Instacart Market Basket Analysis

ML Portfolio Project

Maria Opekhtina

ML-B17, April 2023

Agenda

Company Intro and Background

Data

Visualizations

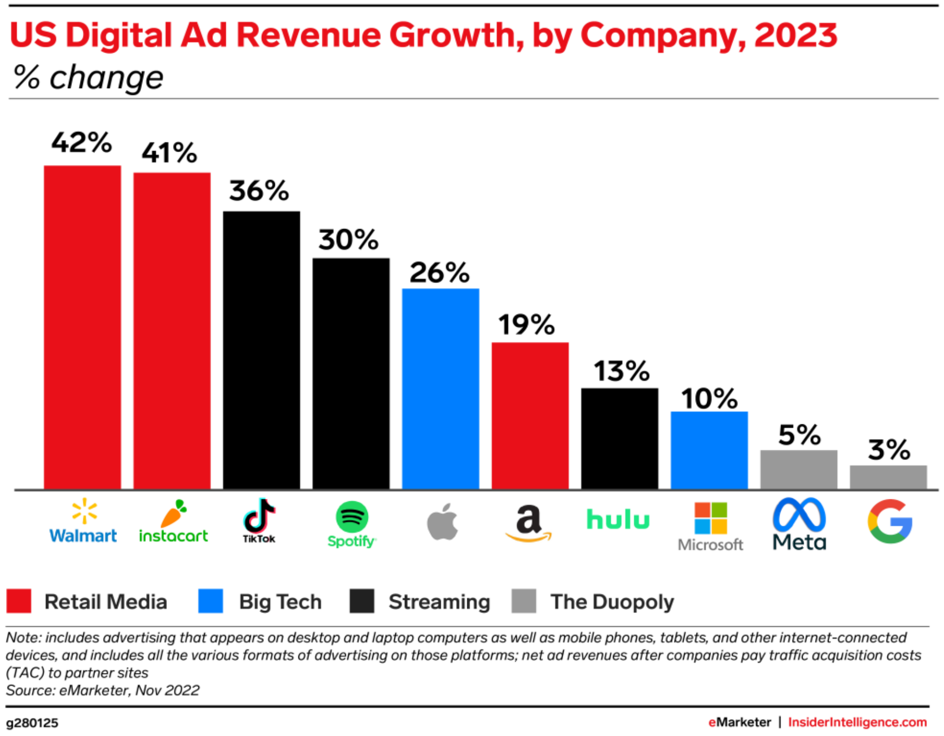
Code

Model Evaluation

Challenges and Next Steps

What is Instacart?

Instacart is an American delivery company that operates a grocery delivery and pick-up service in the United States and Canada. The company offers its services via a website and mobile app. The service allows customers to order groceries from participating retailers with the shopping being done by a personal shopper.



- Instacart's success is largely due to its push into advertising, as well as growth for its Instacart+ membership program.
- US digital grocery sales grew by 15.8% in 2022 and are set to grow another 14.8% this year.
- New initiatives - Instacart Business and Instacart Health,
- In 2023, Instacart plans to roll out a chatbot powered by ChatGPT to answer food-related questions and help shoppers find products

How does it work?

✓ Pro

- available throughout the United States and Canada
- optional cost-saving Instacart+ subscription option
- same-day delivery available
- partners with several large grocery store chains

✕ Con

- not available in some rural areas
- without Instacart+, delivery fee and service fee apply
- Instacart prices may be more expensive than those in-store
- some may find the service difficult to use

Project Objective

To predict which products appeared in the user's latest order based on customer's purchase history

- Better recommendations
- Inventory Management
- Targeted advertisement
- Promotions
- Customer retention
- User satisfaction

Dataset

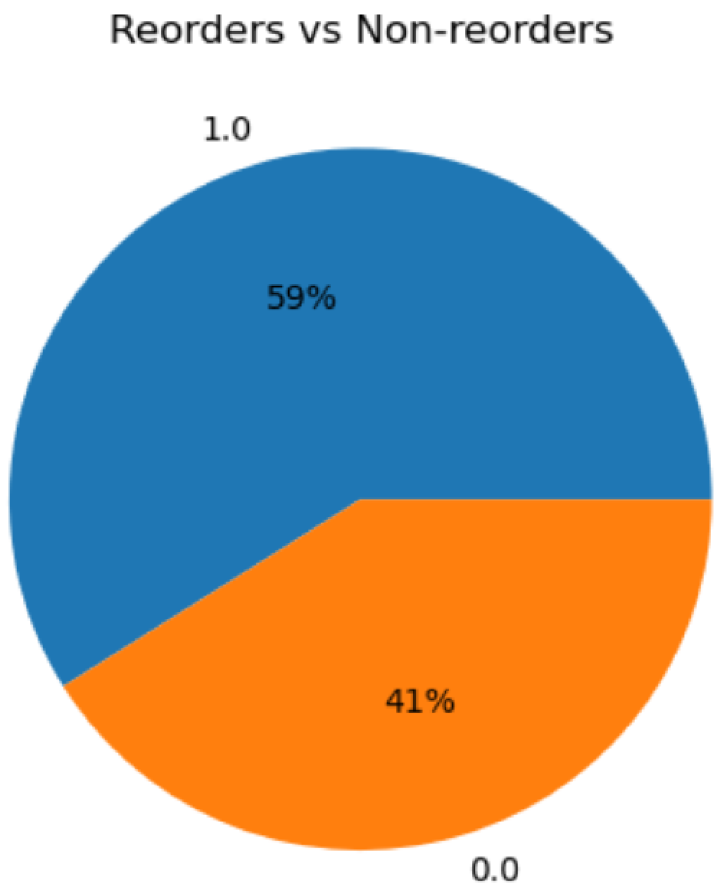
- The dataset is anonymized and contains a sample of over 3 million grocery orders from more than 200,000 Instacart users
- 6 tables with order and product information

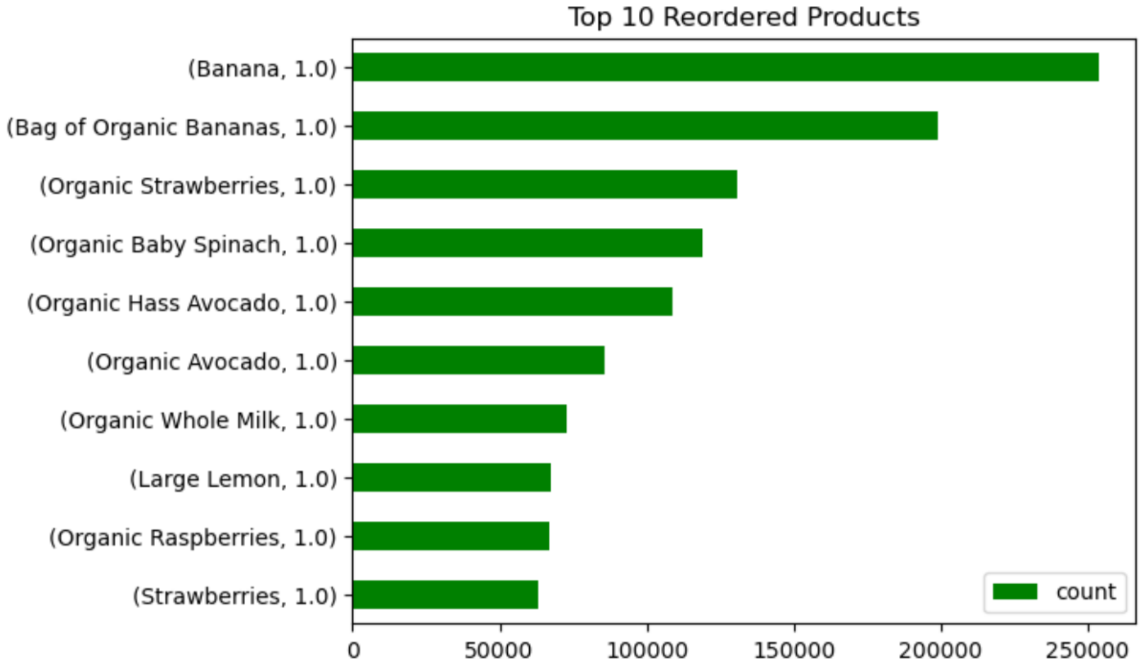
	order_id	user_id	eval_set	order_number	order_dow	order_hour_of_day	days_since_prior_order	product_id	add_to_cart_order	reordered	product_name	aisle_id	department_id	aisle	department
0	2539329	1	prior	1	2	8	NaN	196.0	1.0	0.0	Soda	77	7	soft drinks	beverages
1	2539329	1	prior	1	2	8	NaN	14080.0	2.0	0.0	Organic Unsweetened Vanilla Almond Milk	91	16	soy lactosefree	dairy eggs
2	2539329	1	prior	1	2	8	NaN	12424.0	3.0	0.0	Original Beef Jerky	23	19	popcorn jerky	snacks
3	2539329	1	prior	1	2	8	NaN	26080.0	4.0	0.0	Aged White Cheddar Popcorn	23	19	popcorn jerky	snacks
4	2539329	1	prior	1	2	8	NaN	26400.0	5.0	0.0	XL Pick-A-Size Paper Towel Rolls	54	17	paper goods	household
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
20641986	2977660	206209	prior	13	1	12	7.0	14200.0	5.0	1.0	Tomato Paste	9	9	pasta sauce	dry goods pasta
20641987	2977660	206209	prior	13	1	12	7.0	38720.0	6.0	0.0	Brownie Crunch High Protein Bar	3	19	energy granola bars	snacks
20641988	2977660	206209	prior	13	1	12	7.0	31470.0	7.0	0.0	High Protein Bar Chunky Peanut Butter	3	19	energy granola bars	snacks
20641989	2977660	206209	prior	13	1	12	7.0	6570.0	8.0	0.0	Chocolate Peanut Butter Protein Bar	3	19	energy granola bars	snacks
20641990	2977660	206209	prior	13	1	12	7.0	22910.0	9.0	0.0	Roasted & Salted Shelled Pistachios	117	19	nuts seeds dried fruit	snacks

20641991 rows × 15 columns

Interesting Findings

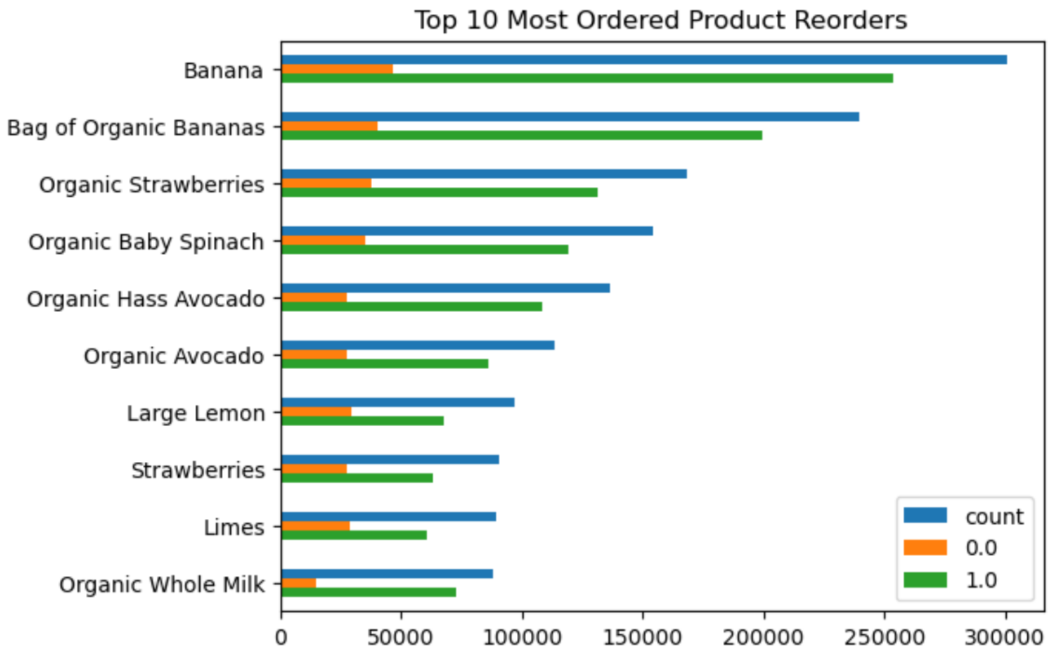
- Most of the orders that were placed had been reordered items
- Most reordered items are fruits and vegetables

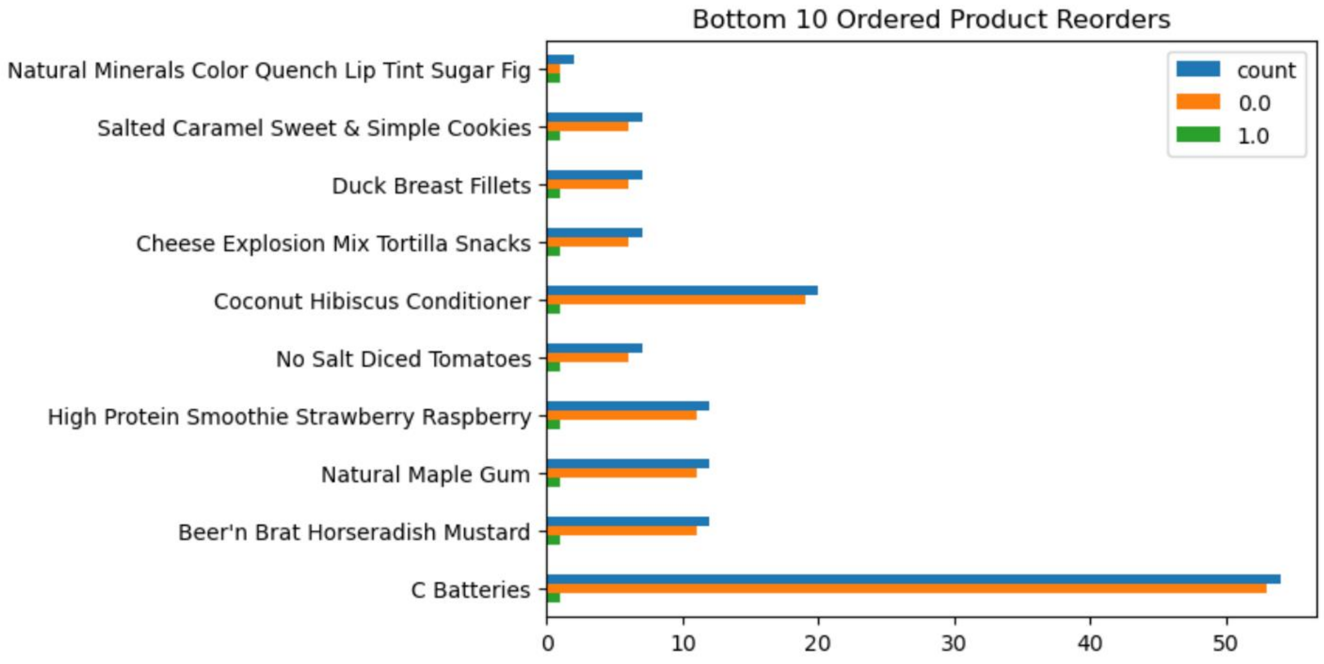




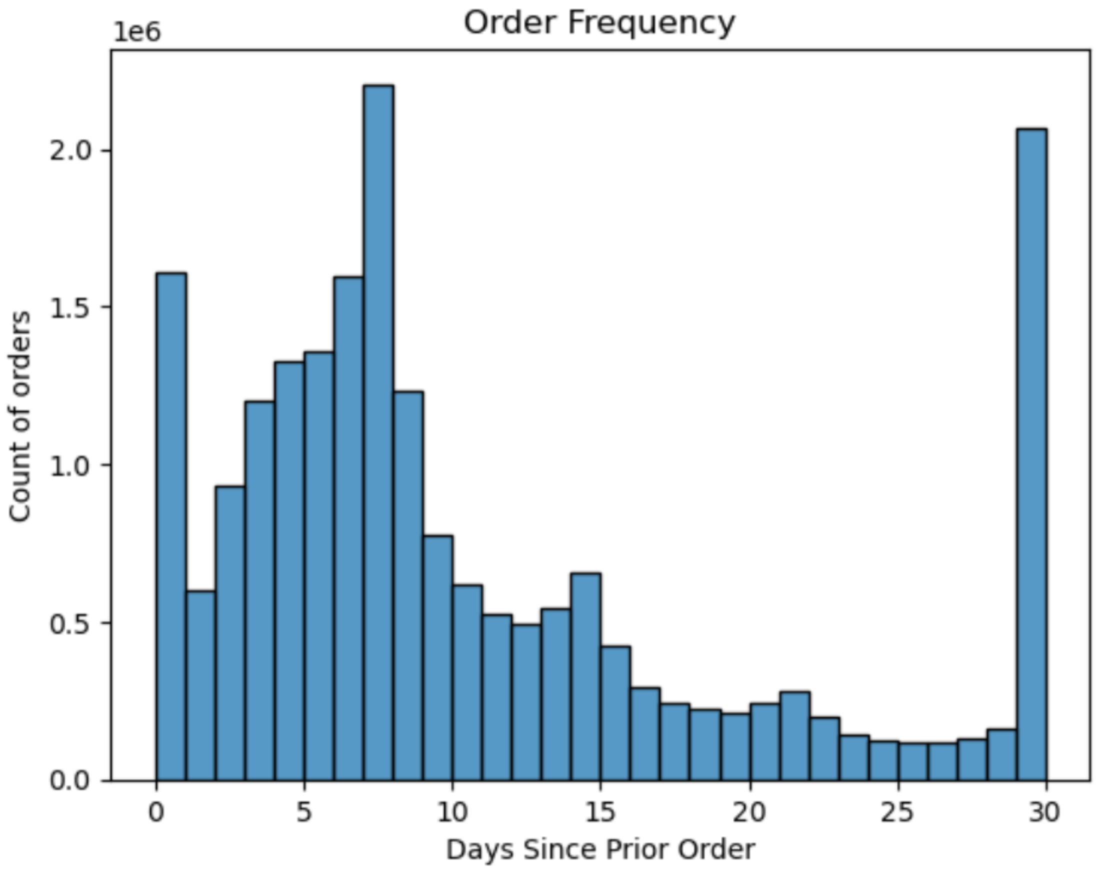
Most and Least Ordered Products

- Top 10 most ordered products are nicely aligned with top 10 reorders
- Bottom 10 are completely different, however indicating that people are likely not enjoying the items





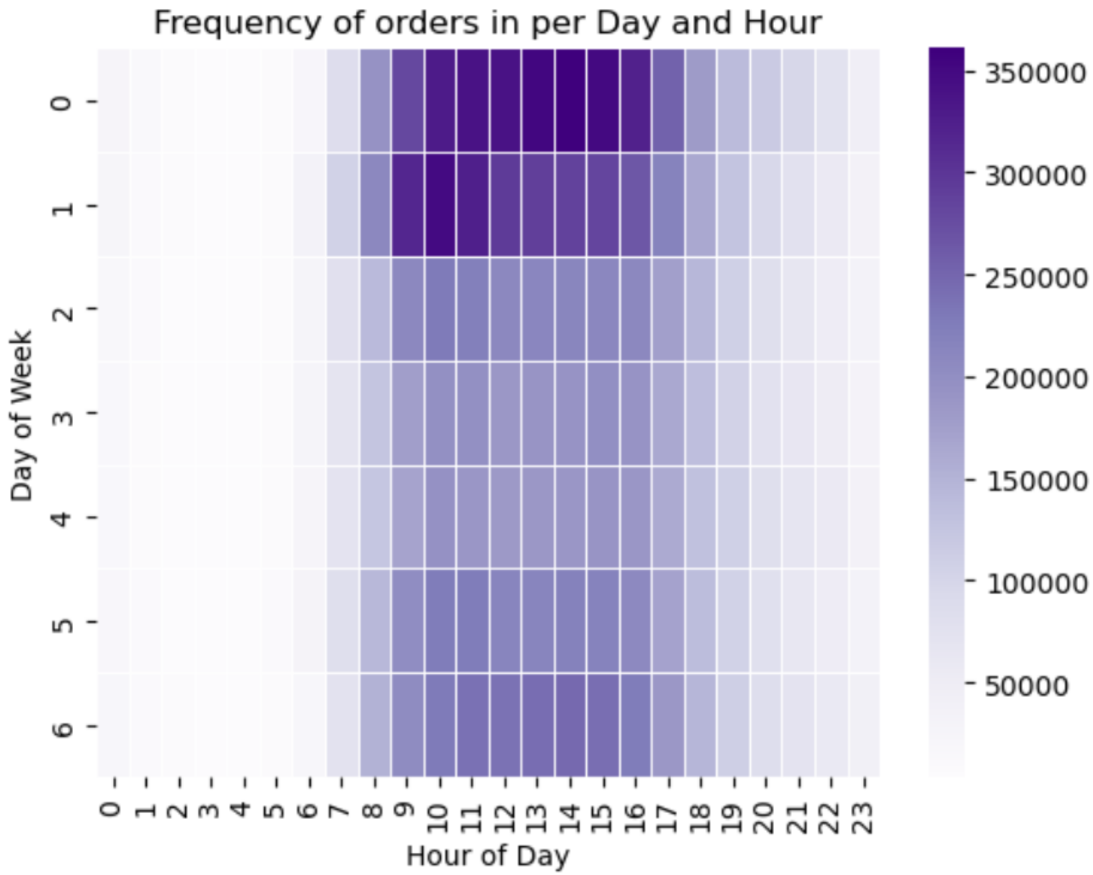
Order Frequency

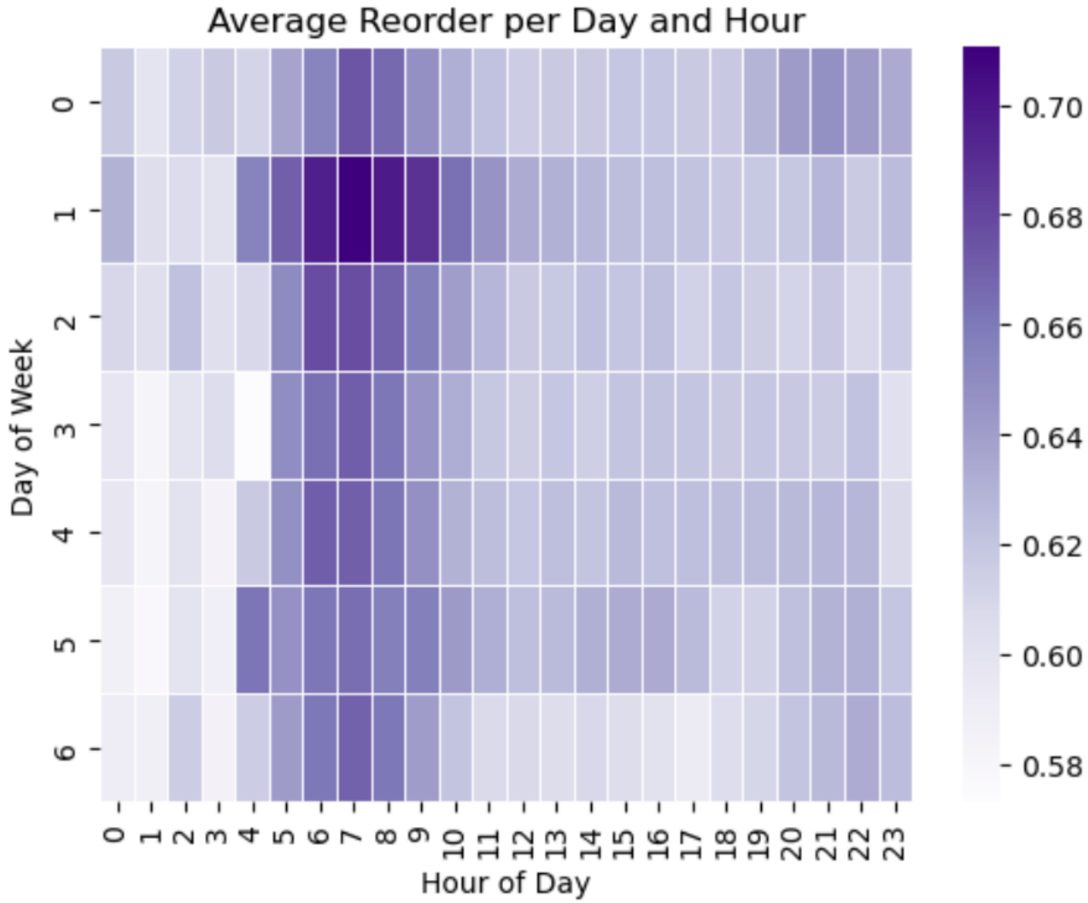


Peaks:

- 0 days since prior order:
  - ➢ indicates 1st order
- 7 days:
  - ➢ users shop weekly
- 30 days:
  - ➢ users shop monthly

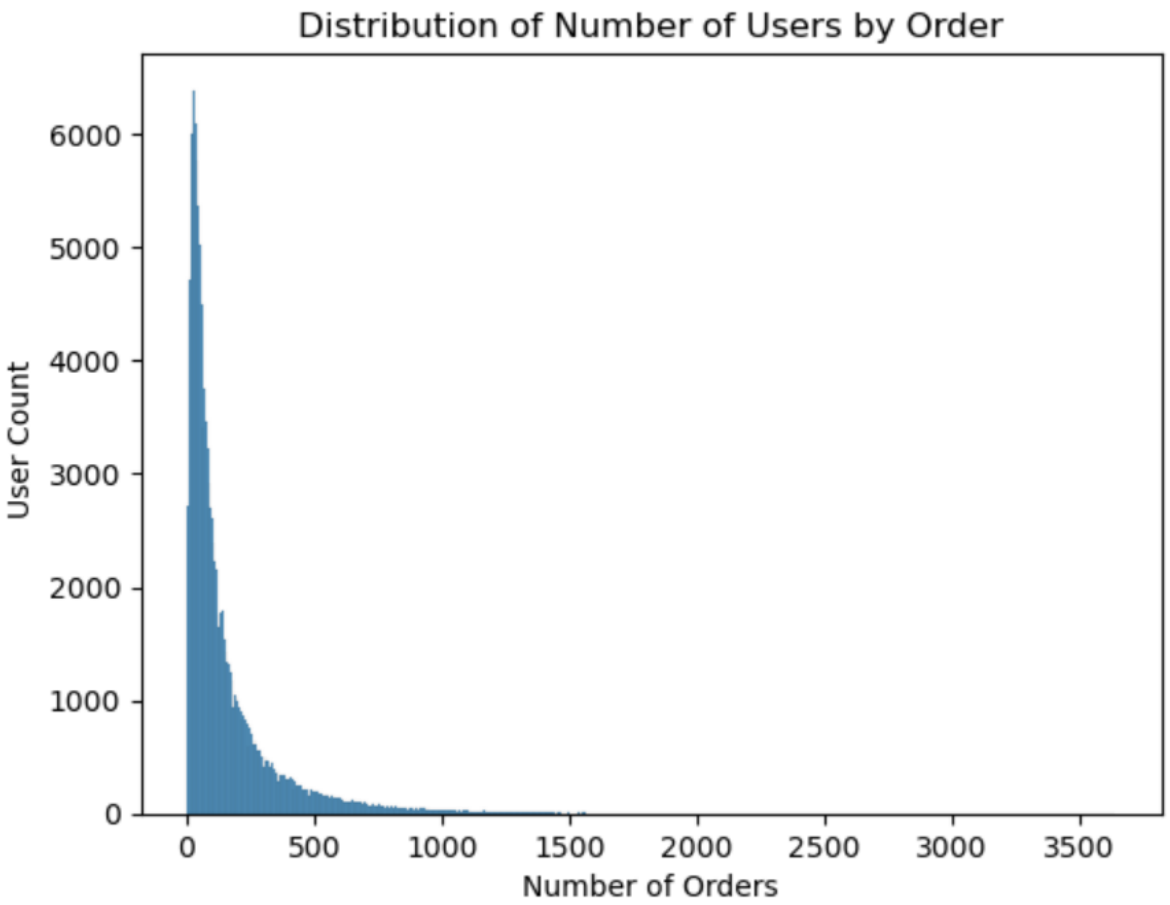
When users shop

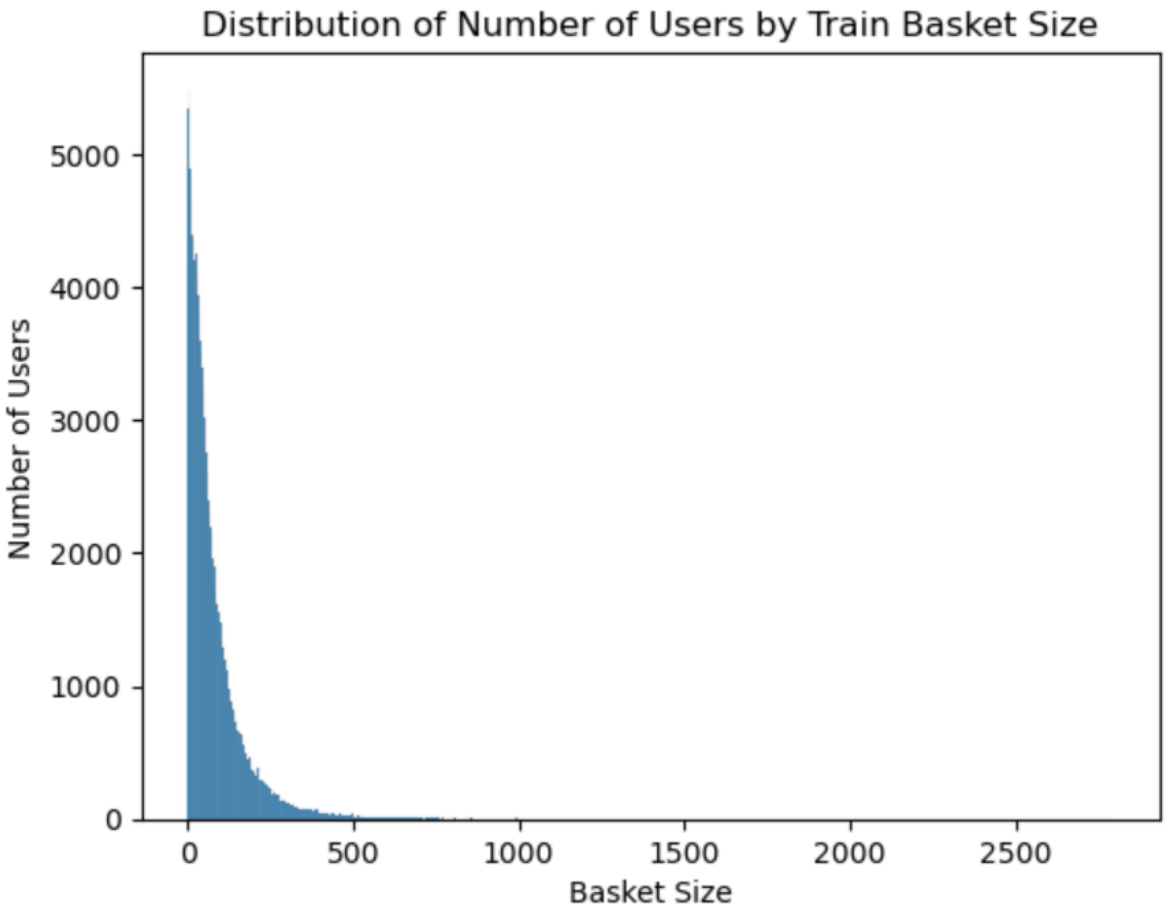




How Users Shop

Data is most likely being skewed by business accounts





Feature Engineering

Label

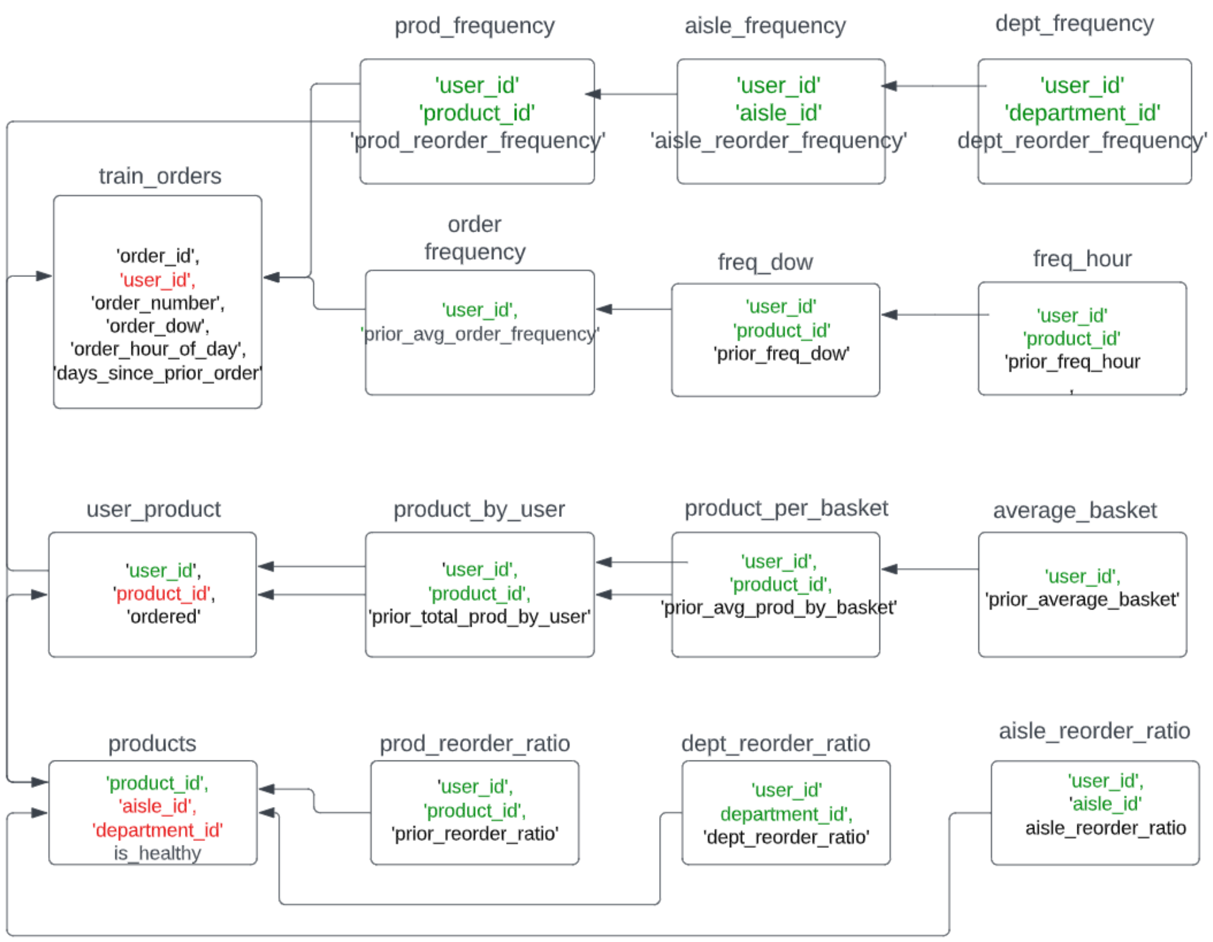
	user_id	product_id	ordered
0	1	196.0	1
1	1	14080.0	0
2	1	12424.0	0
3	1	26080.0	0
4	1	26400.0	0
...	...	...	...
8040479	206209	39232.0	0
8040480	206209	38720.0	0
8040481	206209	31472.0	0
8040482	206209	6568.0	0
8040483	206209	22912.0	0

8040484 rows × 3 columns

NLP

organic	5030
chocolate	2448
with	2236
free	2197
cheese	2095
chicken	1540
original	1452
sauce	1291
cream	1285
yogurt	1161
mix	1147
natural	1131
milk	1106
tea	1101
whole	1076

Features Schema



Final Table size:
1260057 rows × 23 columns

Models and Metrics

Models used:

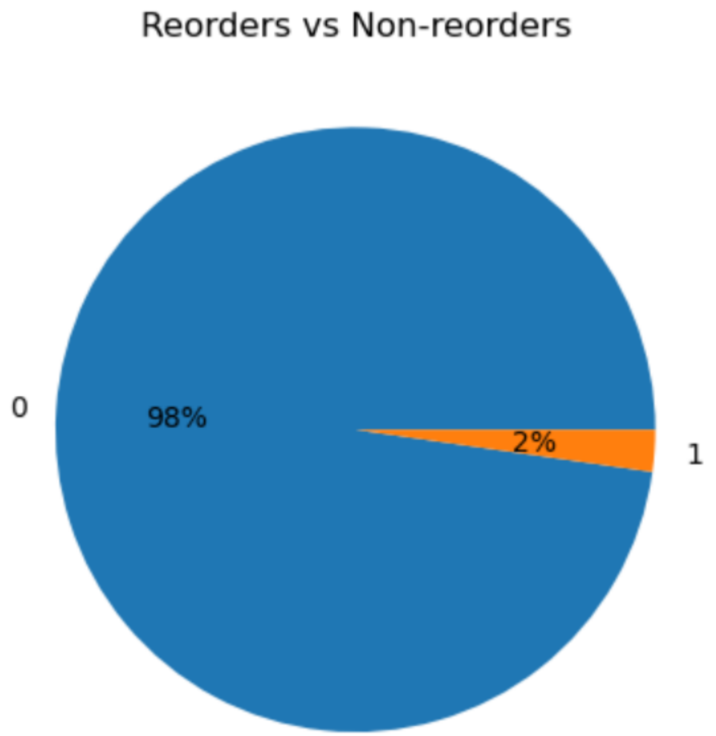
- Logistic Regression
- Decision Tree
- XGBoost
- LightGBM
- Keras Model

Evaluation metrics:

- f1 score

  - target is very unbalanced, so relying on accuracy will not work since our model will predict 0 very precisely
- ROC AUC

  - a good indicator for whether the model is able to separate classes



LightGBM

48]:
```
%%time

model = lgb.LGBMClassifier()

pipe = Pipeline([
    ('preprocessing', preprocessor),
    ('rus', RandomUnderSampler(random_state=13)),
    ('model', model)
])

param_grid = {
    'model__random_state': [13],
    'model__max_depth': [5,10,15,20],
    'model__n_estimators': [40,140,240,340],
    'model__learning_rate': [0.5,.1,.2]
    }

lgb_search = GridSearchCV(pipe, param_grid=param_grid, cv = 5, verbose = True, n_jobs = -1, scoring='f1_macro')
lgb_search.fit(X_train, y_train)

print(f'best parameters:', lgb_search.best_params_)
lgb_search.best_score_
```

```
Fitting 5 folds for each of 48 candidates, totalling 240 fits
best parameters: {'model__learning_rate': 0.2, 'model__max_depth': 5, 'model__n_estimators': 340, 'model__random_state': 13}
Wall time: 13min 40s
```

ut[548]: 0.5724775901723136

Logistic regression

```
%%time

model = LogisticRegression()

pipe = Pipeline([
    ('preprocessing', preprocessor),
    ('rus', RandomUnderSampler(random_state=13)),
    ('model', model)
])

param_grid = {
    'model__max_iter': [1000],
    'model__random_state':[13],
    'model__C':[.01,.05,.1,.5,1,5,10],
    'model__penalty':['l2']
    }

lr_search = GridSearchCV(pipe, param_grid=param_grid, cv = 5, verbose = True, n_jobs = -1, scoring='f1_macro')

lr_search.fit(X_train, y_train)

print(f'best parameters:', lr_search.best_params_)
lr_search.best_score_
```

```
Fitting 5 folds for each of 7 candidates, totalling 35 fits
best parameters: {'model__C': 0.01, 'model__max_iter': 1000, 'model__penalty': 'l2', 'model__random_state': 13}
Wall time: 48.3 s
```

0]: 0.3820393219623558

Decision Tree

]:
```
%%time

model = DecisionTreeClassifier()

pipe = Pipeline([
    ('preprocessing', preprocessor),
    ('rus', RandomUnderSampler(random_state=13)),
    ('model', model)
])

param_grid = {
    'model__random_state': [13],
    'model__max_depth': [5,10,15,20],
    'model__min_samples_split': [2,3,4,5]
}
dt_search = GridSearchCV(pipe, param_grid=param_grid, cv = 5, verbose = True, n_jobs = -1, scoring='f1_macro')
dt_search.fit(X_train, y_train)

print(f'best parameters:', dt_search.best_params_)
dt_search.best_score_
```

```
Fitting 5 folds for each of 16 candidates, totalling 80 fits
best parameters: {'model__max_depth': 10, 'model__min_samples_split': 3, 'model__random_state': 13}
Wall time: 1min 51s
```

XGBoost

```
%%time

model = XGBClassifier()

pipe = Pipeline([
    ('preprocessing', preprocessor),
    ('rus', RandomUnderSampler(random_state=13)),
    ('model', model)
])

param_grid = {
    'model__random_state': [13],
    'model__max_depth': [5,10,15,20],
    'model__n_estimators': [40,140,240,340],
    'model__eta': [.05,.1,.2]
}

xgb_search = GridSearchCV(pipe, param_grid=param_grid, cv = 5, verbose = True, n_jobs = -1, scoring='f1_macro')
xgb_search.fit(X_train, y_train)

print(f'best parameters:', xgb_search.best_params_)
xgb_search.best_score_
```

```
Fitting 5 folds for each of 48 candidates, totalling 240 fits
best parameters: {'model__eta': 0.2, 'model__max_depth': 5, 'model__n_estimators': 340, 'model__random_state': 13}
Wall time: 53min 7s
```

4]: 0.5729772010981871

Pipeline and Stacking

Pipeline

Steps:

- ColumnTransformer:
  - separated columns into nums = numeric, one_hot_cats = categories to one-hot encode and freq_cats = categories to frequency encode
  - MinMaxScaler() performed better during test runs compared to standard scaler
    - There are no huge outliers in this dataset due to it's nature, so scaling should be sufficient
  - RareLabelEncoder() needed to be used on freq_cats before CountEncoder due to the CountEncoder() creating NaN values
- RandomUnderSampler()
  - performed better during test runs compared to over samling
- Model - variable to be reset

```
nums = ['order_number','days_since_prior_order', 'order_size', 'prior_total_prod_by_user',
       'prior_avg_prod_by_basket', 'prior_average_basket', 'prior_avg_order_frequency',
       'product_reorder_ratio', 'dept_reorder_ratio', 'aisle_reorder_ratio',
       'prod_reorder_frequency', 'aisle_reorder_frequency', 'dept_reorder_frequency']

one_hot_cats = ['order_dow', 'order_hour_of_day', 'prior_freq_dow', 'prior_freq_hour', 'is_healthy']

freq_cats = ['user_id', 'product_id', 'aisle_id', 'department_id']
```

```
transformers = [('scaler', MinMaxScaler(), nums),
                ('oh_encoding', OneHotEncoder(drop='first'), one_hot_cats),
                ('rare_encoding', RareLabelEncoder(replace_with=0), freq_cats),
                ('freq_encoding', CountEncoder(), freq_cats)]


preprocessor = ColumnTransformer(transformers=transformers, remainder='drop')
```

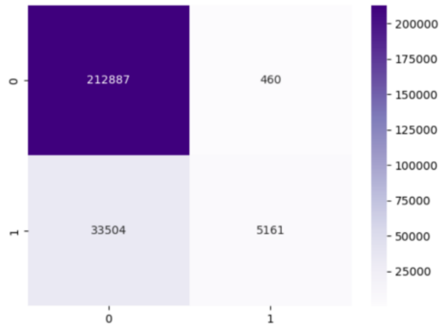
```
pipe = Pipeline([
    ('preprocessing', preprocessor),
    ('rus', RandomUnderSampler(random_state=13)),
    ('model', model)
])
```

Stacking

Previously fitted model with the best parameters were stacked to check if they would improve the overall performance.

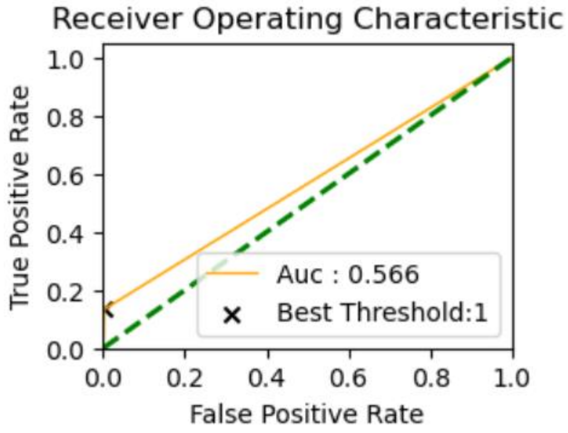
Unfortunately, there was not improvement observed neither in f1 scoring, nor the AUC. The model is unable to effectively differentiate between the classes, no matter which final estimator was used

```
%%time

estimators = [
    ('lr',  lr.named_steps['model']),
    ('dt',  dt.named_steps['model']),
    ('lgb', lgb_model.named_steps['model']),
    ('xgb', xgb.named_steps['model'])

]
stacked_pipe = Pipeline([
    ('preprocessing', preprocessor),
    ('rus', RandomUnderSampler(random_state=13)),
    ('clf', StackingClassifier(estimators=estimators, final_estimator=lgb.LGBMClassifier()))
])

stacked_pipe.fit(X_train, y_train)

y_pred_val = stacked_pipe.predict(X_val)

reports(y_pred_val, y_val)
```



	precision	recall	f1-score	support
0	0.86	1.00	0.93	213347
1	0.92	0.13	0.23	38665
accuracy			0.87	252012
macro avg	0.89	0.57	0.58	252012
weighted avg	0.87	0.87	0.82	252012

Wall time: 3min 4s



Neural Networks – MLP Sequential

Model Training and validation

```python
np.random.seed(13)
tf.random.set_seed(13)

mlp = Sequential()

mlp.add(InputLayer(input_shape=(76, )))
mlp.add(Dense(300, activation='relu'))
mlp.add(Dense(150, activation='relu'))

mlp.add(Dense(2, activation='sigmoid'))
```

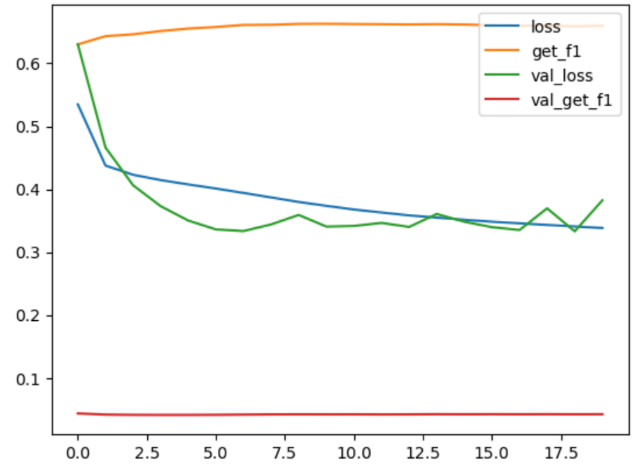
```python
mlp.summary()
```

```
Model: "sequential_75"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_59 (Dense)            (None, 300)               23100

 dense_60 (Dense)            (None, 150)               45150

 dense_61 (Dense)            (None, 2)                 302

=================================================================
Total params: 68,552
Trainable params: 68,552
Non-trainable params: 0
_________________________________________________________________
```

```python
mlp.compile(loss='sparse_categorical_crossentropy', optimizer='sgd', metrics=[get_f1])
```

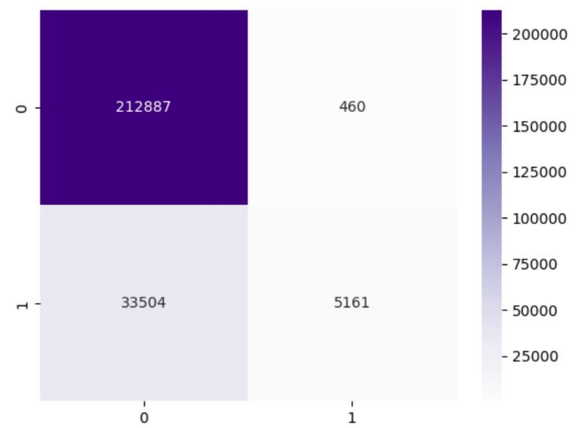
```python
%%time
history = mlp.fit(X_resampled, y_resampled, epochs=20, validation_data=(X_val_scaled, y_val))
```

```
Epoch 1/20
1054/1054 [==============================] - 28s 26ms/step - loss: 0.5347 - get_f1: 0.6301 - val_loss: 0.6300 - val_get_f1:
0.0436
Epoch 2/20
1054/1054 [==============================] - 29s 28ms/step - loss: 0.4376 - get_f1: 0.6432 - val_loss: 0.4663 - val_get_f1:
0.0418
Epoch 3/20
1054/1054 [==============================] - 29s 28ms/step - loss: 0.4229 - get_f1: 0.6462 - val_loss: 0.4063 - val_get_f1:
0.0414
Epoch 4/20
1054/1054 [==============================] - 29s 27ms/step - loss: 0.4145 - get_f1: 0.6513 - val_loss: 0.3730 - val_get_f1:
0.0413
Epoch 5/20
1054/1054 [==============================] - 29s 27ms/step - loss: 0.4076 - get_f1: 0.6553 - val_loss: 0.3501 - val_get_f1:
0.0413
Epoch 6/20
1054/1054 [==============================] - 29s 28ms/step - loss: 0.4011 - get_f1: 0.6578 - val_loss: 0.3361 - val_get_f1:
0.0414
Epoch 7/20
1054/1054 [==============================] - 29s 28ms/step - loss: 0.3941 - get_f1: 0.6610 - val_loss: 0.3335 - val_get_f1:
0.0418
Epoch 8/20
1054/1054 [==============================] - 29s 27ms/step - loss: 0.3869 - get_f1: 0.6613 - val_loss: 0.3440 - val_get_f1:
0.0420
Epoch 9/20
1054/1054 [==============================] - 30s 29ms/step - loss: 0.3797 - get_f1: 0.6627 - val_loss: 0.3591 - val_get_f1:
0.0421
Epoch 10/20
1054/1054 [==============================] - 30s 29ms/step - loss: 0.3736 - get_f1: 0.6629 - val_loss: 0.3406 - val_get_f1:
0.0421
Epoch 11/20
1054/1054 [==============================] - 29s 28ms/step - loss: 0.3677 - get_f1: 0.6625 - val_loss: 0.3416 - val_get_f1:
0.0421
Epoch 12/20
1054/1054 [==============================] - 29s 27ms/step - loss: 0.3630 - get_f1: 0.6622 - val_loss: 0.3464 - val_get_f1:
0.0420
Epoch 13/20
```

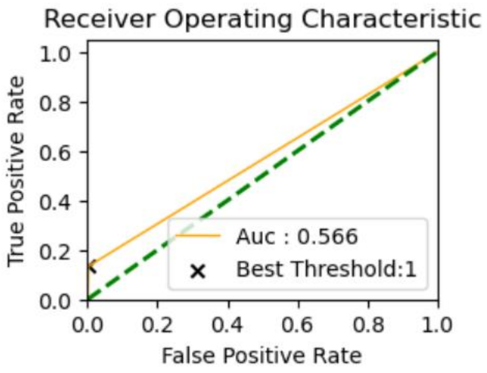


Model Evaluation

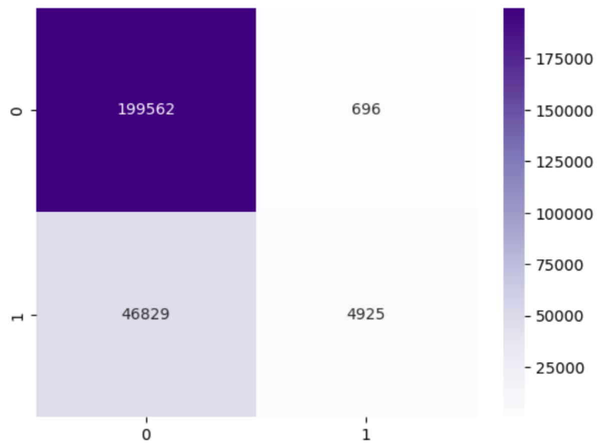
Stacked Model



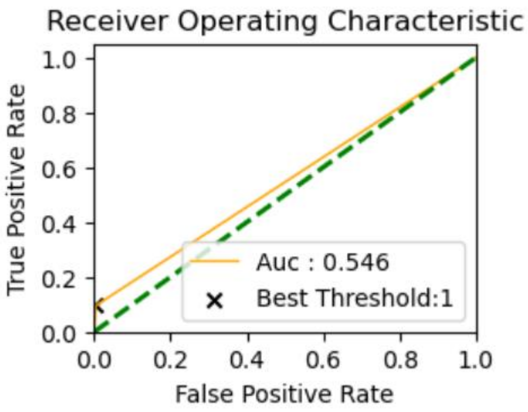

```
              precision    recall  f1-score   support

           0       0.86      1.00      0.93    213347
           1       0.92      0.13      0.23     38665

    accuracy                           0.87    252012
   macro avg       0.89      0.57      0.58    252012
weighted avg       0.87      0.87      0.82    252012
```

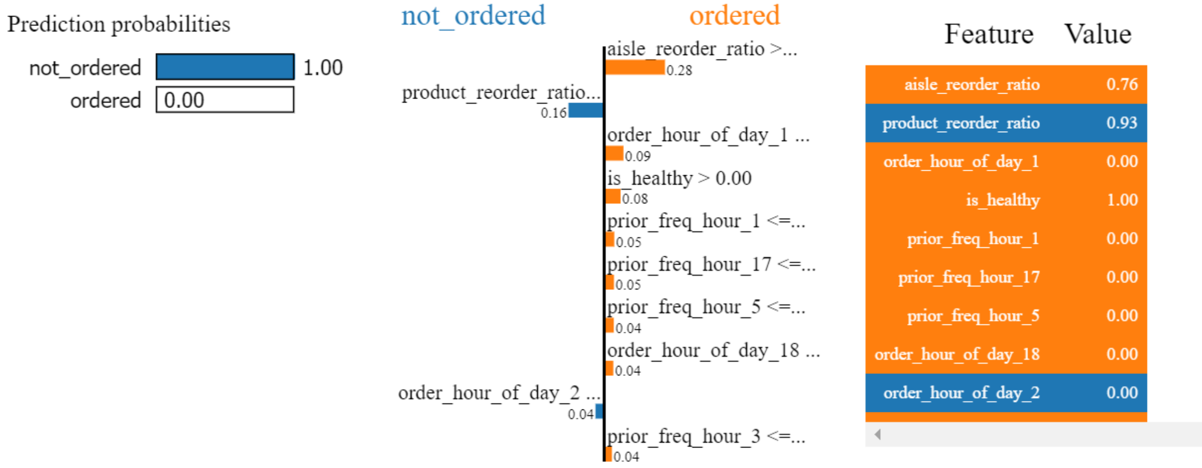


Keras Model




```
              precision    recall  f1-score   support

           0       0.81      1.00      0.89    200258
           1       0.88      0.10      0.17     51754

    accuracy                           0.81    252012
   macro avg       0.84      0.55      0.53    252012
weighted avg       0.82      0.81      0.75    252012
```





Challenges

- Unfamiliar problem
- A large chunk of time was spent on data preparation and analysis
- Highly imbalanced classes
- Difficult to interpret findings when using blackbox models
- Challenging to navigate a pipeline
- Models did not end up performing as well as intended and the scores were way below the average of 0.35 f1 score.

Next Steps

Some things that can improve the model performance:

- adding more/better features will definitely help
- trying different types of encoders
- trying other ML algorithms that people have used: RandomForest, Catboost
- Try a wider scope in terms of hyperparameter tuning, trying wider range of parameters and re-running the SearchGrid to optimize
- Association rules search with Apriori Algorithm