4/14/2025

6219 COMP - Technology:

Implementation Summary

FINAL YEAR, 2ND SEMESTER.
COMPUTER SCIENCE AND MATHEMATICS.

Table of Contents

Introduction:

Urban transportation systems produce enormous values of data that can be used to improve productivity and optimize services. In this report, unsupervised machine learning clustering techniques, K-means and DBSCAN, will be used to analyse Uber pickups in a NYC dataset, to identify high-demand zones. This report intends to identify trends in Uber's operational hotspots by clustering pickup zones into spatial clusters. These clusters can help guide strategies for infrastructure planning, surge pricing or driver allocation. The implementation process will include the Preprocessing of the data, Elbow Method clustering for optimal K and evaluation using the Silhouette Score. The selected approach is justified, by comparing results obtained with other clustering techniques.

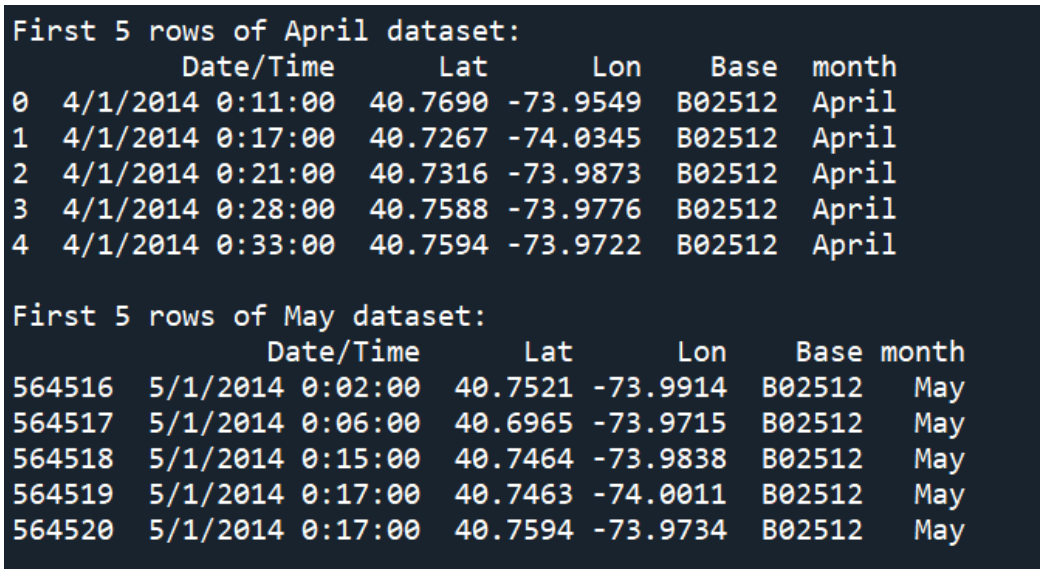
Part 1: Selection and Justification of Dataset and AI Technique:

Dataset Properties and Selection Justification:

For this report, the Uber Pickups in New York City dataset was selected, with an emphasis on the April and May 2014 data. This dataset provided details about Uber pickups in New York City, including the date and time of pickup, the latitude and longitude coordinates, and the Uber base code. It was published by FiveThirtyEight and is accessible on Kaggle (FiveThirtyEight, 2020). The dataset provides a significant amount of data for insightful analysis, with over 1 million pickup records for April and May combined. The Kaggle dataset is made available under the Creative Commons CCO 1.0 Universal (Public Domain Dedication) license. This implies that the data is completely available for use, alteration and distribution, even for commercial use.

The dataset was chosen for several important reasons. First, urban mobility system heavily depends on transportation data, especially from ride-sharing devices like Uber, which has a direct impact on traffic patterns, public transportation use and urban planning decisions. Secondly, the dataset includes both temporal (date and time) and spatial (latitude and longitude) dimensions, enabling extensive multidimensional analysis. Thirdly, the complex urban environment, diverse neighbourhoods and various transportation needs of New York City, makes it the perfect place to identify significant trends in demand for ridesharing.

The dataset has characteristics that make it well-suitable for analysis in a real-world scenario: Spatial coordinates: Each record has accurate latitude and longitude coordinates; Temporal information: Date and time stamps permit the analysis of patterns at various times of day, as well as week; Base codes: Identifiers for various Uber bases provide additional context; Clean structure: The data is well-structured with few missing values; Strong statistical significance: millions of records offer strong numerical significance; Two-months span: Comparison and trend analysis are made possible with data from two consecutive months (April and May).



First 5 rows of April dataset:

	Date/Time	Lat	Lon	Base	month
0	4/1/2014 0:11:00	40.7690	-73.9549	B02512	April
1	4/1/2014 0:17:00	40.7267	-74.0345	B02512	April
2	4/1/2014 0:21:00	40.7316	-73.9873	B02512	April
3	4/1/2014 0:28:00	40.7588	-73.9776	B02512	April
4	4/1/2014 0:33:00	40.7594	-73.9722	B02512	April

First 5 rows of May dataset:

	Date/Time	Lat	Lon	Base	month
564516	5/1/2014 0:02:00	40.7521	-73.9914	B02512	May
564517	5/1/2014 0:06:00	40.6965	-73.9715	B02512	May
564518	5/1/2014 0:15:00	40.7464	-73.9838	B02512	May
564519	5/1/2014 0:17:00	40.7463	-74.0011	B02512	May
564520	5/1/2014 0:17:00	40.7594	-73.9734	B02512	May

Figure 1: Sample of raw Uber pickup data.

As seen in Figure 1, is a table that displays the first 5 rows of the combined April and May 2014 Uber datasets. The data includes the values of pickup timestamps (Date/Time), location coordinates (Lat/Lon), base codes, and labels for each month. This sample shows the original data structure before any processing or cleaning was done for the clustering analysis.

AI Technique Explanation and Justification:

Clustering is the main Artificial Intelligence (AI) technique chosen for this dataset, which utilises both the K-means and DBSCAN (Density-Based Spatial Clustering of Applications with Noise) algorithms. Clustering is an unsupervised machine learning technique that does not require labelled training data; instead, it groups similar data points together, according to their features (McGregor, 2020). Each data point is assigned to the cluster with the closest mean (centroid) in K-means clustering, which divides into k separate clusters. The algorithm recalculates centroids until convergence, assigning points constantly to the closest centroid

(Zubair et. al., 2022). DBSCAN, in contrast, classifies points in low-density regions as noise or outliers and clusters together points that are densely packed in space (Ester et. al., 2017). DBSCAN can find clusters of any shape, and unlike K-means, does not require a predetermined number of clusters.

The clustering techniques were selected because of several reasons: Pattern discovery: Clustering can detect temporal trends and high-demands regions in the pickup data by revealing natural groupings without the need for defined labels; Spatial analysis: Both algorithms are good at analysing spatial data; K-means can identify concentrated pickup zones, while DBSCAN can detect irregularly formed clusters that follow urban geography; Complementary methods: K-means offers distinct, comprehensible clusters with well-defined centroids, whereas DBSCAN provides complementary viewpoints on the data; Business relevance: Business driven-decisions like driver allocation, surge pricing methods and service optimisation can be directly impacted by the clusters that are produced; Computational Efficiency: Both algorithms are suitable for the millions of records in the Uber dataset, since they can process large datasets effectively.

The combination of the advantages and disadvantages of both the K-means and DBSCAN clustering techniques provides an extensive overview of the busiest pickup zones in New York City. By identifying both temporal and spatial hotspots in Uber pickup demand, this implementation seeks to provide information for strategies related to service optimisation, driver allocation and transportation planning in a real-world scenario.

Part 2: Data Preparation:

Data Loading and Initial Exploration:

The initial stage of the data preparation process included loading and examining the Uber pickup statistics for April and May of the year 2014. The two monthly datasets were loaded by the data exploration script (`data_exploration.py`), which added a month identifier column to each dataset and combined them into a single dataset for analysis, as seen in Figure 2.

```python
# Load April and May Uber datasets
print("Loading April and May datasets...")
try:
    df_april = pd.read_csv('C:/6219COMP_Technology/AI Technique/Transportation_Dataset/uber-raw-data-apr14.csv')
    df_may = pd.read_csv('C:/6219COMP_Technology/AI Technique/Transportation_Dataset/uber-raw-data-may14.csv')

    df_april['month'] = 'April'
    df_may['month'] = 'May'

    # Combine datasets
    df = pd.concat([df_april, df_may], ignore_index=True)
    print(f"Datasets loaded and combined successfully! Total records: {len(df)}")
```

Figure 2: Data Loading and Merging Script.

Figure 2 shows the snippet of code for data collection. The code reads two CSV files with the raw Uber pickup data for April and May. A new month column is added to each dataset to distinguish the records after merging. The datasets are then combined into a single dataset, with which the total amount of records is outputted. The combined dataset is the foundation of future analysis and visualizations.

```
Loading April and May datasets...
Datasets loaded and combined successfully! Total records: 1216951
```

Figure 3: Dataset Load Confirmation.

Figure 3 reflects this successful execution for the code shown in Figure 2, as well as the effective loading and combination of the April and May datasets. The printed statement guarantees that the data is prepared for additional analysis by displaying the total amount of combined records (i.e. 1,216,951).

The basic structure of the data, upon initial investigation, comprised of the following columns: `Date/Time`: The pickup timestamp; `Lat`: Latitude coordinate; `Lon`: Longitude coordinate; `Base`: Uber base code; `month`: additional column that indicates if the record is from the month of April or May. The preprocessing procedures were made easier, as the investigation verified that there were no missing values in the dataset. The basic statistics for each month were calculated, to understand the coordinate distribution and identify any significant variations between the April and May data, as seen in Figure 4 below.

```
Missing Values:
Date/Time    0
Lat          0
Lon          0
Base         0
month        0
dtype: int64

April Dataset Statistics:
                 Lat            Lon
count  564516.000000  564516.000000
mean       40.740005     -73.976817
std         0.036083       0.050426
min        40.072900     -74.773300
25%        40.722500     -73.997700
50%        40.742500     -73.984800
75%        40.760700     -73.970000
max        42.116600     -72.066600

May Dataset Statistics:
                 Lat            Lon
count  652435.000000  652435.000000
mean       40.740072     -73.975004
std         0.037537       0.054165
min        40.106700     -74.929000
25%        40.722400     -73.997000
50%        40.743300     -73.983900
75%        40.761400     -73.968100
max        41.322500     -72.180100
```

Figure 4: Output of the initial investigation done in the `data_exploration.py` script.

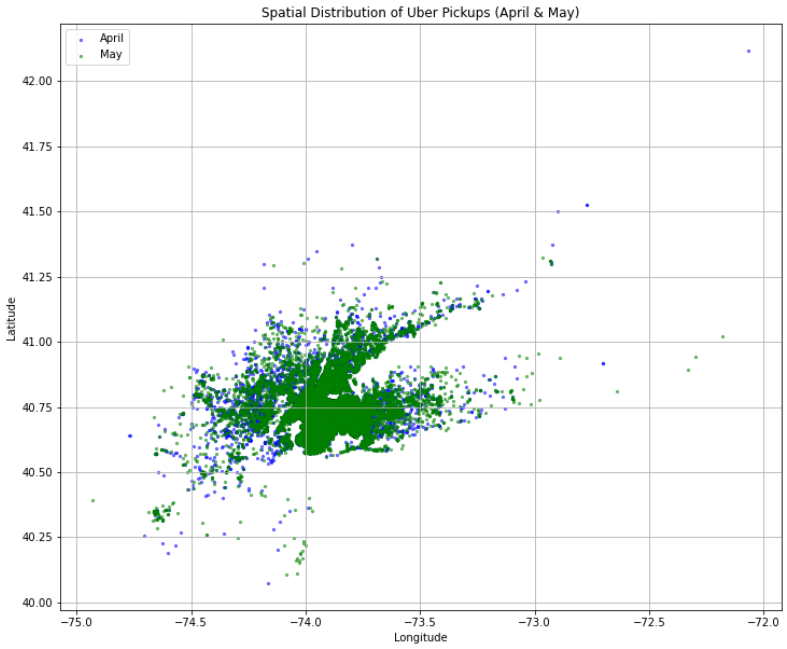


Figure 5: Spatial distribution of raw Uber pickups coloured by month.

In Figure 5 above, the scatter plot depicts the geographical distribution of all Uber pickups in New York City, where the blue points indicate April pickups, and the green points indicate May pickups. The data points of the visualization demonstrate the concentration of pickups in Manhattan and the surrounding areas, as well as the distinct outline of the island.

To process the Uber pickup timestamps, additional features were extracted from the `Date/Time` column to aid in temporal analysis, which includes `hour`: Hour of the day (0-23); and `day_of_week`: Day of the week (0 = Monday, 6 = Sunday), as seen in Figure 6.

```
# Creates temporal features
print("\nCreating temporal features...")

df['Date/Time'] = pd.to_datetime(df['Date/Time'])
df['hour'] = df['Date/Time'].dt.hour
df['day_of_week'] = df['Date/Time'].dt.dayofweek
```

Figure 6: Extracting Temporal Features from Date/Time column in the dataset.

These derived features help show distinct temporal patterns, which makes it easier to identify daily pickup patterns, analyse ridesharing demand trends and identify peak activity hours. Understanding these features is essential for optimising ride availability and predicting consumer behaviour.

```
# 2. Hourly distribution - shows when pickups occur throughpout the day per month
plt.figure(figsize=(14, 6))
for i, month in enumerate(['April', 'May']):
    plt.subplot(1, 2, i+1)
    hourly = df[df['month'] == month]['hour'].value_counts().sort_index()
    sns.barplot(x=hourly.index, y=hourly.values, color='skyblue')
    plt.title(f'Pickups by Hour - {month}')
    plt.xlabel('Hour of Day')
    plt.ylabel('Number of Pickups')
    plt.grid(True, axis='y')
plt.tight_layout()
plt.savefig('visualizations/hourly_distribution.png')
plt.show()
plt.close()

# 3. Daily distribution - shows patterns across days of the week for each month
plt.figure(figsize=(14, 6))
for i, month in enumerate(['April', 'May']):
    plt.subplot(1, 2, i+1)
    daily = df[df['month'] == month]['day_of_week'].value_counts().sort_index()
    sns.barplot(x=daily.index, y=daily.values, color='salmon')
    plt.title(f'Pickups by Day of Week - {month}')
    plt.xlabel('Day of Week (0=Mon, 6=Sun)')
    plt.ylabel('Number of Pickups')
    plt.xticks(ticks=range(7), labels=['Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat', 'Sun'])
    plt.grid(True, axis='y')
plt.tight_layout()
plt.savefig('visualizations/daily_distribution.png')
plt.show()
plt.close()

# 4. Heatmaps: Hour x Day of Week - shows temporal patterns
plt.figure(figsize=(16, 10))
for i, month in enumerate(['April', 'May']):
    plt.subplot(2, 1, i+1)
    pivot = pd.crosstab(df[df['month'] == month]['day_of_week'], df[df['month'] == month]['hour'])
    sns.heatmap(pivot, cmap='YlGnBu')
    plt.title(f'Heatmap of Pickups by Hour and Day - {month}')
    plt.xlabel('Hour of Day')
    plt.ylabel('Day of Week')
    plt.yticks(ticks=range(7), labels=['Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat', 'Sun'], rotation=0)
plt.tight_layout()
plt.savefig('visualizations/hour_day_heatmap.png')
plt.show()
plt.close()
```

Figure 7: Temporal Analysis of Uber Pickups: Hourly, Daily and Heatmap Distributions.

Figure 7 shows the code that generates three visualisations that analyse Uber pickup patterns: an hourly distribution demonstrating peak periods of activity, a daily distribution showing fluctuations in pickups across weekdays, and a heatmap that shows the variations of ride demands hourly and daily. These visualizations work together to reveal relevant information about rider behaviour, which facilitates effective resource allocation and scheduling.

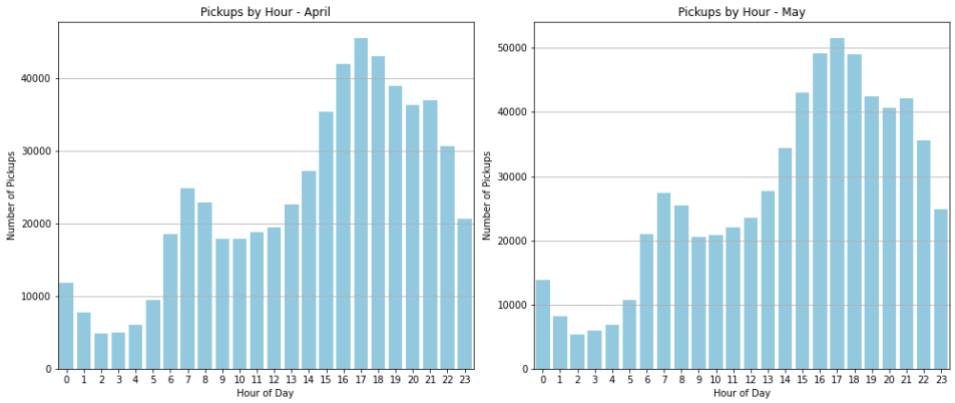


Figure 8: Hourly distribution of pickups for April and May.

In Figure 8 above, the number of pickups for April (left) and May (right) are displayed in the bar charts by hour of the day. Similar trends are observed in both months, with low activity in the early morning hours (2-5 AM), a morning peak around (8-9 AM), and the largest activity occurring in the evenings (17-19 PM). Pickup volumes are consistently higher in May, compared to April for most hours.

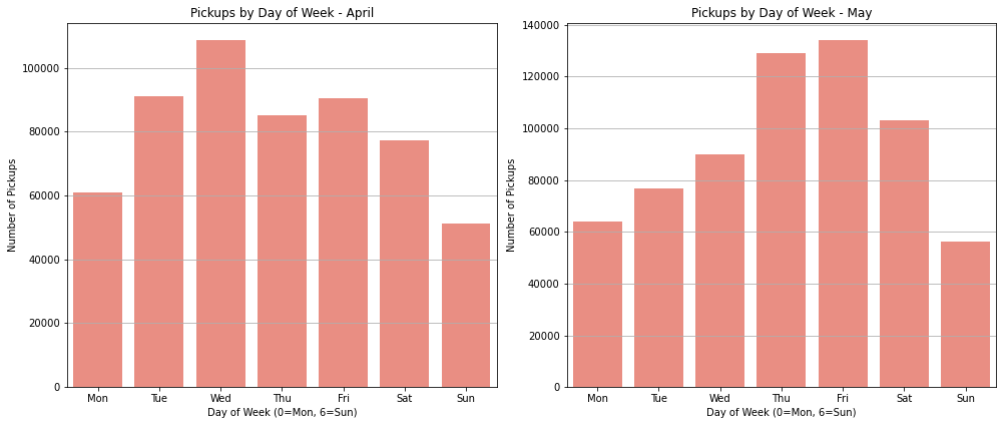


Figure 9: Day of week distribution of pickups for April and May.

In Figure 9, the bar charts show the number of pickups for April (left) and May (right) by day of the week. The visualization shows that weekday activity is higher in both months, especially

Tuesday – Friday, while weekend activity is lower. Compared to April, May exhibits considerably greater volumes on Thursday and Friday.

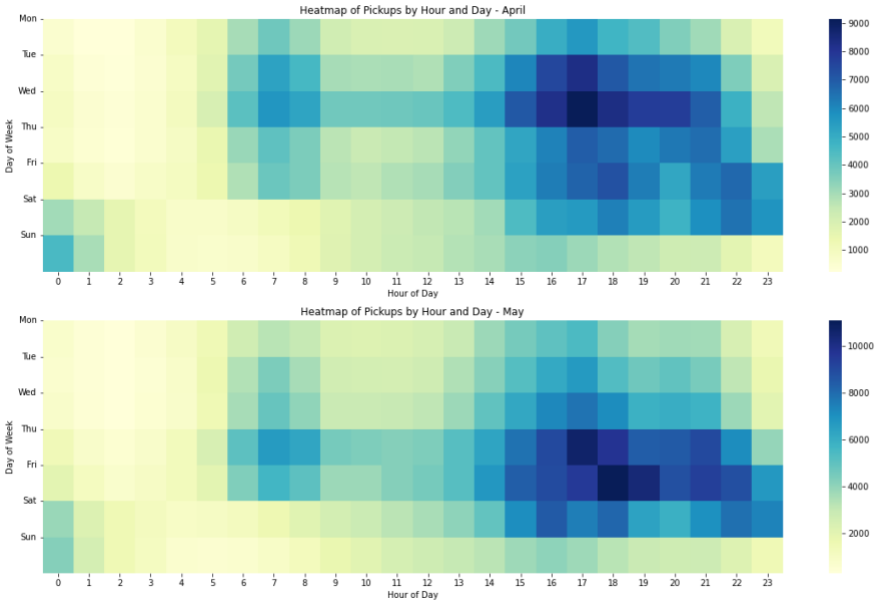


Figure 10: Heatmap showing pickup patterns by hour and day of week.

In Figure 10, the heatmaps shows the pickup density for April (top) and May (bottom) by hour (x-axis) and day of the week (y-axis). Higher pickup volumes are indicated by darker shades of blue. The trends show regular patterns from April to May, with a decrease in weekend morning activities and weekday evening peaks, particularly Wednesday to Friday, from (17-19 PM).

Additionally, the exploration phase also observed how pickups were distributed around various Uber bases, which added context for comprehending service trends.

```
# 5. Base distribution - shows pickup patterns by Uber base
plt.figure(figsize=(14, 6))
for i, month in enumerate(['April', 'May']):
    plt.subplot(1, 2, i+1)
    base_counts = df[df['month'] == month]['Base'].value_counts()
    sns.barplot(x=base_counts.index, y=base_counts.values, palette='viridis')
    plt.title(f'Pickups by Base - {month}')
    plt.xlabel('Base')
    plt.ylabel('Number of Pickups')
    plt.grid(True, axis='y')
plt.tight_layout()
plt.savefig('visualizations/base_distribution.png')
plt.show()
plt.close()
```

Figure 11: Distribution of Uber Pickups Across Bases for April and May Code.

The code in Figure 11, compares Uber pickup distributions across bases in April and May using bar charts. The number of pickups per base is counted, the dataset is filtered for each month and is then visualised. The resulting visualisation illustrates monthly patterns in Uber operations and shows variances in demand across several bases, as seen in Figure 12.

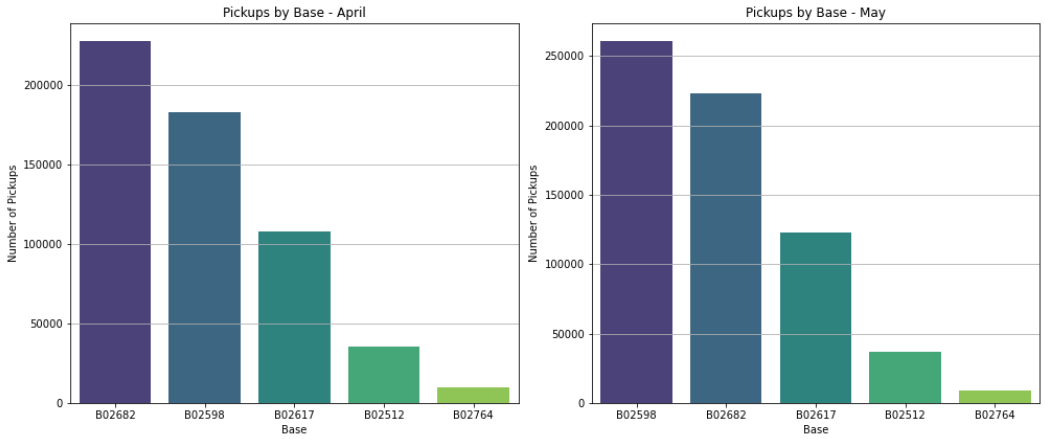


Figure 12: Distribution of Uber Pickups Across Bases for April and May Visualisation.

In Figure 12, the visualisation shows the total number of Uber pickups for April and May, distributed across 5 bases. The two bar charts demonstrate patterns in service demand by showing differences in pickup volume throughout the two months. In May, pickups at Base B02598 are significantly higher than in April, although Base B02682 shows a minor decrease. The pickup numbers for the other bases remain relatively constant. These trends offer insightful information about changes in operations and passenger demand at various Uber hubs.

Data Preprocessing Steps and Results:

To prepare the data for the clustering implementation, several preprocessing steps were added to the `data_preprocessing.py` script, following the initial exploration. First, Extraction of Temporal Features: Expanding upon the exploration stage, the preprocessing script defined the process of extracting temporal features from the `Date/Time` column, resulting in: `hour`, which indicates the time of day (0-23); `day_of_week`, indicates the day of the week (0 = Monday, 6 = Sunday); `day_of_month`, indicates the day of the month; `month_num`, indicates the month number (4 = April, 5 = May). Secondly, Missing Value Handling: The preprocessing script has strong handling for possible missing values, even if the initial investigation revealed no missing values.

```
print("Temporal features added: hour, day_of_week, day_of_month, month_num")

# Checks and handles any missing value in the dataset
print("\n Handling Missing Values")

missing = df.isnull().sum()
print("Missing values per column:")
print(missing)

if missing.sum() == 0:
    print("No missing values found.")
else:
    # Fill numerical columns with median, categorical columns with mode
    for col in df.select_dtypes(include=np.number).columns:
        df[col].fillna(df[col].median(), inplace=True)
    for col in df.select_dtypes(include='object').columns:
        df[col].fillna(df[col].mode()[0], inplace=True)
    print("Missing values filled.")
```

Figure 13: Handling Missing Values in Uber Pickup Data.

The code in Figure 13, checks for missing values to optimise usability of the Uber pickup data. The dataset integrity for the clustering analysis is maintained by filling numerical columns with their median values (the middle number in an organised list), if they are discovered and the categorical columns with the mode value (the value that appears the most frequently).

```
 Temporal Feature Extraction
Temporal features added: hour, day_of_week, day_of_month, month_num

 Handling Missing Values
Missing values per column:
Date/Time       0
Lat             0
Lon             0
Base            0
month           0
hour            0
day_of_week     0
day_of_month    0
month_num       0
dtype: int64
No missing values found.
```

Figure 14: Output for the Handling of Missing Values and Temporal Feature Extraction in Uber Pickup Data.

The `hour`, `day_of_week`, `day_of_month` and `month_num` temporal features were successfully extracted, and the output in Figure 14, verifies that there are no missing values. If any missing data existed, data consistency would be guaranteed by the established preparation method.

Thirdly, Geographic Outlier Removal: Due to either data submission or GPS problems, the spatial visualisation showed certain coordinates that were beyond the New York City area. Geographical boundaries were established to concentrate the analysis on real NYC pickups: Between 40.5° and 41.0° is latitude and between -74.3° and -73.7° is longitude. Outliers whose coordinates fell outside of the specified ranges were eliminated from records.

```
# Visualizes the effect of removing the outlier
print("Creating outlier comparison maps...")
plt.figure(figsize=(15, 10))

nyc_bounds = {'lat_min': 40.5, 'lat_max': 41.0, 'lon_min': -74.3, 'lon_max': -73.7}
raw_outliers = raw_df[(raw_df['Lat'] < nyc_bounds['lat_min']) | (raw_df['Lat'] > nyc_bounds['lat_max']) |
                      (raw_df['Lon'] < nyc_bounds['lon_min']) | (raw_df['Lon'] > nyc_bounds['lon_max'])]
raw_inliers = raw_df[(raw_df['Lat'] >= nyc_bounds['lat_min']) & (raw_df['Lat'] <= nyc_bounds['lat_max']) &
                     (raw_df['Lon'] >= nyc_bounds['lon_min']) & (raw_df['Lon'] <= nyc_bounds['lon_max'])]

# Before cleaning
plt.subplot(2, 1, 1)
plt.scatter(raw_inliers['Lon'], raw_inliers['Lat'], s=5, alpha=0.5, c='gray', label='Inliers')
plt.scatter(raw_outliers['Lon'], raw_outliers['Lat'], s=20, alpha=0.8, c='red', label='Outliers')
plt.title('Raw Data - Outliers Highlighted')
plt.xlabel('Longitude')
plt.ylabel('Latitude')
plt.legend()
plt.grid(True)

# After cleaning
plt.subplot(2, 1, 2)
plt.scatter(preprocessed_df['Lon'], preprocessed_df['Lat'], s=5, alpha=0.5, c='blue', label='Cleaned')
plt.title('Cleaned Data - Outliers Removed')
plt.xlabel('Longitude')
plt.ylabel('Latitude')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.savefig(f"{visual_dir}/outliers_removal_comparison_by_month.png")
plt.show()
plt.close()
```

Figure 15: Code that displays the effect of removing outliers from a dataset of geographical coordinates.

The code in Figure 15, defines geographical boundaries for New York City, finds the outliers in Uber pickup locations and creates charts to compare the raw data with highlighted outliers and the cleaned dataset after outlier removal.

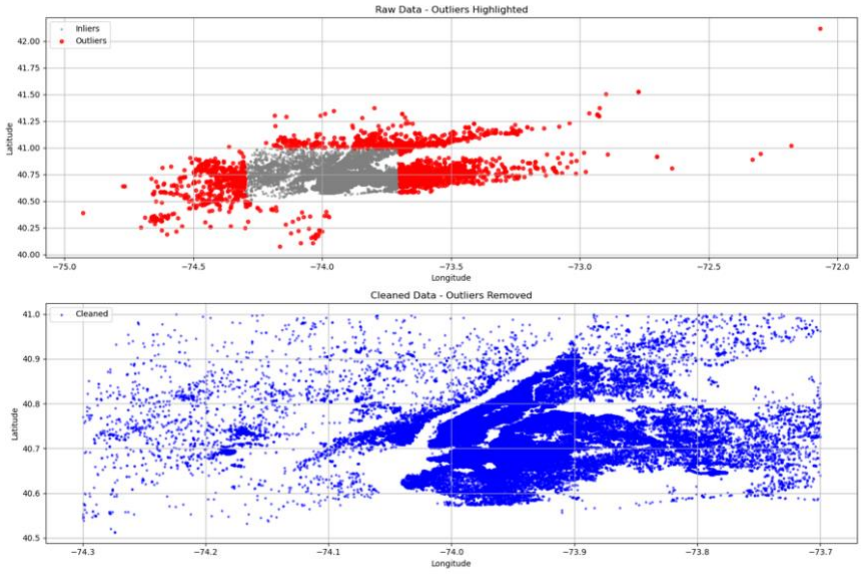


Figure 16: Comparison showing raw data with outliers highlighted and cleaned data.

In Figure 16 above, the raw data is displayed in the top panel, with normal points and the outliers highlighted in red. After the outliers are removed, the cleaned dataset is displayed in the bottom panel. The geographic boundaries used for data cleaning and the efficiency of the outlier removal displayed in Figure 15 are shown in this visualisation.

```python
# Compares spatial distribution by month
print("\nCreating spatial distribution comparison by month...")
plt.figure(figsize=(15, 7))
for i, df in enumerate([raw_df, preprocessed_df]):
    plt.subplot(1, 2, i+1)
    plt.scatter(df[df['month'] == 'April']['Lon'], df[df['month'] == 'April']['Lat'], s=5, alpha=0.5, label='April', c='blue')
    plt.scatter(df[df['month'] == 'May']['Lon'], df[df['month'] == 'May']['Lat'], s=5, alpha=0.5, label='May', c='green')
    plt.title(f"{'Raw' if i == 0 else 'Cleaned'} Data - Spatial Distribution")
    plt.xlabel('Longitude')
    plt.ylabel('Latitude')
    plt.legend()
    plt.grid(True)
plt.tight_layout()
plt.savefig(f"{visual_dir}/spatial_distribution_comparison_by_month.png")
plt.show()
plt.close()
```

Figure 17: Code that generates the before and after pre-processing of the spatial distribution.

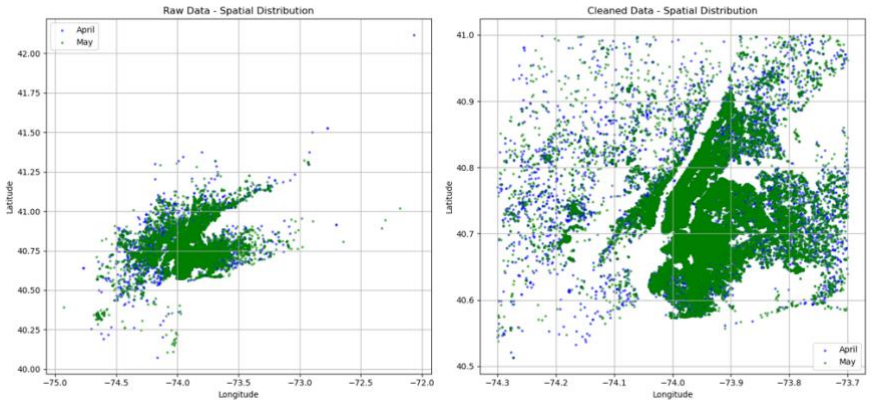


Figure 18: Comparison of spatial distribution before and after pre-processing.

In Figure 17 above, is the code that creates the visualisation shown in Figure 18. The spatial distribution of Uber pickups before (left) and after (right) cleaning is compared in Figure 18. The raw data displays a broader geographic area, including outliers, whereas the cleaned data includes a zoomed-in picture of the NYC area. The cleaned data makes Manhattan Island's unique shape much more apparent. After cleaning, the seasonal patterns are maintained, as supported by the comparable spatial distributions of April (blue) and May (green). The preprocessing techniques successfully eliminated geographical outliers, while preserving the essential spatial patterns of the dataset, as shown by the visualisation.

Fourth, Feature Scaling: Standardising was used to guarantee that every feature contributed equally to the clustering process, because clustering techniques are sensitive to the size of the features. The following three features were scaled: Only spatial features (`Lat`, `Lon`): for clustering based on location; Spatial and temporal features (`Lat`, `Lon`, `hour`, `day_of_week`: for time-aware clustering; Spatial, Temporal and Month (`Lat`, `Lon`, `hour`, `day_of_week`, `month_num`: to examine potential variances between months. Each feature was transformed using the StandardScaler feature from scikit-learn to have a unit variance and zero mean (Boisberranger et. al., 2025), as seen in Figure 19 below.

```
# Scale features for clustering technique
print("\n Feature Scaling for Clustering")

# 1. Spatial features only - for pure location-based clustering
X_spatial = df[['Lat', 'Lon']]
X_spatial_scaled = StandardScaler().fit_transform(X_spatial)

# 2. Spatial + Temporal features - for time-aware clustering
X_temp_spatial = df[['Lat', 'Lon', 'hour', 'day_of_week']]
X_temp_spatial_scaled = StandardScaler().fit_transform(X_temp_spatial)

# 3. Spatial + Temporal + Month - checks monthly pattern trends
X_with_month = df[['Lat', 'Lon', 'hour', 'day_of_week', 'month_num']]
X_with_month_scaled = StandardScaler().fit_transform(X_with_month)

print("Features scaled and ready for clustering")
```

Figure 19: Code that handles the scaling of the features.

Fifth, Data Storage: Several forms of the pre-processed data were saved for the clustering process. The completed pre-processed dataset is in CSV format and the feature arrays are scaled as NumPy files for quick loading. After removing about 5,000 geographical outliers, the pre-processing stage produced a cleaned dataset with 1, 212,397 records, as seen in Figure 21 below. Better data quality for clustering was ensured by the comparison of raw and pre-processed data, which had no effect on the general distribution patterns.

```
# Saves all processed data for the clustering phase
print("\n Saving Preprocessed Data")

# Save data as CSV file
processed_csv_path = os.path.join(processed_dir, 'preprocessed_uber_data.csv')
df.to_csv(processed_csv_path, index=False)
print(f"Cleaned data saved to: {processed_csv_path}")

# Save NumPy arrays
np.save(os.path.join(processed_dir, 'X_spatial_scaled.npy'), X_spatial_scaled)
np.save(os.path.join(processed_dir, 'X_temporal_spatial_scaled.npy'), X_temp_spatial_scaled)
np.save(os.path.join(processed_dir, 'X_with_month_scaled.npy'), X_with_month_scaled)
np.save(os.path.join(processed_dir, 'month_data.npy'), df['month_num'].values)

print("Scaled features saved as .npy files")
print("\nData preprocessing complete.")
```

Figure 20: Code that handles Data Storage.

```
 Removing Geographical Outliers
Removed outliers. Records removed: 4554
Remaining records: 1212397

 Feature Scaling for Clustering
Features scaled and ready for clustering

 Saving Preprocessed Data
Cleaned data saved to: /Users/adunoluwaoguntuga/Desktop/OneDrive - Liverpool John
Moores University/6219COMP_Technology/AI Technique/Transportation_Dataset/processed/
preprocessed_uber_data.csv
Scaled features saved as .npy files

Data preprocessing complete.
```

Figure 21: Console output of Geographical Outliers, Feature Extraction and saved files.

In Figure 20 above, is the code that handles the saving of the pre-processed files, and in Figure 21, is the output of the saved files, as well as with the record of the removed geographical outliers.

```
# Compares hourly distribution by month
print("Creating hourly distribution comparison by month...")
plt.figure(figsize=(15, 7))
for i, df in enumerate([raw_df, preprocessed_df]):
    plt.subplot(1, 2, i+1)
    for month, color in zip(['April', 'May'], ['blue', 'green']):
        sns.histplot(df[df['month'] == month]['hour'], bins=24, kde=True, color=color, alpha=0.5, label=month)
    plt.title(f"{'Raw' if i == 0 else 'Cleaned'} Data - Hourly Distribution")
    plt.xlabel('Hour of Day')
    plt.ylabel('Count')
    plt.legend()
    plt.grid(True)
plt.tight_layout()
plt.savefig(f"{visual_dir}/hourly_distribution_comparison_by_month.png")
plt.show()
plt.close()

# Compares day of week distribution by month
print("Creating day of week distribution comparison by month...")
plt.figure(figsize=(15, 7))
for i, df in enumerate([raw_df, preprocessed_df]):
    plt.subplot(1, 2, i+1)
    sns.countplot(x='day_of_week', hue='month', data=df)
    plt.title(f"{'Raw' if i == 0 else 'Cleaned'} Data - Day of Week Distribution")
    plt.xlabel('Day of Week')
    plt.xticks(range(7), ['Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat', 'Sun'])
    plt.ylabel('Count')
    plt.grid(True, axis='y')
plt.tight_layout()
plt.savefig(f"{visual_dir}/day_of_week_comparison_by_month.png")
plt.show()
plt.close()

# Compare data volume between raw and cleaned datasets by month
print("Creating data volume comparison bar chart...")
plt.figure(figsize=(10, 6))
volume = pd.DataFrame({
    'Raw': raw_df['month'].value_counts(),
    'Cleaned': preprocessed_df['month'].value_counts()
})
volume.plot(kind='bar', ax=plt.gca())
plt.title('Monthly Data Volume: Raw vs Cleaned')
plt.xlabel('Month')
plt.ylabel('Number of Records')
plt.grid(True, axis='y')
plt.xticks(rotation=0)
plt.tight_layout()
plt.savefig(f"{visual_dir}/data_volume_comparison.png")
plt.show()
plt.close()
```

Figure 22: Code that generates the Raw and Cleaned Data Visualizations.

Figure 22 contains the code that creates visualizations to display data distributions across different features and compares the raw and the cleaned datasets. It generates three distinct plots: hourly distribution by month, day-of-week distribution by month and a bar chart comparing monthly data volumes. The three output visualisations can be seen in the Figures below.

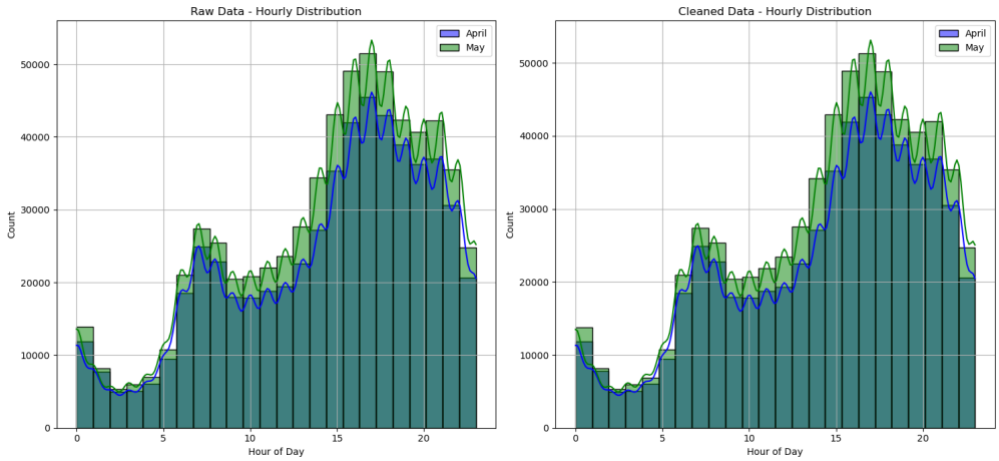


Figure 23: Comparison of hourly distribution before and after pre-processing.

Figure 23 shows a layered bar-line chart in which the hourly pickup distributions before (left) and after (right) data cleansing are displayed in a side-by-side comparison. The consistency of patterns between the raw and cleaned data suggests that the integrity of the time-based analysis was maintained, because the removal of outliers did not significantly change the temporal patterns.

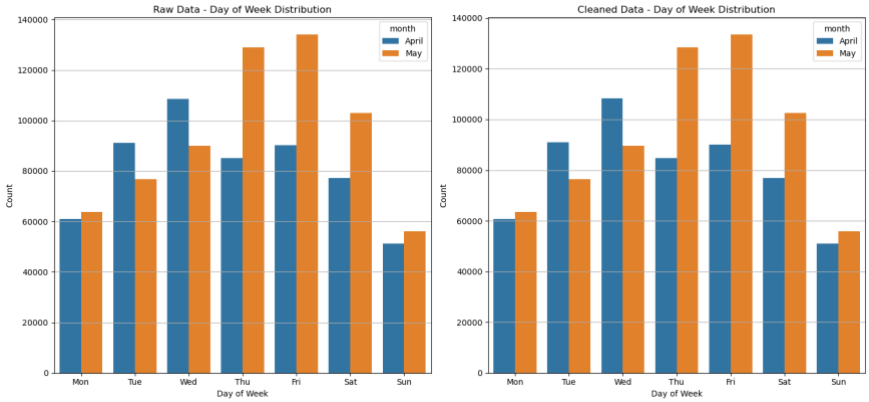


Figure 24: Comparison of day-of-week distribution before and after pre-processing.

In Figure 24, this visualisation shows the comparison of the day-of-week pickup patterns before pre-processing (left) and after pre-processing (right). The regular patterns of weekday peaks and weekend dips in both the raw and cleaned data, show that the cleaning procedure eliminated geographic outliers, while maintaining the weekly patterns.

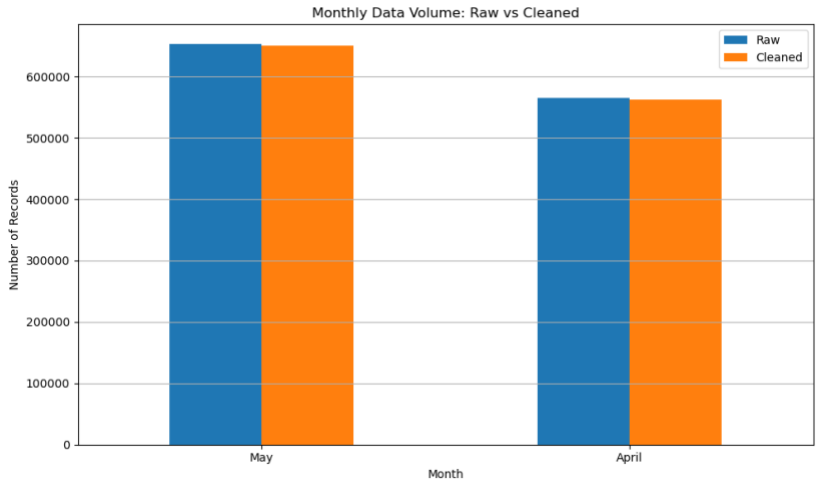


Figure 25: Data volume comparison between raw and cleaned datasets by month.

In Figure 25, the bar chart visualisation compares the quantity of records in the raw and cleaned datasets for each month. Since, there was a small difference between the raw and cleaned bars, most of the original dataset was preserved, and that the outlier removal only had a minor impact on about (4-5%) of the data.

Part 3: AI Technique Implementation and Evaluation:

Implementation and Parameter Tuning:

The clustering implementation (`clustering_process.py`) focused on using the pre-processed Uber pickup data to apply the K-means and DBSCAN techniques. The clustering analysis was performed, using a balanced sample of 5,000 records (2,500 from each month) to guarantee a fair comparison between the patterns in April and May, as seen in Figure 26, below. This sampling strategy increased computational efficiency, while preserving representative patterns.

```
# Samples equally from each month to keep the comparison unbiased
df_apr = df[df['month'] == 'April'].sample(n=2500, random_state=42)
df_may = df[df['month'] == 'May'].sample(n=2500, random_state=42)
df = pd.concat([df_apr, df_may], ignore_index=True)
```

Figure 26: Code that creates a balanced sample of the records.

Finding the ideal number of clusters (k) for the K-means implementation was a crucial first step that was accomplished using two complementary techniques: The Elbow Method: the sum of squared distances (inertia) is plotted against various values of k. The resulting curve's "elbow" shows that the number of clusters and cluster density are well-balanced. The implementation tested k values between 2 and 10. The Silhouette Score: is a measurement,

which ranges from -1 to 1 (higher is better), assesses how similar points are to their own cluster compared to other clusters (Ankita, 2025). According to (Kassambara, 2017), silhouette scores were calculated for every k value.

```
# Creates visualization to help determine optimal k
# Elbow method - look for the 'bend' in the curve
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(k_range, inertia, 'bo-')
plt.title('Elbow Method')
plt.xlabel('k')
plt.ylabel('Inertia')
plt.grid(True)

# Silhouette scores - higher is better
plt.subplot(1, 2, 2)
plt.plot(list(k_range), silhouette_scores, 'ro-')
plt.title('Silhouette Scores')
plt.xlabel('k')
plt.ylabel('Score')
plt.grid(True)
plt.tight_layout()
plt.savefig(f'{visual_dir}/kmeans_elbow_silhouette.png')
plt.show()
plt.close()
```

Figure 27: Code that generates the visualisations to determine the optimal k value.

In Figure 27 above, the code helps determine the ideal number of clusters (k) in a K-means clustering issue by creating two subplots: one for the Elbow Method and one for the Silhouette Scores. The visualisations are saved as a file for additional reporting or analysis.

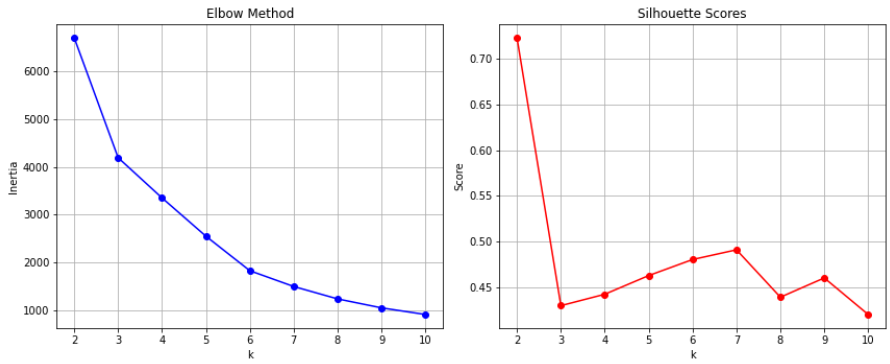


Figure 28: Elbow method and Silhouette score plots for determining optimal k.

In Figure 28, is the output of the code that defines the optimal k value. The Elbow Method (left) helps determine where adding more clusters delivers decreasing results by showing how inertia reduces, as the number of clusters increases. This is enhanced by the Silhouette Scores (right), which displays the average quality of clustering across various k values. Better-defined clusters are indicated by higher scores.

Using these techniques, the optimal number of clusters was determined to be k-5, which provides a fair balance between interpretability and detail. The following parameters were used to create the final K-means model: `n_clusters=5`: Number of clusters; `random_state=42`: for reproducibility; `n_int=10`: the number of times the algorithm is executed using various centroid seeds. To determine the main pickup hotspots in New York City, the generated clusters were spatially visualised.

```
# Visualize spatial clusters
plt.figure(figsize=(12, 8))
colors = plt.cm.viridis(np.linspace(0, 1, optimal_k))
for i in range(optimal_k):
    cluster = df[df['cluster_kmeans'] == i]
    plt.scatter(cluster['Lon'], cluster['Lat'], s=10, c=[colors[i]], label=f'Cluster {i}', alpha=0.5)
plt.title('K-means Spatial Clustering')
plt.xlabel('Longitude')
plt.ylabel('Latitude')
plt.grid(True)
plt.legend()
plt.savefig(f'{visual_dir}/kmeans_clusters.png')
plt.show()
plt.close()
```

Figure 29: Code that generates the K-means Spatial Clustering Visualisation.

This code assigns points to optimal_k clusters by using K-means clustering on geographic data. The output is saved as an image file, after each cluster is labelled and plotted by latitude and longitude values on a map.

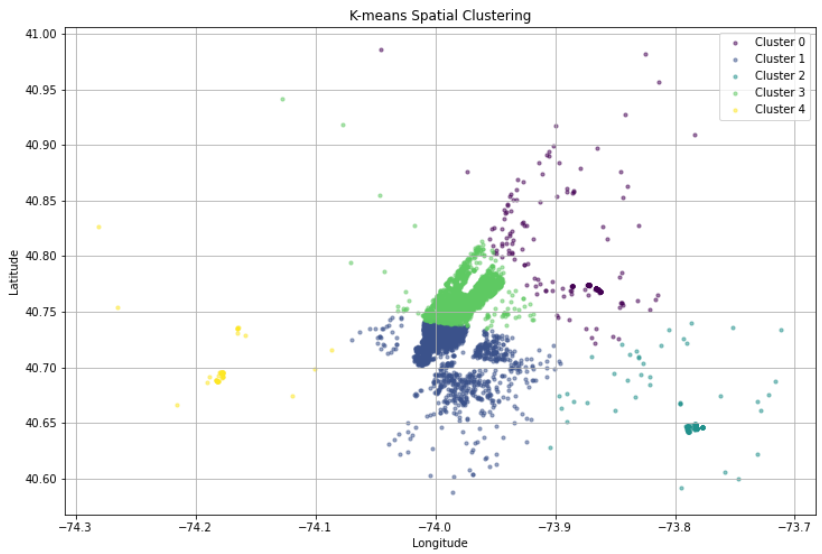


Figure 30: K-means spatial clustering visualization showing 5 clusters.

In Figure 30, is the visualization of K-means spatial clustering which displays five clusters. The five k-means identified clusters are displayed in this scatter plot, each of which is represented by a distinct colour. When the geographic coordinates are compared to actual maps of NYC, these groups represent different regions: Cluster 0 (purple) is concentrated in what

appears to be Midtown Manhattan, Cluster 1 (blue) is associated with the Lower Manhattan/Financial District area, Cluster 2 (teal) is associated with parts that could be Brooklyn/Queens, Cluster 3 (light green) is associated with Upper Manhattan/Upper East Side area, and Cluster 4 (yellow) is associated with areas that align with major transportation hubs, which may include airports.

Using the following parameters, DBSCAN was implemented: `eps=0.1`: the maximum distance between two samples that allows one to be considered in the neighbourhood of the other; `min_samples=15`: the minimum number of samples in a neighbourhood for a point to be considered a core point. During parameter tuning, several `eps` values (ranging from 0.05 to 0.2) were examined; ` eps=0.1`, was chosen, as it offered the best trade-off between minimising the number of noise points and detecting significant clusters. As suggested by (Fong et. al., 2014), these parameters were chosen according to the density of the data and the scale of the standardised spatial features.

```
# Implements DBSCAN clustering
print("\n DBSCAN Clustering")

dbscan = DBSCAN(eps=0.1, min_samples=15)
df['cluster_dbscan'] = dbscan.fit_predict(X_spatial_scaled)

# Visualize DBSCAN custers
plt.figure(figsize=(12, 8))
labels = sorted(df['cluster_dbscan'].unique(), key=lambda x: (x == -1, x))
colors = plt.cm.viridis(np.linspace(0, 1, len(labels)))

# Creates a custom order for the legend
legend_handles = []
legend_labels = []

# Plots all clusters to get the visualization
for i, label in enumerate(labels):
    cluster = df[df['cluster_dbscan'] == label]
    color = 'black' if label == -1 else colors[i]
    scatter = plt.scatter(cluster['Lon'], cluster['Lat'], s=10, c=[color],
                          label=f'Cluster {label}', alpha=0.5)
    # Store handle and label for ordered legend
    legend_handles.append(scatter)
    legend_labels.append(f'Cluster {label}')

plt.title('DBSCAN Spatial Clustering')
plt.xlabel('Longitude')
plt.ylabel('Latitude')
plt.grid(True)

# Create legend with custom order
plt.legend(handles=legend_handles, labels=legend_labels)
plt.savefig(f'{visual_dir}/dbscan_clusters.png')
```

Figure 31: Code that generates the DBSCAN Spatial Clustering Visualization.

Using DBSCAN clustering on geographic data, the visualization code in Figure 31 groups points according to density (eps=0.1, min_samples = 15. It saves the generated visualisation and uses a colour gradient to visualise the clusters, with noise points (cluster -1) highlighted in black.

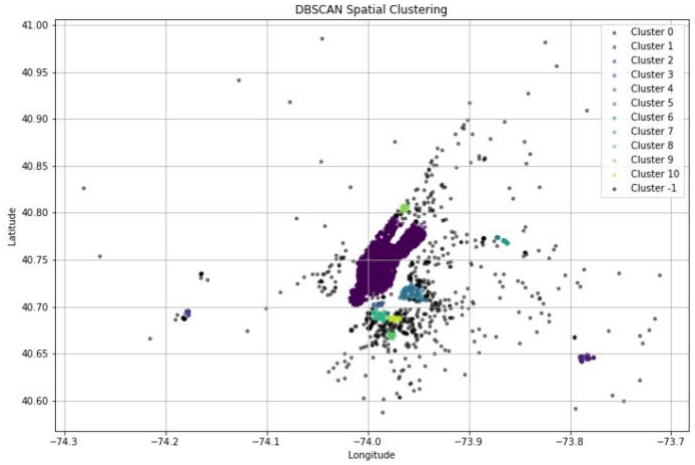


Figure 32: DBSCAN spatial clustering visualization showing clusters and noise points.

This scatter plot displays the DBSCAN-identified clusters, with the grey dots representing noise points (designated as -1). DBSCAN automatically detects isolated points as noise and counts the number of clusters.

To understand the temporal features of each spatial cluster, heatmaps displaying pickup patterns by hour of the day, and day of the week, were split by month. This approach enabled a thorough examination of the data's temporal and spatial trends.

```
# Creates temporal heatmaps for each cluster to analyze time patterns
print("\n Heatmaps by Cluster")
for cluster in range(optimal_k):
    plt.figure(figsize=(12, 10))
    for idx, month in enumerate(['April', 'May']):
        plt.subplot(2, 1, idx + 1)
        subset = df[(df['cluster_kmeans'] == cluster) & (df['month'] == month)]
        pivot = pd.crosstab(subset['day_of_week'], subset['hour'])
        sns.heatmap(pivot, cmap='coolwarm', cbar=True)
        plt.title(f'Cluster {cluster} - {month}')
        plt.xlabel('Hour')
        plt.ylabel('Day of Week')
        plt.yticks(ticks=range(7), labels=['Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat', 'Sun'], rotation=0)
    plt.tight_layout()
    plt.savefig(f'{visual_dir}/cluster_{cluster}_heatmap_by_month.png')
    plt.show()
    plt.close()

print("\nClustering analysis complete! Results saved in 'visualizations' folder.")
```

Figure 33: Code that generates temporal heatmaps for each cluster to analyse temporal patterns.

In Figure 33, is the code that generates the Heatmaps by clusters visualization by hour, based on the day of the week. The following figures are the outputted visualizations that were created.

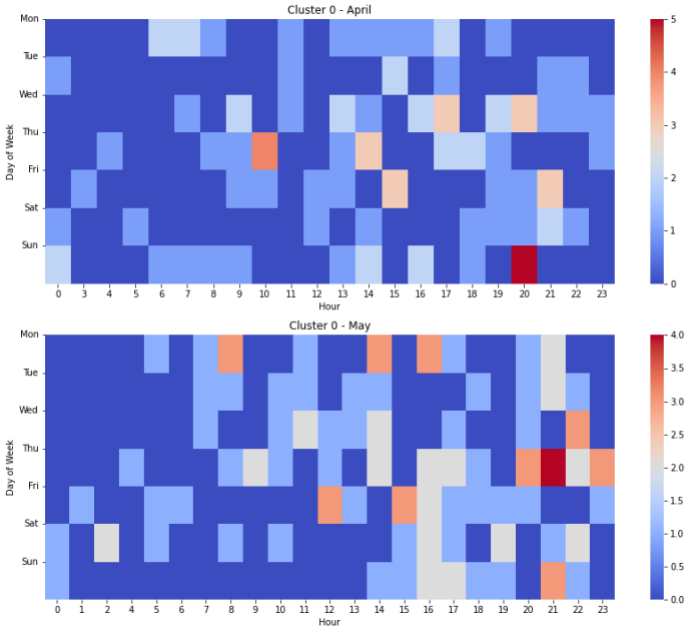


Figure 34: Temporal heatmap for Cluster 0 showing patterns by hour, day and month.

The temporal trends for Cluster 0 (Midtown Manhattan) are displayed in this heatmap visualisation for April (top) and May (bottom) by hour and day of the week. Strong weekday trends can be seen in this cluster, with peaks occurring during evening rush hours (17-19 PM), especially Tuesday – Thursday. Compared to April, May exhibits more evening activity than in April.

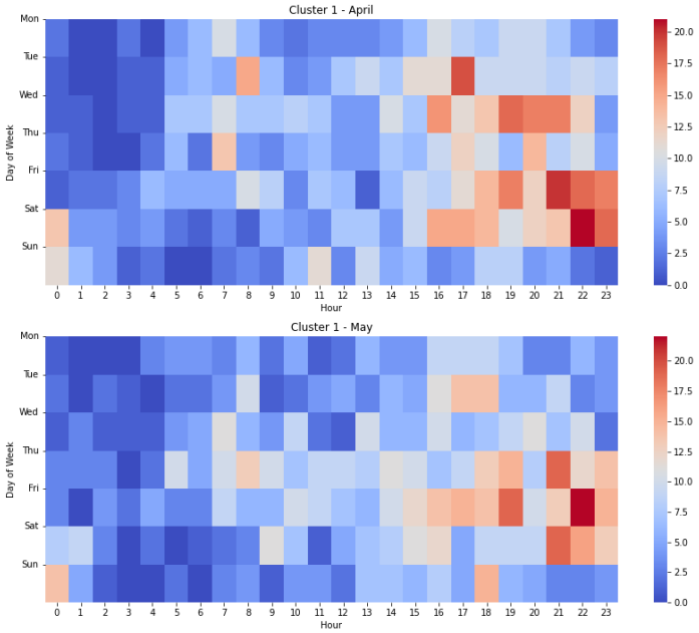


Figure 35: Temporal heatmap for Cluster 1 showing patterns by hour, day and month.

The temporal patterns for Cluster 1 (Lower Manhattan/ Financial District) are displayed in these heatmap visualization by day of week and hour. The cluster has distinct patterns of business hours, with weekdays (9 AM – 6PM), having the most activity. The patterns between April (top) and May (bottom) are quite comparable, indicating that Uber pickups are frequently used in the business district.

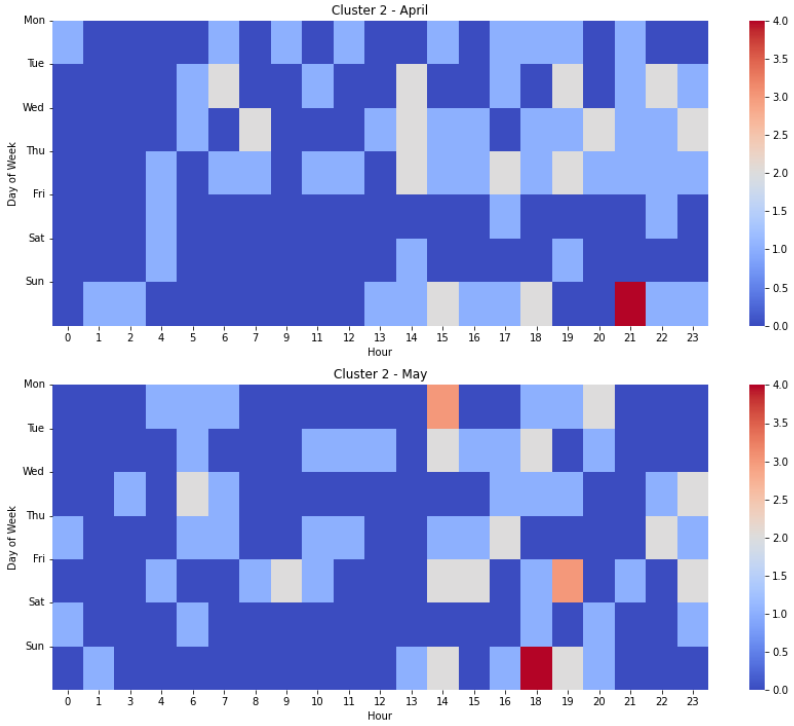


Figure 36: Temporal heatmap for Cluster 2 showing patterns by hour, day and month.

The temporal trends for Cluster 2 (Upper Manhattan/Upper East Side) are displayed in these heatmaps by hour and day of the week. The activity in this cluster is more evenly distributed throughout the day, with noticeable weekend activity, especially in the evenings. Compared to April, May had more weekend activity, especially on Friday and Saturday evenings.

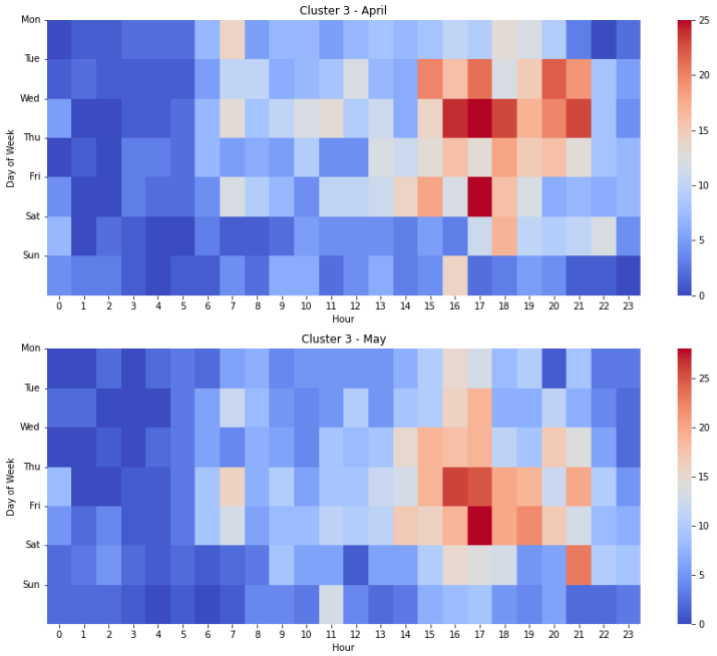


Figure 37: Temporal heatmaps for Cluster 3 showing patterns by hour, day and month.

The temporal trends for Cluster 3 (Outer Boroughs, which includes portions of Brooklyn and Queens) are displayed in these heatmaps by hour and day of the week. Compared to other clusters, this one is busier in the early mornings (6 – 7 AM) and late evenings (15 – 19 PM). Although, the volume is a little higher in May, the trends are similar in April and May.

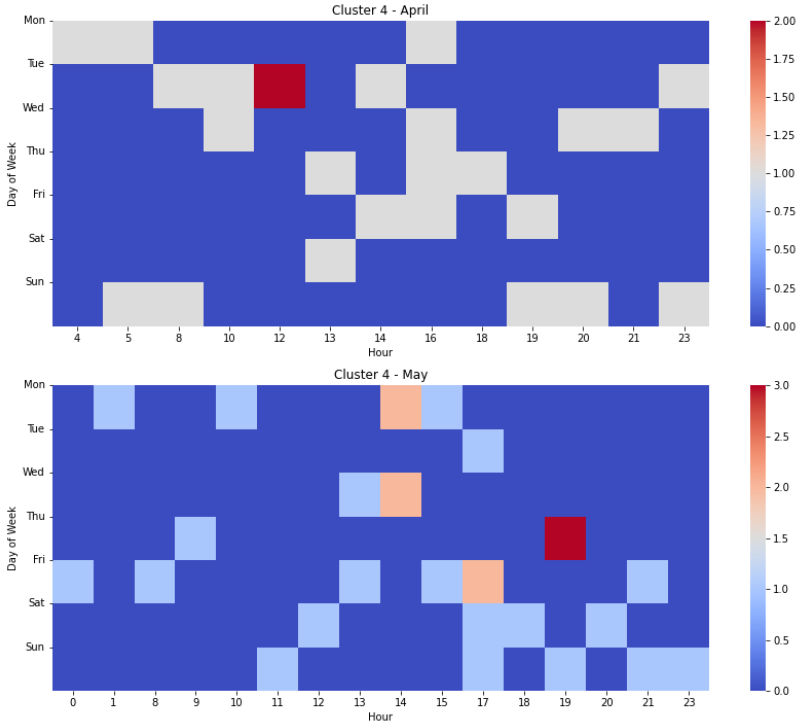


Figure 38: Temporal heatmap for Cluster 4 showing patterns by hour, day and month.

The temporal patterns for Cluster 4 (Transportation Hubs, including airports) are displayed by hour and day of the week in these heatmaps. This cluster exhibits scattered activity with distinct hourly peaks that most likely correspond to the normal arrival and departure times of flights. More travel over the spring may have contributed to the greater overall volume in May, especially on Fridays and Sundays.

The visualization of Cluster 4's heatmap shows a sparse, event-driven pattern that has a stark contrast to the regular patterns in the other clusters. This suggests that the area serves a different urban role, most likely as a transportation hub, where activity is driven by certain events, such as the arrival of flights.

Clustering Strategy and Parameter Optimization:

The implementation of both clustering techniques required several important decisions and parameter tuning. For K-means, the parameters were: `n_clusters=5`: The elbow method and silhouette scores showed that, when evaluating values ranging from 2 to 10, five clusters offered the optimal balance between detail and interpretability; `random_state=42: This seed value guarantees that the outcomes can be reproduced; `n_int=10`: runs the algorithm multiple times with different centroid seeds to avoid poor clustering results and improve cluster stability. According to (Sharma, 2025), the K-means technique operates in the following order; initialises k centroids using the k-means ++ or at random, assigns each data point to the closest centroid, recalculates centroids using the mean of all points assigned to each cluster; and steps (2 – 3) are repeated, until convergence (centroids no longer change significantly). K-means sufficiently identified five different pickup spots in NYC using the Uber dataset, each with its own specific temporal and spatial features.

The parameters for DBSCAN were: `eps=0.1`: it specifies the maximum distance between two points, which means that one point should be considered to be in close proximity to the other; `min_samples=15`: it defines the minimum number of points needed to create a dense region (core point), and the value was chosen based on the scale of the standardised spatial features. The value was also selected, to strike a balance between finding significant clusters and preventing excessive fragmentation. According to (Thailappan, 2024), DBSCAN operates by: 1): identifying core points that have at least `min_samples` points with a distance of `eps`; 2): clusters are formed by connecting core points within a range of `eps`; 3): assigning non-core

points that are within `eps` distance of a core point as border points; 4): labelling non-core or border points as noise (-1). In the Uber dataset, the DBSCAN technique found noise points that represented isolated pickups, as well as several clusters of various sizes and shapes.

The sampling plan was created to preserve computing efficiency and provide a fair comparison of the patterns in April and May. A balanced sample of 5,000 records, 2,500 for each month was utilised. This sampling strategy reduced computational requirements, while maintaining the representative patterns of the data. For feature selection, the main clustering analysis concentrated on spatial features (latitude and longitude) to find geographical hotspots, even though three different feature sets (spatial only, spatial and temporal, and spatial, temporal and month) were scaled during the preprocessing stage. To determine when pickups take place in various locations, the temporal patterns within each spatial cluster were then examined.

Evaluation Results Explanation and Interpretation:

The clustering findings were assessed using qualitative analysis, as well as quantitative measures. The degree to which the clusters are well-separated was assessed quantitatively using silhouette scores, which ranges from -1 to 1 (higher is better). A comparison of silhouette scores between K-means and DBSCAN provides information about the quality of the clustering (Prasad et. al., 2017).

```
# Create legend with custom order
plt.legend(handles=legend_handles, labels=legend_labels)
plt.savefig(f'{visual_dir}/dbscan_clusters.png')

# Evaluates clustering quality using silhouette scores
print("\n Evaluation")
scores = {
    'K-means': silhouette_score(X_spatial_scaled, df['cluster_kmeans']),
    'DBSCAN': silhouette_score(X_spatial_scaled[df['cluster_dbscan'] != -1], df['cluster_dbscan'][df['cluster_dbscan'] != -1])
    if len(set(df['cluster_dbscan'])) > 1 and -1 in df['cluster_dbscan'].values else 0
}

# Plot visualisation comparison
plt.figure(figsize=(8, 6))
plt.bar(scores.keys(), scores.values(), color=['blue', 'red'])
plt.title('Silhouette Score Comparison')
plt.ylabel('Score')
for i, score in enumerate(scores.values()):
    plt.text(i, score + 0.01, f'{score:.3f}', ha='center')
plt.ylim(0, max(scores.values()) * 1.2)
plt.grid(True, axis='y')
plt.savefig(f'{visual_dir}/silhouette_comparison.png')
```

Figure 39: Code that generates comparison of Silhouette scores.

In Figure 39, is the code that assesses the performance of the K-means and DBSCAN clustering techniques by calculating and comparing their silhouette scores.

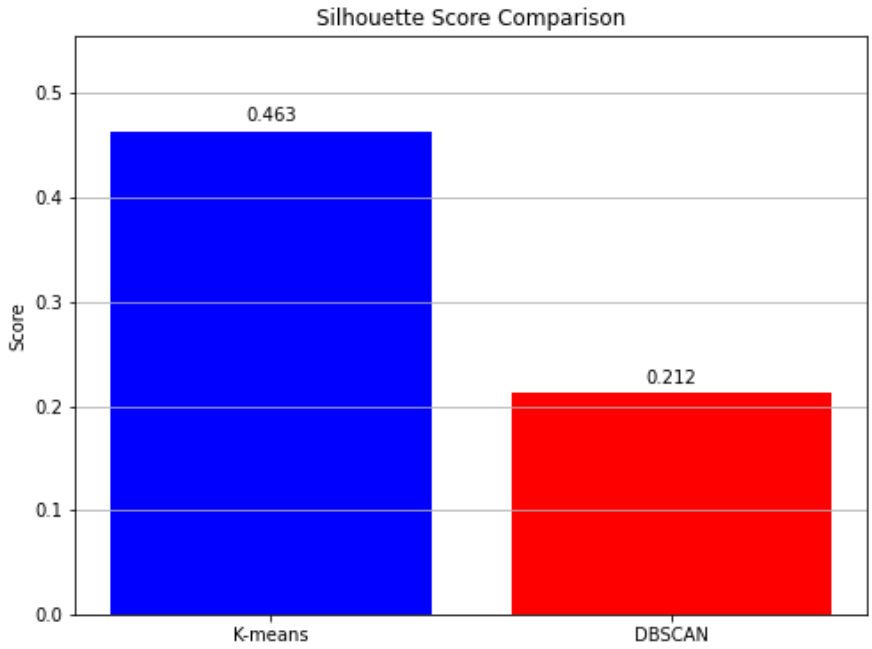


Figure 40: Bar chart comparing silhouette scores for K-means and DBSCAN.

The bar graph in Figure 40 compares silhouette scores, with (0.463) K-means provides better defined clusters for the dataset, as opposed to DBSCAN with (0.212), excluding the noise points. Nevertheless, both algorithms produced positive silhouette scores, which indicates that the data had significant cluster structures.

The five K-means clusters that were identified for qualitative analysis indicates different NYC pickup locations, each with their own distinct features: Cluster 0: Manhattan's Midtown – the geographic centre: is around 40.75°N, -73.98°W; its features: include many business activities, tourist attractions and shopping areas; temporal patterns: the highest weekday peaks occur during rush hours in the morning (8-9 AM) and evening (5-7 PM). Comparing the months, May had a higher level of evening activity than April. Cluster 1: Lower Manhattan/Financial District - the geographic location: is around 40.71°N, -74.01°W; its features: include government buildings, business and financial hubs; its temporal patterns: is focused on weekdays between (9 AM – 6 PM) for business; Comparison of months: April and May have very similar patterns.

Cluster 2: Upper Manhattan/Upper East Side – its geographic location is approximately around 40.78°N, -73.95°W; its characteristics: includes proximity to Central Park, residential areas and museums; temporal patterns: weekend activities are more equally distributed throughout the day. Compared to April, May had more weekend activity. Cluster 3: Outer Boroughs (parts

of Brooklyn and Queens) – its geographic centre is around 40.69°N, -73.93°W; its features include residential areas and districts with excessive nightlife activities; temporal patterns – increased activity in the early morning and late nights; Comparative analysis by month: May showed slightly higher volume, but similar patterns, compared to April. Cluster 4: Transportation Hubs (including airports) – its geographic centre: includes several locations, such as the JFK and LaGuardia regions; its features: include major transportation stations and airports; temporal patterns: consists of peaks that correspond to the usual arrival and departure times of flights. In comparison to April, May had a larger volume overall, especially on Fridays and Sundays.

Clusters 0 and 1 exhibit distinct business-hour patterns, whereas Clusters 2 and 3 exhibit more evening and weekend activity. The temporal heatmaps for each cluster indicates different patterns in pickup times. These patterns support the anticipated use of Uber in different city regions.

The DBSCAN clustering technique revealed more information: Core Urban locations – the densest pickup locations in Manhattan, especially those surrounding Midtown and Lower Manhattan, were successfully identified using DBSCAN; Noise Points – a small amount (about 18%) of noise points were categorised as noise (-1), signifying irregular pickups in less dense areas; Irregular Cluster Shapes – in contrast to the circular structures of K-means, DBSCAN identified irregularly shaped clusters that better follow the natural geography of New York City, such as around parks and major avenues; Month Comparison – the distribution of data points in April and May within DBSCAN clusters was similar, indicating consistent spatial patterns in both months.

The temporal analysis of clusters showed significant patterns in pickup times across various parts of the city. Hourly patterns: Early Morning (3 – 5 AM): lowest overall activity, with relatively greater proportions in Cluster 4 (transportation Hubs); Morning Rush Hour (7 – 9 AM): dominated pickups in Clusters 0 and 1 (Midtown and Lower Manhattan), with greater patterns being identified on weekdays; Midday (11 AM – 2 PM): more evenly distributed across clusters; Evening Rush Hour (5 – 7 PM): highest activity in Clusters 0, 1, and 2; Late Night (10 PM – 2 AM): has the most activity in Cluster 3 (certain areas of Brooklyn and Queens. Day of Week patterns: Weekdays (Monday – Friday): Clusters 0 and 1 dominated, indicating business activity; Friday – displayed distinct patterns with activities on both the

weekend and business days; Weekend (Saturday – Sunday): Clusters 2 and 3 had higher relative activity; Sunday night – Cluster 4 had increased activity, probably because of airport traffic.

Month-to-Month Comparison: according to the visualization displayed in Figure 25, May has a larger pickup volume than April. According to the daily distribution charts displayed in Figure 24, weekend activity seems to have increased the most between April and May; the hourly heatmaps in Cluster 2 revealed a slight shift towards later evening activity in May, but overall, the temporal patterns were consistent across the months. The heatmap of cluster 2 showed higher weekend activity in May than in April. These temporal patterns reveal important information for service optimisation, surge pricing methods and driver allocation. For instance, to satisfy anticipated demands, drivers can be directed to various locations, depending on the time of day and day of week.

Part 4: Comparative Analysis of AI Techniques:

Although, K-means and DBSCAN were effective in analysing the Uber pickup data, several clustering techniques each with their own unique advantages and disadvantages could have been used. An example of a technique that could have been used is Hierarchical Clustering. This technique creates a tree of clusters (dendrogram) by either starting with a single cluster and splitting it (divisive) or starting with all points as separate clusters and combining them (agglomerative), according to (Aggarwal, 2013). These are the benefits of the technique, compared to the two that were used: it offers a hierarchical framework that illustrates the relationships between clusters at various levels; it does not need the number of clusters to be determined; it can also be represented as a dendrogram to assist in identifying the right number of clusters. However, there are several limitations to this technique, when compared to the ones that were used in the clustering analysis: it is less appropriate for huge datasets, because of its higher processing cost ($O(n^2)$ or worse), it is less effective for the Uber dataset's millions of records, especially when the other months are added into the analysis. Also, depending on how the distance between clusters are determined, the results may vary, based on how the choice of linkage criteria (single, complete, or average) affects how the algorithm groups the data.

A modification of DBSCAN, OPTICS (Ordering Points to Identify the Clustering Structure), addresses some of its limitations by generating an ordering of points that represents the density-

based clustering structure (Doran et. al., 2019). Unlike DBSCAN which has a global density threshold, OPTICS can detect clusters with different densities; it creates a reachability plot to help with the visualization of the cluster structure and doesn't need a defined `eps` value. However, it is limited, because it has greater computational requirements, it is more difficult to understand and implement. Also, the relatively consistent density of urban pickup data may not produce benefits related to the added complexity.

Mean Shift Clustering is a non-parametric clustering technique that uses the mean of points inside a region to update candidates for centroids (Boisberranger et. al., 2025). Its advantages compared to the techniques that were used include: it doesn't need the number of clusters to be predetermined; it is less sensitive to outliers than K-means; and is able to identify clusters of any shape. However, its limitations are that it is computationally demanding when dealing with massive datasets and it can be difficult to choose the bandwidth parameter; hence, the extensive Uber dataset might converge slowly.

The Gaussian Mixture Models (GMM) employs the Expectation-Maximization algorithm to determine the parameters of these distributions, assuming that the data points are produced from a combination of several Gaussian distributions (Boisberranger et. al., 2025). Its advantage over the used techniques includes that it provides soft assignments (probabilities) of points to clusters, is more adaptable to the size and shapes of clusters than K-means, can represent cluster assignment uncertainty. However, it is less efficient than the techniques used, as it requires the number of components to be specified (like K-means); requires more computing than K-means. Also, it might have too many components, which could cause an overfitting and, it is less obvious for geographic data, as it doesn't assign points to just one group.

Several factors led to the selection of K-means and DBSCAN as the best techniques for this dataset. Their results are more reliable when compared to other studies because both techniques are well-established in the research and have recognised attributes and limitations. K-means offers distinct, comprehensible clusters with well-defined centroids, whereas DBSCAN is better at handling noise and irregular shapes. They make a great pairing for computational efficiency in handling the millions of records in the Uber dataset as both techniques are capable of processing large datasets efficiently. Furthermore, the pair provides more detailed insight, compared to when either technique is used alone and the resulting clusters from both techniques

are easily to interpret clearly in the context of NYC geography and transportation patterns. Ultimately, both clustering techniques are well-suited for spatial clustering, which was the main objective of the analysis.

Conclusion:

In conclusion, K-means and DBSCAN clustering techniques were utilised in this report to analyse Uber pickup trends in New York City. Five different clusters were identified, each with its own temporal features and representing various functional regions of the city. Without any prior knowledge of city geography, the algorithm's ability to recognise functionally distinct urban zones, is demonstrated by the distinct temporal signature of Cluster 4, when compared to residential and business districts. This highlights the potential of unsupervised learning, when uncovering significant patterns in urban mobility data. The combined capabilities of K-means (well-defined clusters with centroids) and DBSCAN (managing noise and irregular forms) provides a detailed understanding of pickup hotspots. These results have real-world implications for better urban transportation planning, surge pricing strategy implementation and driver allocation optimisation.

References:

- Aggarwal, C.C. (2013) *DATA CLUSTERING Algorithms and Applications*, *ResearchGate*. Available at: https://www.researchgate.net/publication/331534089_DATA_CLUSTERING_Algorithms_and_Applications (Accessed: 16 April 2025).
- Ankita (2025) *K-means: Getting the optimal number of clusters*, *Analytics Vidhya*. Available at: https://www.analyticsvidhya.com/blog/2021/05/k-mean-getting-the-optimal-number-of-clusters/#h-what-is-silhouette-score-k-means (Accessed: 16 April 2025).
- Boisberranger, J. du (2025) *2.3. clustering*, *scikit learn*. Available at: https://scikit-learn.org/stable/modules/clustering.html (Accessed: 16 April 2025).
- Doran, D. (2019) *dbscan : Fast Density-Based Clustering with R*, *ResearchGate*. Available at: https://www.researchgate.net/publication/336920948_dbscan_Fast_Density-Based_Clustering_with_R (Accessed: 16 April 2025).

- Ester, M. (2017) *DBSCAN Revisited, Revisited: Why and How You Should (Still) Use DBSCAN*, *ACM Digital Library*. Available at: https://dl.acm.org/ (Accessed: 16 April 2025).
- FiveThirtyEight (2020) *Uber Pickups in New York City*, *Kaggle*. Available at: https://www.kaggle.com/datasets/fivethirtyeight/uber-pickups-in-new-york-city (Accessed: 16 April 2025).
- Fong, S. (2014) *DBSCAN: Past, present and future*, *ResearchGate*. Available at: https://www.researchgate.net/publication/262116837_DBSCAN_Past_present_and_future (Accessed: 16 April 2025).
- Kassambara, A. (2017) *Practical Guide To Cluster Analysis in R*, *xsliulab.github.io*. Available at: https://xsliulab.github.io/Workshop/2021/week10/r-cluster-book.pdf (Accessed: 16 April 2025).
- McGregor, M. (2020) *8 Clustering Algorithms in Machine Learning that All Data Scientists Should Know*, *freeCodeCamp*. Available at: https://www.freecodecamp.org/news/8-clustering-algorithms-in-machine-learning-that-all-data-scientists-should-know/ (Accessed: 16 April 2025).
- Prasad, M. (2017) *A review of clustering techniques and developments*, *ScienceDirect*. Available at: https://www.sciencedirect.com/science/article/pii/S0925231217311815 (Accessed: 16 April 2025).
- Sharma, P. (2025) *K-Means Clustering Algorithm*, *Analytics Vidhya*. Available at: https://www.analyticsvidhya.com/blog/2019/08/comprehensive-guide-k-means-clustering/#Understanding_the_Different_Evaluation_Metrics_for_Clustering (Accessed: 16 April 2025).
- Thailappan, D. (2024) *Understand The DBSCAN Clustering Algorithm!*, *Analytics Vidhya*. Available at: https://www.analyticsvidhya.com/blog/2021/06/understand-the-dbscan-clustering-algorithm/ (Accessed: 16 April 2025).
- Zubair, Md. (2022) *An Improved K-means Clustering Algorithm Towards an Efficient Data-Driven Modeling*, *SpringerLink*. Available at: https://link.springer.com/article/10.1007/s40745-022-00428-2 (Accessed: 16 April 2025).