Double-click (or enter) to edit

```
from google.colab import files
uploaded = files.upload()
```

Choose Files StudentsPerformance.csv

- **StudentsPerformance.csv**(text/csv) - 57021 bytes, last modified: 16/4/2025 - 100% done

Step 1: Import Libraries & Upload Dataset

We'll use Google Colab's built-in uploader to upload the `StudentsPerformance.csv` file.

```
import pandas as pd

# Load CSV into a DataFrame
df = pd.read_csv('StudentsPerformance.csv')

# Show top 5 rows
df.head()
```

	gender	race/ethnicity	parental level of education	lunch	test preparation course	math score	reading score	writing score
0	female	group B	bachelor's degree	standard	none	72	72	74
1	female	group C	some college	standard	completed	69	90	88
2	female	group B	master's degree	standard	none	90	95	93
3	male	group A	associate's degree	free/reduced	none	47	57	44

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

Step 2: Load Dataset & Display Top 5 Rows

Now that the dataset is uploaded, we'll load it using `pandas.read_csv()` and take a quick look at the first few rows using `head()`.

```
# Clean column names: strip whitespace, convert to lowercase, replace spaces with underscores
df.columns = df.columns.str.strip().str.lower().str.replace(' ', '_')
df.head()
```

	gender	race/ethnicity	parental_level_of_education	lunch	test_preparation_course	math_score	reading_score	writing_sco
0	female	group B	bachelor's degree	standard	none	72	72	
1	female	group C	some college	standard	completed	69	90	
2	female	group B	master's degree	standard	none	90	95	
3	male	group A	associate's degree	free/reduced	none	47	57	
4	male	group C	some college	standard	none	76	78	

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

Step 3: Clean Column Names

We clean the column names to make them easier to work with:

- Remove leading/trailing spaces
- Convert to lowercase
- Replace spaces with underscores

```
# Dataset Shape
print("Dataset Shape:", df.shape)

# Dataset Info
print("\nDataset Info:")
```

```
df.info()

# Summary Statistics
print("\nSummary Statistics:")
display(df.describe())

# Check for missing values
print("\nMissing Values:")
print(df.isnull().sum())
```

```
Dataset Shape: (1000, 8)

Dataset Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 8 columns):
 #   Column                       Non-Null Count  Dtype
---  ------                       --------------  -----
 0   gender                       1000 non-null   object
 1   race/ethnicity               1000 non-null   object
 2   parental_level_of_education  1000 non-null   object
 3   lunch                        1000 non-null   object
 4   test_preparation_course      1000 non-null   object
 5   math_score                   1000 non-null   int64
 6   reading_score                1000 non-null   int64
 7   writing_score                1000 non-null   int64
dtypes: int64(3), object(5)
memory usage: 62.6+ KB

Summary Statistics:
```

	math_score	reading_score	writing_score
count	1000.00000	1000.000000	1000.000000
mean	66.08900	69.169000	68.054000
std	15.16308	14.600192	15.195657
min	0.00000	17.000000	10.000000
25%	57.00000	59.000000	57.750000
50%	66.00000	70.000000	69.000000
75%	77.00000	79.000000	79.000000
max	100.00000	100.000000	100.000000

```
Missing Values:
gender                         0
race/ethnicity                 0
parental_level_of_education    0
lunch                          0
test_preparation_course        0
math_score                     0
reading_score                  0
writing_score                  0
dtype: int64
```

Step 4: Dataset Overview — Shape, Info, Describe, Null Values

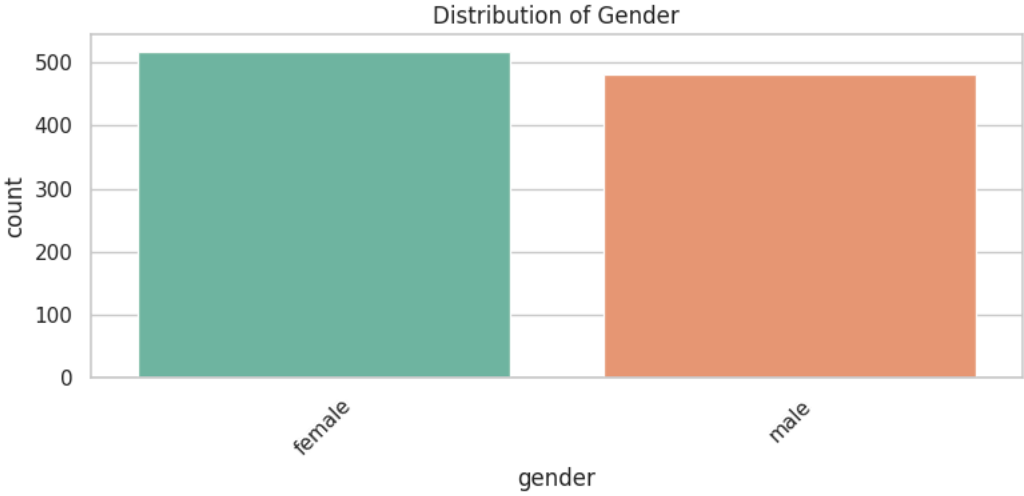
We will perform some basic exploratory checks to understand the structure of the dataset:

- **Shape:** Total number of rows and columns.
- **Info:** Data types and non-null counts.
- **Describe:** Summary statistics for numerical columns.
- **Missing Values:** Check if there are any null or missing values.

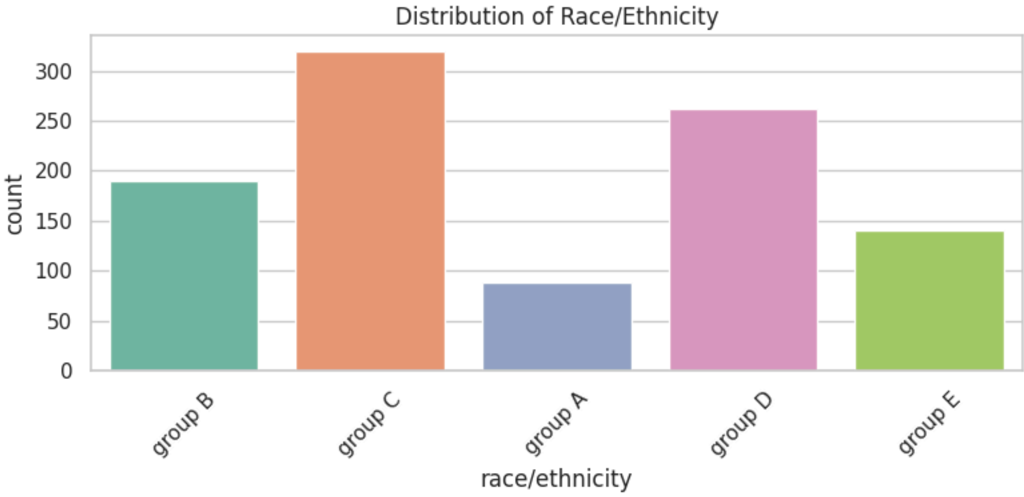
```
import seaborn as sns
import matplotlib.pyplot as plt

# Set a theme for the plots
sns.set(style="whitegrid")

# List of categorical columns
categorical_cols = ['gender', 'race/ethnicity', 'parental_level_of_education', 'lunch', 'test_preparation_course']

# Plot countplots for each categorical column
for col in categorical_cols:
    plt.figure(figsize=(8, 4))
    sns.countplot(data=df, x=col, palette='Set2')
```

```
plt.title(f'Distribution of {col.replace("_", " ").title()}')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

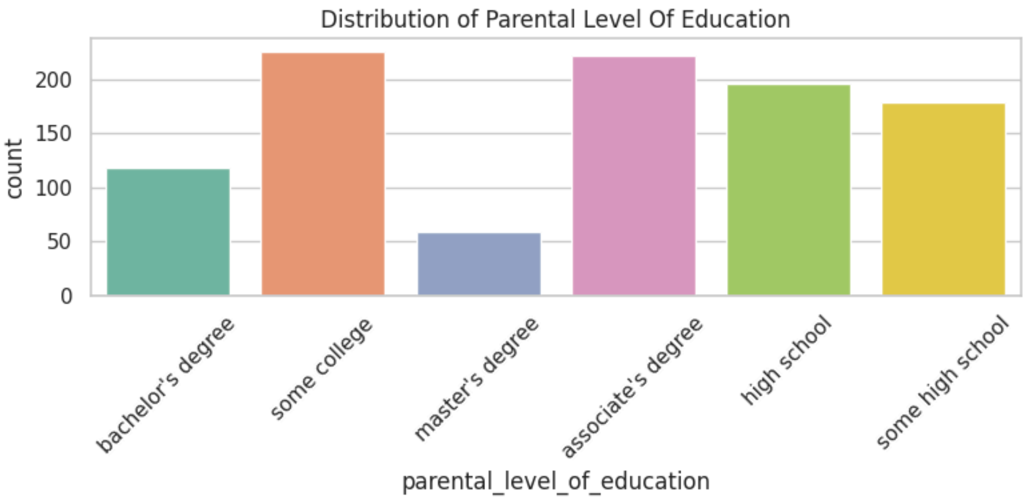
```
<ipython-input-8-1c998d9a73ce>:13: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set

  sns.countplot(data=df, x=col, palette='Set2')
```



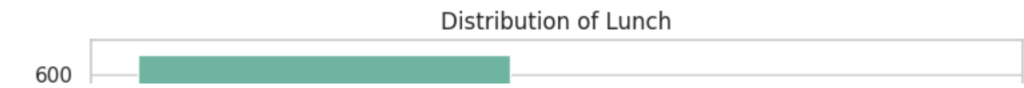

```
<ipython-input-8-1c998d9a73ce>:13: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set

  sns.countplot(data=df, x=col, palette='Set2')
```

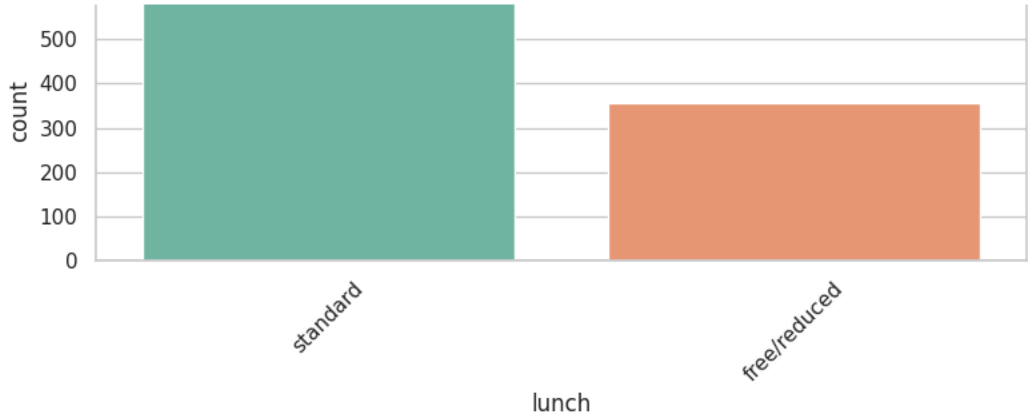



```
<ipython-input-8-1c998d9a73ce>:13: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set

  sns.countplot(data=df, x=col, palette='Set2')
```

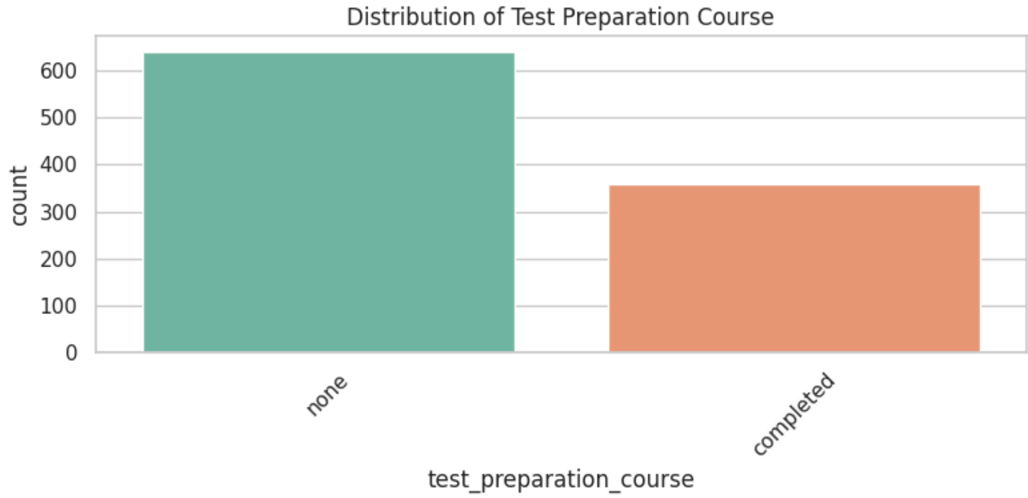



```
<ipython-input-8-1c998d9a73ce>:13: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set

  sns.countplot(data=df, x=col, palette='Set2')
```






```
<ipython-input-8-1c998d9a73ce>:13: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set

  sns.countplot(data=df, x=col, palette='Set2')
```



Step 5: Categorical Column Analysis

We'll explore the distribution of categorical features using count plots:

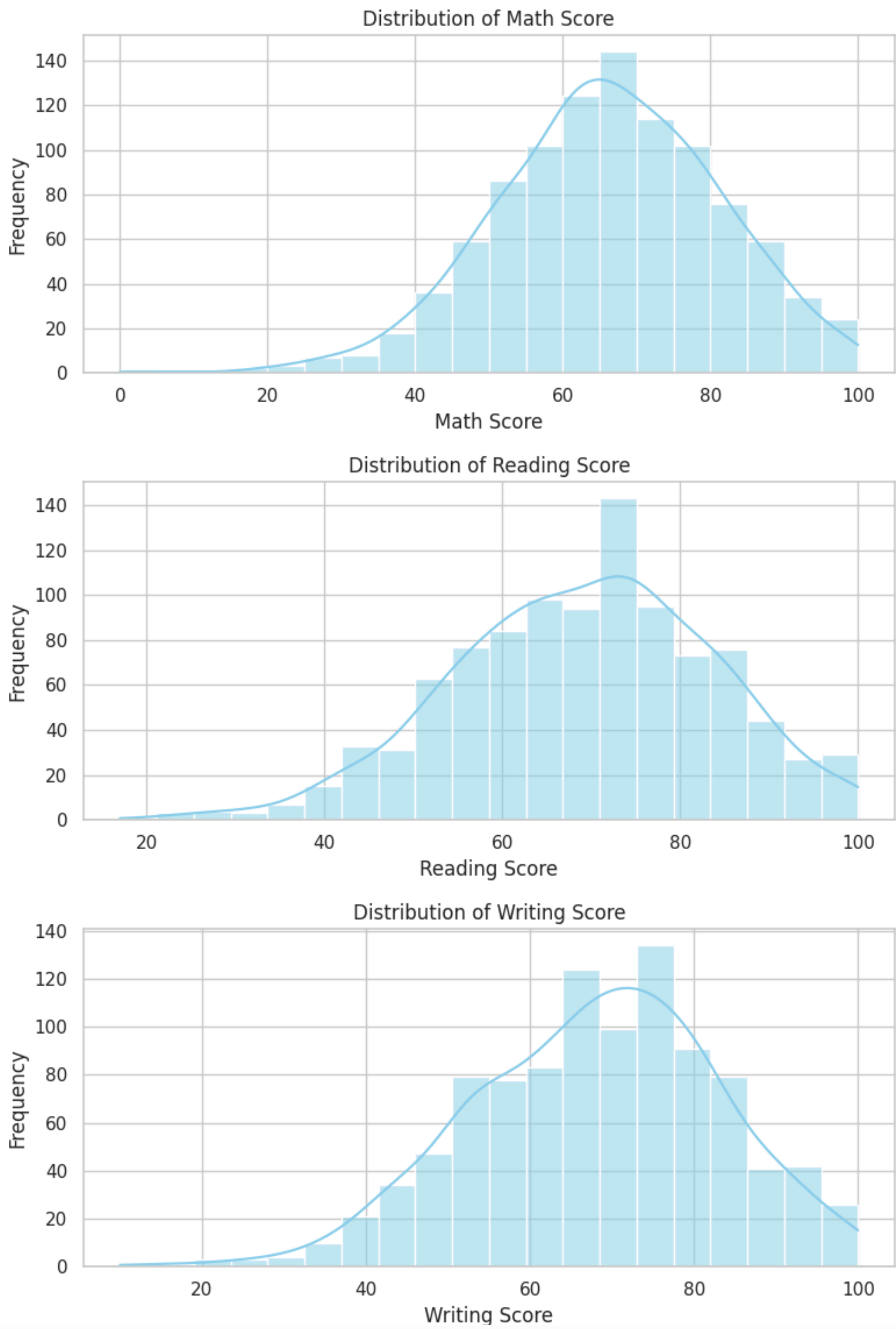
- `gender`
- `race/ethnicity`
- `parental level of education`
- `lunch`
- `test preparation course`

These plots help us understand the data balance and possible biases in the dataset.

```
# Display basic statistics for numerical columns
df.describe()[['math_score', 'reading_score', 'writing_score']]
```

	math_score	reading_score	writing_score
count	1000.00000	1000.000000	1000.000000
mean	66.08900	69.169000	68.054000
std	15.16308	14.600192	15.195657
min	0.00000	17.000000	10.000000
25%	57.00000	59.000000	57.750000
50%	66.00000	70.000000	69.000000
75%	77.00000	79.000000	79.000000
max	100.00000	100.000000	100.000000

```
# Plot distribution for each numerical score
num_cols = ['math_score', 'reading_score', 'writing_score']

for col in num_cols:
    plt.figure(figsize=(8, 4))
    sns.histplot(df[col], kde=True, color='skyblue', bins=20)
    plt.title(f'Distribution of {col.replace("_", " ").title()}')
    plt.xlabel(col.replace("_", " ").title())
    plt.ylabel("Frequency")
    plt.tight_layout()
    plt.show()
```



Step 6: Numerical Column Analysis

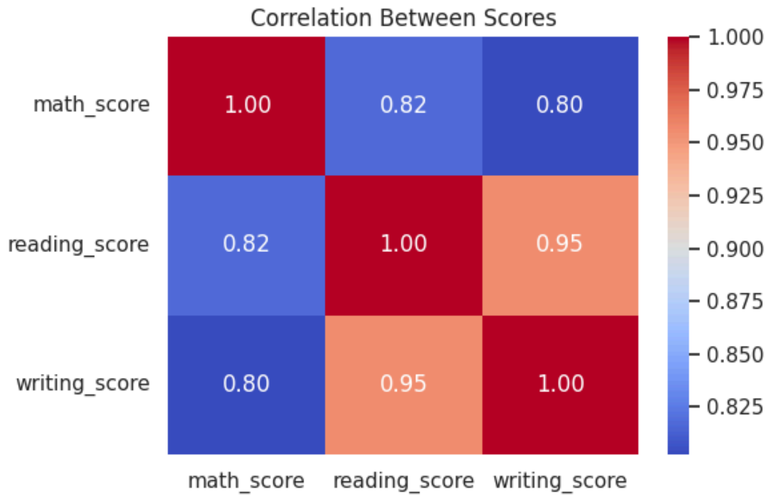
We'll now explore the distribution of the numerical scores:

- `math score`
- `reading score`
- `writing score`

First, we use `.describe()` to get summary statistics — mean, std, min, max, etc.
Then, we use histograms to visualize the spread and detect any skewness or outliers in each score.

```
# Correlation matrix for numerical columns
correlation_matrix = df[['math_score', 'reading_score', 'writing_score']].corr()

# Plot heatmap
plt.figure(figsize=(6, 4))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title("Correlation Between Scores")
```

```
plt.title("Correlation Between Scores")
plt.tight_layout()
plt.show()
```



Step 7: Correlation Analysis

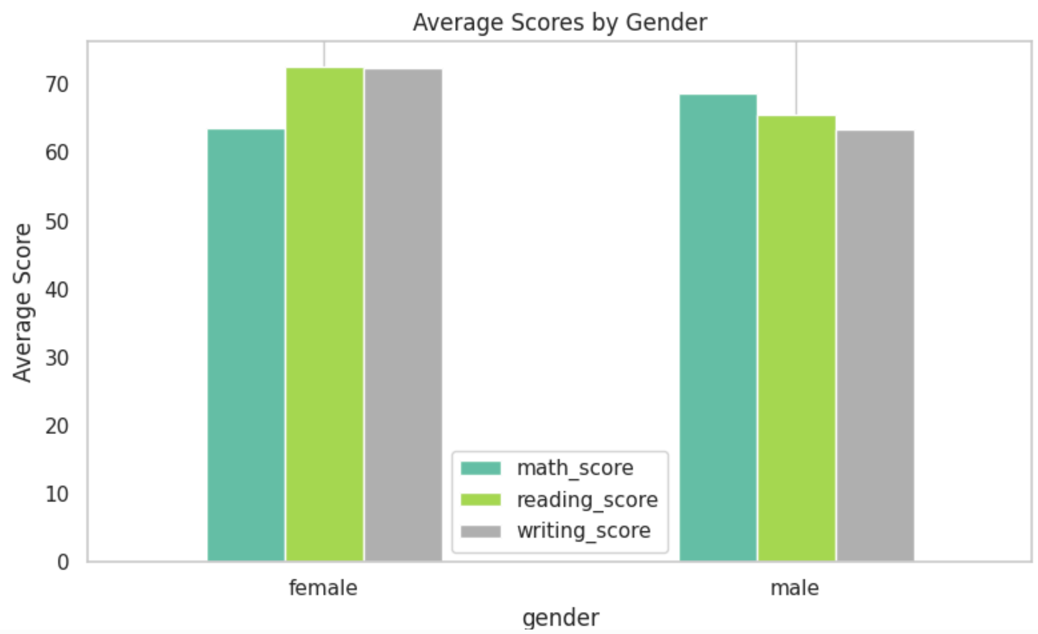
We analyze the relationship between `math score`, `reading score`, and `writing score` using a **correlation heatmap**.

The values range from:

- `+1` : Strong positive correlation
- `0` : No correlation
- `-1` : Strong negative correlation

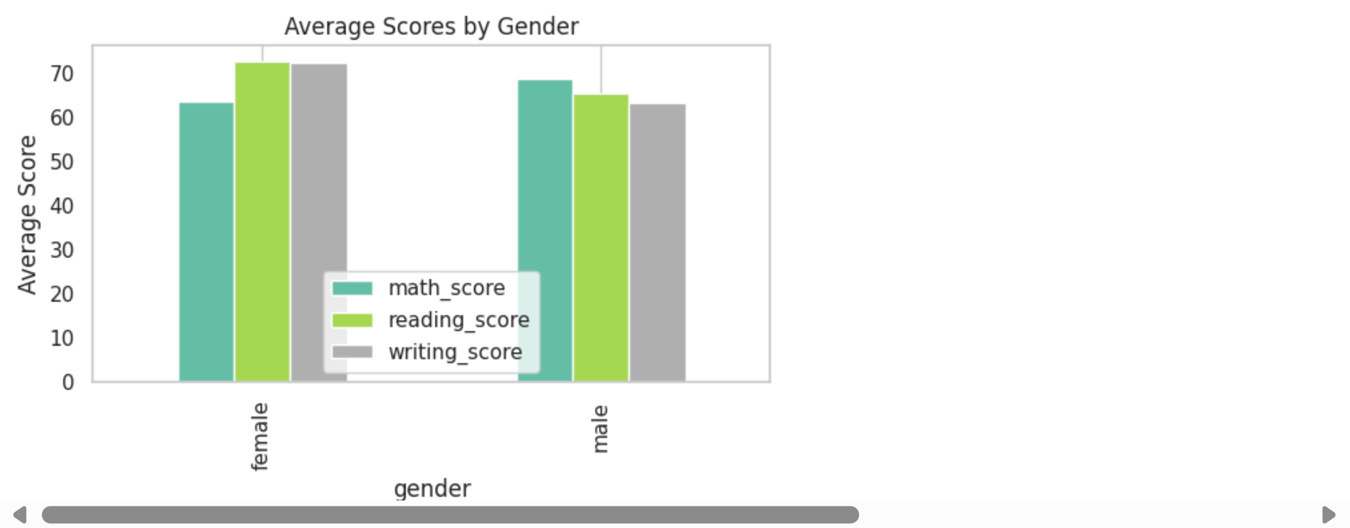
This step helps us understand if students who score high in one subject also perform similarly in others.

```
# Group by gender and calculate average scores
gender_group = df.groupby('gender')[['math_score', 'reading_score', 'writing_score']].mean().reset_index()

# Plot
gender_group.plot(x='gender', kind='bar', figsize=(8,5), colormap='Set2')
plt.title("Average Scores by Gender")
plt.ylabel("Average Score")
plt.xticks(rotation=0)
plt.grid(axis='y')
plt.tight_layout()
plt.show()
```




```
# Step 8A: Gender vs Average Scores
```

```
# Group by gender and calculate mean scores
gender_group = df.groupby('gender')[['math_score', 'reading_score', 'writing_score']].mean().reset_index()

# Plot
gender_group.plot(x='gender', kind='bar', figsize=(6,4), colormap='Set2')
plt.title("Average Scores by Gender")
plt.ylabel("Average Score")
plt.grid(axis='y')
plt.tight_layout()
plt.show()
```



Step 8B: Race/Ethnicity vs Average Scores

We analyze how student performance varies based on race/ethnicity. This can highlight trends or disparities in academic performance among different racial/ethnic groups.

```
# Group by race/ethnicity and calculate mean scores
race_group = df.groupby('race/ethnicity')[['math_score', 'reading_score', 'writing_score']].mean().reset_index()

# Plot
race_group.plot(x='race/ethnicity', kind='bar', figsize=(8,5), colormap='Set1')
plt.title("Average Scores by Race/Ethnicity")
plt.ylabel("Average Score")
plt.xticks(rotation=0)
plt.grid(axis='y')
plt.tight_layout()
plt.show()
```