# Ride Demand Prediction Project: Complete Python Code

import pandas as pd

import matplotlib.pyplot as plt

import requests

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

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import mean\_absolute\_error, mean\_squared\_error

# -------------------------------

# 1. LOAD AND EXPLORE DATA

# -------------------------------

# Load Ride Data

print("Step 1: Loading Ride Data...")

ride\_data = pd.read\_csv("cleaned\_data.csv") # Original ride data

print("Ride Data Loaded Successfully!")

print("\nRide Data Preview:")

print(ride\_data.head())

# Load Weather Data

print("\nLoading Weather Data...")

weather\_data = pd.read\_csv("New York City Weather.csv") # Weather data collected via API

print("Weather Data Loaded Successfully!")

print("\nWeather Data Preview:")

print(weather\_data.head())

# Check basic statistics

print("\nRide Data Summary:")

print(ride\_data.describe())

print("\nWeather Data Summary:")

print(weather\_data.describe())

# -------------------------------

# 2. FETCH WEATHER DATA FROM API

# -------------------------------

def fetch\_weather\_data():

print("\nStep 2: Fetching Weather Data from API...")

# API endpoint and parameters

api\_key = "YOUR\_API\_KEY" # Replace with your Visual Crossing API key

url = f"https://weather.visualcrossing.com/VisualCrossingWebServices/rest/services/timeline/New+York+City,USA/2022-01-01/2022-12-31?unitGroup=metric&key={api\_key}&contentType=csv"

# Fetch data

response = requests.get(url)

if response.status\_code == 200:

with open("New\_York\_City\_Weather.csv", "wb") as f:

f.write(response.content)

print("Weather data saved as 'New\_York\_City\_Weather.csv'.")

else:

print(f"Failed to fetch weather data. Status Code: {response.status\_code}")

# Uncomment the following line to fetch weather data via API

# fetch\_weather\_data()

# -------------------------------

# 3. DATA CLEANING AND MERGING

# -------------------------------

print("\nStep 3: Cleaning and Merging Data...")

# Merge the two datasets on pickup\_datetime

merged\_data = pd.merge(ride\_data, weather\_data, left\_on="pickup\_datetime", right\_on="datetime", how="left")

# Keep only necessary columns

columns\_to\_keep = ["pickup\_datetime", "PUlocationID", "temp", "precip", "windspeed", "humidity", "conditions"]

merged\_data = merged\_data[columns\_to\_keep]

# Save the merged and cleaned data

merged\_data.to\_csv("refined\_merged\_data.csv", index=False)

print("Merged and cleaned data saved as 'refined\_merged\_data.csv'.")

# -------------------------------

# 4. HANDLING MISSING DATA

# -------------------------------

print("\nStep 4: Handling Missing Data...")

# Load the merged data

data = pd.read\_csv("refined\_merged\_data.csv")

# Convert pickup\_datetime to datetime

data["pickup\_datetime"] = pd.to\_datetime(data["pickup\_datetime"])

# Resample data monthly and interpolate missing values

data\_resampled = data.resample("M", on="pickup\_datetime").sum()

data\_resampled = data\_resampled.interpolate(method="linear")

# Save the interpolated data

data\_resampled.to\_csv("interpolated\_data.csv", index=False)

print("Missing values filled and saved as 'interpolated\_data.csv'.")

# -------------------------------

# 5. FEATURE ENGINEERING

# -------------------------------

print("\nStep 5: Feature Engineering...")

# Reload the data

data = pd.read\_csv("interpolated\_data.csv")

# Add month feature

data["month"] = pd.to\_datetime(data["pickup\_datetime"]).dt.month

# Add is\_weekend feature

data["day\_of\_week"] = pd.to\_datetime(data["pickup\_datetime"]).dt.day\_name()

data["is\_weekend"] = data["day\_of\_week"].isin(["Saturday", "Sunday"])

# Add weather flags

data["is\_rainy"] = data["precip"] > 0.1

data["is\_windy"] = data["windspeed"] > 20

data["is\_humid"] = data["humidity"] > 70

# Save feature-engineered data

data.to\_csv("feature\_engineered\_data.csv", index=False)

print("Feature-engineered data saved as 'feature\_engineered\_data.csv'.")

# -------------------------------

# 6. EXPLORATORY DATA ANALYSIS

# -------------------------------

print("\nStep 6: Exploratory Data Analysis...")

# Load feature-engineered data

data = pd.read\_csv("feature\_engineered\_data.csv")

# Monthly Ride Demand

monthly\_demand = data.groupby("month")["PUlocationID"].count()

monthly\_demand.plot(kind="bar", color="skyblue", figsize=(8, 5))

plt.title("Monthly Ride Demand")

plt.xlabel("Month")

plt.ylabel("Number of Rides")

plt.savefig("Figures/monthly\_demand.png")

plt.show()

# Ride Demand on Rainy vs Non-Rainy Days

rainy\_impact = data.groupby("is\_rainy")["PUlocationID"].count()

rainy\_impact.plot(kind="bar", color=["orange", "blue"], figsize=(6, 4))

plt.title("Ride Demand on Rainy vs Non-Rainy Days")

plt.xlabel("Rainy (True/False)")

plt.ylabel("Number of Rides")

plt.xticks([0, 1], ["Non-Rainy", "Rainy"], rotation=0)

plt.savefig("Figures/rainy\_vs\_nonrainy.png")

plt.show()

# -------------------------------

# 7. PREDICTIVE MODELING

# -------------------------------

print("\nStep 7: Predictive Modeling...")

# Define Features and Target

X = data[["is\_weekend", "is\_rainy", "is\_windy", "is\_humid", "month"]]

y = data["PUlocationID"]

# Train-Test Split

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

# Train Linear Regression Model

model = LinearRegression()

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

# Predict on Test Data

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

# Evaluate the Model

mae = mean\_absolute\_error(y\_test, y\_pred)

rmse = mean\_squared\_error(y\_test, y\_pred) \*\* 0.5

print("\nModel Evaluation Metrics:")

print(f"Mean Absolute Error (MAE): {mae}")

print(f"Root Mean Squared Error (RMSE): {rmse}")

# Visualize Actual vs Predicted Ride Counts

plt.figure(figsize=(8, 5))

plt.scatter(y\_test, y\_pred, alpha=0.7, color="blue")

plt.plot([y\_test.min(), y\_test.max()], [y\_test.min(), y\_test.max()], "r--", lw=2)

plt.xlabel("Actual Ride Count")

plt.ylabel("Predicted Ride Count")

plt.title("Actual vs Predicted Ride Counts")

plt.savefig("Figures/actual\_vs\_predicted.png")

plt.show()