```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
# tiny synthetic DataFrame with Titanic-like columns
df = pd.DataFrame({
'survived': [0,1,1,0,1,0,1,0],
'pclass': [3,1,2,3,1,2,3,1],
'SEX': [' male',np.nan,'female','MaLe','female ','MALE','male','female'],
'age': [12,38,26,35,np.nan,54,2,27],
'sibsp': [1,3,0,1,0,0,3,0],
'parch': [0,1,0,0,0,0,1,2],
'fARE': [7.25,np.nan,7.925,8.05,53.1,51.8625,21.075,11.1333],
'embarked': ['S','C','S','S','S','S','S',np.nan]
})
# Keep an original copy for comparisons later
df_orig = df.copy()
print(df.head())
```

```
   survived  pclass      SEX   age  sibsp  parch    fARE embarked
0         0       3     male  12.0      1      0   7.250        S
1         1       1      NaN  38.0      3      1     NaN        C
2         1       2   female  26.0      0      0   7.925        S
3         0       3     MaLe  35.0      1      0   8.050        S
4         1       1  female    NaN      0      0  53.100        S
```

```
#1 Drop unnecessary columns from a DataFrame.
# Drop Unncessary columns usinf drop()
df.drop(['embarked'], axis=1, inplace=True)
print(df.head()) # print
```

```
   survived  pclass      SEX   age  sibsp  parch    fARE
0         0       3     male  12.0      1      0   7.250
1         1       1      NaN  38.0      3      1     NaN
2         1       2   female  26.0      0      0   7.925
3         0       3     MaLe  35.0      1      0   8.050
4         1       1  female    NaN      0      0  53.100
```

```
#2 Rename columns for better readability.
# Rename pclass and sibsp using rename()
df.rename(columns={'pclass': 'Passenger_Class', 'sibsp':'Siblings_Spouses'}, inplace=True)
print(df.head()) # print
```

```
   survived  Passenger_Class      SEX   age  Siblings_Spouses  parch    fARE
0         0                3     male  12.0                 1      0   7.250
1         1                1      NaN  38.0                 3      1     NaN
2         1                2   female  26.0                 0      0   7.925
3         0                3     MaLe  35.0                 1      0   8.050
4         1                1  female    NaN                 0      0  53.100
```

```
#3 Reorder columns in a DataFrame.
 # Suffle the column name based on order
df = df[['Passenger_Class', 'age', 'SEX', 'survived', 'Siblings_Spouses', 'parch', 'fARE']]
print(df.head()) # print
```

```
   Passenger_Class   age      SEX  survived  Siblings_Spouses  parch    fARE
0                3  12.0     male         0                 1      0   7.250
1                1  38.0      NaN         1                 3      1     NaN
2                2  26.0   female         1                 0      0   7.925
3                3  35.0     MaLe         0                 1      0   8.050
4                1   NaN  female          1                 0      0  53.100
```

```
#4 Reset the index of a DataFrame
# Reset the index to original dataframe
df.reset_index(drop=True, inplace=True)
print(df.head()) # print
```

```
   Passenger_Class   age      SEX  survived  Siblings_Spouses  parch    fARE
0                3  12.0     male         0                 1      0   7.250
1                1  38.0      NaN         1                 3      1     NaN
2                2  26.0   female         1                 0      0   7.925
3                3  35.0     MaLe         0                 1      0   8.050
4                1   NaN  female          1                 0      0  53.100
```

```
#5 Set a specific column as the new index
# Set index value as survived
df.set_index('survived', inplace=True)
print(df.head()) # print
```

```
          Passenger_Class   age      SEX  Siblings_Spouses  parch    fARE
survived
0                       3  12.0     male                 1      0   7.250
1                       1  38.0      NaN                 3      1     NaN
1                       2  26.0   female                 0      0   7.925
0                       3  35.0     MaLe                 1      0   8.050
1                       1   NaN  female                  0      0  53.100
```

```
#6 Detect duplicate rows and remove them
# Remove duplicate rows using drop_duplicate()
df.drop_duplicates(inplace=True)
print(df.head()) # print
```

```
          Passenger_Class   age      SEX  Siblings_Spouses  parch    fARE
survived
0                       3  12.0     male                 1      0   7.250
1                       1  38.0      NaN                 3      1     NaN
1                       2  26.0   female                 0      0   7.925
0                       3  35.0     MaLe                 1      0   8.050
1                       1   NaN  female                  0      0  53.100
```

```
#7 Detect inconsistent formatting in text columns and standardize them.
# Formate the inconsistent data
df['SEX'] = df['SEX'].str.strip()
print(df.head()) # print
```

```
          Passenger_Class   age     SEX  Siblings_Spouses  parch    fARE
survived
0                       3  12.0    male                 1      0   7.250
1                       1  38.0     NaN                 3      1     NaN
1                       2  26.0  female                 0      0   7.925
0                       3  35.0    male                 1      0   8.050
1                       1   NaN  female                 0      0  53.100
```

```
#8 Convert all column names to lowercase
# convert column name to lowercase
df.columns = df.columns.str.lower()
print(df.head()) # print
```

```
          passenger_class   age     sex  siblings_spouses  parch    fare
survived
0                       3  12.0    male                 1      0   7.250
1                       1  38.0     NaN                 3      1     NaN
1                       2  26.0  female                 0      0   7.925
0                       3  35.0    male                 1      0   8.050
1                       1   NaN  female                 0      0  53.100
```

```
#9 Identify missing values using .isnull().sum()
# Finding null value and compute sum
print(df.isnull().sum())
```

```
passenger_class     0
age                 1
sex                 1
```

```
siblings_spouses    0
parch               0
fare                1
dtype: int64
```

```
#10 Drop rows with any missing values.
# dropna() drops the missing value
df.dropna(inplace=True)
print(df.head()) # print
```

```
          passenger_class   age     sex  siblings_spouses  parch     fare
survived
0                       3  12.0    male                 1      0   7.2500
1                       2  26.0  female                 0      0   7.9250
0                       3  35.0    male                 1      0   8.0500
0                       2  54.0    male                 0      0  51.8625
1                       3   2.0    male                 3      1  21.0750
```

```
#11 Drop rows where a specific column has missing data
#dropna() drops columun with missing data
df.dropna(subset=['fARE'], inplace=True)
print(df.head()) # print
```

```
   survived  pclass      SEX   age  sibsp  parch     fARE embarked
0         0       3     male  12.0      1      0   7.2500        S
2         1       2   female  26.0      0      0   7.9250        S
3         0       3     MaLe  35.0      1      0   8.0500        S
4         1       1  female    NaN      0      0  53.1000        S
5         0       2     MALE  54.0      0      0  51.8625        S
```

```
#12 Fill missing numeric values with column mean.
# fill missing values with the mean value
df['age'].fillna(df['age'].mean(), inplace=True)
print(df.head()) # print
```

```
   survived  pclass      SEX   age  sibsp  parch     fARE embarked
0         0       3     male  12.0      1      0   7.2500        S
2         1       2   female  26.0      0      0   7.9250        S
3         0       3     MaLe  35.0      1      0   8.0500        S
4         1       1  female   26.0      0      0  53.1000        S
5         0       2     MALE  54.0      0      0  51.8625        S
/tmp/ipython-input-1128134902.py:2: FutureWarning: A value is trying to be set on a copy of
The behavior will change in pandas 3.0. This inplace method will never work because the int

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


  df['age'].fillna(df['age'].mean(), inplace=True)
```

```
#13 Fill missing categorical values with the mode.
 # fill missing value with mode value
df['embarked'].fillna(df['embarked'].mode()[0], inplace=True)
print(df.head()) # print
```

```
   survived  pclass      SEX   age  sibsp  parch    fARE embarked
0         0       3     male  12.0      1      0   7.250        S
1         1       1      NaN  38.0      3      1     NaN        C
2         1       2   female  26.0      0      0   7.925        S
3         0       3     MaLe  35.0      1      0   8.050        S
4         1       1  female    NaN      0      0  53.100        S
/tmp/ipython-input-2486643981.py:2: FutureWarning: A value is trying to be set on a copy of
The behavior will change in pandas 3.0. This inplace method will never work because the int

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


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

```
#14 Use forward fill and backward fill techniques.
# replace null with next value
df['age'].fillna(method='ffill')
# replace null with prev value
df['age'].fillna(method='bfill')
print(df.head()) # print
```

```
   survived  pclass     SEX   age  sibsp  parch    fARE embarked
0         0       3    male  12.0      1      0   7.250        S
1         1       1     NaN  38.0      3      1     NaN        C
2         1       2  female  26.0      0      0   7.925        S
3         0       3    MaLe  35.0      1      0   8.050        S
4         1       1  female  35.0      0      0  53.100        S
/tmp/ipython-input-4233193395.py:2: FutureWarning: Series.fillna with 'method' is deprecate
  df['age'].fillna(method='ffill')
/tmp/ipython-input-4233193395.py:3: FutureWarning: Series.fillna with 'method' is deprecate
  df['age'].fillna(method='bfill')
```

```
#15 Use interpolation to fill numeric gaps in time-series data
# Fill null values using interpolation
df['fARE'].interpolate(method='linear', direction='forward')
print(df.head()) # print
```

```
   survived  pclass     SEX   age  sibsp  parch     fARE embarked
0         0       3    male  12.0      1      0   7.2500        S
1         1       1     NaN  38.0      3      1   7.5875        C
2         1       2  female  26.0      0      0   7.9250        S
3         0       3    MaLe  35.0      1      0   8.0500        S
4         1       1  female  35.0      0      0  53.1000        S
```

```
#16 Create a new column as the sum of two existing columns.
 # adds values of 2 columns
df['new_column'] = df['sibsp'] + df['parch']
print(df.head()) # print
```

```
   survived  pclass     SEX   age  sibsp  parch    fARE embarked  new_column
0         0       3    male  12.0      1      0   7.250        S           1
1         1       1     NaN  38.0      3      1     NaN        C           4
2         1       2  female  26.0      0      0   7.925        S           0
3         0       3    MaLe  35.0      1      0   8.050        S           1
4         1       1  female   NaN      0      0  53.100        S           0
```

```
#17 Create a binary column based on a condition.
# checks for age > 18
df['is_adult'] = df['age'] >= 18
print(df.head()) # print
```

```
   survived  pclass     SEX   age  sibsp  parch    fARE embarked  new_column  \
0         0       3    male  12.0      1      0   7.250        S           1
1         1       1     NaN  38.0      3      1     NaN        C           4
2         1       2  female  26.0      0      0   7.925        S           0
3         0       3    MaLe  35.0      1      0   8.050        S           1
4         1       1  female   NaN      0      0  53.100        S           0

   is_adult
0     False
1      True
2      True
3      True
4     False
```

```
#18 Extract year/month/day from a datetime column.
# range() is used to generate date
df['date'] = pd.date_range('2022-01-01', periods=len(df), freq='D')
# prints year
df['year'] = df['date'].dt.year
```

```
# prints month
df['month'] = df['date'].dt.month
# prints date
df['day'] = df['date'].dt.day
# print dataset
print(df.head())
```

```
   age   fare embarked  survived       date  year  month  day
0   22   7.25        S         0 2022-01-01  2022      1    1
1   38  71.83        C         1 2022-01-02  2022      1    2
2   26   8.05        S         1 2022-01-03  2022      1    3
3   35  53.10        S         1 2022-01-04  2022      1    4
4   28   8.46        S         0 2022-01-05  2022      1    5
```

```
#19 Create a column showing length of string in a text field
# prints length of the data present in sex
df['sex_length'] = df['SEX'].str.len()
print(df.head()) # print
```

```
   survived  pclass     SEX   age  sibsp  parch    fARE embarked  sex_length
0         0       3    male  12.0      1      0   7.250        S         5.0
1         1       1     NaN  38.0      3      1     NaN        C         NaN
2         1       2  female  26.0      0      0   7.925        S         6.0
3         0       3    MaLe  35.0      1      0   8.050        S         4.0
4         1       1  female    NaN      0      0  53.100        S         7.0
```

```
#20 Bin a continuous variable into fixed intervals
# bin age value into discrete intervals
bins = [0, 18, 65, float('inf')]
# add labels based on values
labels = ['minor', 'adult', 'senior']
df['age_group'] = pd.cut(df['age'], bins=bins, labels=labels)
print(df.head()) # print
```

```
   survived  pclass     SEX   age  sibsp  parch    fARE embarked  sex_length  \
0         0       3    male  12.0      1      0   7.250        S         5.0
1         1       1     NaN  38.0      3      1     NaN        C         NaN
2         1       2  female  26.0      0      0   7.925        S         6.0
3         0       3    MaLe  35.0      1      0   8.050        S         4.0
4         1       1  female    NaN      0      0  53.100        S         7.0

  age_group
0     minor
1     adult
2     adult
3     adult
4       NaN
```

```
#21 Create a new column showing whether a value is above or below median
# checks for fare whether above or below median
df['fare_above_median'] = df['fARE'] > df['fARE'].median()
print(df.head()) # print
```

```
   survived  pclass     SEX   age  sibsp  parch    fARE embarked  sex_length  \
0         0       3    male  12.0      1      0   7.250        S         5.0
1         1       1     NaN  38.0      3      1     NaN        C         NaN
2         1       2  female  26.0      0      0   7.925        S         6.0
3         0       3    MaLe  35.0      1      0   8.050        S         4.0
4         1       1  female    NaN      0      0  53.100        S         7.0

  age_group  fare_above_median
0     minor              False
1     adult              False
2     adult              False
3     adult              False
4       NaN               True
```

```
#22 Encode a categorical variable using one-hot encoding
```

```
# converts categorial data into a numerical format
df = pd.get_dummies(df, columns=['embarked'])
print(df.head()) # print
```

```
   survived  pclass     SEX   age  sibsp  parch    fARE  sex_length  \
0         0       3    male  12.0      1      0   7.250         5.0
1         1       1     NaN  38.0      3      1     NaN         NaN
2         1       2  female  26.0      0      0   7.925         6.0
3         0       3    MaLe  35.0      1      0   8.050         4.0
4         1       1  female   NaN      0      0  53.100         7.0

  age_group  fare_above_median  embarked_C  embarked_S
0     minor              False       False        True
1     adult              False        True       False
2     adult              False       False        True
3     adult              False       False        True
4       NaN               True       False        True
```

```
# Use the following for problem # 23
# Sample dataset
data = {
'age': [22, 38, 26, 35, 28, 54, 2, 19, 40, 30],
'fare': [7.25, 71.83, 8.05, 53.10, 8.46, 51.86, 21.07, 30.0, 27.75, 13.0],
'embarked': ['S', 'C', 'S', 'S', 'S', 'Q', 'S', 'C', 'Q', 'S'],
'survived': [0, 1, 1, 1, 0, 0, 1, 0, 1, 0] # target variable
}
df = pd.DataFrame(data)
# Features and target
x = df[['age', 'fare', 'embarked']]
y = df['survived']
print(df.head())
```

```
   age   fare embarked  survived
0   22   7.25        S         0
1   38  71.83        C         1
2   26   8.05        S         1
3   35  53.10        S         1
4   28   8.46        S         0
```

```
#23 Split data into train and test data
# Importing the train_test_split function from sklearn for splitting the dataset
from sklearn.model_selection import train_test_split
# Splitting the dataset into training (80%) and testing (20%) sets
# 'x' is the feature data, and 'y' is the target (labels) data
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)
# Display the first few rows of the training feature data
print(x_train.head())
# Display the first few rows of the testing feature data
print(x_test.head())
# Display the first few rows of the training target labels
print(y_train.head())
# Display the first few rows of the testing target labels
print(y_test.head())
```

```
   age   fare embarked
5   54  51.86        Q
0   22   7.25        S
7   19  30.00        C
2   26   8.05        S
9   30  13.00        S
   age   fare embarked
8   40  27.75        Q
1   38  71.83        C
5    0
0    0
7    0
2    1
9    0
```

```
Name: survived, dtype: int64
8    1
1    1
Name: survived, dtype: int64
```