Metropolitan Transportation Authority (MTA)

By The Integrators (Rachael Gottbetter, William Kinum, & Kyle Tenczar)

Purpose:
Track customer experience by measuring wait times, travel delays, and overall journey performance across subway lines.

Key Metrics

Key Data Fields:

- **month (DATE):** Month metrics are recorded (yyyy-mm-dd).
- **division (TEXT):** Subway division – A Division (numbered lines + S 42nd) or B Division (lettered lines).
- **line (TEXT):** Specific subway line (e.g., 1, 2, A, C, E, etc.).
- **period (TEXT):** Peak vs. Off-Peak service periods.

Performance Metrics:

- **num_passengers (NUMERIC):** Total estimated monthly passengers per line.
- **additional platform time (NUMERIC):** Avg. extra wait time at platform (minutes).
- **additional train time (NUMERIC):** Avg. extra onboard time (minutes).
- **total_apt / total_att (NUMERIC):** Total accumulated extra minutes waiting or onboard.

Customer Journey Outcomes:

- **over_five_mins (NUMERIC):** Estimated number of customers delayed >5 minutes.
- **over_five_mins_perc (PERCENT):** % of customers delayed >5 minutes.
- **customer journey time performance (PERCENT):** % of customers completing journeys within 5 minutes of schedule.

Classification

Research Questions

- What factors most influence customer journey time performance?
- Can we accurately predict when performance will exceed the 90% threshold?
- Do different subway divisions have unique performance characteristics?
- How do peak vs. off-peak periods impact performance?
- Which subway lines consistently outperform or underperform?

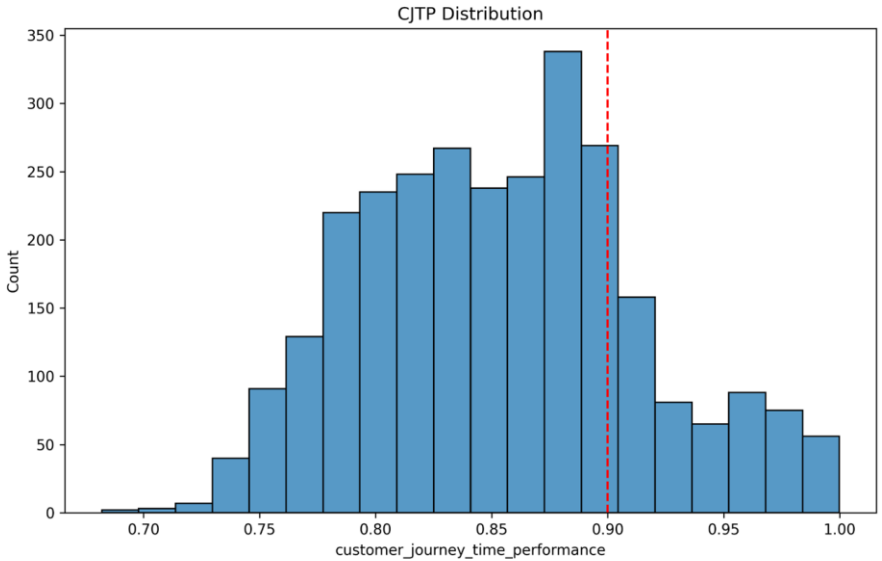
Data Preprocessing

Cleaning and Feature Engineering:

- Renamed columns for consistency
- Created derived metrics:
  - Total additional time
  - Per-passenger platform and train delays
  - Monthly and yearly patterns
- Established target variables:
  - Binary: high_performance (≥90% CJTP)
  - Multi-class: performance_class (0=excellent, 1=acceptable, 2=needs improvement)

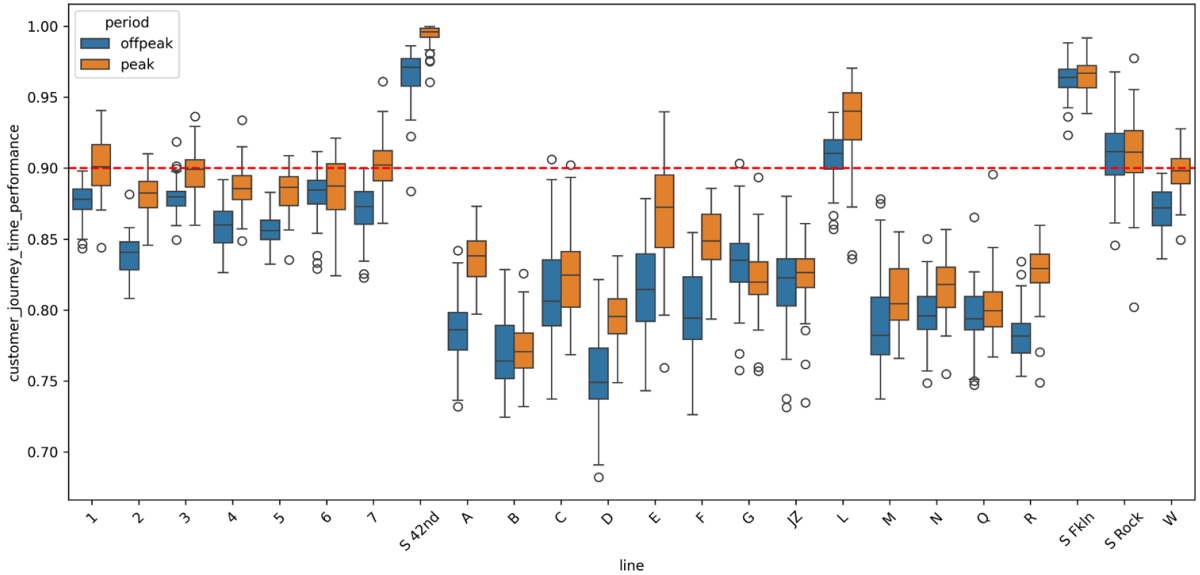
Key Findings: Performance Distribution

- Most observations fall between 75-90% performance
- Clear threshold at 90% for "high performance"
- Multiple peaks suggest different operational modes



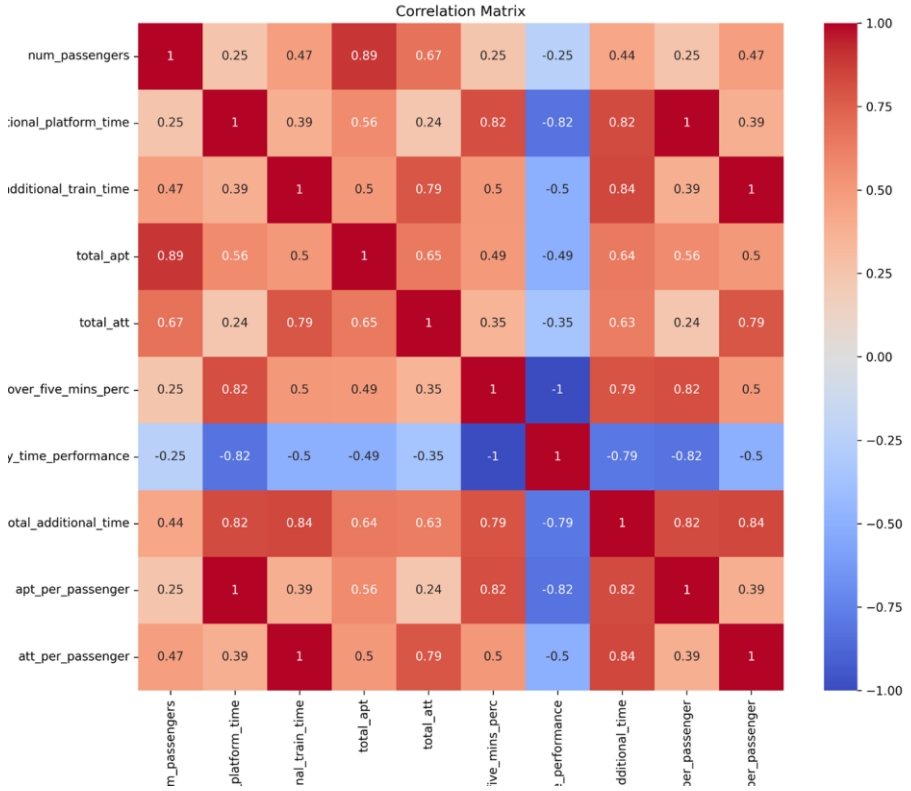
Key Findings: Line Performance

- S 42nd and S Plain lines consistently exceed 90% threshold
- Lines B, C, and D show consistent underperformance
- Peak periods typically outperform off-peak
- Lines 7, L, and 1 show strong performance in both periods



Key Findings: Correlation Analysis

- Strong negative correlation between delays >5min and performance
- Platform and train delays highly correlated with performance
- Passenger volume correlates strongly with platform delays
- Total additional time strongly predicts performance



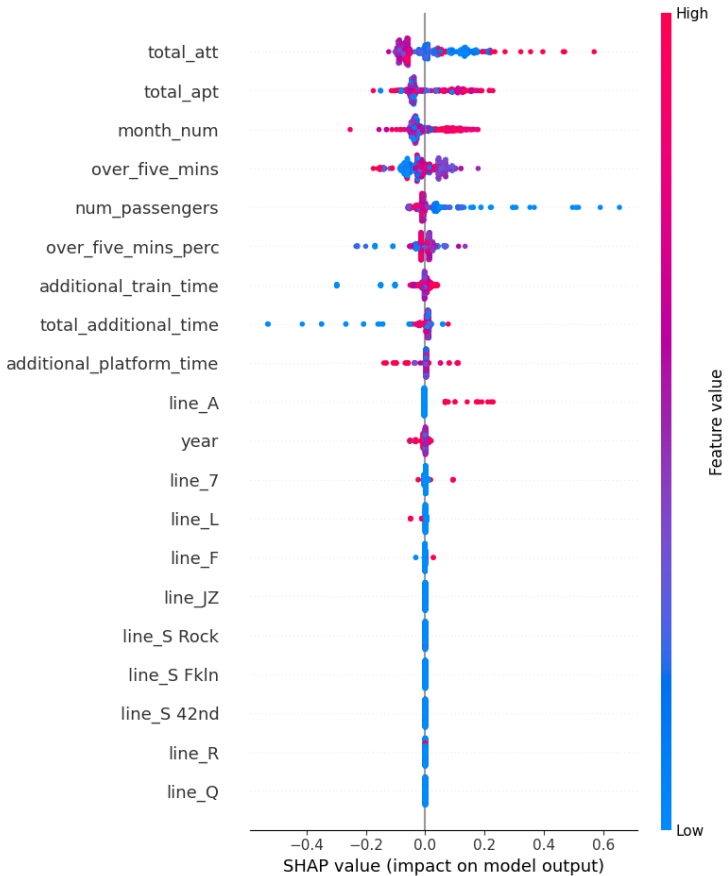
Key Findings: SHAP Feature Importance Analysis

Top Influential Features:

1. **total_att** (Total Additional Train Time) - Highest impact with wide distribution of effects
2. **total_apt** (Total Additional Platform Time) - Strong influence with positive and negative impacts
3. **month_num** - Seasonal effects show significant positive correlation
4. **over_five_mins** - Count of delays over 5 minutes shows mixed impact
5. **num_passengers** - Passenger volume shows complex relationship with performance

Insights:

- **Train Delays** have stronger impact than platform delays
- **Seasonal patterns** (month_num) play a significant role in performance
- **Line A** is the only subway line with substantial positive impact
- **Passenger volume** shows an interesting pattern - fewer passengers (blue dots) appearing on the negative side
- Most individual subway lines have minimal impact compared to system-wide metrics



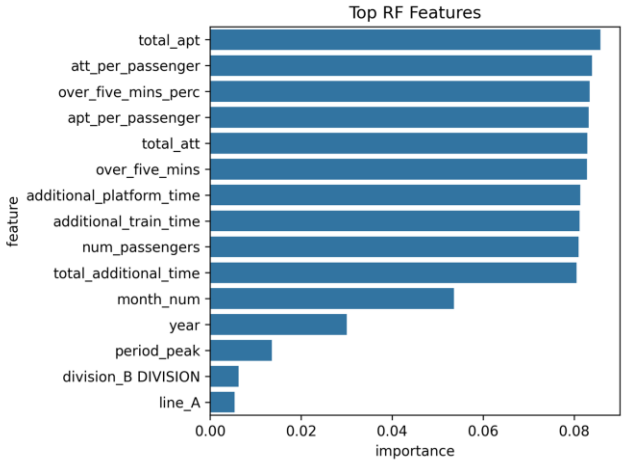
Model Results

Random Forest

Performance:

- Binary Classification Accuracy: ~93%
- Multi-class Classification Accuracy: ~87%

Top Features:

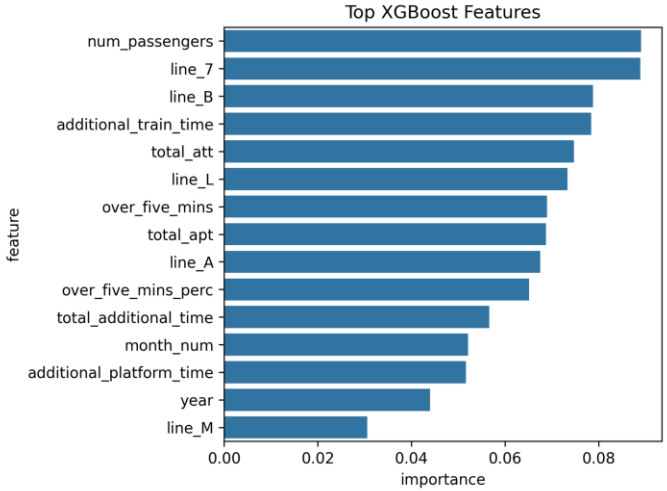


XGBoost

Performance:

- Binary Classification Accuracy: ~93%
- Multi-class Classification Accuracy: ~87%

Top Features:



Division-Specific Models

Performance:

- A Division Model Accuracy: ~94%
- B Division Model Accuracy: ~91%

Key Differences:

- A Division more sensitive to platform waiting time
- B Division more influenced by train travel time
- Different lines within divisions show unique patterns

Insights and Recommendations

Actionable Insights

1. **Platform Time Management**
    - Reduce waiting time, especially during peak periods
    - Focus on high-volume stations
2. **Line-Specific Interventions**
    - Prioritize resources for consistently underperforming lines (B, C, D)
    - Study and replicate successful practices from top performers (S 42nd, S Plain)
3. **Delay Reduction**
    - Critical impact of delays exceeding 5 minutes
    - Early intervention system based on predictive models

Implementation Recommendations

Short-term:

- Deploy monitoring dashboard tracking key performance predictors
- Implement threshold alerts for potential performance issues
- Optimize crew and train allocation based on passenger volume patterns

Long-term:

- Integrate predictive models into operations planning
- Develop line-specific improvement strategies
- Create division-specific performance standards

What Next?

Future Research Directions

- Temporal analysis (holidays, events, weather impacts)
- Integration with infrastructure maintenance data
- Passenger satisfaction correlation analysis
- Expanded feature engineering (transfer patterns, line connectivity)

Regression

Potential Regression Missions

- Journey Times
  - Predict changes in train service times to update train journey time
  - Uses: operators, schedulers, maintenance, etc.
- Passenger Wait Times
  - Predict time passengers wait for trains at specific stations during different times of the year
  - Uses: customer satisfaction, performance, fare management
- Customer Journey Time Performance
  - Predict percentage of customers who complete journeys on time
  - Uses: customer satisfaction, performance, fare management

Goals

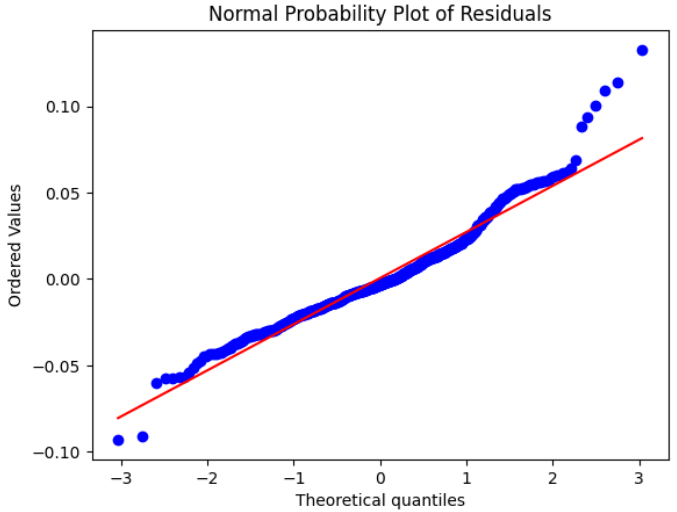
- Predict Customer Journey Time Performance
  - Percentage of customers who complete their journey on time
- Use feature engineering to create new variables and improve model performance
  - See Classification section
- Test multiple regression models for best fit
  - Simple regression, LASSO, Ridge
- Run model adequacy checks, diagnostics, and validation

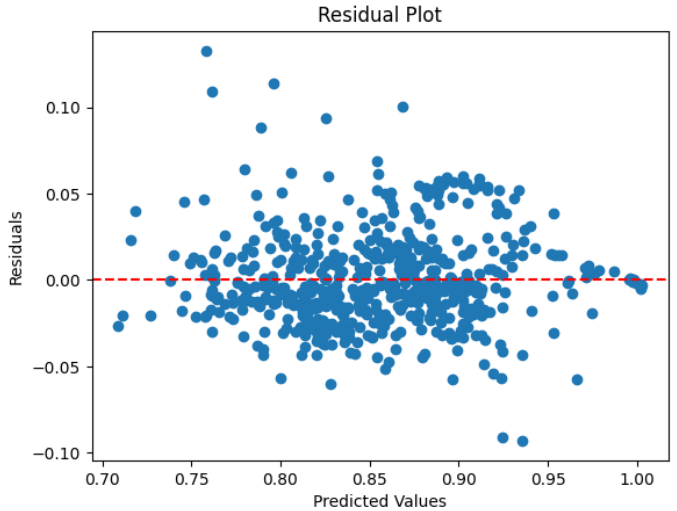
Our Approach

1. Feature Engineering
2. Correlation Analysis
3. Run regression models
    - Simple Linear Regression
    - LASSO Regression
    - Ridge Regression
4. Model Adequacy Checking
5. Residual Analysis
6. Model Validation

Simple Linear Regression

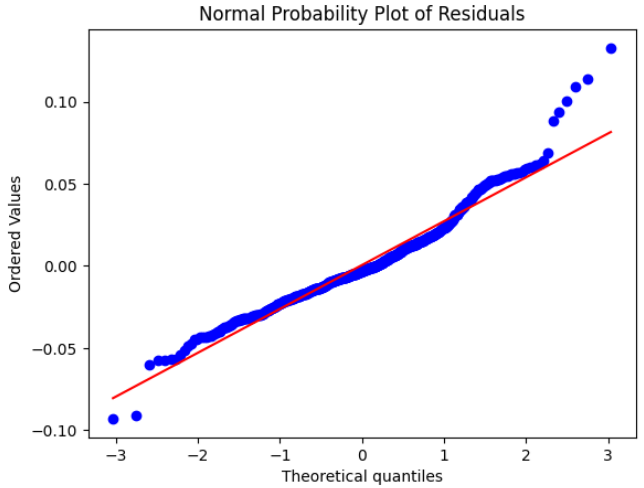
Simple Linear Regression	
Metric	**Value**
AIC	(16,444.51)
R2	80.3%
Cross-Validation Score	(0.0007)
Bootstrap .632 Estimate	0.0007

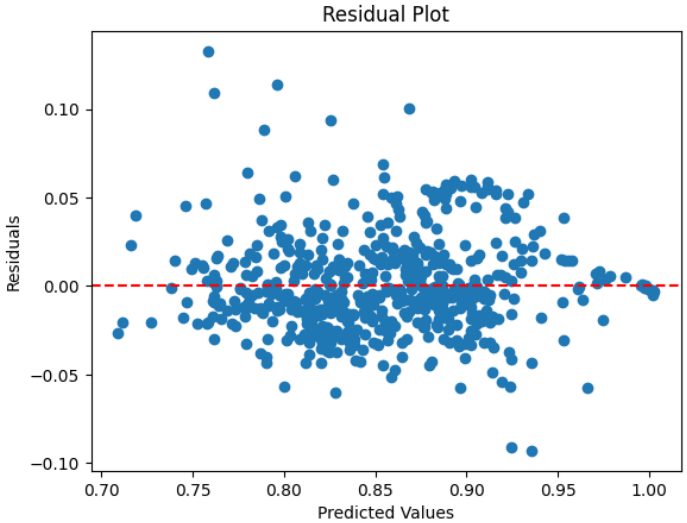




Ridge Model

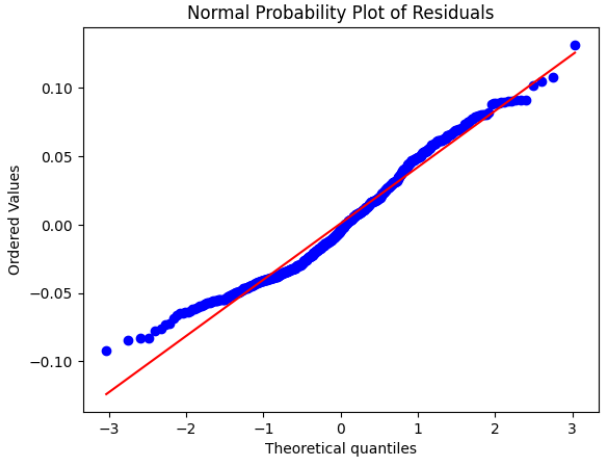
Ridge Regression	
Metric	**Value**
AIC	90,718.63
R2	80.7%
Cross-Validation Score	(0.0007)
Bootstrap .632 Estimate	0.0007

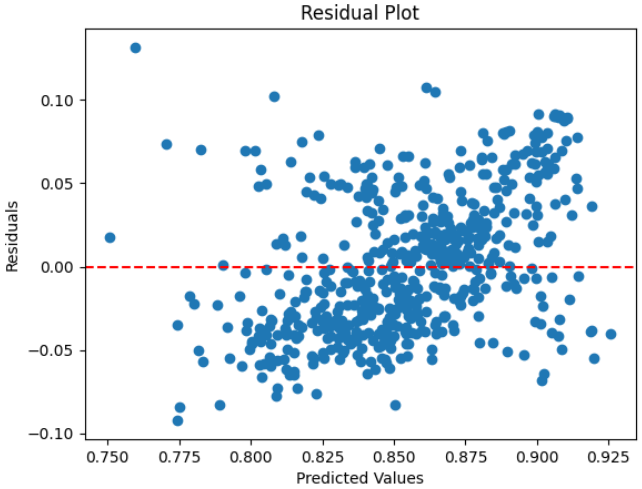




LASSO Model

LASSO Regression	
Metric	**Value**
AIC	38,554.02
R2	55.1%
Cross-Validation Score	(0.0016)
Bootstrap .632 Estimate	0.0016





Conclusions

- Based on the tests regression analysis is generally able to predict Customer Journey Time Performance with good accuracy
- Of all models tested, the simple linear model performed best
- Most assumptions held
  - Possibly some issues with thin tails in the residuals
  - Further data transformations may be necessary to improve model performance

Unsupervised Learning
-
Clustering

Goal

We aimed to classify New York City subway lines into performance categories based on customer journey metrics using **unsupervised learning**. The objective was to discover natural groupings of train lines—without prior labels—based on attributes like platform wait time, total delay, and customer journey time performance (CJTP).

Goal

Identifying patterns without predefined labels yields the following advantages:

1) **Improve Operational Efficiency**
   By grouping lines with similar delay patterns or service characteristics, transit authorities can **target interventions** (e.g., scheduling changes or staffing) more effectively.

2) **Simplify Complex Data**
   With multiple performance metrics per line, clustering condenses that into a few categories (“good”, “mediocre”, “bad”), making the data easier to interpret for non-technical stakeholders.

3) **Support Resource Allocation**
   Once underperforming clusters are identified, resources (like maintenance crews or service redesign efforts) can be **allocated more strategically**.

Methods

Two clustering techniques were applied:

1. K-Means Clustering

- **Approach**: Partitioned lines into 3 clusters based on Euclidean distances to cluster centroids.
- **Preprocessing**: Standardized aggregated metrics for each subway line.
- **Labeling**: Clusters were interpreted post hoc as “good,” “mediocre,” or “bad” based on average CJTP.
- **Visualization**: PCA and pairplots were used to examine clustering structure.

Methods

Two clustering techniques were applied:

2. Hierarchical Clustering

- **Approach**: Agglomerative clustering using **Ward's method**, merging clusters to minimize within-cluster variance.
- **Preprocessing**: Same as for K-Means.
- **Visualization**: Dendrogram and PCA used to identify and interpret 3 clusters.
- **Labeling**: Same strategy based on CJTP.

K-Means

Algorithm Steps

1. **Initialization**: Select k initial centroids $\mu_1, \ldots, \mu_k$, typically by random sampling from the data.
2. **Assignment Step**: Assign each point x_j to the cluster with the nearest centroid:

$$C_i = \{x_j : \|x_j - \mu_i\|^2 \leq \|x_j - \mu_l\|^2 \quad \text{for all } l = 1, \ldots, k\}$$

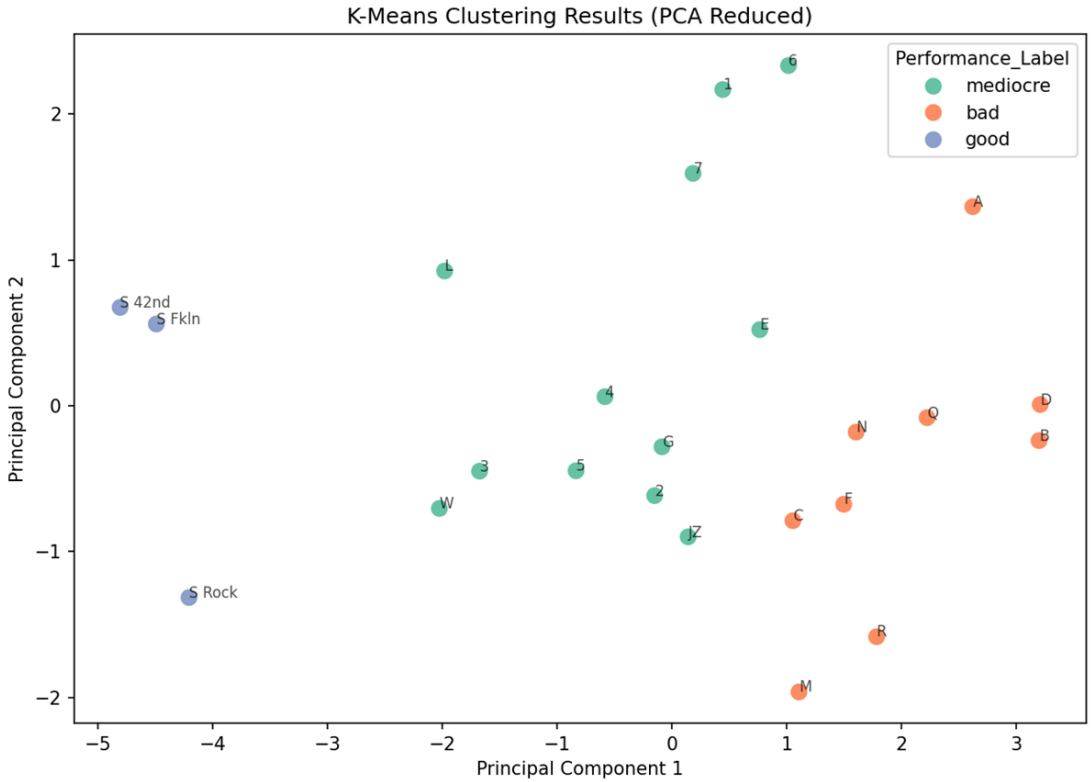
3. **Update Step**: Recompute the centroid of each cluster:

$$\mu_i = \frac{1}{|C_i|} \sum_{x \in C_i} x$$

4. **Repeat**: Repeat steps 2 and 3 until the assignments no longer change or a maximum number of iterations is reached.

K-Means: Visualize Results

We can make a 2D visualization of the different clusters by reducing the features down to two principal components.



K-Means: Results

Cluster Summary Statistics by Performance Label

Performance Label	Add. Platform Time	Add. Train Time	Total APT	Total ATT	Over 5 Mins	Over 5 Mins %	CJTP
bad	1.585	0.431	3,877,926.21	1,179,860.20	486,586.39	0.199	0.801
good	0.646	-0.246	100,942.39	-6,982.24	4,574.39	0.049	0.951
mediocre	1.147	0.290	2,844,412.18	975,640.36	311,307.80	0.129	0.871

Hierarchical Clustering

Algorithm Steps (Agglomerative Clustering with Ward's Method)

Given a dataset $\{x_1, x_2, \ldots, x_n\} \subset \mathbb{R}^d$:

1. **Initialization**: Begin with n singleton clusters, each containing one point.
2. **Distance Matrix**: Compute the pairwise Euclidean distance between all points:

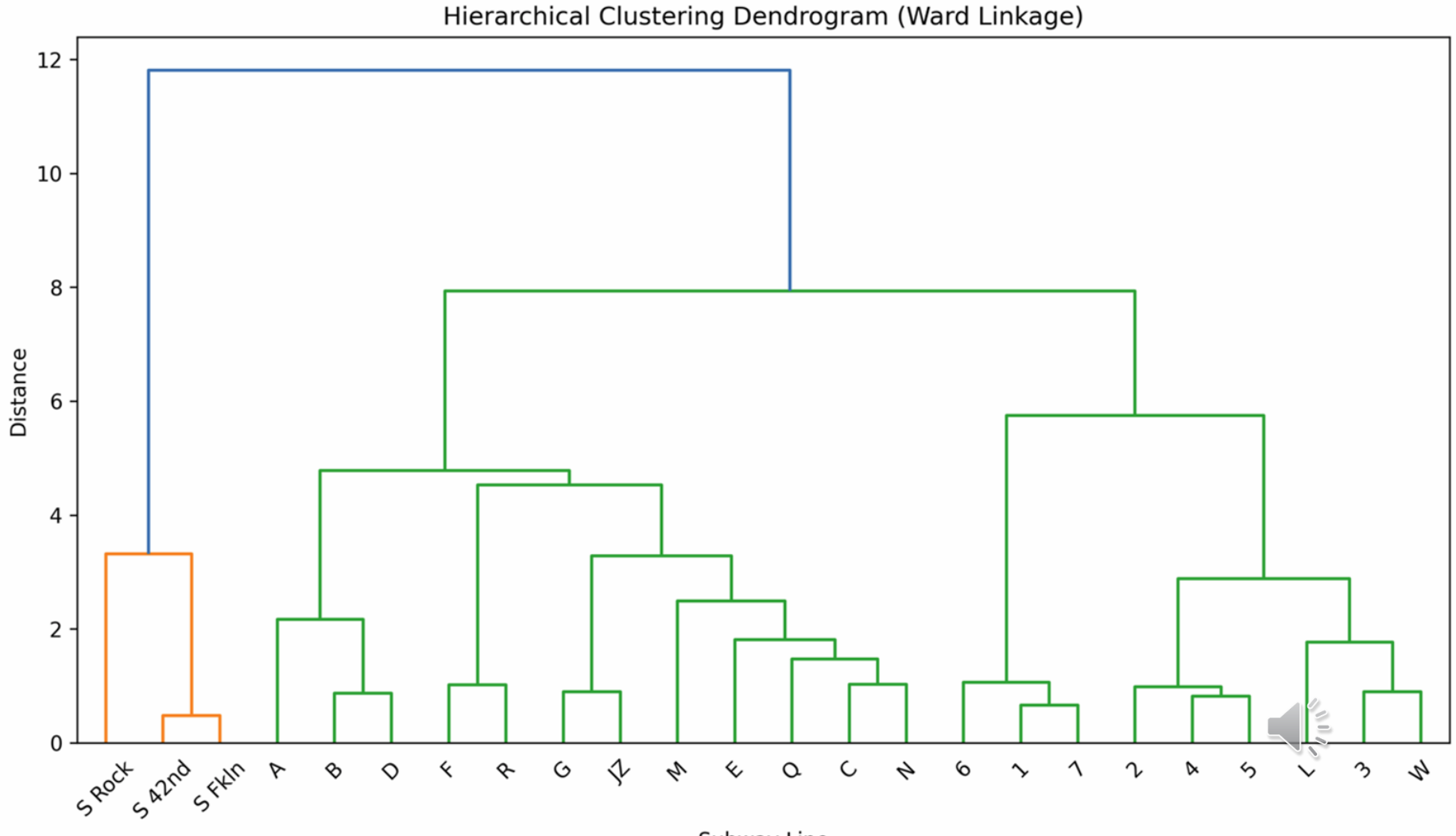
$$D(i, j) = \|x_i - x_j\|$$

3. **Merge Closest Clusters**: Merge the two clusters A and B that result in the minimal increase in total within-cluster variance.
4. **Ward's Criterion**: The distance between clusters A and B is computed as:

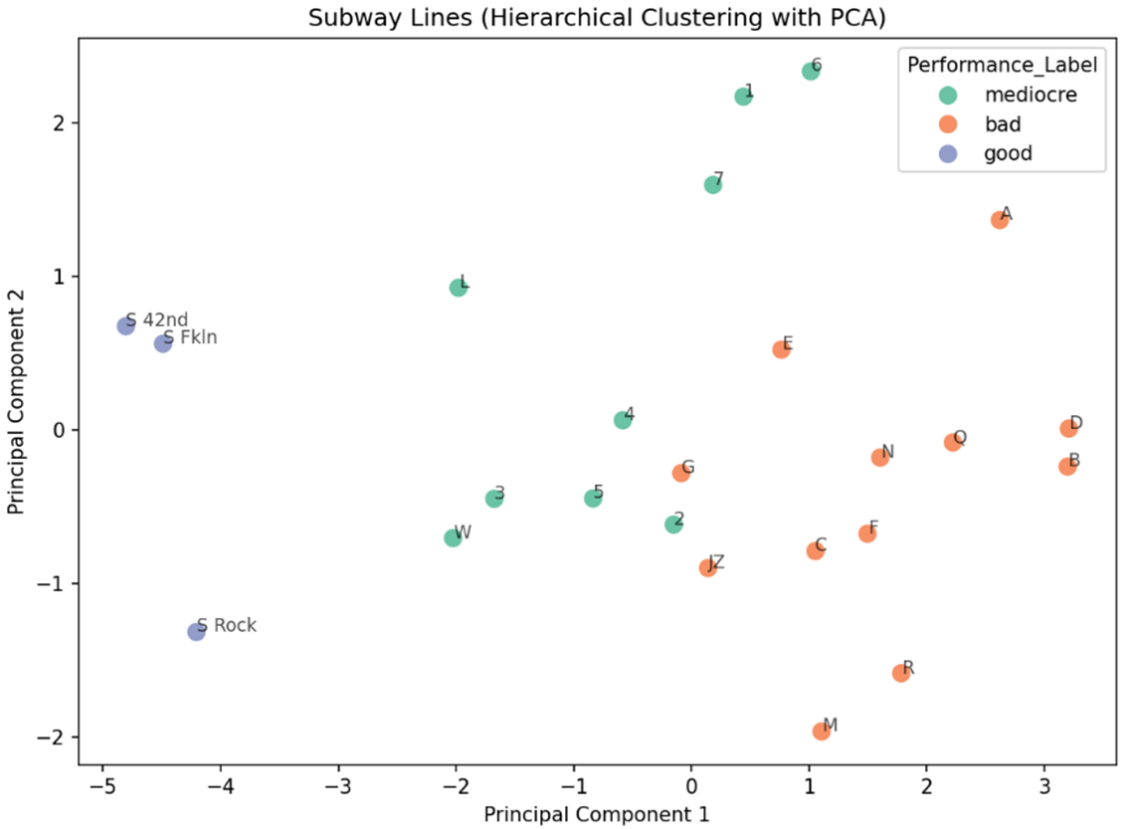
$$D_{\text{Ward}}(A, B) = \frac{|A||B|}{|A| + |B|} \|\mu_A - \mu_B\|^2$$

where μ_A and μ_B are the centroids of clusters A and B.

5. **Repeat**: Repeat steps 3–4 until all data points are merged into a single cluster.


Hierarchical Clustering Dendrogram (Ward Linkage)
Distance
0
2
4
6
8
10
12
S Rock
S 42nd
S Fkln
A
B
D
F
R
G
JZ
M
E
Q
C
N
6
1
7
2
4
5
L
3
W

Hierarchical Clustering: 2D Visualization



Hierarchical Clustering: Results

Cluster Summary Statistics by Performance Label (Hierarchical Clustering)

Performance Label	Add. Platform Time	Add. Train Time	Total APT	Total ATT	Over 5 Mins	Over 5 Mins %	CJTP
bad	1.501	0.446	3,450,488.00	1,110,255.00	437,468.26	0.191	0.809
good	0.646	-0.246	100,942.39	-6,982.24	4,574.39	0.049	0.951
mediocre	1.114	0.223	3,069,825.00	1,000,373.00	318,372.45	0.116	0.884

Comparison of Clustering Results

Subway Line Cluster Assignments (K-Means)

Line	Cluster	Performance Label
1	0	mediocre
2	0	mediocre
3	0	mediocre
4	0	mediocre
5	0	mediocre
6	0	mediocre
7	0	mediocre
A	1	bad
B	1	bad
C	1	bad
D	1	bad
E	0	mediocre
F	1	bad
G	0	mediocre
JZ	0	mediocre
L	0	mediocre
M	1	bad
N	1	bad
Q	1	bad
R	1	bad
S 42nd	2	good
S Fkln	2	good
S Rock	2	good
W	0	mediocre

Subway Line Cluster Assignments (Hierarchical Clustering)

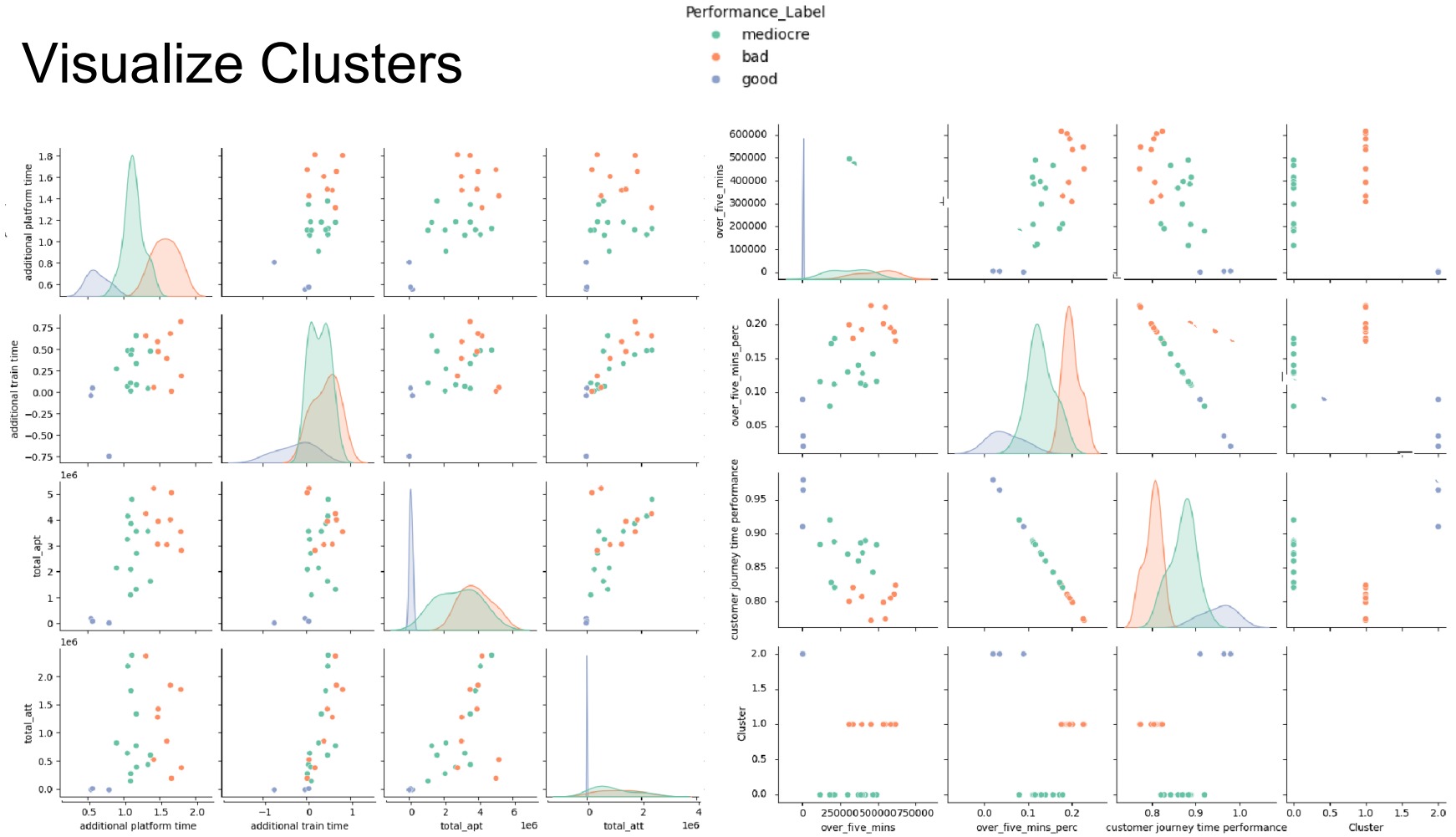
Line	Cluster	Performance Label
1	0	mediocre
2	0	mediocre
3	0	mediocre
4	0	mediocre
5	0	mediocre
6	0	mediocre
7	0	mediocre
A	2	bad
B	2	bad
C	2	bad
D	2	bad
E	2	bad
F	2	bad
G	2	bad
JZ	2	bad
L	0	mediocre
M	2	bad
N	2	bad
Q	2	bad
R	2	bad
S 42nd	1	good
S Fkln	1	good
S Rock	1	good
W	0	mediocre

Quantitative Comparisons

In both cases:

- The bad clusters had the highest values for additional platform time, total APT, and the proportion of trips over five minutes. They also had the lowest customer journey time performance (CJTP), around 0.80.
- The good cluster exhibited the lowest wait times and the highest CJTP, around 0.95.
- The mediocre cluster fell in between, with moderate values for all metrics.

Visualize Clusters



Discussion of Clustering Results

Agreements

Both clustering methods produced three distinct groups of subway lines labeled as *good*, *mediocre*, and *bad* based on aggregate performance metrics. The methods were largely consistent in their classifications.

•The good cluster, consisting of the shuttle lines (S 42nd, S Fkln, and S Rock), was identical in both methods.

• Most of the major numbered lines (1–7) and lettered lines such as L and W were placed in the mediocre category by both approaches.

• The bad cluster consistently included heavily used lines such as A, B, C, D, F, M, N, Q, and R.

Discrepancies:

•Lines E, G, and JZ were classified as mediocre by K-Means but as bad by Hierarchical clustering.

•These differences suggest that the Hierarchical method is more *conservative*, grouping borderline performers with lower-performing lines.

Discussion of Clustering Results

- The K-Means and Hierarchical clustering methods led to broadly consistent but subtly different categorizations of subway line performance.
- K-Means offered a more centroid-driven partitioning, which may be suitable when performance categories are assumed to be compact and spherical in feature space.
- Hierarchical clustering provided a more nuanced dendrogram-based approach, yielding slightly stricter categorization and better interpretability through visualization.
- Depending on the application, either method may be appropriate. For general-purpose clustering with known k, K-Means is efficient and effective. However, for understanding relative performance in more detail and revealing nested groupings, Hierarchical clustering is advantageous.

Classification Code

```
# 1. Data Loading and Preprocessing
# ---------------------------------

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import OneHotEncoder
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, accuracy_score
from sklearn.pipeline import Pipeline
import xgboost as xgb
import shap
from pathlib import Path
import joblib

# Load Data
file_path = Path('C://Users//e10933818//Desktop//Theory of Machine Learning//MTA_Subway_Customer_Journey-Focused_Metrics__Beginning_2020 (2).csv')
if file_path.exists():
    df = pd.read_csv(file_path)
    print("Data loaded successfully")
else:
    raise FileNotFoundError(f"File not found at: {file_path}")

# Rename columns
column_mapping = {
    'additional platform time': 'additional_platform_time',
    'additional train time': 'additional_train_time',
    'customer journey time performance': 'customer_journey_time_performance'
}
df = df.rename(columns=column_mapping)

# Feature Engineering
df['total_additional_time'] = df['additional_platform_time'] + df['additional_train_time']
df['apt_per_passenger'] = df['total_apt'] / df['num_passengers']
df['att_per_passenger'] = df['total_att'] / df['num_passengers']
df['month_num'] = pd.to_datetime(df['month']).dt.month
df['year'] = pd.to_datetime(df['month']).dt.year
df['high_performance'] = df['customer_journey_time_performance'] >= 0.90

def performance_class(cjtp):
    if cjtp >= 0.9:
        return 0
    elif cjtp >= 0.8:
        return 1
    else:
        return 2

df['performance_class'] = df['customer_journey_time_performance'].apply(performance_class)

# EDA
plt.figure(figsize=(10, 6))
sns.histplot(df['customer_journey_time_performance'], bins=20)
plt.axvline(0.9, color='red', linestyle='--')
plt.title('CJTP Distribution')
plt.savefig('cjtp_distribution.png', dpi=300)
plt.close()

plt.figure(figsize=(12, 6))
sns.boxplot(x='line', y='customer_journey_time_performance', hue='period', data=df)
plt.axhline(0.9, color='red', linestyle='--')
plt.xticks(rotation=45)
plt.tight_layout()
plt.savefig('boxplot_performance.png', dpi=300)
plt.close()

numeric_cols = ['num_passengers', 'additional_platform_time', 'additional_train_time', 'total_apt', 'total_att',
                'over_five_mins_perc', 'customer_journey_time_performance', 'total_additional_time',
                'apt_per_passenger', 'att_per_passenger']
plt.figure(figsize=(12, 10))
sns.heatmap(df[numeric_cols].corr(), annot=True, cmap='coolwarm')
plt.title('Correlation Matrix')
```

Regression Code

```
# Import Packages
import numpy as np
import pandas as pd
import statsmodels.api as sm
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score, KFold
from sklearn.utils import resample
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import LabelEncoder
import itertools
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import matplotlib.pyplot as plt
from scipy.stats import probplot
from sklearn.model_selection import LeaveOneOut
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
from itertools import combinations



        # Load Data
df = pd.read_csv("C:\\Users\\hotke\\OneDrive\\Documents\\JHU\\Theory of ML\\Project\\MTA_Subway_Customer_Journey-Focused_Metrics__Beginning_2020.csv")
label_encoder = LabelEncoder()
```

Clustering Code

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.decomposition import PCA

          # Load the dataset
df = pd.read_csv("MTA_Subway_Customer_Journey-Focused_Metrics__Beginning_2020.csv")

          # Select numeric columns for clustering
numeric_cols = ["additional platform time", "additional train time", "total_apt", "total_att",
                "over_five_mins", "over_five_mins_perc", "customer journey time performance"]

          # Ensure the selected columns are numeric
df[numeric_cols] = df[numeric_cols].apply(pd.to_numeric, errors='coerce')

          # Aggregate by 'line' and compute mean
agg_df = df.groupby("line")[numeric_cols].mean().reset_index()
```