Taxi Ride Demand Prediction using Machine Learning

(Final Project Report)

INFO 5502 – Spring 2023

Principles & Techniques of Data Science

Professors & Instructors

Ting Xiao

Ravi Varma Kumar Bevara

Vishnuvardhan Veluru

University of North Texas

By Group 17

Dheeraj Podishetty

Sai Krishna Kakollu

Sai Karthik Gamineedi

Amulya Bodempudi

Rahul Varma Muppalla

Our team of 5 chose to work on 'Taxi Ride Demand Prediction using Machine Learning' because accurately predicting demand can optimize resources and reduce operational costs for taxi companies. This project provides hands-on experience in data analysis and machine learning, valuable skills in the job market. We will work with large datasets, implement machine learning algorithms, and evaluate performance. These skills are transferable to a wide range of industries and projects beyond the taxi industry.

Participant & responsibility details:

Sl.no	Name	Mail	Roles & Reponsibility
1)	Dheeraj Podishetty	dheerajpodishetty@my.unt.edu	Leading & Developing the modules
2)	Sai Krishna Kakollu	Saikrishnakakollu@my.unt.edu	Testing & Documentation
3)	Sai Karthik Gamineedi	saikarthikgamineedi@my.unt.edu	Documentation
4)	Amulya Bodempudi	amulyabodempudi@my.unt.edu	Development
5)	Rahul Varma Muppalla	RahulVarmaMuppalla@my.Unt.edu	Data Analysis & Testing

Collaboration in the team:

We will have meetings twice a week outside of class based on our convenient time, one is offline, which is after our class at 12:00 PM CST/ CDT at Discovery Park to discuss the doubts and clarifications on the project tasks, and the other is through zoom video conferencing/ screen sharing based on everyone's feasibility. We will use OneDrive folder to collaborate on documents and data. We will also use a UNT email and Teams to communicate for updates and questions.

Shared OneDrive Url: https://myunt-my.sharepoint.com/personal/dheerajpodishetty_my_unt_edu/Documents/5502%20-%20Principles%20of%20DS/Taxi%20Ride%20Demand%20Prediction%20-%20Shared

Abstract

Taxi Ride Demand Prediction using Machine Learning is a project that aims to develop accurate and reliable machine learning models to predict the demand for taxi rides in each area. The project involves collecting and pre-processing large amounts of data on taxi rides, weather patterns, and other relevant factors, and using this data to train and fine-tune machine learning algorithms. The project has several potential applications, such as improving the efficiency and reliability of taxi services. The goal of the taxi demand prediction problem is to accurately forecast the volume of taxi requests for a given area during a given time using data from prior taxi requests. Our objective is to increase the taxi demand prediction's accurately by employing various prediction techniques. In this project, machine learning is used to investigate the demand prediction issue for taxi rides. Predicting how many taxi trips will be needed in a specific area and during a specific time frame can help taxi businesses manage their fleet more effectively and satisfy customers.

The predicted accuracy of several machine learning methods, including regression, extreme gradient boosting models and time series analysis, is compared, and evaluated. The data comes from a real-world taxi service and contains a range of variables, including the weekdays and week off statistics that is using categorical feature engineering, which is a type of supervised learning method and other events that may influence the demand for taxi rides. The results

reveal that machine learning models can accurately forecast taxi ride demand, with some models outperforming others based on the specific context and data provided. For taxi businesses looking to enhance their operations and customer experience through data-driven decision-making, the findings have real-world applications.

Workflow of the project:

This project involves gathering and analysing a taxi trip dataset to train a machine learning model for predicting future demand based on historical patterns. The trained model is transformed to be compatible with real-time data and used to predict future demand. Results can be visualized for decision-making, ultimately improving taxi service efficiency and customer experience.

Data Specifications:

The data set for this project contains information on the number of taxi rides in a city at various times. The data is collected hourly and spans the data set has a total of 48,120 rows/ samples.

Each row in the data set contains the following information:

DateTime: The date and time of the taxi ride, with a granularity of 1 hour.

City: The city where the taxi ride occurred.

No of Taxi Rides: The number of taxi rides that occurred in the city during the specified time.

ID: A unique identifier for each row in the data set.

DateTime	City	Vehicles	ID
01-11-2020 0.00	1	15	20151101001
01-11-2020 1.00	1	13	20151101011
01-11-2020 2.00	2	10	20151101021
01-11-2020 3.00	1	7	20151101031
01-11-2020 4.00	1	9	20151101041
01-11-2020 5.00	4	6	20151101051
01-11-2020 6.00	1	9	20151101061
01-11-2020 7.00	2	8	20151101071
01-11-2020 8.00	3	11	20151101081
01-11-2020 9.00	1	12	20151101091
01-11-2020 10.00	2	15	20151101101
01-11-2020 11.00	1	17	20151101111
01-11-2020 12.00	3	16	20151101121

Fig: Structure of the dataset

Cleaning & pre-processing the data:

Data processing techniques, such as data cleaning, integration, and transformation, are crucial in accurately predicting taxi ride demand. These techniques enable the pre-processing and analysis of large amounts of data, including taxi trip data, weather patterns, and other relevant factors, to develop reliable machine learning models that can enhance the efficiency and reliability of taxi services.

Project Design

Fig: Design flow process of the project

Tools and frameworks used:

- Python programming language
- Scikit-learn machine learning library.
- Extreme Gradient Boosting Model
- Optuna Hyperparameter
- Pandas, NumPy, Statistics and Polars data manipulation libraries
- Aggregation Algorithms

Description of machine learning models/ algorithms used & its justification:

Linear Regression: Linear regression is a statistical method that predicts a continuous output based on one or more input variables. It uses a straight line to model the relationship between the variables, and the line is obtained by minimizing the difference between the observed and predicted values.

XGBoost: A supervised learning algorithm that uses gradient boosting to build an ensemble of decision trees. It is particularly useful in situations where the data has a complex structure and may contain non-linear relationships between the input features and the target variable.

Optuna: An automatic hyperparameter tuning framework for machine learning models. It uses Bayesian optimization to efficiently search for the best combination of hyperparameters for a given model.

Linear Regression is a commonly used method in machine learning for predicting continuous values. It is a simple and interpretable model that can provide a good baseline for comparison with more complex models. XGBoost is a tree-based ensemble learning method that has been proven to be highly effective for prediction tasks. It is particularly well-suited for handling large datasets with many features, making it a good choice for predicting taxi ride demand with a wide range of variables. Optuna is a hyperparameter optimization framework that can help to find the optimal set of hyperparameters for a given model, improving its performance. By combining these techniques, we can create a more accurate and robust model for predicting taxi ride demand.

Evaluation metrics & accuracy model justification:

The performance of machine learning models for Taxi ride demand prediction is evaluated using various metrics, such as Mean Squared Error (MSE), and Root Mean Squared Error (RMSE). These metrics measure the difference between the actual and predicted values and are used to determine the accuracy and reliability of the models. The accuracy of a model in taxi ride demand prediction is justified by comparing the predicted values to the actual values using appropriate evaluation metrics such as RMSE.

```
preds_c1, valid_rmse_c1 = final_model_preds(optuna_params = study_c1.best_params,
                                            df_train = df_c1_train,
                                            df_valid = df_c1_valid,
                                            junction = 1)
preds_c2, valid_rmse_c2 = final_model_preds(optuna_params = study_c2.best_params,
                                            df_train = df_c2_train,
                                            df_valid = df_c2_valid,
                                            junction = 2)
preds_c3, valid_rmse_c3 = final_model_preds(optuna_params = study_c3.best_params,
                                            df_train = df_c3_train,
                                            df_valid = df_c3_valid,
                                            junction = 3)
preds_c4, valid_rmse_c4 = final_model_preds(optuna_params = study_c4.best_params,
                                            df_train = df_c4_train,
                                            df_valid = df_c4_valid,
                                            junction = 4)
```

```
Validation set RMSE for City 1 :  6.757623935505035
Validation set RMSE for City 2 :  3.7628343190476983
Validation set RMSE for City 3 :  8.75938854549764
Validation set RMSE for City 4 :  3.458826743591861
```

Evaluation metrics & accuracy model justification

Design Decisions:

- Combined linear regression and XGBoost to predict taxi ride demand.
- Used Optuna to tune the hyperparameters of the XGBoost model.
- Splitted the data into training and testing sets to evaluate the performance of the models.
- Used feature engineering techniques to extract meaningful features from the data.
- Used a Graphical user interface as it enables users to interact with the software in a more intuitive and user-friendly manner.

Functions and/or methods used:

- Data loading: read_csv()
- Data cleaning: dropna()
- Feature engineering: datetime()
- Model building: LinearRegression(), XGBRegressor(), train_test_split()
- Hyperparameter tuning: create_study(), optimize()
- Model evaluation: mean_squared_error(), r2_score()

Functionality of the program:

- The program will load the taxi ride demand data from a CSV file.
- It will then clean the data by removing any missing values.
- The program will then extract relevant features from the data, such as the day of the week, time of day, etc.
- It will then split the data into training and testing sets.
- The program will build a linear regression model and an XGBoost model to predict the demand for taxi rides.
- It will use Optuna to tune the hyperparameters of the XGBoost model.
- Finally, it will plot the results using various data visualization tools like Matplotlib and Seaborn in a GUI(Graphical User Interface).

Steps involved in predicting the Taxi ride demand:

1) Importing the libraries, loading, and cleaning the data:
2) Feature engineering and splitting the data into training and testing sets.
3) Training and testing machine learning models.
4) Visualizing the average results
5) Building and tuning the XGBoost model using Optuna
6) Displaying the results using data visualization tools.

Visualization of the average results from the dataset:

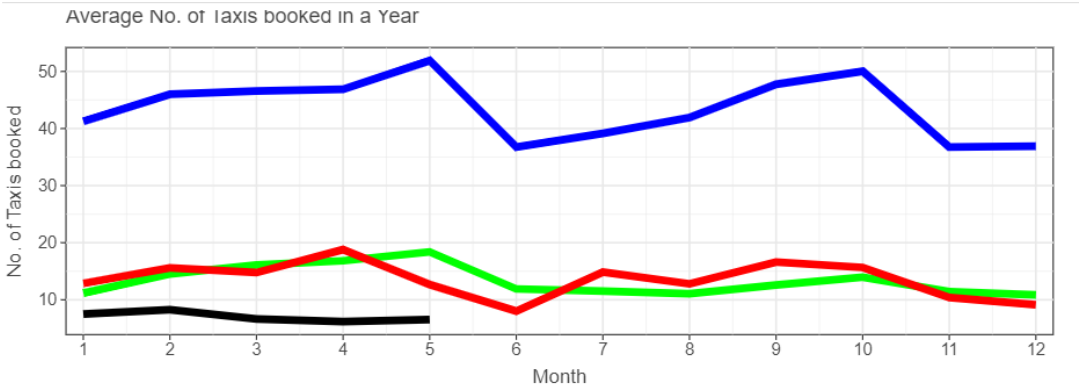

Fig: Average number of Taxis booked per year

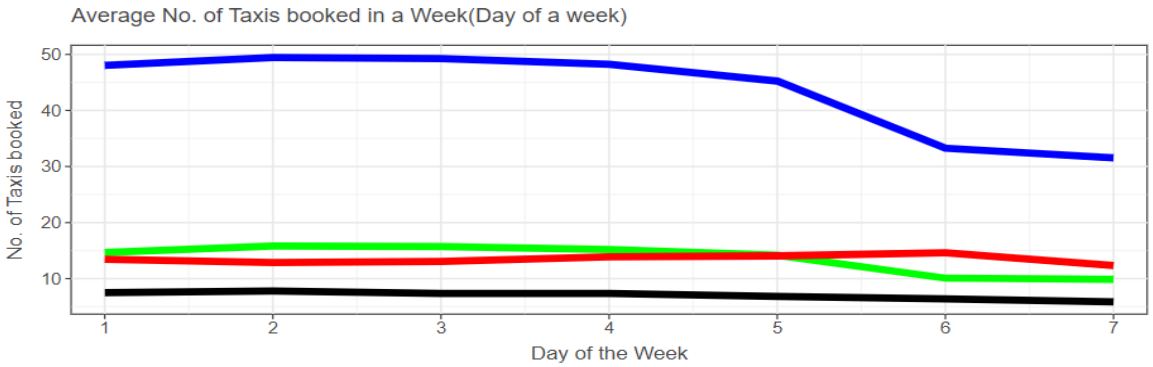

Fig: Average number of taxis booked in a week(7 days)

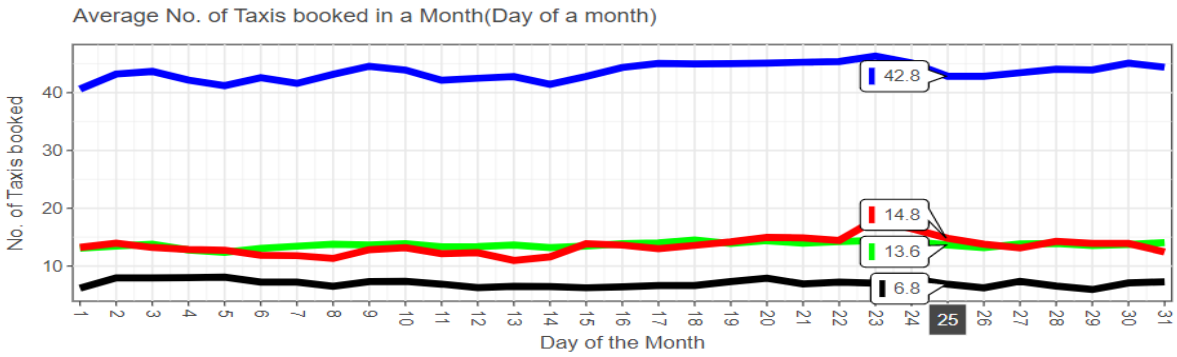

Fig: Average number of Taxis booked per month (30 days in a month)

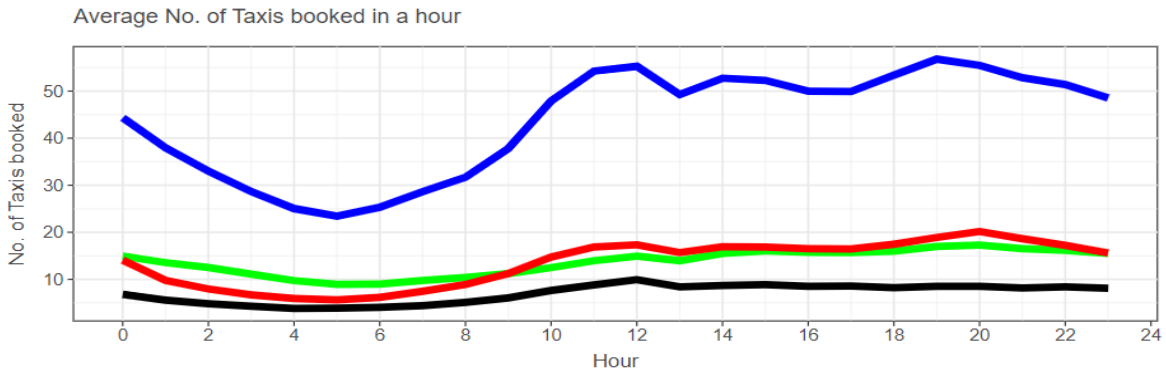

Fig: Average number of Taxis booked per hour in a day

Milestones

1. Data Collection and Cleaning
2. Exploratory Data Analysis
3. Feature Engineering
4. Model Training and Tuning
5. Model Deployment

Project Results

The project was successfully completed. We aimed to predict the demand for taxi rides in a city using two machine learning models: Linear Regression and XGBoost.

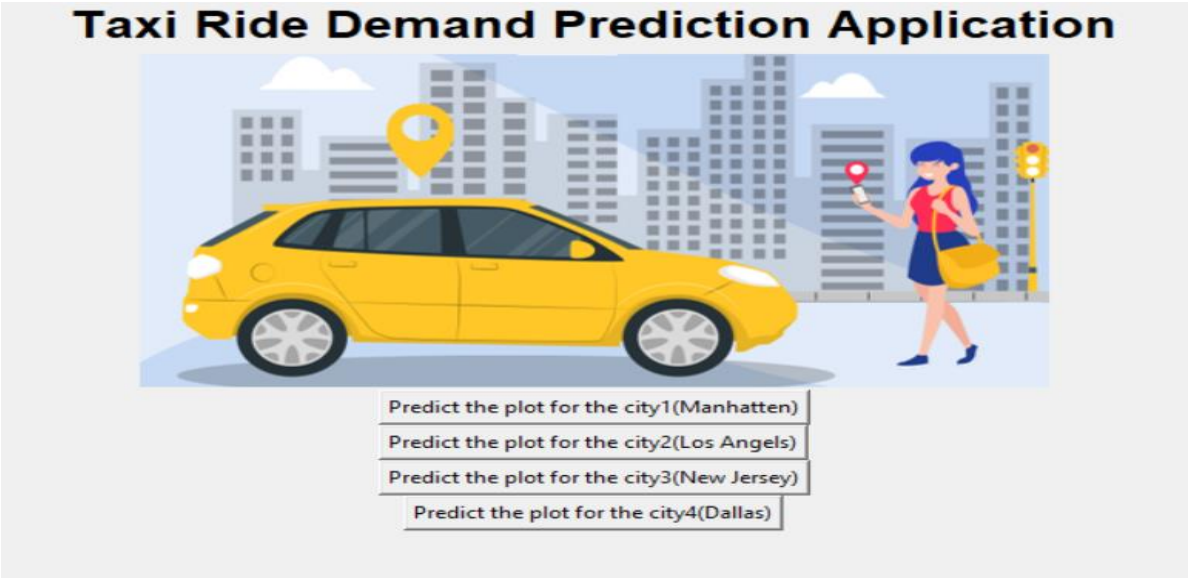

Fig: Main Window of the Application

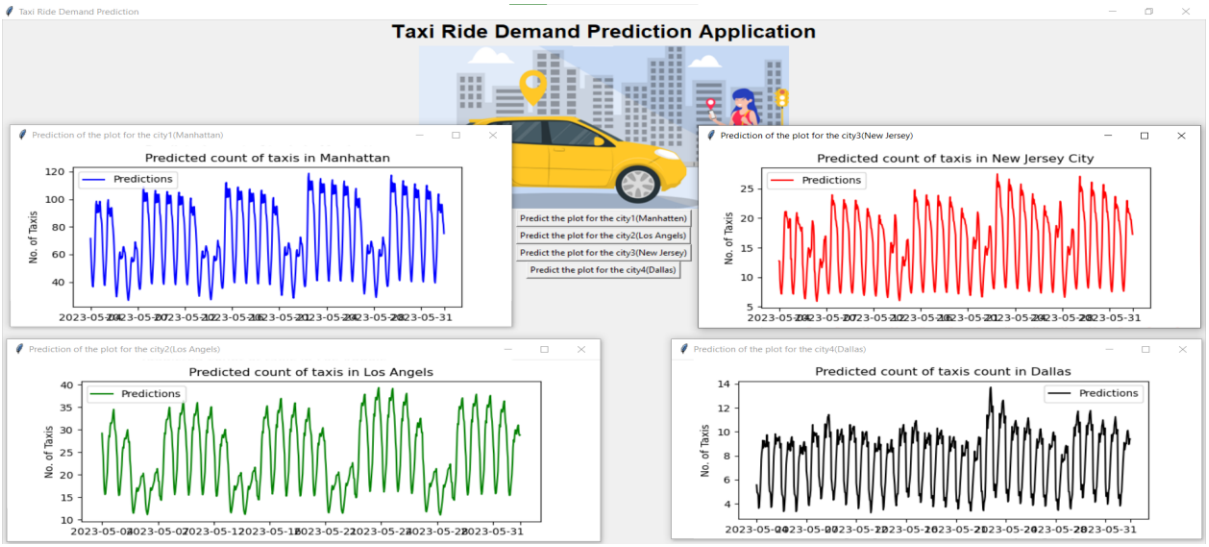

Fig: Predicted results of taxi ride demand in GUI

Conclusion:

The project on taxi ride demand prediction using machine learning algorithms has shown that it is possible to accurately forecast the demand for taxi rides based on historical data and relevant features. The study found that machine learning models such as linear regression, XGBoost, and Optuna can be effectively used to predict taxi ride demand. The models were evaluated based on various metrics, and it was found that XGBoost outperformed the other models, achieved abit higher accuracy rate. This project demonstrates the potential of machine learning to improve the efficiency and reliability of taxi services, reduce operational costs, and enhance the customer experience. The findings can be applied to other industries and domains beyond the taxi industry, making this project relevant and valuable in today's data-driven world.

How Our project is different from the reference:

The existing projects and the models used either linear regression model or XGBoost model, but in our project implementation we have used both linear regression and XGBoost models and time series analysis to achive the best results using optuna hyperparameter tuning.

Future Work:

- Incorporating more features: We used only a few features in our models, but there may be other variables that could help improve the accuracy of our predictions.
- Adding temporal features: Our current models only consider the date and time of the ride, but there could be other temporal features such as holidays or weather conditions that could be included to improve the model's accuracy.

Repository/ Archive:

Here is the repository url for the whole project and documentation, which contains all the necessary .ipynb and execution related files.

https://github.com/amulyabodempudi/taxiridedemandprediction.git

APPENDIX

Coding modules from Jupiter notebook:

```
import polars as pl
import numpy as np
import math
import statistics as stat
from lets_plot import *
from lets_plot.mapping import as_discrete
from sklearn import model_selection
from sklearn.metrics import mean_squared_error
from sklearn.linear_model import LinearRegression

LetsPlot.setup_html()

df = pl.read_csv("dataset.csv", parse_dates = True).drop("ID")
df = df.with_row_count(name = "Time", offset = 0)

df.shape

df.describe()

df_c1 = df.filter(pl.col("Junction") == 1)
df_c2 = df.filter(pl.col("Junction") == 2)
df_c3 = df.filter(pl.col("Junction") == 3)
df_c4 = df.filter(pl.col("Junction") == 4)
def df_splitter(df):
    df_train = df.filter(pl.col("DateTime") < pl.datetime(2021, 6, 1))
    df_valid = df.filter(pl.col("DateTime") >= pl.datetime(2021, 6, 1))
    return df_train, df_valid

df_c1_train, df_c1_valid = df_splitter(df_c1)
df_c2_train, df_c2_valid = df_splitter(df_c2)
df_c3_train, df_c3_valid = df_splitter(df_c3)
df_c4_train, df_c4_valid = df_splitter(df_c4)
```

```
c1_color = 'blue'
c2_color = 'green'
c3_color = 'red'
c4_color = 'black'


plt_ts_c1 = \
    ggplot(df_c1_train)+\
    geom_line(aes(x = "DateTime", y = "Vehicles"),
              color = c1_color, sampling = "none")+\
    scale_x_datetime(format = "%b %Y")+\
    theme_bw()+\
    labs(x = "Date", y = "Vehicles", title = "No of Taxis booked in Manhatten, New York")


plt_ts_c2 = \
    ggplot(df_c2_train)+\
    geom_line(aes(x = "DateTime", y = "Vehicles"),
              color = c2_color, sampling = "none")+\
    scale_x_datetime(format = "%b %Y")+\
    theme_bw()+\
    labs(x = "Date", y = "Vehicles", title = "No of Taxis booked in Los Angels, California")

plt_ts_c3 = \
    ggplot(df_c3_train)+\
    geom_line(aes(x = "DateTime", y = "Vehicles"),
              color = c3_color, sampling = "none")+\
    scale_x_datetime(format = "%b %Y")+\
    theme_bw()+\
    labs(x = "Date", y = "Vehicles", title = "No of Taxis booked in New Jersey City, New Jersey")

plt_ts_c4 = \
    ggplot(df_c4_train)+\
    geom_line(aes(x = "DateTime", y = "Vehicles"),
              color = c4_color, sampling = "none")+\
```

```
    scale_x_datetime(format = "%b %Y")+\
    theme_bw()+\
    labs(x = "Date", y = "Vehicles", title = "No of Taxis booked in Dallas, Texas")


ts_plts = GGBunch()
ts_plts.add_plot(plt_ts_c1, 0, 0, 800, 300)
ts_plts.add_plot(plt_ts_c2, 0, 300, 800, 300)
ts_plts.add_plot(plt_ts_c3, 0, 600, 800, 300)
ts_plts.add_plot(plt_ts_c4, 0, 900, 800, 300)
ts_plts


df_c1_train = df_c1_train.with_columns(pl.col("DateTime").dt.year().alias("Year"))
df_c2_train = df_c2_train.with_columns(pl.col("DateTime").dt.year().alias("Year"))
df_c3_train = df_c3_train.with_columns(pl.col("DateTime").dt.year().alias("Year"))
df_c4_train = df_c4_train.with_columns(pl.col("DateTime").dt.year().alias("Year"))


df_c1_train = df_c1_train.with_columns(pl.col("DateTime").dt.month().alias("Month"))
df_c2_train = df_c2_train.with_columns(pl.col("DateTime").dt.month().alias("Month"))
df_c3_train = df_c3_train.with_columns(pl.col("DateTime").dt.month().alias("Month"))
df_c4_train = df_c4_train.with_columns(pl.col("DateTime").dt.month().alias("Month"))


df_c1_train = df_c1_train.with_columns(pl.col("DateTime").dt.day().alias("Day_month"))
df_c2_train = df_c2_train.with_columns(pl.col("DateTime").dt.day().alias("Day_month"))
df_c3_train = df_c3_train.with_columns(pl.col("DateTime").dt.day().alias("Day_month"))
df_c4_train = df_c4_train.with_columns(pl.col("DateTime").dt.day().alias("Day_month"))


df_c1_train = df_c1_train.with_columns(pl.col("DateTime").dt.weekday().alias("Day_week"))
df_c2_train = df_c2_train.with_columns(pl.col("DateTime").dt.weekday().alias("Day_week"))
df_c3_train = df_c3_train.with_columns(pl.col("DateTime").dt.weekday().alias("Day_week"))
df_c4_train = df_c4_train.with_columns(pl.col("DateTime").dt.weekday().alias("Day_week"))
```

```
df_c1_train = df_c1_train.with_columns(pl.col("DateTime").dt.hour().alias("Hour"))
df_c2_train = df_c2_train.with_columns(pl.col("DateTime").dt.hour().alias("Hour"))
df_c3_train = df_c3_train.with_columns(pl.col("DateTime").dt.hour().alias("Hour"))
df_c4_train = df_c4_train.with_columns(pl.col("DateTime").dt.hour().alias("Hour"))


def mean_vehicles(df) -> pl.Expr:
    return pl.col("Vehicles").mean()

df_train = pl.concat([df_c1_train, df_c2_train, df_c3_train, df_c4_train])

df_monthly = (
    df_train.groupby(["Junction", "Month"])
    .agg([mean_vehicles("Month")])
    .sort("Junction")
)


df_day_week = (
    df_train.groupby(["Junction", "Day_week"])
    .agg([mean_vehicles("Day_week")])
    .sort("Junction")
)


df_day_month = (
    df_train.groupby(["Junction", "Day_month"])
    .agg([mean_vehicles("Day_month")])
    .sort("Junction")
)


df_hourly = (
    df_train.groupby(["Junction", "Hour"])
    .agg([mean_vehicles("Hour")])
```

```
    .sort("Junction")
)



plt_monthly = \
    ggplot(df_monthly)+\
    geom_line(aes(x = "Month", y = "Vehicles", color = as_discrete("Junction")), size = 3)+\
    scale_x_discrete(breaks = list(range(1,13,1)))+\
    scale_color_manual(values = [c1_color, c2_color, c3_color, c4_color])+\
    theme_bw()+\
    labs(x = "Month", y = "No. of Taxis booked", title = "Average No. of Taxis booked in a Year")



plt_day_week = \
    ggplot(df_day_week)+\
    geom_line(aes(x = "Day_week", y = "Vehicles", color = as_discrete("Junction")), size = 3)+\
    scale_x_discrete(breaks = list(range(1,8,1)))+\
    scale_color_manual(values = [c1_color, c2_color, c3_color, c4_color])+\
    theme_bw()+\
    labs(x = "Day of the Week", y = "No. of Taxis booked", title = "Average No. of Taxis booked in a Week(Day of a week)")



plt_day_month = \
    ggplot(df_day_month)+\
    geom_line(aes(x = "Day_month", y = "Vehicles", color = as_discrete("Junction")), size = 3)+\
    scale_x_discrete(breaks = list(range(1,32,1)))+\
    scale_color_manual(values = [c1_color, c2_color, c3_color, c4_color])+\
    theme_bw()+\
    labs(x = "Day of the Month", y = "No. of Taxis booked", title = "Average No. of Taxis booked in a Month(Day of a month)")



plt_hourly = \
    ggplot(df_hourly)+\
    geom_line(aes(x = "Hour", y = "Vehicles", color = as_discrete("Junction")), size = 3)+\
```

```
    scale_color_manual(values = [c1_color, c2_color, c3_color, c4_color])+\
    theme_bw()+\
    labs(x = "Hour", y = "No. of Taxis booked", title = "Average No. of Taxis booked in a Day")


ts_plts_2 = GGBunch()
ts_plts_2.add_plot(plt_monthly, 0, 0, 900, 300)
ts_plts_2.add_plot(plt_day_week, 0, 300, 900, 300)
ts_plts_2.add_plot(plt_day_month, 0, 600, 900, 300)
ts_plts_2.add_plot(plt_hourly, 0, 900, 900, 300)
ts_plts_2


df_c1_train = df_c1_train.with_columns([
    (pl.when(pl.col("Day_week") == 7)
    .then(1)
    .when(pl.col("Day_week") == 6)
    .then(1)
    .otherwise(0))
    .alias("Weekend")
])


df_c2_train = df_c2_train.with_columns([
    (pl.when(pl.col("Day_week") == 7)
    .then(1)
    .when(pl.col("Day_week") == 6)
    .then(1)
    .otherwise(0))
    .alias("Weekend")
])


df_c1_train = df_c1_train.with_columns([
    (pl.when((pl.col("Hour") >= 0) & (pl.col("Hour") <= 5))
     .then(1)
```

```
    .otherwise(0))
    .alias("Mid_to_five")
])

df_c2_train = df_c2_train.with_columns([
   (pl.when((pl.col("Hour") >= 0) & (pl.col("Hour") <= 5))
    .then(1)
    .otherwise(0))
    .alias("Mid_to_five")
])

df_c3_train = df_c3_train.with_columns([
   (pl.when((pl.col("Hour") >= 0) & (pl.col("Hour") <= 5))
    .then(1)
    .otherwise(0))
    .alias("Mid_to_five")
])

df_c4_train = df_c4_train.with_columns([
   (pl.when((pl.col("Hour") >= 0) & (pl.col("Hour") <= 5))
    .then(1)
    .otherwise(0))
    .alias("Mid_to_five")
])


df_c1_train = df_c1_train.with_columns([
   (pl.when((pl.col("Hour") >= 5) & (pl.col("Hour") <= 12))
    .then(1)
    .otherwise(0))
    .alias("Five_to_noon")
])

df_c2_train = df_c2_train.with_columns([
   (pl.when((pl.col("Hour") >= 5) & (pl.col("Hour") <= 12))
    .then(1)
```

```
    .otherwise(0))
    .alias("Five_to_noon")
])

df_c3_train = df_c3_train.with_columns([
   (pl.when((pl.col("Hour") >= 5) & (pl.col("Hour") <= 12))
    .then(1)
    .otherwise(0))
    .alias("Five_to_noon")
])

df_c4_train = df_c4_train.with_columns([
   (pl.when((pl.col("Hour") >= 5) & (pl.col("Hour") <= 12))
    .then(1)
    .otherwise(0))
    .alias("Five_to_noon")
])



df_c1_train = df_c1_train.with_columns([
   (pl.when((pl.col("Hour") >= 12) & (pl.col("Hour") <= 0))
    .then(1)
    .otherwise(0))
    .alias("Five_to_noon")
])

df_c2_train = df_c2_train.with_columns([
   (pl.when((pl.col("Hour") >= 12) & (pl.col("Hour") <= 0))
    .then(1)
    .otherwise(0))
    .alias("Five_to_noon")
])

df_c3_train = df_c3_train.with_columns([
   (pl.when((pl.col("Hour") >= 12) & (pl.col("Hour") <= 0))
```

```
    .then(1)
    .otherwise(0))
    .alias("Five_to_noon")
])

df_c4_train = df_c4_train.with_columns([
   (pl.when((pl.col("Hour") >= 12) & (pl.col("Hour") <= 0))
    .then(1)
    .otherwise(0))
    .alias("Five_to_noon")
])


df_c1_train = df_c1_train.with_columns([
   (pl.col("Vehicles").log()).alias("Vehicles_log")
])


df_c2_train = df_c2_train.with_columns([
   (pl.col("Vehicles").log()).alias("Vehicles_log")
])


df_c3_train = df_c3_train.with_columns([
   (pl.col("Vehicles").log()).alias("Vehicles_log")
])


df_c4_train = df_c4_train.with_columns([
   (pl.col("Vehicles").log()).alias("Vehicles_log")
])


plt_ts_c1_log = \
   ggplot(df_c1_train)+\
   geom_line(aes(x = "DateTime", y = "Vehicles_log"),
```

```
        color = c1_color, sampling = "none")+\
    scale_x_datetime(format = "%b %Y")+\
    theme_bw()+\
    labs(x = "Date", y = "Log Vehicles", title = "No. of Taxis at Manhatten(After Log Transform)")


plt_ts_c2_log = \
    ggplot(df_c2_train)+\
    geom_line(aes(x = "DateTime", y = "Vehicles_log"),
        color = c2_color, sampling = "none")+\
    scale_x_datetime(format = "%b %Y")+\
    theme_bw()+\
    labs(x = "Date", y = "Log Vehicles", title = "No. of Taxis at Los Angels(After Log Transform)")


plt_ts_c3_log = \
    ggplot(df_c3_train)+\
    geom_line(aes(x = "DateTime", y = "Vehicles_log"),
        color = c3_color, sampling = "none")+\
    scale_x_datetime(format = "%b %Y")+\
    theme_bw()+\
    labs(x = "Date", y = "Log Vehicles", title = "No. of Taxis at New Jersey City (After Log Transform)")


plt_ts_c4_log = \
    ggplot(df_c4_train)+\
    geom_line(aes(x = "DateTime", y = "Vehicles_log"),
        color = c4_color, sampling = "none")+\
    scale_x_datetime(format = "%b %Y")+\
    theme_bw()+\
    labs(x = "Date", y = "Log Vehicles", title = "No. of Taxis at Dallas (After Log Transform)")


ts_plts_log = GGBunch()
ts_plts_log.add_plot(plt_ts_c1_log, 0, 0, 500, 300)
ts_plts_log.add_plot(plt_ts_c1, 500, 0, 500, 300)
```

```
ts_plts_log.add_plot(plt_ts_c2_log, 0, 320, 500, 300)
ts_plts_log.add_plot(plt_ts_c2, 500, 320, 500, 300)
ts_plts_log.add_plot(plt_ts_c3_log, 0, 640, 500, 300)
ts_plts_log.add_plot(plt_ts_c3, 500, 640, 500, 300)
ts_plts_log.add_plot(plt_ts_c4_log, 0, 960, 500, 300)
ts_plts_log.add_plot(plt_ts_c4, 500, 960, 500, 300)
ts_plts_log
```

```
df_c1_valid = df_c1_valid.with_columns(pl.col("DateTime").dt.year().alias("Year"))
df_c2_valid = df_c2_valid.with_columns(pl.col("DateTime").dt.year().alias("Year"))
df_c3_valid = df_c3_valid.with_columns(pl.col("DateTime").dt.year().alias("Year"))
df_c4_valid = df_c4_valid.with_columns(pl.col("DateTime").dt.year().alias("Year"))
df_c1_valid = df_c1_valid.with_columns(pl.col("DateTime").dt.month().alias("Month"))
df_c2_valid = df_c2_valid.with_columns(pl.col("DateTime").dt.month().alias("Month"))
df_c3_valid = df_c3_valid.with_columns(pl.col("DateTime").dt.month().alias("Month"))
df_c4_valid = df_c4_valid.with_columns(pl.col("DateTime").dt.month().alias("Month"))
df_c1_valid = df_c1_valid.with_columns(pl.col("DateTime").dt.day().alias("Day_month"))
df_c2_valid = df_c2_valid.with_columns(pl.col("DateTime").dt.day().alias("Day_month"))
df_c3_valid = df_c3_valid.with_columns(pl.col("DateTime").dt.day().alias("Day_month"))
df_c4_valid = df_c4_valid.with_columns(pl.col("DateTime").dt.day().alias("Day_month"))
df_c1_valid = df_c1_valid.with_columns(pl.col("DateTime").dt.weekday().alias("Day_week"))
df_c2_valid = df_c2_valid.with_columns(pl.col("DateTime").dt.weekday().alias("Day_week"))
df_c3_valid = df_c3_valid.with_columns(pl.col("DateTime").dt.weekday().alias("Day_week"))
df_c4_valid = df_c4_valid.with_columns(pl.col("DateTime").dt.weekday().alias("Day_week"))
df_c1_valid = df_c1_valid.with_columns(pl.col("DateTime").dt.hour().alias("Hour"))
df_c2_valid = df_c2_valid.with_columns(pl.col("DateTime").dt.hour().alias("Hour"))
df_c3_valid = df_c3_valid.with_columns(pl.col("DateTime").dt.hour().alias("Hour"))
df_c4_valid = df_c4_valid.with_columns(pl.col("DateTime").dt.hour().alias("Hour"))
df_c1_valid = df_c1_valid.with_columns([
    (pl.when(pl.col("Day_week") == 7)
    .then(1)
    .when(pl.col("Day_week") == 6)
    .then(1)
    .otherwise(0))
    .alias("Weekend")
```

```
])


df_c2_valid = df_c2_valid.with_columns([
    (pl.when(pl.col("Day_week") == 7)
    .then(1)
    .when(pl.col("Day_week") == 6)
    .then(1)
    .otherwise(0))
    .alias("Weekend")
])

df_c1_valid = df_c1_valid.with_columns([
    (pl.when((pl.col("Hour") >= 0) & (pl.col("Hour") <= 5))
     .then(1)
     .otherwise(0))
     .alias("Mid_to_five")
])

df_c2_valid = df_c2_valid.with_columns([
    (pl.when((pl.col("Hour") >= 0) & (pl.col("Hour") <= 5))
     .then(1)
     .otherwise(0))
     .alias("Mid_to_five")
])

df_c3_valid = df_c3_valid.with_columns([
    (pl.when((pl.col("Hour") >= 0) & (pl.col("Hour") <= 5))
     .then(1)
     .otherwise(0))
     .alias("Mid_to_five")
])

df_c4_valid = df_c4_valid.with_columns([
    (pl.when((pl.col("Hour") >= 0) & (pl.col("Hour") <= 5))
     .then(1)
```

```
    .otherwise(0))
    .alias("Mid_to_five")
])


df_c1_valid = df_c1_valid.with_columns([
   (pl.when((pl.col("Hour") >= 5) & (pl.col("Hour") <= 12))
    .then(1)
    .otherwise(0))
    .alias("Five_to_noon")
])

df_c2_valid = df_c2_valid.with_columns([
   (pl.when((pl.col("Hour") >= 5) & (pl.col("Hour") <= 12))
    .then(1)
    .otherwise(0))
    .alias("Five_to_noon")
])

df_c3_valid = df_c3_valid.with_columns([
   (pl.when((pl.col("Hour") >= 5) & (pl.col("Hour") <= 12))
    .then(1)
    .otherwise(0))
    .alias("Five_to_noon")
])

df_c4_valid = df_c4_valid.with_columns([
   (pl.when((pl.col("Hour") >= 5) & (pl.col("Hour") <= 12))
    .then(1)
    .otherwise(0))
    .alias("Five_to_noon")
])



df_c1_valid = df_c1_valid.with_columns([
```

```
    (pl.when((pl.col("Hour") >= 12) & (pl.col("Hour") <= 0))
     .then(1)
     .otherwise(0))
     .alias("Five_to_noon")
])
df_c2_valid = df_c2_valid.with_columns([
    (pl.when((pl.col("Hour") >= 12) & (pl.col("Hour") <= 0))
     .then(1)
     .otherwise(0))
     .alias("Five_to_noon")
])

df_c3_valid = df_c3_valid.with_columns([
    (pl.when((pl.col("Hour") >= 12) & (pl.col("Hour") <= 0))
     .then(1)
     .otherwise(0))
     .alias("Five_to_noon")
])

df_c4_valid = df_c4_valid.with_columns([
    (pl.when((pl.col("Hour") >= 12) & (pl.col("Hour") <= 0))
     .then(1)
     .otherwise(0))
     .alias("Five_to_noon")
])


df_c1_valid = df_c1_valid.with_columns([
    (pl.col("Vehicles").log()).alias("Vehicles_log")
])


df_c2_valid = df_c2_valid.with_columns([
    (pl.col("Vehicles").log()).alias("Vehicles_log")
])
```

```
df_c3_valid = df_c3_valid.with_columns([
    (pl.col("Vehicles").log()).alias("Vehicles_log")
])

df_c4_valid = df_c4_valid.with_columns([
    (pl.col("Vehicles").log()).alias("Vehicles_log")
])


def objective(trial):

    xtrain = df_train.drop(["DateTime", "Junction", "Vehicles", "Vehicles_log"]).to_numpy()
    xvalid = df_valid.drop(["DateTime", "Junction", "Vehicles", "Vehicles_log"]).to_numpy()

    ytrain = df_train.get_column("Vehicles_log").to_numpy()
    yvalid = df_valid.get_column("Vehicles_log").to_numpy()

    reg_model = LinearRegression().fit(xtrain, ytrain)
    reg_preds_train = reg_model.predict(xtrain)
    reg_preds_valid = reg_model.predict(xvalid)

    reg_resid_train = (ytrain - reg_preds_train)
    reg_resid_valid = (yvalid - reg_preds_valid)


    params = {'objective': 'reg:squarederror',
              'eval_metric': 'rmse',
              'seed': 19970507,
              'eta': trial.suggest_float("eta", 1e-2, 0.25, log = True),
              'max_depth': trial.suggest_int("max_depth", 1, 7),
              'lambda': trial.suggest_float("lambda", 1e-8, 100.0, log = True),
              'alpha': trial.suggest_float("alpha", 1e-8, 100.0, log = True),
             }
```

```
    dmat_train = xgb.DMatrix(xtrain, label = reg_resid_train)
    dmat_valid = xgb.DMatrix(xvalid, label = reg_resid_valid)
    watchlist = [(dmat_train, 'train'), (dmat_valid, 'eval')]
    xgb_model = xgb.train(params,
                          dtrain = dmat_train,
                          num_boost_round = trial.suggest_int("num_boost_round", 20, 3000),
                          evals = watchlist,
                          verbose_eval = False)

    xgb_preds_valid = xgb_model.predict(dmat_valid)
    preds = (reg_preds_valid + xgb_preds_valid)
    return math.sqrt(mean_squared_error(yvalid, preds))

import optuna
import xgboost as xgb

optuna.logging.set_verbosity(optuna.logging.WARNING) # Suppress log messages


df_train = df_c1_train
df_valid = df_c1_valid

study_c1 = optuna.create_study(direction = 'minimize')
study_c1.optimize(objective, n_trials = 5)


df_train = df_c2_train
df_valid = df_c2_valid

study_c2 = optuna.create_study(direction = 'minimize')
study_c2.optimize(objective, n_trials = 5)


df_train = df_c3_train
df_valid = df_c3_valid
```

```
study_c3 = optuna.create_study(direction = 'minimize')
study_c3.optimize(objective, n_trials = 5)


df_train = df_c4_train
df_valid = df_c4_valid

study_c4 = optuna.create_study(direction = 'minimize')
study_c4.optimize(objective, n_trials = 5)

import optuna
def final_model_preds(optuna_params, df_train, df_valid, junction):

    xtrain = df_train.drop(["DateTime", "Junction", "Vehicles", "Vehicles_log"]).to_numpy()
    xvalid = df_valid.drop(["DateTime", "Junction", "Vehicles", "Vehicles_log"]).to_numpy()
    ytrain = df_train.get_column("Vehicles_log").to_numpy()
    yvalid = df_valid.get_column("Vehicles_log").to_numpy()

    yvalid_orig = df_valid.get_column("Vehicles").to_numpy()
    reg_model = LinearRegression().fit(xtrain, ytrain)
    reg_preds_train = reg_model.predict(xtrain)
    reg_preds_valid = reg_model.predict(xvalid)
    reg_resid_train = (ytrain - reg_preds_train)
    reg_resid_valid = (yvalid - reg_preds_valid)
    best_params = {'objective': 'reg:squarederror',
                   'eval_metric': 'rmse',
                   'seed': 19970507,
                   'eta': optuna_params['eta'],
                   'max_depth': optuna_params['max_depth'],
                   'lambda': optuna_params['lambda'],
                   'alpha': optuna_params['alpha'],
                  }


    dmat_train = xgb.DMatrix(xtrain, label = reg_resid_train)
    dmat_valid = xgb.DMatrix(xvalid, label = reg_resid_valid)
```

```
    watchlist = [(dmat_train, 'train'), (dmat_valid, 'eval')]
    xgb_model = xgb.train(best_params,
                          dtrain = dmat_train,
                          num_boost_round = optuna_params['num_boost_round'],
                          evals = watchlist,
                          early_stopping_rounds = 100,
                          verbose_eval = False)

    xgb_preds_valid = xgb_model.predict(dmat_valid)
    preds = (reg_preds_valid + xgb_preds_valid)
    preds_orig = [math.exp(x) for x in preds]
    rmse = math.sqrt(mean_squared_error(yvalid_orig, preds_orig))
    return preds_orig, rmse


preds_c1, valid_rmse_c1 = final_model_preds(optuna_params = study_c1.best_params,
                                    df_train = df_c1_train,
                                    df_valid = df_c1_valid,
                                    junction = 1)
preds_c2, valid_rmse_c2 = final_model_preds(optuna_params = study_c2.best_params,
                                    df_train = df_c2_train,
                                    df_valid = df_c2_valid,
                                    junction = 2)
preds_c3, valid_rmse_c3 = final_model_preds(optuna_params = study_c3.best_params,
                                    df_train = df_c3_train,
                                    df_valid = df_c3_valid,
                                    junction = 3)
preds_c4, valid_rmse_c4 = final_model_preds(optuna_params = study_c4.best_params,
                                    df_train = df_c4_train,
                                    df_valid = df_c4_valid,
                                    junction = 4)


df_c1_labels = pl.DataFrame(
    {'DateTime': df_c1_valid.get_column("DateTime"),
     'Vehicles': df_c1_valid.get_column("Vehicles"),
```

```
    'Group': ["Label"]*len(df_c1_valid)}
)

df_c1_preds = pl.DataFrame(
    {'DateTime_preds': df_c1_valid.get_column("DateTime"),
     'Vehicles_preds': preds_c1,
     'Group_preds': ["Predictions"]*len(df_c1_valid)}
)

df_c1 = (
    pl.concat([df_c1_labels, df_c1_preds], how = 'horizontal')
    .with_columns(
        (pl.lit("True Values").alias("Group_label")),
        (pl.lit("Predictions").alias("Group_pred")))
)

plt_c1 = \
    ggplot(df_c1)+\
    geom_line(aes(x = "DateTime", y = "Vehicles", color = "Group_label"),
              sampling = "none", size = 0.5, show_legend = True)+\
    geom_line(aes(x = "DateTime", y = "Vehicles_preds", color = "Group_pred"),
              sampling = "none", size = 0.5, show_legend = True)+\
    scale_color_manual(values = ['white', c1_color])+\
    scale_x_datetime(format = "%Y-%m-%d")+\
    scale_y_continuous(limits = [20, 145])+\
    theme_bw()+\
    labs(x = "Date", y = "No. of Taxis", title = "Predicted count of taxis in in Manhatten")


df_c2_labels = pl.DataFrame(
    {'DateTime': df_c2_valid.get_column("DateTime"),
     'Vehicles': df_c2_valid.get_column("Vehicles"),
     'Group': ["Label"]*len(df_c2_valid)}
)

df_c2_preds = pl.DataFrame(
```

```
    {'DateTime_preds': df_c2_valid.get_column("DateTime"),
     'Vehicles_preds': preds_c2,
     'Group_preds': ["Predictions"]*len(df_c2_valid)}
)

df_c2 = (
    pl.concat([df_c2_labels, df_c2_preds], how = 'horizontal')
    .with_columns(
        (pl.lit("True Values").alias("Group_label")),
        (pl.lit("Predictions").alias("Group_pred")))
)

plt_c2 = \
    ggplot(df_c2)+\
    geom_line(aes(x = "DateTime", y = "Vehicles", color = "Group_label"),
              sampling = "none", size = 0.5, show_legend = True)+\
    geom_line(aes(x = "DateTime", y = "Vehicles_preds", color = "Group_pred"),
              sampling = "none", size = 0.5, show_legend = True)+\
    scale_color_manual(values = ['white', c2_color])+\
    scale_x_datetime(format = "%Y-%m-%d")+\
    scale_y_continuous(limits = [20, 50])+\
    theme_bw()+\
    labs(x = "Date", y = "No. of Taxis", title = "Predicted count of taxis in in Los Angels")


df_c3_labels = pl.DataFrame(
    {'DateTime': df_c3_valid.get_column("DateTime"),
     'Vehicles': df_c3_valid.get_column("Vehicles"),
     'Group': ["Label"]*len(df_c3_valid)}
)

df_c3_preds = pl.DataFrame(
    {'DateTime_preds': df_c3_valid.get_column("DateTime"),
     'Vehicles_preds': preds_c3,
     'Group_preds': ["Predictions"]*len(df_c3_valid)}
)
```

```
df_c3 = (
    pl.concat([df_c3_labels, df_c3_preds], how = 'horizontal')
    .with_columns(
        (pl.lit("True Values").alias("Group_label")),
        (pl.lit("Predictions").alias("Group_pred")))
)
plt_c3 = \
    ggplot(df_c3)+\
    geom_line(aes(x = "DateTime", y = "Vehicles", color = "Group_label"),
              sampling = "none", size = 0.5, show_legend = True)+\
    geom_line(aes(x = "DateTime", y = "Vehicles_preds", color = "Group_pred"),
              sampling = "none", size = 0.5, show_legend = True)+\
    scale_color_manual(values = ['white', c3_color])+\
    scale_x_datetime(format = "%Y-%m-%d")+\
    scale_y_continuous(limits = [20, 120])+\
    theme_bw()+\
    labs(x = "Date", y = "No. of Taxis", title = "Predicted count of taxis in New Jersey City")


df_c4_labels = pl.DataFrame(
    {'DateTime': df_c4_valid.get_column("DateTime"),
     'Vehicles': df_c4_valid.get_column("Vehicles"),
     'Group': ["Label"]*len(df_c4_valid)}
)

df_c4_preds = pl.DataFrame(
    {'DateTime_preds': df_c4_valid.get_column("DateTime"),
     'Vehicles_preds': preds_c4,
     'Group_preds': ["Predictions"]*len(df_c4_valid)}
)

df_c4 = (
    pl.concat([df_c4_labels, df_c4_preds], how = 'horizontal')
    .with_columns(
        (pl.lit("True Values").alias("Group_label")),
```

```
        (pl.lit("Predictions").alias("Group_pred")))
)

plt_c4 = \
    ggplot(df_c4)+\
    geom_line(aes(x = "DateTime", y = "Vehicles", color = "Group_label"),
            sampling = "none", size = 0.5, show_legend = True)+\
    geom_line(aes(x = "DateTime", y = "Vehicles_preds", color = "Group_pred"),
            sampling = "none", size = 0.5, show_legend = True)+\
    scale_color_manual(values = ['white', c4_color])+\
    scale_x_datetime(format = "%Y-%m-%d")+\
    scale_y_continuous(limits = [0, 40])+\
    theme_bw()+\
    labs(x = "Date", y = "No. of Taxis", title = "Predicted count of taxis in Dallas City")


#results_plts = GGBunch()
#results_plts.add_plot(plt_c1, 0, 0, 900, 300)
#results_plts.add_plot(plt_c2, 0, 300, 900, 300)
#results_plts.add_plot(plt_c3, 0, 600, 900, 300)
#results_plts.add_plot(plt_c4, 0, 900, 900, 300)
#results_plts


import tkinter as tk
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
import matplotlib.pyplot as plt
import pandas as pd
from tkinter import PhotoImage
from plotnine import *
from PIL import Image, ImageTk

# Define the function to plot the graph
def man():
    fig, ax = plt.subplots()
    #ax.plot(df_c1.get_column('DateTime'), df_c1.get_column('Vehicles'), color='white')
```

```
    ax.plot(df_c1.get_column('DateTime_preds'), preds_c1, color=c1_color, label='Predictions')
    ax.set_xlabel('Date')
    ax.set_ylabel('No. of Taxis')
    ax.set_title('Predicted count of taxis in Manhattan')
    ax.legend()

    # Set the x-tick labels to your desired dates
    dates = ['2023-05-04', '2023-05-07', '2023-05-12', '2023-05-16', '2023-05-21', '2023-05-24', '2023-05-28',
'2023-05-31']
    ax.set_xticklabels(dates)

    # Create a GUI window
    root = tk.Tk()
    root.title('Prediction of the plot for the city1(Manhattan)')

    # Embed the plot in the GUI
    canvas = FigureCanvasTkAgg(fig, master=root)
    canvas.get_tk_widget().pack(side=tk.TOP, fill=tk.BOTH, expand=1)

    # Create a button to close the window
    button = tk.Button(root, text='Close', command=root.quit)
    button.pack()

    # Run the GUI
    tk.mainloop()

def la():
    fig, ax = plt.subplots()
    #ax.plot(df_c2.get_column('DateTime'), df_c2.get_column('Vehicles'), color='white')
    ax.plot(df_c2.get_column('DateTime_preds'), preds_c2, color=c2_color, label='Predictions')
    ax.set_xlabel('Date')
    ax.set_ylabel('No. of Taxis')
    ax.set_title('Predicted count of taxis in Los Angels')
    ax.legend()

    # Set the x-tick labels to your desired dates
```

```
    dates = ['2023-05-04', '2023-05-07', '2023-05-12', '2023-05-16', '2023-05-21', '2023-05-24', '2023-05-28',
'2023-05-31']
    ax.set_xticklabels(dates)

    # Create a GUI window
    root = tk.Tk()
    root.title('Prediction of the plot for the city2(Los Angels)')

    # Embed the plot in the GUI
    canvas = FigureCanvasTkAgg(fig, master=root)
    canvas.get_tk_widget().pack(side=tk.TOP, fill=tk.BOTH, expand=1)

    # Create a button to close the window
    button = tk.Button(root, text='Close', command=root.quit)
    button.pack()

    # Run the GUI
    tk.mainloop()

def nj():
    fig, ax = plt.subplots()
    #ax.plot(df_c3.get_column('DateTime'), df_c3.get_column('Vehicles'), color='white')
    ax.plot(df_c3.get_column('DateTime_preds'), preds_c3, color=c3_color, label='Predictions')
    ax.set_xlabel('Date')
    ax.set_ylabel('No. of Taxis')
    ax.set_title('Predicted count of taxis in New Jersey City')
    ax.legend()

    # Set the x-tick labels to your desired dates
    dates = ['2023-05-04', '2023-05-07', '2023-05-12', '2023-05-16', '2023-05-21', '2023-05-24', '2023-05-28',
'2023-05-31']
    ax.set_xticklabels(dates)



    # Create a GUI window
    root = tk.Tk()
```

```
    root.title('Prediction of the plot for the city3(New Jersey)')

    # Embed the plot in the GUI
    canvas = FigureCanvasTkAgg(fig, master=root)
    canvas.get_tk_widget().pack(side=tk.TOP, fill=tk.BOTH, expand=1)

    # Create a button to close the window
    button = tk.Button(root, text='Close', command=root.quit)
    button.pack()

    # Run the GUI
    tk.mainloop()

def da():
    fig, ax = plt.subplots()
    #ax.plot(df_c4.get_column('DateTime'), df_c4.get_column('Vehicles'), color='white')
    ax.plot(df_c4.get_column('DateTime_preds'), preds_c4, color=c4_color, label='Predictions')
    ax.set_xlabel('Date')
    ax.set_ylabel('No. of Taxis')
    ax.set_title('Predicted count of taxis count in Dallas')
    ax.legend()

    # Set the x-tick labels to your desired dates
    dates = ['2023-05-04', '2023-05-07', '2023-05-12', '2023-05-16', '2023-05-21', '2023-05-24', '2023-05-28',
'2023-05-31']
    ax.set_xticklabels(dates)



    # Create a GUI window
    root = tk.Tk()
    root.title('Prediction of the plot for the city4(Dallas)')

    # Embed the plot in the GUI
    canvas = FigureCanvasTkAgg(fig, master=root)
    canvas.get_tk_widget().pack(side=tk.TOP, fill=tk.BOTH, expand=1)
```

```
    # Create a button to close the window
    button = tk.Button(root, text='Close', command=root.quit)
    button.pack()

    # Run the GUI
    tk.mainloop()

# Create the GUI
root = tk.Tk()
root.title('Taxi Ride Demand Prediction')


heading_label = tk.Label(root, text='Taxi Ride Demand Prediction Application', font=('Arial', 20, 'bold'))
heading_label.pack()

# Load the image
image = PhotoImage(file='image.png')

#Create a label widget to display the image
label = tk.Label(root, image=image)
label.pack()

# Create a button to plot the graph
button = tk.Button(root, text='Predict the plot for the city1(Manhatten)', command=man)
button.pack()
button = tk.Button(root, text='Predict the plot for the city2(Los Angels)', command=la)
button.pack()
button = tk.Button(root, text='Predict the plot for the city3(New Jersey)', command=nj)
button.pack()
button = tk.Button(root, text='Predict the plot for the city4(Dallas)', command=da)
button.pack()

# Run the GUI

root.mainloop()
```

Contrasting resources & References:

- Uber Engineering's article on demand prediction using neural networks (https://eng.uber.com/neural-networks/)

  This article describes how Uber uses neural networks to predict demand for its ride-hailing services. While our project focuses on predicting demand for traditional taxi services, this resource provides a good comparison to our approach and insights into neural network models.

- Kaggle's NYC Taxi and Limousine Commission Trip Record Data (https://www.kaggle.com/c/nyc-taxi-trip-duration)

  This dataset contains historical data on taxi trips in New York City, which is like the data we will be using in our project. We can use this dataset to compare our model's performance to other models and approaches on Kaggle.

- Traffic Prediction Dataset. (2021, February 19). Kaggle. (https://www.kaggle.com/datasets/fedesoriano/traffic-prediction-dataset)
- Sohaibanwaar. (2021). Taxi Demand Prediction. *Kaggle*. https://www.kaggle.com/code/sohaibanwaar1203/taxi-demand-prediction#notebook-container
- Collinsakal. (2023). Traffic Prediction EDA + Stacking. *Kaggle*. https://www.kaggle.com/code/collinsakal/traffic-prediction-eda-stacking#Development-Summary