```
print("Name:  BhavyaShah"     )
```

```
Name: BhavyaShah
```

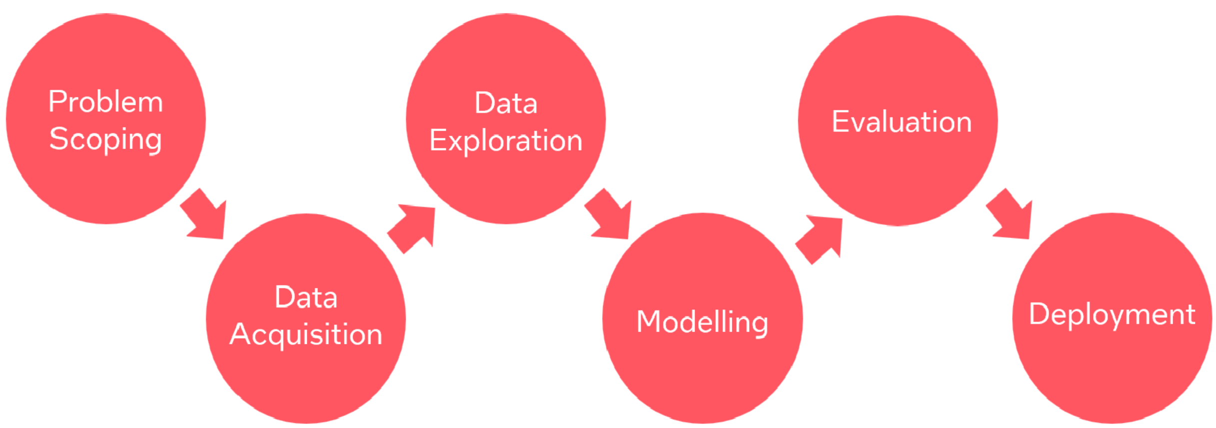
Order to Delivery (OTD) Time Forecasting

- Ease of e-commerce over the last decade, along with the recent COVID-19 pandemic, has seen a marked shift in consumer behavior and expectations when it comes to purchase - and more importantly, the delivery of goods. This has caused a paradigm shift in functioning of the supply chain. Along with delivery speed, consumers feel that the **transparency** around delivery time and shipment status are an equally important aspect of the fulfilment process. This has direct implications in customer churn/retention. More than half of consumers are less likely to shop with a retailer if the item is not delivered within 2 days of date promised. (source)

Order to Delivery Time Forecasting as a Supply Chain Optimization Usecase:

OTD forecasting is a key aspect of supply chain optimization as it helps to ensure that customers receive their products in a timely manner, it enables companies to make informed decisions about inventory, logistics and production which in turn helps to improve the overall efficiency of the supply chain.

AI Project Cycle



Context: Understanding the Problem Statement -------- Problem Scoping (AI Project Cycle - Step 1)

An ML based predictive solution for providing delivery time forecasting

- An ML based predictive solution for providing delivery time forecasting can provide great insights to e-commerce platforms. From a customer-facing point of view these insights would help e-commerce platforms to give more accurate delivery forecasts to customers and decrease customer churn. In addition, the ML based predictive solution will also allow e-commerce platforms to take pre-emptive actions to mitigate delays, costs, and loss of revenue.

- For the individual components of the ensemble model, we will use XGB, RF and SVM. These components will then be fed to a Voting Model, which is an ensemble model that combines the individual predictions to provide a final, consensus prediction. The final consensus prediction can be (1) a prediction for a wait time for a package and (2) if a delay will occur.

e2e-flow_stock

Import the useful Packages & Libraries

```
from math import radians, sin, cos, asin, sqrt
# This line imports specific functions from the math library that are
used for calculating the haversine distance between two points on a
globe.

import pandas as pd
# This line imports the Pandas library, which is a popular library for
data manipulation and analysis.

from sklearn.preprocessing import LabelEncoder, StandardScaler
# This line imports the LabelEncoder and StandardScaler classes from
the scikit-learn library, which are used for preprocessing data for
machine learning models.

import time
# This line imports the time module, which is used for timing the
execution of code

import numpy as np
# This line imports the NumPy library, which provides support for
numerical computing with Python.

----------------------------------------------------------------------
-----
ModuleNotFoundError                       Traceback (most recent call
last)
Cell In[7], line 4
      1 from math import radians, sin, cos, asin, sqrt
      2 # This line imports specific functions from the math library
that are used for calculating the haversine distance between two
points on a globe.
----> 4 import pandas as pd
      5 # This line imports the Pandas library, which is a popular
library for data manipulation and analysis.
      7 from sklearn.preprocessing import LabelEncoder, StandardScaler
```

```
ModuleNotFoundError: No module named 'pandas'
```

To know more about math click here

To know more about pandas click here

To know more about Sklearn' Preprocessing package click here

To know more about time click here

To know more about numpy click here

Dataset: Data Acquisition (AI Project Cycle - Step 2)

`Dataset downloading steps`

DataSet: https://www.kaggle.com/datasets/olistbr/brazilian-ecommerce

"brazilian-ecommerce.zip" will get downloaded in the current data folder folderd In the data folder execute the following command to unzip (Note: You may need to install unzip using: sudo apt-get install unzip) unzip brazilian-ecommerce.zip

Source - Before we delve into the proposed architecture of the solution pipeline, it is critical to understand the dataset and its schema. The dataset consists of real-world delivery details sourced from a Brazilian E-commerce Company which was scrubbed and anonymized. It can be found here.

It consists of multiple tables which include relevant information about the customer, seller, order, location etc. The individual tables are interconnected as shown in the following schema. Relevant features will be extracted/designed from this data and then be used to train our supervised ML model.

dataset-schema

Load/Read the Dataset

```
orders = pd.read_csv("./data/olist_orders_dataset.csv")
# This line reads the CSV file "olist_orders_dataset.csv" located in
the "./data/" directory and stores its contents as a Pandas DataFrame
named "orders".

items = pd.read_csv("./data/olist_order_items_dataset.csv")
# This line reads the CSV file "olist_order_items_dataset.csv" located
in the "./data/" directory and stores its contents as a Pandas
DataFrame named "items".

customers = pd.read_csv("./data/olist_customers_dataset.csv")
# This line reads the CSV file "olist_customers_dataset.csv" located
in the "./data/" directory and stores its contents as a Pandas
DataFrame named "customers".
```

```
sellers = pd.read_csv("./data/olist_sellers_dataset.csv")
# This line reads the CSV file "olist_sellers_dataset.csv" located in
the "./data/" directory and stores its contents as a Pandas DataFrame
named "sellers".

geo = pd.read_csv("./data/olist_geolocation_dataset.csv")
# This line reads the CSV file "olist_geolocation_dataset.csv" located
in the "./data/" directory and stores its contents as a Pandas
DataFrame named "geo".

products = pd.read_csv("./data/olist_products_dataset.csv")
# This line reads the CSV file "olist_products_dataset.csv" located in
the "./data/" directory and stores its contents as a Pandas DataFrame
named "products".
```

View the data

```
orders.head()

# The code above assumes that a Pandas DataFrame named orders has been
previously defined, and calls the head() method on that DataFrame.

# The head() method is used to display the first few rows of a
DataFrame, by default the first five rows. This can be useful for
quickly checking the structure and content of a DataFrame.

# So, this line of code will output the first five rows of the orders
DataFrame, assuming it has been previously defined.
```

```
                           order_id                       customer_id
\
0  e481f51cbdc54678b7cc49136f2d6af7  9ef432eb6251297304e76186b10a928d

1  53cdb2fc8bc7dce0b6741e2150273451  b0830fb4747a6c6d20dea0b8c802d7ef

2  47770eb9100c2d0c44946d9cf07ec65d  41ce2a54c0b03bf3443c3d931a367089

3  949d5b44dbf5de918fe9c16f97b45f8a  f88197465ea7920adcdbec7375364d82

4  ad21c59c0840e6cb83a9ceb5573f8159  8ab97904e6daea8866dbdbc4fb7aad2c


  order_status order_purchase_timestamp    order_approved_at  \
0    delivered      2017-10-02 10:56:33  2017-10-02 11:07:15
1    delivered      2018-07-24 20:41:37  2018-07-26 03:24:27
2    delivered      2018-08-08 08:38:49  2018-08-08 08:55:23
3    delivered      2017-11-18 19:28:06  2017-11-18 19:45:59
4    delivered      2018-02-13 21:18:39  2018-02-13 22:20:29

  order_delivered_carrier_date order_delivered_customer_date  \
0          2017-10-04 19:55:00           2017-10-10 21:25:13
```

```
1          2018-07-26 14:31:00          2018-08-07 15:27:45
2          2018-08-08 13:50:00          2018-08-17 18:06:29
3          2017-11-22 13:39:59          2017-12-02 00:28:42
4          2018-02-14 19:46:34          2018-02-16 18:17:02

  order_estimated_delivery_date
0           2017-10-18 00:00:00
1           2018-08-13 00:00:00
2           2018-09-04 00:00:00
3           2017-12-15 00:00:00
4           2018-02-26 00:00:00
```

```
products.head()
```

```
                         product_id  product_category_name  \
0  1e9e8ef04dbcff4541ed26657ea517e5             perfumaria
1  3aa071139cb16b67ca9e5dea641aaa2f                  artes
2  96bd76ec8810374ed1b65e291975717f          esporte_lazer
3  cef67bcfe19066a932b7673e239eb23d                  bebes
4  9dc1a7de274444849c219cff195d0b71  utilidades_domesticas

   product_name_lenght  product_description_lenght  product_photos_qty
\
0                 40.0                       287.0                 1.0

1                 44.0                       276.0                 1.0

2                 46.0                       250.0                 1.0

3                 27.0                       261.0                 1.0

4                 37.0                       402.0                 4.0


   product_weight_g  product_length_cm  product_height_cm
product_width_cm
0             225.0               16.0               10.0
14.0
1            1000.0               30.0               18.0
20.0
2             154.0               18.0                9.0
15.0
3             371.0               26.0                4.0
26.0
4             625.0               20.0               17.0
13.0
```

```
customers.head()
```

```
                        customer_id                customer_unique_id
\
```

```
0  06b8999e2fba1a1fbc88172c00ba8bc7  861eff4711a542e4b93843c6dd7febb0

1  18955e83d337fd6b2def6b18a428ac77  290c77bc529b7ac935b93aa66c333dc3

2  4e7b3e00288586ebd08712fdd0374a03  060e732b5b29e8181a18229c7b0b2b5e

3  b2b6027bc5c5109e529d4dc6358b12c3  259dac757896d24d7702b9acbbff3f3c

4  4f2d8ab171c80ec8364f7c12e35b23ad  345ecd01c38d18a9036ed96c73b8d066


   customer_zip_code_prefix          customer_city customer_state
0                     14409                 franca             SP
1                      9790  sao bernardo do campo             SP
2                      1151              sao paulo             SP
3                      8775        mogi das cruzes             SP
4                     13056               campinas             SP

sellers.head()

                          seller_id  seller_zip_code_prefix  \
0  3442f8959a84dea7ee197c632cb2df15                   13023
1  d1b65fc7debc3361ea86b5f14c68d2e2                   13844
2  ce3ad9de960102d0677a81f5d0bb7b2d                   20031
3  c0f3eea2e14555b6faeea3dd58c1b1c3                    4195
4  51a04a8a6bdcb23deccc82b0b80742cf                   12914

         seller_city seller_state
0           campinas           SP
1         mogi guacu           SP
2     rio de janeiro           RJ
3          sao paulo           SP
4  braganca paulista           SP
```

From a business perspective, we need to tackle two main challenges to ensure transparency for the customer:

1. Extracting insights/estimates into forecasted delivery
2. Increasing the accuracy of those insights and estimates

We can achieve the first objective by implementing multiple pipelines with separate objectives and the second one by utilizing techniques such as ensemble modeling to increase the accuracy of stand-alone ML components.

A schematic of the proposed reference architecture is shown in the following figure. We start off with Data Ingestion from the multiple tables of the dataset, followed by merging and preprocessing for feature extraction. We can use the features to extract delivery insights in the form of predicted wait time** as well as **likelihood of delivery delay**.

The former is a regression problem and the latter is a classification problem.

Data Preprocessing ------ Data Exploration(AI Project Cycle - Step 3)

View the information of data

```
# info() helps summarize the dataset- It gives basic information like number of non-null values, datatypes and memory usage
# It is a good practise to start by this information
products.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32951 entries, 0 to 32950
Data columns (total 9 columns):
 #   Column                      Non-Null Count  Dtype
---  ------                      --------------  -----
 0   product_id                  32951 non-null  object
 1   product_category_name       32341 non-null  object
 2   product_name_lenght         32341 non-null  float64
 3   product_description_lenght  32341 non-null  float64
 4   product_photos_qty          32341 non-null  float64
 5   product_weight_g            32949 non-null  float64
 6   product_length_cm           32949 non-null  float64
 7   product_height_cm           32949 non-null  float64
 8   product_width_cm            32949 non-null  float64
dtypes: float64(7), object(2)
memory usage: 2.3+ MB
```

```
customers.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 99441 entries, 0 to 99440
Data columns (total 5 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   customer_id               99441 non-null  object
 1   customer_unique_id        99441 non-null  object
 2   customer_zip_code_prefix  99441 non-null  int64
 3   customer_city             99441 non-null  object
 4   customer_state            99441 non-null  object
dtypes: int64(1), object(4)
memory usage: 3.8+ MB
```

```
sellers.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3095 entries, 0 to 3094
Data columns (total 4 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   seller_id               3095 non-null   object
```

```
 1   seller_zip_code_prefix  3095 non-null   int64
 2   seller_city             3095 non-null   object
 3   seller_state            3095 non-null   object
dtypes: int64(1), object(3)
memory usage: 96.8+ KB
```

```
orders.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 99441 entries, 0 to 99440
Data columns (total 8 columns):
 #   Column                         Non-Null Count  Dtype
---  ------                         --------------  -----
 0   order_id                       99441 non-null  object
 1   customer_id                    99441 non-null  object
 2   order_status                   99441 non-null  object
 3   order_purchase_timestamp       99441 non-null  object
 4   order_approved_at              99281 non-null  object
 5   order_delivered_carrier_date   97658 non-null  object
 6   order_delivered_customer_date  96476 non-null  object
 7   order_estimated_delivery_date  99441 non-null  object
dtypes: object(8)
memory usage: 6.1+ MB
```

Merge/Clean Data

To know more about how pandas dataframe merge function works click here

```
# Get the seller zip code of each order

middle = items[['order_id', 'seller_id']]
# This line creates a new dataframe middle that contains only the
order_id and seller_id columns from the items dataframe.

middle_2 = middle.merge(sellers[['seller_id',
'seller_zip_code_prefix']], on="seller_id", how="outer")
# This line merges the middle dataframe with the seller_id and
seller_zip_code_prefix columns from the sellers dataframe, creating a
new dataframe middle_2. The outer join type is used, which means that
all rows from both dataframes are included in the merged dataframe,
and missing values are filled with NaN.

orders = orders.merge(middle_2, on="order_id", how="left")
# This line merges the orders dataframe with the middle_2 dataframe on
the order_id column, creating a new orders dataframe. The left join
type is used, which means that all rows from the orders dataframe are
included in the merged dataframe, and missing values from the middle_2
dataframe are filled with NaN.

orders.head()
```

```
                            order_id                       customer_id  \
0  e481f51cbdc54678b7cc49136f2d6af7  9ef432eb6251297304e76186b10a928d

1  53cdb2fc8bc7dce0b6741e2150273451  b0830fb4747a6c6d20dea0b8c802d7ef

2  47770eb9100c2d0c44946d9cf07ec65d  41ce2a54c0b03bf3443c3d931a367089

3  949d5b44dbf5de918fe9c16f97b45f8a  f88197465ea7920adcdbec7375364d82

4  ad21c59c0840e6cb83a9ceb5573f8159  8ab97904e6daea8866dbdbc4fb7aad2c


  order_status order_purchase_timestamp    order_approved_at  \
0    delivered      2017-10-02 10:56:33  2017-10-02 11:07:15
1    delivered      2018-07-24 20:41:37  2018-07-26 03:24:27
2    delivered      2018-08-08 08:38:49  2018-08-08 08:55:23
3    delivered      2017-11-18 19:28:06  2017-11-18 19:45:59
4    delivered      2018-02-13 21:18:39  2018-02-13 22:20:29

  order_delivered_carrier_date order_delivered_customer_date  \
0          2017-10-04 19:55:00           2017-10-10 21:25:13
1          2018-07-26 14:31:00           2018-08-07 15:27:45
2          2018-08-08 13:50:00           2018-08-17 18:06:29
3          2017-11-22 13:39:59           2017-12-02 00:28:42
4          2018-02-14 19:46:34           2018-02-16 18:17:02

  order_estimated_delivery_date                         seller_id  \
0           2017-10-18 00:00:00  3504c0cb71d7fa48d967e0e4c94d59d9
1           2018-08-13 00:00:00  289cdb325fb7e7f891c38608bf9e0962
2           2018-09-04 00:00:00  4869f7a5dfa277a7dca6462dcf3b52b2
3           2017-12-15 00:00:00  66922902710d126a0e7d26b0e3805106
4           2018-02-26 00:00:00  2c9e548be18521d1c43cde1c582c6de8

   seller_zip_code_prefix
0                  9350.0
1                 31570.0
2                 14840.0
3                 31842.0
4                  8752.0

# Get customer zip code of each order
orders = orders.merge(customers[['customer_id',
'customer_zip_code_prefix']],
                  on='customer_id', how="left")
# The code above performs a left merge operation between two Pandas
dataframes named "orders" and "customers" using the "customer_id"
column as the joining key. It then selects the "customer_id" and
"customer_zip_code_prefix" columns from the "customers" dataframe and
merges them with the "orders" dataframe based on the matching
```

```
"customer_id" column.


orders.head()

                           order_id                       customer_id  \
0  e481f51cbdc54678b7cc49136f2d6af7  9ef432eb6251297304e76186b10a928d

1  53cdb2fc8bc7dce0b6741e2150273451  b0830fb4747a6c6d20dea0b8c802d7ef

2  47770eb9100c2d0c44946d9cf07ec65d  41ce2a54c0b03bf3443c3d931a367089

3  949d5b44dbf5de918fe9c16f97b45f8a  f88197465ea7920adcdbec7375364d82

4  ad21c59c0840e6cb83a9ceb5573f8159  8ab97904e6daea8866dbdbc4fb7aad2c


  order_status order_purchase_timestamp    order_approved_at  \
0    delivered      2017-10-02 10:56:33  2017-10-02 11:07:15
1    delivered      2018-07-24 20:41:37  2018-07-26 03:24:27
2    delivered      2018-08-08 08:38:49  2018-08-08 08:55:23
3    delivered      2017-11-18 19:28:06  2017-11-18 19:45:59
4    delivered      2018-02-13 21:18:39  2018-02-13 22:20:29

  order_delivered_carrier_date order_delivered_customer_date  \
0          2017-10-04 19:55:00           2017-10-10 21:25:13
1          2018-07-26 14:31:00           2018-08-07 15:27:45
2          2018-08-08 13:50:00           2018-08-17 18:06:29
3          2017-11-22 13:39:59           2017-12-02 00:28:42
4          2018-02-14 19:46:34           2018-02-16 18:17:02

  order_estimated_delivery_date                         seller_id  \
0           2017-10-18 00:00:00  3504c0cb71d7fa48d967e0e4c94d59d9
1           2018-08-13 00:00:00  289cdb325fb7e7f891c38608bf9e0962
2           2018-09-04 00:00:00  4869f7a5dfa277a7dca6462dcf3b52b2
3           2017-12-15 00:00:00  66922902710d126a0e7d26b0e3805106
4           2018-02-26 00:00:00  2c9e548be18521d1c43cde1c582c6de8

   seller_zip_code_prefix  customer_zip_code_prefix
0                  9350.0                      3149
1                 31570.0                     47813
2                 14840.0                     75265
3                 31842.0                     59296
4                  8752.0                      9195

# Clean geo df
geo = geo[~geo['geolocation_zip_code_prefix'].duplicated()]
'''
This line first selects the 'geolocation_zip_code_prefix' column from
the 'geo' DataFrame using the indexing operator []. The duplicated()
```

```
method is then called on this column to create a boolean mask of rows
that have duplicate values in this column.

The tilde operator (~) is used to invert this boolean mask, so that
the mask contains True for rows that do not have duplicate values in
this column.
'''

geo.head()
```

```
   geolocation_zip_code_prefix  geolocation_lat  geolocation_lng  \
0                         1037       -23.545621       -46.639292
1                         1046       -23.546081       -46.644820
3                         1041       -23.544392       -46.639499
4                         1035       -23.541578       -46.641607
5                         1012       -23.547762       -46.635361

  geolocation_city geolocation_state
0        sao paulo                SP
1        sao paulo                SP
3        sao paulo                SP
4        sao paulo                SP
5        são paulo                SP
```

```
# add seller coordinates to the orders
orders = orders.merge(geo, left_on="seller_zip_code_prefix",
                      right_on="geolocation_zip_code_prefix",
how="left")

# The code above performs a left join of two dataframes - "orders" and
"geo" - using the "seller_zip_code_prefix" column of the "orders"
dataframe and the "geolocation_zip_code_prefix" column of the "geo"
dataframe as the join keys.

orders.head()
```

```
                           order_id                       customer_id
\
0  e481f51cbdc54678b7cc49136f2d6af7  9ef432eb6251297304e76186b10a928d

1  53cdb2fc8bc7dce0b6741e2150273451  b0830fb4747a6c6d20dea0b8c802d7ef

2  47770eb9100c2d0c44946d9cf07ec65d  41ce2a54c0b03bf3443c3d931a367089

3  949d5b44dbf5de918fe9c16f97b45f8a  f88197465ea7920adcdbec7375364d82

4  ad21c59c0840e6cb83a9ceb5573f8159  8ab97904e6daea8866dbdbc4fb7aad2c


  order_status order_purchase_timestamp    order_approved_at  \
0    delivered      2017-10-02 10:56:33  2017-10-02 11:07:15
```

```
1    delivered      2018-07-24 20:41:37  2018-07-26 03:24:27
2    delivered      2018-08-08 08:38:49  2018-08-08 08:55:23
3    delivered      2017-11-18 19:28:06  2017-11-18 19:45:59
4    delivered      2018-02-13 21:18:39  2018-02-13 22:20:29

  order_delivered_carrier_date order_delivered_customer_date  \
0          2017-10-04 19:55:00           2017-10-10 21:25:13
1          2018-07-26 14:31:00           2018-08-07 15:27:45
2          2018-08-08 13:50:00           2018-08-17 18:06:29
3          2017-11-22 13:39:59           2017-12-02 00:28:42
4          2018-02-14 19:46:34           2018-02-16 18:17:02

  order_estimated_delivery_date                         seller_id  \
0           2017-10-18 00:00:00  3504c0cb71d7fa48d967e0e4c94d59d9
1           2018-08-13 00:00:00  289cdb325fb7e7f891c38608bf9e0962
2           2018-09-04 00:00:00  4869f7a5dfa277a7dca6462dcf3b52b2
3           2017-12-15 00:00:00  66922902710d126a0e7d26b0e3805106
4           2018-02-26 00:00:00  2c9e548be18521d1c43cde1c582c6de8

   seller_zip_code_prefix  customer_zip_code_prefix  \
0                  9350.0                      3149
1                 31570.0                     47813
2                 14840.0                     75265
3                 31842.0                     59296
4                  8752.0                      9195

   geolocation_zip_code_prefix  geolocation_lat  geolocation_lng  \
0                       9350.0       -23.680114       -46.452454
1                      31570.0       -19.810119       -43.984727
2                      14840.0       -21.362358       -48.232976
3                      31842.0       -19.840168       -43.923299
4                       8752.0       -23.551707       -46.260979

  geolocation_city geolocation_state
0             maua                SP
1   belo horizonte                MG
2          guariba                SP
3   belo horizonte                MG
4  mogi das cruzes                SP

# add customer coordinates to the orders
orders = orders.merge(geo, left_on="customer_zip_code_prefix",
                      right_on="geolocation_zip_code_prefix",
how="left",
                      suffixes=("_seller", "_customer"))
# This code merges two Pandas DataFrames, orders and geo, on a common
column, customer_zip_code_prefix in orders and
geolocation_zip_code_prefix in geo. The resulting merged DataFrame
contains all the columns from both DataFrames.
```

```
# Clean orders
# 1-Filter out orders with multiple sellers Because each order only
has one delivery date
df = orders.groupby(by="order_id").nunique()
# Groups the orders by order_id and calculates the number of unique
values in each group using the nunique() method.

mono_orders = pd.Series(df[df['seller_id'] == 1].index)
# Selects the indices of the resulting DataFrame where the seller_id
column equals 1 and stores them in a Pandas Series called mono_orders.

filtered_orders = orders.merge(mono_orders, how='inner')
# Merges the original orders DataFrame with mono_orders based on the
order_id column using an inner join and stores the resulting DataFrame
in a variable called filtered_orders.

# 2-drop rows with missing values
filtered_orders = filtered_orders.drop(columns=["order_approved_at"])
# This line drops the "order_approved_at" column from the
filtered_orders DataFrame. This column is not necessary for the
analysis being performed.

filtered_orders = filtered_orders.dropna()
# This line drops any rows in the filtered_orders DataFrame that
contain missing values. This is a common data preprocessing step to
ensure that the dataset is clean and complete.


filtered_orders.head()

                           order_id                       customer_id
\
0  e481f51cbdc54678b7cc49136f2d6af7  9ef432eb6251297304e76186b10a928d

1  53cdb2fc8bc7dce0b6741e2150273451  b0830fb4747a6c6d20dea0b8c802d7ef

2  47770eb9100c2d0c44946d9cf07ec65d  41ce2a54c0b03bf3443c3d931a367089

3  949d5b44dbf5de918fe9c16f97b45f8a  f88197465ea7920adcdbec7375364d82

4  ad21c59c0840e6cb83a9ceb5573f8159  8ab97904e6daea8866dbdbc4fb7aad2c


  order_status order_purchase_timestamp
order_delivered_carrier_date  \
0    delivered      2017-10-02 10:56:33          2017-10-04 19:55:00

1    delivered      2018-07-24 20:41:37          2018-07-26 14:31:00

2    delivered      2018-08-08 08:38:49          2018-08-08 13:50:00
```

```
3    delivered      2017-11-18 19:28:06          2017-11-22 13:39:59

4    delivered      2018-02-13 21:18:39          2018-02-14 19:46:34


  order_delivered_customer_date order_estimated_delivery_date  \
0           2017-10-10 21:25:13           2017-10-18 00:00:00
1           2018-08-07 15:27:45           2018-08-13 00:00:00
2           2018-08-17 18:06:29           2018-09-04 00:00:00
3           2017-12-02 00:28:42           2017-12-15 00:00:00
4           2018-02-16 18:17:02           2018-02-26 00:00:00

                          seller_id  seller_zip_code_prefix  \
0  3504c0cb71d7fa48d967e0e4c94d59d9                  9350.0
1  289cdb325fb7e7f891c38608bf9e0962                 31570.0
2  4869f7a5dfa277a7dca6462dcf3b52b2                 14840.0
3  66922902710d126a0e7d26b0e3805106                 31842.0
4  2c9e548be18521d1c43cde1c582c6de8                  8752.0

   customer_zip_code_prefix  geolocation_zip_code_prefix_seller  \
0                      3149                              9350.0
1                     47813                             31570.0
2                     75265                             14840.0
3                     59296                             31842.0
4                      9195                              8752.0

   geolocation_lat_seller  geolocation_lng_seller
geolocation_city_seller  \
0              -23.680114              -46.452454
maua
1              -19.810119              -43.984727          belo
horizonte
2              -21.362358              -48.232976
guariba
3              -19.840168              -43.923299          belo
horizonte
4              -23.551707              -46.260979         mogi das
cruzes

  geolocation_state_seller  geolocation_zip_code_prefix_customer  \
0                       SP                                3149.0
1                       MG                               47813.0
2                       SP                               75265.0
3                       MG                               59296.0
4                       SP                                9195.0

   geolocation_lat_customer  geolocation_lng_customer  \
0                -23.574809                -46.587471
1                -12.169860                -44.988369
2                -16.746337                -48.514624
```

```
3                  -5.767733                -35.275467
4                 -23.675037                -46.524784

  geolocation_city_customer geolocation_state_customer
0                 sao paulo                         SP
1                 barreiras                         BA
2                vianopolis                         GO
3   sao goncalo do amarante                         RN
4               santo andre                         SP

# Define Function to calculate distance
def haversine_distance(lon1, lat1, lon2, lat2):
    """
    Compute distance between two pairs of (lat, lng)
    """
    lon1, lat1, lon2, lat2 = map(radians, [lon1, lat1, lon2, lat2])
    dlon = lon2 - lon1
    dlat = lat2 - lat1
    a = sin(dlat / 2) ** 2 + cos(lat1) * cos(lat2) * sin(dlon / 2) **
2
    return 2 * 6371 * asin(sqrt(a))

# This line defines a function called haversine_distance that takes
four arguments: lon1, lat1, lon2, and lat2. These arguments represent
the longitude and latitude coordinates of two points on the globe.

def get_package_size(items, products):
    # Get Package Size
    df_tmp = items[['order_id',
'product_id']].merge(products[['product_id', 'product_length_cm',
'product_height_cm',

'product_width_cm', 'product_weight_g']],
                                                     on="product_id",
                                                     how="outer")
    df_tmp.loc[:, "product_size_cm3"] = \
        df_tmp['product_length_cm']*df_tmp['product_width_cm'] *
df_tmp['product_height_cm']
    orders_size_weight = df_tmp.groupby("order_id",
as_index=False).sum()[['order_id', 'product_size_cm3',

'product_weight_g']]
    return orders_size_weight

# The code above defines a function called get_package_size that takes
two arguments: items and products, which are Pandas DataFrames
containing information about products and the items ordered.

def object_to_int(dataframe_series):
```

```
    if dataframe_series.dtype == 'object':
        # This line checks if the data type of the input
dataframe_series is "object", which typically represents string or
categorical data.

        dataframe_series =
LabelEncoder().fit_transform(dataframe_series)
        # This line uses the LabelEncoder() method from the scikit-
learn library to encode the string or categorical data as integers.
This is a common preprocessing step in machine learning to convert
non-numeric data into a format that can be used by algorithms. The
fit_transform() method fits the encoder to the data and transforms it.

    return dataframe_series

#It gives the numerical statistical information of the dataframe
orders.describe()

'''
The code above applies the describe() method to a Pandas DataFrame
called orders, which provides summary statistics of the data in the
DataFrame. Here's a brief explanation of what each statistic means:

count: the number of non-missing values in each column
mean: the average value of each column
std: the standard deviation of each column
min: the minimum value of each column
25%: the first quartile of each column (25th percentile)
50%: the second quartile of each column (50th percentile, equivalent
to the median)
75%: the third quartile of each column (75th percentile)
max: the maximum value of each column
The describe() method is useful for quickly getting an overview of the
data in a DataFrame, including identifying potential outliers or
unusual patterns in the data.
'''

"\nThe code above applies the describe() method to a Pandas DataFrame
called orders, which provides summary statistics of the data in the
DataFrame. Here's a brief explanation of what each statistic means:\n\
ncount: the number of non-missing values in each column\nmean: the
average value of each column\nstd: the standard deviation of each
column\nmin: the minimum value of each column\n25%: the first quartile
of each column (25th percentile)\n50%: the second quartile of each
column (50th percentile, equivalent to the median)\n75%: the third
quartile of each column (75th percentile)\nmax: the maximum value of
each column\nThe describe() method is useful for quickly getting an
overview of the data in a DataFrame, including identifying potential
outliers or unusual patterns in the data.\n"
```

Find the total number of missing values feature wise

```
# It returns the number of null values in the dataframe for every
column feature
# Using info() we can view the number of non-null values whereas
isnull() gives the number of null values
orders.isnull().sum()
```

```
order_id                                  0
customer_id                               0
order_status                              0
order_purchase_timestamp                  0
order_approved_at                       161
order_delivered_carrier_date           1968
order_delivered_customer_date          3229
order_estimated_delivery_date             0
seller_id                               775
seller_zip_code_prefix                  775
customer_zip_code_prefix                  0
geolocation_zip_code_prefix_seller     1028
geolocation_lat_seller                 1028
geolocation_lng_seller                 1028
geolocation_city_seller                1028
geolocation_state_seller               1028
geolocation_zip_code_prefix_customer    306
geolocation_lat_customer                306
geolocation_lng_customer                306
geolocation_city_customer               306
geolocation_state_customer              306
dtype: int64
```

Scaling the data

```
def scaling(X_train, X_test):
    '''
    The code above defines a function called scaling that performs
feature scaling using the StandardScaler class from scikit-learn.
    '''
    sc_X = StandardScaler()
    X_train_scaled = sc_X.fit_transform(X_train)
    X_test_scaled = sc_X.fit_transform(X_test)
    return X_train_scaled, X_test_scaled
```

Splitting dataset into separate training and test set

```
def split_data(final_df, columns_for_train, columns_for_pred, i_flag):
    '''
    This function is a convenient way to split a DataFrame into
training and testing sets for machine learning purposes. The i_flag
parameter is used to avoid a warning message related to the NumPy
```

```
library, and the columns_for_train and columns_for_pred parameters
allow for flexible selection of features and target variables.
    '''

    if i_flag:
        from sklearnex import patch_sklearn  # pylint: disable=C0415,
E0401
        patch_sklearn()
    from sklearn.model_selection import train_test_split  # pylint:
disable=C0415

    x_train, x_test, y_train, y_test =
train_test_split(final_df[columns_for_train],
final_df[columns_for_pred],
                                                        test_size=0.3,
random_state=42)
    return x_train, x_test, y_train, y_test

filtered_orders['distance'] = filtered_orders.apply(lambda row:
haversine_distance(row["geolocation_lng_seller"],

row["geolocation_lat_seller"],

row["geolocation_lng_customer"],

row["geolocation_lat_customer"],),
                                                    axis=1,)
# This line creates a new column in the filtered_orders DataFrame
called "distance" by applying the haversine_distance function to
calculate the distance between the seller and the customer for each
row in the DataFrame.


orders_size_weight = get_package_size(items, products)
filtered_orders = filtered_orders.merge(orders_size_weight,
on='order_id', how='left')
# These lines merge information about the package size and weight for
each order from the items and products DataFrames into the
filtered_orders DataFrame.


# process time columns
time_columns = ['order_purchase_timestamp',
'order_delivered_customer_date', 'order_estimated_delivery_date']
for column in time_columns:
    filtered_orders.loc[:, column] =
pd.to_datetime(filtered_orders[column])
# These lines convert several time-related columns in the
filtered_orders DataFrame from string format to datetime format using
the pd.to_datetime function.
```

```
filtered_orders.loc[:, "wait_time"] =
(filtered_orders['order_delivered_customer_date'] -

filtered_orders['order_purchase_timestamp']).dt.days
filtered_orders.loc[:, "est_wait_time"] =
(filtered_orders['order_estimated_delivery_date'] -

filtered_orders['order_purchase_timestamp']).dt.days
# These lines calculate the wait time and estimated wait time for each
order in days, based on the difference between the purchase time and
the delivered/estimated delivery time.


filtered_orders.loc[:, "purchase_dow"] =
filtered_orders.order_purchase_timestamp.dt.dayofweek
filtered_orders.loc[:, "year"] =
filtered_orders.order_purchase_timestamp.dt.year
filtered_orders.loc[:, "purchase_month"] =
filtered_orders.order_purchase_timestamp.dt.month
# These lines extract the day of the week, year, and month from the
order_purchase_timestamp column in the filtered_orders DataFrame.


final_df = filtered_orders[['purchase_dow', 'purchase_month', 'year',
'product_size_cm3', 'product_weight_g',
                            'geolocation_state_customer',
'geolocation_state_seller', 'distance',
                            'wait_time', 'est_wait_time']]
# This line creates a new DataFrame called final_df by selecting
certain columns from the filtered_orders DataFrame.

final_df['delay'] = final_df['wait_time'] - final_df['est_wait_time']
final_df['delay'] = final_df['delay'] > 0
final_df['delay'] = final_df['delay'].astype(int)
# These lines calculate the delay time for each order and encode it as
a binary variable indicating whether there was a delay or not.

final_df_enc = final_df.apply(lambda x: object_to_int(x))
# This line applies the object_to_int function to each column in the
final_df DataFrame to convert categorical variables into integer
format. The object_to_int function is not shown in the code provided,
but it likely uses the LabelEncoder class from the scikit-learn
library to perform the conversion.

final_df_enc.head()
```

```
----------------------------------------------------------------------
-----
MergeError                                Traceback (most recent call
```

```
last)
Cell In[29], line 10
      6 # This line creates a new column in the filtered_orders
DataFrame called "distance" by applying the haversine_distance
function to calculate the distance between the seller and the customer
for each row in the DataFrame.
      9 orders_size_weight = get_package_size(items, products)
---> 10 filtered_orders = filtered_orders.merge(orders_size_weight,
on='order_id', how='left')
     11 # These lines merge information about the package size and
weight for each order from the items and products DataFrames into the
filtered_orders DataFrame.
     12
     13
     14 # process time columns
     15 time_columns = ['order_purchase_timestamp',
'order_delivered_customer_date', 'order_estimated_delivery_date']

File ~/.local/lib/python3.12/site-packages/pandas/core/frame.py:10832,
in DataFrame.merge(self, right, how, on, left_on, right_on,
left_index, right_index, sort, suffixes, copy, indicator, validate)
  10813 @Substitution("")
  10814 @Appender(_merge_doc, indents=2)
  10815 def merge(
   (...)
  10828     validate: MergeValidate | None = None,
  10829 ) -> DataFrame:
  10830     from pandas.core.reshape.merge import merge
> 10832     return merge(
  10833         self,
  10834         right,
  10835         how=how,
  10836         on=on,
  10837         left_on=left_on,
  10838         right_on=right_on,
  10839         left_index=left_index,
  10840         right_index=right_index,
  10841         sort=sort,
  10842         suffixes=suffixes,
  10843         copy=copy,
  10844         indicator=indicator,
  10845         validate=validate,
  10846     )

File
~/.local/lib/python3.12/site-packages/pandas/core/reshape/merge.py:184
, in merge(left, right, how, on, left_on, right_on, left_index,
right_index, sort, suffixes, copy, indicator, validate)
    169 else:
```

```
    170     op = _MergeOperation(
    171         left_df,
    172         right_df,
   (...)
    182         validate=validate,
    183     )
--> 184     return op.get_result(copy=copy)

File
~/.local/lib/python3.12/site-packages/pandas/core/reshape/merge.py:888
, in _MergeOperation.get_result(self, copy)
    884     self.left, self.right =
self._indicator_pre_merge(self.left, self.right)
    886 join_index, left_indexer, right_indexer =
self._get_join_info()
--> 888 result = self._reindex_and_concat(
    889     join_index, left_indexer, right_indexer, copy=copy
    890 )
    891 result = result.__finalize__(self, method=self._merge_type)
    893 if self.indicator:

File
~/.local/lib/python3.12/site-packages/pandas/core/reshape/merge.py:840
, in _MergeOperation._reindex_and_concat(self, join_index,
left_indexer, right_indexer, copy)
    837 left = self.left[:]
    838 right = self.right[:]
--> 840 llabels, rlabels = _items_overlap_with_suffix(
    841     self.left._info_axis, self.right._info_axis, self.suffixes
    842 )
    844 if left_indexer is not None and not
is_range_indexer(left_indexer, len(left)):
    845     # Pinning the index here (and in the right code just
below) is not
    846     #  necessary, but makes the `.take` more performant if we
have e.g.
    847     #  a MultiIndex for left.index.
    848     lmgr = left._mgr.reindex_indexer(
    849         join_index,
    850         left_indexer,
   (...)
    855         use_na_proxy=True,
    856     )

File
~/.local/lib/python3.12/site-packages/pandas/core/reshape/merge.py:275
7, in _items_overlap_with_suffix(left, right, suffixes)
   2755     dups.extend(rlabels[(rlabels.duplicated()) &
(~right.duplicated())].tolist())
   2756 if dups:
```

```
-> 2757     raise MergeError(
   2758         f"Passing 'suffixes' which cause duplicate columns
{set(dups)} is "
   2759         f"not allowed.",
   2760     )
   2762 return llabels, rlabels

MergeError: Passing 'suffixes' which cause duplicate columns
{'product_weight_g_x', 'product_size_cm3_x'} is not allowed.
```

```python
# Export the final_df_enc DataFrame to a CSV file
final_df_enc.to_csv('final_OTD_time_forecasting_dataframe.csv',
index=False)
```

```
----------------------------------------------------------------------
-----
NameError                                 Traceback (most recent call
last)
Cell In[31], line 2
      1 # Export the final_df_enc DataFrame to a CSV file
----> 2
final_df_enc.to_csv('final_OTD_time_forecasting_dataframe.csv',
index=False)

NameError: name 'final_df_enc' is not defined
```

```python
# EDA: Check for missing values in the dataset
print("Missing values in orders dataset:")
print(orders.isnull().sum())
print("\nMissing values in items dataset:")
print(items.isnull().sum())
print("\nMissing values in customers dataset:")
print(customers.isnull().sum())
print("\nMissing values in sellers dataset:")
print(sellers.isnull().sum())
print("\nMissing values in geo dataset:")
print(geo.isnull().sum())
print("\nMissing values in products dataset:")
print(products.isnull().sum())
```

```
Missing values in orders dataset:
order_id                                   0
customer_id                                0
order_status                               0
order_purchase_timestamp                   0
order_approved_at                        161
order_delivered_carrier_date            1968
order_delivered_customer_date           3229
order_estimated_delivery_date              0
seller_id                                775
```

```
seller_zip_code_prefix                      775
customer_zip_code_prefix                      0
geolocation_zip_code_prefix_seller         1028
geolocation_lat_seller                     1028
geolocation_lng_seller                     1028
geolocation_city_seller                    1028
geolocation_state_seller                   1028
geolocation_zip_code_prefix_customer        306
geolocation_lat_customer                    306
geolocation_lng_customer                    306
geolocation_city_customer                   306
geolocation_state_customer                  306
dtype: int64

Missing values in items dataset:
order_id               0
order_item_id          0
product_id             0
seller_id              0
shipping_limit_date    0
price                  0
freight_value          0
dtype: int64

Missing values in customers dataset:
customer_id                 0
customer_unique_id          0
customer_zip_code_prefix    0
customer_city               0
customer_state              0
dtype: int64

Missing values in sellers dataset:
seller_id                 0
seller_zip_code_prefix    0
seller_city               0
seller_state              0
dtype: int64

Missing values in geo dataset:
geolocation_zip_code_prefix    0
geolocation_lat                0
geolocation_lng                0
geolocation_city               0
geolocation_state              0
dtype: int64

Missing values in products dataset:
product_id                       0
product_category_name          610
```

```
product_name_lenght           610
product_description_lenght    610
product_photos_qty            610
product_weight_g                2
product_length_cm               2
product_height_cm               2
product_width_cm                2
dtype: int64
```

```python
# EDA: Check for duplicates in the dataset
print("Duplicates in orders dataset:", orders.duplicated().sum())
print("Duplicates in items dataset:", items.duplicated().sum())
print("Duplicates in customers dataset:",
customers.duplicated().sum())
print("Duplicates in sellers dataset:", sellers.duplicated().sum())
print("Duplicates in geo dataset:", geo.duplicated().sum())
print("Duplicates in products dataset:", products.duplicated().sum())
```

```
Duplicates in orders dataset: 12640
Duplicates in items dataset: 0
Duplicates in customers dataset: 0
Duplicates in sellers dataset: 0
Duplicates in geo dataset: 0
Duplicates in products dataset: 0
```

```python
# EDA: Summary statistics for numerical columns
print("Summary statistics for orders dataset:")
print(orders.describe())
print("\nSummary statistics for items dataset:")
print(items.describe())
print("\nSummary statistics for customers dataset:")
print(customers.describe())
print("\nSummary statistics for sellers dataset:")
print(sellers.describe())
print("\nSummary statistics for geo dataset:")
print(geo.describe())
print("\nSummary statistics for products dataset:")
print(products.describe())
```

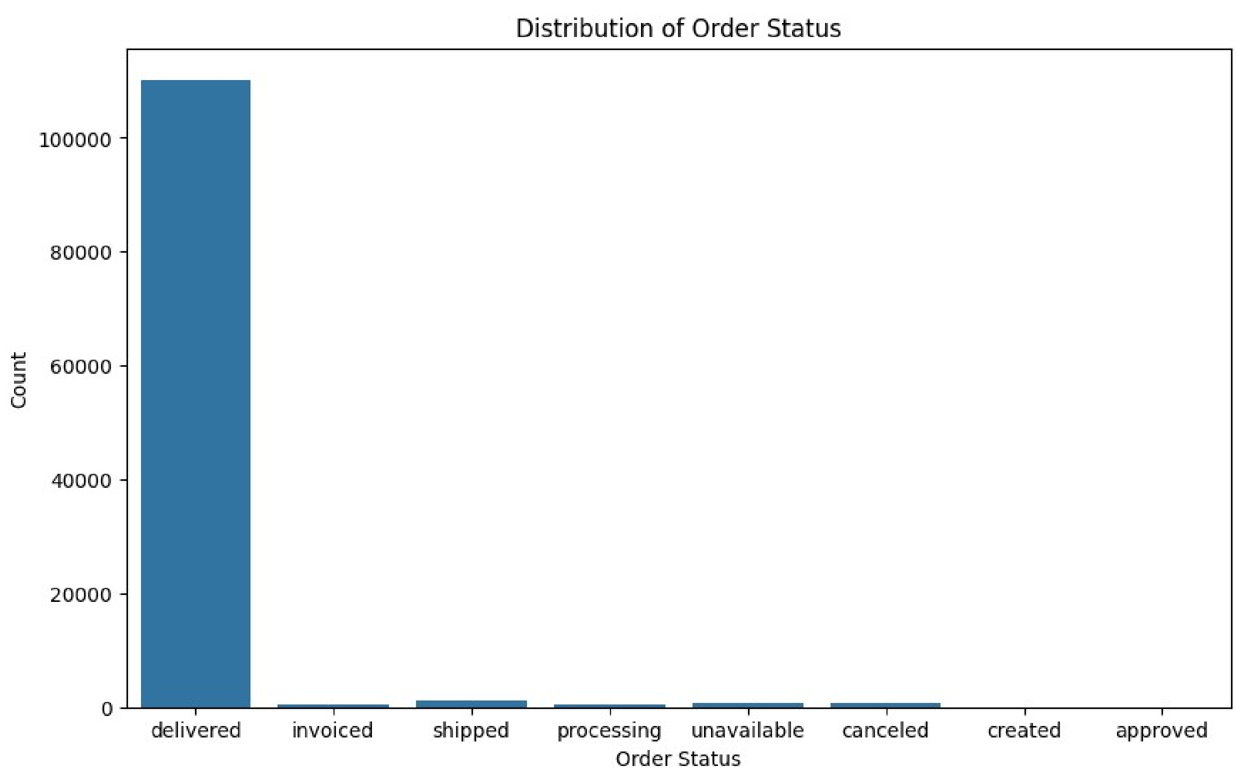
```
Summary statistics for orders dataset:
       seller_zip_code_prefix  customer_zip_code_prefix  \
count           112650.000000             113425.000000
mean             24439.170431              35102.472965
std              27596.030909              29864.919733
min               1001.000000               1003.000000
25%               6429.000000              11250.000000
50%              13568.000000              24320.000000
75%              27930.000000              59020.000000
max              99730.000000              99990.000000
```

```
       geolocation_zip_code_prefix_seller  geolocation_lat_seller  \
count                       112397.000000           112397.000000
mean                         24435.840191              -22.800558
std                          27593.085486                2.697063
min                           1001.000000              -36.605374
25%                           6429.000000              -23.610305
50%                          13568.000000              -23.422313
75%                          27345.000000              -21.766477
max                          99730.000000               -2.546079

       geolocation_lng_seller  geolocation_zip_code_prefix_customer  \
count           112397.000000                         113119.000000
mean               -47.235919                          35025.376285
std                  2.341211                          29852.263889
min                -67.809656                           1003.000000
25%                -48.831547                          11088.500000
50%                -46.747050                          24240.000000
75%                -46.518082                          58418.000000
max                -34.847856                          99990.000000

       geolocation_lat_customer  geolocation_lng_customer
count             113119.000000             113119.000000
mean                 -21.237918                -46.204648
std                    5.577710                  4.045243
min                  -36.605374                -72.666706
25%                  -23.590818                -48.110471
50%                  -22.931096                -46.633493
75%                  -20.193636                -43.642427
max                   42.184003                 -8.577855

Summary statistics for items dataset:
       order_item_id          price  freight_value
count  112650.000000  112650.000000  112650.000000
mean        1.197834     120.653739      19.990320
std         0.705124     183.633928      15.806405
min         1.000000       0.850000       0.000000
25%         1.000000      39.900000      13.080000
50%         1.000000      74.990000      16.260000
75%         1.000000     134.900000      21.150000
max        21.000000    6735.000000     409.680000

Summary statistics for customers dataset:
       customer_zip_code_prefix
count              99441.000000
mean               35137.474583
std                29797.938996
min                 1003.000000
25%                11347.000000
50%                24416.000000
75%                58900.000000
```

```
max                99990.000000

Summary statistics for sellers dataset:
       seller_zip_code_prefix
count             3095.000000
mean             32291.059451
std              32713.453830
min               1001.000000
25%               7093.500000
50%              14940.000000
75%              64552.500000
max              99730.000000

Summary statistics for geo dataset:
       geolocation_zip_code_prefix  geolocation_lat  geolocation_lng
count                 19015.000000     19015.000000     19015.000000
mean                  42711.591901       -19.062087       -46.058008
std                   30905.051745         7.319402         5.380751
min                    1001.000000       -36.605374       -72.927296
25%                   12721.500000       -23.564386       -49.000445
50%                   38240.000000       -22.429252       -46.632544
75%                   70656.500000       -15.615448       -43.255324
max                   99990.000000        42.184003       121.105394

Summary statistics for products dataset:
       product_name_lenght  product_description_lenght
product_photos_qty  \
count         32341.000000                32341.000000
32341.000000
mean             48.476949                  771.495285
2.188986
std              10.245741                  635.115225
1.736766
min               5.000000                    4.000000
1.000000
25%              42.000000                  339.000000
1.000000
50%              51.000000                  595.000000
1.000000
75%              57.000000                  972.000000
3.000000
max              76.000000                 3992.000000
20.000000

       product_weight_g  product_length_cm  product_height_cm  \
count      32949.000000       32949.000000       32949.000000
mean        2276.472488          30.815078          16.937661
std         4282.038731          16.914458          13.637554
min            0.000000           7.000000           2.000000
25%          300.000000          18.000000           8.000000
```

```
50%        700.000000        25.000000        13.000000
75%       1900.000000        38.000000        21.000000
max      40425.000000       105.000000       105.000000

       product_width_cm
count      32949.000000
mean          23.196728
std           12.079047
min            6.000000
25%           15.000000
50%           20.000000
75%           30.000000
max          118.000000
```

```
# EDA: Visualize the distribution of order status in the orders
dataset
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(10, 6))
sns.countplot(data=orders, x='order_status')
plt.title('Distribution of Order Status')
plt.xlabel('Order Status')
plt.ylabel('Count')
plt.show()
```



Data Cleaning and Preprocessing

```
# Data Cleaning: Handle missing values
# Drop rows with missing delivery dates in the orders dataset
orders = orders.dropna(subset=['order_delivered_customer_date'])

# Fill missing values in the products dataset with the mode for
categorical columns
products['product_category_name'] =
products['product_category_name'].fillna(products['product_category_na
me'].mode()[0])

# Feature Engineering: Transform categorical variables into numerical
variables
# Encode 'order_status' in the orders dataset
label_encoder = LabelEncoder()
orders['order_status'] =
label_encoder.fit_transform(orders['order_status'])

# Encode 'product_category_name' in the products dataset
products['product_category_name'] =
label_encoder.fit_transform(products['product_category_name'])

# Added: Feature Engineering
# Calculate delivery time in days
orders['order_purchase_timestamp'] =
pd.to_datetime(orders['order_purchase_timestamp'])
orders['order_delivered_customer_date'] =
pd.to_datetime(orders['order_delivered_customer_date'])
orders['delivery_time_days'] =
(orders['order_delivered_customer_date'] -
orders['order_purchase_timestamp']).dt.days

# Drop rows with negative or zero delivery time (invalid data)
orders = orders[orders['delivery_time_days'] > 0]

# Display the new feature
print("\nDelivery Time in Days:")
print(orders['delivery_time_days'].describe())
```

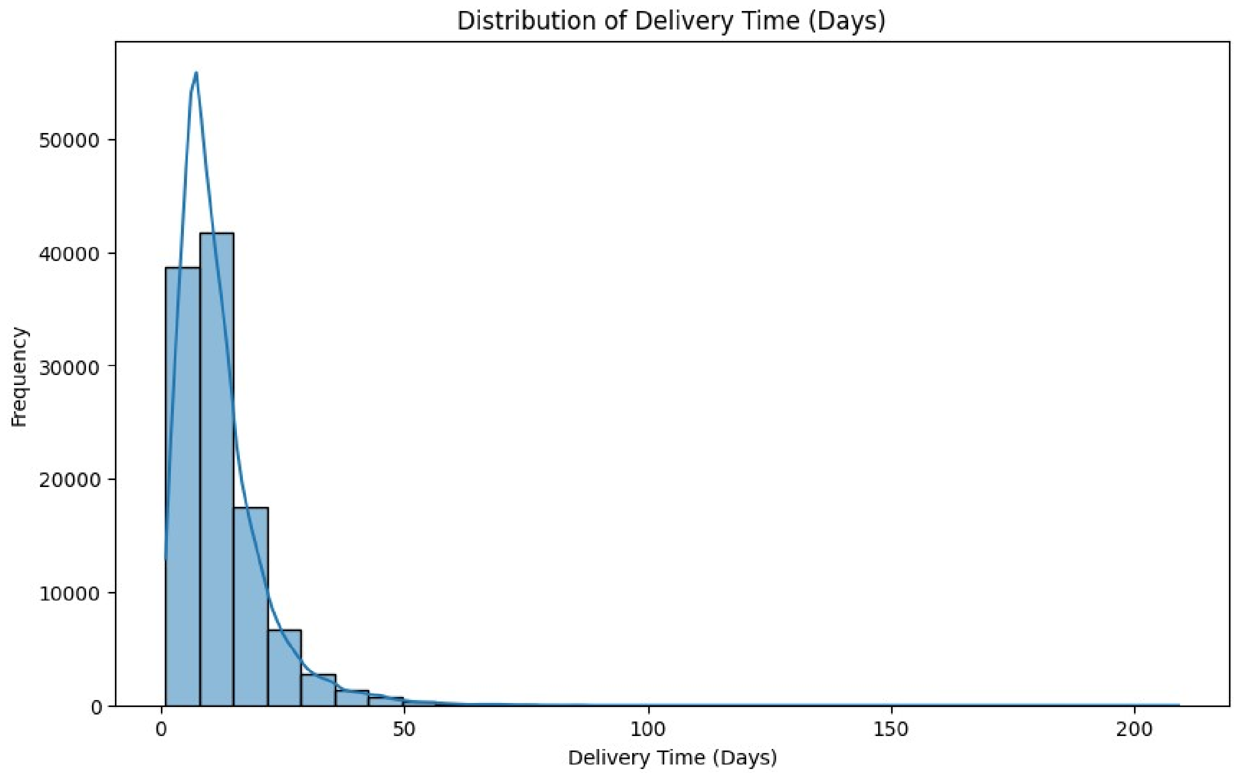
```
Delivery Time in Days:
count    110178.000000
mean         12.009684
std           9.450981
min           1.000000
25%           6.000000
50%          10.000000
75%          15.000000
max         209.000000
Name: delivery_time_days, dtype: float64
```

```python
# Added: Split the data into training and test sets
# Define features (X) and target (y)
X = orders[['order_status', 'delivery_time_days']]  # Example features
y = orders['delivery_time_days']  # Target variable

# Split the data into training (80%) and test (20%) sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Display the shapes of the resulting datasets
print("\nTraining Set Shape:", X_train.shape, y_train.shape)
print("Test Set Shape:", X_test.shape, y_test.shape)
```

```
Training Set Shape: (88142, 2) (88142,)
Test Set Shape: (22036, 2) (22036,)
```

Data Visualization

```python
# Added: Visualize the distribution of delivery time
plt.figure(figsize=(10, 6))
sns.histplot(orders['delivery_time_days'], bins=30, kde=True)
plt.title("Distribution of Delivery Time (Days)")
plt.xlabel("Delivery Time (Days)")
plt.ylabel("Frequency")
plt.show()

# Visualize the relationship between order status and delivery time
plt.figure(figsize=(10, 6))
sns.boxplot(data=orders, x='order_status', y='delivery_time_days')
plt.title("Delivery Time by Order Status")
plt.xlabel("Order Status")
plt.ylabel("Delivery Time (Days)")
plt.show()
```


Distribution of Delivery Time (Days)
Frequency
50000
40000
30000
20000
10000
0
0
50
100
150
200
Delivery Time (Days)


Delivery Time by Order Status
Delivery Time (Days)
200
150
100
50
0
0
1
Order Status