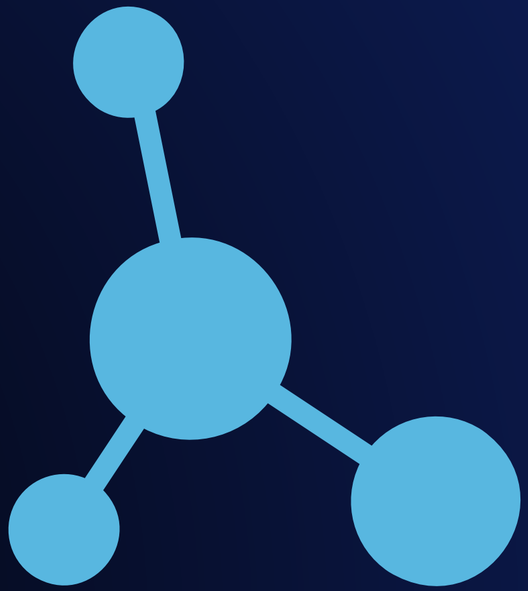
FORECASTING PRODUCT REORDERS IN E-COMMERCE

ML Project

Karina Oborska-Balkowiec

TABLE OF CONTENT



1 INTRODUCTION

2 DATA OVERVIEW

3 FEATURE ENGINEERING

4 MODEL DEVELOPMENT

5 CONSLUSIONS & NEXT STEPS

BUSINESS PROBLEM DESCRIPTION

INSTACART'S SHOPPING OPTIMIZATION

- Instacart, is an American delivery company based in San Francisco that operates a grocery delivery and pick-up service in the United States and Canada accessible via a website and mobile app.
- By analyzing customer purchase habits, Instacart aims to predict which items a customer will buy again or try for the first time.
- The data science team at Instacart uses transactional data to build models that predict future buying behaviors. These predictions help in providing personalized product recommendations and efficient inventory management.

BENEFITS OF PREDICTING PRODUCTS REORDER

- Inventory management
- Improved customer satisfaction
- Efficient resource allocation
- Data-driven insights

PROJECT AIM

The main aim of the project is to design and implement a classification model capable of forecasting class probabilities. Steps needed to achieve it:

- DATA PREPARATION
- EXPLORATORY DATA ANALYSIS
- FEATURE ENGINEERING
- MODEL SELECTION AND OPTIMIZATION
- DEVELOPMENT OF SELECTED MODEL

Expected outcome is to build a model that can accurately predict when products need to be reordered. This will help the company manage inventory better and ensure that products are always available for customers, improving their shopping experience.

DATASET & KEY CHALLENGES

DATASET INFORMATION

- Instacart Market Basket Analysis dataset is a part of a competition made by kaggle 7 years ago.
- The dataset consists of 20,641,991 records across 15 variables, detailing prior orders.
- These are sequence data and time is reflected by order number for each client.
- Each observation reflects product which was added to the cart of particular order of particular client.
- There are 131 209 clients with a number of orders in range from 3 to 99 and each order consits of up to 145 products maximum.

KEY CHALLENGES

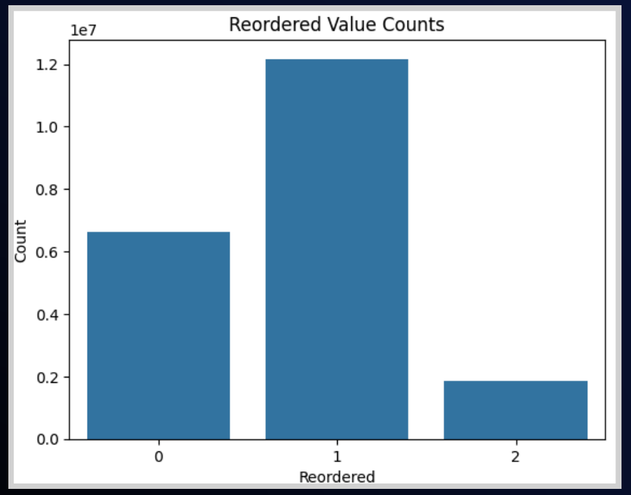
- Sequence data
- Risk of data leakage
- Large amount of data
- Multiclass task with high imbalance in data.

```
df_train.head(15)
```

	order_id	product_id	add_to_cart_order	reordered	product_name	aisle_id	department_id	aisle	department	user_id	eval_set	order_number	order_dow	order_hour_of_day	days_since_prior_order
0	2	33120	1	1	Organic Egg Whites	86	16	eggs	dairy eggs	202279	prior	3	5	9	8.0
1	2	28985	2	1	Michigan Organic Kale	83	4	fresh vegetables	produce	202279	prior	3	5	9	8.0
2	2	9327	3	0	Garlic Powder	104	13	spices seasonings	pantry	202279	prior	3	5	9	8.0
3	2	45918	4	1	Coconut Butter	19	13	oils vinegars	pantry	202279	prior	3	5	9	8.0
4	2	30035	5	0	Natural Sweetener	17	13	baking ingredients	pantry	202279	prior	3	5	9	8.0
5	2	17794	6	1	Carrots	83	4	fresh vegetables	produce	202279	prior	3	5	9	8.0
6	2	40141	7	1	Original Unflavored Gelatine Mix	105	13	doughs gelatins bake mixes	pantry	202279	prior	3	5	9	8.0
7	2	1819	8	1	All Natural No Stir Creamy Almond Butter	88	13	spreads	pantry	202279	prior	3	5	9	8.0
8	2	43668	9	0	Classic Blend Cole Slaw	123	4	packaged vegetables fruits	produce	202279	prior	3	5	9	8.0
9	3	33754	1	1	Total 2% with Strawberry Lowfat Greek Strained...	120	16	yogurt	dairy eggs	205970	prior	16	5	17	12.0
10	3	24838	2	1	Unsweetened Almondmilk	91	16	soy lactosefree	dairy eggs	205970	prior	16	5	17	12.0
11	3	17704	3	1	Lemons	123	4	packaged vegetables	produce	205970	prior	16	5	17	12.0

INSIGHTS ABOUT DATASET

While the original task was framed as a binary classification, analysis revealed a distinct third class of orders that contain exclusively non-reordered products. This discovery led to expanding the classification to a multiclass problem, introducing substantial imbalance with approximately 132k initial orders and around 116k orders placed among subsequent ones, complicating the model's ability to generalize across the different classes.



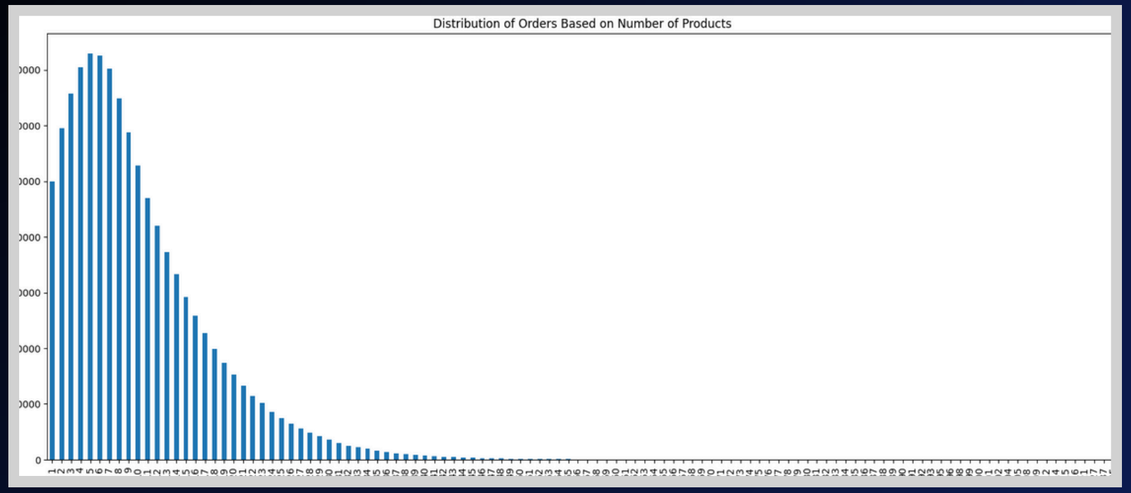
Some customers placed all their orders on the same day, totaling 16 users with 199 records all. These were considered outliers due to their unusual ordering pattern and were subsequently removed from the dataset.

```
Average number of orders for typical clients: 15.605405776222817
Average number of orders for clients placing many orders same day: 3.5625

Average number of products for typical clients: 157.33912632533747
Average number of products for clients placing many orders same day: 12.4375

Ratio of clients placing orders same day vs. typical clients: 0.0001
```

The data often shows a right-skewed distribution, which became more apparent after developing additional features. Although some data points initially seemed like outliers, they actually depict typical purchasing behaviors of regular clients.



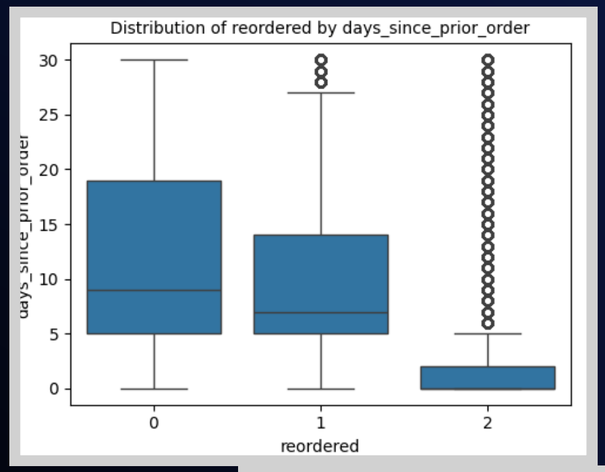
INSIGHTS CONNECTED WITH DISTRIBUTION OF DATA

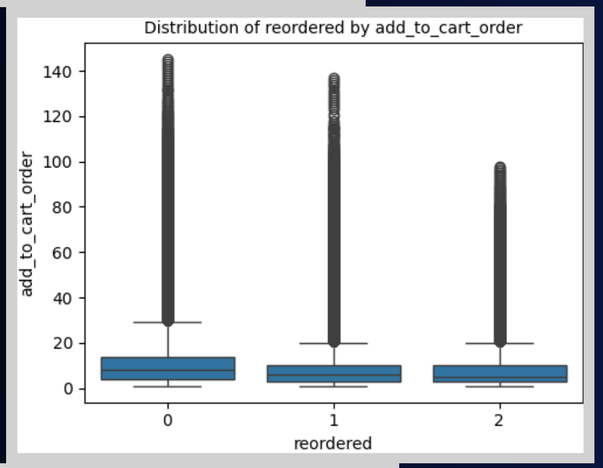
- Non-reordered products often appear toward the end of a shopping session, suggesting they may be less essential or impulse buys.

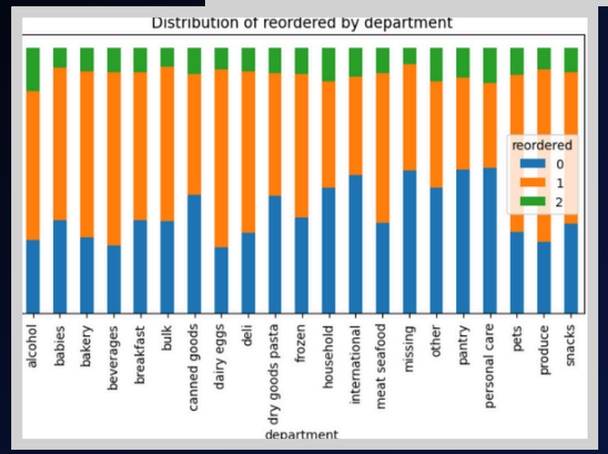
- The number of days since the last order helps differentiate the classes distinctly. Products that are not reordered tend to have shorter gaps since their last purchase, indicating that these may be less critical or spontaneous buys. In contrast, products with longer intervals between purchases are less frequently reordered, suggesting they might not be regular necessities.

- Variables that didn't provide too much different patterns between classes were day of week and hour of the day.

- There is a significant correlation between the type of product, its aisle, or department and the likelihood of reordering. This relationship can guide targeted inventory strategies, allowing for more precise stocking and marketing efforts that cater to the most commonly reordered items, enhancing overall efficiency and customer satisfaction.







FEATURE ENGINEERING - PHASE 1

In the first phase I decided to create features based on 3 types of functions as below.

USER BEHAVIOUR FEATURES

These features are related to the client's level, not to individual products. Examples are:

- Total number of orders.
- Average number of products per order.
- Average time between orders.
- If product was added warly till 25 percentile.

PRODUCT TYPE FEATRES

These are features connected with the type of a product, aisle and department. This function ranks each type of product, aisle, and department based on frequency of appearance for each client.

PRODUCT USAGE FEATURES

These are features connected with information on product's level. Examples are:

- Number of days when product was last ordered.
- Average days between product's orders.
- Average position in card for user - product.
- Ratio of times a product was reordered compared to total orders.

CONCLUSIONS AFTER THIS PHASE:

- Some of them differentiate classes very well what delivered more insights about existing patterns.
- I realized due to data leakage I will not be able to use all preliminary given variables like for example add to cart order but also features like number of days product was last time ordered were useless, so I needed to create more general features bsed on that.
- Categorical features also were less useful as for testing set I could only use features counted based on training set.

FEATURE ENGINEERING - PHASE 2

Focused on creating general features for testing applicability and reformulated binary counts into ratios and averages.

BINARY VARIABLES

Transformed into a ratio and more general values variables like consecutive reorders or added early.

TIME BETWEEN ORDERS

Developed into features that compare order timing variations to mode times.

PRODUCT CART POSITION

Revised to an average placement metrics, enhancing its predictive value.

ORDER-BASED FEATURES

FAdjusted to reflect relative order placement in sequences, addressing dataset splits.

TEMPORAL FEATURES

Converted day and hour data into categorical percentages, reflecting shopping patterns.

CLASS-SPECIFIC FEATURES

Enhanced features for class 2 which is underrepresented to better capture their unique behaviors.

SUMMARY

- Some of the created variables helped to identify unique patterns for each class, but their actual impact will be evaluated during the modeling phase.
- After combining features from the training set, it's noticed about 562k missing user-product pairs between the training and test sets. They will be filled with zeros at the product level.
- The only feature that didn't align with the training set was the 'order position,' which has a value of 1 assigned to indicate the most recent order.
- At this level variables which were used to count new features and those that can provide data leakage were deleted.

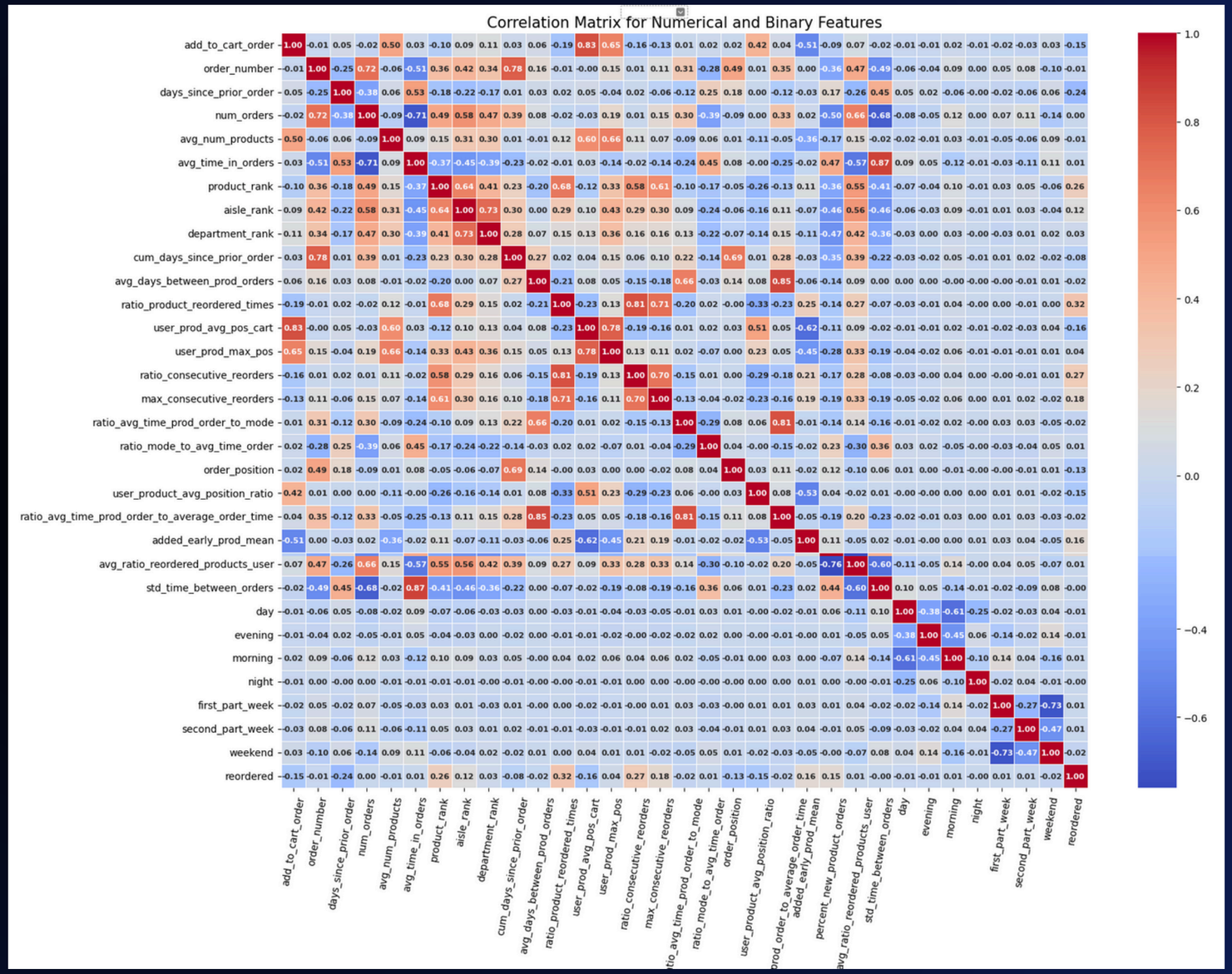
CORRELATION MATRIX

- Correlation matrix is not the best tool to in case of checking correlation of variables with multi-class target. However is still valuable for identifying highly correlated variables.

- For now, the focus is on removing variables that pose a risk of data leakage. Remaining variables will be retained for further evaluation of their utility after initial model deployment.

- add_to_cart_order and user_prod_avg_pos_cart, suggesting redundancy. To prevent data leakage, add_to_cart_order will be removed

- order_number shows a strong correlation with both cum_days_since_prior_order and num_orders, indicating it may carry redundant information and will also be removed to reduce complexity. Especially it was replaced by order_position.



SELECTED ALGORITHMS

LOGISTIC REGRESSION

Even if it is not the best choice for this problem as it assumes a linear relationship between the features it was used to establish baseline.

RANDOM FOREST

Can handle non-linearity and features interactions than logistic regression. It is less sensitive to feature scaling.

XGBOOST

It has got capability to handle large and complex patterns. With its gradient boosting mechanism can model both linear and non-linear relationships.

CATBOOST

It has built-in mechanism to handle categorical data, however not possible to use in this case due to large number of unique values.

GRU

The type of neural network that handles sequential data well. It seems to be a great choice having sequence data. Comparing to LSTM it should be faster without achieving lower results.

PREPROCESSING METHODS

- One-Hot encoding
- James Stein encoding
- Standard Scaler

- One-Hot encoding
- James Stein encoding

- Target encoding
- James Stein encoding

- James Stein encoding

- Min Max Scaler
- Embeddings

SUMMARY

- Standard Scaler was used for logistic regression to normalize feature scales, enhancing model accuracy. For GRU, Min-Max Scaler was applied to keep input values between 0 and 1, which is optimal for neural network training.
- After Random Forest model one-hot encoded variables were deleted as they didn't have any influence on performance according to feature importance.
- For encoding was mainly used James Stein method as it works well with many single appeared values. In XGBoost target encoding was used for department and aisle (they have less unique values), but these variables weren't effective at all.

GATED RECURRENT UNIT

- Promising potential due to ability of handling sequential data.
- Min-Max scaling & Embeddings as appropriate methods for preprocessing.
- A special data generator was crucial due to large dataset.
- The model's performance was limited by the data structure, which did not fully reflect the sequential nature of orders. Because of this, the model was deployed on 20k samples to check baseline performance.
- Further adjustments to the desired structure (4-dimensional data, sequence based on orders, and predictions at the product level) would require significant computational power.

```python
def create_sequences(df, batch_size=20, max_products=145):
    """
    Function generates batches in appropriate shape with padding.
    Args: df, batch size and expected maximum number of products
    Returns: batch in shape (batch size = number of orders in batch, number of products, features)
    """
    # optimizing data types to improve performance
    df['user_id'] = df['user_id'].astype(np.int32)
    df['order_id'] = df['order_id'].astype(np.int32)
    df['reordered'] = df['reordered'].astype(np.int8)

    # sorting data to hold them in sequence
    df = df.sort_values(by=['user_id', 'order_position']).reset_index(drop=True)

    # grouping by user and order
    grouped = df.groupby(['user_id', 'order_id'])

    current_batch_products, current_batch_targets = [], []

    for (user_id, order_position), group in grouped:
        # converting products with its features and target column to numpy as it is more effective than using lists
        products = group.drop(columns=['user_id', 'order_id', 'reordered']).to_numpy(dtype=np.float32)
        targets = group['reordered'].to_numpy(dtype=np.int8)

        # padding for supplementing number of products to equal for all orders as it is expected by GRU
        padded_products = np.zeros((max_products, products.shape[1]), dtype=np.float32)
        padded_targets = np.zeros(max_products, dtype=np.int8)

        # filling with existing products
        length = min(len(products), max_products)
        padded_products[:length] = products[:length]
        padded_targets[:length] = targets[:length]

        current_batch_products.append(padded_products)
        current_batch_targets.append(padded_targets)

        # in case the batch is full returning batches
        if len(current_batch_products) >= batch_size:
            yield np.stack(current_batch_products, dtype=np.float32), np.expand_dims(np.stack(current_batch_targets, dtype=np.int8), axis=-1)
            current_batch_products, current_batch_targets = [], []

    # return of the last batch
    if current_batch_products:
        yield np.stack(current_batch_products, dtype=np.float32), np.expand_dims(np.stack(current_batch_targets, dtype=np.int8), axis=-1)
```

```python
# GRU parameters
max_products = 145
feature_size = 58

# input layer: (batch_size = orders, max_products, feature_size)
inputs = Input(shape=(max_products, feature_size))

# masking of null values
masked = Masking(mask_value=0.0)(inputs)

# prforming GRU for orders taking into consideration single products
x = GRU(128, return_sequences=True, dropout = 0.2)(masked)

# output layer for each product's predictions and for 3 classes problem
outputs = TimeDistributed(Dense(3, activation='softmax'))(x)

# creating model
model = Model(inputs, outputs)
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=[CategoricalAccuracy()])

model.summary()
```

Model: "functional"

Layer (type)	Output Shape	Param #	Connected to
input_layer (InputLayer)	(None, 145, 58)	0	-
not_equal (NotEqual)	(None, 145, 58)	0	input_layer[0][0]
masking (Masking)	(None, 145, 58)	0	input_layer[0][0]
any (Any)	(None, 145)	0	not_equal[0][0]
gru (GRU)	(None, 145, 128)	72,192	masking[0][0], any[0][0]
time_distributed (TimeDistributed)	(None, 145, 3)	387	gru[0][0], any[0][0]

Total params: 72,579 (283.51 KB)
Trainable params: 72,579 (283.51 KB)
Non-trainable params: 0 (0.00 B)

COMPARISON OF BASE RESULTS FOR ALL MODELS

I focused on F1-score because the dataset is imbalanced, and accuracy alone would not properly reflect the model's performance.

	LOGISTIC REGRESSION	RANDOM FOREST	XGBOOST	CATBOOST	GRU
Class 0	0,14	0,70	0,77	0,74	0,72
Class 1	0,47	0,78	0,82	0,81	0,75
Class 2	0,09	0,16	0,26	0,25	0,15
Macro avg	0,23	0,55	0,615	0,60	0,54

SUMMARY

- Logistic Regression and GRU were not used on the entire dataset due to its complexity and the extensive computational resources required, respectively.
- All models struggled the most with predicting the underrepresented second class, highlighting a common challenge across the board.
- XGBoost achieved the best results, which were further optimized in subsequent steps.
- Although GRU shows potential, its development is hindered by significant time and computational demands.

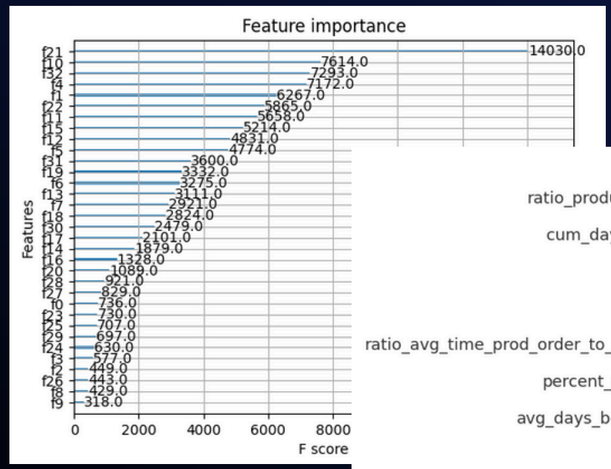
XGBOOST DEVELOPMENT

- Based on feature importance and SHAP analysis some variables connected with department, aisle, day and hour were excluded due to lack of influence on model - the result remains almost the same.

- In SHAP analysis we can observe that for classes 0 and 1 the most important feature was ratio how many times product was reordered vs. all orders. Whereas for class 2 the biggest influence had how many orders with new products client did comparing to all orders.

- Hyperparameters tunning helped to improve macro avg. F1 score from 0,6151 to 0,6157 - it's not sucha big difference, but maybe further adjusting parameters can bring more improvement.

- The last optimization that was made was probability calibration. However it didn't significantly improve the result (macro avg. f1 score was 0,616), so it would be better to focus on finding separate solution for 2 class.




```
[ ] # parameters for testing
    param_grid = {
        'n_estimators': [200, 300],
        'max_depth': [6, 8],
        'learning_rate': [0.05, 0.1],
        'colsample_bytree': [0.8, 1.0],
        'reg_alpha': [0, 1]
    }

    grid = ParameterGrid(param_grid)

    best_score = 0
    best_params = {}

    # testing different combinations of parameters
    for params in grid:

        model = xgb.XGBClassifier(
            objective='multi:softmax', eval_metric='mlogloss', random_state=42, **params
```

```
[ ] # extracting probabilities for all classes
    y_probs = xgb_model.predict_proba(X_test_transformed_new)

    # setting up tresholds based on number assigned to class vs real number of samples
    optimal_threshold_0 = 0.7
    optimal_threshold_1 = 0.3
    optimal_threshold_2 = 0.5

    # Classification based on new tresholds
    y_probs[:, 0] = np.where(y_probs[:, 0] >= optimal_threshold_0, y_probs[:, 0], 0)
    y_probs[:, 1] = np.where(y_probs[:, 1] >= optimal_threshold_1, y_probs[:, 1], 0)
    y_probs[:, 2] = np.where(y_probs[:, 2] >= optimal_threshold_2, y_probs[:, 2], 0)

    y_pred_adjusted = np.argmax(y_probs, axis=1)

    print(classification_report(y_test_xgb_new, y_pred_adjusted))
```

CONCLUSIONS & NEXT STEPS

CONCLUSIONS

- XGBoost delivered the best results, driven by focused improvements in hyperparameter tuning, feature selection, and class imbalance management.
- High F1-scores for Class 0 and 1 (0.77 and 0.82) show the model's strong ability in balancing precision and recall. Class 2 (F1-score: 0.26) highlights the challenges with underrepresented classes.
- By accurately predicting reorder times for products (particularly for Class 0 and Class 1 where F1-scores are high), the model can help businesses avoid running out of stock. This directly contributes to avoiding potential sales losses and maintaining customer satisfaction.

NEXT STEPS

- Investigate whether a Gated Recurrent Unit (GRU) model could surpass XGBoost's performance, considering its potential in handling sequential data.
- Continue refining XGBoost through advanced feature engineering and hyperparameter adjustments. Consider developing a tailored model for the problematic Class 2.
- Plan to validate the optimized model on the untouched test set after improvements, ensuring robustness and generalizability.
- Manage inherent project challenges such as large data volume.
- The underperformance in predicting Class 2 (low-reorder likelihood products) suggests a potential area for further development. Improving predictions for this class could help in accurately managing products that are crucial but infrequently ordered, which could be pivotal during seasonal peaks or promotions.


Thank you!