```
# Import required libraries
import seaborn as sns
import pandas as pd

# Load Titanic dataset from seaborn
titanic = sns.load_dataset("titanic")

# Display first 5 rows
titanic.head()
```

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive	alone
0	0	3	male	22.0	1	0	7.2500	S	Third	man	True	NaN	Southampton	no	False
1	1	1	female	38.0	1	0	71.2833	C	First	woman	False	C	Cherbourg	yes	False
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	False	NaN	Southampton	yes	True
3	1	1	female	35.0	1	0	53.1000	S	First	woman	False	C	Southampton	yes	False
4	0	3	male	35.0	0	0	8.0500	S	Third	man	True	NaN	Southampton	no	True

Next steps: Generate code with titanic View recommended plots New interactive sheet

```
# Summary of missing values before cleaning
titanic.isnull().sum()
```

	0
survived	0
pclass	0
sex	0
age	177
sibsp	0
parch	0
fare	0
embarked	2
class	0
who	0
adult_male	0
deck	688
embark_town	2
alive	0
alone	0

dtype: int64

```
# Fill missing 'age' with mean
titanic['age'].fillna(titanic['age'].mean(), inplace=True)

# Fill missing 'embarked' with mode
titanic['embarked'].fillna(titanic['embarked'].mode()[0], inplace=True)

# Drop irrelevant column 'deck' (similar to Cabin in Kaggle dataset)
titanic.drop(columns=['deck'], inplace=True)

# Drop 'embark_town' (duplicate info of 'embarked')
titanic.drop(columns=['embark_town'], inplace=True)
```

```
/tmp/ipython-input-1848358971.py:2: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assigr
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting valu

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].me
```

```
  titanic['age'].fillna(titanic['age'].mean(), inplace=True)
/tmp/ipython-input-1848358971.py:5: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assigr
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting valu

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].me


  titanic['embarked'].fillna(titanic['embarked'].mode()[0], inplace=True)
```

```
# Summary of missing values after cleaning
titanic.isnull().sum()
```

	0
survived	0
pclass	0
sex	0
age	0
sibsp	0
parch	0
fare	0
embarked	0
class	0
who	0
adult_male	0
alive	0
alone	0

dtype: int64

```
discussion = """
Handling missing values is crucial in data preprocessing because:
1. Machine learning models cannot handle missing data directly.
2. Missing values may lead to biased results if ignored.
3. Filling Age with mean preserves the central tendency.
4. Filling Embarked with mode keeps the most common category.
5. Dropping Cabin (deck) removes a column with too many missing values,
   preventing noise and overfitting.

Thus, proper handling of missing data ensures better model performance
and reliable analysis.
"""
print(discussion)
```

```
Handling missing values is crucial in data preprocessing because:
1. Machine learning models cannot handle missing data directly.
2. Missing values may lead to biased results if ignored.
3. Filling Age with mean preserves the central tendency.
4. Filling Embarked with mode keeps the most common category.
5. Dropping Cabin (deck) removes a column with too many missing values,
   preventing noise and overfitting.

Thus, proper handling of missing data ensures better model performance
and reliable analysis.
```

2)Student Dataset – Standardizing Categorical Data

```
import pandas as pd

# Create mock student dataset with inconsistent gender entries
data = {
```

```
    'Name': ['Amit', 'Divya', 'Rahul', 'Sneha', 'Pavan', 'Priya'],
    'Gender': ['M', 'Male', 'male', 'F', 'Female', 'female'],
    'Age': [20, 21, 22, 20, 23, 21]
}

students = pd.DataFrame(data)

# Show dataset before cleaning
students
```

	Name	Gender	Age
0	Amit	M	20
1	Divya	Male	21
2	Rahul	male	22
3	Sneha	F	20
4	Pavan	Female	23
5	Priya	female	21

Next steps: Generate code with students | View recommended plots | New interactive sheet

```
# Convert all gender values to lowercase
students['Gender'] = students['Gender'].str.lower()

# Replace standardized values
students['Gender'] = students['Gender'].replace({
    'm': 'Male',
    'male': 'Male',
    'f': 'Female',
    'female': 'Female'
})

# Show dataset after cleaning
students
```

	Name	Gender	Age
0	Amit	Male	20
1	Divya	Male	21
2	Rahul	Male	22
3	Sneha	Female	20
4	Pavan	Female	23
5	Priya	Female	21

Next steps: Generate code with students | View recommended plots | New interactive sheet

```
# Unique values in Gender after cleaning
students['Gender'].unique()
```

```
array(['Male', 'Female'], dtype=object)
```

```
discussion = """
Consistency in categorical data is important because:
1. Inconsistent labels (like 'M', 'male', 'Male') are treated as different categories.
2. This causes errors in grouping, filtering, and statistical analysis.
3. Standardizing ensures uniformity, improves data quality,
   and prevents misleading results.
4. Clean categorical data is essential for reliable machine learning models.

Example: If 'Male' appears in 3 different forms, the model might treat
them as 3 separate genders, which is incorrect.
"""
print(discussion)
```

```
Consistency in categorical data is important because:
1. Inconsistent labels (like 'M', 'male', 'Male') are treated as different categories.
2. This causes errors in grouping, filtering, and statistical analysis.
3. Standardizing ensures uniformity, improves data quality,
   and prevents misleading results.
4. Clean categorical data is essential for reliable machine learning models.

Example: If 'Male' appears in 3 different forms, the model might treat
them as 3 separate genders, which is incorrect.
```