1.

Loading Data

```
#Using Pandas for CSV:

import pandas as pd

# Load data from CSV
file_path = "C:/Users/shivakumar.kvskr/Downloads/population/world_population_data.csv"
data = pd.read_csv(file_path)

# Display the first few rows of the dataset
print(data.head())
```

```
# Using Pandas for Excel
# import pandas as pd

# # Load data from Excel
# file_path = 'example_data.xlsx'
# data = pd.read_excel(file_path)

# # Display the first few rows of the dataset
# print(data.head())
```

```
#Using Pandas for SQL (Assuming you have a SQL database)
# import pandas as pd
# from sqlalchemy import create_engine

# # Connect to the SQL database
# engine = create_engine('your_database_connection_string')

# # Load data from SQL query
# query = 'SELECT * FROM your_table_name'
# data = pd.read_sql(query, engine)

# # Display the first few rows of the dataset
# print(data.head())
```

```
#Using NumPy for Text File
# import numpy as np

# # Load data from text file
# file_path = 'example_data.txt'
# data = np.loadtxt(file_path, delimiter=',')

# # Display the loaded data
# print(data)
```

```
# Using CSV Module for CSV
# import csv

# # Load data from CSV using the csv module
# file_path = 'example_data.csv'
# with open(file_path, 'r') as file:
#     reader = csv.reader(file)
#     data = list(reader)

# # Display the loaded data
# print(data)
```

2.

Describing Data:

```
# Display general information about the dataset
print(data.info())
```

```
# Display summary statistics for numerical columns
print(data.describe())
```

```
# Display the first few rows of the dataset
print(data.head())
```

```
# Display the last few rows of the dataset
print(data.tail())
```

```
#Count Rows and Columns
row_count, col_count = data.shape
print(f"Number of Rows: {row_count}, Number of Columns: {col_count}")
```

```
#Count Distinct Values:
unique_values_count = data['1990 population'].nunique()
print(f"Number of Unique Values in '1990 population': {unique_values_count}")
```

```
# Quantiles:Calculate quantiles for a numeric column.
quantiles = data['2020 population'].quantile([0.25, 0.5, 0.75])
print("Quantiles:")
print(quantiles)
```

```
# # Data Resampling (Time Series): Resample time series data to a different frequency.
# data['datetime_column'] = pd.to_datetime(data['datetime_column'])
# resampled_data = data.resample('D', on='datetime_column').mean()
# print("Resampled Data:")
# print(resampled_data.head())
```

3.

Data Visualization:

```
#Histogram
import matplotlib.pyplot as plt
data['1980 population'].hist()
plt.title('1980 population')
plt.show()
```

```
# Boxplot
import seaborn as sns
sns.boxplot(x='continent', y='rank', data=data)
plt.title('Boxplot')
plt.show()
```

```
#Scatter Plot:
plt.scatter(data['continent'], data['rank'])
plt.title('Scatter Plot')
plt.xlabel('continent')
plt.ylabel('rank')
plt.show()
```

4.

Data Distribution Analysis:

```
#Kernel Density Estimation (KDE)
sns.kdeplot(data['rank']) #numerical_column
plt.title('Kernel Density Estimation')
plt.show()
```

Categorical Data Analysis:

```
# Countplot
sns.countplot(x='continent', data=data) #category
plt.title('Countplot')
plt.show()
```

Data Relationships:

```
# Pairplot
import warnings

# Suppress seaborn warning about tight layout
warnings.filterwarnings("ignore", category=UserWarning, module="seaborn")

sns.pairplot(data)
plt.title('Pairplot')
plt.show()
```

5.

Missing Data Analysis:

```
#Heatmap for Missing Values:
sns.heatmap(data.isnull(), cbar=False, cmap='viridis')
plt.title('Missing Data Heatmap')
plt.show()
```

Outlier Analysis:

```
# Boxplot for Outliers:
sns.boxplot(x='rank', data=data) #numerical_column
plt.title('Boxplot for Outliers')
plt.show()
```

```
# #Time Series Analysis:
# data['Date'] = pd.to_datetime(data['Date'])
# data.set_index('Date')['numerical_column'].plot()
# plt.title('Time Series Plot')
# plt.show()
```

6.

Handling Inconsistent Data:

```
#Standardizing Text Data:
# Convert text data to lowercase
data['continent'] = data['continent'].str.lower() #textcolumn
```

```
# Handling Categorical Data:
# Convert categorical data to a consistent format (e.g., uppercase)
data['country'] = data['country'].str.upper()
```

Dealing with Data Types:

```
# # Converting Data Types:
# # Convert a column to a different data type (e.g., from object to datetime)
# data['date_column'] = pd.to_datetime(data['date_column'])
```

Handling Inconsistent Data Formats:

```
# #Removing Special Characters
# # Remove special characters from a column
# data['country'] = data['country'].str.replace('[^a-zA-Z0-9]', '') #text_column
```

7.

Handling Missing Values:

```
#Removing Rows with Missing Values:
# Remove rows with any missing values
data.dropna(inplace=True)
```

```
# # Imputing Missing Values:
# # Fill missing values with a specified value (e.g., mean)
# data['column_name'].fillna(data['column_name'].mean(), inplace=True)
```

```
# # Fill missing values in numerical columns with the mean
# data.fillna(data.mean(), inplace=True)

# # Fill missing values in categorical columns with the most frequent value
# data.fillna(data.mode().iloc[0], inplace=True)
```

```
#Removing Duplicates:
# Remove duplicate rows based on all columns
data.drop_duplicates(inplace=True)
```

```
# Handling Outliers
# Identifying Outliers:
# Using z-score to identify outliers
from scipy.stats import zscore

z_scores = zscore(data['density (km²)']) #numericals
outliers = (z_scores > 3) | (z_scores < -3)
data_no_outliers = data[~outliers]
```

```
# Trimming or Capping Outliers:
# Trim or cap outliers based on a threshold
import numpy as np
# Set the upper threshold value
upper_threshold = 1000  # Replace 1000 with your desired threshold value
# Trim or cap outliers based on the upper threshold
data['rank'] = np.where(data['rank'] > upper_threshold, upper_threshold, data['rank'])
```

```
# Set the upper threshold based on the 95th percentile
upper_threshold = data['rank'].quantile(0.95)

# Trim or cap outliers based on the upper threshold
data['rank'] = np.where(data['rank'] > upper_threshold, upper_threshold, data['rank'])
# Display the first few rows of the modified dataset
# print(data.head())
```

8.

Handling Zero or Negative Values:

```
# Replacing Zero or Negative Values:
# Replace zero or negative values with a specified value
data['2020 population'] = data['2020 population'].apply(lambda x: max(x, 0))  #numerical_column
```

Removing Duplicates:

```
# Remove duplicate rows based on all columns
data.drop_duplicates(inplace=True)
```

Checking Missing Values:

```
# Check for missing values in the entire dataset
print(data.isnull().sum())
```

```
# Check for missing values in a specific column
print(data['growth rate'].isnull().sum())
```

9.

Feature Scaling

```
#Min-Max Scaling
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
data['area (km²)'] = scaler.fit_transform(data[['area (km²)']]) #numerical_column
```

```
#Standardization (Z-score Scaling)
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
data['area (km²)'] = scaler.fit_transform(data[['area (km²)']]) #numerical_column
```

Feature Engineering

```
# Creating a New Feature:
# Combine existing features to create a new one
data['new_feature'] = data['2023 population'] + data['2022 population']
print(data['new_feature'])
```

10.

Handling Text Data

```
# Tokenization (Splitting Text into Words)
# Tokenize a text column
data['country'] = data['country'].apply(lambda x: x.split()) #text_column
```

```
#Bag-of-Words Representation (Count Vectorization)
from sklearn.feature_extraction.text import CountVectorizer

vectorizer = CountVectorizer()
text_matrix = vectorizer.fit_transform(data['continent']) #text_column
```

11.

transform lists to strings

```
data['country'] = data['country'].apply(lambda x: ', '.join(map(str, x)) if isinstance(x, list) else x)

# Convert object columns to categorical data type
data['country'] = data['country'].astype('category')
```

```
#transform lists to strings
data['cca3'] = data['cca3'].apply(lambda x: ', '.join(map(str, x)) if isinstance(x, list) else x)

# Convert object columns to categorical data type
data['cca3'] = data['cca3'].astype('category')
```

```
#transform lists to strings
data['continent'] = data['continent'].apply(lambda x: ', '.join(map(str, x)) if isinstance(x, list) else x)

# Convert object columns to categorical data type
data['continent'] = data['continent'].astype('category')
```

```
#Filtering Rows Where 'text_column' Contains a Specific Substring
substring = 'America'
filtered_data_contains_substring = data[data['continent'].str.contains(substring, case=False)]
print(filtered_data_contains_substring)
```

12.

data filtering

```
#Filtering Rows Based on a Single Condition
filtered_data = data[data['rank'] > 222] #numerical_column
print(filtered_data)
```

```
#Filtering Rows Based on Multiple Conditions:
filtered_data_multiple_conditions = data[(data['rank'] > 200) & (data['continent'])]
print(filtered_data_multiple_conditions)
```

```
#Filtering Rows with Null Values
filtered_data_null_values = data[data['1980 population'].isnull()]
print(filtered_data_null_values)
```

```
#Filtering Rows with Non-Null Values
filtered_data_non_null_values = data[data['rank'].notnull()]
print(filtered_data_non_null_values)
```

```
#Filtering Rows with Unique Values:
filtered_data_unique_values = data[data['continent'].duplicated(keep=False)]
print(filtered_data_unique_values)
```

```
#Filtering Rows with Duplicates
filtered_data_duplicates = data[data.duplicated(subset=['2023 population', '1980 population'], keep=False)]
print(filtered_data_duplicates)
```

```
print(data.columns)
```

```
print(data['country'].isnull().sum())
```

```
print(data['country'].dtype)
```

13.

aggregation and grouping

```
# Aggregating mean for 'numerical_column'
mean_numerical = data['density (km²)'].mean() #numerical_column
print("Mean of  density (km²):", mean_numerical)
```

```
#Aggregating Sum for a Numerical Column:
# Aggregating sum for 'numerical_column'
sum_numerical = data['density (km²)'].sum() #numerical_column
print("Sum of density (km²):", sum_numerical)
```

```
#Grouping Data by a Categorical Column and Calculating Mean
# Grouping data by 'categorical_column' and calculating mean for each group
grouped_mean_data = data.groupby('continent')['area (km²)'].mean()
print(grouped_mean_data)
```

```
# Grouping Data by Multiple Categorical Columns and Calculating Mean:
# Grouping data by multiple categorical columns and calculating mean for each group
grouped_mean_data_multiple_columns = data.groupby(['continent', 'cca3'])['rank'].mean()
print(grouped_mean_data_multiple_columns)
```

```
#Aggregating with Multiple Functions Using GroupBy
# Grouping data by 'categorical_column' and calculating both mean and sum for each group
grouped_data_multiple_aggs = data.groupby('cca3').agg({'2023 population': ['mean', 'sum']})
print(grouped_data_multiple_aggs)
```

```
#Counting Occurrences in a Categorical Column
# Counting occurrences of each unique value in 'categorical_column'
value_counts = data['country'].value_counts()
print(value_counts)
```

```
#Aggregating and Counting Based on Grouping
# Grouping data by 'categorical_column' and calculating mean and count for each group
grouped_data_mean_count = data.groupby('cca3').agg({'2023 population': 'mean', 'country': 'count'})
print(grouped_data_mean_count)
```

14.

exporting the data

```
# Export to CSV
data.to_csv('output_data.csv', index=False)
```

```
# Export to Excel
data.to_excel('output_data.xlsx', index=False)
```

```
# Export to JSON
data.to_json('output_data.json', orient='records')
```

```
# from sqlalchemy import create_engine

# # Create SQLite engine
# engine = create_engine('sqlite:///output_data.db')

# # Export to SQL database
# data.to_sql('table_name', engine, index=False, if_exists='replace')
```

```
# Export to Parquet format
data.to_parquet('output_data.parquet', index=False)
```

```
# Export to Pickle format
data.to_pickle('output_data.pkl')
```

```
# # Copy to clipboard
# data.to_clipboard(index=False)
```

```
# Export to HDF5 with format="table"
data.to_hdf('output_data.h5', key='table', mode='w', format='table')
```

Tasks on New Dataset: THE World University Rankings 2016-2024

1. Display general information about the dataset
2. Display summary statistics for numerical columns
3. Display the last few rows of the dataset
4. Display the Count of Rows and Columns
5. Count Distinct Values for Research Quality
6. Data Visualization using Histogram for Overall Score
7. Data Visualization using Boxplot for Name, Overall Score
8. Density Estimation for International Students
9. Find the missing values
10. Heatmap for all the missing values
11. Remove the duplicates
12. Fill the missing values with mean
13. Find the outliers using z-score
14. Boxplot for outliers
15. Create a new column by combining teaching, research environment, research quality
16. Filter the data whose Teaching is >80 and Research Quality < 85
17. Convert object columns to categorical data type and print
18. Grouping Data by Multiple Categorical Columns and Calculating Mode
19. Grouping data by country and calculating mean and count for Student Population
20. Export all the solutions to 19 questions into JSON with your last 3 digits of the roll number and section. (example format – 131_A or 153_B)