import pandas as pd
# Sample DataFrame
data = {'Date': ['2024-01-01', '2024-01-01', '2024-01-02', '2024-01-02'],
        'City': ['Kathmandu', 'Pokhara', 'Kathmandu', 'Pokhara'],
        'Temperature': [15, 18, 16, 19]}
df = pd.DataFrame(data)
# Pivot: Reshape data to show cities as columns
pivoted_df = df.pivot(index='Date', columns='City', values='Temperature')
print(pivoted_df)
#-------------------------------------------------------------------------
#-------------------------Melting-----------------------------------------
#-------------------------------------------------------------------------
# Melt: Convert wide data back to long format
melted_df = pd.melt(pivoted_df.reset_index(), id_vars=['Date'],
                    var_name='City', value_name='Temperature')
print(melted_df)
```

4. Data Transformation - Data Scaling .

Sample Code from Slide - 41 - Data Transformation - Min-Max Scaling.

```
import pandas as pd
from sklearn.datasets import load_iris
iris = load_iris() # Load the Iris dataset
iris_df = pd.DataFrame(data=iris['data'], columns=iris['feature_names'])
# Min-Max Scaling using Pandas
iris_minmax_scaled = (iris_df - iris_df.min()) / (iris_df.max() - iris_df.min()) print("Original Iris
DataFrame:")
print(iris_df.head())
print("\nMin-Max Scaled Iris DataFrame:")
print(iris_minmax_scaled.head()) # Display scaled data
```

5. Data Transformation - Handling Categorical Variables:

Sample Code from Slide - 42 - Handling Categorical Variables - Ordinal or Label Encoding.

```
import pandas as pd
# Sample DataFrame with ordinal categories
df = pd.DataFrame({'Category': ['Low', 'Medium', 'High', 'Low', 'High']})
# Ordinal encoding using map
ordinal_mapping = {'Low': 1, 'Medium': 2, 'High': 3}
df['Category_Ordinal'] = df['Category'].map(ordinal_mapping)
print(df)
```

Sample Code from Slide - 43 - Handling Categorical Variables - One Hot Encoding.

```
import pandas as pd
df_municipalities = pd.DataFrame({'Municipality': ['Kathmandu', 'Bhaktapur', 'Lalitpur', 'Madhyapur Thimi', 'Kirtipur']})
one_hot_encoding = pd.get_dummies(df_municipalities['Municipality'], prefix='Municipality') df_encoded =
pd.concat([df_municipalities, one_hot_encoding], axis=1)
print(df_encoded)# Display the result
```

6. Merging and joining DataFrames:

Sample Code from Slide - 44 - Merging and Joining DataFrames - Concatenation.

```
import pandas as pd
# Sample DataFrames
df1 = pd.DataFrame({'A': [1, 2], 'B': [3, 4]})
df2 = pd.DataFrame({'A': [5, 6], 'B': [7, 8]})
# Row-wise concatenation
combined_rows = pd.concat([df1, df2], axis=0)
print("Row-wise concatenation:")
print(combined_rows)
# Column-wise concatenation
combined_cols = pd.concat([df1, df2], axis=1)
print("\nColumn-wise concatenation:")
print(combined_cols)
```

Sample Code from Slide - 46 - Merging and Joining DataFrames - Merge.

```
# Sample DataFrames
df1 = pd.DataFrame({'ID': [1, 2, 3], 'Name': ['Alice', 'Bob', 'Charlie']})
df2 = pd.DataFrame({'ID': [2, 3, 4], 'Score': [85, 90, 88]})
# Inner join
inner_merged = pd.merge(df1, df2, on='ID', how='inner')
print("Inner Join:")
print(inner_merged)
# Left join
left_merged = pd.merge(df1, df2, on='ID', how='left')
print("\nLeft Join:")
print(left_merged)
# Outer join
outer_merged = pd.merge(df1, df2, on='ID', how='outer')
print("\nOuter Join:")
print(outer_merged)
```

3 To - Do - Task

Please Complete all the problem listed below.

3.1 Warming Up Exercises - Basic Inspection and Exploration: Problem 1

- Data Read, Write and Inspect:

Complete all following Task:

- Dataset for the Task: "bank.csv"

1. Load the provided dataset and import in pandas DataFrame.

```
import pandas as pd

# 1. Load the dataset
bank = pd.read_csv("bank.csv")
bank.head()
```

	age	job	marital	education	default	balance	housing	loan	contact	day	month	duration	campaign	pdays	previous	poutcome	y
0	58	management	married	tertiary	no	2143	yes	no	unknown	5	may	261	1	-1	0	unknown	no
1	44	technician	single	secondary	no	29	yes	no	unknown	5	may	151	1	-1	0	unknown	no
2	33	entrepreneur	married	secondary	no	2	yes	yes	unknown	5	may	76	1	-1	0	unknown	no
3	47	blue-collar	married	unknown	no	1506	yes	no	unknown	5	may	92	1	-1	0	unknown	no
4	33	unknown	single	unknown	no	1	no	no	unknown	5	may	198	1	-1	0	unknown	no

2. Check info of the DataFrame and identify following:

   (a) columns with dtypes=object

   (b) unique values of those columns.

   (c) check for the total number of null values in each column.

```
import pandas as pd

# Load your dataset (replace with the correct filename if needed)
df = pd.read_csv("bank.csv")

# (a) Columns with dtype = object
object_cols = df.select_dtypes(include="object").columns
print("Object columns:", object_cols.tolist())

# (b) Unique values of those columns
for col in object_cols:
    print(f"\nUnique values in column '{col}':")
    print(df[col].unique())

# (c) Total number of null values in each column
print("\nNull values per column:")
print(df.isnull().sum())
```

```
Object columns: ['job', 'marital', 'education', 'default', 'housing', 'loan', 'contact', 'month', 'poutcome', 'y']

Unique values in column 'job':
['management' 'technician' 'entrepreneur' 'blue-collar' 'unknown'
 'retired' 'admin.' 'services' 'self-employed' 'unemployed' 'housemaid'
 'student']

Unique values in column 'marital':
['married' 'single' 'divorced']

Unique values in column 'education':
['tertiary' 'secondary' 'unknown' 'primary']

Unique values in column 'default':
['no' 'yes']

Unique values in column 'housing':
['yes' 'no']

Unique values in column 'loan':
['no' 'yes']

Unique values in column 'contact':
['unknown' 'cellular' 'telephone']

Unique values in column 'month':
['may' 'jun' 'jul' 'aug' 'oct' 'nov' 'dec' 'jan' 'feb' 'mar' 'apr' 'sep']

Unique values in column 'poutcome':
['unknown' 'failure' 'other' 'success']

Unique values in column 'y':
['no' 'yes']

Null values per column:
age          0
job          0
marital      0
education    0
default      0
balance      0
housing      0
loan         0
contact      0
day          0
month        0
duration     0
campaign     0
pdays        0
previous     0
poutcome     0
y            0
dtype: int64
```

3. Drop all the columns with dtypes object and store in new DataFrame, also write the DataFrame in ".csv" with name "banknumericdata.csv"

```python
import pandas as pd

# Load the dataset
df = pd.read_csv("bank.csv")

# Identify object dtype columns
object_cols = df.select_dtypes(include="object").columns
print("Dropping object columns:", object_cols.tolist())

# Drop object columns
df_numeric = df.drop(columns=object_cols)

# Save numeric DataFrame to CSV
df_numeric.to_csv("banknumericdata.csv", index=False)

print("Numeric DataFrame saved as banknumericdata.csv")
```

```
Dropping object columns: ['job', 'marital', 'education', 'default', 'housing', 'loan', 'contact', 'month', 'poutcome', 'y']
Numeric DataFrame saved as banknumericdata.csv
```

4. Read "banknumericdata.csv" and Find the summary statistics.

```python
import pandas as pd

# Read the numeric-only CSV
df_num = pd.read_csv("banknumericdata.csv")

# Summary statistics
print(df_num.describe())
```

```
                age        balance           day      duration      campaign  \
count  45211.000000   45211.000000  45211.000000  45211.000000  45211.000000
mean      40.936210    1362.272058     15.806419    258.163080      2.763841
std       10.618762    3044.765829      8.322476    257.527812      3.098021
min       18.000000   -8019.000000      1.000000      0.000000      1.000000
25%       33.000000      72.000000      8.000000    103.000000      1.000000
50%       39.000000     448.000000     16.000000    180.000000      2.000000
75%       48.000000    1428.000000     21.000000    319.000000      3.000000
max       95.000000  102127.000000     31.000000   4918.000000     63.000000

              pdays      previous
count  45211.000000  45211.000000
mean      40.197828      0.580323
std      100.128746      2.303441
min       -1.000000      0.000000
25%       -1.000000      0.000000
50%       -1.000000      0.000000
75%       -1.000000      0.000000
max      871.000000    275.000000
```

Problem 2 - Data Imputations:

Complete all the following Task:

- Dataset for the Task: "medical_student.csv"

1. Load the provided dataset and import in pandas DataFrame.

```python
df_medical = pd.read_csv("medical_students.csv")

print(df_medical.head())
df_medical.info()
```

```
...     Student ID   Age  Gender      Height     Weight Blood Type        BMI  \
0         1.0  18.0  Female  161.777924  72.354947          O  27.645835
1         2.0   NaN    Male  152.069157  47.630941          B        NaN
2         3.0  32.0  Female  182.537664  55.741083          A  16.729017
3         NaN  30.0    Male  182.112867  63.332207          B  19.096042
4         5.0  23.0  Female         NaN  46.234173          O        NaN

   Temperature  Heart Rate  Blood Pressure  Cholesterol Diabetes Smoking
0          NaN        95.0           109.0        203.0       No     NaN
1    98.714977        93.0           104.0        163.0       No      No
2    98.260293        76.0           130.0        216.0      Yes      No
3    98.839605        99.0           112.0        141.0       No     Yes
4    98.480008        95.0             NaN        231.0       No      No
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200000 entries, 0 to 199999
Data columns (total 13 columns):
 #   Column          Non-Null Count   Dtype
---  ------          --------------   -----
 0   Student ID      180000 non-null  float64
 1   Age             180000 non-null  float64
 2   Gender          180000 non-null  object
 3   Height          180000 non-null  float64
 4   Weight          180000 non-null  float64
 5   Blood Type      180000 non-null  object
 6   BMI             180000 non-null  float64
 7   Temperature     180000 non-null  float64
 8   Heart Rate      180000 non-null  float64
 9   Blood Pressure  180000 non-null  float64
 10  Cholesterol     180000 non-null  float64
 11  Diabetes        180000 non-null  object
 12  Smoking         180000 non-null  object
dtypes: float64(9), object(4)
memory usage: 19.8+ MB
```

2. Check info of the DataFrame and identify column with missing (null) values.

```python
# Check DataFrame structure
df_medical.info()

# Identify columns with missing values
print("\nMissing values per column:")
print(df_medical.isnull().sum())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200000 entries, 0 to 199999
Data columns (total 13 columns):
 #   Column          Non-Null Count   Dtype  
---  ------          --------------   -----  
 0   Student ID      180000 non-null  float64
 1   Age             180000 non-null  float64
 2   Gender          180000 non-null  object 
 3   Height          180000 non-null  float64
 4   Weight          180000 non-null  float64
 5   Blood Type      180000 non-null  object 
 6   BMI             180000 non-null  float64
 7   Temperature     180000 non-null  float64
 8   Heart Rate      180000 non-null  float64
 9   Blood Pressure  180000 non-null  float64
 10  Cholesterol     180000 non-null  float64
 11  Diabetes        180000 non-null  object 
 12  Smoking         180000 non-null  object 
dtypes: float64(9), object(4)
memory usage: 19.8+ MB

Missing values per column:
Student ID        20000
Age               20000
Gender            20000
Height            20000
Weight            20000
Blood Type        20000
BMI               20000
```

```
Missing values per column:
Student ID        20000
Age               20000
Gender            20000
Height            20000
Weight            20000
Blood Type        20000
BMI               20000
Temperature       20000
Heart Rate        20000
Blood Pressure    20000
Cholesterol       20000
Diabetes          20000
Smoking           20000
dtype: int64
```

3. For the column with missing values fill the values using various techniques we discussed above. Try to explain why did you select the particular methods for particular column.

```python
import pandas as pd

# Load dataset
df = pd.read_csv("medical_students.csv")

# Check missing values
missing_summary = df.isnull().sum()
print("Missing values per column:\n", missing_summary)

# Technique 1: Forward fill for time-series or ordered data
df_ffill = df.fillna(method="ffill")

# Technique 2: Mean fill for continuous numeric columns
for col in df.select_dtypes(include=["float64", "int64"]).columns:
    if df[col].isnull().sum() > 0:
        df[col] = df[col].fillna(df[col].mean())

# Technique 3: Mode fill for categorical columns
for col in df.select_dtypes(include="object").columns:
    if df[col].isnull().sum() > 0:
        df[col] = df[col].fillna(df[col].mode()[0])

# Technique 4: Fill with 0 for columns where missing means absence
if "failures" in df.columns:
    df["failures"] = df["failures"].fillna(0)

print("\nMissing values after imputation:\n", df.isnull().sum())
```

```
Missing values per column:
 Student ID        20000
Age                20000
Gender             20000
Height             20000
Weight             20000
Blood Type         20000
BMI                20000
Temperature        20000
Heart Rate         20000
Blood Pressure     20000
Cholesterol        20000
Diabetes           20000
Smoking            20000
dtype: int64
/tmp/ipython-input-3936218557.py:11: FutureWarning: DataFrame.fillna with 'method' is deprecated and will raise in a future version. Use obj.ffill() or obj.bfill() instead.
  df_ffill = df.fillna(method="ffill")

Missing values after imputation:
 Student ID        0
Age                0
Gender             0
Height             0
Weight             0
Blood Type         0
BMI                0
Temperature        0
Heart Rate         0
Blood Pressure     0
Cholesterol        0
Diabetes           0
Smoking            0
dtype: int64
```

4. Check for any duplicate values present in Dataset and do necessary to manage the duplicate items. *{*Hint: dataset.duplicated.sum()*}*

```
# Count duplicate rows
duplicate_count = df_medical.duplicated().sum()
print("Number of duplicate rows:", duplicate_count)

# Drop duplicates if any
df_medical_cleaned = df_medical.drop_duplicates()

# Confirm removal
print("Remaining duplicates after drop:", df_medical_cleaned.duplicated().sum())
```

```
Number of duplicate rows: 7644
Remaining duplicates after drop: 0
```

3.2 Exercises - Data Cleaning and Transformations with ”Titanic Dataset”: Dataset Used: "titanic.csv"

Problem - 1:

Create a DataFrame that is subsetted for the columns ’Name’, ’Pclass’, ’Sex’, ’Age’, ’Fare’, and ’Survived’. Retain only those rows where ’Pclass’ is equal to 1, representing first-class passengers. What is the mean, median, maximum value, and minimum value of the ’Fare’ column?

```python
import pandas as pd

# Load Titanic dataset
df = pd.read_csv("Titanic.csv")

# Subset required columns
subset = df[["Name", "Pclass", "Sex", "Age", "Fare", "Survived"]]

# Filter for first-class passengers
first_class = subset[subset["Pclass"] == 1]

# Fare statistics
fare_mean = first_class["Fare"].mean()
fare_median = first_class["Fare"].median()
fare_max = first_class["Fare"].max()
fare_min = first_class["Fare"].min()

print("Mean Fare:", fare_mean)
print("Median Fare:", fare_median)
print("Max Fare:", fare_max)
print("Min Fare:", fare_min)
```

```
Mean Fare: 84.1546875
Median Fare: 60.287499999999994
Max Fare: 512.3292
Min Fare: 0.0
```

Problem - 2:

How many null values are contained in the ’Age’ column in your subsetted DataFrame? Once you’ve found this out, drop them from your DataFrame.

```python
import pandas as pd

# Load Titanic dataset
df = pd.read_csv("Titanic.csv")

# Subset required columns
subset = df[["Name", "Pclass", "Sex", "Age", "Fare", "Survived"]]

# Filter for first-class passengers
first_class = subset[subset["Pclass"] == 1]

# Count nulls in Age column
null_age_count = first_class["Age"].isnull().sum()
print("Number of null values in Age column:", null_age_count)

# Drop rows with null Age
first_class_cleaned = first_class.dropna(subset=["Age"])

# Confirm removal
print("Remaining nulls in Age column after drop:", first_class_cleaned["Age"].isnull().sum())
```

```
Number of null values in Age column: 30
Remaining nulls in Age column after drop: 0
```

Problem - 3:

The ’Embarked’ column in the Titanic dataset contains categorical data representing the ports of embarka tion:

- ’C’ for Cherbourg
- ’Q’ for Queenstown
- ’S’ for Southampton

Task:

1. Use one-hot encoding to convert the ’Embarked’ column into separate binary columns (’Embarked C’, ’Embarked Q’, ’Embarked S’).
2. Add these new columns to the original DataFrame.
3. Drop the original ’Embarked’ column.
4. Print the first few rows of the modified DataFrame to verify the changes.

```python
import pandas as pd

# Load Titanic dataset
df = pd.read_csv("Titanic.csv")

# One-hot encode the 'Embarked' column
embarked_dummies = pd.get_dummies(df["Embarked"], prefix="Embarked")

# Add new columns to the original DataFrame
df = pd.concat([df, embarked_dummies], axis=1)

# Drop the original 'Embarked' column
df = df.drop(columns=["Embarked"])

# Print first few rows to verify
print(df.head())
```

```
   PassengerId  Survived  Pclass  \
0            1         0       3
1            2         1       1
2            3         1       3
3            4         1       1
4            5         0       3

                                                Name     Sex   Age  SibSp  \
0                            Braund, Mr. Owen Harris    male  22.0      1
1  Cumings, Mrs. John Bradley (Florence Briggs Th...  female  38.0      1
2                             Heikkinen, Miss. Laina  female  26.0      0
3       Futrelle, Mrs. Jacques Heath (Lily May Peel)  female  35.0      1
4                           Allen, Mr. William Henry    male  35.0      0

   Parch            Ticket     Fare Cabin  Embarked_C  Embarked_Q  Embarked_S
0      0         A/5 21171   7.2500   NaN       False       False        True
1      0          PC 17599  71.2833   C85        True       False       False
2      0  STON/O2. 3101282   7.9250   NaN       False       False        True
3      0            113803  53.1000  C123       False       False        True
4      0            373450   8.0500   NaN       False       False        True
```

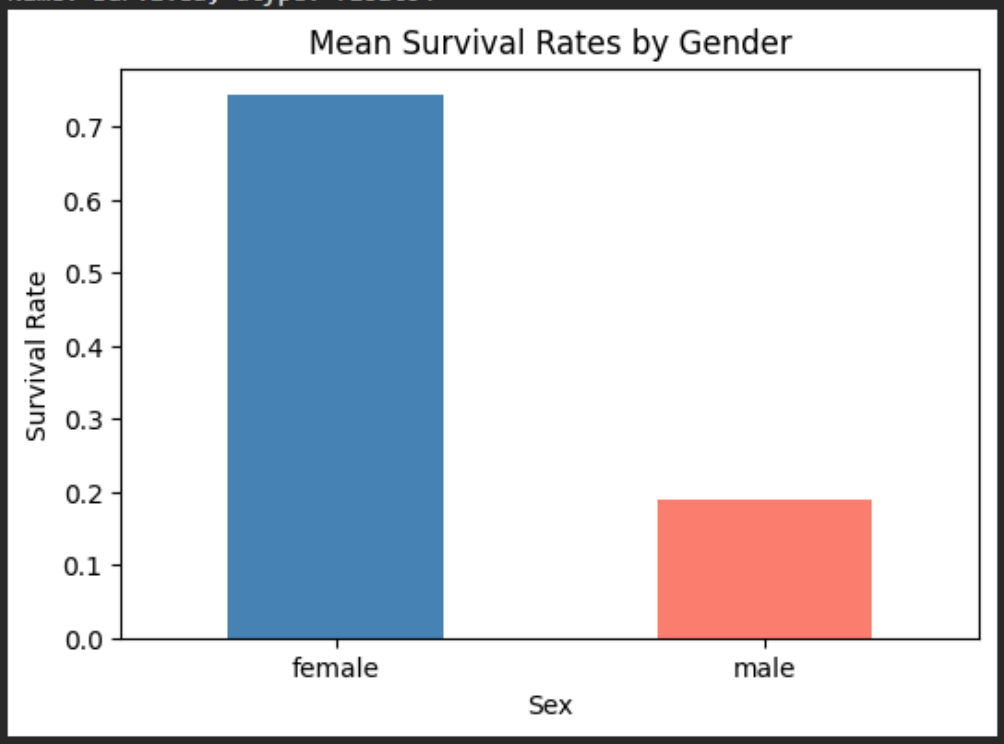
Problem - 4:

Compare the mean survival rates (’Survived’) for the different groups in the ’Sex’ column. Draw a visual ization to show how the survival distributions vary by gender.

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

# Load Titanic dataset
df = pd.read_csv("Titanic.csv")

# Compare mean survival rates by gender
survival_rates = df.groupby("Sex")["Survived"].mean()
print("Mean survival rates by gender:\n", survival_rates)

# Visualization
plt.figure(figsize=(6,4))
survival_rates.plot(kind="bar", color=["steelblue", "salmon"])
plt.title("Mean Survival Rates by Gender")
plt.ylabel("Survival Rate")
plt.xlabel("Sex")
plt.xticks(rotation=0)
plt.show()
```

```
Mean survival rates by gender:
 Sex
female    0.742038
male      0.188908
Name: Survived, dtype: float64
```


```
Mean survival rates by gender:
 Sex
female    0.742038
male      0.188908
Name: Survived, dtype: float64
```

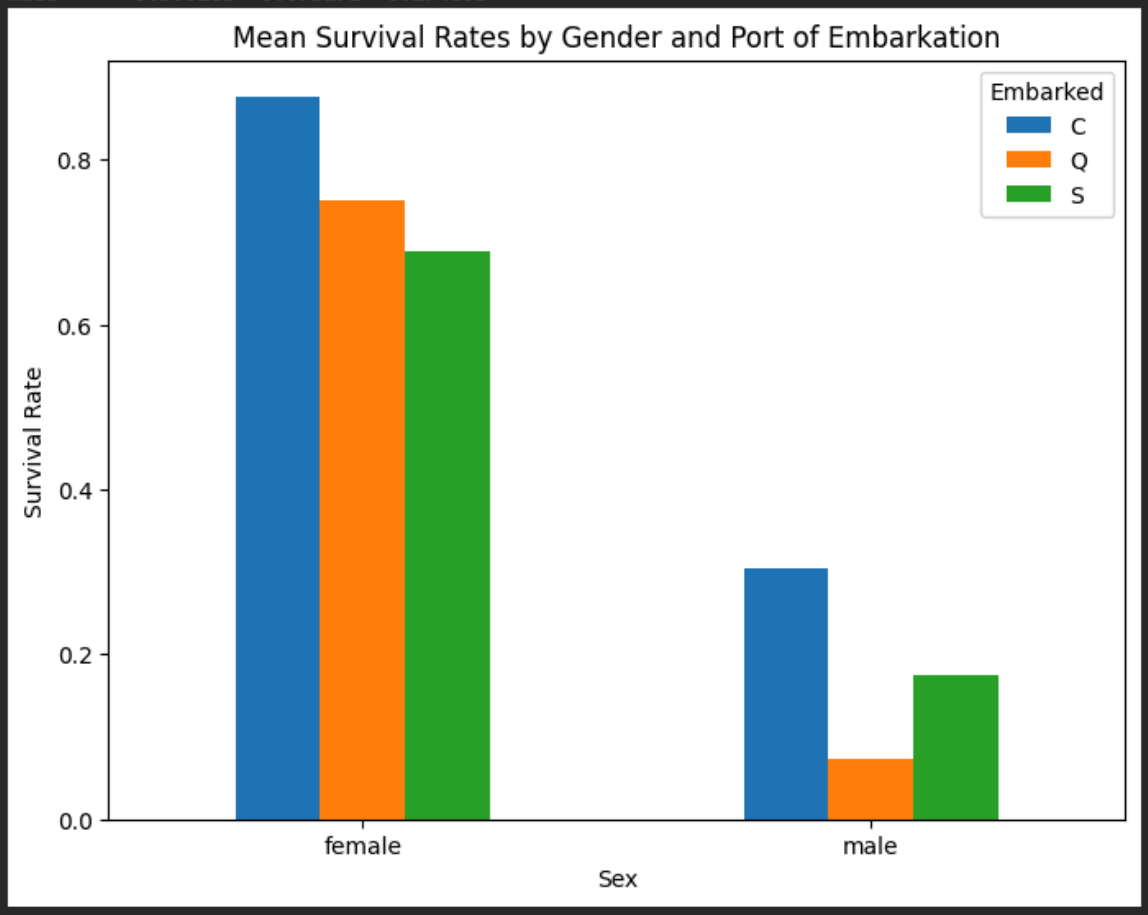
Problem - 5:

Draw a visualization that breaks your visualization from Exercise 3 down by the port of embarkation (’Em barked’). In this instance, compare the ports ’C’ (Cherbourg), ’Q’ (Queenstown), and ’S’ (Southampton).

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

# Load Titanic dataset
df = pd.read_csv("Titanic.csv")

# Group by Sex and Embarked, compute mean survival rates
survival_rates = df.groupby(["Sex", "Embarked"])["Survived"].mean().unstack()

print("Mean survival rates by Sex and Embarked:\n", survival_rates)

# Visualization
survival_rates.plot(kind="bar", figsize=(8,6))
plt.title("Mean Survival Rates by Gender and Port of Embarkation")
plt.ylabel("Survival Rate")
plt.xlabel("Sex")
plt.xticks(rotation=0)
plt.legend(title="Embarked")
plt.show()
```

```
Mean survival rates by Sex and Embarked:
 Embarked         C         Q         S
Sex
female    0.876712  0.750000  0.689655
male      0.305263  0.073171  0.174603
```

```
Mean survival rates by Sex and Embarked:
Embarked         C         Q         S
Sex
female    0.876712  0.750000  0.689655
male      0.305263  0.073171  0.174603
```



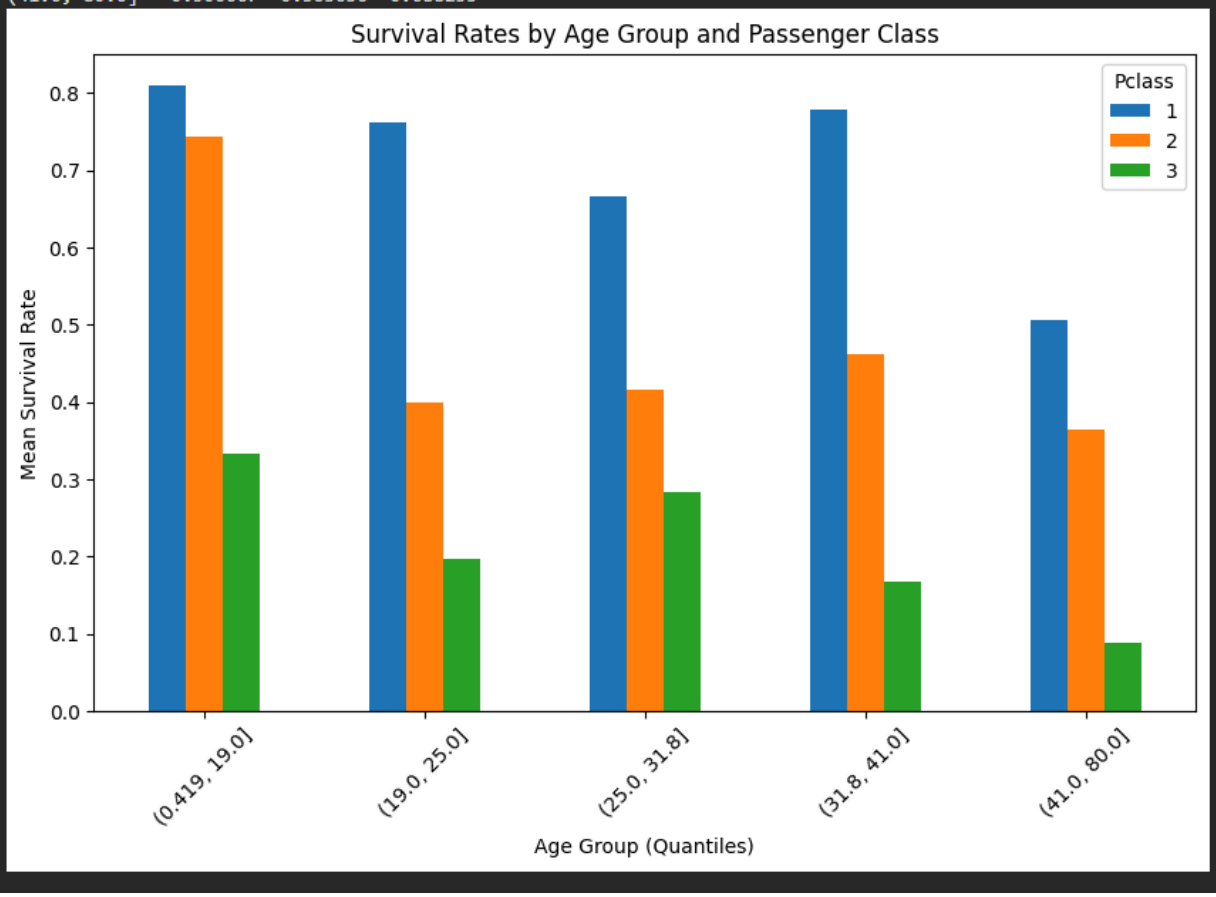
Problem - 6*{Optional}*:

Show how the survival rates (’Survived’) vary by age group and passenger class (’Pclass’). Break up the ’Age’ column into five quantiles in your DataFrame, and then compare the means of ’Survived’ by class and age group. Draw a visualization using a any plotting library to represent this graphically.

```
import pandas as pd
import matplotlib.pyplot as plt

# Load Titanic dataset
df = pd.read_csv("Titanic.csv")

# Break Age into 5 quantile groups
df["AgeGroup"] = pd.qcut(df["Age"].dropna(), 5)

# Compute mean survival rates by AgeGroup and Pclass
survival_by_group = df.groupby(["AgeGroup", "Pclass"])["Survived"].mean().unstack()

print("Mean survival rates by AgeGroup and Pclass:\n", survival_by_group)

# Visualization
survival_by_group.plot(kind="bar", figsize=(10,6))
plt.title("Survival Rates by Age Group and Passenger Class")
plt.ylabel("Mean Survival Rate")
plt.xlabel("Age Group (Quantiles)")
plt.xticks(rotation=45)
plt.legend(title="Pclass")
plt.show()
```

```
/tmp/ipython-input-2291309212.py:11: FutureWarning: The default of observed=False is deprecated and will be changed to True in a future version of pandas. Pass observed=False to retain current behavior or observed=True to a
  survival_by_group = df.groupby(["AgeGroup", "Pclass"])["Survived"].mean().unstack()
Mean survival rates by AgeGroup and Pclass:
 Pclass               1         2         3
AgeGroup
(0.419, 19.0]  0.809524  0.742857  0.333333
(19.0, 25.0]   0.761905  0.400000  0.197674
(25.0, 31.8]   0.666667  0.416667  0.283582
(31.8, 41.0]   0.777778  0.461538  0.166667
(41.0, 80.0]   0.506667  0.363636  0.088235
```

```
(19.0, 25.0]   0.761905  0.400000  0.197674
(25.0, 31.8]   0.666667  0.416667  0.283582
(31.8, 41.0]   0.777778  0.461538  0.166667
(41.0, 80.0]   0.506667  0.363636  0.088235
```



The - End