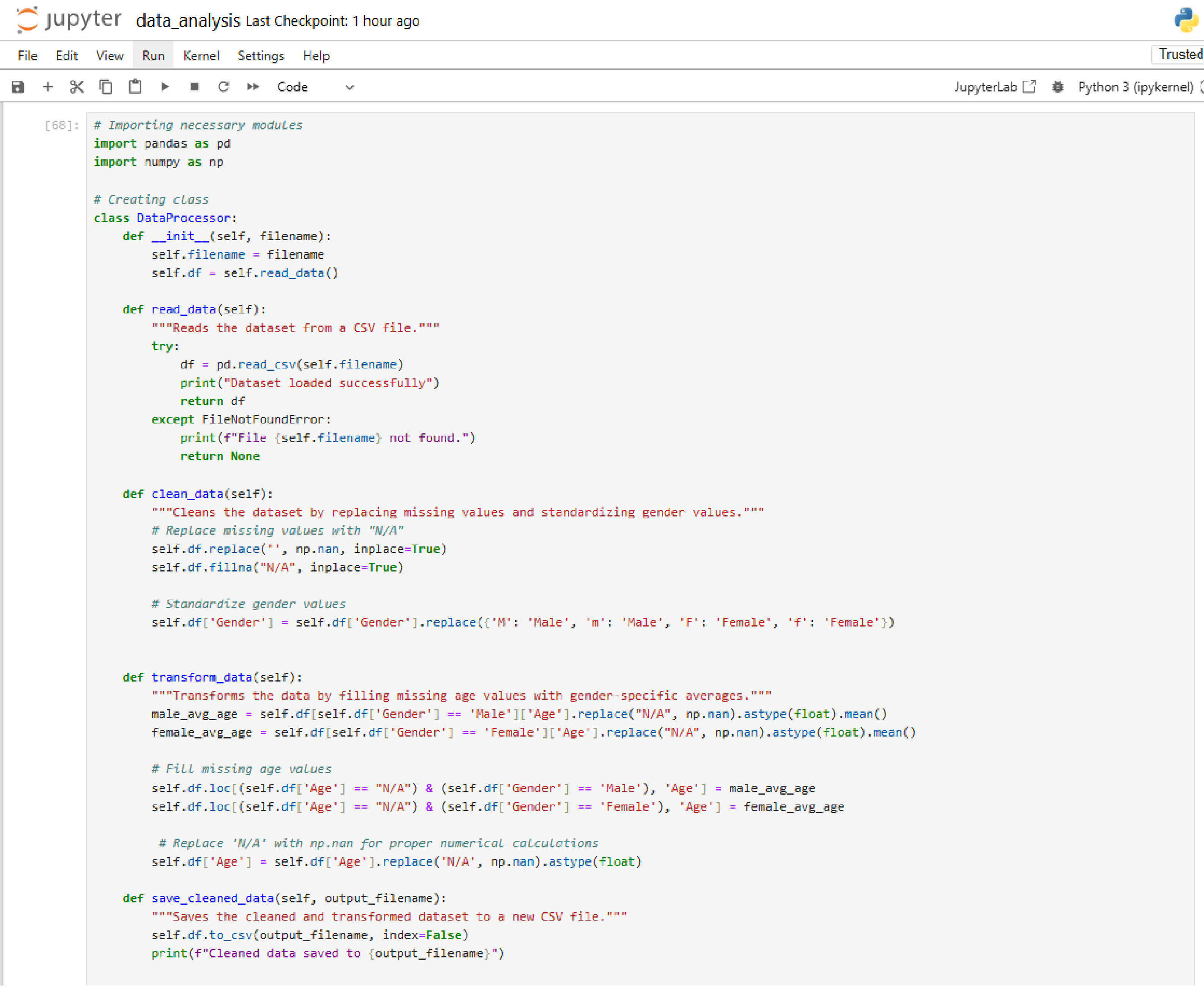
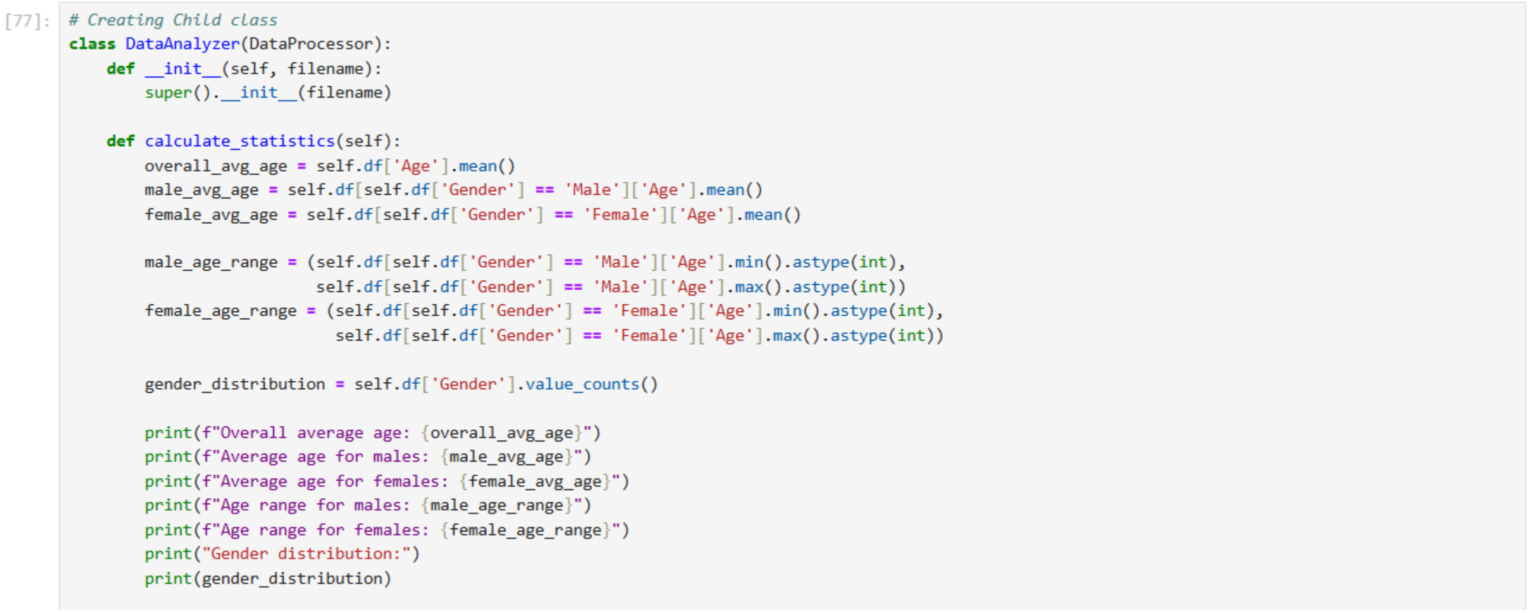
Name: Mugilmithran Kathiravan
Std ID: C0934419
Assignment II

1. Creating parent class with methods.

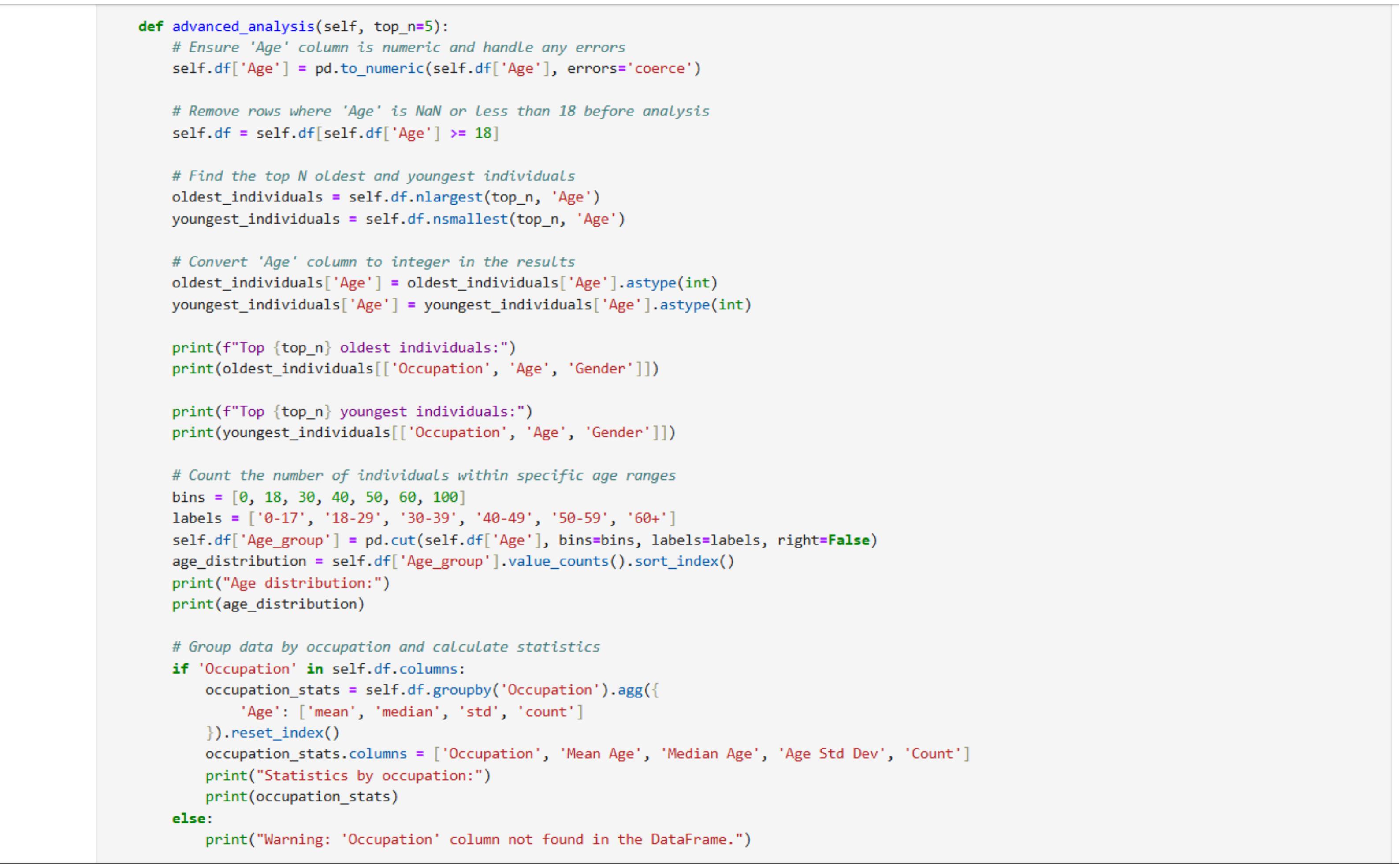


```
# Importing necessary modules
import pandas as pd
import numpy as np

# Creating class
class DataProcessor:
    def __init__(self, filename):
        self.filename = filename
        self.df = self.read_data()

    def read_data(self):
        """Reads the dataset from a CSV file."""
        try:
            df = pd.read_csv(self.filename)
            print("Dataset loaded successfully")
            return df
        except FileNotFoundError:
            print(f"File {self.filename} not found.")
            return None

    def clean_data(self):
        """Cleans the dataset by replacing missing values and standardizing gender values."""
        # Replace missing values with "N/A"
        self.df.replace('', np.nan, inplace=True)
        self.df.fillna("N/A", inplace=True)

        # Standardize gender values
        self.df['Gender'] = self.df['Gender'].replace({'M': 'Male', 'm': 'Male', 'F': 'Female', 'f': 'Female'})


    def transform_data(self):
        """Transforms the data by filling missing age values with gender-specific averages."""
        male_avg_age = self.df[self.df['Gender'] == 'Male']['Age'].replace("N/A", np.nan).astype(float).mean()
        female_avg_age = self.df[self.df['Gender'] == 'Female']['Age'].replace("N/A", np.nan).astype(float).mean()

        # Fill missing age values
        self.df.loc[(self.df['Age'] == "N/A") & (self.df['Gender'] == 'Male'), 'Age'] = male_avg_age
        self.df.loc[(self.df['Age'] == "N/A") & (self.df['Gender'] == 'Female'), 'Age'] = female_avg_age

         # Replace 'N/A' with np.nan for proper numerical calculations
        self.df['Age'] = self.df['Age'].replace('N/A', np.nan).astype(float)

    def save_cleaned_data(self, output_filename):
        """Saves the cleaned and transformed dataset to a new CSV file."""
        self.df.to_csv(output_filename, index=False)
        print(f"Cleaned data saved to {output_filename}")
```

2. Creating child class with parent class inheritance and it's own methods.

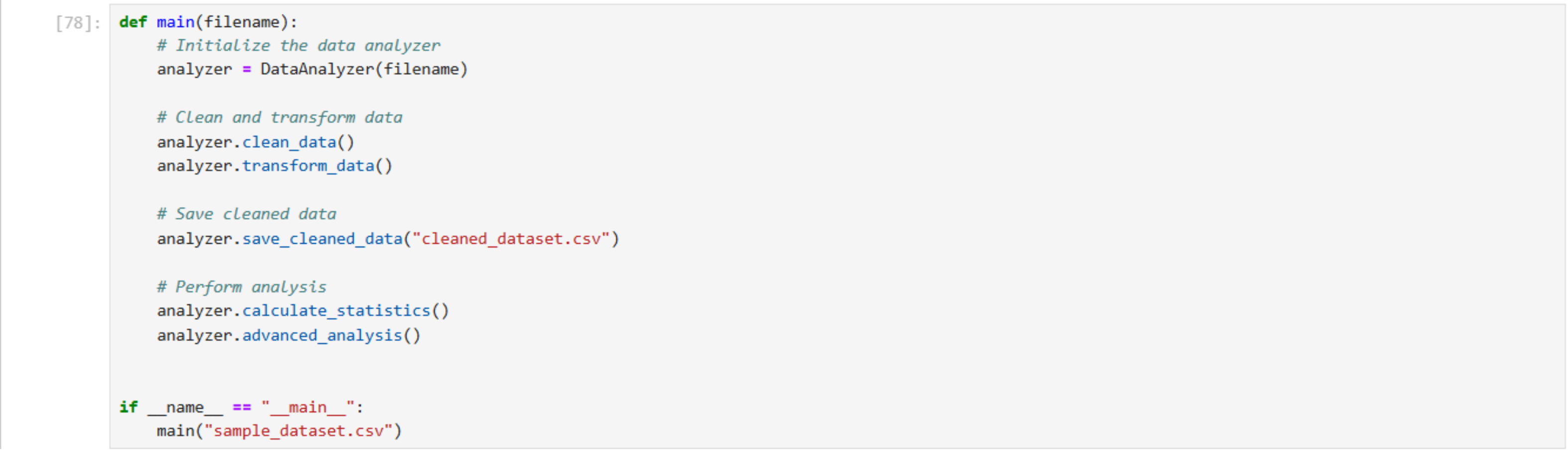


```
[77]: # Creating Child class
      class DataAnalyzer(DataProcessor):
          def __init__(self, filename):
              super().__init__(filename)

          def calculate_statistics(self):
              overall_avg_age = self.df['Age'].mean()
              male_avg_age = self.df[self.df['Gender'] == 'Male']['Age'].mean()
              female_avg_age = self.df[self.df['Gender'] == 'Female']['Age'].mean()

              male_age_range = (self.df[self.df['Gender'] == 'Male']['Age'].min().astype(int),
                                self.df[self.df['Gender'] == 'Male']['Age'].max().astype(int))
              female_age_range = (self.df[self.df['Gender'] == 'Female']['Age'].min().astype(int),
                                  self.df[self.df['Gender'] == 'Female']['Age'].max().astype(int))

              gender_distribution = self.df['Gender'].value_counts()

              print(f"Overall average age: {overall_avg_age}")
              print(f"Average age for males: {male_avg_age}")
              print(f"Average age for females: {female_avg_age}")
              print(f"Age range for males: {male_age_range}")
              print(f"Age range for females: {female_age_range}")
              print("Gender distribution:")
              print(gender_distribution)
```



```
def advanced_analysis(self, top_n=5):
    # Ensure 'Age' column is numeric and handle any errors
    self.df['Age'] = pd.to_numeric(self.df['Age'], errors='coerce')

    # Remove rows where 'Age' is NaN or less than 18 before analysis
    self.df = self.df[self.df['Age'] >= 18]

    # Find the top N oldest and youngest individuals
    oldest_individuals = self.df.nlargest(top_n, 'Age')
    youngest_individuals = self.df.nsmallest(top_n, 'Age')

    # Convert 'Age' column to integer in the results
    oldest_individuals['Age'] = oldest_individuals['Age'].astype(int)
    youngest_individuals['Age'] = youngest_individuals['Age'].astype(int)

    print(f"Top {top_n} oldest individuals:")
    print(oldest_individuals[['Occupation', 'Age', 'Gender']])

    print(f"Top {top_n} youngest individuals:")
    print(youngest_individuals[['Occupation', 'Age', 'Gender']])

    # Count the number of individuals within specific age ranges
    bins = [0, 18, 30, 40, 50, 60, 100]
    labels = ['0-17', '18-29', '30-39', '40-49', '50-59', '60+']
    self.df['Age_group'] = pd.cut(self.df['Age'], bins=bins, labels=labels, right=False)
    age_distribution = self.df['Age_group'].value_counts().sort_index()
    print("Age distribution:")
    print(age_distribution)

    # Group data by occupation and calculate statistics
    if 'Occupation' in self.df.columns:
        occupation_stats = self.df.groupby('Occupation').agg({
            'Age': ['mean', 'median', 'std', 'count']
        }).reset_index()
        occupation_stats.columns = ['Occupation', 'Mean Age', 'Median Age', 'Age Std Dev', 'Count']
        print("Statistics by occupation:")
        print(occupation_stats)
    else:
        print("Warning: 'Occupation' column not found in the DataFrame.")
```

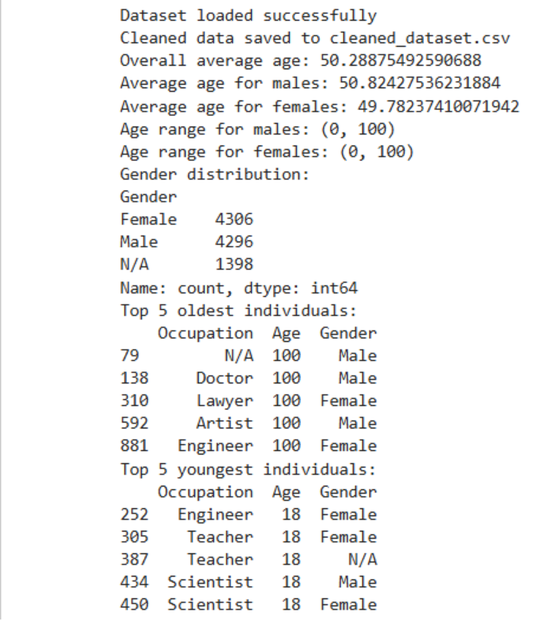
3. Code execution



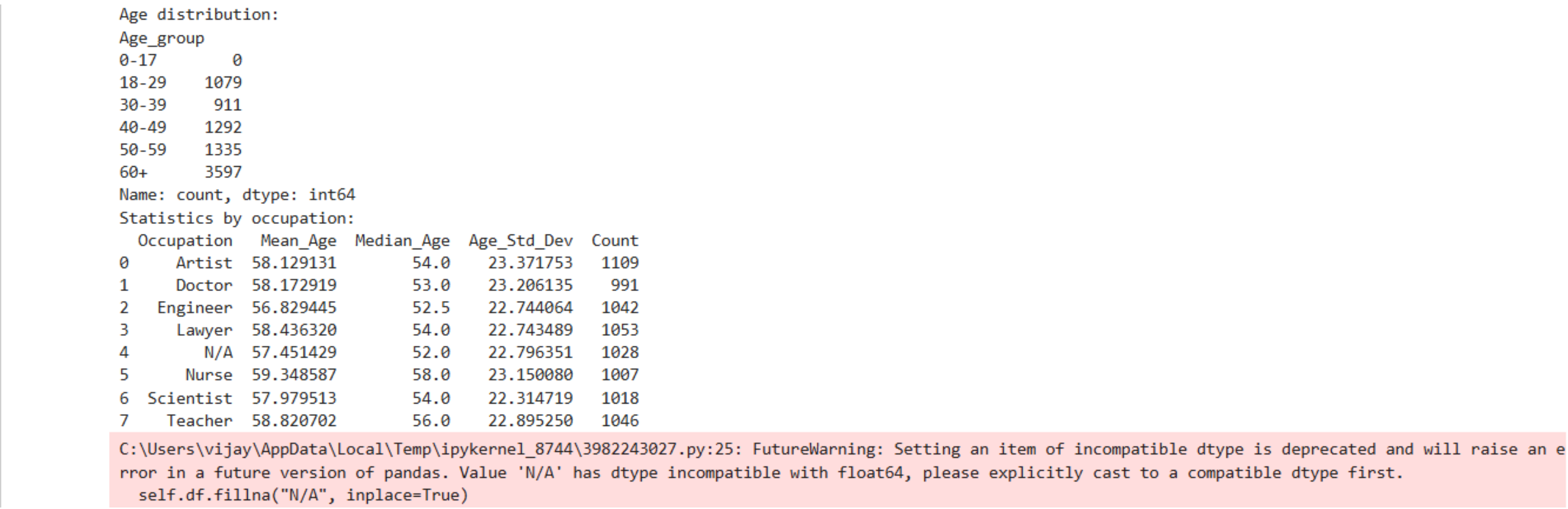
[78]:

```
def main(filename):
    # Initialize the data analyzer
    analyzer = DataAnalyzer(filename)

    # Clean and transform data
    analyzer.clean_data()
    analyzer.transform_data()

    # Save cleaned data
    analyzer.save_cleaned_data("cleaned_dataset.csv")

    # Perform analysis
    analyzer.calculate_statistics()
    analyzer.advanced_analysis()


if __name__ == "__main__":
    main("sample_dataset.csv")
```

4. Output



```
Dataset loaded successfully
Cleaned data saved to cleaned_dataset.csv
Overall average age: 50.28875492590688
Average age for males: 50.82427536231884
Average age for females: 49.78237410071942
Age range for males: (0, 100)
Age range for females: (0, 100)
Gender distribution:
Gender
Female    4306
Male      4296
N/A       1398
Name: count, dtype: int64
Top 5 oldest individuals:
    Occupation  Age  Gender
79         N/A  100    Male
138     Doctor  100    Male
310     Lawyer  100  Female
592     Artist  100    Male
881   Engineer  100  Female
Top 5 youngest individuals:
    Occupation  Age  Gender
252   Engineer   18  Female
305    Teacher   18  Female
387    Teacher   18     N/A
434  Scientist   18    Male
450  Scientist   18  Female
```



```
Age distribution:
Age_group
0-17        0
18-29    1079
30-39     911
40-49    1292
50-59    1335
60+      3597
Name: count, dtype: int64
Statistics by occupation:
  Occupation   Mean_Age  Median_Age  Age_Std_Dev  Count
0     Artist  58.129131        54.0    23.371753   1109
1     Doctor  58.172919        53.0    23.206135    991
2   Engineer  56.829445        52.5    22.744064   1042
3     Lawyer  58.436320        54.0    22.743489   1053
4        N/A  57.451429        52.0    22.796351   1028
5      Nurse  59.348587        58.0    23.150080   1007
6  Scientist  57.979513        54.0    22.314719   1018
7    Teacher  58.820702        56.0    22.895250   1046
C:\Users\vijay\AppData\Local\Temp\ipykernel_8744\3982243027.py:25: FutureWarning: Setting an item of incompatible dtype is deprecated and will raise an e
rror in a future version of pandas. Value 'N/A' has dtype incompatible with float64, please explicitly cast to a compatible dtype first.
  self.df.fillna("N/A", inplace=True)
```