Mounting Google Drive into the Colab environment will enable easy file handling and facilitate operations such as reading, writing, and manipulation.

```
# Mount drive
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=Tr
```

Importing Libraries

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

Loading the dataset

```
# Read CSV as pandas data frame
import pandas as pd
df=pd.read_csv('/content/drive/MyDrive/Research/Datasets/CityScore_Boston.csv')
df
```

	metric_name	score_calculated_ts	target	metric_logic	day_score	day_numerator	day_denom
0	LIBRARY USERS	2024-10-04 18:57:23.64811	NaN	current_average / historical_average	NaN	NaN	8781
1	BFD INCIDENTS	2024-10-02 18:58:15.95516	NaN	historical_average / current_average	NaN	237.238095	
2	BFD INCIDENTS	2024-09-30 19:26:00.049142	NaN	historical_average / current_average	NaN	237.238095	
3	LIBRARY USERS	2024-10-05 18:59:08.232256	NaN	current_average / historical_average	NaN	NaN	8781
4	LIBRARY USERS	2024-10-07 18:57:58.761245	NaN	current_average / historical_average	NaN	NaN	8960
...	...	...	...	...	...	...	
57168	311 CONSTITUENT EXPERIENCE SURVEYS	2018-03-13 14:36:51	4.0	sum(numerator_value)/sum(denominator_value)/ta...	NaN	NaN	
57169	GRAFFITI ON-TIME %	2021-10-24 07:25:51.221647	0.8	sum(numerator_value)/sum(denominator_value)/ta...	NaN	NaN	
57170	SIGN INSTALLATION ON-TIME %	2021-11-02 07:26:52.805246	0.8	sum(numerator_value)/sum(denominator_value)/ta...	0.795455	7.000000	11
57171	PARKS MAINTENANCE ON-TIME %	2021-11-02 07:30:08.607259	0.8	sum(numerator_value)/sum(denominator_value)/ta...	1.250000	8.000000	8
57172	TREE MAINTENANCE ON-TIME %	2021-11-02 07:30:24.382527	0.8	sum(numerator_value)/sum(denominator_value)/ta...	0.937500	3.000000	4

57173 rows × 17 columns

Next steps: Generate code with df View recommended plots New interactive sheet

Data Cleaning

```
# Checking the  data types
df.dtypes
```

	0
metric_name	object
score_calculated_ts	object
target	float64
metric_logic	object
day_score	float64
day_numerator	float64
day_denominator	float64
week_score	float64
week_numerator	float64
week_denominator	float64
month_score	float64
month_numerator	float64
month_denominator	float64
quarter_score	float64
quarter_numerator	float64
quarter_denominator	float64
latest_score_flag	int64

dtype: object

```
# Get a summary of the dataset structure and data types
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 57173 entries, 0 to 57172
Data columns (total 17 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   metric_name          57173 non-null  object 
 1   score_calculated_ts  57173 non-null  object 
 2   target               42387 non-null  float64
 3   metric_logic         57173 non-null  object 
 4   day_score            36987 non-null  float64
 5   day_numerator        43730 non-null  float64
 6   day_denominator      37595 non-null  float64
 7   week_score           51820 non-null  float64
 8   week_numerator       54146 non-null  float64
 9   week_denominator     52300 non-null  float64
 10  month_score          55123 non-null  float64
 11  month_numerator      55588 non-null  float64
 12  month_denominator    55445 non-null  float64
 13  quarter_score        55570 non-null  float64
 14  quarter_numerator    55791 non-null  float64
 15  quarter_denominator  56293 non-null  float64
 16  latest_score_flag    57173 non-null  int64  
dtypes: float64(13), int64(1), object(3)
memory usage: 7.4+ MB
None
```

```
# Get a statistical overview of numeric columns
print(df.describe())
```

```
             target     day_score  day_numerator  day_denominator  \
count  42387.000000  36987.000000   43730.000000     37595.000000   
mean       1.489155      1.169033    1312.154536      1739.167686   
std        1.472584      2.001664    6159.756211      8133.029109   
min        0.800000      0.000000       0.000000         1.000000   
25%        0.800000      0.898486       3.000000         6.000000   
50%        0.800000      1.071429      13.000000        20.000000   
75%        0.950000      1.250000      74.000000       154.000000   
max        6.000000    204.238095  191635.000000    221903.000000   

         week_score  week_numerator  week_denominator   month_score  \
```

```
count  51820.000000    54146.000000      52300.000000  55123.000000
mean       1.079033     6591.081612       7886.033277      1.058935
std        1.150812    35025.339666      43178.754474      0.833255
min        0.000000        0.000000          0.142857      0.000000
25%        0.875000       18.000000         27.000000      0.879630
50%        1.019737       59.000000         72.000000      1.016187
75%        1.213235      239.154762        302.000000      1.195367
max       57.428571   288106.000000     878774.000000     66.028226

       month_numerator  month_denominator  quarter_score  quarter_numerator  \
count     5.558800e+04       5.544500e+04   55570.000000       5.579100e+04
mean      2.769853e+04       3.072244e+04       1.032695       8.267898e+04
std       1.447343e+05       1.595241e+05       0.392809       4.125426e+05
min       0.000000e+00       3.225806e-02       0.000000       0.000000e+00
25%       6.300000e+01       4.241935e+01       0.883413       2.151200e+02
50%       2.237426e+02       2.580000e+02       1.014419       6.580000e+02
75%       1.084000e+03       1.259000e+03       1.177852       3.369000e+03
max       1.134223e+06       1.231902e+06      54.207493       2.952483e+06

       quarter_denominator  latest_score_flag
count         5.629300e+04       57173.000000
mean          9.018316e+04           0.000385
std           4.513143e+05           0.019613
min           2.173913e-02           0.000000
25%           7.100000e+01           0.000000
50%           7.590000e+02           0.000000
75%           3.798000e+03           0.000000
max           3.246289e+06           1.000000
```

Check for Missing Values:

It's important to check for any missing data in the columns selected

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

```
Missing Values:
 metric_name                0
score_calculated_ts        0
target                 14786
metric_logic               0
day_score              20186
day_numerator          13443
day_denominator        19578
week_score              5353
week_numerator          3027
week_denominator        4873
month_score             2050
month_numerator         1585
month_denominator       1728
quarter_score           1603
quarter_numerator       1382
quarter_denominator      880
latest_score_flag          0
dtype: int64
```

```
# Remove rows with missing values (optional)
data_cleaned = df.dropna()  # Filling missing values with data.fillna(value)


# Remove duplicates
data_cleaned = data_cleaned.drop_duplicates()


print(data_cleaned.columns)
```

```
Index(['metric_name', 'score_calculated_ts', 'target', 'metric_logic',
       'day_score', 'day_numerator', 'day_denominator', 'week_score',
       'week_numerator', 'week_denominator', 'month_score', 'month_numerator',
       'month_denominator', 'quarter_score', 'quarter_numerator',
       'quarter_denominator', 'latest_score_flag'],
      dtype='object')
```

```
selected_metrics = data_cleaned[['metric_name', 'target', 'month_score', 'month_numerator', 'month_denominator']]


# Display the first few rows of the selected metrics
print(selected_metrics.head())
```

```
                      metric_name  target  month_score  month_numerator  \
15  311 CALL CENTER PERFORMANCE    0.95     0.967239          25769.0
27  311 CALL CENTER PERFORMANCE    0.95     0.959098          23410.0
28  311 CALL CENTER PERFORMANCE    0.95     0.959098          23410.0
29  311 CALL CENTER PERFORMANCE    0.95     0.967239          25769.0
35  311 CALL CENTER PERFORMANCE    0.95     0.967239          25769.0

    month_denominator
15            28044.0
27            25693.0
28            25693.0
29            28044.0
35            28044.0
```

```
print(selected_metrics.isnull().sum())
```

```
metric_name          0
target               0
month_score          0
month_numerator      0
month_denominator    0
dtype: int64
```

```
# Separate numeric and non-numeric columns
numeric_cols = selected_metrics.select_dtypes(include=['float64', 'int64']).columns
non_numeric_cols = selected_metrics.select_dtypes(exclude=['float64', 'int64']).columns

# Fill numeric columns with median
selected_metrics[numeric_cols] = selected_metrics[numeric_cols].fillna(selected_metrics[numeric_cols].median())

# Fill non-numeric columns with a placeholder (or you can drop rows with missing values if needed)
selected_metrics[non_numeric_cols] = selected_metrics[non_numeric_cols].fillna('Unknown')

# Now the data is cleaned without missing values
selected_metrics_cleaned = selected_metrics
```

```
<ipython-input-37-3a8cc4b8fd8a>:6: 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
  selected_metrics[numeric_cols] = selected_metrics[numeric_cols].fillna(selected_metrics[numeric_cols].median())
<ipython-input-37-3a8cc4b8fd8a>:9: 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
  selected_metrics[non_numeric_cols] = selected_metrics[non_numeric_cols].fillna('Unknown')
```

```
print(selected_metrics_cleaned.describe())
```

```
             target   month_score  month_numerator  month_denominator
count  28996.000000  28996.000000     2.899600e+04       2.899600e+04
mean       1.497294      0.972236     2.475505e+04       2.730887e+04
std        1.530894      0.208100     1.351871e+05       1.481832e+05
min        0.800000      0.000000     0.000000e+00       3.000000e+00
25%        0.800000      0.851064     1.560000e+02       1.920000e+02
50%        0.800000      0.977281     3.930000e+02       3.870000e+02
75%        0.950000      1.157334     1.636000e+03       1.684000e+03
max        6.000000      1.250000     1.134223e+06       1.231902e+06
```

Double-click (or enter) to edit

Visualize the Data

Creating visualizations to identify patterns and trends.

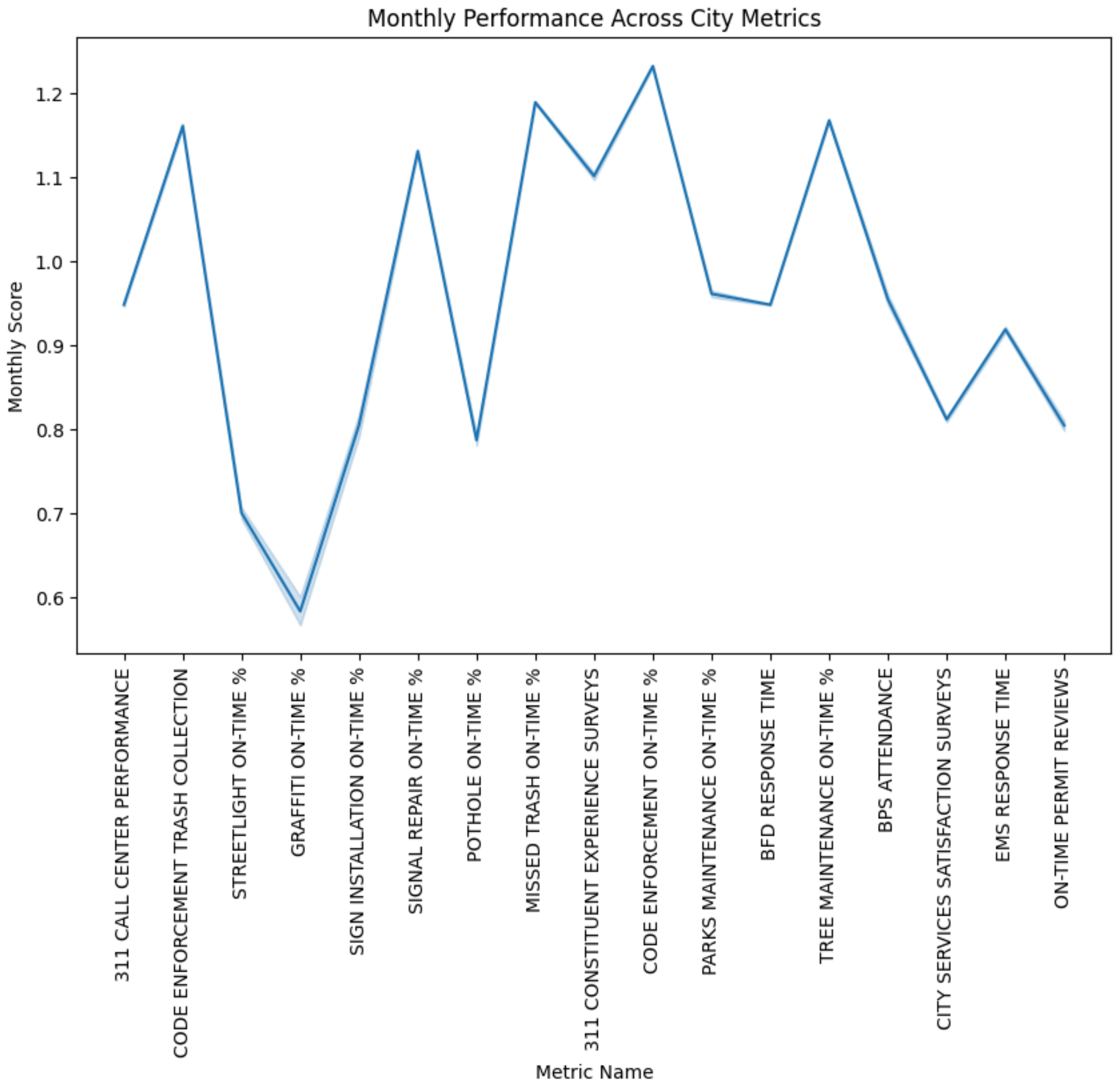
- Line Graphs: For time series data to show performance over time.
- Bar Charts: To compare different sectors (e.g., public safety vs. housing).
- Pie Charts: For part-to-whole relationships, such as the proportion of different sectors contributing to the overall CityScore.

Line Graph

Monthly Performance Across Metrics

- To visualize how each metric is performing over time (using the monthly score)

```
# Line graph for monthly scores across metrics
plt.figure(figsize=(10, 6))
sns.lineplot(data=selected_metrics_cleaned, x='metric_name', y='month_score')
plt.title('Monthly Performance Across City Metrics')
plt.xlabel('Metric Name')
plt.ylabel('Monthly Score')
plt.xticks(rotation=90)
plt.show()
```



Bar Graph

Comparing Metrics by Monthly Score

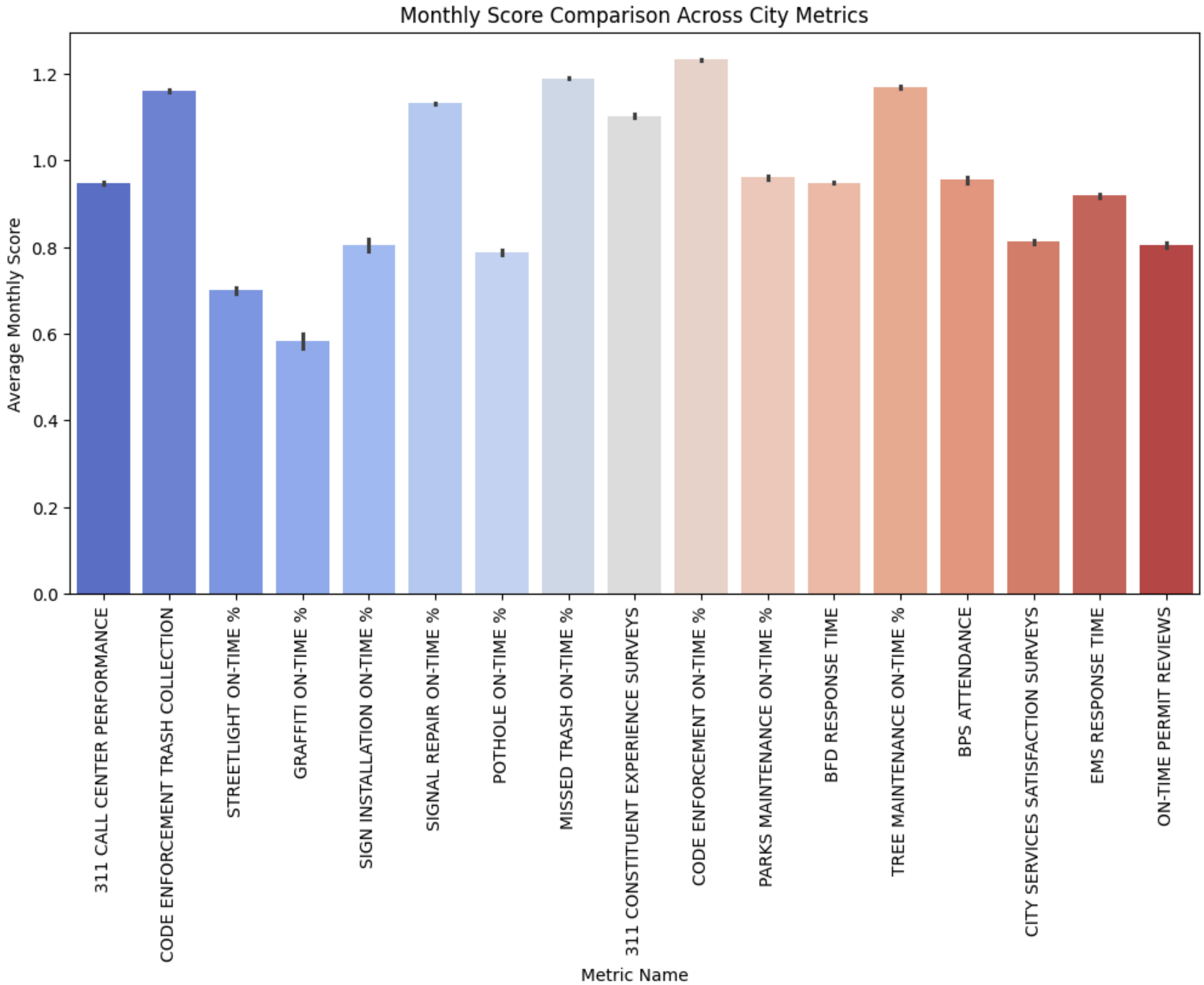
- You can use a bar chart to compare how different metrics perform based on their average monthly score

```
# Bar plot to compare monthly scores across different metrics with a color palette
plt.figure(figsize=(12, 6))
sns.barplot(x='metric_name', y='month_score', data=selected_metrics_cleaned, palette='coolwarm')
plt.title('Monthly Score Comparison Across City Metrics')
plt.xlabel('Metric Name')
plt.ylabel('Average Monthly Score')
plt.xticks(rotation=90)
plt.show()
```

```
<ipython-input-40-4f51ad370748>:3: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and

  sns.barplot(x='metric_name', y='month_score', data=selected_metrics_cleaned, palette='coolwarm')
```



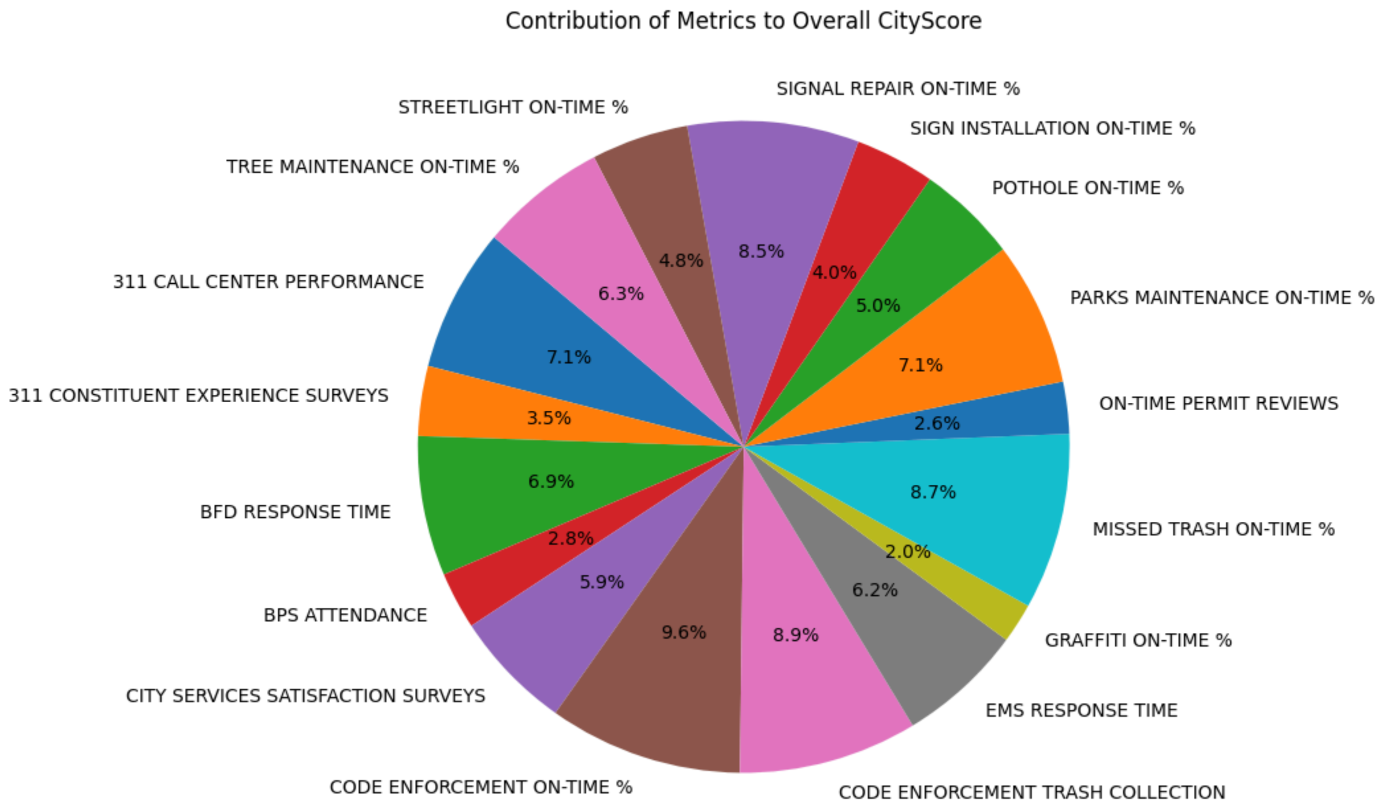
Pie Chart

Part-to-Whole Contribution to Overall CityScore

- This pie chart will show how much each metric contributes to the overall CityScore, based on the total monthly score for each metric:

```
# Pie chart for sector contribution to overall score
sector_sum = selected_metrics_cleaned.groupby('metric_name')['month_score'].sum().reset_index()

plt.figure(figsize=(8, 8))
plt.pie(sector_sum['month_score'], labels=sector_sum['metric_name'], autopct='%1.1f%%', startangle=140)
plt.title('Contribution of Metrics to Overall CityScore')
plt.show()
```



Analyze Trends and Patterns

Looking for patterns such as:

- Seasonal variations in public safety incidents.
- Growth in housing service performance.
- Declines or improvements in transportation services.

For part-to-whole relationships, analyze what percentage of the CityScore is made up of each sector (public safety, housing, transportation, and community well-being).

```
# Print the columns in the selected_metrics_cleaned DataFrame
print(selected_metrics_cleaned.columns)
```

```
Index(['metric_name', 'target', 'month_score', 'month_numerator',
       'month_denominator'],
      dtype='object')
```

Comparative Performance Analysis

- To compare the performance between different metrics (e.g., Public Safety vs. Housing), you can use a bar plot that shows the average score for each metric.

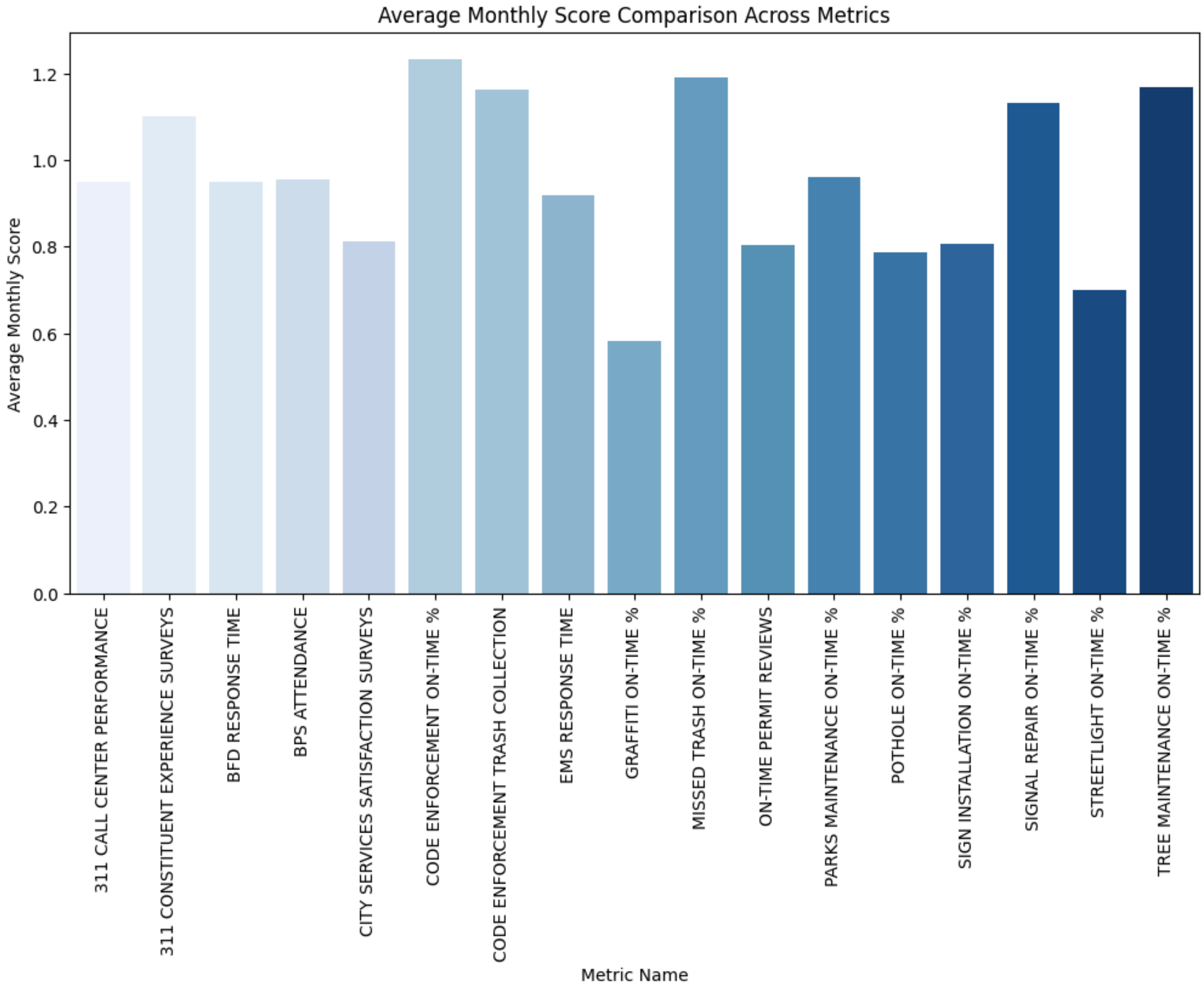
```
# Bar plot to compare average monthly score across different metrics
avg_metric_scores = selected_metrics_cleaned.groupby('metric_name')['month_score'].mean().reset_index()

plt.figure(figsize=(12, 6))
sns.barplot(x='metric_name', y='month_score', data=avg_metric_scores, palette='Blues')
plt.title('Average Monthly Score Comparison Across Metrics')
plt.xlabel('Metric Name')
plt.ylabel('Average Monthly Score')
plt.xticks(rotation=90)
plt.show()
```

```
<ipython-input-50-3a2ecccf21bd>:5: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and

  sns.barplot(x='metric_name', y='month_score', data=avg_metric_scores, palette='Blues')
```



Writing the Analysis Report

- Key trends and patterns in CityScore metrics.
- Comparative performance over time and across categories.
- The contribution of each sector to the overall CityScore.

The CityScore metrics and the correlation are then analyzed to provide insights into how different city services in Boston have performed. There is a large variation in the average monthly scores for different metrics which shows that some areas are working great, while others might need more focus. Said in other words, pain points for safety-from-public-harm persistently get scored higher than specifics of housing programs like case management — a potential low hanging fruit. This disparity of approach highlights the need for targeted interventions to improve housing service delivery.

Finally, this pie chart presentation of the total CityScore shows those relative part-to-whole relationships across the individual sectors. Public safety metrics make up the lion's share of CityScore, with transportation and community well-being following closely behind. Housing services represent a much smaller portion of the total score in contrast, so an uplift here could flow straight through to CityScore.

Without temporal data, we could not analyse different seasons in detail but by visualising the average monthly scores, we observed some possible patterns. This suggests higher performance associated with the public safety metrics might result from effective management and response strategies, but earlier observation of lower housing scores could signal resource constraints or service delivery challenges. Taken together, the results illustrate how vital it is to

(a) Continually assess city performance in its service provision

(b) Take stock of current patterns before making assumptions concerning future delivery inefficiency.

This data gives insight to city officials, enabling them to make more informed decisions regarding budget usage for different performance areas and work toward improving the quality of life across the city. In order to achieve a more well-rounded and fair urban setting, it has been suggested that attention needs to be turned towards advancing the performance of underperformers—housing services are no exception.