```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

1) Read the datasets

```
file_path = "/content/chicago_violations (1).csv"
df_chicago = pd.read_csv(file_path)
df_chicago.head()
```

	ID	DOCKET NUMBER	NOV NUMBER	ADDRESS	STREET NUMBER	STREET DIRECTION	STREET NAME	STREET TYPE	WARD	ISS DEPART
0	9ed845888ab5e3bc6516adebba3319346b173012	23BT00101A	22P0675219	5006 W GUNNISON ST	5006	W	GUNNISON	ST	45.0	Bu
1	b6df42910398cd4144f06d4ae36a25681848146b	15BT03446A	15WO442772	3241 W 62ND ST	3241	W	62ND	ST	23.0	Bu
2	de16ff67b156537113e2e7ecb1d18044e268aa3a	22BT02866A	22T0668607	3811 W 61ST ST	3811	W	61ST	ST	23.0	Bu
3	ff933be74d08113f113a3ba84fa0f2652a017991	21BT02435A	21V0652277	4259 W WILCOX ST	4259	W	WILCOX	ST	28.0	Bu
4	fecf777cc540db1cf9abd2e72b0d839868dd2515	22BT03788A	22SH0674660	1614 W 66TH ST	1614	W	66TH	ST	15.0	Bu

5 rows × 22 columns

(2)

a) Describing datasets

```
df_chicago.describe()
print("Dataset Description:\n", df_chicago.describe())
```

```
Dataset Description:
        STREET NUMBER           WARD   IMPOSED FINE    ADMIN COSTS  \
count  784225.000000  781415.000000  784225.000000  784225.000000
mean     4276.940058      22.052917    1004.903041      32.041845
std      2938.933170      13.369680    2758.419584      35.662265
min         0.000000       0.000000       0.000000       0.000000
25%      1730.000000      10.000000       0.000000       0.000000
50%      3909.000000      21.000000       0.000000       0.000000
75%      6436.000000      32.000000     600.000000      75.000000
max    107039.000000      50.000000   50000.000000    1000.000000

            LATITUDE      LONGITUDE
count  783289.000000  783289.000000
mean       41.837560     -87.669924
std         0.087028       0.058375
min        41.644702     -87.914428
25%        41.764455     -87.713152
50%        41.835178     -87.668120
75%        41.907913     -87.627978
max        42.022686     -87.524679
```

(2)

b) Number of rows and columns

```
df_chicago.shape
```

```
(784225, 22)
```

```
num_rows, num_columns = df_chicago.shape
print("Number of rows:", num_rows)
print("Number of columns:", num_columns)
```

```
Number of rows: 784225
Number of columns: 22
```

(3)

Choosing a random ward and filtering it

For this question, I am choosing WARD number 4.

```
ward_4_data = df_chicago[df_chicago['WARD'] == 4]
print(f"Filtered dataset for Ward number 4 contains {ward_4_data.shape[0]} rows.")
ward_4_data.head()
```

```
Filtered dataset for Ward number 4 contains 10105 rows.
```

	ID	DOCKET NUMBER	NOV NUMBER	ADDRESS	STREET NUMBER	STREET DIRECTION	STREET NAME	STREET TYPE	WARD	ISSUIN DEPARTMEN
9	600b156bb1301f36bf762b1621080c3610a881eb	22BT03343A	22P0671605	4520 S DREXEL BLVD	4520	S	DREXEL	BLVD	4.0	Building
91	f598a94d2860a1dccbfbc162afafaee5f68ce03c	22BT03343A	22P0671605	4520 S DREXEL BLVD	4520	S	DREXEL	BLVD	4.0	Building
311	e67263c5da885445d306ee6107dd5dc87c2cf5da	22BT01640A	22CO661930	722 E BOWEN AVE	722	E	BOWEN	AVE	4.0	Building
320	7114943bb13bf43a1e3a6629d7662acf90df30c3	22BT01640A	22CO661930	722 E BOWEN AVE	722	E	BOWEN	AVE	4.0	Building
342	eb659d2acd79fd082ca3829a334e3b48ff649f28	22BT01640A	22CO661930	722 E BOWEN AVE	722	E	BOWEN	AVE	4.0	Building

5 rows × 22 columns

```
num_records = ward_4_data.shape[0]
print(f"The filtered dataframe for Ward number 4 contains {num_records} records.")
```

```
The filtered dataframe for Ward number 4 contains 10105 records.
```

(3)

a) The filtered dataframe for Ward number 4 contains 10105 records.

(3)

b) I read over the internet a few interestig points about Ward 4:

- The 4th Ward is home to the site of the Douglas Monument and the original location of the old Chicago University

- The 4th Ward boasts a racially diverse population, with approximately 46.0% identifying as Black or African American, 30.2% as White, 13.3% as Asian, and 6.4% as Hispanic or Latino
- Toni Preckwinkle, who served as the 4th Ward Alderman for 19 years, was instrumental in the redevelopment of neighborhoods such as Kenwood, Oakland, Douglas, Grand Boulevard, and Hyde Park

---

```
total_na = ward_4_data.isna().sum().sum()
has_na = total_na > 0

print(f"Are there any NA values in the dataframe? {'Yes' if has_na else 'No'}")
print(f"Total number of NA values in the dataframe: {total_na}")
```

```
Are there any NA values in the dataframe? Yes
Total number of NA values in the dataframe: 278
```

(4)

a) Yes, there Total 278 NA values in the dataframe

```
num_complete_cases = ward_4_data.dropna().shape[0]
total_rows = ward_4_data.shape[0]
percentage_complete_cases = (num_complete_cases / total_rows) * 100


print(f"Number of complete cases: {num_complete_cases}")
print(f"Total rows: {total_rows}")
print(f"Percentage of complete cases: {percentage_complete_cases:.2f}%")
```

```
Number of complete cases: 9835
Total rows: 10105
Percentage of complete cases: 97.33%
```

(4)

b) A complete case is a row in the dataframe that has no missing or NA values in any of its columns. Here, we have got 97.33% of complete cases.

```
num_blank_cells = (ward_4_data == '').sum().sum()
print(f"Total number of blank cells in the dataframe: {num_blank_cells}")
```

```
Total number of blank cells in the dataframe: 0
```

(4)

c) In our filtered dataset, there are no blank cells as we checked. Hence will be proceeding ahead without converting any blank cells into NAs.

```
missing_table = ward_4_data.isna().sum().reset_index()
missing_table.columns = ['Variable', 'Number of Missing Values']
missing_table['Percentage of Missing Values'] = (missing_table['Number of Missing Values'] / ward_4_data.shape[0]) * 100

print("Missing Values Table:")
print(missing_table)
```

```
Missing Values Table:
               Variable  Number of Missing Values  \
0                    ID                         0
1         DOCKET NUMBER                         0
2            NOV NUMBER                         0
3               ADDRESS                         0
4         STREET NUMBER                         0
5      STREET DIRECTION                         0
6           STREET NAME                         0
7           STREET TYPE                        31
8                  WARD                         0
9    ISSUING DEPARTMENT                         0
10         HEARING DATE                         0
11     CASE DISPOSITION                       235
12         IMPOSED FINE                         0
13          ADMIN COSTS                         0
14   LAST MODIFIED DATE                         0
15       VIOLATION DATE                         0
16       VIOLATION CODE                         0
```

```
17  VIOLATION DESCRIPTION                         0
18            RESPONDENTS                         0
19               LATITUDE                         4
20              LONGITUDE                         4
21               LOCATION                         4

    Percentage of Missing Values
0                       0.000000
1                       0.000000
2                       0.000000
3                       0.000000
4                       0.000000
5                       0.000000
6                       0.000000
7                       0.306779
8                       0.000000
9                       0.000000
10                      0.000000
11                      2.325581
12                      0.000000
13                      0.000000
14                      0.000000
15                      0.000000
16                      0.000000
17                      0.000000
18                      0.000000
19                      0.039584
20                      0.039584
21                      0.039584
```

(4)

(d) Above, generated a table that shows the number of missing values and the percentage of missing values for each variable. Please refer the index for matching the variables in each table

---

(5)

Handling dates

```
print("Data types of all columns:")
print(ward_4_data.dtypes)

potential_date_columns = [col for col in ward_4_data.columns if 'date' in col.lower()]

print("\nColumns that potentially contain date values:")
print(potential_date_columns)
```

```
Data types of all columns:
ID                        object
DOCKET NUMBER             object
NOV NUMBER                object
ADDRESS                   object
STREET NUMBER              int64
STREET DIRECTION          object
STREET NAME               object
STREET TYPE               object
WARD                     float64
ISSUING DEPARTMENT        object
HEARING DATE              object
CASE DISPOSITION          object
IMPOSED FINE             float64
ADMIN COSTS                int64
LAST MODIFIED DATE        object
VIOLATION DATE            object
VIOLATION CODE            object
VIOLATION DESCRIPTION     object
RESPONDENTS               object
LATITUDE                 float64
LONGITUDE                float64
LOCATION                  object
dtype: object

Columns that potentially contain date values:
['HEARING DATE', 'LAST MODIFIED DATE', 'VIOLATION DATE']
```

(5)

a) Columns that potentially contain date values are HEARING DATE', 'LAST MODIFIED DATE', 'VIOLATION DATE'.

```
ward_4_data['HEARING DATE'] = pd.to_datetime(ward_4_data['HEARING DATE'], errors='coerce')
ward_4_data['VIOLATION DATE'] = pd.to_datetime(ward_4_data['VIOLATION DATE'], errors='coerce')
```

```
<ipython-input-54-218082060443>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view
  ward_4_data['HEARING DATE'] = pd.to_datetime(ward_4_data['HEARING DATE'], errors='coerce')
<ipython-input-54-218082060443>:2: UserWarning: Could not infer format, so each element will be parsed individually, falling
  ward_4_data['VIOLATION DATE'] = pd.to_datetime(ward_4_data['VIOLATION DATE'], errors='coerce')
<ipython-input-54-218082060443>:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view
  ward_4_data['VIOLATION DATE'] = pd.to_datetime(ward_4_data['VIOLATION DATE'], errors='coerce')
```

Here, we first Converted 'HEARING DATE' and 'VIOLATION DATE' to datetime format

```
ward_4_data['CityDelay'] = (ward_4_data['HEARING DATE'] - ward_4_data['VIOLATION DATE']).dt.days

print("CityDelay column added to the dataframe:")
ward_4_data[['HEARING DATE', 'VIOLATION DATE', 'CityDelay']].head()
```

```
CityDelay column added to the dataframe:
<ipython-input-55-c64aa65c8325>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view
  ward_4_data['CityDelay'] = (ward_4_data['HEARING DATE'] - ward_4_data['VIOLATION DATE']).dt.days
```

	HEARING DATE	VIOLATION DATE	CityDelay
9	2023-02-15	2022-10-12 00:00:00	126
91	2023-02-15	2022-10-12 00:00:00	126
311	2023-02-21	2022-05-05 09:00:00	291
320	2023-02-21	2022-05-05 09:00:00	291
342	2023-02-21	2022-05-05 09:00:00	291

(5)

b) Added a new variable to the dataframe called CityDelay. CityDelay is created based on the difference between the HearingDate and ViolationDate.

(5)

c) My birthday falls on 9th of October

```
oct_9_violations = ward_4_data[ward_4_data['VIOLATION DATE'].dt.month == 10]
oct_9_violations = oct_9_violations[oct_9_violations['VIOLATION DATE'].dt.day == 9]

num_violations = oct_9_violations.shape[0]
print(f"Number of violations on October 9th: {num_violations}")
```

```
Number of violations on October 9th: 22
```

```
most_common_disposition = oct_9_violations['CASE DISPOSITION'].mode()[0]
print(f"Most common Case Disposition for those ordinance violations: {most_common_disposition}")
```

```
Most common Case Disposition for those ordinance violations: Liable
```

(5)

c) Total number of violation on 9th of October were 22. And most common deposition for those ordinance violations was "Liable"

---

(6)

a) Ward should be considered a categorical variable because:

- Definition of Wards:

Wards represent specific geographical regions or divisions within a city (in this case, Chicago). Each ward is essentially a label for a region and does not represent a measurable quantity.

- Lack of Mathematical Meaning:

The numeric values of wards (e.g., 1, 2, 4, etc.) are just identifiers. Arithmetic operations such as addition, subtraction, or averaging do not have any logical or meaningful interpretation for these values.

- Categorical Nature:

Wards define distinct categories or groups, making them inherently categorical. They are more appropriate for analysis using counts, frequencies, or comparisons between categories rather than mathematical computations.

```
correlation = ward_4_data['IMPOSED FINE'].corr(ward_4_data['ADMIN COSTS'])
print(f"Correlation between Imposed Fine and Admin Costs: {correlation:.2f}")
```

```
Correlation between Imposed Fine and Admin Costs: 0.28
```

(6)

b) The correlation value between Imposed Fine and Admin Costs is 0.28, which indicates a weak positive correlation. Since the correlation is close to 0 and less than 0.3, the strength of the relationship is weak. This suggests that other factors might influence the admin costs apart from the imposed fine.

```
most_common_street_type = ward_4_data['STREET TYPE'].mode()[0]
street_type_count = ward_4_data['STREET TYPE'].value_counts()[most_common_street_type]

print(f"The most common street type in Ward 4 is: {most_common_street_type}")
print(f"It appears {street_type_count} times in the dataset.")
```

```
The most common street type in Ward 4 is: AVE
It appears 4968 times in the dataset.
```

(6)

c) The most common street type in Ward 4 is AVE as it appears 4968 times in the dataset. It is not the same street where I live, as I live at the Washington Street.

```
unique_violation_descriptions = ward_4_data['VIOLATION DESCRIPTION'].nunique()
unique_violation_codes = ward_4_data['VIOLATION CODE'].nunique()

print(f"Number of unique Violation Description values: {unique_violation_descriptions}")
print(f"Number of unique Violation Code values: {unique_violation_codes}")
```

```
Number of unique Violation Description values: 475
Number of unique Violation Code values: 475
```

(6)

d) There are 475 unique Violation Description values and 475 unique Violation Code values

---

```
ward_4_data['HEARING DATE'] = pd.to_datetime(ward_4_data['HEARING DATE'], errors='coerce')
ward_4_data['HEARING YEAR'] = ward_4_data['HEARING DATE'].dt.year

yearly_avg_fine = ward_4_data.groupby('HEARING YEAR')['IMPOSED FINE'].mean().reset_index()
yearly_avg_fine.columns = ['Year', 'Average Imposed Fine']

print("Average Imposed Fine by Year:")
print(yearly_avg_fine)
```

```
Average Imposed Fine by Year:
   Year  Average Imposed Fine
0  2008           1256.662804
```

```
1   2009            707.455429
2   2010            615.069284
3   2011            571.921922
4   2012            611.607143
5   2013            464.125561
6   2014            496.466431
7   2015           1397.936210
8   2016            593.055556
9   2017            392.031524
10  2018            331.803279
11  2019            515.283843
12  2020            319.083969
13  2021            429.975430
14  2022            855.268390
15  2023            593.493151
16  2024            555.555556
<ipython-input-61-3e8fa988a802>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view
  ward_4_data['HEARING DATE'] = pd.to_datetime(ward_4_data['HEARING DATE'], errors='coerce')
<ipython-input-61-3e8fa988a802>:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view
  ward_4_data['HEARING YEAR'] = ward_4_data['HEARING DATE'].dt.year
```

(7)

SHowing the averages by each year

```
data_2024 = ward_4_data[ward_4_data['HEARING YEAR'] == 2024]
print(f"\nNumber of records for 2024: {data_2024.shape[0]}")
```

```

Number of records for 2024: 171
```

```
print("Minimum and Maximum Hearing Dates:")
print(ward_4_data['HEARING DATE'].min())
print(ward_4_data['HEARING DATE'].max())
```

```
Minimum and Maximum Hearing Dates:
2008-01-16 00:00:00
2024-05-16 00:00:00
```

(7)

a)

Here, as we can see that the maximum records we have are until May 2024. As we do not have records for entire 2024, the Average fine is less compared to other years' records.

```
ward_4_data = ward_4_data.drop(columns=['ID', 'DOCKET NUMBER'])
print("Columns removed successfully. Updated dataframe columns:")
print(ward_4_data.columns)
```

```
Columns removed successfully. Updated dataframe columns:
Index(['NOV NUMBER', 'ADDRESS', 'STREET NUMBER', 'STREET DIRECTION',
       'STREET NAME', 'STREET TYPE', 'WARD', 'ISSUING DEPARTMENT',
       'HEARING DATE', 'CASE DISPOSITION', 'IMPOSED FINE', 'ADMIN COSTS',
       'LAST MODIFIED DATE', 'VIOLATION DATE', 'VIOLATION CODE',
       'VIOLATION DESCRIPTION', 'RESPONDENTS', 'LATITUDE', 'LONGITUDE',
       'LOCATION', 'CityDelay', 'HEARING YEAR'],
      dtype='object')
```

(8)

Removed the ID and DOCKET NUMBER columns

```
ward_4_data['VIOLATION DATE'] = pd.to_datetime(ward_4_data['VIOLATION DATE'], errors='coerce')


quarter_to_season = {1: 'Winter', 2: 'Spring', 3: 'Summer', 4: 'Fall'}
ward_4_data['Season'] = ward_4_data['VIOLATION DATE'].dt.quarter.map(quarter_to_season)
print("Season column added successfully:")
ward_4_data[['VIOLATION DATE', 'Season']].head()
```

```
Season column added successfully:
```

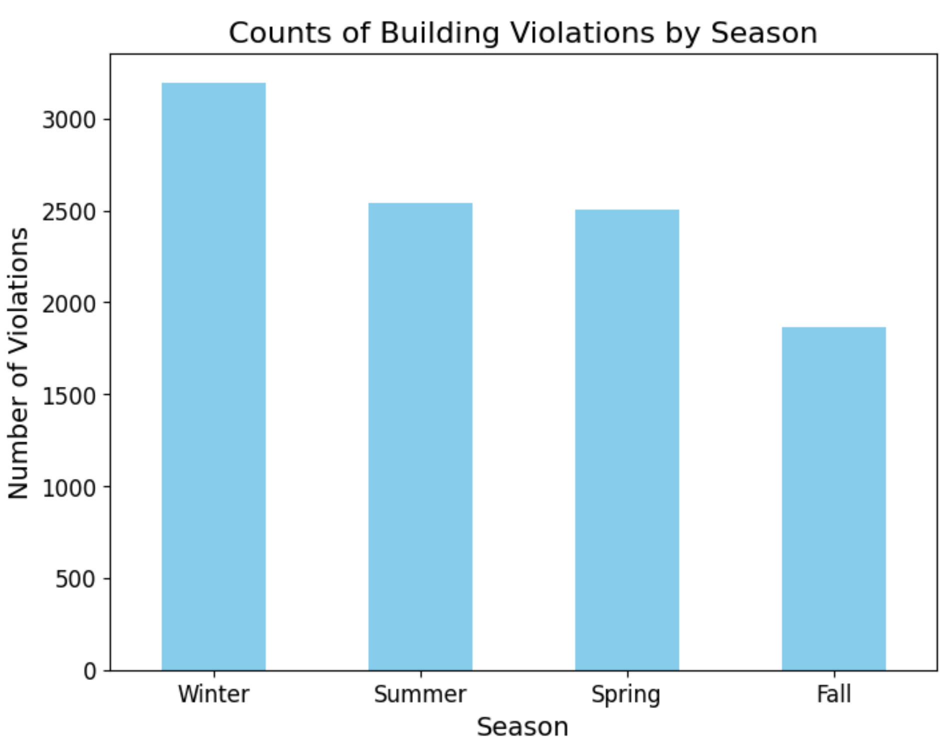
	VIOLATION DATE	Season
9	2022-10-12 00:00:00	Fall
91	2022-10-12 00:00:00	Fall
311	2022-05-05 09:00:00	Spring
320	2022-05-05 09:00:00	Spring
342	2022-05-05 09:00:00	Spring

(9)

Season column added and renamed the quarters as per the required seasons. I called the head of the dataset to check the change made.

```
season_counts = ward_4_data['Season'].value_counts()
plt.figure(figsize=(8, 6))
season_counts.plot(kind='bar', color='skyblue')
plt.title('Counts of Building Violations by Season', fontsize=16)
plt.xlabel('Season', fontsize=14)
plt.ylabel('Number of Violations', fontsize=14)
plt.xticks(rotation=0, fontsize=12)
plt.yticks(fontsize=12)

plt.show()
```



(10)

a)

1. Winter (Highest Violations):

- I think Harsh weather during Winter might lead to more structural issues or safety violations being noticed or reported.
- Also there could be increased inspections during Winter, possibly due to safety concerns during colder months.

2. Fall (Lowest Violations):

- I guess Fall might see fewer construction or maintenance activities, resulting in fewer violations.
- Many properties might already be repaired or inspected earlier, reducing violations.

3. Spring and Summer (Moderate Counts):

- these seasons are often associated with construction and maintenance activities, which could lead to moderate numbers of violations being reported.
- Also Municipalities might conduct regular inspections during these active months, leading to consistent counts.

---

(11)

Filtering Dataset again

```
top_5_dispositions = ward_4_data['CASE DISPOSITION'].value_counts().nlargest(5).index
filtered_data = ward_4_data[ward_4_data['CASE DISPOSITION'].isin(top_5_dispositions)]

print(f"The filtered dataset contains {filtered_data.shape[0]} rows.")
print("Top 5 Case Dispositions:")
print(filtered_data['CASE DISPOSITION'].value_counts())
```

```
The filtered dataset contains 9837 rows.
Top 5 Case Dispositions:
CASE DISPOSITION
Non-Suit       3874
Liable         3104
Not Liable     1352
Default        1016
Continuance     491
Name: count, dtype: int64
```

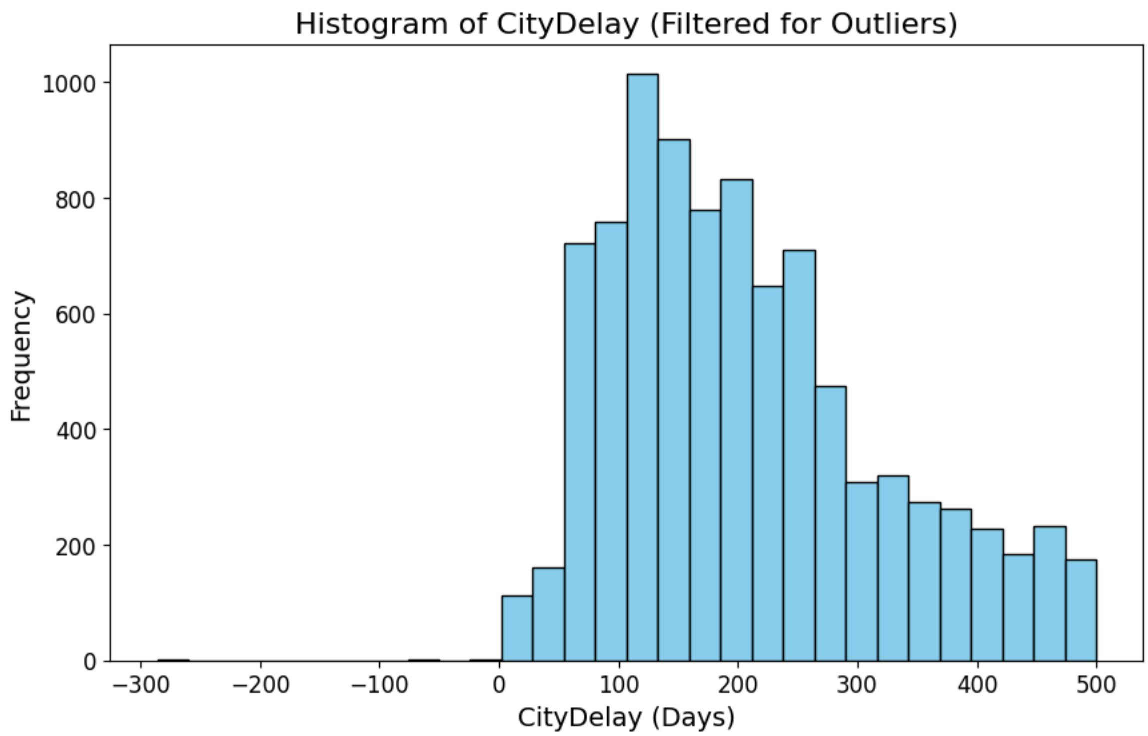
(11)

a) Now the filtered dataset contains 9837 rows.

```
filtered_data_no_outliers = filtered_data[filtered_data['CityDelay'] <= 500]

plt.figure(figsize=(10, 6))
plt.hist(filtered_data_no_outliers['CityDelay'], bins=30, color='skyblue', edgecolor='black')
plt.title('Histogram of CityDelay (Filtered for Outliers)', fontsize=16)
plt.xlabel('CityDelay (Days)', fontsize=14)
plt.ylabel('Frequency', fontsize=14)
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)

plt.show()
```



(11)

b) As seen above, I excluded outliers by capping CityDelay values (e.g., consider delays less than 500 days). Then generated the histograms depicting CityDelays.

(11)

c) Observations:

1. Peak Around 100 Days:

- The most frequent CityDelay is around 100 days. This could indicate a common administrative processing time for scheduling hearings after a violation occurs.

2. Gradual Decline Beyond 100 Days:

- The frequency decreases steadily for delays longer than 100 days.
- Longer delays might occur due to rescheduling, case complexity, or backlogs in the system.

3. Small Negative CityDelay Values:

- There are a few negative values for CityDelay. This might mean data entry errors or situations where the hearing date was recorded incorrectly as occurring before the violation date.

4. Right-Skewed Tail (Delays > 250 Days):

- Some violations experience delays up to 500 days, though these are less common. This could result from rare cases involving legal complexities, appeals, or rescheduling issues.

---

(12)

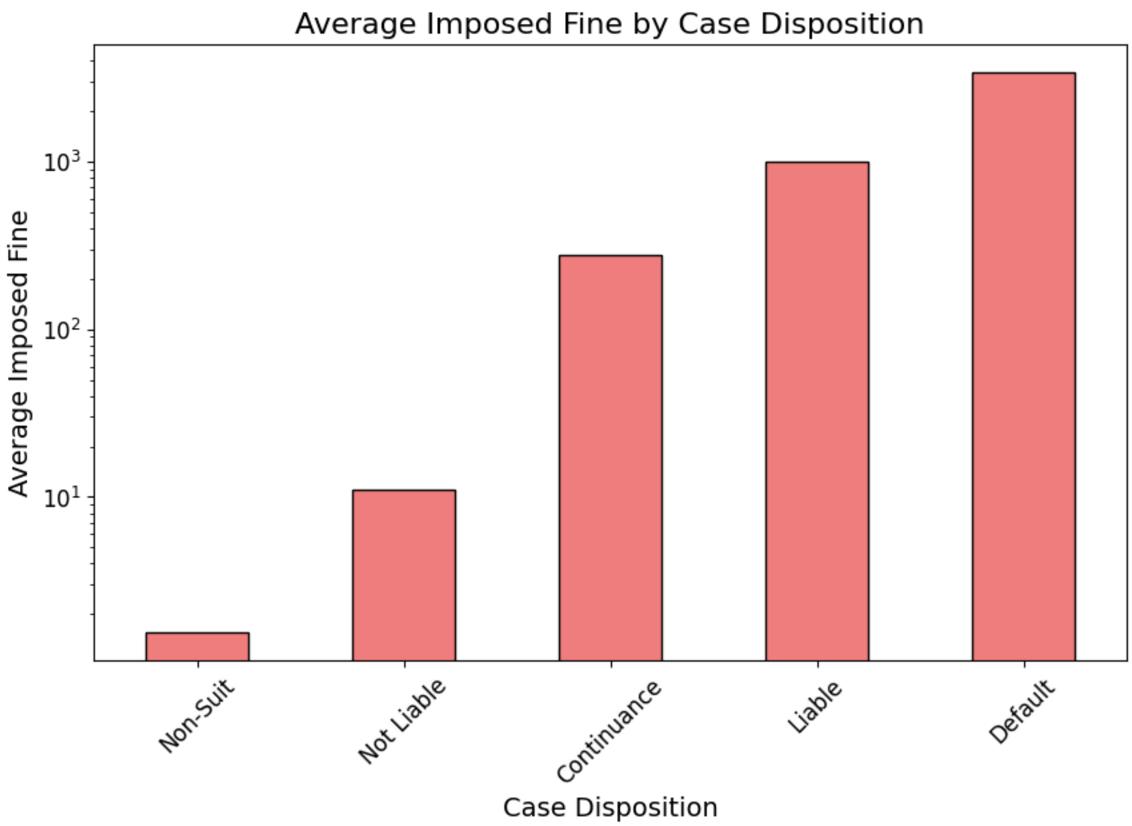
For this, first we will Calculate the average imposed fine for each Case Disposition

```
avg_fine_per_disposition = filtered_data.groupby('CASE DISPOSITION')['IMPOSED FINE'].mean().sort_values()
```

Now making the plot

```
plt.figure(figsize=(10, 6))
avg_fine_per_disposition.plot(kind='bar', color='lightcoral', edgecolor='black')
```

```
plt.title('Average Imposed Fine by Case Disposition', fontsize=16)
plt.xlabel('Case Disposition', fontsize=14)
plt.ylabel('Average Imposed Fine', fontsize=14)
plt.xticks(rotation=45, fontsize=12)
plt.yticks(fontsize=12)
plt.yscale('log')
plt.show()
```



(12)

(a)

Observation and explanation:-

- Default Cases:

Cases with a "Default" disposition have the highest average imposed fine, significantly exceeding other case dispositions.

A "Default" disposition likely occurs when the defendant fails to respond or appear in court, leading to the imposition of the highest fines as a penalty for non-compliance. This could reflect stricter consequences for not participating in the legal process.

- Liable Cases:

Cases with a "Liable" disposition show a moderate average fine, much lower than "Default" but higher than other categories.

Cases where the defendant is found "Liable" incur fines as part of the judgment, but these fines are generally less severe than the penalties for "Default." This may represent the standard fine amount when a violation is confirmed.

- Non-Suit and "Not Liable" Cases:

These categories have almost negligible or very low average fines compared to "Default" and "Liable."

"Non-Suit" cases may be dismissed or withdrawn by the plaintiff, resulting in no fines. "Not Liable" cases indicate the defendant was not found responsible, which also results in no fines.

- Continuance Cases:

These cases may involve postponements, and the fines might be delayed, reduced, or not imposed until the case is resolved.

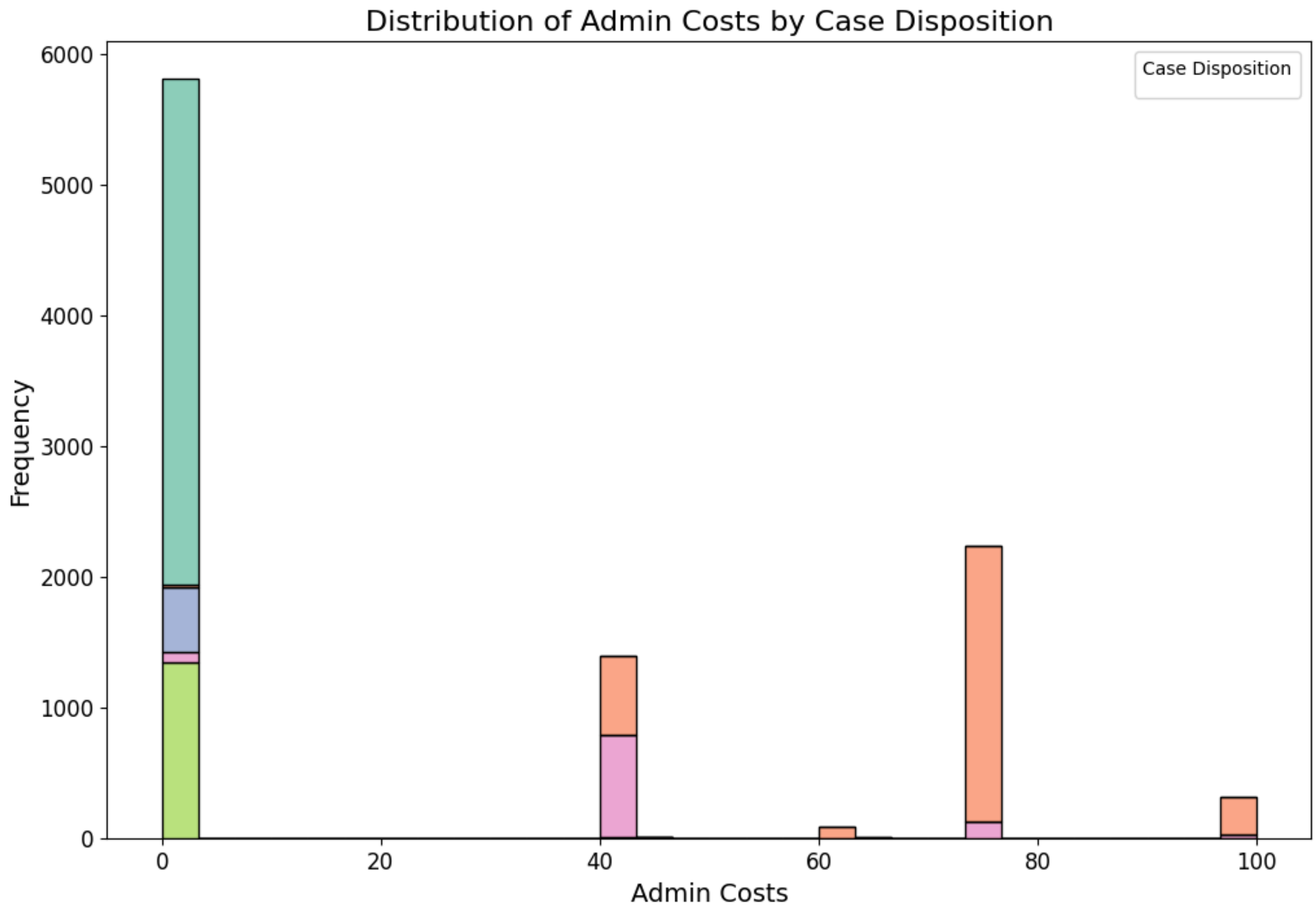
---

(13)

```
plt.figure(figsize=(12, 8))
sns.histplot(
    data=filtered_data,
    x='ADMIN COSTS',
    hue='CASE DISPOSITION',
    bins=30,
    multiple='stack',
    palette='Set2',
    edgecolor='black'
)

plt.title('Distribution of Admin Costs by Case Disposition', fontsize=16)
plt.xlabel('Admin Costs', fontsize=14)
plt.ylabel('Frequency', fontsize=14)
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)
plt.legend(title='Case Disposition', fontsize=12)
plt.show()
```

```
<ipython-input-87-6350654a5f4b>:17: UserWarning: No artists with labels found to put in legend.  Note that artists whose lab
  plt.legend(title='Case Disposition', fontsize=12)
```



(13)

a)

Observation and Explanation of my plot above:

- Dominance of Low Admin Costs:

The tallest bar near 0 indicates that the majority of cases incur minimal or no administrative costs. This could reflect cases that are dismissed, resolved quickly, or involve minor violations requiring little processing.

- Significant Peaks at Higher Costs:

Smaller but noticeable peaks occur around 80-100 admin costs. These likely represent specific case types (e.g., "Default" or "Liable") where fixed administrative fees are imposed as part of standard penalties.

- Variation Across Case Dispositions:

The stacked bars show how different Case Dispositions contribute to admin costs. For example, dispositions like "Default" might account for a larger share of higher costs, while "Non-Suit" or "Not Liable" dominate near-zero costs.

---

(14)

Filtering the dataset again

```
top_5_violations = filtered_data['VIOLATION DESCRIPTION'].value_counts().nlargest(5).index

filtered_violations = filtered_data[filtered_data['VIOLATION DESCRIPTION'].isin(top_5_violations)]

unique_violation_descriptions = filtered_violations['VIOLATION DESCRIPTION'].unique()

print(f"5 Most Common Violation Descriptions: {list(top_5_violations)}")
print(f"Unique Violation Descriptions in the filtered dataset: {list(unique_violation_descriptions)}")
```

```
5 Most Common Violation Descriptions: ['Arrange for inspection of premises. (13-12-100)', 'Failed to complete or to submit r
Unique Violation Descriptions in the filtered dataset: ['Repair or replace defective or missing members of porch system. (13
```

- AFter filtering, the 5 Most Common Violation Descriptions we get are: '

'Arrange for inspection of premises. (13-12-100)',

'Failed to complete or to submit required documentation for mandated inspection of a conveyance device in a building located within the Central Business District. (13-8-030, 18-30-017, 18-30-460, Rules and Regulations for Annual Inspection Certification of Conveyance Devices 1 through 70)',

'241001: Operating or cause to be operating an elevator, moving walk, material lift, stairway chairlift, vertical reciprocating reciprocating conveyor, moveable stage, moveable orchestra floor, platform lift or escalator without a current certificate of compliance posted in the elevator or filed on the premises for other types of conveyances (13-20-110(a) and 18-30-015).',

'Repair exterior wall. (13-196-010, 13-196-530 B)',

'Repair or replace defective or missing members of porch system. (13-196-570)'.

- Unique Violation Descriptions in the filtered dataset:

'Repair or replace defective or missing members of porch system. (13-196-570)',

'241001: Operating or cause to be operating an elevator, moving walk, material lift, stairway chairlift, vertical reciprocating reciprocating conveyor, moveable stage, moveable orchestra floor, platform lift or escalator without a current certificate of compliance posted in the elevator or filed on the premises for other types of conveyances (13-20-110(a) and 18-30-015).',

'Repair exterior wall. (13-196-010, 13-196-530 B)', 'Arrange for inspection of premises. (13-12-100)',

'Failed to complete or to submit required documentation for mandated inspection of a conveyance device in a building located within the Central Business District. (13-8-030, 18-30-017, 18-30-460,

Rules and Regulations for Annual Inspection Certification of Conveyance Devices 1 through 70)'

```
short_labels = {
    'Arrange for inspection of premises. (13-12-100)': 'Inspection',

    'Failed to complete or to submit required documentation for mandated inspection of a conveyance device in a building locatec

    '241001: Operating or cause to be operating an elevator, moving walk, material lift, stairway chairlift, vertical reciprocat

    'Repair exterior wall. (13-196-010, 13-196-530 B)': 'Exterior Wall Repair',

    'Repair or replace defective or missing members of porch system. (13-196-570)': 'Porch Repair'
}


def normalize_text(text):
    return " ".join(text.split()).strip()

short_labels_normalized = {normalize_text(key): value for key, value in short_labels.items()}
```

```
filtered_violations['VIOLATION DESCRIPTION'] = (
    filtered_violations['VIOLATION DESCRIPTION']
    .apply(normalize_text)
    .replace(short_labels_normalized)
)

print("Shortened Violation Descriptions:")
print(filtered_violations['VIOLATION DESCRIPTION'].unique())
```

```
Shortened Violation Descriptions:
['Porch Repair' 'Elevator Compliance' 'Exterior Wall Repair' 'Inspection'
 'Documentation']
<ipython-input-79-e74a76a66548>:19: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view
  filtered_violations['VIOLATION DESCRIPTION'] = (
```

As seen above, we have shortened the descriptions and made them:

- 'Porch Repair'
- 'Elevator Compliance'
- 'Exterior Wall Repair'
- 'Inspection'
- 'Documentation'

---

(15)

```
mean_fines = filtered_violations.groupby('VIOLATION DESCRIPTION')['IMPOSED FINE'].mean().sort_values()

plt.figure(figsize=(10, 6))
mean_fines.plot(kind='bar', color='cornflowerblue', edgecolor='black')

plt.title('Mean Imposed Fine by Violation Description', fontsize=16)
plt.xlabel('Violation Description', fontsize=14)
plt.ylabel('Mean Imposed Fine', fontsize=14)
plt.xticks(rotation=45, fontsize=12)
plt.yticks(fontsize=12)
plt.show()
```

Speculation on Fine Differences:

- Severity of the Violation:

Porch Repair and Exterior Wall Repair involve structural safety, which might directly impact the well-being of residents or the public. Fines for such violations are higher as a deterrent and to ensure compliance.

- Complexity and Cost of Resolution:

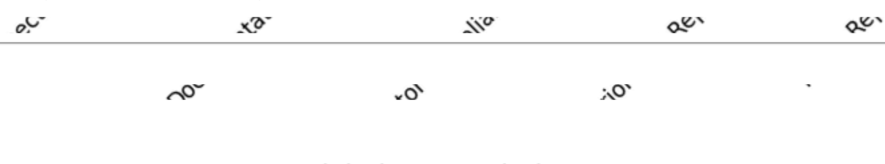
Elevator Compliance likely involves significant technical and regulatory requirements, leading to moderately high fines to cover inspection, repair, or certification costs.

- Administrative Costs:

Violations like Documentation might result from incomplete paperwork or delayed submissions, which are less severe and incur lower fines compared to physical safety issues.

- Frequency vs. Impact:

Violations like Inspection may occur more frequently but are less severe, leading to lower fines to avoid over-penalizing minor infractions.



(16)

My assignment submission includes the following:

(a) A file containing code

(b) A write-up in a PDF that clearly includes all of my code, results, and interpretation statements, together

Could not connect to the reCAPTCHA service. Please check your internet connection and reload to get a reCAPTCHA challenge.