28 July 2025

Niranjan Kumar

NYC Air Quality Analytics

Python Data Cleaning & Exploration Report

Topics

1. Data Loading
2. Data Understanding
3. Data Exploration
4. Data Cleaning
5. Data Analysis (Exploratory Data Analysis)
6. Data Visualization

1. **Data Loading**
   - For Python

Data Loading

- *Dataset is in CSV (Comma Seperated Values)*

```
df = pd.read_csv("Air_Quality.csv")
```

2. **Data Understandinng**
   - Shape

- *Rows - 18862*
- *Columns - 12*

```
# To identify the number of rows and columns in a data set
df.shape
```

```
(18862, 12)
```

   - Exploring first 5 rows and columns

```
# To identify the first five rows and columns in a data set
df.head(5)
```

	Unique ID	Indicator ID	Name	Measure	Measure Info	Geo Type Name	Geo Join ID	Geo Place Name	Time Period	Start_Date	Data Value	Message
0	336867	375	Nitrogen dioxide (NO2)	Mean	ppb	CD	407	Flushing and Whitestone (CD7)	Winter 2014-15	12-01-2014	23.97	NaN
1	336741	375	Nitrogen dioxide (NO2)	Mean	ppb	CD	107	Upper West Side (CD7)	Winter 2014-15	12-01-2014	27.42	NaN
2	550157	375	Nitrogen dioxide (NO2)	Mean	ppb	CD	414	Rockaway and Broad Channel (CD14)	Annual Average 2017	01-01-2017	12.55	NaN
3	412802	375	Nitrogen dioxide (NO2)	Mean	ppb	CD	407	Flushing and Whitestone (CD7)	Winter 2015-16	12-01-2015	22.63	NaN
4	412803	375	Nitrogen dioxide (NO2)	Mean	ppb	CD	407	Flushing and Whitestone (CD7)	Summer 2016	06-01-2016	14.00	NaN

   - Identifying all the column names

```
# TO identify the name of all the columns
df.columns
```

```
Index(['Unique ID', 'Indicator ID', 'Name', 'Measure', 'Measure Info',
       'Geo Type Name', 'Geo Join ID', 'Geo Place Name', 'Time Period',
       'Start_Date', 'Data Value', 'Message'],
      dtype='object')
```

   - Dropping the usless columns

```
# List of columns to drop that are not required for the Data Analysis
cols_to_drop = [
    'Unique ID',
    'Indicator ID',
    'Geo Type Name',
    'Geo Join ID',
    'Message'
]

# Drop the columns
df.drop(columns=cols_to_drop, inplace=True)
```

   - Check for the Data Types

```
df.dtypes
```

```
Name               object
Measure            object
Measure Info       object
Geo Place Name     object
Time Period        object
Start_Date         object
Data Value        float64
dtype: object
```

- Check for the missing (null) values

```
df.isna().sum()
```

```
Name              0
Measure           0
Measure Info      0
Geo Place Name    0
Time Period       0
Start_Date        0
Data Value        0
dtype: int64
```

3. Data Exploration

- Name

Name

- Data type is object
- No missing values
- Need for standarization of the column
- Total 18 unique values
- Highest is Nitrogen dioxide (NO2) and the values count is 6345
- Lowest is Boiler Emissions- Total PM2.5 Emissions and the value counts is 96
- `Data Cleaning`
- Convert the data type
- Categorize them or create a new column (Featured Enginerring)
- Trim spaces
- Fix spacing
- Fix tittle case
- Convert empty string to NA if available

- Measure

Measure

- No missing values in the data set
- Total 8 unique values
- highest is from Mean and the count is 14805
- Lowest is from Estimated annual rate (age 30+) and the count is 240
- Data type needs to be converted
- `Data Cleaning`
- Convert the data type
- Categorize them or create a new column (Featured Enginerring)
- Trim spaces
- Fix spacing
- Fix tittle case
- Convert empty string to NA if available
- Creating a new Featured Column
- Binning the Measures in to age categories

- Measure Info

`Measure Info`

- The Data type is object
- No missing values in the column
- Highest is ppb and the count is 8460
- Lowest is per 100,000 and the count is 240
- Total unique values are - 8
- Data type needs to be converted
- `Data Cleaning`
- Data type conversion
- Trim spaces
- Fix spacing
- Fix tittle case
- Convert empty string to NA if available
- merging per 100,000 & per 100,000 adults

- Geo Place Name

`Geo Place Name`

- Data Type object
- No missing values
- Total 114 unique categories
- Highest values are from East New York and the value counts are 281
- Lowest values are from Southern SI and the value counts are 105
- `Data Cleaning`
- Data type conversion
- Trim spaces
- Fix spacing
- Fix tittle case
- Convert empty string to NA if available

- Time Period

`Time Period`

- Total 57 uniqe values
- No missing values
- Data type is object
- Highest value is from 2012-2014 and the value count is 480
- Lowest value is from 2014 and the value count is 96
- `Data Cleaning`
- Convert the data type
- Categorize them or create a new column (Featured Enginerring)
- Trim spaces
- Fix spacing
- Fix tittle case
- Convert empty string to NA if available

- Start Date

`Start_Date`

- Data type is object
- No missing values
- One category is in wrong format MM/DD/YY other are in DD/MM/YY
- The highest values is from 01-01-2015 and the count is 906
- The lowest value is from 01/01/2014 and the value count is 96
- Total 46 unique categories of Date
- `Data Cleaning`
- Convert data type
- Convert the column drom MM/DD/YY to DD/MM/YY ie standarize the column

- Date Value

`Data Value`

- Data type is float64
- No missing values
- Large numbers of outliers present in the data set
- Total 7375 values present
- Count 18862.000000
- mean 21.051580
- std 23.564920
- min 0.000000
- 25% 8.742004
- 50% 14.790000
- 75% 26.267500
- max 424.700000
- `Data Cleaning`
- Round upto 2 decimal place
- Deal with the outliers

4. Data Cleaning

- Name

```
# Strip leading/trailing spaces
df['Name'] = df['Name'].str.strip()

# Replace multiple spaces with single space
df['Name'] = df['Name'].str.replace(r'\s+', ' ', regex=True)

# Convert to Title Case (if that makes sense for your data)
df['Name'] = df['Name'].str.title()

# Optionally, check unique values after cleaning
print(df['Name'].unique())
```

```
# Add space after dash if missing
df['Name'] = df['Name'].str.replace(r'Boiler Emissions-\s*', 'Boiler Emissions - ', regex=True)

# Standardize chemical symbols:
df['Name'] = df['Name'].str.replace(r'No2', 'NO2', regex=False)
df['Name'] = df['Name'].str.replace(r'Pm 2.5', 'PM2.5', regex=False)
df['Name'] = df['Name'].str.replace(r'Pm2.5', 'PM2.5', regex=False)
df['Name'] = df['Name'].str.replace(r'O3', 'O3', regex=False)

# Double check unique values again
print(df['Name'].unique())
```

- Measure

```python
# Trim spaces
df['Measure'] = df['Measure'].str.strip()

# Fix casing — choose one style (title case recommended)
df['Measure'] = df['Measure'].str.title()

# Convert to category for efficiency
df['Measure'] = df['Measure'].astype('category')

# Check unique values after cleaning
print(df['Measure'].unique())
```

```python
def extract_age_group(measure):
    if 'Under Age 18' in measure:
        return 'Under 18'
    elif 'Age 18+' in measure:
        return '18+'
    elif 'Age 30+' in measure:
        return '30+'
    else:
        return 'All Ages'

df['Age_Group'] = df['Measure'].apply(extract_age_group)
```

- Measure Info

```python
# Trim spaces
df['Measure Info'] = df['Measure Info'].str.strip()

# Consistent casing — I recommend lowercase or title case (choose one)
df['Measure Info'] = df['Measure Info'].str.lower()

# Convert to category
df['Measure Info'] = df['Measure Info'].astype('category')

# Check unique values
print(df['Measure Info'].unique())
```

- Geo Place Name

```python
# Standardization function
def standardize_measure(info):
    info = str(info).strip().lower()

    # Handle unit variations
    if any(unit in info for unit in ['µg/m3', 'âµg/m3', 'mcg/m3', 'ug/m3']):
        return 'mcg/m3'

    # Handle population rate variations
    if 'per 100,000' in info:
        if 'children' in info:
            return 'per 100,000 children'
        return 'per 100,000 adults'

    return info

# Apply standardization
df['Measure_Standardized'] = df['Measure Info'].apply(standardize_measure)

# Get value counts
measure_counts = df['Measure_Standardized'].value_counts().reset_index()
measure_counts.columns = ['Measure_Standardized', 'Count']

# Create desired output format
output_measures = [
    'ppb',
    'mcg/m3',
    'per 100,000 adults',
    'per 100,000 children',
    'per square mile',
    'number'
]

# Merge and format
result = pd.DataFrame({'Measure Info': output_measures})
result = result.merge(
    measure_counts,
    left_on='Measure Info',
    right_on='Measure_Standardized',
    how='left'
).drop(columns='Measure_Standardized')

# Fill NA with 0 and format numbers
result['Count'] = result['Count'].fillna(0).astype(int)
result['Count'] = result['Count'].apply(lambda x: f"{x:,}")

# Print formatted output
print(result.to_markdown(index=False, stralign='left'))
```

- Geo Place name

```
# Trim spaces
df['Geo Place Name'] = df['Geo Place Name'].str.strip()

# Replace multiple spaces with a single space
df['Geo Place Name'] = df['Geo Place Name'].str.replace(r'\s+', ' ', regex=True)

# Title case for consistency
df['Geo Place Name'] = df['Geo Place Name'].str.title()

# Replace empty strings with NaN
df['Geo Place Name'] = df['Geo Place Name'].replace('', np.nan)

# Convert to category for memory optimization
df['Geo Place Name'] = df['Geo Place Name'].astype('category')

# Check unique values and sample
print(df['Geo Place Name'].unique())
```

- Time Period

```
def extract_start_year(time_period):
    match = re.search(r'(\d{4})', str(time_period))
    if match:
        return int(match.group(1))
    return np.nan

df['Start Year'] = df['Time Period'].apply(extract_start_year).astype('Int64')

# Drop End Year and Duration columns if present
df = df.drop(columns=['End Year', 'Duration'], errors='ignore')

print(df[['Time Period', 'Start Year']].head(10))
```

- Start Date

```
def parse_dates(date_str):
    for fmt in ('%d/%m/%y', '%m/%d/%y', '%Y-%m-%d', '%d-%m-%Y', '%m-%d-%Y'):
        try:
            return pd.to_datetime(date_str, format=fmt)
        except (ValueError, TypeError):
            continue
    return pd.NaT  # if no format matches

df['Start_Date'] = df['Start_Date'].apply(parse_dates)

# Check for any remaining null dates
print(df['Start_Date'].isna().sum())

# Confirm the dtype
print(df['Start_Date'].dtype)

# Sample check
print(df['Start_Date'].head())
```

- Date Value

```
Q1 = df['Data Value'].quantile(0.25)
Q3 = df['Data Value'].quantile(0.75)
IQR = Q3 - Q1

lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Cap outliers and update df in place
df['Data Value'] = df['Data Value'].clip(lower=lower_bound, upper=upper_bound)

# Now count outliers again on the capped data
outliers_after_capping = df[(df['Data Value'] < lower_bound) | (df['Data Value'] > upper_bound)].shape[0]
print(f"Outliers after capping: {outliers_after_capping}")
```

5. Exploraory Data Analysis

Categorizing columns as continous, categorical, time series

```
categorical = ['Name', 'Measure', 'Measure Info','Measure_Standardized', 'Geo Place Name', 'Time Period', 'Age_Group']
continuous = ['Data Value']
time_series = ['Start_Date', 'Start Year']
```

- Univariate Analysis
    - Categorical

```
print("=== Name ===")
print(df['Name'].value_counts(dropna=False))

plt.figure(figsize=(8,4))
sns.countplot(data=df, y='Name', order=df['Name'].value_counts().index, palette='coolwarm')
plt.title('Count Plot of Name')
plt.xlabel('Count')
plt.ylabel('Name')
plt.tight_layout()
plt.show()
```

```
print("=== Measure ===")
print(df['Measure'].value_counts(dropna=False))

plt.figure(figsize=(6,3))
sns.countplot(data=df, y='Measure', order=df['Measure'].value_counts().index, palette='magma')
plt.title('Count Plot of Measure')
plt.xlabel('Count')
plt.ylabel('Measure')
plt.tight_layout()
plt.show()
```

```
print("=== Measure_Standardized ===")
print(df['Measure_Standardized'].value_counts(dropna=False))

plt.figure(figsize=(6,3))
sns.countplot(data=df, y='Measure_Standardized', order=df['Measure_Standardized'].value_counts().index, palette='viridis')
plt.title('Count Plot of Measure Standardized')
plt.xlabel('Count')
plt.ylabel('Measure_Standardized')
plt.tight_layout()
plt.show()
```

```python
print("=== Geo Place Name ===")
print(df['Geo Place Name'].value_counts(dropna=False))

plt.figure(figsize=(10,6))
sns.countplot(data=df, y='Geo Place Name', order=df['Geo Place Name'].value_counts().index[:30], palette='cubehelix')
plt.title('Top 30 Geo Place Name by Count')
plt.xlabel('Count')
plt.ylabel('Geo Place Name')
plt.tight_layout()
plt.show()
```

```python
print("=== Time Period ===")
print(df['Time Period'].value_counts(dropna=False))

plt.figure(figsize=(8,10))
sns.countplot(data=df, y='Time Period', order=df['Time Period'].value_counts().index[:30], palette='plasma')
plt.title('Top 30 Time Period by Count')
plt.xlabel('Count')
plt.ylabel('Time Period')
plt.tight_layout()
plt.show()
```

```python
print("=== Age_Group ===")
print(df['Age_Group'].value_counts(dropna=False))

plt.figure(figsize=(4,2))
sns.countplot(data=df, x='Age_Group', order=df['Age_Group'].value_counts().index, palette='pastel')
plt.title('Count Plot of Age Group')
plt.xlabel('Age Group')
plt.ylabel('Count')
plt.tight_layout()
plt.show()
```

- Continous

```python
print("=== Data Value Summary ===")
print(df['Data Value'].describe())

plt.figure(figsize=(10,4))

# Histogram
plt.subplot(1, 2, 1)
sns.histplot(df['Data Value'], bins=30, kde=True, color='skyblue')
plt.title('Distribution of Data Value')
plt.xlabel('Data Value')

# Boxplot
plt.subplot(1, 2, 2)
sns.boxplot(x=df['Data Value'], color='orange')
plt.title('Boxplot of Data Value')

plt.tight_layout()
plt.show()
```

- Univariate analysis for Timeseries not possible

- Bultivariate Analysis
    - Categorical vs Continous

```
print("=== Name vs Data Value ===")
plt.figure(figsize=(15, 10))
sns.boxplot(data=df, x='Name', y='Data Value')
plt.xticks(rotation=45)
plt.title('Data Value across Names')
plt.tight_layout()
plt.show()
```

```
print("=== Measure vs Data Value ===")
plt.figure(figsize=(10, 8))
sns.boxplot(data=df, x='Measure', y='Data Value')
plt.title('Data Value by Measure')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

```
print("=== Measure Info vs Data Value ===")
plt.figure(figsize=(10, 5))
sns.boxplot(data=df, x='Measure Info', y='Data Value')
plt.title('Data Value by Measure Info')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

```
print("=== Measure Standaridized vs Data Value ===")
plt.figure(figsize=(10, 8))
sns.boxplot(data=df, x='Measure_Standardized', y='Data Value')
plt.title('Data Value by Standardization')
plt.tight_layout()
plt.show()
```

```
print("=== Time Period vs Data Value ===")
plt.figure(figsize=(12, 5))
sns.boxplot(data=df, x='Time Period', y='Data Value')
plt.title('Data Value across Time Periods')
plt.xticks(rotation=90)
plt.tight_layout()
plt.show()
```

```python
print("=== Geo Place Name vs Data Value ===")
# Get top 5 Geo Place Names
top_places = df['Geo Place Name'].value_counts().head(5).index.tolist()

# Filter dataset
filtered_df = df[df['Geo Place Name'].isin(top_places)].copy()

# Force categorical order for the plot
filtered_df['Geo Place Name'] = pd.Categorical(
    filtered_df['Geo Place Name'],
    categories=top_places,
    ordered=True
)

# Plot
plt.figure(figsize=(12, 6))
sns.boxplot(data=filtered_df, x='Geo Place Name', y='Data Value')
plt.title('Data Value across Top 5 Geo Locations')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

```python
print("=== Age Group vs Data Value ===")
plt.figure(figsize=(8, 5))
sns.boxplot(data=df, x='Age_Group', y='Data Value')
plt.title('Data Value by Age Group')
plt.tight_layout()
plt.show()
```

- Categorical vs Categorical

```python
print("=== Measure vs Measure Standardized ===")
pd.crosstab(df['Measure'], df['Measure_Standardized']).plot(kind='bar', stacked=True, figsize=(12, 6))
plt.title('Measure vs Measure_Standardized')
plt.xlabel('Measure')
plt.ylabel('Count')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

```python
print("=== Age Group vs Measure ===")
pd.crosstab(df['Measure'], df['Age_Group']).plot(kind='bar', stacked=True, figsize=(10, 6))
plt.title('Measure vs Age Group')
plt.xlabel('Measure')
plt.ylabel('Count')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

```
print("=== Age Group vs Time Period ===")
plt.figure(figsize=(14, 6))
sns.countplot(data=df, x='Time Period', hue='Age_Group')
plt.title('Time Period vs Age Group')
plt.xticks(rotation=90)
plt.tight_layout()
plt.show()
```

```
print("=== Age Group vs Measure ===")
cross = pd.crosstab(df['Measure'], df['Age_Group'])

plt.figure(figsize=(8, 5))
sns.heatmap(cross, annot=True, fmt='d', cmap='YlGnBu')
plt.title('Heatmap of Measure vs Age Group')
plt.tight_layout()
plt.show()
```

- Multivariate Analysis

```
pivot = df.pivot_table(
    values='Data Value',
    index='Geo Place Name',
    columns='Measure',
    aggfunc='mean'
)

plt.figure(figsize=(12, 8))
sns.heatmap(pivot.head(10), annot=True, cmap='coolwarm', fmt=".1f")
plt.title('Average Data Value by Measure and Geo Location (Top 10 Locations)')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

-

```
plt.figure(figsize=(12, 6))
sns.lineplot(data=df, x='Start Year', y='Data Value', hue='Measure', estimator='mean')
plt.title('Trend of Average Data Value by Measure Over Years')
plt.tight_layout()
plt.show()
```

```python
plt.figure(figsize=(14, 6))
sns.boxplot(data=df, x='Age_Group', y='Data Value', hue='Measure')
plt.title('Distribution of Data Value across Age Groups and Measures')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

6. Data Visualization

Data Analysis (Business Problem based)

1. Pollution Levels & Sources

Question: What are the average, minimum, and maximum concentrations of pollutants across different districts? How do vehicle emissions and boiler usage contribute?

```python
# Pollution levels by district (Geo Place Name) and pollutant (Name)
pollution_stats = df.groupby(['Geo Place Name', 'Name'])['Data Value'].agg(['mean', 'min', 'max']).reset_index()

# Emission sources - vehicle miles and boiler emissions
vehicle_boiler = df[df['Name'].isin(['Annual Vehicle Miles Traveled', 'Annual Vehicle Miles Traveled (Trucks)', 'Annual Vehicle Miles Traveled (Cars)',
                              'Boiler Emissions - Total Nox Emissions', 'Boiler Emissions - Total So2 Emissions', 'Boiler Emissions - Total PM2.5 E

emissions_summary = vehicle_boiler.groupby(['Geo Place Name', 'Name'])['Data Value'].mean().reset_index()

print(pollution_stats.head(10))
print(emissions_summary.head(10))
```

2. Temporal Trends

Question: How do air quality and health outcomes vary by season and over the years? Are there improvements or worsening trends?

```python
# Average Data Value by Time Period and Year for key pollutants & health measures
temporal_trends = df.groupby(['Start Year', 'Time Period', 'Name'])['Data Value'].mean().reset_index()

# Plot line plot for selected pollutants over years
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(14,7))
selected_measures = ['Nitrogen Dioxide (NO2)', 'Fine Particles (PM2.5)', 'Ozone (O3)']
sns.lineplot(data=temporal_trends[temporal_trends['Name'].isin(selected_measures)],
             x='Start Year', y='Data Value', hue='Name')
plt.title('Pollutant Trends Over Years')
plt.show()
```

3. Geographic Insights

Question: Which areas are most affected by pollution and health risks? How does population density relate to air quality?

```python
# Average pollutant and health impact by Geo Place Name
geo_impact = df.groupby(['Geo Place Name', 'Name'])['Data Value'].mean().reset_index()

# Top 10 areas with highest average NO2 pollution
top_no2_areas = geo_impact[(geo_impact['Name']=='Nitrogen Dioxide (NO2)')].sort_values('Data Value', ascending=False).head(10)

print(top_no2_areas)
```

4. Health Impact Assessment

Question: What is the relationship between pollution levels and hospitalization/emergency rates?

```
# Select pollution and health impact measures
pollution_measures = ['Nitrogen Dioxide (NO2)', 'Fine Particles (PM2.5)', 'Ozone (O3)']
health_measures = ['Asthma Emergency Department Visits Due To PM2.5', 'Asthma Emergency Departments Visits Due To Ozone',
                   'Cardiovascular Hospitalizations Due To PM2.5 (Age 40+)', 'Respiratory Hospitalizations Due To PM2.5 (Age 20+)', 'Deaths Due To PM2.5'

# Create pivot tables per Geo Place Name and Start Year for pollution and health
pollution_pivot = df[df['Name'].isin(pollution_measures)].pivot_table(index=['Geo Place Name', 'Start Year'], columns='Name', values='Data Value', aggfun
health_pivot = df[df['Name'].isin(health_measures)].pivot_table(index=['Geo Place Name', 'Start Year'], columns='Name', values='Data Value', aggfunc='mea

# Merge pollution and health data on Geo Place Name and Year
merged_df = pollution_pivot.merge(health_pivot, on=['Geo Place Name', 'Start Year'])

# Calculate correlation matrix
corr_matrix = merged_df[pollution_measures + health_measures].corr()

print(corr_matrix)
```

5. Demographic Patterns

Question: How do different age groups experience health impacts in relation to air quality?

```
# Average Data Value by Age Group and Measure for health-related data
health_data = df[df['Name'].str.contains('Asthma|Cardiovascular|Respiratory|Deaths', case=False)]

age_group_impact = health_data.groupby(['Age_Group', 'Name'])['Data Value'].mean().reset_index()

import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(14,7))
sns.barplot(data=age_group_impact, x='Age_Group', y='Data Value', hue='Name')
plt.title('Health Impact by Age Group')
plt.xticks(rotation=45)
plt.show()
```

6. Outlier Detection & Data Quality

Question: Are there extreme values? How were they handled?

```
# Summary stats to identify outliers
print(df['Data Value'].describe())

# Plot boxplot for Data Value to visually inspect outliers
plt.figure(figsize=(10, 5))
sns.boxplot(x=df['Data Value'])
plt.title('Boxplot for Data Value')
plt.show()

# Capping example: cap Data Value at 99th percentile
cap_value = df['Data Value'].quantile(0.99)
df['Data Value Capped'] = df['Data Value'].clip(upper=cap_value)
```

6. Measure Standardization

Question: How were units standardized?

```
# Check unique Measure Info values before and after standardization
print("Before standardization:\n", df['Measure Info'].value_counts())

# Assuming you used the earlier function standardize_measure()
# Reapply or verify
df['Measure_Standardized'] = df['Measure Info'].apply(standardize_measure)

print("\nAfter standardization:\n", df['Measure_Standardized'].value_counts())
```