

# Uber Trip Demand Forecasting – Complete Project Documentation

## 1.Data Collection & Understanding

### Dataset Description

The dataset contains daily aggregated operational data for Uber dispatch bases.

#### Data Type:

Structured, tabular, time-series dataset.

#### Time Granularity:

Daily records.

### Columns Overview

Column	Type	Description
dispatching_base_number	Categorical	Unique base ID
date	Date	Daily timestamp
active_vehicles	Integer	Number of active vehicles (capacity)
trips	Integer	Total completed trips (target variable)

## 2.Problem Definition

### Goal

To predict **daily trip demand** using historical operational data.

### Business Problem

Uber operations face:

- Demand fluctuations
- Over/under fleet allocation
- Idle vehicle inefficiency

- Poor high-demand preparedness

### ❑ Core Question:

Can we forecast tomorrow's demand using historical patterns?

## 3.Data Preprocessing

We prepared raw data before modeling.

### ✔ Converted `date` column to `datetime`

Enabled time-based feature extraction.

### ✔ Extracted time-based features:

- `day_of_week`
- `month`

### ✔ Created Weekend Indicator:

Binary feature (`is_weekend`)

### ✔ Sorted Data Chronologically:

To preserve time-series order.

## 4. Feature Engineering

This is where model intelligence was built.

### Lag Features (Time Dependency)

Created:

- `lag_1` → Previous day demand
- `lag_7` → Previous week demand

These capture seasonality and trend.

## Operational Efficiency Feature

Created:

```
trips_per_vehicle = trips / active_vehicles
```

Captures dispatch efficiency.

## One-Hot Encoding

Encoded:

```
dispatching_base_number
```

Into:

- dispatching\_base\_number\_B02598
- dispatching\_base\_number\_B02617
- dispatching\_base\_number\_B02682
- dispatching\_base\_number\_B02764
- dispatching\_base\_number\_B02765

## Final Feature Count

Total features used in modeling:

12 engineered features

# 5. Model Development

## Baseline Model

Linear Regression

Used to establish performance benchmark.

## Final Model

Random Forest Regressor

## Why?

- Captures nonlinear relationships
- Handles outliers better
- Works well with tabular data
- No heavy feature scaling required

# 6. Model Validation Strategy

❑ Important Decision

We did NOT use random split.

Instead:

## ✓ Time-Based Split

- Train on past data
- Test on future data

Why?

Because time-series must preserve order.

## Evaluation Metrics

- MAE (Mean Absolute Error)
- RMSE (Root Mean Squared Error)
- R<sup>2</sup> Score

## Result

Random Forest outperformed Linear Regression in:

- Lower MAE
- Lower RMSE
- Better stability

Selected as final production model.

# 7. Model Serialization (Saving)

After final tuning:

```
pickle.dump(best_rf, file)
```

Saved as:

```
uber_demand_model.pkl
```

Why?

- Reusability
- Deployment readiness
- No need for retraining

## 8. Web Application Development (Streamlit)

Converted ML model into interactive product.

### Framework Used

Streamlit

### App Features

- Wide dashboard layout
- Two-column input structure
- Sidebar model info
- Real-time prediction
- KPI metric display
- Plotly visualization

### Prediction Flow

1. User enters features
2. Features aligned with training format
3. Model predicts demand
4. Output displayed instantly

## 9. Version Control (GitHub)

Initialized repository:

```
git init
git add .
git commit
```

Purpose:

- Track changes
- Maintain version history
- Enable cloud deployment

## GitHub Integration

Created new GitHub repository.

Connected local project:

```
git remote add origin <repo_url>
git push -u origin main
```

GitHub became:

- ✓ Source of truth
- ✓ Cloud deployment trigger
- ✓ Portfolio showcase

## 11.Cloud Deployment (Streamlit Cloud)

Connected GitHub repo to:

<https://share.streamlit.io>

Selected:

- Repository
- Branch
- app.py

Streamlit automatically:

- Installed dependencies
- Loaded model
- Deployed live app

# Final Output

Public web application:

Accessible globally via public URL.

No local machine required.

## 12.Challenges Faced

- Feature mismatch errors
- Git remote conflicts
- Plotly dependency error
- Model version warnings
- One-hot encoding alignment issues

## 13.Solutions Implemented

- Matched feature order using DataFrame
- Used force push for Git conflicts
- Updated requirements.txt
- Fixed model serialization
- Rebuilt Git repo cleanly

## 14.Business Impact

This solution enables:

- Better fleet allocation
- Improved operational efficiency
- Demand spike preparedness
- Reduced idle vehicle cost
- Data-driven dispatch planning

## 15.Technical Stack

- Python
- Pandas
- NumPy
- Scikit-Learn

- Plotly
- Streamlit
- Git
- GitHub
- Streamlit Cloud

## 16.Full Lifecycle Completed

You successfully executed:

```
Data
→ Feature Engineering
→ Model Training
→ Validation
→ Model Saving
→ Web App Development
→ Git Version Control
→ GitHub Integration
→ Cloud Deployment
```

This is complete end-to-end ML lifecycle.

## Final Statement

This project demonstrates the ability to transform raw operational data into a deployed, production-ready machine learning forecasting application aligned with real business objectives.