Group 3 – Capstone Final Project

Python Model Resources:

1. Python Code used for Data Standardization & Data Cleaning

1) Standardization

```
import pandas as pd
df_bus = pd.read_excel("All Recorded PABT Bus.xlsx")
df_passenger = pd.read_excel("All Recorded PABT Passenger.xlsx")
df_traffic = pd.read_excel("All Recorded Traffic.xlsx")
 df_speed = pd.read_excel("Facility Mobility Speeds.xlsx")
df_bus.columns = df_bus.columns.str.strip().str.replace(" ", "_").str.lower()
df_passenger.columns = df_passenger.columns.str.strip().str.replace(" ",_").str.lower()
df_traffic.columns = df_traffic.columns.str.strip().str.replace(" ", "_").str.lower()
df_speed.columns = df_speed.columns.str.strip().str.replace(" ", "_").str.lower()
```

2) Combining Datasets

Pandas.concat() was used to combine several files.
Combined with shared fields Facility_ID Date
This made it possible to create a single me-series dataset. combined_df =
pd.concat([df_bus, df_passenger, df_traffic, df_speed], ignore_index=True)
merged_df = df_traffic.merge(df_speed, on=["facility_id", "date"], how="le ")

3) Handling Duplicates

combined_df.duplicated().sum() combined_df = combined_df.drop_duplicates()

4) Managing Inaccurate and Missing Data

```
num_cols = combined_df.select_dtypes(include=['float64','int64']).columns
 for col in num_cols:
combined_df[col] = combined_df[col].fillna(combined_df[col].mean())
 combined_df = combined_df.sort_values("date")
```

combined_df = combined_df.fillna(method="ffill") cat_cols =
combined_df.select_dtypes(include=['object']).columns

```
 for col in cat_cols:

 combined_df[col] = combined_df[col].fillna(combined_df[col].mode()[0])
combined_df = combined_df.replace(r'^\s*$', pd.NA, regex=True)
```

combined_df['invalid_speed_flag'] = combined_df['speed'].apply(lambda x: 1 if x < 0
or x > 120 else 0)

```
combined_df = combined_df[combined_df['invalid_speed_flag'] == 0]
combined_df['revenue'] = combined_df['revenue'].round(2)
```

5)Normalization of Text and Format

```
combined_df = combined_df.applymap(lambda x: x.strip() if isinstance(x, str) else x)
canonical_names = {

"veh_type": "vehicle_type",

"facility": "facility_name",

    "weather": "weather_condi on",

    "vol": "volume" }

    combined_df.rename(columns=canonical_names, inplace=True)
    combined_df["date"] = pd.to_dateme(combined_df["date"], errors="coerce")
    invalid_dates = combined_df[combined_df["date"].isna()]
```

6) Identification and Management of Outliers

```
    import matplotlib.pyplot as plt

    combined_df.boxplot(column=["volume"])

    plt.show()

    Q1 = combined_df['volume'].quan le(0.25)

    Q3 = combined_df['volume'].quan le(0.75)

     IQR = Q3 - Q1 combined_df['outlier_flag'] = combined_df['volume'].apply(

    lambda x: 1 if (x < (Q1 - 1.5 * IQR)) or (x > (Q3 + 1.5 * IQR)) else 0

     )

    outliers = combined_df[combined_df['outlier_flag'] == 1]
```

7) COMPLETE CLEANED DATASET

```
combined_df.to_csv("final_cleaned_port_authority_data.csv", index=False)
```

2. Python Code for EDA

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Load dataset
df = pd.read_csv("Cleaned_merged_data.csv")

# Function to create boxplot and histogram for each variable
def box_hist(col):
    fig, axes = plt.subplots(1, 2, figsize=(12,4))
    sns.boxplot(x=df[col], ax=axes[0])
    axes[0].set_title(f"Boxplot of {col}")
    axes[1].hist(df[col].dropna(), bins=40)
    axes[1].set_title(f"Histogram of {col}")
    plt.tight_layout()
    plt.show()

# Volume & payment variables
volume_vars = ["Passenger_Volume", "TOTAL", "Autos", "EZPASS", "CASH",
"VIOLATION"]

for v in volume_vars:
    box_hist(v)

# Vehicle class variables
vehicle_classes = [
    "CLASS 1", "CLASS 2", "CLASS 3", "CLASS 4", "CLASS 5",
    "CLASS 6", "CLASS 7", "CLASS 8", "CLASS 9", "CLASS 11"
]

for v in vehicle_classes:
    box_hist(v)

# Truck & bus volumes
for v in ["Small_T", "Large_T", "Buses", "Volume bus"]:
    box_hist(v)
```

```
# Time variables
for v in ["TIME", "DAY", "Month", "Year"]:
    box_hist(v)

# Facility & operations
for v in ["LANE", "FAC", "Facility_Order"]:
    box_hist(v)

# Speed & congestion
for v in ["Avg_Speed", "freeflow", "Delta"]:
    box_hist(v)

# Correlation checks
sns.scatterplot(x=df["CLASS 1"], y=df["Autos"])
plt.title("CLASS 1 vs Autos")
plt.show()

sns.scatterplot(x=df["Facility_Order"], y=df["Avg_Speed"])
plt.title("Facility_Order vs Avg_Speed")
plt.show()

sns.scatterplot(x=df["Facility_Order"], y=df["freeflow"])
plt.title("Facility_Order vs freeflow")
plt.show()
```

3. Python Code for Answering Project Goals:

Busiest Times & Factors Affecting Traffic

```
import pandas as pd

import matplotlib.pyplot as plt



# Load Dataset

df = pd.read_csv("Cleaned_merged_data.csv")

df.fillna(0, inplace=True)
```

```
df["DATE"] = pd.to_datetime(df["DATE"])


# Create HOUR
if "HOUR" not in df.columns:
    df["HOUR"] = df["TIME"].apply(lambda x: int(str(int(x)).zfill(4)[:2]))
```

QUESTION 1: WHAT FACTORS AFFECT TRAFFIC?

```
factor_importance = {
    "Avg Speed": 0.9983,
    "EZPASS": 0.00078,
    "Hour": 0.00037,
    "Violations": 0.00034,
    "Month": 0.00014,
    "Peak Hour": 0.00005
}


fi_df = pd.DataFrame(list(factor_importance.items()), columns=["Factor", "Importance"])
print("\nTOP FACTORS AFFECTING TRAFFIC:\n")
print(fi_df.sort_values(by="Importance", ascending=False))


plt.figure()
plt.barh(fi_df["Factor"], fi_df["Importance"])
```

```
plt.title("Top Factors Affecting Traffic Usage")

plt.xlabel("Importance")

plt.ylabel("Factor")

plt.tight_layout()

plt.show()
```

QUESTION 2: VIOLATIONS BY TIME

Code:

```
#By year

import pandas as pd
df = pd.read_csv(" Cleaned_merged_data.txt", sep="\t", na_values="NULL")
df["DATE"] = pd.to_datetime(df["DATE"])
data = df[["DATE", "VIOLATION"]]
filtered = data[
   (data["DATE"] >= "2019-01-01") &
   (data["DATE"] <= "2025-12-31")
violations_per_year =
( filtered .groupby(filtered["DATE"].dt.year)["VIOLATION"] .sum() .reset_index() )
violations_per_year.columns = ["Year", "Total_Violations"]
print(violations_per_year)
```

Plot:

```
import matplotlib.pyplot as plt
plt.figure(figsize=(8,5))
bars = plt.bar( violations_by_year["Year"],
violations_by_year["Total_Violations"])
for bar in bars:
   height = bar.get_height()
   plt.text( bar.get_x() + bar.get_width() / 2, height,
      f"{int(height):,}",   # comma-separated
      ha="center", va="bottom", fontsize=9)
plt.xlabel("Year")
plt.ylabel("Total Violations")
plt.title("Violations per Year")
plt.tight_layout()
```

```
plt.show()
```

Code:

```
# By Month

filtered["Month_Num"] = filtered["DATE"].dt.month
filtered["Month_Name"] = filtered["DATE"].dt.strftime("%B")
violations_by_month_overall =
( filtered .groupby("Month_Num")["VIOLATION"] .sum() .reset_index() )
violations_by_month_overall["Month_Name"] =
violations_by_month_overall["Month_Num"].apply( lambda x: pd.to_datetime(str(x),
format="%m").strftime("%B") )
violations_by_month_overall = violations_by_month_overall.sort_values("Month_Num")
violations_by_month_overall = violations_by_month_overall[["Month_Name",
"VIOLATION"]] violations_by_month_overall.columns = ["Month", "Total_Violations"]
print(violations_by_month_overall)
```

Plot:

```
import matplotlib.pyplot as plt
plt.figure(figsize=(10,6))
bars = plt.bar(
   violations_by_month_overall["Month"],
   violations_by_month_overall["Total_Violations"])
 for bar in bars:
   height = bar.get_height()
   plt.text( bar.get_x() + bar.get_width() / 2, height,
      f"{int(height):,}",   # comma-separated numbers
      ha="center", va="bottom", fontsize=9)
plt.xticks(rotation=45)
plt.xlabel("Month")
plt.ylabel("Total Violations")
plt.title("Overall Violations by Month (2019–2025)")
plt.tight_layout()
plt.show()
```

Code:

```
# By Week

violations_by_week = (
   filtered.groupby([filtered["DATE"].dt.isocalendar().year,
```

```
        filtered["DATE"].dt.isocalendar().week])["VIOLATION"].sum().reset_index())
violations_by_week.columns = ["Year", "Week", "Total_Violations"]
print("\nViolations by Week:")
print(violations_by_week)
```

Plot:

```
import matplotlib.pyplot as plt
weekly_pivot = violations_by_week.pivot(index="Week", columns="Year",
values="Total_Violations")
plt.figure(figsize=(14,7))
for year in weekly_pivot.columns:
 Plt.plot( weekly_pivot.index, weekly_pivot[year], marker='o', label=str(year))
plt.xlabel("Week Number")
plt.ylabel("Total Violations")
plt.title("Weekly Toll Violations by Year (2019-2025)")
plt.xticks(range(1, 53))  # Weeks 1-52
plt.legend(title="Year")
plt.grid(True)
plt.tight_layout()
plt.show()
```

Code:

```
# By Facility
violations_by_facility =
( filtered .groupby("FAC_B")["VIOLATION"] .sum() .reset_index())
violations_by_facility.columns = ["Facility", "Total_Violations"]
print(violations_by_facility)
```

Plot:

```
import matplotlib.pyplot as plt
violations_by_facility = (filtered.groupby("FAC_B")["VIOLATION"].sum().reset_index())
violations_by_facility["VIOLATION_MILLIONS"] =
violations_by_facility["VIOLATION"] / 1_000_000
plt.figure(figsize=(10,6))
bars = plt.bar(violations_by_facility["FAC_B"],
violations_by_facility["VIOLATION_MILLIONS"], color='skyblue')
 for bar, value in zip(bars, violations_by_facility["VIOLATION"]):
   plt.text(bar.get_x() + bar.get_width()/2, bar.get_height(), f"{value:,}", ha='center',
va='bottom', fontsize=9)
plt.xlabel("Facility")
plt.ylabel("Total Violations (millions)")
plt.title("Total Toll Violations per Facility")
```

```
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

QUESTION 3 - BUSIEST TIMES

```
# Daily

daily = df.groupby("DATE")["Passenger_Volume"].sum()

# Weekly

weekly = daily.resample("W").sum()

# Monthly

monthly = daily.resample("M").sum()

# Yearly

yearly = daily.resample("Y").sum()

# Hourly

hourly = df.groupby("HOUR")["Passenger_Volume"].mean()

# BUSIEST TIME RESULTS (REAL)

busiest_day = daily.idxmax(), int(daily.max())

busiest_week = weekly.idxmax(), int(weekly.max())

busiest_month = monthly.idxmax(), int(monthly.max())

busiest_year = yearly.idxmax(), int(yearly.max())

busiest_hour = hourly.idxmax(), int(hourly.max())


print("\n==============================")

print("REAL BUSIEST TIME RESULTS")
```

```
print("==============================")
print("Busiest Hour:", busiest_hour[0], "→", busiest_hour[1])
print("Busiest Day:", busiest_day[0].date(), "→", busiest_day[1])
print("Busiest Week:", busiest_week[0].date(), "→", busiest_week[1])
print("Busiest Month:", busiest_month[0].strftime("%B %Y"), "→", busiest_month[1])
print("Busiest Year:", busiest_year[0].year, "→", busiest_year[1])

# VISUAL: BUSIEST HOUR
plt.figure()
plt.plot(hourly.index, hourly.values)
plt.title("Average Passenger Volume by Hour")
plt.xlabel("Hour of Day")
plt.ylabel("Passenger Volume")
plt.show()

# VISUAL: BUSIEST MONTH
plt.figure()
plt.plot(monthly.index, monthly.values)
plt.title("Monthly Passenger Volume (REAL DATA)")
plt.show()

# VISUAL: BUSIEST YEAR
```

```
plt.figure()

plt.plot(yearly.index, yearly.values)

plt.title("Yearly Passenger Volume (REAL DATA)")

plt.show()
```

QUESTION 4 – CONGESTION PRCING IN 2025

```
df_2025 = df[df["Year"] == 2025].copy()

q4_fac = df_2025.groupby("FAC").agg(avg_speed=("Avg_Speed",
"mean"),avg_passenger("Passenger_Volume", "mean")).reset_index()



# Visualization

plt.figure(figsize=(10,6))

plt.scatter(q4_fac["avg_speed"], q4_fac["avg_passenger"], s=100)

for i, txt in enumerate(q4_fac["FAC"]):

    plt.annotate(txt, (q4_fac["avg_speed"].iloc[i],

                       q4_fac["avg_passenger"].iloc[i]))

plt.xlabel("Average Speed")

plt.ylabel("Average Passenger Volume")

plt.title("Congestion Proxy by Facility (2025)")

plt.grid(True)

plt.show()
```

QUESTION 5 – FORECAST FACILITY USAGE BEYOND 2025

```
import numpy as np
```

```
import pandas as pd
import matplotlib.pyplot as plt

# Aggregate yearly passenger totals
yearly_total = (df.groupby("Year")["Passenger_Volume"].sum().sort_index())
# Fit a linear trend using polynomial regression (degree = 1)
coef = np.polyfit(yearly_total.index.values.astype(int),yearly_total.values.astype(float),1)
trend = np.poly1d(coef)

# Forecast next 5 years beyond last observed year
forecast_years = np.arange(yearly_total.index.max() + 1,yearly_total.index.max() + 6)
forecast_vals = trend(forecast_years)

# Visualization
plt.figure(figsize=(10, 5))
yearly_total.index,yearly_total.values, 'o-',label="Observed")
plt.plot(forecast_years,forecast_vals, 's--',label="Forecast")
plt.xlabel("Year")
plt.ylabel("Total Passenger Volume")
plt.title("Trend-Based Forecast Beyond Last Observed Year")
plt.legend()
plt.grid(True)
plt.show()
```

4. PYTHON CODE USED FOR DEVELOPING MACHINE LEARNING MODELS

Import Libraries

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import StandardScaler, OneHotEncoder

from sklearn.compose import ColumnTransformer

from sklearn.pipeline import Pipeline

from sklearn.metrics import mean_squared_error, r2_score, accuracy_score, roc_auc_score

from sklearn.ensemble import RandomForestRegressor, RandomForestClassifier

from statsmodels.tsa.statespace.sarimax import SARIMAX
```

Load Dataset

```
df = pd.read_csv("Cleaned_merged_data.csv")

print("Dataset Shape:", df.shape)

print(df.columns.tolist())
```

Basic Cleaning

```
df.fillna(0, inplace=True)

df["DATE"] = pd.to_datetime(df["DATE"])
```

3.1 Regression Model - Passenger Volume Prediction

Objective: Predict Passenger_Volume using traffic, payment, speed, and time variables.

Features

```
df["HOUR"] = df["TIME"].apply(lambda x: int(str(int(x)).zfill(4)[:2]) if pd.notna(x) else 0)

df["is_peak"] = df["HOUR"].apply(lambda x: 1 if (7 <= x <= 9 or 16 <= x <= 18) else 0)

X = df[['EZPASS', 'VIOLATION', 'Month', 'LANEMODE', 'Avg_Speed', 'HOUR','is_peak']]

y = df['Passenger_Volume']
```

Preprocessing

```
preprocessor = ColumnTransformer([('num', StandardScaler(), ['EZPASS', 'VIOLATION', 'Month', 'Avg_Speed', 'HOUR', 'is_peak']), ('cat', OneHotEncoder(handle_unknown='ignore'), ['LANEMODE'])])
```

Random Forest Regression Model

```
model = RandomForestRegressor(n_estimators=200, random_state=42)

pipeline = Pipeline([ ('preprocessor', preprocessor), ('model', model)])

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

pipeline.fit(X_train, y_train)

y_pred = pipeline.predict(X_test)

rmse = mean_squared_error(y_test, y_pred, squared=False)

r2 = r2_score(y_test, y_pred)

print("Regression RMSE:", rmse)

print("Regression R2:", r2)
```

Feature Importance

```
feature_names = pipeline.named_steps['preprocessor'].get_feature_names_out()
feature_importance_df = pd.DataFrame({
    'Feature': feature_names,
    'Importance': pipeline.named_steps['model'].feature_importances_
}).sort_values(by='Importance', ascending=False)
print(feature_importance_df)
```

3.2 Classification Model - Peak Traffic Identification

Target = Passenger_Carrier Target Variable Creation

```
X = df[['Month', 'Avg_Speed', 'HOUR', 'is_peak']]
y = df['Passenger_Carrier']
```

CLASSIFICATION PIPELINE

```
preprocessor = ColumnTransformer([
    ('num', StandardScaler(), ['Month', 'Avg_Speed', 'HOUR', 'is_peak'])
])
```

Random Forest Classifier Model

```
model = RandomForestClassifier(n_estimators=200, random_state=42)
pipeline = Pipeline([
    ('preprocessor', preprocessor),
    ('model', model)])
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, stratify=y)
```

```
pipeline.fit(X_train, y_train)
y_pred = pipeline.predict(X_test)
print("Classification Accuracy:", accuracy_score(y_test, y_pred))
```

Feature Importance

```
feature_names = X.columns
feature_importance_df = pd.DataFrame({
    'Feature': feature_names,
    'Importance': pipeline.named_steps['model'].feature_importances_
}).sort_values(by='Importance', ascending=False)
print(feature_importance_df)
```

3.3 Time Series Forecasting - Facility Usage Beyond 2025

Daily Passenger Volume Aggregation

```
ts = df.groupby("DATE")["Passenger_Volume"].sum()
ts = ts.asfreq("D")
ts = ts.fillna(method="ffill")
train_size = int(len(ts) * 0.8)
train = ts.iloc[:train_size]
test = ts.iloc[train_size:]
```

SARIMAX Model

```
model = SARIMAX(train, order=(0,0,0), seasonal_order=(0,1,0,7))

results = model.fit(disp=False)

forecast = results.forecast(steps=len(test))

rmse = mean_squared_error(test, forecast, squared=False)

r2 = r2_score(test, forecast)

print("Time Series RMSE:", rmse)

print("Time Series R2:", r2)
```

Forecast Plot

```
plt.figure()

plt.plot(train.index, train, label="Train")

plt.plot(test.index, test, label="Actual")

plt.plot(test.index, forecast, label="Forecast")

plt.legend()

plt.title("Passenger Volume Forecast")

plt.show()
```