E-Commerce Analysis

Tools & Libraries

- **Python Libraries:** pandas, numpy, scikit-learn, Prophet, matplotlib, seaborn
- **Machine Learning:** Random Forest Regressor for delivery time prediction
- **Time Series Forecasting:** Prophet for demand forecasting

Project Overview

This project focuses on predicting **delivery times** and **future product demand** using the Olist Brazilian e-commerce dataset.
The project has two main components:

1. **Delivery Time Prediction:** Predicting how many days late or early an order will be delivered.
2. **Demand Forecasting:** Predicting future product demand (number of orders) using historical order data.

Both components together help e-commerce companies **optimize logistics, plan inventory, and improve customer satisfaction**.

```
# import libraries

import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error, r2_score
import matplotlib.pyplot as plt
[31]
```

```
# loading data

import zipfile, os
zip_path = "archive (13).zip"
extract_path = "olist_data"

with zipfile.ZipFile(zip_path, 'r') as zip_ref:
    zip_ref.extractall(extract_path)

os.listdir(extract_path)
[8]
```

```
['olist_customers_dataset.csv',
 'olist_geolocation_dataset.csv',
 'olist_orders_dataset.csv',
 'olist_order_items_dataset.csv',
 'olist_order_payments_dataset.csv',
 'olist_order_reviews_dataset.csv',
 'olist_products_dataset.csv',
 'olist_sellers_dataset.csv',
```

```
orders = pd.read_csv(os.path.join(extract_path, "olist_orders_dataset.csv"), parse_dates=['order_purchase_timestamp','order_delivered_customer_dat
 'order_estimated_delivery_date'])
order_items = pd.read_csv(os.path.join(extract_path, "olist_order_items_dataset.csv"))
products = pd.read_csv(os.path.join(extract_path, "olist_products_dataset.csv"))
[10]
```

```
orders.head(5)
[19]
```

5 rows ⌄ 5 rows × 8 cols

	order_id	customer_id	order_status	order_purchase_timestamp	order_approved_at	orde
0	e481f51cbdc54678b7cc49136f2d6af7	9ef432eb6251297304e76186b10a928d	delivered	2017-10-02 10:56:33	2017-10-02 11:07:15	2017
1	53cdb2fc8bc7dce0b6741e2150273451	b0830fb4747a6c6d20dea0b8c802d7ef	delivered	2018-07-24 20:41:37	2018-07-26 03:24:27	2018
2	47770eb9100c2d0c44946d9cf07ec65d	41ce2a54c0b03bf3443c3d931a367089	delivered	2018-08-08 08:38:49	2018-08-08 08:55:23	2018
3	949d5b44dbf5de918fe9c16f97b45f8a	f88197465ea7920adcdbec7375364d82	delivered	2017-11-18 19:28:06	2017-11-18 19:45:59	2017
4	ad21c59c0840e6cb83a9ceb5573f8159	8ab97904e6daea8866dbdbc4fb7aad2c	delivered	2018-02-13 21:18:39	2018-02-13 22:20:29	2018

```
# size of datsets

print("Orders:", orders.shape)
print("Order Items:", order_items.shape)
print("Products:", products.shape)
[20]
```

```
Orders: (99441, 8)
Order Items: (112650, 7)
Products: (32951, 9)
```

```
# merging the datasets

data = order_items.merge(orders, on="order_id", how="left")
data = data.merge(products, on="product_id", how="left")
[11]
```

```
data.head(3)
[21]
```

3 rows ⌄ 3 rows × 21 cols Static Outp

	order_id	order_item_id	product_id	seller_id	shipping_limit_date	price	freight_
0	00010242fe8c5a6d1ba2dd792cb16214	1	4244733e06e7ecb4970a6e2683c13e61	48436dade18ac8b2bce089ec2a041202	2017-09-19 09:45:35	58.9	
1	00018f77f2f0320c557190d7a144bdd3	1	e5f2d52b802189ee658865ca93d83a8f	dd7ddc04e1b6c2c614352b383efe2d36	2017-05-03 11:05:13	239.9	
2	000229ec398224ef6ca0657da4fc703e	1	c777355d18b72b67abbeef9df44fd0fd	5b51032eddd242adc84c38acab88f23d	2018-01-18 14:48:30	199.0	

```
# Feature Engineering

data = data.dropna(subset=['order_delivered_customer_date','order_estimated_delivery_date'])
data['delivery_delay'] = (data['order_delivered_customer_date'] - data['order_estimated_delivery_date']).dt.days
data['purchase_to_delivery'] = (data['order_delivered_customer_date'] - data['order_purchase_timestamp']).dt.days
data['estimated_time'] = (data['order_estimated_delivery_date'] - data['order_purchase_timestamp']).dt.days
[12]
```

```
features = data[['price','freight_value','product_weight_g','product_length_cm','product_height_cm','product_width_cm','purchase_to_delivery','estimated_time']]
target = data['delivery_delay']
features = features.fillna(features.median())
target = target.fillna(0)
[15]
```


```
features = data[['price','freight_value','product_weight_g','product_length_cm','product_height_cm','product_width_cm','purchase_to_delivery','estimated_time']]
target = data['delivery_delay']
features = features.fillna(features.median())
target = target.fillna(0)
[15]
```


```
# Train and test split , scale and model

X_train, X_test, y_train, y_test = train_test_split(features, target, test_size=0.2, random_state=42)

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

model = RandomForestRegressor(n_estimators=100, random_state=42)
model.fit(X_train, y_train)
[16]
```

```
RandomForestRegressor(random_state=42)
```

```
# Evaluation

y_pred = model.predict(X_test)
mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print("MAE:", mae)
print("R2:", r2)
[18]
```

```
MAE: 0.4350900018311996
R2: 0.9969352621158742
```

```
# Prediction 1

sample_X = X_test[:5]
sample_y = y_test[:5]
sample_pred = model.predict(sample_X)

print("Actual Delivery Delays:", sample_y.values)
print("Predicted Delivery Delays:", sample_pred)
[22]
```

```
Actual Delivery Delays: [-18 -14   1  -6  -8]
Predicted Delivery Delays: [-17.91       -14.57         0.49        -6.           -8.76666667]
```

```
# Prediction 2

sample_X = X_test[20:30]
sample_y = y_test[20:30]
sample_pred = model.predict(sample_X)

print("Actual Delivery Delays:", sample_y.values)
print("Predicted Delivery Delays:", sample_pred)
[23]
```

```
Actual Delivery Delays: [ -7  -9 -11 -20 -12 -18  -9 -14 -13 -21]
Predicted Delivery Delays: [ -6.51        -9.53       -10.77       -20.59       -11.81857143
 -17.43        -9.32       -13.75       -13.74       -21.16      ]
```

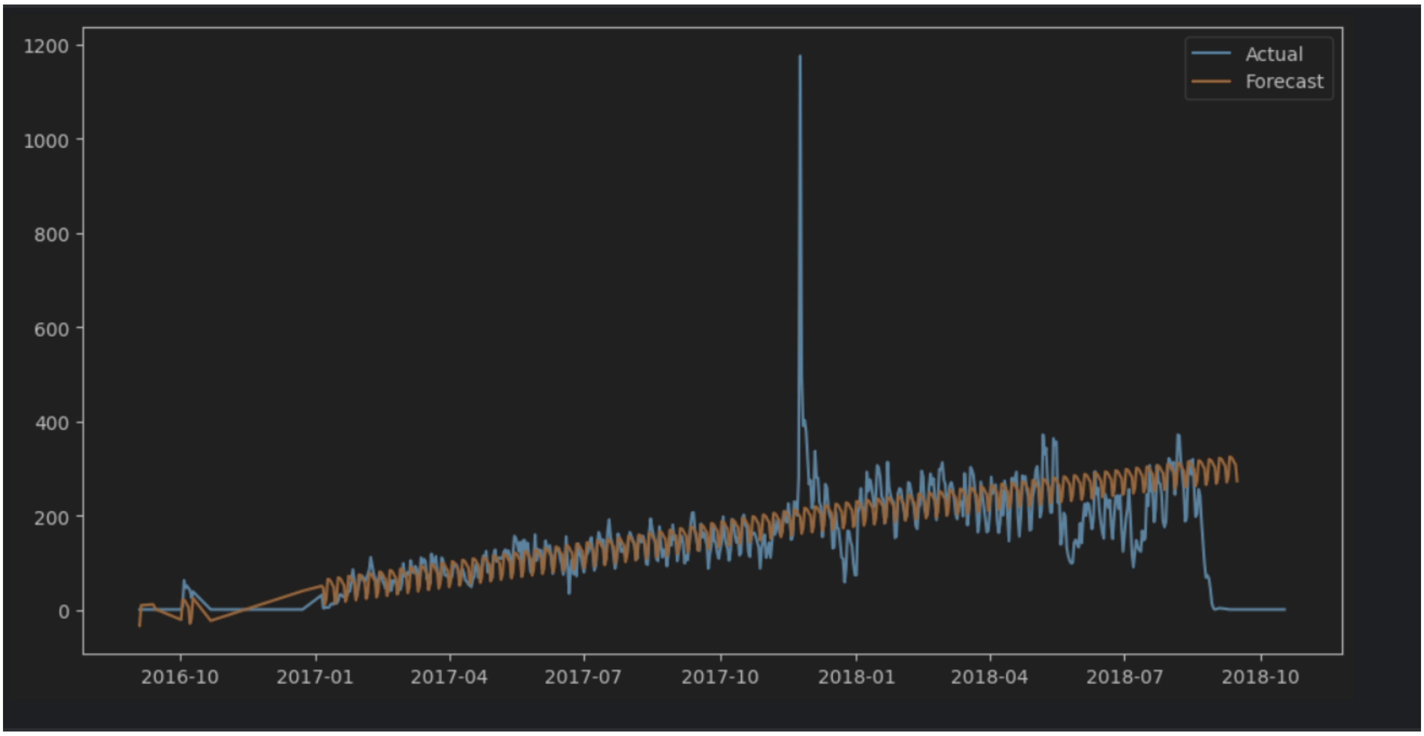
```
# Aggregate orders per day

orders['order_purchase_date'] = orders['order_purchase_timestamp'].dt.date
daily_orders = orders.groupby('order_purchase_date').size().reset_index(name='num_orders')
daily_orders = daily_orders.set_index('order_purchase_date')
[24]
```

```
# Train/test split

train_size = int(len(daily_orders)*0.8)
train = daily_orders.iloc[:train_size]
test = daily_orders.iloc[train_size:]
[25]
```

```
# importing module prophet

from prophet import Prophet

df = train.reset_index().rename(columns={'order_purchase_date':'ds','num_orders':'y'})
model = Prophet()
model.fit(df)

future = model.make_future_dataframe(periods=len(test))
forecast = model.predict(future)
[29]
```

```
18:52:23 - cmdstanpy - INFO - Chain [1] start processing
18:52:23 - cmdstanpy - INFO - Chain [1] done processing
```

```
# Actual vs Forecasted daily orders

plt.figure(figsize=(12,6))
plt.plot(daily_orders.index, daily_orders['num_orders'], label='Actual')
plt.plot(forecast['ds'], forecast['yhat'], label='Forecast')
plt.legend()
plt.show()
[30]
```



```python
from sklearn.metrics import mean_absolute_error

y_true = test['num_orders'].values
y_pred = forecast['yhat'][-len(test):].values
mae = mean_absolute_error(y_true, y_pred)
print("MAE:", mae)
```
[32]

```
MAE: 111.86564540968585
```

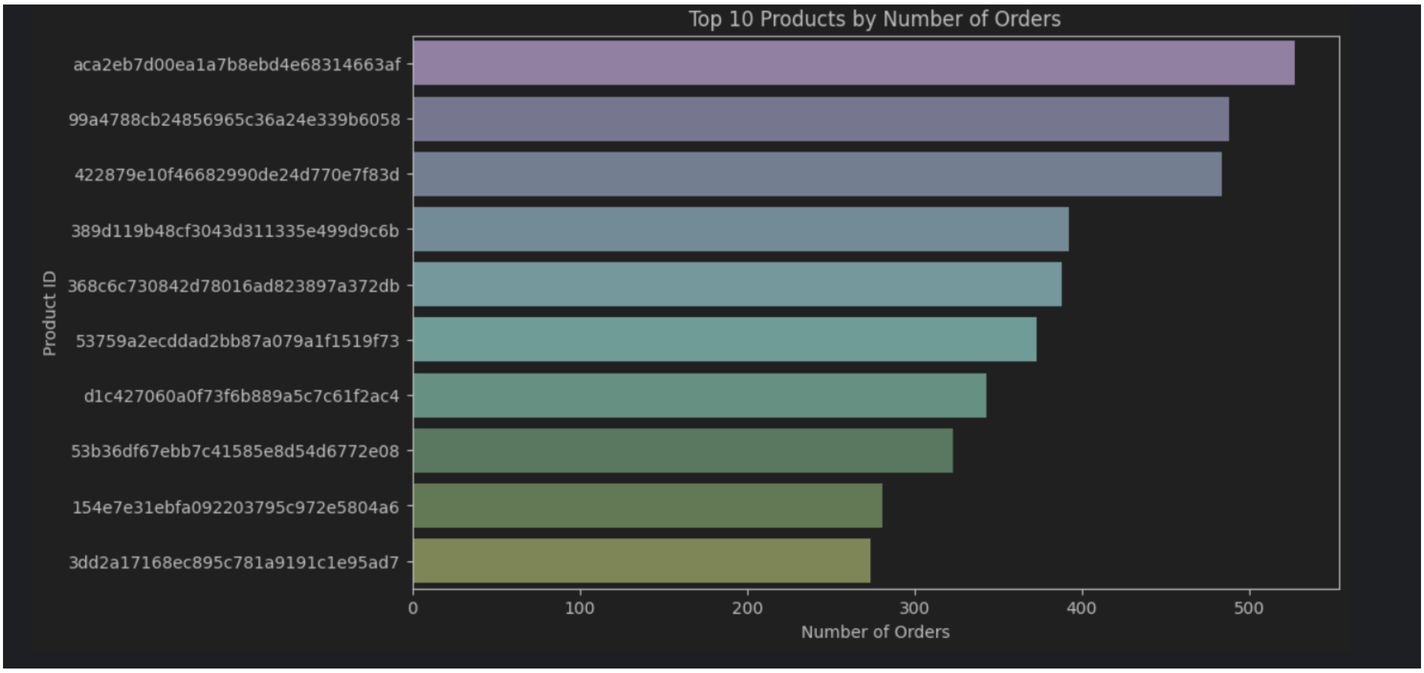
```python
# Prediction 3

train = daily_orders.iloc[:-7]
model = Prophet()
model.fit(train)

future = model.make_future_dataframe(periods=7)
forecast = model.predict(future)

print(forecast[['ds','yhat','yhat_lower','yhat_upper']].tail(7))
```
[35]

```
18:55:18 - cmdstanpy - INFO - Chain [1] start processing
18:55:18 - cmdstanpy - INFO - Chain [1] done processing
             ds        yhat  yhat_lower  yhat_upper
627  2018-09-21  107.888760   43.501123  174.384059
628  2018-09-22   73.050019    7.562915  133.891225
629  2018-09-23   87.184096   17.194584  156.291229
630  2018-09-24  132.213685   65.272119  202.442328
631  2018-09-25  134.261008   64.610669  201.469690
632  2018-09-26  131.086486   70.010645  196.375822
633  2018-09-27  121.491717   55.784001  188.457773
```

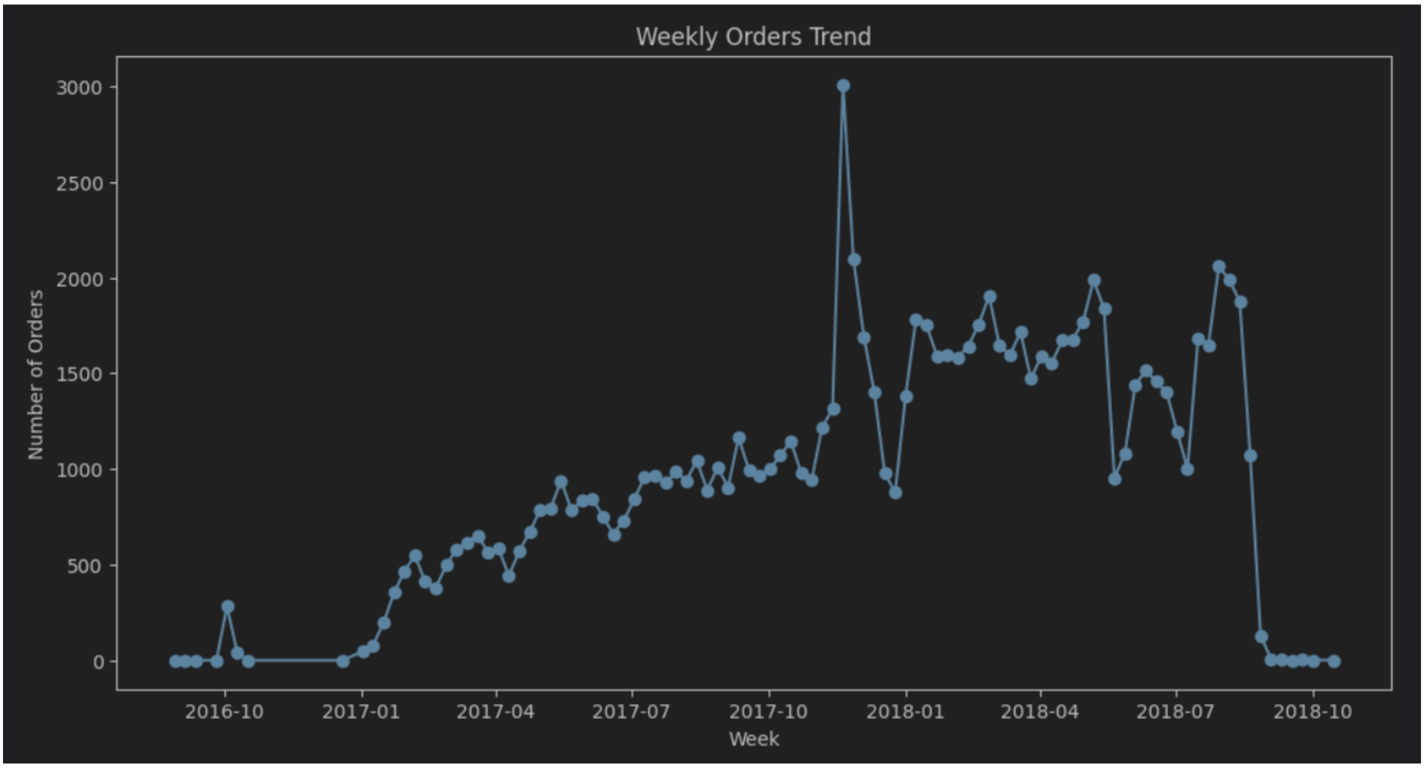
```python
# Prediction 4

test = daily_orders.iloc[-7:]
y_true = test['y'].values
y_pred = forecast['yhat'][-7:].values

print("Actual Orders:", y_true)
print("Predicted Orders:", y_pred)
```
[36]

```
Actual Orders: [1 1 1 1 1 1 1]
Predicted Orders: [107.88876041  73.05001866  87.18409575 132.213685   134.2610075
 131.08648634 121.49171726]
```

```python
# Top 10 products by orders

import seaborn as sns
top_products = order_items['product_id'].value_counts().head(10)
plt.figure(figsize=(10,6))
sns.barplot(x=top_products.values, y=top_products.index, palette='viridis')
plt.xlabel('Number of Orders')
plt.ylabel('Product ID')
plt.title('Top 10 Products by Number of Orders')
plt.show()
```
[40]




```
# weekly order trends

orders['order_week'] = orders['order_purchase_timestamp'].dt.to_period('W').apply(lambda r: r.start_time)
weekly_orders = orders.groupby('order_week').size()

plt.figure(figsize=(12,6))
plt.plot(weekly_orders.index, weekly_orders.values, marker='o')
plt.xlabel('Week')
plt.ylabel('Number of Orders')
plt.title('Weekly Orders Trend')
plt.show()
[39]
```



Dataset Used

We used the following tables from 'Brazilian E-Commerce Public Dataset by Olist' dataset :

Dataset	Usage
orders	Contains order timestamps, estimated and actual delivery dates
order_items	Product details in each order
products	Product features like weight, dimensions, category
order_reviews	Customer reviews (optional for ratings-based models)

Delivery Time Prediction

Predict the **delivery delay** in days:

delivery_dela = actual_delivery_date - estimated_delivery_date

- Positive → order delivered late
- Negative → order delivered early

Algorithm Used

Random Forest Regressor

- Ensemble learning method using multiple decision trees.
- Predictions are obtained by **averaging results** of all trees.
- Advantages:
  - Handles **non-linear relationships** well
  - Works with **mixed feature types**
  - Robust to **outliers and overfitting**
- Input: Features mentioned above
- Output: Predicted delivery delay in da

Demand Forecasting

Objective

Predict **future number of orders per day/week** to anticipate demand spikes and optimize inventory/logistics.

Algorithm Used

Facebook Prophet (Additive Time Series Model)

- **Model Formula:**

y(t)=g(t)+s(t)+h(t)+ϵty(t) = g(t) + s(t) + h(t) + epsilon_ty(t)=g(t)+s(t)+h(t)+ϵt

Where:

- $g(t)$ → trend component
- $s(t)$ → seasonality (daily, weekly, yearly)
- $h(t)$ → holidays/events
- ε_t → noise
- **Advantages:**
  - Handles **missing data, outliers, and irregular intervals**
  - Automatically models **trend + seasonality**
  - Produces **uncertainty intervals**

Forecasting Steps

1. The time series data was first **split into training and testing sets** to evaluate the model's performance.
2. The **Prophet model** was then fitted on the training data to learn trends and seasonality patterns in historical order volumes.
3. A **future dataframe** was generated for the required prediction horizon, representing the dates for which we wanted to forecast demand.
4. The model was used to **predict future orders (yhat)**, along with the **lower (yhat_lower) and upper (yhat_upper) confidence intervals**, providing a range of expected values.

Integration of Both Projects

- **Delivery Time Prediction** → estimates how late or early an order will be delivered
- **Demand Forecasting** → estimates how many orders are expected per day/week

Use Case:

- High predicted demand → potential **delivery delays**
- Low predicted demand → faster deliveries
- Combined, these models help **logistics and inventory planning**.

Visualizations

Visualization	Purpose
Actual vs Forecasted Orders (Line Plot)	Shows demand forecasting accuracy
Predicted vs Actual Delivery Delay (Scatter Plot)	Measures delivery prediction performance
Top 10 Products by Orders (Bar Plot)	Highlights high-demand products
Weekly Orders Trend (Line Plot)	Shows order seasonality patterns

Conclusion

1. **Delivery Time Prediction**: Using a Random Forest Regressor, we predicted whether an order will be delivered early, on time, or late based on product features, price, shipping cost, and historical delivery patterns. The model provides actionable insights that can help logistics teams optimize delivery schedules and improve customer satisfaction.
2. **Demand Forecasting**: Using Facebook Prophet, we forecasted future daily order volumes by capturing trends, seasonality, and fluctuations in historical order data. Accurate demand forecasting enables better inventory management, resource planning, and preparation for high-order periods, reducing delays and operational costs.
3. **Integration**: By combining delivery time prediction with demand forecasting, the system allows e-commerce platforms to anticipate high-demand periods and potential delivery delays simultaneously, creating a **comprehensive solution for logistics planning and decision-making**.

Overall, this project highlights the **practical use of machine learning and time series forecasting** to improve operational efficiency in e-commerce, and it can be further extended with additional features such as customer segmentation, product recommendations, or dynamic delivery routing for a more robust predictive system