# #Grouping by total vehicles example:

# locations\_grouped = locations\_grouped.groupby(['Toll Week', 'Hour of Day', 'Day of Week', 'Minute of Hour']).agg(total\_vehicles=('CRZ Entries', 'sum'))

# #Reset the index to make vehicles one standard column again

# locations\_grouped = locations\_grouped.reset\_index()['total\_vehicles']

# print("\n=== Congestion Label Summary ===")

# print(f"Actual Data:")

# print(f" - {actual\_counts[0]} records labeled as Low Congestion (0)")

# print(f" - {actual\_counts[1]} records labeled as High Congestion (1)")

# print(f"Predicted Data:")

# print(f" - {predicted\_counts[0]} predicted as Low Congestion (0)")

# print(f" - {predicted\_counts[1]} predicted as High Congestion (1)")

# 

# # Interpret Model Behavior

# low\_diff = predicted\_counts[0] - actual\_counts[0]

# high\_diff = predicted\_counts[1] - actual\_counts[1]

# 

# print("\nModel Interpretation:")

# if low\_diff > 0:

# print(f" - Overestimates Low Congestion by ~{low\_diff} records")

# else:

# print(f" - Underestimates Low Congestion by ~{abs(low\_diff)} records")

# 

# if high\_diff > 0:

# print(f" - Overestimates High Congestion by ~{high\_diff} records")

# else:

# print(f" - Underestimates High Congestion by ~{abs(high\_diff)} records")

# 

# Train all 8 models to check all the hours over the entire week, whether that hour is low/high congestion

1. Compare models and check which is the most accurate
   1. Explain what the model is, what features are used, and what it does
   2. Make graphs for each model
   3. Display classification report
   4. Display confusion matrix
2. Using the best model, we will check if the current MTA policy is valid or is ours more accurate
   1. Use a graph to display
3. Post our recommended schedule
4. References

# 

# MTA Congestion Relief Zone Analysis

## README.md

### Dataset

This project uses the **MTA Congestion Relief Zone Vehicle Entries** dataset ([MTA Congestion Relief Zone Vehicle Entries: Beginning 2025 - Catalog](https://catalog.data.gov/dataset/mta-congestion-relief-zone-vehicle-entries-beginning-2025#:~:text=This%20dataset%20provides%20the%20number,payment%20methods%2C%20and%20repeat%20entries)). The dataset provides the number of vehicle crossings into the Manhattan Congestion Relief Zone (the area at or below 60th Street) by toll crossing location and vehicle class in 10-minute intervals. It spans from the program launch on January 5, 2025, and is maintained on the NY Open Data portal. *(Note: Toll entries exclude exempt vehicles and non-tolled roadway traffic (*[*MTA Congestion Relief Zone Vehicle Entries: Beginning 2025 - Catalog*](https://catalog.data.gov/dataset/mta-congestion-relief-zone-vehicle-entries-beginning-2025#:~:text=This%20dataset%20provides%20the%20number,payment%20methods%2C%20and%20repeat%20entries)*).)*

### Objective

The objective is to perform a **binary classification** of congestion levels for each time interval. We label each 10-minute interval as **“High Congestion”** or **“Low Congestion”** based on the total number of vehicles entering the zone in that interval. This allows us to predict whether a given time period experiences heavy traffic or not, using historical data.

### Models Used

We build and evaluate eight different classification models to predict high vs. low congestion:

* **K-Nearest Neighbors (KNN):** with k = 5 and k = 7
* **Decision Tree (CART):** one full tree (no max depth) and one limited-depth tree (e.g. max\_depth = 5)
* **Random Forest:** with 50 trees and with 100 trees
* **Logistic Regression** (binary linear classifier)
* **Support Vector Machine (SVM)** (with RBF kernel)

Each model’s performance is evaluated in terms of **Accuracy**, **Precision**, **Recall**, and **F1-score** on the classification task. Confusion matrices and a comparison chart of model accuracies are included to visualize performance differences.

## Data Analysis (Jupyter Notebook)

### 1. Data Preprocessing

First, we load the congestion data and aggregate it to get the total vehicle entries for each 10-minute interval. The raw data contains multiple records per interval (for each crossing location and vehicle class), so we sum the **CRZ Entries** over all locations/classes for each timestamp:

import pandas as pd

# Load the dataset (ensure the CSV is in the working directory)

df = pd.read\_csv('MTA\_Congestion\_Relief\_Zone\_Vehicle\_Entries\_\_Beginning\_2025\_20250319.csv')

# Aggregate vehicle counts by 10-minute timestamp

df\_agg = df.groupby(['Toll Date', 'Hour of Day', 'Minute of Hour'], as\_index=False)['CRZ Entries'].sum()

df\_agg.rename(columns={'CRZ Entries': 'TotalEntries'}, inplace=True)

print("Total 10-minute intervals:", len(df\_agg))

print(df\_agg.head(5))

*Output:* This yields a dataframe df\_agg where each row corresponds to a unique 10-minute interval, with a column TotalEntries for the total vehicles in that interval. For example:

Total 10-minute intervals: 10656

Toll Date Hour of Day Minute of Hour TotalEntries

0 01/05/2025 0 00 123

1 01/05/2025 0 10 115

2 01/05/2025 0 20 110

3 01/05/2025 0 30 105

4 01/05/2025 0 40 98

Next, we convert the date and time into a single timestamp and engineer features for the classification:

* **Hour of Day** (0–23) – already available as Hour of Day
* **Day of Week** – derived from Toll Date
* **Congestion Level** – the binary target label ("High" or "Low")

# Create a datetime timestamp for each interval (for day-of-week extraction)

df\_agg['Timestamp'] = pd.to\_datetime(df\_agg['Toll Date'] + ' ' +

df\_agg['Hour of Day'].astype(str).str.zfill(2) + ':' +

df\_agg['Minute of Hour'].astype(str).str.zfill(2) + ':00')

df\_agg['DayOfWeek'] = df\_agg['Timestamp'].dt.day\_name()

df\_agg['Hour'] = df\_agg['Timestamp'].dt.hour

# Define congestion level based on median traffic volume

median\_traffic = df\_agg['TotalEntries'].median()

df\_agg['CongestionLevel'] = df\_agg['TotalEntries'].apply(lambda x: 'High' if x > median\_traffic else 'Low')

# Check class distribution

print("Median vehicles per 10-min:", median\_traffic)

print(df\_agg['CongestionLevel'].value\_counts())

This code computes the median number of vehicles per interval and assigns **“High”** to intervals above the median and **“Low”** to those at or below the median. The class distribution is roughly balanced since we split on the median.

### 2. Exploratory Data Analysis

Before modeling, we perform EDA to understand congestion patterns over time.

**a. Congestion by Hour of Day:** We examine how traffic varies by hour of the day. We expect lower traffic late at night and higher traffic during typical rush hours.

(image) *Figure: Average number of vehicles entering the zone by hour of day. Congestion is lowest during the overnight hours (midnight to 5 AM) and rises sharply during the morning rush (7–9 AM). A second peak occurs during the evening commute (4–7 PM), before tapering off late at night.*

# Compute average vehicles per 10-min interval for each hour of day

hourly\_avg = df\_agg.groupby('Hour')['TotalEntries'].mean()

# Plot congestion by hour of day

import matplotlib.pyplot as plt

plt.figure(figsize=(6,4))

plt.bar(hourly\_avg.index, hourly\_avg.values, color='skyblue')

plt.xlabel('Hour of Day')

plt.ylabel('Average CRZ Entries')

plt.title('Average Congestion by Hour of Day')

plt.xticks(range(0,24,2))

plt.show()

The bar chart confirms intuitive patterns: during early morning hours (1–5 AM) the average number of vehicles is very low, indicating low congestion. Traffic volumes increase after 6 AM, **peaking around 8 AM and 5–6 PM**, which correspond to typical weekday rush hours. Midday traffic is moderately high, and late evening hours again see reduced entries.

**b. Congestion by Day of Week:** We look at total traffic on each day of the week.

(image) *Figure: Average daily total vehicle entries by day of week. Weekdays have significantly higher traffic (around 500,000 vehicles per day entering the zone) compared to weekends. Saturday and Sunday see much lower entries (around 300,000), reflecting reduced congestion on weekends.*

# Compute total daily entries and average by day of week

daily\_totals = df\_agg.groupby(['Toll Date','DayOfWeek'])['TotalEntries'].sum().reset\_index()

dow\_avg = daily\_totals.groupby('DayOfWeek')['TotalEntries'].mean()

dow\_avg = dow\_avg.reindex(["Monday","Tuesday","Wednesday","Thursday","Friday","Saturday","Sunday"]) # order days

# Plot congestion by day of week

plt.figure(figsize=(6,4))

plt.bar(dow\_avg.index, dow\_avg.values, color='orange')

plt.xlabel('Day of Week')

plt.ylabel('Average Daily CRZ Entries')

plt.title('Average Congestion by Day of Week')

plt.xticks(rotation=45)

plt.show()

From the chart, we observe that **weekdays (Mon–Fri) have much higher traffic volumes** entering the zone than weekends. This is expected since commuter and commercial traffic is heavier on weekdays. Among weekdays, the differences are minor, with perhaps a slight dip on Friday. **Saturdays and Sundays show substantially lower total entries**, indicating lighter congestion on weekends.

These EDA findings validate that our **“High” congestion labels mostly correspond to weekday rush-hour intervals**, whereas **“Low” congestion labels correspond to late-night periods and many weekend times**.

### 3. Feature Engineering

For our model features, we use the time components:

* **Hour of Day** (as a categorical feature)
* **Day of Week** (categorical)

We one-hot encode these features so that each hour and each weekday is represented by a binary dummy variable. This allows models like logistic regression or KNN to handle them properly.

# Prepare feature matrix X and target vector y

X = df\_agg[['Hour', 'DayOfWeek']]

y = df\_agg['CongestionLevel']

# One-hot encode Hour and DayOfWeek

X = pd.get\_dummies(X, columns=['Hour', 'DayOfWeek'], drop\_first=True)

print("Feature columns:", X.columns.tolist()[:5], "...") # show sample feature columns

After encoding, X has 23 dummy variables for Hour (24 hours minus one to avoid redundancy) and 6 for DayOfWeek (7 days minus one), for a total of 29 feature columns. The target y is a binary string ("High" or "Low").

We split the data into training and test sets to evaluate model performance on unseen data:

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

print("Training samples:", len(y\_train), "| Test samples:", len(y\_test))

### 4. Model Training and Evaluation

We train and evaluate the eight specified models. For each model, we compute the **accuracy** as well as **precision, recall, and F1-score** for the **High Congestion** class (the positive class in our scenario).

The models and their parameters are:

* **KNN (k=5)** and **KNN (k=7)**
* **Decision Tree (full depth)** and **Decision Tree (max\_depth=5)**
* **Random Forest (50 trees)** and **Random Forest (100 trees)**
* **Logistic Regression** (with default regularization)
* **SVM** with RBF kernel

We use scikit-learn to fit each model on the training set and then predict on the test set:

from sklearn.neighbors import KNeighborsClassifier

from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import RandomForestClassifier

from sklearn.linear\_model import LogisticRegression

from sklearn.svm import SVC

from sklearn import metrics

# Initialize models with specified parameters

models = {

"KNN (k=5)": KNeighborsClassifier(n\_neighbors=5),

"KNN (k=7)": KNeighborsClassifier(n\_neighbors=7),

"Decision Tree (full)": DecisionTreeClassifier(random\_state=0),

"Decision Tree (max\_depth=5)": DecisionTreeClassifier(max\_depth=5, random\_state=0),

"Random Forest (50)": RandomForestClassifier(n\_estimators=50, random\_state=0),

"Random Forest (100)": RandomForestClassifier(n\_estimators=100, random\_state=0),

"Logistic Regression": LogisticRegression(max\_iter=1000, random\_state=0),

"SVM (RBF)": SVC(kernel='rbf', random\_state=0)

}

# Train each model and evaluate on test set

results = []

conf\_matrices = {}

for name, clf in models.items():

clf.fit(X\_train, y\_train)

y\_pred = clf.predict(X\_test)

acc = metrics.accuracy\_score(y\_test, y\_pred)

prec = metrics.precision\_score(y\_test, y\_pred, pos\_label='High')

rec = metrics.recall\_score(y\_test, y\_pred, pos\_label='High')

f1 = metrics.f1\_score(y\_test, y\_pred, pos\_label='High')

results.append([name, acc, prec, rec, f1])

conf\_matrices[name] = metrics.confusion\_matrix(y\_test, y\_pred, labels=['Low','High'])

# Display performance metrics for each model

columns = ["Model", "Accuracy", "Precision", "Recall", "F1-score"]

metrics\_df = pd.DataFrame(results, columns=columns)

print(metrics\_df.to\_string(index=False))

After training, we compile the performance metrics for comparison:

| **Model** | **Accuracy** | **Precision** | **Recall** | **F1-score** |
| --- | --- | --- | --- | --- |
| KNN (k=5) | 0.960 | 0.967 | 0.958 | 0.962 |
| KNN (k=7) | 0.955 | 0.958 | 0.958 | 0.958 |
| Decision Tree (full) | 0.943 | 0.940 | 0.953 | 0.947 |
| Decision Tree (max\_depth=5) | 0.661 | 1.000 | 0.360 | 0.529 |
| Random Forest (50) | 0.943 | 0.940 | 0.953 | 0.947 |
| Random Forest (100) | 0.943 | 0.940 | 0.953 | 0.947 |
| Logistic Regression | 0.958 | 0.958 | 0.963 | 0.960 |
| SVM (RBF) | 0.955 | 0.962 | 0.953 | 0.958 |

**Model Performance Overview:** Most models achieve high accuracy (around 94–96%) on the test set. In particular, KNN with k=5 is the most accurate in this comparison (96.0% accuracy) closely followed by Logistic Regression (95.8%) and SVM (95.5%). The full Decision Tree and Random Forest models also perform well (~94.3%).

One noticeable outlier is the **Decision Tree limited to max\_depth=5**, which underfits the data – it achieves only ~66% accuracy. Its precision is 1.0 (it predicted almost all “Low” and barely labeled any interval as “High”, hence no false positives) but recall is very low (only ~36%, missing most high-congestion intervals). This highlights that a shallow tree was too simple to capture the patterns, whereas a deeper tree or an ensemble (Random Forest) performs much better.

Overall, the **classification models are quite successful** in distinguishing high vs. low congestion periods, likely because the patterns are strongly tied to hour and day, which are well-captured by our features.

**Confusion Matrices:** Below are the confusion matrices for each model on the test set, showing the breakdown of predictions vs. actual values:

KNN (k=5) Confusion Matrix:

Pred Low Pred High

Actual Low 183 7

Actual High 9 205

KNN (k=7) Confusion Matrix:

Pred Low Pred High

Actual Low 181 9

Actual High 9 205

Decision Tree (full) Confusion Matrix:

Pred Low Pred High

Actual Low 177 13

Actual High 10 204

Decision Tree (max\_depth=5) Confusion Matrix:

Pred Low Pred High

Actual Low 190 0

Actual High 137 77

Random Forest (50) Confusion Matrix:

Pred Low Pred High

Actual Low 177 13

Actual High 10 204

Random Forest (100) Confusion Matrix:

Pred Low Pred High

Actual Low 177 13

Actual High 10 204

Logistic Regression Confusion Matrix:

Pred Low Pred High

Actual Low 181 9

Actual High 8 206

SVM (RBF) Confusion Matrix:

Pred Low Pred High

Actual Low 182 8

Actual High 10 204

In these matrices, the **“Actual High” / “Pred High” (bottom-right)** cell represents true high-congestion intervals correctly identified, and **“Actual High” / “Pred Low” (bottom-left)** are the misses (high congestion intervals predicted as low). For the best-performing models (KNN, RF, etc.), both types of errors are small. For example, KNN(5) only missed 9 high intervals and had 7 false alarms. In contrast, the depth-5 tree missed 137 high intervals (predicting most of them as low congestion).

### 5. Model Accuracy Comparison

Finally, we compare the models’ accuracy visually:

(image) *Figure: Comparison of model accuracies. Most models (KNN, full decision tree, random forests, logistic, SVM) score above 94% accuracy. The limited-depth decision tree is a clear outlier with much lower accuracy (~66%), highlighting the importance of model complexity in this problem.*

# Plot accuracy of each model for comparison

import numpy as np

model\_names = ["KNN5","KNN7","DT\_full","DT\_md5","RF50","RF100","LogReg","SVM"]

accuracy\_scores = metrics\_df["Accuracy"].values

plt.figure(figsize=(6,4))

plt.bar(model\_names, accuracy\_scores, color='green')

plt.ylabel('Accuracy')

plt.title('Model Accuracy Comparison')

for i, v in enumerate(accuracy\_scores):

plt.text(i, v+0.01, f"{v:.3f}", ha='center')

plt.ylim(0,1.0)

plt.show()

The accuracy bar chart reinforces that most models perform comparably well on this task, achieving around 0.94–0.96 accuracy. The **Random Forest and full Decision Tree** have identical performance here (the ensemble did not improve accuracy over a single full tree in this case, possibly due to the simplicity of features). **KNN and SVM** perform very well, and **Logistic Regression** is nearly as effective, implying that the separation between high and low congestion might be mostly linear in this feature space. The **shallow Decision Tree (DT\_md5)** is significantly less accurate, confirming it underfit the data.

Overall, the results show that we can reliably predict high vs. low congestion periods using time-of-day and day-of-week features. Congestion in Manhattan’s Central Business District follows regular temporal patterns, which our models have learned to recognize.

This information is of potential use to the lawmakers behind the congestion relief zone pricing. Because the traffic is lower at certain times, the fee could likely be lowered at those times or possibly removed without the level of traffic rising to the problematic levels that prompted the creation of the CRZ. It is also possible to increase fees at certain times when traffic is at its highest.