Semester Project - Data Science

Necessary modules and data sources

In [ ]:

```
# data manipulation
import pandas as pd

# XGBoost model
import xgboost as xgb

# data visualization
import seaborn as sns
import matplotlib as plt
from IPython.display import display, Image

# data preprocessing
from sklearn.preprocessing import LabelEncoder

# Hyperparameter tuning
from sklearn.model_selection import RandomizedSearchCV, cross_val_score
from sklearn.model_selection import GridSearchCV

# model training and testing
from sklearn.model_selection import train_test_split

# model evaluation
from sklearn.metrics import accuracy_score, classification_report
```

In [ ]:

```
op = pd.read_csv('Instacart/order_products__train.csv')
opp = pd.read_csv('Instacart/order_products__prior.csv')
aisles = pd.read_csv('Instacart/aisles.csv')
departments = pd.read_csv('Instacart/departments.csv')
orders = pd.read_csv('Instacart/orders.csv')
products = pd.read_csv('Instacart/products.csv')
```

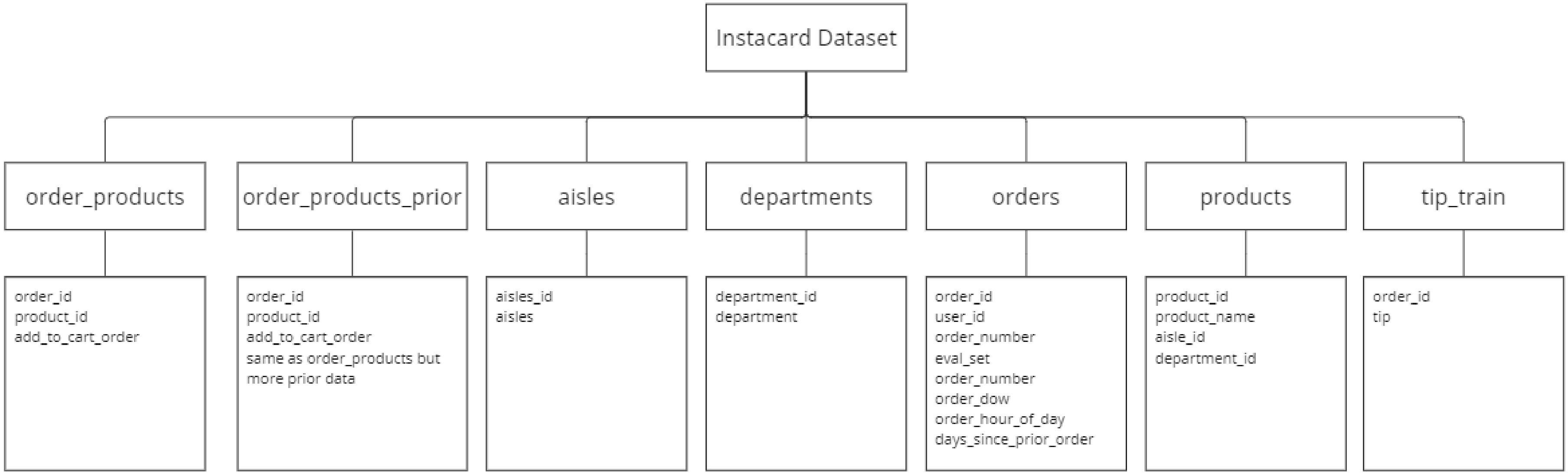
```
tip_train = pd.read_csv('Instacart/tip_trainingsdaten1_.csv',index_col=0)
tip_test = pd.read_csv('Instacart/tip_testdaten1_template.csv',index_col=0)
```

Looking into the data

- First we should get an overview of the overall data placement and which keys connect which tables

Let´s look at the given tables

```
In [ ]: display(Image(filename='images/instacard.png'))
```



- We can see that for order, we can use the key order_id to reference more information from other tables. The table order_products provides a possibility to retrieve information based on both the orders and products
- Furthermore, we can then use product_id to also find information about what aisles and departments match certain products
- The table order_products_prior allows us to use the comparative information of this and the orders table

Data Analysis

- We now take a closer look at the underlying data structure and see if we can find important information and correlations
- We start by analysing the orders table

```
In [ ]: orders
```

Out[ ]:

	order_id	user_id	eval_set	order_number	order_dow	order_hour_of_day	days_since_prior_order
0	2539329	1	prior	1	2	8	NaN
1	2398795	1	prior	2	3	7	15.0
2	473747	1	prior	3	3	12	21.0
3	2254736	1	prior	4	4	7	29.0
4	431534	1	prior	5	4	15	28.0
...	...	...	...	...	...	...	...
3421078	2266710	206209	prior	10	5	18	29.0
3421079	1854736	206209	prior	11	4	10	30.0
3421080	626363	206209	prior	12	1	12	18.0
3421081	2977660	206209	prior	13	1	12	7.0
3421082	272231	206209	train	14	6	14	30.0

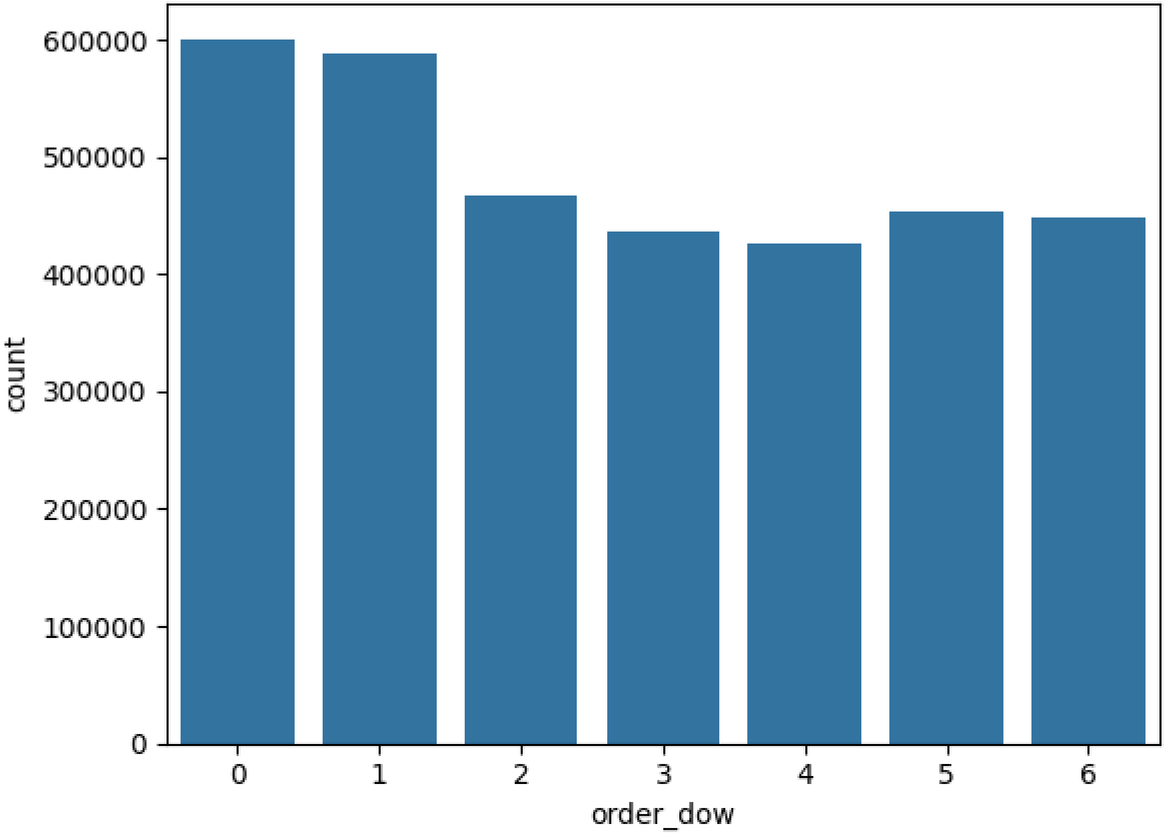
3421083 rows × 7 columns

Categorical Data

- Let us analyse the categorical data

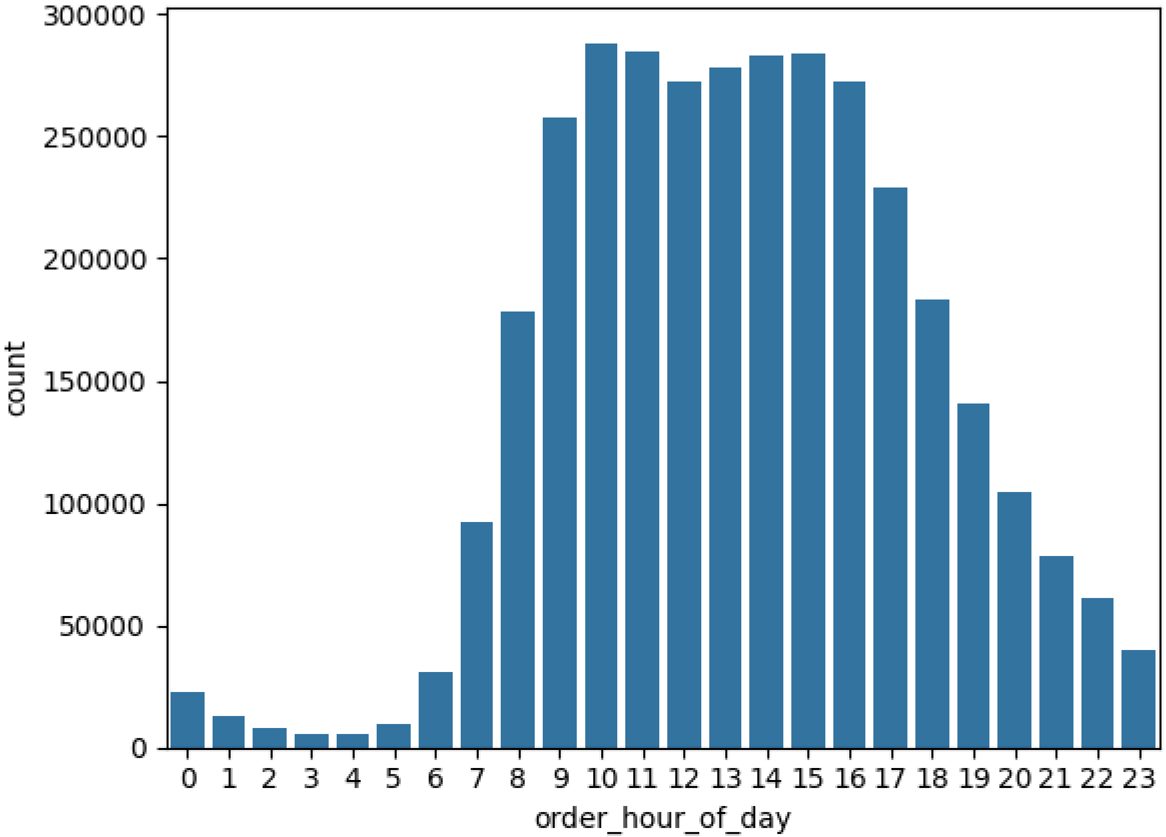
```
In [ ]: # day of week is categorical
        sns.countplot(data=orders,x='order_dow')
```

```
Out[ ]: <Axes: xlabel='order_dow', ylabel='count'>
```



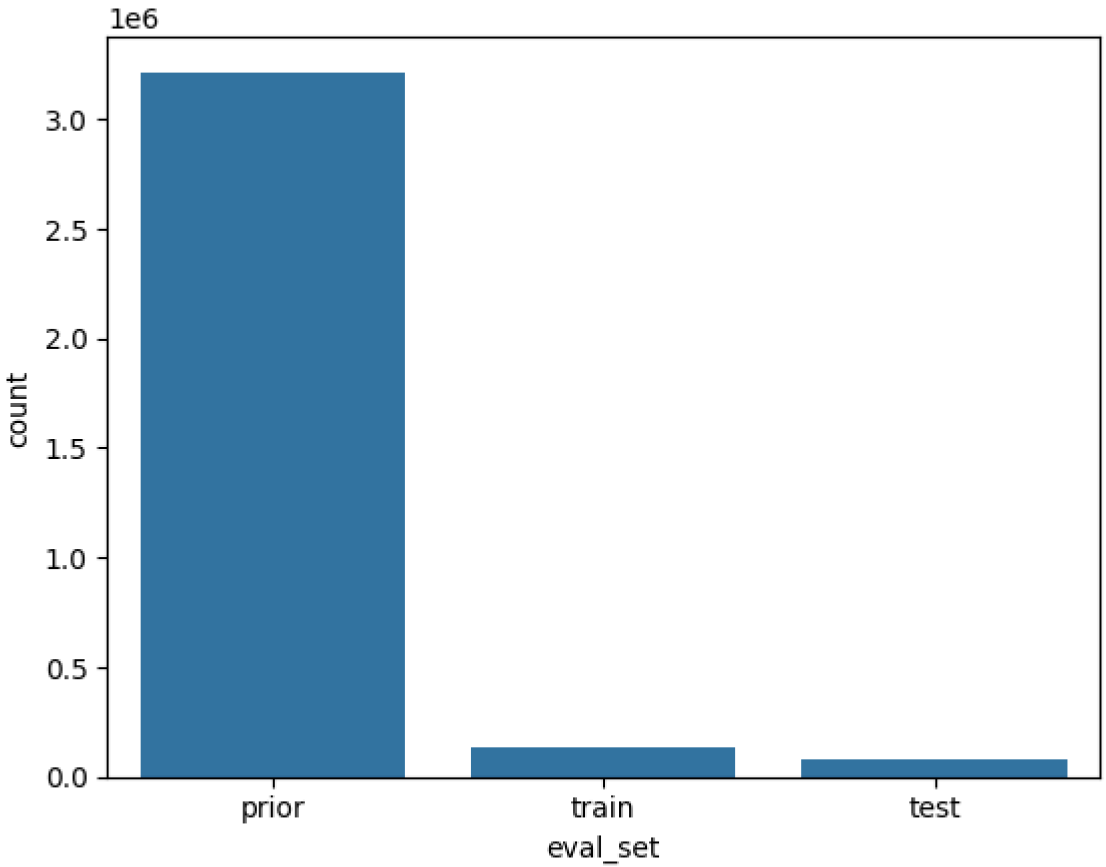
```
In [ ]: # Let us also take a look at "order hour of day" since it is another categorical value
        sns.countplot(data=orders,x='order_hour_of_day')
        # As we can see most orders are placed during the hours 8am to around 5pm, which makes sense
```

```
Out[ ]: <Axes: xlabel='order_hour_of_day', ylabel='count'>
```



```
In [ ]: sns.countplot(data = orders, x ='eval_set')
```

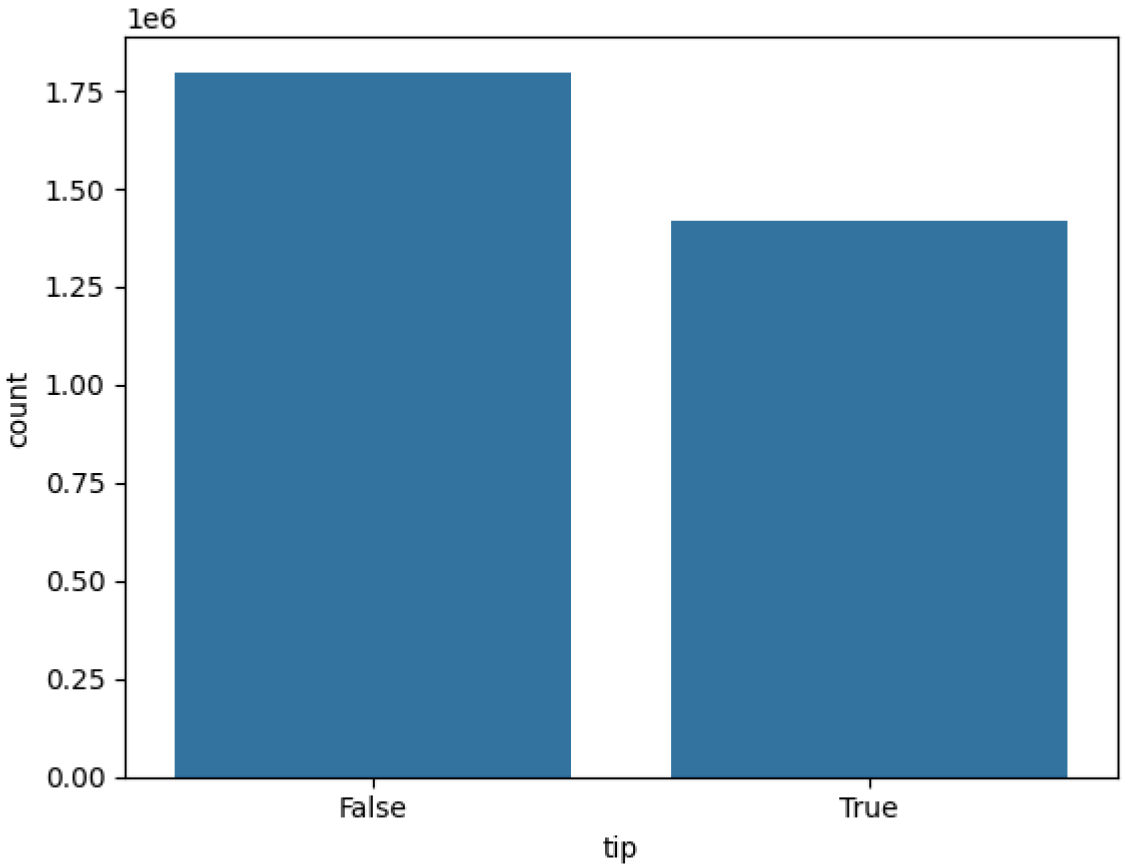
```
Out[ ]: <Axes: xlabel='eval_set', ylabel='count'>
```



- For the table tip_train (since "tip" is also categorical)

```
In [ ]: # Lets look at the target attribute tip, which is categorical
        sns.countplot(data=tip_train, x='tip')
```

```
Out[ ]: <Axes: xlabel='tip', ylabel='count'>
```



Missing Values

- We should also check if there is missing data

In [ ]:

```
print(orders.isnull().sum().sort_values(ascending=False))
orders.isnull().mean().value_counts()
```

```
days_since_prior_order    206209
order_id                       0
user_id                        0
eval_set                       0
order_number                   0
order_dow                      0
order_hour_of_day              0
dtype: int64
```

```
Out[ ]:  0.000000    6
         0.060276    1
         Name: count, dtype: int64
```

- We can see that days_since_prior_order has 206209 missing entries, which corresponds to around 6% of the total data in that feature
- For all other tables and features, there are no missing entries

Feature Engineering

We have developed various features through extensive analysis. Below, we will provide a brief summary of these features, which will be examined in detail afterward.

Feature Engineering Summary

- Feature I: user_tipping_ratio: Represents the average tipping frequency of each user, calculated as the total tips divided by the number of orders with tips.

- Feature II: products_per_order: Indicates the average number of products per order.

- Feature III: day of week (dow) one-hot-encoding

- Feature IV: total_products_bought_per_user: Represents the total number of products bought by each user.

- Feature V: top_5_aisles_tip one hot encoding: Encodes the top 5 aisles where tipping occurs the most.

- Feature VI: department 5 one hot encoding

- Feature VII: tip_probability_sum_per_order: Represents the average probability of receiving a tip for each product, calculated as the mean tip amount for each product. We then sum up, which indicates the sum of tip probabilities for all products in an order.

- Feature VIII: hour_of_day_categorized: Categorizes the hour of the day into predefined time slots. (night, morning, day, evening)

Preparing dataframes for feature engineering

We build three data frames up simultaneously to prevent data leakage. This is our "data pipeline":

- Our train split
- Our test split
- test split from our professor, which we have to predict (tip_testdaten1_template.csv)

In [ ]:

```
# Data with the label "tip"
orders_with_tip = pd.merge(tip_train, orders, on='order_id', how="left")

# test split from our professor
test_to_predict = pd.merge(tip_test, orders, on='order_id', how="left")
test_to_predict = test_to_predict.drop('tip',axis=1)
```

In [ ]:

```
label_enc = LabelEncoder()

Y = label_enc.fit_transform(orders_with_tip['tip'])
X = orders_with_tip
train_df, test_df, y_train, y_test = train_test_split(X, Y, test_size=0.2, random_state=42)
test_df = test_df.drop("tip", axis=1)
```

Feature I: user_tipping_ratio

- Calculate the count of orders and the count of tips per user.
- Calculate the percentage of tipping per user.
- Merge the feature to all orders per user.

In [ ]:

```
# Building the feature
users_with_tip = orders_with_tip.groupby('user_id').agg({'order_id': 'count', 'tip': 'sum'})
users_with_tip['user_tipping_ratio'] = users_with_tip['tip'] / users_with_tip['order_id']
```

```
orders_with_user_tip_ratio = pd.merge(orders_with_tip, users_with_tip['user_tipping_ratio'], on='user_id', how='left')
orders_with_user_tip_ratio = orders_with_user_tip_ratio.drop_duplicates(['user_id'])
orders_with_user_tip_ratio = orders_with_user_tip_ratio[['user_id','user_tipping_ratio']]
```

In [ ]:

```
# Adding the new feature to our three base dataframes
train_df = pd.merge(train_df, orders_with_user_tip_ratio[["user_id","user_tipping_ratio"]], on="user_id", how='left')
test_df = pd.merge(test_df, orders_with_user_tip_ratio[["user_id","user_tipping_ratio"]], on="user_id", how='left')
test_to_predict= pd.merge(test_to_predict, orders_with_user_tip_ratio[["user_id","user_tipping_ratio"]], on="user_id", how='le
```

Feature II: products_per_order

- calculate the count of products per order

In [ ]:

```
# Building the feature
count_products = opp.groupby('order_id').agg({'product_id': 'count'})
count_products = count_products.rename(columns={'product_id': 'products_per_order'})
count_products = count_products.reset_index()
```

In [ ]:

```
# Adding the new feature to our three base dataframes
train_df_1 = pd.merge(count_products, train_df, on='order_id', how='right')
test_df_1 = pd.merge(count_products, test_df, on='order_id', how='right')
test_to_predict_1 = pd.merge(count_products, test_to_predict, on='order_id', how='right')
```

Feature III: day of week (dow) one-hot-encoding

- One-hot-encode dow to use them as feature in our model.

In [ ]:

```
train_df_2 = pd.get_dummies(train_df_1, columns=['order_dow'], prefix='dow')
train_df_2[["dow_0","dow_1","dow_2","dow_3","dow_4","dow_5","dow_6"]] = train_df_2[["dow_0","dow_1","dow_2","dow_3","dow_4","d
```

In [ ]:

```
test_df_2 = pd.get_dummies(test_df_1, columns=['order_dow'], prefix='dow')
test_df_2[["dow_0","dow_1","dow_2","dow_3","dow_4","dow_5","dow_6"]] = test_df_2[["dow_0","dow_1","dow_2","dow_3","dow_4","dow
```

In [ ]:

```
test_to_predict_2 = pd.get_dummies(test_to_predict_1, columns=['order_dow'], prefix='dow')
test_to_predict_2[["dow_0","dow_1","dow_2","dow_3","dow_4","dow_5","dow_6"]] = test_to_predict_2[["dow_0","dow_1","dow_2","dow
```

Feature IV: total_products_bought_per_user

- Calculate the total number of bought products in the entire customer lifetime

In [ ]:

```
# Building the feature
total_products_per_user = pd.merge(opp, orders, on='order_id', how='right')
total_products_per_user = total_products_per_user.groupby('user_id').size().reset_index(name='total_products_bought_per_user')
```

In [ ]:

```
# Adding the new feature to our three base dataframes
train_df_3 = pd.merge(train_df_2, total_products_per_user, on='user_id', how='left')
test_df_3 = pd.merge(test_df_2, total_products_per_user, on='user_id', how='left')
test_to_predict_3 = pd.merge(test_to_predict_2, total_products_per_user, on='user_id', how='left')
```

Feature V: top_5_aisles_tip

- One-hot-encoding for the top 5 aisles with the highest tip-chance.
- Adding up the number, how many products in an order are from aisle_x

In [ ]:

```
top_5_aisles = [62,28,27,134,124]

aisles_df = pd.merge(orders, opp, on='order_id', how='left')
aisles_df = pd.merge(aisles_df, products, on='product_id', how='left')

aisles_df = aisles_df[['order_id','aisle_id']]
for aisle_id in top_5_aisles:
    aisles_df[f'aisle_{aisle_id}'] = (aisles_df['aisle_id'] == aisle_id).astype(int)

aisles_df = aisles_df.groupby('order_id').agg({'aisle_62': 'sum', 'aisle_28': 'sum', 'aisle_27': 'sum', 'aisle_134': 'sum', 'a
aisles_df.head(5)
```

Out[ ]:

	aisle_62	aisle_28	aisle_27	aisle_134	aisle_124
order_id					
1	0	0	0	0	0
2	0	0	0	0	0
3	0	0	0	0	0
4	0	0	0	0	0
5	0	0	0	0	0

In [ ]:

```
train_df_4 = pd.merge(train_df_3, aisles_df, on='order_id', how='left')
test_df_4 = pd.merge(test_df_3, aisles_df, on='order_id', how='left')
test_to_predict_4 = pd.merge(test_to_predict_3, aisles_df, on='order_id', how='left')
```

Feature VI: department 5 one hot encoding

- One-hot-encode department 5 with ~80 % tip probability.
- Adding up the number, how many products in an order are from department_5

In [ ]:

```
departments_df = pd.merge(orders, opp, on='order_id', how='left')
departments_df = pd.merge(departments_df, products, on='product_id', how='left')
departments_df = departments_df[['order_id','department_id']]
departments_df = departments_df[departments_df['department_id'] == 5]
departments_df = departments_df.groupby('order_id').agg({'department_id': 'count'})
departments_df.head(5)
```

Out[ ]:

	department_id
order_id	
13	2
191	2
194	2
282	3
283	2

In [ ]:

```
train_df_5 = pd.merge(train_df_4, departments_df, on='order_id', how='left').fillna(0)
test_df_5 = pd.merge(test_df_4, departments_df, on='order_id', how='left').fillna(0)
test_to_predict_5 = pd.merge(test_to_predict_4, departments_df, on='order_id', how='left').fillna(0)
```

Feature VII: tip_probability_per_product

- We group the products by product id and look at which products are contained in an order that leads to a tip.
- We use the mean because we take all probabilities of all products per order_id and then assign a mean probability to reflect all the products in a given order.

In [ ]:

```
tip_products = pd.merge(opp, orders, on='order_id', how='left')
tip_products = pd.merge(tip_products, tip_train, on='order_id', how='left')
tip_per_products = tip_products.groupby('product_id')['tip'].mean()
tip_products = pd.merge(tip_products, tip_per_products.rename('tip_probability_sum_per_order'), on='product_id', how='left')
tip_per_order = tip_products.groupby('order_id').agg({'tip_probability_sum_per_order': 'sum'})
```

In [ ]:

```
train_df_6 = pd.merge(train_df_5, tip_per_order, on='order_id', how='left').fillna(0)
test_df_6 = pd.merge(test_df_5, tip_per_order, on='order_id', how='left').fillna(0)
test_to_predict_6 = pd.merge(test_to_predict_5, tip_per_order, on='order_id', how='left').fillna(0)
```

Feature VIII: hour_of_day_categorized

- We categorize the different hours of the day into categorical data in order to use them as a feature.

In [ ]:

```
def categorize_hour(hour):
    if 0 <= hour < 6:
        return 'night'
    elif 6 <= hour < 12:
        return 'morning'
    elif 12 <= hour < 18:
        return 'afternoon'
    elif 18 <= hour < 24:
        return 'evening'
```

In [ ]:

```
train_df_6['hour_of_day_categorized'] = train_df_6['order_hour_of_day'].apply(categorize_hour)
train_df_6 = pd.get_dummies(train_df_6, columns=['hour_of_day_categorized'])
```

In [ ]:

```
test_df_6['hour_of_day_categorized'] = test_df_6['order_hour_of_day'].apply(categorize_hour)
test_df_6 = pd.get_dummies(test_df_6, columns=['hour_of_day_categorized'])
```

In [ ]:

```
test_to_predict_6['hour_of_day_categorized'] = test_to_predict_6['order_hour_of_day'].apply(categorize_hour)
test_to_predict_6 = pd.get_dummies(test_to_predict_6, columns=['hour_of_day_categorized'])
```

Our final dataframe with all features to train our model

In [ ]:

```
train_df_6
```

Out[ ]:

	order_id	products_per_order	tip	user_id	eval_set	order_number	order_hour_of_day	days_since_prior_order	user_tipping_rati
0	345739	3	False	6091	prior	11	14	18.0	0.06250
1	1389585	19	False	84074	prior	5	11	7.0	0.54545
2	1491809	4	True	161504	prior	20	16	5.0	0.90000
3	3069719	12	False	151647	prior	16	12	8.0	0.27272
4	953328	19	False	201251	prior	6	20	2.0	0.14285
...	...	...	...	...	...	...	...	...	.
2571894	1590861	1	True	108167	prior	30	9	2.0	0.46666
2571895	167456	10	True	150943	prior	5	23	30.0	0.70000
2571896	2451680	15	False	142758	prior	3	17	30.0	0.42857
2571897	2957599	3	False	177784	prior	19	12	11.0	0.58064
2571898	1666049	5	True	142142	prior	40	13	4.0	0.77611

2571899 rows × 28 columns

Model selection and training

- Following the implementation of various engineered features, we will outline the steps taken to determine our final machine learning model
- This encompasses the steps we took from the initial naive ZeroR approach and benchmark to our final optimized model in terms of accuracy

ZeroR

- We chose ZeroR as a simplistic classification strategy and benchmark
- ZeroR works by disregarding the predictor variables and relying solely on the target variable for decision-making

In [ ]:
```
yes_tips = tip_train.loc[tip_train["tip"] == 1]
no_tips = tip_train.loc[tip_train["tip"] == 0]

zeroR_prediction = len(yes_tips) if len(yes_tips) > len(no_tips) else len(no_tips)
```

In [ ]:
```
zeroR_accuracy = zeroR_prediction / len(tip_train)
zeroR_accuracy
```

Out[ ]:
```
0.5591559731423378
```

XGBClassifier

- We chose the XGBoostClassifier as implemented machine learning algorithm for our engineered features
- It constructs an ensemble of weak learners, often decision trees
- By combining the predictions of these weak learners, the XGBoostClassifier enhances prediction accuracy

In [ ]:
```
df = train_df_6.copy()
# Convert 'tip' which is categorical into numerical data so it's usable for our machine learning model
df['tip'] = LabelEncoder().fit_transform(df['tip'])
# Deal with missing values (NaN) by converting them to 0
df = df.fillna(0)
# Delete the columns with little value for our prediction
df = df.drop(['order_id', 'user_id','eval_set','order_number','tip'], axis=1)
```

In [ ]:
```
df_final_test = test_df_6.copy()
# Deal with missing values (NaN) by converting them to 0
df_final_test = df_final_test.fillna(0)
# Delete the columns with little value for our prediction
df_final_test = df_final_test.drop(['order_id', 'user_id','eval_set','order_number'], axis=1)
```

In [ ]:

```
df_final_test_to_predict = test_to_predict_6.copy()
# Deal with missing values (NaN) by converting them to 0
df_final_test_to_predict = df_final_test_to_predict.fillna(0)
# Delete the columns with little value for our prediction
df_final_test_to_predict = df_final_test_to_predict.drop(['order_id', 'user_id','eval_set','order_number'], axis=1)
```

In [ ]:

```
model = xgb.XGBClassifier(random_state=42, use_label_encoder=False, eval_metric='mlogloss', n_jobs=-1)
```

In [ ]:

```
# Cross validation of 5 because this provides a good balance between variance and bias in estimating
# the performance of the model
cv_scores = cross_val_score(model, df, y_train, cv=5, n_jobs=-1)
print("Cross-validation scores:", cv_scores)
print("Mean cross-validation score:", cv_scores.mean())
```

```
Cross-validation scores: [0.80266534 0.80424006 0.80335938 0.80245344 0.80285159]
Mean cross-validation score: 0.8031139635489692
```

In [ ]:

```
model.fit(df, y_train)
y_pred = model.predict(df_final_test)
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))
```

```
Accuracy: 0.8045305027411641
Classification Report:
               precision    recall  f1-score   support

           0       0.82      0.84      0.83    359861
           1       0.79      0.76      0.78    283114

    accuracy                           0.80    642975
   macro avg       0.80      0.80      0.80    642975
weighted avg       0.80      0.80      0.80    642975
```

See intermediate results of our model with XGBClassifier and Feature Engineering

Hyperparameter Tuning

- Hyperparameter tuning should be done in combination with feature engineering

- Ensures optimal model performance by finding the best combination of hyperparameters
- Allows the model to effectively leverage engineered features for improved predictive power

We chose GridSearch for Hyperparameter Tuning because it systematically evaluates all possible combinations of hyperparameters, guaranteeing the best solution. Random search, on the contrary, randomly picks hyperparameter combinations, potentially missing the most optimal one. Furthermore we had a PC with strong specs henceforth we decided upon GridSearch.

GridSearch

- We make sure to set n_jobs to -1 so that we utilize all cores
- We set a cross validation value of 5 because this provides a good balance between variance and bias in estimating the performance of the model
- We set a random state to ensure reproducibility

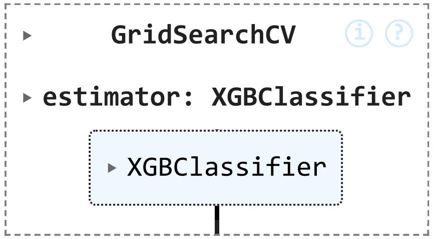
In [ ]:

```python
# Define the parameter grid to search

param_grid = {
            'max_depth': [3, 5],
            'learning_rate': [0.1, 0.01, 0.001],
            'subsample': [0.5, 0.7, 1]
    }

# Perform GridSearchCV with 5-fold cross validation
model_grid_search = GridSearchCV(estimator=model, param_grid=param_grid, cv=5, n_jobs=-2)
model_grid_search.fit(df, y_train)
```

Out[ ]:



```
▸ GridSearchCV
▸ estimator: XGBClassifier
    ▸ XGBClassifier
```

In [ ]:

```python
# Get the best parameters and the best score

best_params = model_grid_search.best_params_
```

```
best_score = model_grid_search.best_score_

print("Best parameters:", best_params)
print("Best score:", best_score)
```

```
Best parameters: {'learning_rate': 0.1, 'max_depth': 5, 'subsample': 1}
Best score: 0.8014871499521508
```

In [ ]:

```
# Create a variable for the best model
model_best_grid = model_grid_search.best_estimator_
```

Now let us see the results of our model after GridSearch

In [ ]:

```
y_pred = model_best_grid.predict(df_final_test)
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))
```

```
Accuracy: 0.8025770830903224
Classification Report:
               precision    recall  f1-score   support

           0       0.82      0.84      0.83    359861
           1       0.78      0.76      0.77    283114

    accuracy                           0.80    642975
   macro avg       0.80      0.80      0.80    642975
weighted avg       0.80      0.80      0.80    642975
```

Results:

Our Gridsearch made our accuracy worse, so we decide to use our base model from XGBoost without hyperparameter optimization from GridSearch

XGBoost Random Forest Classifier

In [ ]:

```
# Instantiate the XGBRFClassifier
xgbrf_classifier = xgb.XGBRFClassifier()
```

```
# Fit the grid search to the training data

xgbrf_classifier.fit(df, y_train)
```

Out[ ]:

▾ XGBRFClassifier

```
XGBRFClassifier(base_score=None, booster=None, callbacks=None,
                colsample_bylevel=None, colsample_bytree=None, device=None,
                early_stopping_rounds=None, enable_categorical=False,
                eval_metric=None, feature_types=None, gamma=None,
                grow_policy=None, importance_type=None,
                interaction_constraints=None, max_bin=None,
                max_cat_threshold=None, max_cat_to_onehot=None,
                max_delta_step=None, max_depth=None, max_leaves=None,
                min_child_weight=None, missing=nan, monotone_constraints=None,
```

Now let us see the results for our alternative ML model

In [ ]:

```
# Predict the labels of the training set
y_pred = xgbrf_classifier.predict(df_final_test)
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))
```

```
Accuracy: 0.7980076985885921
Classification Report:
               precision    recall  f1-score   support

           0       0.80      0.85      0.82    359861
           1       0.79      0.73      0.76    283114

    accuracy                           0.80    642975
   macro avg       0.80      0.79      0.79    642975
weighted avg       0.80      0.80      0.80    642975
```

Now, let us see the results of our improved model

In [ ]:

```
##### überarbeiten nachdem Minhs Programme durchgelaufen für obige Ansätze. Je nachdem die Variabel model ersetzen

y_pred = model.predict(test_df)
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))

#Matrix plot
```

```
Accuracy: 0.8034992996652638
Classification Report:
               precision    recall  f1-score   support

           0       0.82      0.83      0.83   3610805
           1       0.79      0.76      0.78   2876093

    accuracy                           0.80   6486898
   macro avg       0.80      0.80      0.80   6486898
weighted avg       0.80      0.80      0.80   6486898
```

Results:

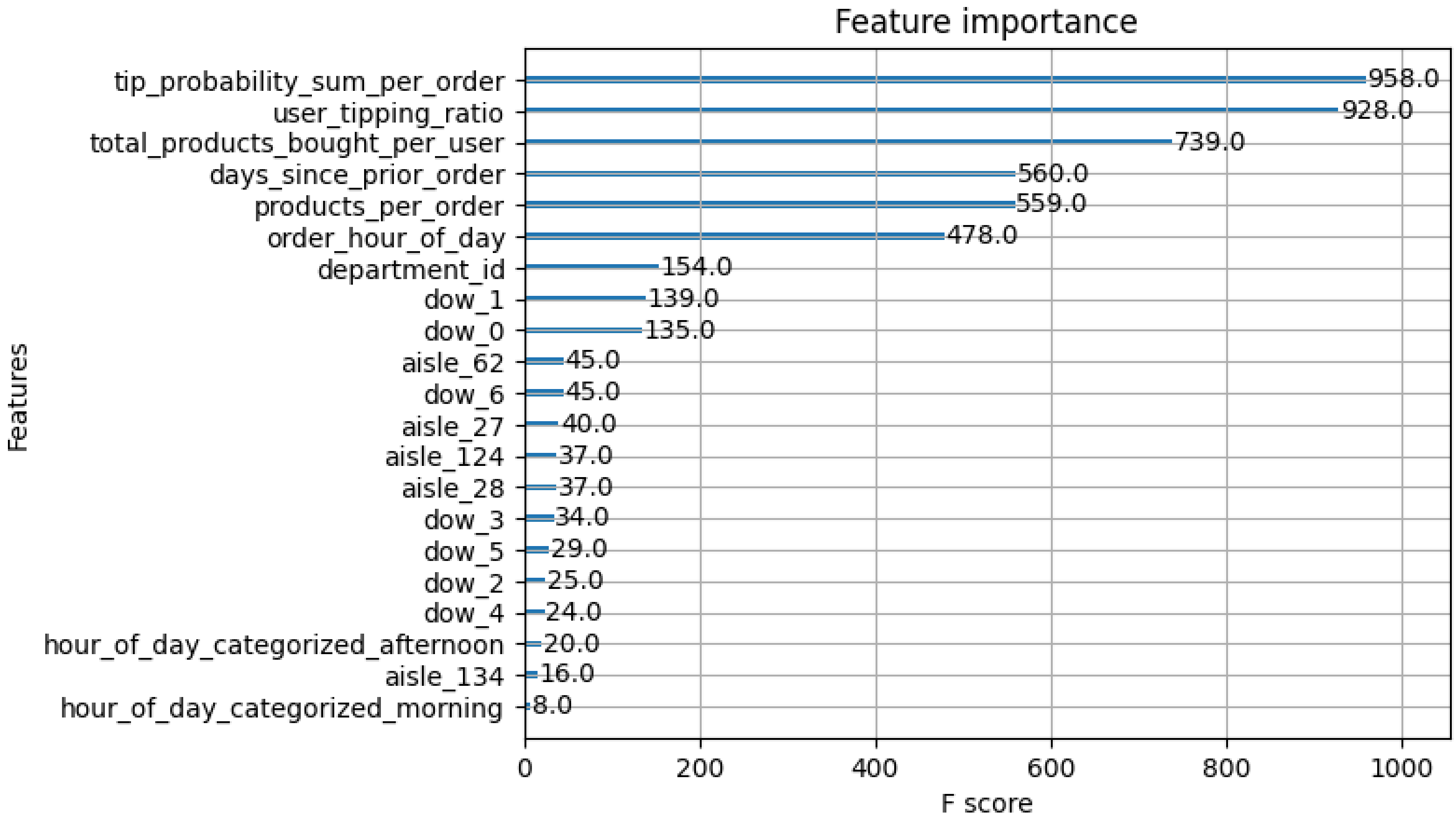
- Our Randomforest achieved a worse accuracy, so we decide to use XGBoost
- We now check our Feature Importances

Report of Insights we found which factors influence tipping

- For detailed feature explanations see the numerically listed individual features
- tip_probability_sum_per_order: The size of the order in combination with the tip probability for a specific product seems to have the biggest influence on tipping behaviour
- user_tipping_ratio: tipping depends on the user himself, this could mean tipping is a personal decision
- total_products_bought_per_user: The total amount of products a customer has bought until now, can be interpreted as representing customer loyalty/customer satisfaction

```
In [ ]: xgb.plot_importance(model)
```

```
Out[ ]: <Axes: title={'center': 'Feature importance'}, xlabel='F score', ylabel='Features'>
```



CSV Export

- After choosing and training the final ML model a CSV export feature was added
- We deencode our binary classifications back to the True and False values
- For our final export, we use more data in our own split so that the model can see more data, since we have already tested the model

In [ ]:

```
final_prediction = model.predict(df_final_test_to_predict)
tip_test_solution = tip_test.copy()
tip_test_solution['tip'] = final_prediction
# We now reformat our solution table to make it fit the training data formats
tip_test_solution.replace({0: False, 1: True}, inplace=True)
# We now write our prediction to csv
tip_test_solution['order_id'] = tip_test_solution['order_id'].astype(str)
tip_test_solution.to_csv('tip-predictions_26_06_final.csv')
```

High Level Summary

- We have achieved an accuracy of 80,3%
- As we can see in our feature importance graph, only a few features influence the model in any impactful manner, but those that do, do so disproportinately compared to features with low importance
- Our Feature Engineering seems to have had the largest impact by far, and we can see that using grid or randomsearch does not improve our results, or even worsens them
- We could try to optimize for our missing values, since we have 6% missing entries in days_since_prior_order, we could try and use k-nearest neighbour as a method for example to see if we can improve results, however due to negligible influence we do not use this, also it may worsen our results
- A better inclusion of the timeline in the models decision making could possibly improve results, however, the opposite could be the case, since for example new customers are better represented in our time-independent model
- The last few (possible) percent points are by far the hardest to achieve

Conclusions for Instacart

- Further data analysis can pinpoint the exact order composition. With this information, the therein contained products can be grouped together by having their aisles reallocated closer to each other to incentivize larger orders and higher spending. This would also translate to higher profits.
- For example, we can see that the orders containing the product group 'alcohol' constitute a large amount of the orders where a tip is given. For these products, more marketing and advertisement efforts can be made.
- As the total_products feature can be used to identify customer readiness to spend, we could use a threshhold as a minimum number of products ordered to gain a small bonus or voucher for example, to further incentivize larger orders and thus increased profits per order.

We could use the p3-quantile for example, if we want to give vouchers to customers who have orders in the 75% or larger order sizes. (see boxplot below)

In [ ]:

```
import plotly.express as px
```

In [ ]:

```
px.box(total_products_per_user['total_products_bought_per_user'])
```

In [ ]:

```
total_products_per_user.describe()
```

Out[ ]:

	user_id	total_products_bought_per_user
count	206209.000000	206209.000000
mean	103105.000000	158.289396
std	59527.555167	204.208233
min	1.000000	4.000000
25%	51553.000000	40.000000
50%	103105.000000	84.000000
75%	154657.000000	189.000000
max	206209.000000	3726.000000

- As we can see, the median order size is around 84, so for example we could offer vouchers for customers who have order sizes that qualify for the third quantile (189 products)
- Products with very high tip probability should be placed (accordingly if size and convenience allows,) at end of bagging area/cashier to incentivize the customer to take it with them

Important notice

- We must however, be aware of the issue of false correlations. For each proposed business impact, we cannot assume with certainty, that there are no unknown variables XYZ that are influencing beside the variables we identify that have a cause relationship with certain decisions like deciding customer satisfaction, readiness to spend etc. Further evaluation is needed to make sure our suggestions can actually work in a practical setting.

- These could include further data gathering, especially newer data, and data after the changes proposed above are made, and comparing results to older (business) data, to see if Key metrics like revenue etc. see any changes.

Final Prediction for unknown test set

- Based on our cross validation results, we expect an accuracy score for the professor's test set of around 80,3%