9. Practice Questions

Use the titanic_df DataFrame we've cleaned and modified to answer the following questions.

Question 1: Create a new DataFrame containing only the passengers who were part of a family (FamilySize > 1). How many such passengers are there?

```
new_df= titanic_df[titanic_df["FamilySize"]>1]
```

Question 2: Find the average fare paid by passengers for each Embarked location.

```
mean=titanic_df.groupby("Embarked Location")["Fare"].mean()
```

Question 3: Create a new feature called AgeGroup that categorizes passengers into 'Child' (age < 18), 'Adult' (18 <= age <= 60), and 'Senior' (age > 60).

```
def age_group(age):
    if age < 18:
        return 'Child'
    elif age <= 60:
        return 'Adult'
    else:
        return 'Senior'

titanic_df['AgeGroup'] = titanic_df['Age'].apply(age_group)
```

Question 4: What was the survival rate for passengers who were alone (IsAlone == 1) versus those who were not?

```
survival_rate = titanic_df.groupby('IsAlone')['Survived'].mean()
```

Question 5: Drop the Name, Ticket, and PassengerId columns as they are not useful for a simple ML model. Display the head of the final, cleaned DataFrame.

```
cleaned_df = titanic_df.drop(['Name', 'Ticket', 'PassengerId'], axis=1)
```

Question 1 (MultiIndex & Grouping): Set a MultiIndex on the DataFrame using 'Pclass', 'Sex', and 'Embarked'. Then, calculate the median Age and Fare for each of these nested groups. Which group had the highest median Fare?

```
df_sol = pd.read_csv('titanic.csv')
multi_df_sol = df_sol.set_index(['Pclass', 'Sex', 'Embarked'])
median_stats = multi_df_sol.groupby(level=['Pclass', 'Sex', 'Embarked'])[['Age', 'Fare']].median()
print("Median Age and Fare for each group:")
print(median_stats)
print("\nGroup with highest median fare:")
print(median_stats['Fare'].idxmax())
print("\n" + "="*50 + "\n")
```

Question 2 (Categorical & .str): a. Create a Title feature from the Name column. b. Some titles like 'Mlle', 'Ms' are synonyms for 'Miss', and 'Mme' is a synonym for 'Mrs'. Use the .str.replace() method or .map() to consolidate them. c. Convert the cleaned Title colu

```
df_sol['Title'] = df_sol['Name'].str.extract(' ([A-Za-z]+)\.', expand=False)
title_map = {'Mlle': 'Miss', 'Ms': 'Miss', 'Mme': 'Mrs'}
df_sol['Title'] = df_sol['Title'].replace(title_map)
df_sol['Title'] = df_sol['Title'].astype('category')
print("Cleaned Title value counts and dtype:")
print(df_sol['Title'].value_counts())
print(df_sol['Title'].dtype)
print("\n" + "="*50 + "\n")
```

Question 3 (.pipe() & Datetime): Imagine the Titanic sailing date was 1912-04-10. Create a dummy column 'PurchaseDate' that is a random number of days (between 1 and 100) before this sailing date. Write a function that takes this DataFrame, converts 'Purcha
difference in days) and PurchaseWeekday (the name of the day of the week). Use .pipe() to apply this function.

```
def create_purchase_date_features(df):
    df_copy = df.copy()
    sailing_date = pd.to_datetime('1912-04-10')
    random_days = np.random.randint(1, 101, size=len(df_copy))
    df_copy['PurchaseDate'] = sailing_date - pd.to_timedelta(random_days, unit='d')
    df_copy['DaysBeforeSailing'] = (sailing_date - df_copy['PurchaseDate']).dt.days
    df_copy['PurchaseWeekday'] = df_copy['PurchaseDate'].dt.day_name()
    return df_copy

date_featured_df = df_sol.pipe(create_purchase_date_features)
print("DataFrame with purchase date features:")
print(date_featured_df[['PurchaseDate', 'DaysBeforeSailing', 'PurchaseWeekday']].head())
print("\n" + "="*50 + "\n")
```

Question 4 (Window Functions): Let's pretend the Fare column represents daily stock prices for a fictional "White Star Line" stock, ordered by PassengerId. Calculate the 10-period rolling average Fare and the 10-period rolling standard deviation of the Fare. Wha

```
df_sol_sorted = df_sol.sort_values(by='PassengerId').reset_index(drop=True)
df_sol_sorted['rolling_fare_mean'] = df_sol_sorted['Fare'].rolling(window=10).mean()
df_sol_sorted['rolling_fare_std'] = df_sol_sorted['Fare'].rolling(window=10).std()
print("Rolling Fare calculations for PassengerId=10 (index 9):")
print(df_sol_sorted.loc[9, ['Fare', 'rolling_fare_mean', 'rolling_fare_std']])
```