Importing libraries

```
import pandas as pd
import io
import plotly.express as px
```

Read CSV file

```
def read_csv_to_df(file_path):
    """
    Reads a CSV file from the given file path and returns it as a DataFrame.

    Parameters:
    file_path (str): The path to the CSV file.

    Returns:
    pd.DataFrame: The DataFrame containing the data from the CSV file.
    """
    try:
        df = pd.read_csv(file_path)
        return df
    except Exception as e:
        print(f"Error reading the CSV file: {e}")
        return None

# Example usage:
# df = read_csv_to_df('path/to/your/file.csv')
# print(df.head())
```

Sum Function

```
def sum_grouped_by(df, group_by_col, eval_col):
    """
    Groups the dataframe by the specified column and calculates the sum of another column.

    Parameters:
    df (pd.DataFrame): The dataframe to operate on.
    group_by_col (str): The column name to group by.
    eval_col (str): The column name to sum.

    Returns:
    pd.DataFrame: A dataframe with the grouped and summed data.
    """
    result = df.groupby(group_by_col)[eval_col].sum().reset_index()
    return result
```

Count Function

```
def count_grouped_by(df, group_by_col, eval_col):
    """
    Groups the dataframe by the specified column and counts the occurrences of another column.

    Parameters:
    df (pd.DataFrame): The dataframe to operate on.
    group_by_col (str): The column name to group by.
    eval_col (str): The column name to count.

    Returns:
    pd.DataFrame: A dataframe with the grouped and counted data.
    """
    result = df.groupby(group_by_col)[eval_col].count().reset_index()
    return result
```

Create New AgeKey Column Function

```
def add_age_group_key(df):
    # Define the mapping for age groups to unique keys
    age_group_mapping = {
        '18 - 24 years': 'AG1',
        '25 - 34 years': 'AG2',
        '35 - 44 years': 'AG3',
        '45 - 54 years': 'AG4',
        '55 - 64 years': 'AG5',
        '65 - 74 years': 'AG6',
        '75 years and over': 'AG7'
    }

    # Create the AgeGroupKey column using the mapping
    df['AgeGroupKey'] = df['Age Group'].map(age_group_mapping)

    return df
```

Clean Dataframe Function

```
def identify_and_remove_total_rows(df):
    """
    Identify rows indicating totals and remove them from the dataframe.

    Parameters:
    df (pd.DataFrame): The dataframe to analyze.

    Returns:
    tuple: A dataframe with total rows, and the cleaned dataframe without total rows.
    """
    # Convert columns to string to ensure consistent comparisons
    df = df.astype(str)

    # Identify rows where any relevant column contains 'Total'
    total_rows = df[df.apply(lambda row: row.astype(str).str.contains('Total', case=False, na=False).any(), axis=1)]

    # Remove total rows from the dataframe
    cleaned_df = df[~df.index.isin(total_rows.index)]

    print("These are the removed rows:", total_rows)
    return cleaned_df
```

Data Quality Function

```python
def perform_dq_checks(df):
    """
    Performs comprehensive data quality checks on the given DataFrame and prints the results.

    Parameters:
    df (pd.DataFrame): The DataFrame to perform DQ checks on.
    """
    if df is None:
        print("DataFrame is None. Exiting DQ checks.")
        return

    print("Performing Data Quality Checks...\n")

    # Check for missing values
    missing_values = df.isnull().sum()
    print("Missing Values in Each Column:\n", missing_values)

    # Check for duplicate rows
    duplicate_rows = df.duplicated().sum()
    print("\nNumber of Duplicate Rows:", duplicate_rows)

    # Check for data types of columns
    data_types = df.dtypes
    print("\nData Types of Columns:\n", data_types)

    # Ensure column names are consistent (e.g., no leading/trailing spaces, all lowercase)
    clean_column_names = [col.strip().lower().replace(' ', '_') for col in df.columns]
    df.columns = clean_column_names
    print("\nCleaned Column Names:\n", df.columns)

    # Check for negative values in columns where they are not expected
    print("\nChecking for Negative Values in Columns:")
    for column in df.select_dtypes(include=['number']).columns:
        negative_values = (df[column] < 0).sum()
        if negative_values > 0:
            print(f"Column '{column}' has {negative_values} negative values")

    # Additional checks can be added here (e.g., consistency checks, custom validations)

    print("\nData Quality Checks Completed.")
```

Testing Functions

```python
def test_read_csv_to_df():
    # Create a sample CSV content
    sample_csv = """col1,col2,col3
    1,2,3
    4,5,6
    7,8,9
    """

    # Use io.StringIO to simulate a file-like object
    file_path = io.StringIO(sample_csv)

    # Call the function with the file-like object
    df = read_csv_to_df(file_path)

    # Expected DataFrame
    expected_df = pd.DataFrame({
        'col1': [1, 4, 7],
        'col2': [2, 5, 8],
        'col3': [3, 6, 9]
    })

    # Check if the DataFrame matches the expected DataFrame
    assert df.equals(expected_df), "Test failed: DataFrame does not match expected output"

    print("test_read_csv_to_df passed!")

# Run the test function
test_read_csv_to_df()
```

```
test_read_csv_to_df passed!
```

```
def test_sum_grouped_by():
    # Sample dataframe
    data = {
        'Operator': ['A', 'A', 'B', 'B', 'C'],
        'Number_of_chargers': [10, 15, 20, 25, 30]
    }
    df = pd.DataFrame(data)

    # Expected result
    expected_data = {
        'Operator': ['A', 'B', 'C'],
        'Number_of_chargers': [25, 45, 30]
    }
    expected_df = pd.DataFrame(expected_data)

    # Result from the function
    result_df = sum_grouped_by(df, 'Operator', 'Number_of_chargers')

    # Assertions
    assert result_df.equals(expected_df), f"Expected {expected_df} but got {result_df}"
    print("test_sum_grouped_by passed!")

# Run the test
test_sum_grouped_by()
```

```
test_sum_grouped_by passed!
```

```
def test_count_grouped_by():
    # Sample dataframe
    data = {
        'Operator': ['A', 'A', 'B', 'B', 'C', 'C', 'C'],
        'Location': ['Loc1', 'Loc2', 'Loc3', 'Loc4', 'Loc5', 'Loc6', 'Loc7']
    }
    df = pd.DataFrame(data)

    # Expected result
    expected_data = {
        'Operator': ['A', 'B', 'C'],
        'Location': [2, 2, 3]
    }
    expected_df = pd.DataFrame(expected_data)

    # Result from the function
    result_df = count_grouped_by(df, 'Operator', 'Location')

    # Assertions
    assert result_df.equals(expected_df), f"Expected {expected_df} but got {result_df}"
    print("test_count_grouped_by passed!")

# Run the test
test_count_grouped_by()
```

```
test_count_grouped_by passed!
```

```
def test_add_age_group_key():
    # Test data
    data_ev = {'Age Group': ['18 - 24 years', '25 - 34 years', '35 - 44 years',
                             '45 - 54 years', '55 - 64 years', '65 - 74 years',
                             '75 years and over'],
               'Ownership': [10, 20, 30, 40, 50, 60, 70]}
    df_ev_ownership = pd.DataFrame(data_ev)

    # Expected result
    expected_data = {'Age Group': ['18 - 24 years', '25 - 34 years', '35 - 44 years',
                                   '45 - 54 years', '55 - 64 years', '65 - 74 years',
                                   '75 years and over'],
                     'Ownership': [10, 20, 30, 40, 50, 60, 70],
                     'AgeGroupKey': ['AG1', 'AG2', 'AG3', 'AG4', 'AG5', 'AG6', 'AG7']}
    expected_df = pd.DataFrame(expected_data)

    # Apply the function
    result_df = add_age_group_key(df_ev_ownership)

    # Check if the result matches the expected output
    assert result_df.equals(expected_df), f"Test failed! \nResult:\n{result_df}\nExpected:\n{expected_df}"

    print("test_add_age_group_key passed!")

# Run the test function
test_add_age_group_key()
```

```
test_add_age_group_key passed!
```

Reading EV Charging Point CSV and analysing data

```
df_ev_sdcc = read_csv_to_df('/content/Public_EV_Charging_Points_SDCC.csv')
# df_ev_sdcc.head(2)
```

```
perform_dq_checks(df_ev_sdcc)
```

```
Performing Data Quality Checks...

Missing Values in Each Column:
 LEA                  0
Location             0
Operator             0
Number_of_chargers   0
Type                 0
Rating               0
ObjectId             0
dtype: int64

Number of Duplicate Rows: 0

Data Types of Columns:
 LEA                  object
Location             object
Operator             object
Number_of_chargers    int64
Type                 object
Rating               object
ObjectId              int64
dtype: object

Cleaned Column Names:
 Index(['lea', 'location', 'operator', 'number_of_chargers', 'type', 'rating',
       'objectid'],
      dtype='object')

Checking for Negative Values in Columns:

Data Quality Checks Completed.
```

```
sum_grouped_by(df_ev_sdcc, 'operator', 'number_of_chargers')
```

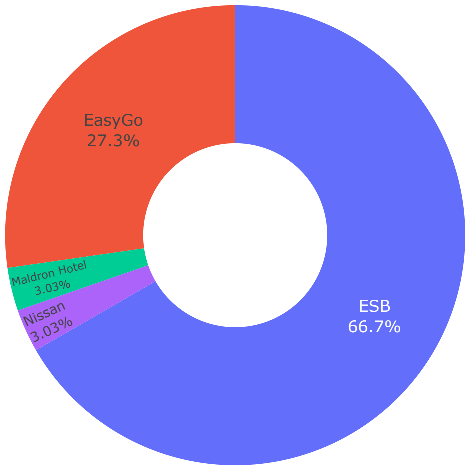
	operator	number_of_chargers
0	ESB	40
1	EasyGo	44
2	Maldron Hotel	2
3	Nissan	1

```
grouped_df = count_grouped_by(df_ev_sdcc, 'operator', 'location')
```

```
def generate_donut_chart(df, names, values):
    fig = px.pie(df, names=names, values=values, hole=0.4, title='Number of Location by Operator in SDCC')
    fig.update_traces(textposition='inside', textinfo='percent+label')
    fig.show()

# Example usage
generate_donut_chart(grouped_df, 'operator', 'location')
```

Number of Location by Operator in SDCC



Reading Vehicle Fleet CSV and cleaning data

```
df_vehicle_fleet = read_csv_to_df('/content/Vehicular_Fleet_FCC.csv')
# df_vehicle_fleet.head(2)
```

```
perform_dq_checks(df_vehicle_fleet)
```

```
Performing Data Quality Checks...

Missing Values in Each Column:
 _Year                     0
Vehicle_Make              1
Vehicle_Class             0
Model/Trim_Description    0
Amount                    0
OBJECTID                  0
dtype: int64

Number of Duplicate Rows: 0

Data Types of Columns:
 _Year                     object
Vehicle_Make              object
Vehicle_Class             object
Model/Trim_Description    object
Amount                    object
OBJECTID                   int64
dtype: object

Cleaned Column Names:
 Index(['_year', 'vehicle_make', 'vehicle_class', 'model/trim_description',
       'amount', 'objectid'],
      dtype='object')

Checking for Negative Values in Columns:

Data Quality Checks Completed.
```

Removing undesired rows

```
df_vehicle_fleet_cleaned = identify_and_remove_total_rows(df_vehicle_fleet)
df_vehicle_fleet_cleaned = df_vehicle_fleet_cleaned.drop(index=27)
```

```
These are the removed rows:    _year vehicle_make vehicle_class model/trim_description amount objectid
26  2023          nan   Total Fleet       Total Fleet 2023    175       26
56  2024        Total   Total Fleet       Total Fleet 2024    196       56
```

Swapping incorrect columns for year 2024

```
# Identify rows for the year 2024
df_2024 = df_vehicle_fleet_cleaned[df_vehicle_fleet_cleaned['_year'] == '2024'].copy()

# Swap the values of 'Vehicle_Make' and 'Vehicle_Class' for the year 2024
df_2024[['vehicle_make', 'vehicle_class']] = df_2024[['vehicle_class', 'vehicle_make']]
#print(df_2024.head(2))

# Identify rows for the year 2023
df_2023 = df_vehicle_fleet_cleaned[df_vehicle_fleet_cleaned['_year'] == '2023']

# Combine the dataframes
df_combined = pd.concat([df_2023, df_2024])

df_combined.head(2)
```

	_year	vehicle_make	vehicle_class	model/trim_description	amount	objectid
0	2023	DAF	Unibody Winter Gritter	18 ton 2 Axle Unibody	8	0
1	2023	Mitsubishi	Fuso Canter	3.5 ton Truck	19	1

Analysing number of manufacturers for the year

```
df_combined.groupby(['_year'])['vehicle_class'].count()
```

```
_year
2023    26
2024    28
Name: vehicle_class, dtype: int64
```

We can see that number of manufacturers for EV has increased from 26 in 2023 to 28 in 2024.

Reading Vehicle Adoption CSV

```
df_ev_ownership =read_csv_to_df('/content/NTA43.20240714140251.csv')
# df_ev_ownership.head(2)
```

Finding Age Group which has highest adoption of EV

```
df_ev_ownership_adoption = df_ev_ownership[df_ev_ownership['Statistic Label']=='Owns an Electric Vehicle (EV)'].sort_values(
df_ev_ownership_adoption.head(5)
```

	C02076V02508	Age Group	C02199V02655	Sex	TLIST(A1)	Year	STATISTIC	Statistic Label	UNIT	VALUE
22	570	65 - 74 years	2	Female	2019	2019	NTA43C01	Owns an Electric Vehicle (EV)	%	3.2
12	500	45 - 54 years	1	Male	2019	2019	NTA43C01	Owns an Electric Vehicle (EV)	%	2.4
16	535	55 - 64 years	1	Male	2019	2019	NTA43C01	Owns an Electric Vehicle (EV)	%	2.2
18	535	55 - 64 years	2	Female	2019	2019	NTA43C01	Owns an Electric Vehicle (EV)	%	2.2
4	415	25 - 34 years	1	Male	2019	2019	NTA43C01	Owns an Electric Vehicle (EV)	%	2.0

Add AgeGroupKey Column

```
# Add AgeGroupKey column
df_ev_ownership_with_key = add_age_group_key(df_ev_ownership)
```

Validating AgeGroupKey Column

```
# Validating mappings of new key column
distinct_combinations = df_ev_ownership_with_key[['Age Group', 'AgeGroupKey']].drop_duplicates()
distinct_combinations
```

	Age Group	AgeGroupKey
0	18 - 24 years	AG1
4	25 - 34 years	AG2
8	35 - 44 years	AG3
12	45 - 54 years	AG4
16	55 - 64 years	AG5
20	65 - 74 years	AG6
24	75 years and over	AG7

Reading EV Interested People CSV

```
df_ev_interested = pd.read_csv('/content/NTA49.20240714112939.csv')
# df_ev_interested.head(2)
```

Add AgeGroupKey Column

```
# Add AgeGroupKey column
df_ev_interested_with_key = add_age_group_key(df_ev_interested)
```

Validating AgeGroupKey Column

```
# Get distinct combinations of AgeGroup and AgeGroupKey
distinct_combinations = df_ev_interested_with_key[['Age Group', 'AgeGroupKey']].drop_duplicates()
distinct_combinations
```

	Age Group	AgeGroupKey
0	18 - 24 years	AG1
18	25 - 34 years	AG2
36	35 - 44 years	AG3
54	45 - 54 years	AG4
72	55 - 64 years	AG5
90	65 - 74 years	AG6
108	75 years and over	AG7

Joining two dataframes

```
# Set AgeGroupKey as the index for both DataFrames
df_ev_interested_with_key.set_index('AgeGroupKey')
df_ev_ownership_with_key.set_index('AgeGroupKey')
joined_df = df_ev_interested_with_key.join(df_ev_ownership_with_key, how='left', lsuffix='_df', rsuffix='_ev')
```

Calculating highest factor for people who want to adopt CSV

```
# Grouping by 'Influencing factor of EV purchase' and calculating the mean, then sorting
df_sorted = df_ev_interested_with_key.groupby('Influencing factor of EV purchase')['VALUE'].mean().sort_values(ascending=Fal
df_sorted.rename(columns={'VALUE': 'mean_value'}, inplace=True)

# Display the sorted DataFrame
df_sorted
```

	Influencing factor of EV purchase	mean_value
0	Making more of a contribution to a better envi...	55.628571
1	More availability of charging points away from...	50.278571
2	Better affordability to run	50.064286
3	Better value	45.771429
4	More availability of overnight charging at low...	29.057143
5	Reduced noise pollution	15.228571
6	Improved health from use	9.271429
7	Other influencing factors	4.992857
8	Higher toll discounts	2.964286