**Project Progress Report- 3**

# Project summary:

To support future operations across the bus terminals operated by the Port Authority— including the George Washington Bridge Bus Station (GWBBS) and Midtown Bus Terminal (MBT)—we developed a forecasting framework that helps anticipate passenger trends and plan for capacity from 2025 to 2030. By leveraging historical data and incorporating environmental factors, our goal is to provide predictive insights that improve scheduling, staffing, and terminal efficiency.

The project integrates four key models: Prophet, XGBoost Regression, Random Forest Classification, and K-Means Clustering. Prophet forecasts overall monthly passenger volume per terminal, while XGBoost predicts individual bus carrier volumes using weather and time- based features. Random Forest identifies months of high traffic intensity, supporting proactive deployment of resources. K-Means helps group similar terminals or carriers to identify shared demand patterns. These models are chosen for their proven performance in transit and logistics industries and collectively help the Port Authority align its operations with future passenger demand.

# Model 1 : Prophet model used to forecast the passengers in each facility for next 5 years 2025-2030.

Prophet was chosen to forecast monthly passenger volumes at key Port Authority terminals (GWBBS and MBT) due to its strength in modeling trends, seasonality, and disruptions— common in transit data. Developed by Meta, Prophet handles holiday effects and changepoints like COVID-19 impacts with minimal tuning and works well with historical monthly data.

Its forecasts through 2030 help the Port Authority anticipate terminal usage, enabling data- driven decisions on budgeting, staffing, maintenance, and capacity planning. By identifying peak and low-demand periods in advance, the model supports operational efficiency and strategic long-term planning.

# Model 1 :Prophet’s Model In Industrial Applications.

Prophet is widely used across industries for forecasting and operational planning. In transportation, companies like Uber and Lyft use it to predict rider demand by time and region, improving fleet coordination. E-commerce platforms like Amazon and Shopify rely on it to forecast delivery and warehouse demand, especially during peak seasons. Retailers use it for inventory planning, and financial institutions apply it to project trends and manage risk. Prophet’s strengths—handling seasonality, changepoints, and trend shifts—make it ideal for accurate, interpretable forecasting.

### Dependent Variable (y) – Monthly Passenger Volume

* Target: Passenger count at GWBBS and MBT terminals (2005–2024).
* Granularity: Monthly, capturing seasonal and holiday travel trends.
* Purpose: Helps the Port Authority plan staffing, scale services, and optimize terminal operations.

### Independent Variable (ds) – Date

* Format: Monthly date column (timestamp).
* Role: Time is the sole regressor; Prophet detects trends, seasonality, and disruptions (e.g., COVID-19) automatically.
* Advantage: Requires no manual feature engineering or handling of missing timepoints.

## Why This Works

* **Simple & Scalable:** Easy to apply across terminals.
* **Efficient:** Minimal data prep required.
* **Actionable:** Clear forecasts that support strategic planning.

By leveraging time-based trends in ridership, Prophet enables the Port Authority to make informed, proactive decisions about future passenger volumes.

### Python code for forecasting using prophet

The forecasting workflow in Python involved sequential steps including dataset cleaning, training Prophet models, generating future timeframes, predicting passenger volumes, clustering facilities with KMeans, classifying high-traffic periods using Random Forest, and exporting the final combined results

pip install scikit-learn

from prophet import Prophet import pandas as pd

# === Step 2: Prepare Training Data for Prophet ===

df = pd.read\_csv('/Users/jay/Downloads/project\_goal\_1.csv')

# Clean specific passenger count columns

for col in ['gwbbs\_Passengers\_Count', 'mbt\_Passengers\_Count', 'mbtpd\_Passengers\_Count']:

df[col] = df[col].fillna(df[col].median())

df\_cleaned = df.drop\_duplicates() df\_cleaned.sort\_values(by=["Year", "Month"], inplace=True)

# Filter training data up to 2024

df\_train = df\_cleaned[df\_cleaned['Year'] <= 2024].copy()

# Prepare Prophet-format data

def prepare\_prophet\_data(df, column):

df['ds'] = pd.to\_datetime(df['Year'].astype(str) + '-' + df['Month'].astype(str), format='%Y-%m')

return df[['ds', column]].rename(columns={column: 'y'})

# Train Prophet models for each terminal models = {}

for column in ['gwbbs\_Passengers\_Count', 'mbt\_Passengers\_Count', 'mbtpd\_Passengers\_Count']:

terminal = column.split('\_')[0]

prophet\_data = prepare\_prophet\_data(df\_train, column) model = Prophet()

model.fit(prophet\_data) models[terminal] = model

# === Step 3: Create Future Dates (2025–2030) === future\_data = pd.DataFrame(columns=['Year', 'Month']) for year in range(2025, 2031):

for month in range(1, 13): future\_data = pd.concat(

[future\_data, pd.DataFrame({'Year': [year], 'Month': [month]})], ignore\_index=True

)

# Add datetime column for forecasting

future\_data['ds'] = pd.to\_datetime(future\_data['Year'].astype(str) + '-' + future\_data['Month'].astype(str), format='%Y-%m')

# === Step 4: Generate Forecasts with Prophet === for terminal, model in models.items():

forecast = model.predict(future\_data[['ds']])

future\_data[f'{terminal}\_Passengers\_Count'] = forecast['yhat'].values

**Model 1.2: XGBoost Time Series Regression – Facility-Level Traffic Forecasting**

## Justification

XGBoost regression is used to predict hourly vehicle volume (total\_vehicles) for each facility by modeling time-based and weather-dependent patterns. It’s chosen for its accuracy, speed, and ability to handle complex feature interactions and outliers common in real-world traffic datasets.

## Dependent Variable

* total\_vehicles: Total vehicle count per hour
* Transformed using log1p() to normalize distribution and improve regression accuracy

## Independent Variables

* **Time-based**: hour, day\_of\_week, month
* **Weather-based**: AWND, PRCP, SNOW, SNWD, TMAX\_2, TMIN

These features help the model capture daily, weekly, monthly, and climate-related fluctuations in traffic flow.

## Outcome

* Forecasts hourly traffic volume at each facility in the test set
* Evaluates prediction performance using RMSE on a validation set
* Provides insights for congestion forecasting, facility-level planning, and real-time traffic strategies

### Python code XGBoost Time Series Regression – Facility-Level Traffic Forecasting

The forecasting workflow in Python involved looping through each facility, cleaning and filtering the data, creating time and weather-based features, splitting data into train and validation sets, fitting an XGBoost regressor, evaluating model performance using RMSE, and preparing future traffic volume predictions for 2025–2030.

from xgboost import XGBRegressor

from sklearn.model\_selection import train\_test\_split from sklearn.metrics import mean\_squared\_error

forecast\_all = []

xgb\_params = { "n\_estimators": 100,

"learning\_rate": 0.05,

"max\_depth": 6,

"subsample": 0.8,

"colsample\_bytree": 0.8,

"random\_state": 42

}

facilities = train\_df["facility"].unique()

for facility in facilities:

print(f"\nTraining XGBoost for facility: {facility}") facility\_data = train\_df[train\_df["facility"] == facility].copy()

# Remove outliers in target variable

q1 = facility\_data["total\_vehicles"].quantile(0.25) q3 = facility\_data["total\_vehicles"].quantile(0.75)

iqr = q3 - q1

facility\_data = facility\_data[ (facility\_data["total\_vehicles"] >= (q1 - 1.5 \* iqr)) & (facility\_data["total\_vehicles"] <= (q3 + 1.5 \* iqr))

]

# Feature engineering

facility\_data["day\_of\_week"] = pd.to\_datetime(facility\_data["date"]).dt.dayofweek facility\_data["month"] = pd.to\_datetime(facility\_data["date"]).dt.month

features = ["hour", "AWND", "PRCP", "SNOW", "SNWD", "TMAX\_2", "TMIN",

"day\_of\_week", "month"]

X = facility\_data[features]

y = np.log1p(facility\_data["total\_vehicles"])

# Train-test split

X\_train, X\_valid, y\_train, y\_valid = train\_test\_split(X, y, test\_size=0.2, shuffle=False)

model = XGBRegressor(\*\*xgb\_params) model.fit(X\_train, y\_train)

# Predict and evaluate

preds = model.predict(X\_valid)

rmse = np.sqrt(mean\_squared\_error(y\_valid, preds)) print(f"RMSE for {facility}: {rmse:.2f}")

# Prepare future data (incomplete in shared snippet) future = test\_df[test\_df["facility"] == facility].copy()

# [Add feature engineering and prediction for 'future' if needed]

### Factors effecting the passengers using XgBoost.

Extreme GradientBoostingRegressor was used for its speed and ability to handle non-linear relationships in large datasets. It helps measure the influence of weather and holiday variables on traffic volume, while offering built-in feature importance for interpretability.

## Dependent Variable

* total\_vehicles: The number of vehicles recorded per timestamp.

## Independent Variables

* facility (encoded), hour, is\_holiday (encoded)
* Weather features: AWND, PRCP, SNOW, TMAX\_2, TMIN

## Outcome

The model highlights which external factors most impact traffic. Results support operational planning during weather events or holidays by identifying traffic-sensitive conditions.

# Python code for to determine how factors effecting the passengers.

The workflow included selecting weather and holiday features, encoding categorical variables, training a extremegradient boosting model, and using permutation importance to identify which features most influence total vehicle volume.

# ==============================================================

# Train Extreme Fast Gradient Boosting for Weather & Holiday Effect

# ==============================================================

rf\_df = df[["total\_vehicles", "facility", "hour", "is\_holiday", "AWND", "PRCP", "SNOW", "TMAX\_2", "TMIN"]].dropna()

rf\_df = pd.get\_dummies(rf\_df, columns=["facility", "is\_holiday"], drop\_first=True) X = rf\_df.drop("total\_vehicles", axis=1)

y = rf\_df["total\_vehicles"]

fast\_model = HistGradientBoostingRegressor(max\_iter=100) fast\_model.fit(X, y)

perm = permutation\_importance(fast\_model, X, y, n\_repeats=5, random\_state=42) importances = pd.Series(perm.importances\_mean, index=X.columns) importances.sort\_values().plot(kind='barh', figsize=(10,6), title="Feature Importance") plt.tight\_layout()

plt.show()

### Model 2 : K-means clustering of facilities by normalized trends

K-Means clustering was used to group transit facilities based on similar yearly passenger volume trends from 2005 to 2024. This unsupervised learning method helps uncover hidden patterns in terminal usage without needing labeled data.

By clustering facilities with similar traffic behavior, the Port Authority can better understand which terminals experience high, medium, or low demand. This segmentation supports targeted decision-making in areas like route optimization, infrastructure investment, and resource allocation. K-Means is efficient, scalable, and easy to interpret, making it ideal for grouping facilities by performance over time

### K-Means Clustering in Public Transit Planning

K-Means clustering is applied to group transit facilities based on their yearly passenger volume trends. This helps the Port Authority identify patterns across terminals, enabling more efficient route management, service scaling, and infrastructure planning.

Independent Variables

* Yearly Passenger Totals: Total number of passengers at each facility for each year (e.g., 2019–2024).
* These are numeric features that reflect how each facility performs annually.

Dependent Variable

* None: K-Means is an unsupervised learning algorithm, so there is no explicit dependent variable. It finds structure (clusters) within the input data based on similarity.

Clustering Outcome

* Each facility is assigned a Cluster label (e.g., 0, 1, 2).
* Facilities in the same cluster have similar yearly passenger trends.
* These clusters help:
  + Group facilities with high, medium, or low traffic.
  + Inform decisions on staffing, maintenance, and investment priorities.

### Python code for k means clustering of facilities by normalized trends

The clustering workflow involved aggregating yearly passenger totals per facility, transforming the data into a wide format, scaling it with StandardScaler, and applying K- Means to group facilities based on similar traffic patterns.

from sklearn.cluster import KMeans

from sklearn.preprocessing import StandardScaler

# Group data: total yearly passengers per facility facility\_yearly = df\_cleaned.groupby(['Facility', 'Year'])['total\_passengers'].sum().reset\_index()

# Reshape into wide format: facilities as rows, years as columns facility\_pivot = facility\_yearly.pivot(index='Facility', columns='Year', values='total\_passengers').fillna(0)

# Scale data for clustering scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(facility\_pivot)

# Apply KMeans with 3 clusters

kmeans = KMeans(n\_clusters=3, random\_state=42) kmeans.fit(X\_scaled)

# Assign cluster label to each facility facility\_pivot['Cluster'] = kmeans.labels\_

**Model 2.2 K-means clustering for grouping carriers on similar demand patterns.**

## Justification

K-Means clustering is used to group carriers at MBT based on similarity in their passenger volume patterns. By normalizing carrier-wise data and assigning cluster labels, we uncover natural groupings (e.g., high-demand vs low-demand carriers). This helps identify shared characteristics across carriers for strategic planning, marketing, or service allocation.

## Independent Variables

* **Passenger volumes** for each carrier:

passenger\_academy, passenger\_greyhound, passenger\_coach\_usa, passenger\_transbridge, passenger\_peterpan, passenger\_cj

These values reflect the monthly traffic trend for each carrier and are used to group them by similar behavior.

## Dependent Variable

* **None** (K-Means is an **unsupervised learning** algorithm).

It doesn’t predict a target — it identifies patterns in the input data.

## Outcome

* Assigns each row (month) a Cluster\_Label from 0 to 2 (for 3 clusters).
* Helps analyze which carriers behave similarly across time.
* Result is exported to CSV for further interpretation or visualization.

### Python code for Kmeans clustering for grouping carriers on similar demand patterns.

The clustering workflow involved selecting monthly passenger volumes for each carrier, scaling the values, applying K-Means with 3 clusters, and assigning each time period a cluster label to group carriers with similar demand patterns.

# === Step 7: KMeans Clustering ===

print("Running KMeans clustering...")

carrier\_data = df[carrier\_cols].copy() # Select carrier columns scaler = StandardScaler() # Normalize the data

normalized = scaler.fit\_transform(carrier\_data)

kmeans = KMeans(n\_clusters=3, random\_state=42) # Apply K-Means with 3 clusters

kmeans.fit(normalized)

df['Cluster\_Label'] = kmeans.labels\_

# Assign cluster labels

# Save cluster label output df.reset\_index()[['Date',

'Cluster\_Label']].to\_csv("downloads/mbt\_carrier\_clusters.csv", index=False)

print("Clustering output → downloads/mbt\_carrier\_clusters.csv")

# Model 2.3 K-Means Clustering – Traffic Level Classification

### Justification

K-Means clustering is used to classify traffic levels (Free, Moderate, Peak) based on total vehicle count and hour of the day, helping to uncover natural patterns in traffic behavior without labeled outcomes.

### Independent Variables

* total\_vehicles: Total count of all vehicle types
* hour: Hour of the day (0–23)

### Dependent Variable

* **None** (K-Means is unsupervised)

### Outcome

* Assigns a traffic\_level label to each record based on clustering:
  + Cluster 0 → Free
  + Cluster 1 → Moderate
  + Cluster 2 → Peak
* Helps with scheduling, congestion analysis, and planning time-specific traffic interventions.

### Python code for K-Means Clustering – Traffic Level Classification

The workflow involved extracting vehicle count and hour features, scaling the data, applying K-Means clustering with 3 groups, and labeling each hour as Free, Moderate, or Peak traffic level based on the assigned cluster.

import matplotlib.pyplot as plt import seaborn as sns

from sklearn.cluster import KMeans

# Ensure numeric data types

num\_cols = ["Auto", "Small\_trucks", "large\_trucks", "busses", "AWND", "PRCP", "SNOW", "SNWD", "TMAX\_2", "TMIN"]

df[num\_cols] = df[num\_cols].apply(pd.to\_numeric, errors="coerce")

# Format time and create timestamp

df["time"] = df["time"].apply(lambda x: f"{int(x):04d}")

df["time\_str"] = df["time"].str[:2] + ":" + df["time"].str[2:]

df["timestamp"] = pd.to\_datetime(df["date"] + " " + df["time\_str"], errors="coerce")

# Calculate total vehicles

df["total\_vehicles"] = df[["Auto", "Small\_trucks", "large\_trucks", "busses"]].sum(axis=1)

# Extract time-based features df["hour"] = df["timestamp"].dt.hour df["day"] = df["timestamp"].dt.day df["month"] = df["timestamp"].dt.month df["year"] = df["timestamp"].dt.year

df["dayofweek"] = df["timestamp"].dt.dayofweek df["is\_weekend"] = df["dayofweek"].isin([5, 6]).astype(int)

# KMeans Clustering

kmeans\_data = df[["total\_vehicles", "hour"]].dropna()

scaled = (kmeans\_data - kmeans\_data.mean()) / kmeans\_data.std()

kmeans = KMeans(n\_clusters=3, random\_state=42, n\_init=10) df.loc[kmeans\_data.index, "traffic\_level"] = kmeans.fit\_predict(scaled)

# Map cluster labels to descriptive names

df["traffic\_level"] = df["traffic\_level"].map({0: "Free", 1: "Moderate", 2: "Peak"})

print(" Feature engineering completed.") df.head()

**Model 3 : Random Forest classification to identify high-traffic months.**

## Justification

Random Forest is used to classify months as **high-traffic** or **not** based on passenger volumes. It is chosen for its robustness, ability to handle categorical and numerical variables, and resistance to overfitting. The ensemble method improves prediction accuracy by combining multiple decision trees.

## Industrial Application

* **Public Transit**: Identifying peak traffic periods helps agencies plan staffing, scheduling, and maintenance.
* **Retail**: Detecting high-sales periods for inventory and staff planning.
* **Healthcare**: Classifying peak patient visit periods for hospital resource allocation.
* **Event Management**: Forecasting high-demand periods to prepare infrastructure and logistics.

## Dependent Variable

* High\_Traffic: Binary label (1 for top 25% passenger months, 0 otherwise)
* Created using the 75th percentile of total\_passengers as threshold.

## Independent Variables

* **Time features**: Month, Year
* **Categorical features**: One-hot encoded Facility and quarter columns

## Outcome

* The model classifies each month as **high-traffic** or **not** with high interpretability.
* Evaluation is done using a classification report (precision, recall, F1-score).
* Enables proactive planning for resource deployment at terminals during peak demand months.

### Python code for Random Forest classification to identify high-traffic months.

The code workflow involved creating a high traffic label based on passenger volume thresholds, encoding categorical variables, defining features, splitting data, training a Random Forest classifier, and evaluating model performance using a classification report.

from sklearn.ensemble import RandomForestClassifier from sklearn.model\_selection import train\_test\_split from sklearn.metrics import classification\_report

# Create high traffic label

threshold = df\_cleaned['total\_passengers'].quantile(0.75)

df\_cleaned['High\_Traffic'] = (df\_cleaned['total\_passengers'] >= threshold).astype(int)

# Encode categorical features

df\_encoded = pd.get\_dummies(df\_cleaned, columns=['Facility', 'quarter'], drop\_first=True)

# Define features and target

features = ['Month', 'Year'] + [col for col in df\_encoded.columns if 'Facility\_' in col or 'quarter\_' in col]

X = df\_encoded[features]

y = df\_encoded['High\_Traffic']

# Split and train Random Forest

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

clf = RandomForestClassifier(n\_estimators=100, random\_state=42) clf.fit(X\_train, y\_train)

# Evaluate model

preds = clf.predict(X\_test) print(classification\_report(y\_test, preds))

## Additional Methods, Tools, and Techniques Used

To answer the company’s project questions effectively, we applied the following data science methods and tools:

### XGBoost Regression

Used to forecast passenger volumes and hourly traffic using time and weather features, with hyperparameter tuning via GridSearchCV and TimeSeriesSplit.

### Prophet Forecasting

Applied for long-term terminal-level forecasting, capturing seasonality, trends, and disruptions with minimal manual setup.

### K-Means Clustering

Grouped carriers and traffic records into Free, Moderate, and Peak levels based on volume and time-of-day patterns.

### Random Forest Classification

Identified high-traffic months using top 25% passenger volume thresholds, with evaluation through precision, recall, and F1-score.

### Feature Engineering & Preprocessing

Included creation of hour, day\_of\_week, month, and weather-based variables, plus outlier handling and normalization.

### Evaluation & Visualization

Used MAE, RMSE, MAPE for regression; classification reports for Random Forest; and generated visual outputs for insights.

These techniques allowed us to generate accurate forecasts, identify traffic patterns, and deliver actionable insights.

### Model Code and Tool Usage Explanation

We have included the full code for each model in our paper. Below is a breakdown of the

**tools used** and the **step-by-step process** followed for each:

### Programming Environment

* + **Python** was the primary language, using **Jupyter Notebooks** for development and visualization.
  + Ensured reproducibility by setting random seeds and organizing code in modular steps.

### Data Handling and Preprocessing

* + **Pandas** and **NumPy** were used for data loading, cleaning, merging, and transformation.
  + Outliers were handled using the IQR method.
  + Feature engineering included variables such as hour, day\_of\_week, month, and total\_vehicles.

### Forecasting Models

* + **XGBoost Regressor**: Applied to forecast traffic volume per facility and carrier.
    - Included GridSearchCV with TimeSeriesSplit for time-aware hyperparameter tuning.
    - Log transformation (log1p) was applied to stabilize target distribution.
  + **Prophet**: Used for monthly time series forecasting by terminal.
    - Automatically captured trends, seasonality, and changepoints with minimal feature engineering.

### Clustering

* + **K-Means Clustering**: Used to classify traffic levels (Free, Moderate, Peak) based on total\_vehicles and hour.
  + **StandardScaler**: Applied to normalize features before clustering for better grouping performance.

### Classification

* + **Random Forest Classifier**: Used to label months as High Traffic or Not based on weather and time variables.
  + Model was evaluated using a classification\_report with metrics like precision, recall, and F1-score.

# Pragmatic Applications: Answering the Company’s Core Questions

All techniques in this project were selected to offer practical value and directly address the company’s operational needs. Forecasting models such as XGBoost and Prophet were applied to predict future passenger volumes and hourly traffic trends, enabling better planning for staffing, scheduling, and facility usage. Random Forest classification was used to identify high-traffic periods based on historical and weather data, helping prioritize resource allocation during peak times. K-Means clustering grouped

traffic patterns into meaningful categories, making it easier to interpret complex data and support data- driven planning. Every model was tied to a specific business question, ensuring that our approach remains grounded in practical application rather than academic theory.

# Individual contribution :

**Jaypreet Singh** gathered and analyzed the data for the project, ensuring quality pre-processing across all datasets. He developed and implemented the full forecasting pipeline using XGBoost and Prophet, created traffic-level clustering using K-Means, and applied Random Forest for high-traffic classification. He also generated evaluation metrics, visualizations, and exported the final datasets and reports.

**V.V.V. Sai Sivaram Vangavolu** gathered, cleaned, and analyzed large-scale historical traffic and weather data from multiple CSV files. He engineered time-based and weather-related features, applied K-Means clustering for traffic level classification, and developed a comprehensive XGBoost regression pipeline to forecast hourly traffic by facility from 2025 to 2030. He also generated advanced visualizations, performed pre- vs post-COVID traffic comparisons, and contributed significantly to the model evaluation and interpretability.

**Akshitha Reddy** focused on understanding how weather conditions and holidays impact passenger traffic. She cleaned and simplified the dataset for modelling, ensuring high data quality and readability. She developed and implemented a Fast Gradient Boosting model (HistGradientBoostingRegressor) to analyze feature importance and determine which factors most significantly affect vehicle volume, contributing valuable interpretability to the forecasting analysis.

**Kamrul Aslam** contributed to the overall design and documentation of the project. He assisted in reviewing the models, validating outputs, and ensuring that the forecasting results aligned with the project objectives. He also helped organize results, reviewed visualizations for clarity, and supported testing and interpretation of traffic trends across facilities. His role ensured that insights were presented clearly and aligned with the company's use cases.

**VSB Rao** played a supporting role in model validation, presentation preparation, and overall coordination of team efforts. He contributed by proofreading the code, cross-verifying data outputs, and ensuring the forecasting and classification results were logically sound. Additionally, he helped draft explanations of the business relevance of the models and contributed to the visual summary of the findings for the final report.

# Conclusion Summary:

### 1. Forecasting Passenger Volume for 2025–2030:

The **Prophet model** was used to predict monthly passenger volumes at GWBBS and MBT terminals from 2025 to 2030. Its ability to capture trends, seasonality, and changepoints

enabled accurate long-term projections. This helps the Port Authority plan capacity and operations for future staging facilities.

### Identifying Key Factors Affecting Passenger Numbers:

To uncover the most influential variables, the **HistGradientBoostingRegressor** was applied. It evaluated the importance of features like weather (TMAX, SNOW, PRCP) and holiday indicators. This revealed how external factors impact demand and helps in adjusting operations during weather-sensitive or holiday periods.

### Forecasting Passenger Volume by Individual Carrier:

The **XGBoost regression model** was trained separately for each carrier using time and weather variables. This provided carrier-level passenger forecasts through 2030, enabling better service planning, resource allocation, and coordination between Port Authority and private bus operators.

### Finding Busiest Times (Weekly, Monthly, Yearly):

Using **K-Means clustering** and quantile classification, traffic levels (Free, Moderate, Peak) were derived by hour, day type, and month. Visualizations across weeks, months, and years revealed peak periods. This supports scheduling, congestion management, and staffing decisions at staging facilities.

### Comparing 2025 Forecasts to Pre-COVID Baseline (2019):

Forecasted 2025 data was compared with actual 2019 traffic to assess recovery trends. Yearly and monthly averages were plotted by facility, revealing which terminals are exceeding, matching, or lagging behind pre-pandemic levels. This gives the Port Authority a data-driven basis for post-COVID strategic adjustments.