Tom Genlis - EPITA 2024 - SCIA - CV

Recommender System based on Neural Networks

Customer Order Prediction with TensorFlow

This notebook is focused on predicting customer orders using various pieces of information about the customer, the vendor, and the orders that have been made. It uses TensorFlow, to train a deep learning model for this task.

The notebook uses different data preprocessing techniques to format and clean the data, which is then used to train a neural network model. The neural network model consists of several densely connected layers and uses the Adam optimizer and binary cross-entropy as the loss function.

Libraries

The script uses several Python libraries, including:

- `pandas` : used for data manipulation and analysis.
- `numpy` : used for numerical computations.
- `tensorflow` : used to build and train the machine learning model.
- `sklearn` : used for data preprocessing and splitting the dataset into training and validation sets.
- `matplotlib` : used for data visualization. tensorflow_addons: provides additional functionality to TensorFlow.
- `sklearn.metrics` : used for computing the f1 score.

In [ ]:

```python
# Import necessary libraries for the task
import pandas as pd
from tensorflow.keras import layers, losses, optimizers
import tensorflow_addons as tfa
import numpy as np
from sklearn.preprocessing import MultiLabelBinarizer
from sklearn.model_selection import train_test_split
import tensorflow as tf
import matplotlib.pyplot as plt
from sklearn.metrics import f1_score
import itertools
```

Datasets

Several datasets are used in this script:

- `test_customers.csv` : Contains information about the test customers.
- `test_locations.csv` : Contains information about the test locations.
- `train_customers.csv` : Contains information about the train customers.
- `train_locations.csv` : Contains information about the train locations.
- `vendors.csv` : Contains information about the vendors.
- `orders.csv` : Contains information about the orders.

In [ ]:

```python
# Read the data from CSV files into pandas dataframes
df_test_customers = pd.read_csv('../data/test_customers.csv')
```

```
df_test_locations = pd.read_csv('../data/test_locations.csv')
df_train_customers = pd.read_csv('../data/train_customers.csv')
df_train_locations = pd.read_csv('../data/train_locations.csv')
df_vendors = pd.read_csv('../data/vendors.csv')
df_orders = pd.read_csv('../data/orders.csv')
```

```
C:\Users\Tom\AppData\Local\Temp\ipykernel_14500\2303872769.py:7: DtypeWarning: Columns (15,1
6,18,19,20) have mixed types. Specify dtype option on import or set low_memory=False.
  df_orders = pd.read_csv('../data/orders.csv')
```

Data Preprocessing

The data is preprocessed through several steps to clean and format the data. This includes dropping unnecessary columns, filling missing values, transforming categorical variables into numerical variables, merging dataframes, and normalizing the data.

Many data are irrelevant, as described by the `describe()` function in pandas which gives us the following output:

In [ ]:

```
print(len(df_train_customers.index))

df_train_customers.describe(include='all')
```

34674

Out[ ]:

	akeed_customer_id	gender	dob	status	verified	language	created_at	updated_a
count	34674	22520	3046.000000	34674.000000	34674.000000	21099	34674	3467
unique	34523	10	NaN	NaN	NaN	1	33650	2940
top	0FOCFVI	Male	NaN	NaN	NaN	EN	2019-10-14 12:20:33	2019-10-0 18:50:3
freq	17	17815	NaN	NaN	NaN	21099	7	1
mean	NaN	NaN	1991.210768	0.998991	0.956538	NaN	NaN	Na
std	NaN	NaN	48.422045	0.031756	0.203898	NaN	NaN	Na
min	NaN	NaN	1.000000	0.000000	0.000000	NaN	NaN	Na
25%	NaN	NaN	1986.000000	1.000000	1.000000	NaN	NaN	Na
50%	NaN	NaN	1993.000000	1.000000	1.000000	NaN	NaN	Na
75%	NaN	NaN	1999.000000	1.000000	1.000000	NaN	NaN	Na
max	NaN	NaN	2562.000000	1.000000	1.000000	NaN	NaN	Na

`language` , `dob` or other sparse columns are real values for less than 10% of the rows, this is why we have to drop them.

Besides some data are float or int 64 types, which is too expensive compared to its real range of values, so we convert them to int8, int16 or bool to save memory.

We keep reiterating this process until we have a clean dataset that can be used to train the model.

We also need to merge the locations of customers and the customers dataframe.

Training Strategy

The training data strategy will be crossing the customers, locations per customer and vendors to get every single possibility of orders. Then the target value will be 1 if the order exists in the order dataframe, otherwise 0.

In [ ]:
```
# Functions to clean and format the data

def format_customers(df_customers, df_locations):
    """
    Function to format the customers data
    """
    # Drop unnecessary columns
    df_customers = df_customers.drop(["language", "dob", "created_at", "updated_at"], axis=1)

    # Clean and map gender data
    df_customers["gender"] = df_customers["gender"].str.strip()
    df_customers["gender"] = df_customers["gender"].str.upper()
    df_customers["gender"] = df_customers["gender"].map({'MALE': 0, 'FEMALE': 1})
    df_customers["gender"] = df_customers.gender.fillna(2).astype(int)

    # Select only verified and active customers
    df_customers = df_customers[(df_customers.verified == 1) & (df_customers.status == 1)].dr
    df_customers = df_customers.rename(columns={"akeed_customer_id": "customer_id"})

    # Merge with location data
    df_customers = pd.merge(df_customers, df_locations[["customer_id", "location_number", "la
    df_customers = df_customers.fillna(0)

    # Convert data types to save memory
    df_customers["gender"] = df_customers["gender"].astype(np.uint8)
    df_customers["location_number"] = df_customers["location_number"].astype(np.uint8)
    df_customers["latitude"] = df_customers["latitude"].astype(np.float16)
    df_customers["longitude"] = df_customers["longitude"].astype(np.float16)

    return df_customers

def format_items(df_orders):
    """
    Function to format the items data
    """
    # Drop unnecessary columns
    df_orders = df_orders[["customer_id", "vendor_id", "deliverydistance", "LOCATION_NUMBER"]

    return df_orders

def format_vendors(df_vendors):
    """
    Function to format the vendors data
    """

    # Drop unnecessary columns
    to_remove = [
            "vendor_category_en",
            "authentication_id",
            "OpeningTime",
            "OpeningTime2",
            "open_close_flags",
            "vendor_tag_name",
            "created_at",
            "updated_at",
            "commission",
            "saturday_to_time1",
            "saturday_from_time2",
            "saturday_from_time1",
            "saturday_to_time2",
            "thursday_to_time1",
            "thursday_from_time1",
            "thursday_from_time2",
```

```python
            "thursday_to_time2",
            "tuesday_to_time1",
            "tuesday_from_time2",
            "tuesday_from_time1",
            "tuesday_to_time2",
            "monday_to_time1",
            "monday_from_time1",
            "monday_from_time2",
            "monday_to_time2",
            "sunday_to_time1",
            "sunday_from_time1",
            "sunday_from_time2",
            "sunday_to_time2",
            "friday_to_time1",
            "friday_from_time1",
            "friday_from_time2",
            "friday_to_time2",
            "wednesday_to_time1",
            "wednesday_from_time1",
            "wednesday_from_time2",
            "wednesday_to_time2",
            "one_click_vendor",
            "country_id",
            "city_id",
            "display_orders",
            "device_type",
            "is_akeed_delivering",
            "language",
            "rank",
            "is_open",
            "verified"
    ]
    df_vendors = df_vendors.drop(to_remove, axis=1)

    # Parse primary_tags
    df_vendors["primary_tags"] = df_vendors["primary_tags"].fillna("{\"primary_tags\":\"0\"}"

    # Parsing and one hot encoding for vendor_tag
    df_vendors_tags = df_vendors.vendor_tag.str.split(",")
    df_vendors_tags = df_vendors_tags.apply(lambda d: d if isinstance(d, list) else [])

    mlb = MultiLabelBinarizer()
    one_hot_encoded_data = mlb.fit_transform(df_vendors_tags)
    one_hot_encoded_data = pd.DataFrame(one_hot_encoded_data, columns=mlb.classes_, dtype=boo

    # Concat df_train_items and one_hot_encoded_data
    df_vendors = df_vendors.drop(["vendor_tag"], axis=1)
    df_vendors = df_vendors.reset_index(drop=True)
    one_hot_encoded_data = one_hot_encoded_data.reset_index(drop=True)
    df_vendors = pd.concat([df_vendors, one_hot_encoded_data], axis=1)

    # Convert data types to save memory
    df_vendors["latitude"] = df_vendors["latitude"].astype(np.float16)
    df_vendors["longitude"] = df_vendors["longitude"].astype(np.float16)
    df_vendors["vendor_category_id"] = df_vendors["vendor_category_id"].astype(np.uint8)
    df_vendors["delivery_charge"] = df_vendors["delivery_charge"].astype(bool)
    df_vendors["serving_distance"] = df_vendors["serving_distance"].astype(np.uint8)
    df_vendors["prepration_time"] = df_vendors["prepration_time"].astype(np.uint8)
    df_vendors["discount_percentage"] = df_vendors["discount_percentage"].astype(np.uint8)
    df_vendors["status"] = df_vendors["status"].astype(bool)
    df_vendors["vendor_rating"] = df_vendors["vendor_rating"].astype(np.float16)
    df_vendors["primary_tags"] = df_vendors["primary_tags"].astype(np.uint16)
    df_vendors["id"] = df_vendors["id"].astype(np.uint16)

    return df_vendors

def format_train_dataset(df_customers, df_orders, df_vendors, df_locations):
    df_customers = format_customers(df_customers, df_locations)
    df_orders = format_items(df_orders)
```

```
    df_vendors = format_vendors(df_vendors)

    # Remove users that are not in df_items
    df_customers = df_customers[df_customers['customer_id'].isin(df_orders['customer_id'])]

    # Cross product of customers and vendors
    # This gives us all the possible triples (customer_id, location_number, vendor_id)
    C = pd.merge(df_customers, df_vendors, how="cross")

    # Check if triple (customer_id, location_number, vendor_id) is in df_orders and put it in
    triplets = set(df_orders[["customer_id", "LOCATION_NUMBER", "vendor_id"]].itertuples(inde
    C["target"] = C.apply(lambda x: 1 if (x["customer_id"], x["location_number"], x["id"]) in

    # Fill NaNs with 0s
    C = C.fillna(0)

    # balance the number of positive and negative samples according to y_train
    C = C.groupby('target').apply(lambda x: x.sample(C['target'].value_counts().min()).reset_
    y_train = C["target"].reset_index(drop=True)
    C = C.drop(["customer_id", "id", "target"], axis=1)
    C = C.reset_index(drop=True)

    return C, y_train
```

In [ ]:
```
# Load data
df_X_train, df_y_train = format_train_dataset(df_train_customers, df_orders, df_vendors, df_t
```

In [ ]:
```
# Convert to numpy
np_X_train = df_X_train.to_numpy()
np_y_train = df_y_train.to_numpy()
```

Now that our data is clean, we need to convert it to tensors, which are multi-dimensional arrays with a uniform type. We can do this by converting the dataframes to numpy arrays and then to tensors, which are supported by TensorFlow.

In [ ]:
```
def dataframe_to_dataset(np_X_train, np_y_train):
    """
    Convert numpy arrays to tensorflow datasets
    """

    # Split train and validation
    X_train, X_val, y_train, y_val = train_test_split(
        np_X_train, np_y_train, train_size=0.80, shuffle=True, random_state=42
    )
    print(X_train.shape, y_train.shape, X_val.shape, y_val.shape)

    size_train = X_train.shape[0]
    size_val = X_val.shape[0]
    input_shape_y = X_train.shape[-1]

    # Convert to tensors
    X_train = tf.keras.utils.normalize(tf.convert_to_tensor(X_train, dtype=tf.float32), axis=
    X_val = tf.keras.utils.normalize(tf.convert_to_tensor(X_val, dtype=tf.float32), axis=1)
    y_train = tf.convert_to_tensor(y_train, dtype=tf.float32)
    y_val = tf.convert_to_tensor(y_val, dtype=tf.float32)

    # Create tensorflow datasets
    train_dataset = tf.data.Dataset.from_tensor_slices((X_train, y_train))
    del X_train, y_train
    valid_dataset = tf.data.Dataset.from_tensor_slices((X_val, y_val))
    del X_val # , y_val

    batch_size = 2048

    # Batch the datasets
    train_dataset = train_dataset.batch(batch_size).prefetch(2)
```

```
    valid_dataset = valid_dataset.batch(batch_size).prefetch(2)

    return input_shape_y, y_val, train_dataset, valid_dataset
```

In [ ]:
```
# Convert to tensorflow datasets
input_shape_y, y_val, train_dataset, valid_dataset = dataframe_to_dataset(np_X_train, np_y_tr
```

```
(128227, 81) (128227,) (32057, 81) (32057,)
```

Machine Learning Model

The script uses a neural network model with several densely connected layers. The model uses the Adam optimizer and binary cross-entropy as the loss function. The model is trained with a batch size of 2048 for a maximum of 100 epochs.

In [ ]:
```
# Create model
model = tf.keras.Sequential([
    layers.Dense(128, activation='relu', input_shape=(input_shape_y,)),
    layers.Dense(64, activation='relu'),
    layers.Dropout(0.2),
    layers.Dense(32, activation='relu'),
    layers.Dense(1, activation='sigmoid')
])

# Compile model
model.compile(optimizer=optimizers.Adam(learning_rate=0.01),
                loss=losses.BinaryCrossentropy(),
                metrics=[tf.keras.metrics.Recall(), tf.keras.metrics.Precision()])

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 128)               10496


_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 128)               10496

 dense_1 (Dense)             (None, 64)                8256

 dropout (Dropout)           (None, 64)                0

 dense_2 (Dense)             (None, 32)                2080

 dense_3 (Dense)             (None, 1)                 33

=================================================================
Total params: 20,865
Trainable params: 20,865
Non-trainable params: 0
_________________________________________________________________
```

Model Training

The model is trained using the processed data, with the output variable being whether or not the customer will place an order. During training, the learning rate is reduced if the validation loss does not improve after a certain number of epochs (patience), and the training is stopped if the validation loss does not improve after a certain number of epochs.

In [ ]:

```
# Reduce learning rate on plateau
reduce_lr = tf.keras.callbacks.ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=5,

# Early stopping
early_stopping = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=10, restore_be

epochs = 100

# Train model
history = model.fit(train_dataset, epochs=epochs, validation_data=valid_dataset, callbacks=[r
```

```
Epoch 1/100
63/63 [==============================] - 7s 34ms/step - loss: 0.6716 - recall: 0.5876 - preci
sion: 0.5741 - val_loss: 0.6443 - val_recall: 0.5444 - val_precision: 0.6444 - lr: 0.0100
Epoch 2/100
63/63 [==============================] - 2s 29ms/step - loss: 0.6214 - recall: 0.6359 - preci
sion: 0.6532 - val_loss: 0.6067 - val_recall: 0.7705 - val_precision: 0.6401 - lr: 0.0100
Epoch 3/100
63/63 [==============================] - 2s 30ms/step - loss: 0.5862 - recall: 0.7229 - preci
sion: 0.6658 - val_loss: 0.5864 - val_recall: 0.5377 - val_precision: 0.7636 - lr: 0.0100
Epoch 4/100
63/63 [==============================] - 2s 39ms/step - loss: 0.5580 - recall: 0.7694 - preci
sion: 0.6746 - val_loss: 0.5463 - val_recall: 0.8431 - val_precision: 0.6735 - lr: 0.0100
Epoch 5/100
63/63 [==============================] - 2s 38ms/step - loss: 0.5426 - recall: 0.7955 - preci
sion: 0.6769 - val_loss: 0.5407 - val_recall: 0.6120 - val_precision: 0.7566 - lr: 0.0100
Epoch 6/100
63/63 [==============================] - 2s 34ms/step - loss: 0.5247 - recall: 0.8041 - preci
sion: 0.6867 - val_loss: 0.5232 - val_recall: 0.8399 - val_precision: 0.6847 - lr: 0.0100
Epoch 7/100
63/63 [==============================] - 2s 32ms/step - loss: 0.5047 - recall: 0.8172 - preci
sion: 0.7032 - val_loss: 0.5575 - val_recall: 0.5688 - val_precision: 0.7405 - lr: 0.0100
Epoch 8/100
63/63 [==============================] - 2s 31ms/step - loss: 0.5685 - recall: 0.8109 - preci
sion: 0.6474 - val_loss: 0.5210 - val_recall: 0.8001 - val_precision: 0.6975 - lr: 0.0100
Epoch 9/100
63/63 [==============================] - 2s 30ms/step - loss: 0.5130 - recall: 0.8178 - preci
sion: 0.6945 - val_loss: 0.5057 - val_recall: 0.8307 - val_precision: 0.7093 - lr: 0.0100
Epoch 10/100
63/63 [==============================] - 2s 31ms/step - loss: 0.5101 - recall: 0.8303 - preci
sion: 0.6908 - val_loss: 0.4881 - val_recall: 0.8299 - val_precision: 0.7194 - lr: 0.0100
Epoch 11/100
63/63 [==============================] - 2s 33ms/step - loss: 0.5121 - recall: 0.8327 - preci
sion: 0.6915 - val_loss: 0.4978 - val_recall: 0.8551 - val_precision: 0.7022 - lr: 0.0100
Epoch 12/100
63/63 [==============================] - 2s 34ms/step - loss: 0.5000 - recall: 0.8427 - preci
sion: 0.6975 - val_loss: 0.4881 - val_recall: 0.8209 - val_precision: 0.7215 - lr: 0.0100
Epoch 13/100
63/63 [==============================] - 2s 33ms/step - loss: 0.5013 - recall: 0.8228 - preci
sion: 0.6993 - val_loss: 0.4905 - val_recall: 0.8454 - val_precision: 0.7107 - lr: 0.0100
Epoch 14/100
63/63 [==============================] - 2s 38ms/step - loss: 0.5117 - recall: 0.8307 - preci
sion: 0.6937 - val_loss: 0.5255 - val_recall: 0.8230 - val_precision: 0.6989 - lr: 0.0100
Epoch 15/100
63/63 [==============================] - 2s 33ms/step - loss: 0.5005 - recall: 0.8377 - preci
sion: 0.6987 - val_loss: 0.4736 - val_recall: 0.8708 - val_precision: 0.7160 - lr: 0.0100
Epoch 16/100
63/63 [==============================] - 2s 28ms/step - loss: 0.4980 - recall: 0.8441 - preci
sion: 0.6993 - val_loss: 0.4794 - val_recall: 0.8285 - val_precision: 0.7240 - lr: 0.0100
Epoch 17/100
63/63 [==============================] - 2s 28ms/step - loss: 0.5037 - recall: 0.8295 - preci
sion: 0.6964 - val_loss: 0.5011 - val_recall: 0.8685 - val_precision: 0.7022 - lr: 0.0100
Epoch 18/100
63/63 [==============================] - 2s 39ms/step - loss: 0.4874 - recall: 0.8504 - preci
sion: 0.7056 - val_loss: 0.4775 - val_recall: 0.8666 - val_precision: 0.7153 - lr: 0.0100
Epoch 19/100
63/63 [==============================] - 2s 36ms/step - loss: 0.4914 - recall: 0.8478 - preci
sion: 0.6989 - val_loss: 0.5003 - val_recall: 0.7828 - val_precision: 0.7270 - lr: 0.0100
Epoch 20/100
62/63 [============================>.] - ETA: 0s - loss: 0.4884 - recall: 0.8493 - precision:
0.7017
Epoch 20: ReduceLROnPlateau reducing learning rate to 0.0019999999552965165.
63/63 [==============================] - 2s 36ms/step - loss: 0.4887 - recall: 0.8491 - preci
sion: 0.7015 - val_loss: 0.5108 - val_recall: 0.7892 - val_precision: 0.7121 - lr: 0.0100
Epoch 21/100
63/63 [==============================] - 3s 45ms/step - loss: 0.4930 - recall: 0.8363 - preci
sion: 0.7025 - val_loss: 0.4796 - val_recall: 0.8677 - val_precision: 0.7109 - lr: 0.0020
Epoch 22/100
63/63 [==============================] - 3s 50ms/step - loss: 0.4755 - recall: 0.8518 - preci
sion: 0.7165 - val_loss: 0.4688 - val_recall: 0.8664 - val_precision: 0.7224 - lr: 0.0020
```

```
Epoch 23/100
63/63 [==============================] - 3s 46ms/step - loss: 0.4671 - recall: 0.8620 - preci
sion: 0.7209 - val_loss: 0.4633 - val_recall: 0.8724 - val_precision: 0.7244 - lr: 0.0020
Epoch 24/100
63/63 [==============================] - 2s 32ms/step - loss: 0.4616 - recall: 0.8655 - preci
sion: 0.7248 - val_loss: 0.4578 - val_recall: 0.8805 - val_precision: 0.7236 - lr: 0.0020
Epoch 25/100
63/63 [==============================] - 2s 33ms/step - loss: 0.4562 - recall: 0.8702 - preci
sion: 0.7259 - val_loss: 0.4538 - val_recall: 0.8746 - val_precision: 0.7295 - lr: 0.0020
Epoch 26/100
63/63 [==============================] - 2s 25ms/step - loss: 0.4532 - recall: 0.8703 - preci
sion: 0.7288 - val_loss: 0.4494 - val_recall: 0.8718 - val_precision: 0.7375 - lr: 0.0020
Epoch 27/100
63/63 [==============================] - 2s 26ms/step - loss: 0.4494 - recall: 0.8733 - preci
sion: 0.7321 - val_loss: 0.4489 - val_recall: 0.8696 - val_precision: 0.7383 - lr: 0.0020
Epoch 28/100
63/63 [==============================] - 2s 26ms/step - loss: 0.4474 - recall: 0.8713 - preci
sion: 0.7345 - val_loss: 0.4458 - val_recall: 0.8682 - val_precision: 0.7440 - lr: 0.0020
Epoch 29/100
63/63 [==============================] - 2s 30ms/step - loss: 0.4449 - recall: 0.8717 - preci
sion: 0.7366 - val_loss: 0.4405 - val_recall: 0.8713 - val_precision: 0.7471 - lr: 0.0020
Epoch 30/100
63/63 [==============================] - 2s 25ms/step - loss: 0.4411 - recall: 0.8750 - preci
sion: 0.7398 - val_loss: 0.4406 - val_recall: 0.8724 - val_precision: 0.7461 - lr: 0.0020
Epoch 31/100
63/63 [==============================] - 2s 30ms/step - loss: 0.4404 - recall: 0.8741 - preci
sion: 0.7406 - val_loss: 0.4443 - val_recall: 0.8729 - val_precision: 0.7421 - lr: 0.0020
Epoch 32/100
63/63 [==============================] - 2s 36ms/step - loss: 0.4416 - recall: 0.8717 - preci
sion: 0.7386 - val_loss: 0.4412 - val_recall: 0.8430 - val_precision: 0.7572 - lr: 0.0020
Epoch 33/100
63/63 [==============================] - 1s 23ms/step - loss: 0.4404 - recall: 0.8732 - preci
sion: 0.7403 - val_loss: 0.4379 - val_recall: 0.8804 - val_precision: 0.7442 - lr: 0.0020
Epoch 34/100
63/63 [==============================] - 1s 22ms/step - loss: 0.4359 - recall: 0.8757 - preci
sion: 0.7436 - val_loss: 0.4334 - val_recall: 0.8792 - val_precision: 0.7520 - lr: 0.0020
Epoch 35/100
63/63 [==============================] - 1s 21ms/step - loss: 0.4359 - recall: 0.8762 - preci
sion: 0.7434 - val_loss: 0.4332 - val_recall: 0.8661 - val_precision: 0.7593 - lr: 0.0020
Epoch 36/100
63/63 [==============================] - 2s 27ms/step - loss: 0.4352 - recall: 0.8762 - preci
sion: 0.7447 - val_loss: 0.4332 - val_recall: 0.8845 - val_precision: 0.7442 - lr: 0.0020
Epoch 37/100
63/63 [==============================] - 2s 34ms/step - loss: 0.4323 - recall: 0.8784 - preci
sion: 0.7454 - val_loss: 0.4318 - val_recall: 0.8714 - val_precision: 0.7567 - lr: 0.0020
Epoch 38/100
63/63 [==============================] - 2s 26ms/step - loss: 0.4337 - recall: 0.8772 - preci
sion: 0.7440 - val_loss: 0.4353 - val_recall: 0.8940 - val_precision: 0.7375 - lr: 0.0020
Epoch 39/100
63/63 [==============================] - 2s 25ms/step - loss: 0.4319 - recall: 0.8811 - preci
sion: 0.7459 - val_loss: 0.4314 - val_recall: 0.8831 - val_precision: 0.7527 - lr: 0.0020
Epoch 40/100
63/63 [==============================] - 2s 25ms/step - loss: 0.4312 - recall: 0.8778 - preci
sion: 0.7453 - val_loss: 0.4315 - val_recall: 0.8759 - val_precision: 0.7550 - lr: 0.0020
Epoch 41/100
63/63 [==============================] - 1s 22ms/step - loss: 0.4301 - recall: 0.8788 - preci
sion: 0.7465 - val_loss: 0.4342 - val_recall: 0.8818 - val_precision: 0.7454 - lr: 0.0020
Epoch 42/100
63/63 [==============================] - 1s 23ms/step - loss: 0.4302 - recall: 0.8773 - preci
sion: 0.7451 - val_loss: 0.4246 - val_recall: 0.8822 - val_precision: 0.7604 - lr: 0.0020
Epoch 43/100
63/63 [==============================] - 1s 24ms/step - loss: 0.4277 - recall: 0.8793 - preci
sion: 0.7478 - val_loss: 0.4299 - val_recall: 0.8935 - val_precision: 0.7447 - lr: 0.0020
Epoch 44/100
63/63 [==============================] - 1s 22ms/step - loss: 0.4276 - recall: 0.8772 - preci
sion: 0.7485 - val_loss: 0.4279 - val_recall: 0.8947 - val_precision: 0.7448 - lr: 0.0020
Epoch 45/100
63/63 [==============================] - 1s 23ms/step - loss: 0.4251 - recall: 0.8800 - preci
sion: 0.7506 - val_loss: 0.4245 - val_recall: 0.8670 - val_precision: 0.7684 - lr: 0.0020
```

```
Epoch 46/100
63/63 [==============================] - 1s 23ms/step - loss: 0.4248 - recall: 0.8805 - preci
sion: 0.7508 - val_loss: 0.4237 - val_recall: 0.8823 - val_precision: 0.7589 - lr: 0.0020
Epoch 47/100
63/63 [==============================] - 2s 25ms/step - loss: 0.4270 - recall: 0.8781 - preci
sion: 0.7484 - val_loss: 0.4274 - val_recall: 0.8848 - val_precision: 0.7506 - lr: 0.0020
Epoch 48/100
63/63 [==============================] - 2s 25ms/step - loss: 0.4229 - recall: 0.8801 - preci
sion: 0.7527 - val_loss: 0.4244 - val_recall: 0.8732 - val_precision: 0.7619 - lr: 0.0020
Epoch 49/100
63/63 [==============================] - 2s 28ms/step - loss: 0.4249 - recall: 0.8788 - preci
sion: 0.7510 - val_loss: 0.4252 - val_recall: 0.8702 - val_precision: 0.7652 - lr: 0.0020
Epoch 50/100
63/63 [==============================] - 2s 34ms/step - loss: 0.4231 - recall: 0.8783 - preci
sion: 0.7523 - val_loss: 0.4222 - val_recall: 0.8793 - val_precision: 0.7622 - lr: 0.0020
Epoch 51/100
63/63 [==============================] - 2s 35ms/step - loss: 0.4243 - recall: 0.8781 - preci
sion: 0.7509 - val_loss: 0.4268 - val_recall: 0.8914 - val_precision: 0.7490 - lr: 0.0020
Epoch 52/100
63/63 [==============================] - 2s 32ms/step - loss: 0.4206 - recall: 0.8815 - preci
sion: 0.7539 - val_loss: 0.4326 - val_recall: 0.8874 - val_precision: 0.7483 - lr: 0.0020
Epoch 53/100
63/63 [==============================] - 2s 32ms/step - loss: 0.4226 - recall: 0.8792 - preci
sion: 0.7527 - val_loss: 0.4196 - val_recall: 0.8870 - val_precision: 0.7605 - lr: 0.0020
Epoch 54/100
63/63 [==============================] - 2s 33ms/step - loss: 0.4206 - recall: 0.8806 - preci
sion: 0.7540 - val_loss: 0.4214 - val_recall: 0.8990 - val_precision: 0.7514 - lr: 0.0020
Epoch 55/100
63/63 [==============================] - 2s 35ms/step - loss: 0.4234 - recall: 0.8789 - preci
sion: 0.7512 - val_loss: 0.4245 - val_recall: 0.8860 - val_precision: 0.7522 - lr: 0.0020
Epoch 56/100
63/63 [==============================] - 2s 30ms/step - loss: 0.4205 - recall: 0.8796 - preci
sion: 0.7528 - val_loss: 0.4219 - val_recall: 0.8796 - val_precision: 0.7607 - lr: 0.0020
Epoch 57/100
63/63 [==============================] - 1s 23ms/step - loss: 0.4218 - recall: 0.8785 - preci
sion: 0.7535 - val_loss: 0.4218 - val_recall: 0.8818 - val_precision: 0.7588 - lr: 0.0020
Epoch 58/100
63/63 [==============================] - 2s 24ms/step - loss: 0.4185 - recall: 0.8796 - preci
sion: 0.7561 - val_loss: 0.4168 - val_recall: 0.8916 - val_precision: 0.7611 - lr: 0.0020
Epoch 59/100
63/63 [==============================] - 2s 27ms/step - loss: 0.4205 - recall: 0.8778 - preci
sion: 0.7542 - val_loss: 0.4166 - val_recall: 0.8900 - val_precision: 0.7628 - lr: 0.0020
Epoch 60/100
63/63 [==============================] - 2s 34ms/step - loss: 0.4170 - recall: 0.8803 - preci
sion: 0.7575 - val_loss: 0.4178 - val_recall: 0.8974 - val_precision: 0.7580 - lr: 0.0020
Epoch 61/100
63/63 [==============================] - 2s 27ms/step - loss: 0.4157 - recall: 0.8813 - preci
sion: 0.7587 - val_loss: 0.4175 - val_recall: 0.8984 - val_precision: 0.7569 - lr: 0.0020
Epoch 62/100
63/63 [==============================] - 1s 22ms/step - loss: 0.4208 - recall: 0.8773 - preci
sion: 0.7534 - val_loss: 0.4165 - val_recall: 0.8716 - val_precision: 0.7718 - lr: 0.0020
Epoch 63/100
63/63 [==============================] - 2s 25ms/step - loss: 0.4155 - recall: 0.8819 - preci
sion: 0.7596 - val_loss: 0.4178 - val_recall: 0.9008 - val_precision: 0.7548 - lr: 0.0020
Epoch 64/100
63/63 [==============================] - 1s 23ms/step - loss: 0.4143 - recall: 0.8820 - preci
sion: 0.7589 - val_loss: 0.4199 - val_recall: 0.8891 - val_precision: 0.7581 - lr: 0.0020
Epoch 65/100
63/63 [==============================] - 1s 23ms/step - loss: 0.4177 - recall: 0.8810 - preci
sion: 0.7552 - val_loss: 0.4179 - val_recall: 0.8677 - val_precision: 0.7750 - lr: 0.0020
Epoch 66/100
63/63 [==============================] - 1s 23ms/step - loss: 0.4206 - recall: 0.8777 - preci
sion: 0.7527 - val_loss: 0.4189 - val_recall: 0.8968 - val_precision: 0.7560 - lr: 0.0020
Epoch 67/100
62/63 [============================>.] - ETA: 0s - loss: 0.4190 - recall: 0.8792 - precision:
0.7542
Epoch 67: ReduceLROnPlateau reducing learning rate to 0.0003999999724328518.
63/63 [==============================] - 1s 23ms/step - loss: 0.4188 - recall: 0.8792 - preci
sion: 0.7544 - val_loss: 0.4167 - val_recall: 0.8632 - val_precision: 0.7753 - lr: 0.0020
```

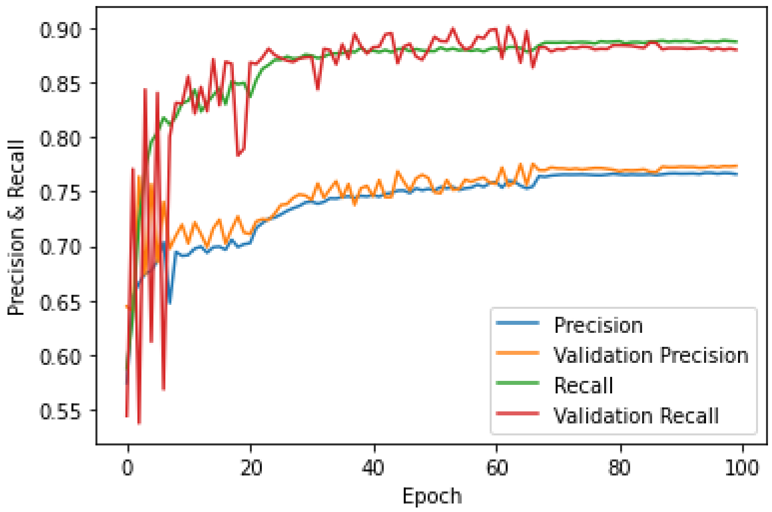
```
Epoch 68/100
63/63 [==============================] - 2s 33ms/step - loss: 0.4092 - recall: 0.8834 - preci
sion: 0.7639 - val_loss: 0.4128 - val_recall: 0.8815 - val_precision: 0.7694 - lr: 4.0000e-04
Epoch 69/100
63/63 [==============================] - 2s 35ms/step - loss: 0.4087 - recall: 0.8861 - preci
sion: 0.7633 - val_loss: 0.4129 - val_recall: 0.8819 - val_precision: 0.7693 - lr: 4.0000e-04
Epoch 70/100
63/63 [==============================] - 2s 28ms/step - loss: 0.4078 - recall: 0.8860 - preci
sion: 0.7645 - val_loss: 0.4124 - val_recall: 0.8778 - val_precision: 0.7720 - lr: 4.0000e-04
Epoch 71/100
63/63 [==============================] - 2s 27ms/step - loss: 0.4073 - recall: 0.8859 - preci
sion: 0.7650 - val_loss: 0.4125 - val_recall: 0.8801 - val_precision: 0.7711 - lr: 4.0000e-04
Epoch 72/100
63/63 [==============================] - 2s 24ms/step - loss: 0.4070 - recall: 0.8863 - preci
sion: 0.7654 - val_loss: 0.4122 - val_recall: 0.8795 - val_precision: 0.7710 - lr: 4.0000e-04
Epoch 73/100
63/63 [==============================] - 2s 25ms/step - loss: 0.4076 - recall: 0.8859 - preci
sion: 0.7653 - val_loss: 0.4121 - val_recall: 0.8818 - val_precision: 0.7702 - lr: 4.0000e-04
Epoch 74/100
63/63 [==============================] - 2s 30ms/step - loss: 0.4064 - recall: 0.8863 - preci
sion: 0.7654 - val_loss: 0.4123 - val_recall: 0.8810 - val_precision: 0.7708 - lr: 4.0000e-04
Epoch 75/100
63/63 [==============================] - 2s 29ms/step - loss: 0.4069 - recall: 0.8865 - preci
sion: 0.7655 - val_loss: 0.4121 - val_recall: 0.8823 - val_precision: 0.7699 - lr: 4.0000e-04
Epoch 76/100
63/63 [==============================] - 2s 26ms/step - loss: 0.4067 - recall: 0.8861 - preci
sion: 0.7652 - val_loss: 0.4119 - val_recall: 0.8820 - val_precision: 0.7707 - lr: 4.0000e-04
Epoch 77/100
63/63 [==============================] - 2s 33ms/step - loss: 0.4064 - recall: 0.8852 - preci
sion: 0.7649 - val_loss: 0.4121 - val_recall: 0.8797 - val_precision: 0.7714 - lr: 4.0000e-04
Epoch 78/100
63/63 [==============================] - 2s 34ms/step - loss: 0.4068 - recall: 0.8872 - preci
sion: 0.7648 - val_loss: 0.4122 - val_recall: 0.8804 - val_precision: 0.7714 - lr: 4.0000e-04
Epoch 79/100
63/63 [==============================] - 2s 26ms/step - loss: 0.4060 - recall: 0.8866 - preci
sion: 0.7653 - val_loss: 0.4122 - val_recall: 0.8802 - val_precision: 0.7706 - lr: 4.0000e-04
Epoch 80/100
63/63 [==============================] - 2s 37ms/step - loss: 0.4064 - recall: 0.8870 - preci
sion: 0.7661 - val_loss: 0.4115 - val_recall: 0.8836 - val_precision: 0.7698 - lr: 4.0000e-04
Epoch 81/100
63/63 [==============================] - 2s 31ms/step - loss: 0.4063 - recall: 0.8873 - preci
sion: 0.7655 - val_loss: 0.4122 - val_recall: 0.8836 - val_precision: 0.7684 - lr: 4.0000e-04
Epoch 82/100
63/63 [==============================] - 2s 33ms/step - loss: 0.4062 - recall: 0.8853 - preci
sion: 0.7651 - val_loss: 0.4113 - val_recall: 0.8834 - val_precision: 0.7695 - lr: 4.0000e-04
Epoch 83/100
63/63 [==============================] - 2s 34ms/step - loss: 0.4057 - recall: 0.8864 - preci
sion: 0.7655 - val_loss: 0.4125 - val_recall: 0.8830 - val_precision: 0.7691 - lr: 4.0000e-04
Epoch 84/100
63/63 [==============================] - 2s 31ms/step - loss: 0.4059 - recall: 0.8865 - preci
sion: 0.7652 - val_loss: 0.4125 - val_recall: 0.8820 - val_precision: 0.7695 - lr: 4.0000e-04
Epoch 85/100
63/63 [==============================] - 2s 35ms/step - loss: 0.4052 - recall: 0.8858 - preci
sion: 0.7653 - val_loss: 0.4120 - val_recall: 0.8811 - val_precision: 0.7701 - lr: 4.0000e-04
Epoch 86/100
63/63 [==============================] - 2s 33ms/step - loss: 0.4056 - recall: 0.8871 - preci
sion: 0.7656 - val_loss: 0.4123 - val_recall: 0.8861 - val_precision: 0.7680 - lr: 4.0000e-04
Epoch 87/100
61/63 [============================>.] - ETA: 0s - loss: 0.4062 - recall: 0.8867 - precision:
0.7649
Epoch 87: ReduceLROnPlateau reducing learning rate to 0.0001.
63/63 [==============================] - 2s 32ms/step - loss: 0.4060 - recall: 0.8871 - preci
sion: 0.7649 - val_loss: 0.4121 - val_recall: 0.8857 - val_precision: 0.7676 - lr: 4.0000e-04
Epoch 88/100
63/63 [==============================] - 3s 41ms/step - loss: 0.4051 - recall: 0.8878 - preci
sion: 0.7655 - val_loss: 0.4107 - val_recall: 0.8798 - val_precision: 0.7726 - lr: 1.0000e-04
Epoch 89/100
63/63 [==============================] - 2s 31ms/step - loss: 0.4044 - recall: 0.8867 - preci
sion: 0.7663 - val_loss: 0.4105 - val_recall: 0.8809 - val_precision: 0.7720 - lr: 1.0000e-04
```

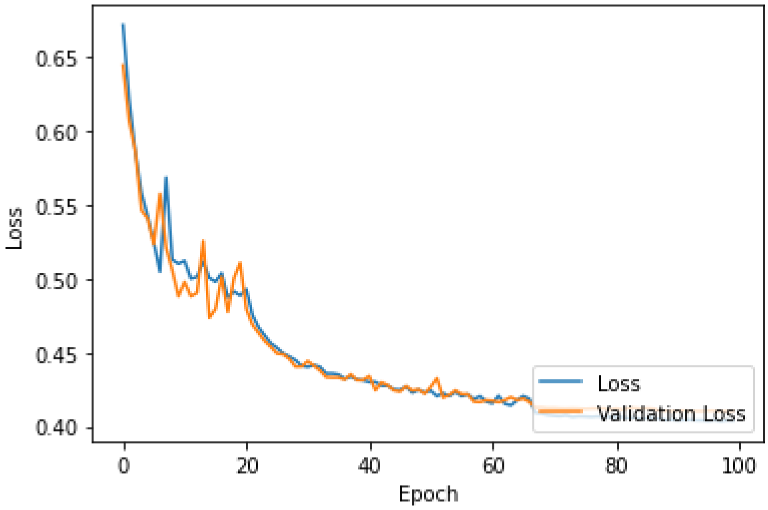
```
Epoch 90/100
63/63 [==============================] - 2s 36ms/step - loss: 0.4045 - recall: 0.8874 - preci
sion: 0.7665 - val_loss: 0.4104 - val_recall: 0.8809 - val_precision: 0.7720 - lr: 1.0000e-04
Epoch 91/100
63/63 [==============================] - 3s 41ms/step - loss: 0.4041 - recall: 0.8871 - preci
sion: 0.7662 - val_loss: 0.4105 - val_recall: 0.8809 - val_precision: 0.7726 - lr: 1.0000e-04
Epoch 92/100
63/63 [==============================] - 2s 38ms/step - loss: 0.4044 - recall: 0.8877 - preci
sion: 0.7663 - val_loss: 0.4107 - val_recall: 0.8804 - val_precision: 0.7723 - lr: 1.0000e-04
Epoch 93/100
63/63 [==============================] - 2s 32ms/step - loss: 0.4045 - recall: 0.8868 - preci
sion: 0.7664 - val_loss: 0.4105 - val_recall: 0.8807 - val_precision: 0.7724 - lr: 1.0000e-04
Epoch 94/100
63/63 [==============================] - 2s 35ms/step - loss: 0.4042 - recall: 0.8859 - preci
sion: 0.7659 - val_loss: 0.4104 - val_recall: 0.8811 - val_precision: 0.7717 - lr: 1.0000e-04
Epoch 95/100
63/63 [==============================] - 2s 34ms/step - loss: 0.4041 - recall: 0.8873 - preci
sion: 0.7670 - val_loss: 0.4103 - val_recall: 0.8813 - val_precision: 0.7719 - lr: 1.0000e-04
Epoch 96/100
63/63 [==============================] - 3s 39ms/step - loss: 0.4038 - recall: 0.8871 - preci
sion: 0.7669 - val_loss: 0.4106 - val_recall: 0.8794 - val_precision: 0.7729 - lr: 1.0000e-04
Epoch 97/100
63/63 [==============================] - 2s 28ms/step - loss: 0.4043 - recall: 0.8868 - preci
sion: 0.7661 - val_loss: 0.4107 - val_recall: 0.8809 - val_precision: 0.7720 - lr: 1.0000e-04
Epoch 98/100
63/63 [==============================] - 2s 24ms/step - loss: 0.4036 - recall: 0.8880 - preci
sion: 0.7668 - val_loss: 0.4103 - val_recall: 0.8795 - val_precision: 0.7731 - lr: 1.0000e-04
Epoch 99/100
63/63 [==============================] - 1s 23ms/step - loss: 0.4042 - recall: 0.8874 - preci
sion: 0.7666 - val_loss: 0.4103 - val_recall: 0.8806 - val_precision: 0.7728 - lr: 1.0000e-04
Epoch 100/100
63/63 [==============================] - 2s 38ms/step - loss: 0.4040 - recall: 0.8869 - preci
sion: 0.7658 - val_loss: 0.4105 - val_recall: 0.8794 - val_precision: 0.7733 - lr: 1.0000e-04
```

In [ ]:

```
# Plot F1score
plt.plot(history.history['precision'], label='Precision')
plt.plot(history.history['val_precision'], label='Validation Precision')
plt.plot(history.history['recall'], label='Recall')
plt.plot(history.history['val_recall'], label='Validation Recall')
plt.xlabel('Epoch')
plt.ylabel('Precision & Recall')
plt.legend(loc='lower right')
plt.show()

# Plot Loss
plt.plot(history.history['loss'], label='Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend(loc='lower right')
plt.show()
```





Model Evaluation

The model's performance is evaluated using precision, recall, and F1 score. The precision and recall for each epoch during training are visualized using matplotlib. The F1 score is calculated after making predictions on the validation set.

In [ ]:
```
# Predict on validation set
y_pred = model.predict(valid_dataset)
y_pred = np.where(y_pred > 0.5, 1, 0)
y_pred = y_pred.reshape(-1)

# Print number of 1s and 0s
print("1 count", np.count_nonzero(y_pred == 1))
print("0 count", np.count_nonzero(y_pred == 0))

# Print F1 Score
print("F1 Score : ", f1_score(y_val.numpy().astype(int), y_pred))
```

```
16/16 [==============================] - 0s 11ms/step
1 count 18385
0 count 13672
F1 Score :  0.8229573673699747
```

In [ ]:
```
def format_customers(df_customers, df_locations):
    df_customers = df_customers.drop(["language", "dob", "created_at", "updated_at"], axis=1)

    df_customers["gender"] = df_customers["gender"].str.strip()
    df_customers["gender"] = df_customers["gender"].str.upper()
    df_customers["gender"] = df_customers["gender"].map({'MALE': 0, 'FEMALE': 1})
    df_customers["gender"] = df_customers.gender.fillna(2).astype(int)

    df_customers = df_customers.drop(["verified", "status"], axis=1)
    df_customers = df_customers.rename(columns={"akeed_customer_id": "customer_id"})

    df_customers = pd.merge(df_customers, df_locations[["customer_id", "location_number", "la
    df_customers = df_customers.fillna(0)

    df_customers["gender"] = df_customers["gender"].astype(np.uint8)
    df_customers["location_number"] = df_customers["location_number"].astype(np.uint8)
    df_customers["latitude"] = df_customers["latitude"].astype(np.float16)
    df_customers["longitude"] = df_customers["longitude"].astype(np.float16)

    return df_customers

def format_items(df_orders):
    df_orders = df_orders[["akeed_order_id", "customer_id", "vendor_id", "deliverydistance"]]

    df_orders = df_orders.drop(["akeed_order_id"], axis=1)
    return df_orders
```

```python
def format_vendors(df_vendors):
    to_remove = [
            "vendor_category_en",
            "authentication_id",
            "OpeningTime",
            "OpeningTime2",
            "open_close_flags",
            "vendor_tag_name",
            "created_at",
            "updated_at",
            "commission",
            "saturday_to_time1",
            "saturday_from_time2",
            "saturday_from_time1",
            "saturday_to_time2",
            "thursday_to_time1",
            "thursday_from_time1",
            "thursday_from_time2",
            "thursday_to_time2",
            "tuesday_to_time1",
            "tuesday_from_time2",
            "tuesday_from_time1",
            "tuesday_to_time2",
            "monday_to_time1",
            "monday_from_time1",
            "monday_from_time2",
            "monday_to_time2",
            "sunday_to_time1",
            "sunday_from_time1",
            "sunday_from_time2",
            "sunday_to_time2",
            "friday_to_time1",
            "friday_from_time1",
            "friday_from_time2",
            "friday_to_time2",
            "wednesday_to_time1",
            "wednesday_from_time1",
            "wednesday_from_time2",
            "wednesday_to_time2",
            "one_click_vendor",
            "country_id",
            "city_id",
            "display_orders",
            "device_type",
            "is_akeed_delivering",
            "language",
            "rank",
            "is_open"
    ]

    df_vendors = df_vendors.drop(to_remove, axis=1)
    # remove all unverified accounts
    # df_vendors = df_vendors[df_vendors["verified"] == 1]
    df_vendors = df_vendors.drop("verified", axis=1)
    df_vendors["primary_tags"] = df_vendors["primary_tags"].fillna("{\"primary_tags\":\"0\"}"

    # one hot encoding for vendor_tag
    df_vendors_tags = df_vendors.vendor_tag.str.split(",")
    df_vendors_tags = df_vendors_tags.apply(lambda d: d if isinstance(d, list) else [])

    mlb = MultiLabelBinarizer()
    one_hot_encoded_data = mlb.fit_transform(df_vendors_tags)
    one_hot_encoded_data = pd.DataFrame(one_hot_encoded_data, columns=mlb.classes_, dtype=boo

    # merge df_train_items and one_hot_encoded_data
    df_vendors = df_vendors.drop(["vendor_tag"], axis=1)
    df_vendors = df_vendors.reset_index(drop=True)
    one_hot_encoded_data = one_hot_encoded_data.reset_index(drop=True)
```

```
    df_vendors = pd.concat([df_vendors, one_hot_encoded_data], axis=1)

    df_vendors["latitude"] = df_vendors["latitude"].astype(np.float16)
    df_vendors["longitude"] = df_vendors["longitude"].astype(np.float16)
    df_vendors["vendor_category_id"] = df_vendors["vendor_category_id"].astype(np.uint8)
    df_vendors["delivery_charge"] = df_vendors["delivery_charge"].astype(bool)
    df_vendors["serving_distance"] = df_vendors["serving_distance"].astype(np.uint8)
    df_vendors["prepration_time"] = df_vendors["prepration_time"].astype(np.uint8)
    df_vendors["discount_percentage"] = df_vendors["discount_percentage"].astype(np.uint8)
    df_vendors["status"] = df_vendors["status"].astype(bool)
    df_vendors["vendor_rating"] = df_vendors["vendor_rating"].astype(np.float16)
    df_vendors["primary_tags"] = df_vendors["primary_tags"].astype(np.uint16)
    df_vendors["id"] = df_vendors["id"].astype(np.uint16)

    return df_vendors

def format_test_dataset(df_customers, df_orders, df_vendors, df_locations):
    df_customers = format_customers(df_customers, df_locations)
    df_orders = format_items(df_orders)
    df_vendors = format_vendors(df_vendors)

    print(df_customers.shape, df_orders.shape, df_vendors.shape)

    # get the cross merge product of users, items and rests

    C = pd.merge(df_customers, df_vendors, how="cross")
    sub = C[["customer_id", "location_number", "id"]]
    C = C.drop(["customer_id", "id"], axis=1)

    return C, sub
```

In [ ]:

```
df_X_test, df_submission = format_test_dataset(df_test_customers, df_orders, df_vendors, df_t
```

```
(16736, 5) (135303, 3) (100, 78)
```

In [ ]:

```
X_test = tf.keras.utils.normalize(tf.convert_to_tensor(df_X_test.to_numpy(), dtype=tf.float32
```

In [ ]:

```
# Predict the test set
y_pred = model.predict(X_test)
# Convert the prediction to binary
y_pred = np.where(y_pred > 0.5, 1, 0)
# Reshape the prediction to a 1D array
y_pred = y_pred.reshape(-1)
```

```
52300/52300 [==============================] - 183s 3ms/step
```

Prediction

Finally, the trained model is used to predict whether a customer will place an order. The predictions are saved to a CSV file.

In [ ]:

```
# Create the submission file
df_submission["CID X LOC_NUM X VENDOR"] = df_submission["customer_id"].astype(str) + " X " +
df_submission["target"] = y_pred
df_submission = df_submission.drop(["customer_id", "location_number", "id"], axis=1)
df_submission.to_csv("submission.csv", index=False)
```

Thank you for reading this notebook, I hope you enjoyed it ! 😄