In [ ]:
```
# Add the Pandas dependency

import pandas as pd
```

In [ ]:
```
# Files to load

school_data_to_load = "Resources 4/schools_complete.csv"
student_data_to_load = "Resources 4/students_complete.csv"
```

In [2]:
```
# Read the school data file and store it in a Pandas DataFrame.

school_data_df = pd.read_csv(school_data_to_load)
school_data_df
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Input In [2], in <cell line: 3>()
      1 # Read the school data file and store it in a Pandas DataFrame.
----> 3 school_data_df = read_csv(school_data_to_load)
      4 school_data_df

NameError: name 'read_csv' is not defined
```

In [ ]:
```
# Determine if there are any missing values in the student data.
student_data_df.count()

# Determine if there are any missing values in the school data.
school_data_df.isnull()

# Determine if there are any missing values in the student data.
student_data_df.isnull()

# Determine if there are not any missing values in the school data.
school_data_df.notnull()
```

In [ ]:
```
# Determine if there are not any missing values in the school data.
school_data_df.notnull()
```

In [ ]:
```
# Files to load
file_to_load = "Resources/missing_grades.csv"

# Read the CSV into a DataFrame
missing_grade_df = pd.read_csv(file_to_load)
missing_grade_df

# Drop the NaNs.
missing_grade_df.dropna()

# Fill in the empty rows with "85".
missing_grade_df.fillna(85)
```

In [ ]:
```
# Determine data types for the school DataFrame.
school_data_df.dtypes

# Determine data types for the student DataFrame.
student_data_df.dtypes
```

```
# Put the student names in a list.
student_names = student_data_df["student_name"].tolist()
student_names
```

In [ ]:
```
# Split the student name and determine the length of the split name.
for name in student_names:
    print(name.split(), len(name.split()))
```

In [ ]:
```
# Create a new list and use it for the for loop to iterate through the list.
students_to_fix = []

# Use an if statement to check the length of the name.
# If the name is greater than or equal to "3", add the name to the list.

for name in student_names:
    if len(name.split()) >= 3:
        students_to_fix.append(name)

# Get the length of the students whose names are greater than or equal to "3".
len(students_to_fix)
```

In [ ]:
```
# Add the prefixes less than or equal to 4 to a new list.
prefixes = []
for name in students_to_fix:
    if len(name.split()[0]) <= 4:
        prefixes.append(name.split()[0])

print(prefixes)


# Add the suffixes less than or equal to 3 to a new list.
suffixes = []
for name in students_to_fix:
    if len(name.split()[-1]) <= 3:
        suffixes.append(name.split()[-1])

print(suffixes)
```

In [ ]:
```
# Get the unique items in the "prefixes" list.
set(prefixes)

# Get the unique items in the "suffixes" list.
set(suffixes)
```

In [ ]:
```
# Strip "Mrs." from the student names
for name in students_to_fix:
    print(name.strip("Mrs."))

# Replace "Dr." with an empty string.
name = "Dr. Linda Santiago"
name.replace("Dr.", "")
```

In [3]:
```
# Add each prefix and suffix to remove to a list.
prefixes_suffixes = ["Dr. ", "Mr. ","Ms. ", "Mrs. ", "Miss ", " MD", " DDS", " DVM", '
```

In [ ]:
```
# Iterate through the "prefixes_suffixes" list and replace them with an empty space, '
```

```
for word in prefixes_suffixes:
    student_data_df["student_name"] = student_data_df["student_name"].str.replace(word
```

In [ ]:
```
# Put the cleaned students' names in another list.
student_names = student_data_df["student_name"].tolist()
student_names

# Create a new list and use it for the for loop to iterate through the list.
students_fixed = []
```

In [ ]:
```
# Create a new list and use it for the for loop to iterate through the list.
students_fixed = []

# Use an if statement to check the length of the name.

# If the name is greater than or equal to 3, add the name to the list.

for name in student_names:
    if len(name.split()) >= 3:
        students_fixed.append(name)

# Get the length of the students' names that are greater than or equal to 3.
len(students_fixed)
```

In [ ]:
```
# Add each prefix and suffix to remove to a list.
prefixes_suffixes = ["Dr. ", "Mr. ","Ms. ", "Mrs. ", "Miss ", " MD", " DDS", " DVM", '

# Iterate through the words in the "prefixes_suffixes" list and replace them with an e
for word in prefixes_suffixes:
    student_data_df["student_name"] = student_data_df["student_name"].str.replace(word
```

In [ ]:
```
# Combine the data into a single dataset.
school_data_complete_df = pd.merge(student_data_df, school_data_df, on=["school_name",
school_data_complete_df.head()

# Get the total number of students.
student_count = school_data_complete_df.count()
student_count
```

In [ ]:
```
school_data_complete_df[column].count()
```

In [ ]:
```
# Calculate the total number of schools.
school_count = school_data_df["school_name"].count()
school_count
```

In [ ]:
```
# Calculate the total number of schools
school_count_2 = school_data_complete_df["school_name"].unique()
school_count_2
```

In [ ]:
```
# Calculate the total budget.
total_budget = school_data_df["budget"].sum()
total_budget
```

In [ ]:
```
# Calculate the average reading score.
average_reading_score = school_data_complete_df["reading_score"].mean()
average_reading_score
```

```
# Calculate the average math score.
average_math_score = school_data_complete_df["math_score"].mean()
average_math_score
```

In [ ]:
```
passing_math = school_data_complete_df["math_score"] >= 70
passing_reading = school_data_complete_df["reading_score"] >= 70

# Get all the students who are passing math in a new DataFrame.
passing_math = school_data_complete_df[school_data_complete_df["math_score"] >= 70]
passing_math.head()
```

In [ ]:
```
# Get all the students that are passing reading in a new DataFrame.
passing_reading = school_data_complete_df[school_data_complete_df["reading_score"] >=
```

In [ ]:
```
# Calculate the number of students passing math.
passing_math_count = passing_math["student_name"].count()

# Calculate the number of students passing reading.
passing_reading_count = passing_reading["student_name"].count()
```

In [ ]:
```
# Calculate the percent that passed math.
passing_math_percentage = passing_math_count / float(student_count) * 100

# Calculate the percent that passed reading.
passing_reading_percentage = passing_reading_count / float(student_count) * 100
```

In [ ]:
```
# Calculate the students who passed both math and reading.
passing_math_reading = school_data_complete_df[(school_data_complete_df["math_score"]

passing_math_reading.head()
```

In [ ]:
```
# Calculate the number of students who passed both math and reading.
overall_passing_math_reading_count = passing_math_reading["student_name"].count()
overall_passing_math_reading_count
```

In [ ]:
```
# Calculate the overall passing percentage.
overall_passing_percentage = overall_passing_math_reading_count / student_count * 100
overall_passing_percentage
```

In [ ]:
```
# Adding a list of values with keys to create a new DataFrame.
district_summary_df = pd.DataFrame(
          [{"Total Schools": school_count,
          "Total Students": student_count,
          "Total Budget": total_budget,
          "Average Math Score": average_math_score,
          "Average Reading Score": average_reading_score,
          "% Passing Math": passing_math_percentage,
         "% Passing Reading": passing_reading_percentage,
        "% Overall Passing": overall_passing_percentage}])
district_summary_df
```

In [4]:
```
# Define the function "say_hello" so it prints "Hello!" when called.
def say_hello():
    print("Hello!")
```

```
# Call the function.
say_hello()
```

In [ ]:
```
# Define the function "say_something" so it prints whatever is passed as the variable
def say_something(something):
    print(something)

# Call the function.
say_something("Hello World")
```

In [5]:
```
Jane_says = "Hi, my name is Jane. I'm learning Python!"
say_something(Jane_says)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Input In [5], in <cell line: 2>()
      1 Jane_says = "Hi, my name is Jane. I'm learning Python!"
----> 2 say_something(Jane_says)

NameError: name 'say_something' is not defined
```

In [ ]:
```
# Define a function that calculates the percentage of students that passed both
# math and reading and returns the passing percentage when the function is called.

def passing_math_percent(pass_math_count, student_count):
    return pass_math_count / float(student_count) * 100
```

In [ ]:
```
passing_math_count = 29370
total_student_count = 39170

# Call the function.
passing_math_percent(passing_math_count, total_student_count)
```

In [ ]:
```
# A list of my grades.
my_grades = ['B', 'C', 'B' , 'D']

# Import pandas.
import pandas as pd
# Convert the my_grades to a Series
my_grades = pd.Series(my_grades)
my_grades
```

In [ ]:
```
map("current_value_1" : "new_value_1",  "current_value_2" : "new_value_2", etc)

# Change the grades by one letter grade.
my_grades.map({'B': 'A', 'C': 'B', 'D': 'C'})
```

In [ ]:
```
# Using the format() function.
my_grades = [92.34, 84.56, 86.78, 98.32]

for grade in my_grades:
    print("{:.0f}".format(grade))
```

In [ ]:
```
# Convert the numerical grades to a Series.
my_grades = pd.Series([92.34, 84.56, 86.78, 78.32])
my_grades
```

```
# Format the grades to the nearest whole number percent.
my_grades.map("{:.0f}".format)
```

In [ ]:
```
# Format the "Total Students" to have the comma for a thousands separator.
district_summary_df["Total Students"] = district_summary_df["Total Students"].map("{:,

district_summary_df["Total Students"]

# Format "Total Budget" to have the comma for a thousands separator, a decimal separat

district_summary_df["Total Budget"] = district_summary_df["Total Budget"].map("${:,.2f

district_summary_df["Total Budget"]
```

In [ ]:
```
# Format the columns.
district_summary_df["Average Math Score"] = district_summary_df["Average Math Score"].

district_summary_df["Average Reading Score"] = district_summary_df["Average Reading Sc

district_summary_df["% Passing Math"] = district_summary_df["% Passing Math"].map("{:.

district_summary_df["% Passing Reading"] = district_summary_df["% Passing Reading"].ma

district_summary_df["% Overall Passing"] = district_summary_df["% Overall Passing"].ma
```

In [ ]:
```
# Reorder the columns in the order you want them to appear.
new_column_order = ["column2", "column4", "column1"]

# Assign a new or the same DataFrame the new column order.
df = df[new_column_order]
```

In [ ]:
```
# Reorder the columns in the order you want them to appear.
new_column_order = ["Total Schools", "Total Students", "Total Budget","Average Math Sc

# Assign district summary df the new column order.
district_summary_df = district_summary_df[new_column_order]
district_summary_df
```

In [ ]:
```
# Add the per_school_types into a DataFrame for testing.
df = pd.DataFrame(per_school_types)
df

# Calculate the total student count.
per_school_counts = school_data_df["size"]
per_school_counts
```

In [ ]:
```
# Calculate the total student count.
per_school_counts = school_data_df.set_index(["school_name"])["size"]
per_school_counts

# Calculate the total school budget.
per_school_budget = school_data_df.set_index(["school_name"])["budget"]
per_school_budget
```

In [ ]:
```
# Calculate the per capita spending.
per_school_capita = per_school_budget / per_school_counts
per_school_capita
```

In [ ]:
```
# Calculate the math scores.
student_school_math = student_data_df.set_index(["school_name"])["math_score"]

# Calculate the average math scores.
per_school_averages = school_data_complete_df.groupby(["school_name"]).mean()
per_school_averages

# Calculate the average test scores.
per_school_math = school_data_complete_df.groupby(["school_name"]).mean()["math_score'

per_school_reading = school_data_complete_df.groupby(["school_name"]).mean()["reading_
```

In [ ]:
```
# Calculate the passing scores by creating a filtered DataFrame.
per_school_passing_math = school_data_complete_df[(school_data_complete_df["math_score

per_school_passing_reading = school_data_complete_df[(school_data_complete_df["reading

# Calculate the percentage of passing math and reading scores per school.
per_school_passing_math = per_school_passing_math / per_school_counts * 100

per_school_passing_reading = per_school_passing_reading / per_school_counts * 100
```

In [ ]:
```
# Calculate the students who passed both math and reading.
per_passing_math_reading = school_data_complete_df[(school_data_complete_df["math_scor

per_passing_math_reading.head()
```

In [ ]:
```
# Calculate the number of students who passed both math and reading.
per_passing_math_reading = per_passing_math_reading.groupby(["school_name"]).count()['

# Calculate the overall passing percentage.
per_overall_passing_percentage = per_passing_math_reading / per_school_counts * 100
```

In [ ]:
```
# Adding a list of values with keys to create a new DataFrame.

per_school_summary_df = pd.DataFrame({
             "School Type": per_school_types,
             "Total Students": per_school_counts,
             "Total School Budget": per_school_budget,
             "Per Student Budget": per_school_capita,
             "Average Math Score": per_school_math,
           "Average Reading Score": per_school_reading,
           "% Passing Math": per_school_passing_math,
           "% Passing Reading": per_school_passing_reading,
           "% Overall Passing": per_overall_passing_percentage})
per_school_summary_df.head()
```

In [ ]:
```
# Format the Total School Budget and the Per Student Budget columns.
per_school_summary_df["Total School Budget"] = per_school_summary_df["Total School Buc

per_school_summary_df["Per Student Budget"] = per_school_summary_df["Per Student Budge


# Display the data frame
per_school_summary_df.head()
```

In [ ]:
```
# Reorder the columns in the order you want them to appear.
```

```
new_column_order = ["School Type", "Total Students", "Total School Budget", "Per Stude

# Assign district summary df the new column order.
per_school_summary_df = per_school_summary_df[new_column_order]

per_school_summary_df.head()
```

In [ ]:
```
# Sort and show top five schools.
top_schools = per_school_summary_df.sort_values(["% Overall Passing"], ascending=False

top_schools.head()

# Sort and show top five schools.
bottom_schools = per_school_summary_df.sort_values(["% Overall Passing"], ascending=Tr

bottom_schools.head()
```

In [ ]:
```
class Cat:
    def __init__(self, name):
        self.name = name

first_cat = Cat('Felix')
print(first_cat.name)
```

In [ ]:
```
class Dog:
    def __init__(self, name, color, sound):
        self.name = name
        self.color = color
        self.sound = sound

    def bark(self):
        return self.sound + ' ' + self.sound

first_dog = Dog('Fido', 'brown', 'woof!')
print(    first_dog.name)
print(first_dog.color)
first_dog.bark()
```

In [ ]:
```
df.head()
```