Food Demand Forecasting Challenge

Abstract

Demand forecasting is one of the most common time series problems in today's world holding great significance in several business applications and much more. In this project, we take up a food demand forecasting problem where the goal is to predict the demand for the following 10 weeks from the previously given data. For the purpose of this problem multiple models were implemented of which boosting models proved to produce the best results. The LightGBM model gave us the best performance with an RMLSE of 0.51 placing us in the top 50 in the leaderboard of this forecasting challenge.

Introduction

Demand forecasting is a key component to every growing online business. Without proper demand forecasting processes in place, it can be nearly impossible to have the right amount of stock on hand at any given time. A food delivery service has to deal with a lot of perishable raw materials which makes it all the more important for such a company to accurately forecast daily and weekly demand.

Too much inventory in the warehouse means more risk of wastage, and not enough could lead to out-of-stocks — and push customers to seek solutions from your competitors. Therefore, reliable forecasting is highly essential and models must be created to achieve such levels of performance.

Objective

In this challenge we have a client as a meal delivery company which operates in multiple cities, they have various fulfillment centers in these cities for dispatching meal orders to their customers. The client wants us to help these centers with demand forecasting for upcoming weeks so that these centers will plan the stock of raw materials accordingly.

The replenishment of majority of raw materials is done on weekly basis and since the raw material is perishable, the procurement planning is of utmost importance. Secondly, staffing of the centers is also one area wherein accurate demand forecasts are really helpful.

Given the following information, the task is to predict the demand for the next 10 weeks (Weeks: 146-155) for the center-meal combinations in the test set:

- Historical data of demand for a product-center combination (Weeks: 1 to 145)
- Product(Meal) features such as category, sub-category, current price and discount
- Information for fulfillment center like center area, city information etc.

Data Dictionary

1. **Weekly Demand data (train.csv):** Contains the historical demand data for all centers, test.csv contains all the following features except the target variable

Variable	Definition
id	Unique ID
week	Week No
center_id	Unique ID for fulfillment center
meal_id	Unique ID for Meal
checkout_price	Final price including discount, taxes & delivery charges
base_price	Base price of the meal
emailer_for_promotion	Emailer sent for promotion of meal
homepage_featured	Meal featured at homepage
num_orders	(Target) Orders Count

2. **fulfilment_center_info.csv:** Contains information for each fulfilment center

Variable	Definition
center_id	Unique ID for fulfillment center
city_code	Unique code for city
region_code	Unique code for region
center_type	Anonymized center type
op_area	Area of operation (in km^2)

3. **meal_info.csv:** Contains information for each meal being served

Variable	Definition
meal_id	Unique ID for the meal
category	Type of meal (beverages/snacks/ soups….)
cuisine	Meal cuisine (Indian/Italian/…)

Evaluation Metric

The evaluation metric for this competition is 100*RMSLE where RMSLE is Root of Mean Squared Logarithmic Error across all entries in the test set.

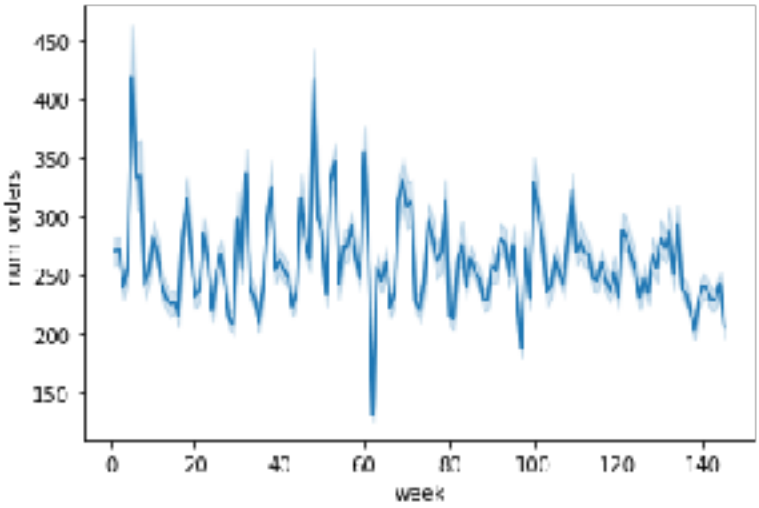
$$\sqrt{\frac{1}{n}\sum_{i=1}^{n}(\log(x_i+1) - \log(y_i+1))^2}$$

Methodology

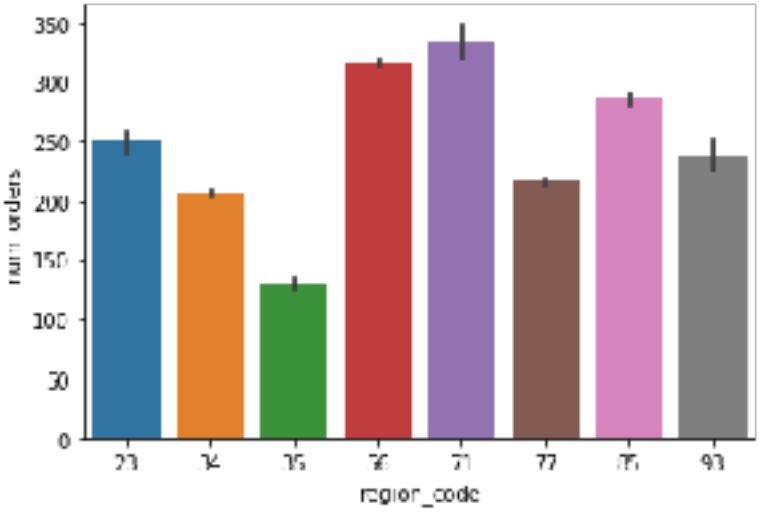
Several different approaches and preprocessing techniques were implemented for the purpose of this challenge some of which helped achieve good results while others didn't turn out to be that useful or rather even degrade performance.

A good amount of preprocessing and exploratory analysis was required to be done on the data before fitting any sort of model. Our exploratory analysis involved various visualizations to identify correlations amongst the various variables and to identify any trends present in the data.

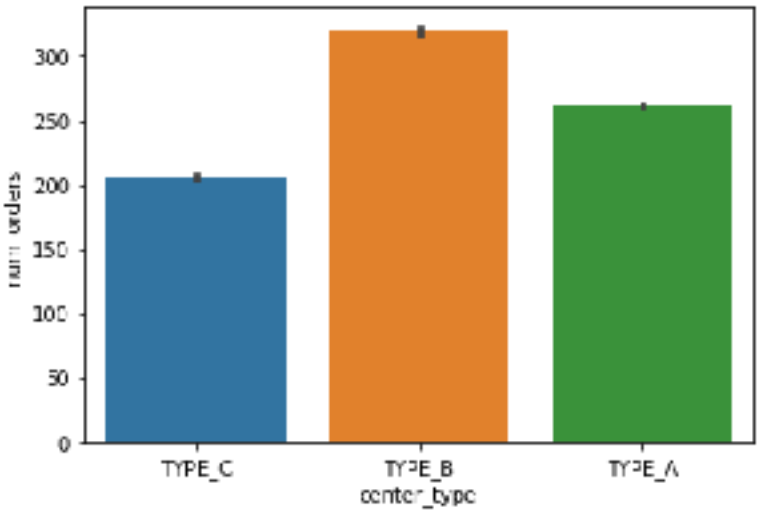
Visualization:



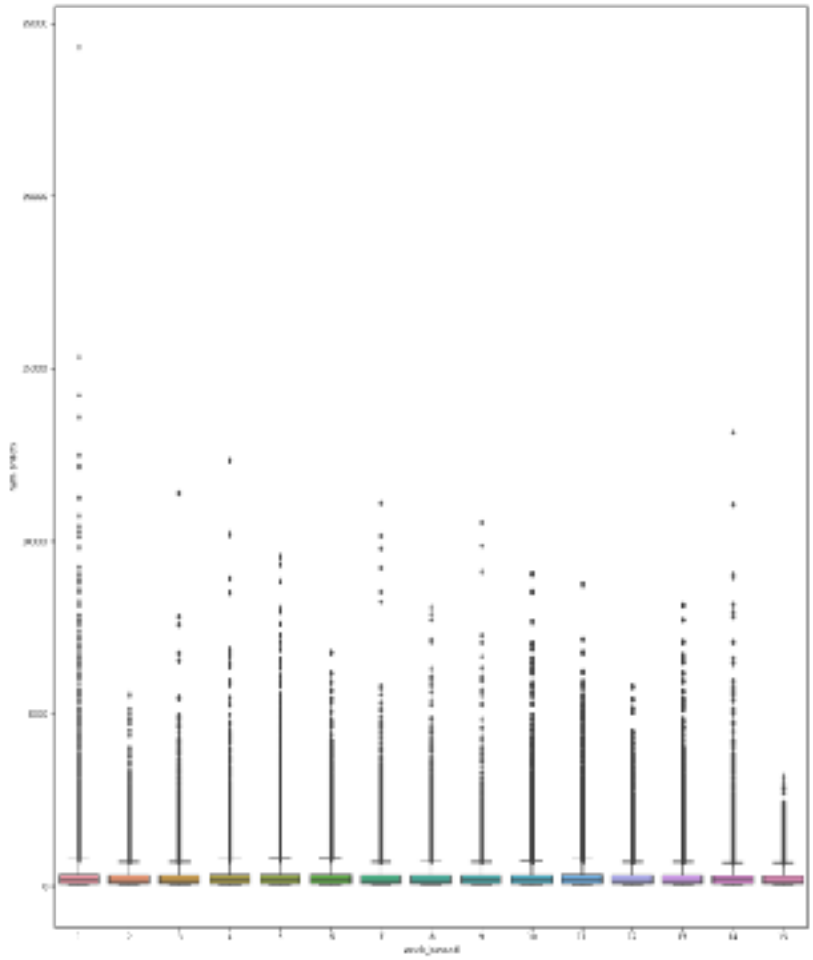
This is a lineplot showing the trend of the number of orders over the weeks of data that was provided to us. We can see a trend similar to a Sin/Cos curve which inspired used to perform the sine transforms on the data.



The above graph shows the num of orders in the different regions.



The graph above shows the number of orders placed in different center types. We can see that type B has the most orders and the centers belonging to center B must have higher stocks of raw materials in comparison to the other centers.



The plot above is the boxplot of the data to understand the distribution. As we can see the data is highly skewed due to the presence of a lot of outliers in it.

Pre-processing:

Outlier detection by means of a plot and removal was performed while ensuring minimal data loss.

The number of orders placed (target variable) was found to be highly right skewd and so we apply a log transformation.

The feature ‘price_last_curr_diff' had a few missing values which were filled with the mean.

Categorical variables namely category, cuisine, center_type, center_id, meal_id, city_region, price_increase and profit/loss were all label encoded. A one-hot encoder was used on label encoded features including emailer_for_promotion and homepage_featured. Feature scaling was carried out on the other numerical attributes using Standard Scaler.

Dimensionality reduction techniques including PCA and LDA were applied and tested however they resulted in poorer performance.

Many boosting models such as Catboost and LGBM internally take care of categorical features on their own and therefore our preprocessing is only applied to the appropriate model as and how required.

Feature Engineering:

Apart from the given attributes we have carried out some feature engineering to create a few extra features which helped model the data better. These include profit, pricediff, profit/loss, price_last_curr_diff and price_increase.

As this is a time series problem we found it useful and rather important to create a few lag features. Lag features are variables which contain data from prior time steps. We can convert time-series data into rows where every row contains data about one observation and includes all previous occurrences of that observation. We have created lag features for pricediff and checkout_price for preceding 10, 11 and 12 weeks.

Sine and cos transformations of the weeks attribute was done to create two features called week_sin and week_cos respectively. The purpose of this was to help capture the trend of the time-series more accurately therefore achieve better results when fit in our model.

Train-Test Split:

We use a train test split of 80-20 to evaluate our model. The training was done on data from weeks 1-135 and the testing was done one weeks 136-145. This method of training in the weeks of data and testing on the last few weeks in chronological order as the task at hand is time series forecasting. This method of validation helped us in generalizing the model better.

Models:

A few of the several models we have used for the given problem statement include:

- Random Forest
- CatBoost
- XGBoost
- LightGBM

Random Forest

Random Forest is basically bagging of decision trees. Random Forests provide an improvement over bagged trees by way of small tweak that decorrelates the trees. This reduces the variance when we average the trees. In bagging we build a number of decision trees bootstrapped training samples, but when building these decision trees, each time a split in a tree is considered, a random selection of ‘m’ predictors is chosen as split candidates form the full set of ‘p’ predictors. The split allows us to use only one of those m predictors.

A fresh selection of m predictors is taken at each split, and typically the number of predictors considered at each split is approximately equal to the square root of the total number of predictors.

Parameters tuned:

- N estimators = 10
- Random state = 0
- Verbose = 2
- N jobs = 1
- Min samples leaf = 5
- Max depth = 20
- Min samples split = 16
- OOB(Out-of-Bag) Score = true

CatBoost

CatBoost is a recently open-sourced machine learning algorithm from Yandex. It can work with diverse data types to help solve a wide range of problems. To top it up, it provides best-in-class accuracy.

It is especially powerful in two ways:

- It yields state-of-the-art results without extensive data training typically required by other machine learning methods, and
- Provides powerful out-of-the-box support for the more descriptive data formats that accompany many business problems.

“CatBoost” name comes from two words “**Cat**egory” and “**Boost**ing”.

We are using CatBoost mainly for three reasons:

Performance: CatBoost provides state of the art results and it is competitive with any leading machine learning algorithm on the performance front.

Handling Categorical features automatically: We can use CatBoost without any explicit pre-processing to convert categories into numbers. CatBoost converts categorical values into numbers using various statistics on combinations of categorical features and combinations of categorical and numerical features.

Robust: It reduces the need for extensive hyper-parameter tuning and lower the chances of overfitting also which leads to more generalized models.

In addition to this, CatBoost does not require conversion of data set to any specific format like XGBoost and LightGBM.

Parameters tuned:

- Iterations = 625
- Learning rate = 0.06
- Depth = 8
- l2 leaf reg = 10
- Loss function = 'RMSE'
- Random seed = 2018

XGBoost

XGBoost stands for e**X**treme **G**radient **B**oosting. It is a popular ensemble learning technique which implements gradient boosting. XGBoost are also highly efficient in execution speed and over all model performance.

We use XGBoost mainly for three reasons:

Regularization: XGBoost has an option to penalize complex models through both L1 and L2 regularization. Regularization helps in preventing overfitting

Handling sparse data: Missing values or data processing steps like one-hot encoding make data sparse. XGBoost incorporates a sparsity-aware split finding algorithm to handle different types of sparsity patterns in the data

Weighted quantile sketch: Most existing tree-based algorithms can find the split points when the data points are of equal weights (using quantile sketch algorithm). However, they are not equipped to handle weighted data. XGBoost has a distributed weighted quantile sketch algorithm to effectively handle weighted data

Parameters tuned:

- Max depth = 50
- Min child weight = 1
- N estimators = 200
- N jobs = -1
- Verbosity = 2
- Learning rate = 0.16

LightGBM

Light GBM is a fast, distributed, high-performance gradient boosting framework based on decision tree algorithm, used for ranking, classification and many other machine learning tasks.

Since it is based on decision tree algorithms, it splits the tree leaf wise with the best fit whereas other boosting algorithms split the tree depth wise or level wise rather than leaf-wise. So, when growing on the same leaf in Light GBM, the leaf-wise algorithm can reduce more loss than the level-wise algorithm and hence results in much better accuracy which can rarely be achieved by any of the existing boosting algorithms.

We use LightGBM mainly for three reasons:

Faster training speed and higher efficiency: Light GBM use histogram-based algorithm i.e. it buckets continuous feature values into discrete bins which fasten the training procedure.

Better accuracy than any other boosting algorithm: It produces much more complex trees by following leaf wise split approach rather than a level-wise approach which is the main factor in achieving higher accuracy. However, it can sometimes lead to overfitting which can be avoided by setting the max_depth parameter.

Compatibility with Large Datasets: It is capable of performing equally good with large datasets with a significant reduction in training time as compared to XGBOOST.

Parameters tuned:

- Learning rate = 0.003
- N estimators = 40000
- Silent = False
- **g

How each model treats Categorical Variables

- **Catboost vs. XGBoost vs. LightGBM**

CatBoost

CatBoost has the flexibility of giving indices of categorical columns so that it can be encoded as one-hot encoding using one_hot_max_size (Use one-hot encoding for all features with number of different values less than or equal to the given parameter value). If you don't pass any anything in cat_features argument, CatBoost will treat all the columns as numerical variables.

LightGBM

Similar to CatBoost, LightGBM can also handle categorical features by taking the input of feature names. It does not convert to one-hot coding, and is much faster than one-hot coding. LGBM uses a special algorithm to find the split value of categorical features.

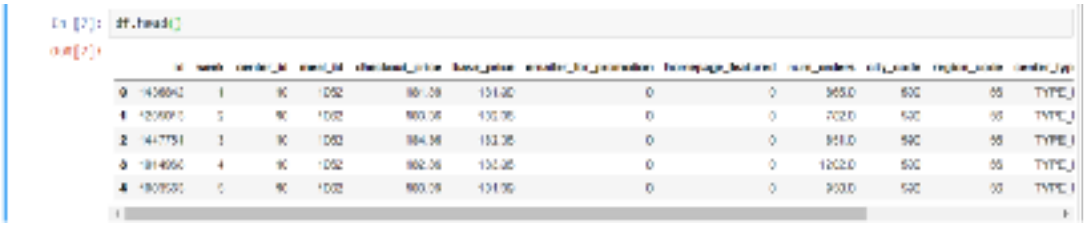
XGBoost

Unlike CatBoost or LGBM, XGBoost cannot handle categorical features by itself, it only accepts numerical values similar to Random Forest. Therefore, one has to perform various encodings like label encoding, mean encoding or one-hot encoding before supplying categorical data to XGBoost.

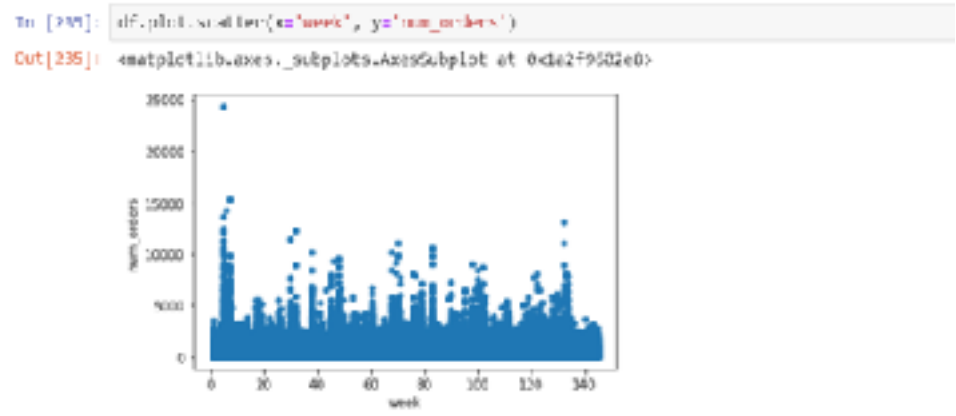
CODE:

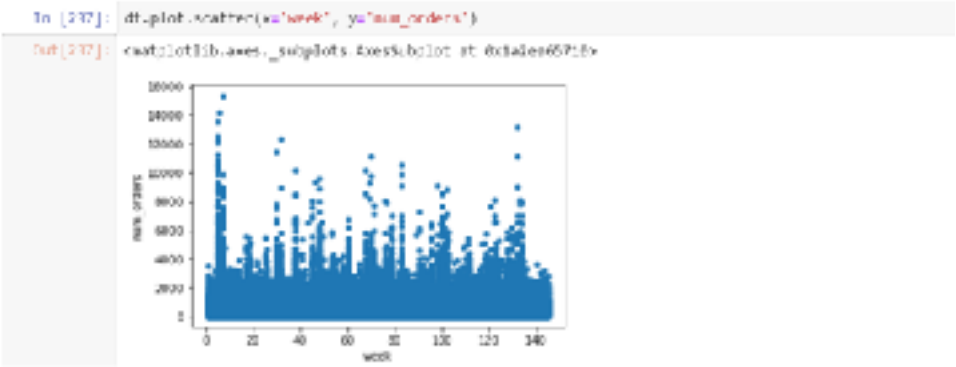
Reading data and merging tables

The tables have been read into pandas data frame and have been merged into a single one.



Outlier removal





Feature Engineering

Creation of features like profit,pricediff,profit/loss,price_diff,etc using the already existent features.

```
df['profit']=df['checkout_price']-df['base_price']
df['pricediff']=df['base_price']-df['checkout_price']
df['profit/loss'] = (df['profit'] > 0).astype(int)
df['price_last_curr_diff'] = (df['checkout_price'].shift(1) - df["checkout_price"])/(df['checkout_price'].shift(1))
df['price_last_curr_diff'] = df['price_last_curr_diff'].fillna(0)
df['price_increase'] = (df['price_last_curr_diff'] < 0).astype(int)
```

Creation of sales lag features for time series forecasting.

```
def create_sales_lag_feats(df, gpby_cols, target_col, lags):
    gpby = df.groupby(gpby_cols)
    for i in lags:
        df['_'.join([target_col, 'lag', str(i)])] = \
            gpby[target_col].shift(i).values + np.random.normal(scale=1.6, size=(len(df),))
    return df
```

```
df = create_sales_lag_feats(df, gpby_cols=['center_id','meal_id'], target_col='pricediff', lags=[10,11,12])
df = create_sales_lag_feats(df, gpby_cols=['center_id','meal_id'], target_col='checkout_price', lags=[10,11,12])
```

Merging categorical variables

```
df_train['city_region'] = \
        df_train['city_code'].astype('str') + '_' + \
        df_train['region_code'].astype('str')

df_test['city_region'] = \
        df_test['city_code'].astype('str') + '_' + \
        df_test['region_code'].astype('str')
```

```
C:\Users\Amrit\Anaconda3\lib\site-packages\ipykernel_launcher.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy
  This is separate from the ipykernel package so we can avoid doing imports until
C:\Users\Amrit\Anaconda3\lib\site-packages\ipykernel_launcher.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy
  import sys
```

Applying sin and cos transform on weeks

```
df_train['week_sin'] = \
        np.sin(2 * np.pi * df_train['week'] / 52.143)
df_train['week_cos'] = \
        np.cos(2 * np.pi * df_train['week'] / 52.143)

df_test['week_sin'] = \
        np.sin(2 * np.pi * df_test['week'] / 52.143)
df_test['week_cos'] = \
        np.cos(2 * np.pi * df_test['week'] / 52.143)
```

Log transformation of the target variable and sorting the final dataframe by weeks

```
df_train["num_orders_log1p"] = np.log1p(df_train['num_orders'])
```

```
C:\Users\Amrit\Anaconda3\lib\site-packages\ipykernel_launcher.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy
  """Entry point for launching an IPython kernel.
```

```
df = df_train.append(df_test).reset_index(drop=True)[df_train.columns]
df = df.sort_values(['center_id', 'meal_id', 'week']).reset_index(drop=True)
```

Preprocessing

Handling of null values

```
df["price_last_curr_diff"]=df["price_last_curr_diff"].fillna(df['price_last_curr_diff'].mean())
df["pricediff_lag_10"]=df["pricediff_lag_10"].fillna(0)
df["pricediff_lag_11"]=df["pricediff_lag_11"].fillna(0)
df["pricediff_lag_12"]=df["pricediff_lag_12"].fillna(0)
df["checkout_price_lag_10"]=df["checkout_price_lag_10"].fillna(0)
df["checkout_price_lag_11"]=df["checkout_price_lag_11"].fillna(0)
df["checkout_price_lag_12"]=df["checkout_price_lag_12"].fillna(0)
```

Standard Scaling and Label Encoding

```
from sklearn.preprocessing import LabelEncoder,StandardScaler
le1=LabelEncoder()
le2=LabelEncoder()
le3=LabelEncoder()
le4=LabelEncoder()
le5=LabelEncoder()
le6=LabelEncoder()
le7=LabelEncoder()
le8=LabelEncoder()
se1=StandardScaler()
se2=StandardScaler()
se3=StandardScaler()
se4=StandardScaler()
se5=StandardScaler()
se6=StandardScaler()
se7=StandardScaler()
fitter=StandardScaler()
df['category'] = le1.fit_transform(df['category'])
df['cuisine'] = le2.fit_transform(df['cuisine'])
df['center_type'] = le3.fit_transform(df['center_type'])
df['center_id'] = le4.fit_transform(df['center_id'])
df['meal_id'] = le5.fit_transform(df['meal_id'])
df['city_region']=le6.fit_transform(df['city_region'])
df['price_increase']=le7.fit_transform(df["price_increase"])
df['profit/loss']=le8.fit_transform(df['profit/loss'])
df['profit'] = se1.fit_transform(df[['profit']])
df['checkout_price'] = se2.fit_transform(df[['checkout_price']])
df['base_price'] = se3.fit_transform(df[['base_price']])
df['op_area'] = se4.fit_transform(df[['op_area']])
df['price_last_curr_diff'] = se5.fit_transform(df[['price_last_curr_diff']])
df['pricediff_lag_10'] = se1.fit_transform(df[['pricediff_lag_10']])
df['pricediff_lag_11'] = se2.fit_transform(df[['pricediff_lag_11']])
df['pricediff_lag_12'] = se3.fit_transform(df[['pricediff_lag_12']])
df['checkout_price_lag_10'] = se1.fit_transform(df[['checkout_price_lag_10']])
df['checkout_price_lag_11'] = se2.fit_transform(df[['checkout_price_lag_11']])
df['checkout_price_lag_12'] = se3.fit_transform(df[['checkout_price_lag_12']])
```

One-hot encoding

```
one hot encoding
one_hot = pd.get_dummies(df['emailer_for_promotion'],prefix='emailer_for_promotion_')
df = df.drop("emailer_for_promotion",axis = 1)
df = df.join(one_hot)
efplist(one_hot.columns)
efp=efp[:-1]

one_hot = pd.get_dummies(df['homepage_featured'],prefix='homepage_featured_')
df = df.drop("homepage_featured",axis = 1)
df = df.join(one_hot)
hflist=list(one_hot.columns)
hf=hf[:-1]

one_hot = pd.get_dummies(df['category'],prefix='category_')
df = df.drop("category",axis = 1)
df = df.join(one_hot)
catlist=list(one_hot.columns)
cat=cat[:-1]

one_hot = pd.get_dummies(df['cuisine'],prefix='cuisine_')
df = df.drop("cuisine",axis = 1)
df = df.join(one_hot)
cuslist=list(one_hot.columns)
cusn=cusn[:-1]

one_hot = pd.get_dummies(df['center_type'],prefix='center_type_')
df = df.drop('center_type',axis = 1)
df = df.join(one_hot)
ctlist=list(one_hot.columns)
ctn=ctn[:-1]
```

```
one_hot = pd.get_dummies(df['price_increase'],prefix='price_increase_')
df = df.drop('price_increase',axis = 1)
df = df.join(one_hot)
pi=list(one_hot.columns)
pi=pi[:-1]

one_hot = pd.get_dummies(df["profit/loss"],prefix="profit/loss_")
df = df.drop('profit/loss',axis = 1)
df = df.join(one_hot)
pl=list(one_hot.columns)
pl=pl[:-1]

one_hot = pd.get_dummies(df["city_region"],prefix="city_region_")
df = df.drop('city_region',axis = 1)
df = df.join(one_hot)
cr=list(one_hot.columns)
cr=cr[:-1]
```

Feature selection

```
In [23]: nfeat=['meal_id','center_id','week','week_sin','week_cos','profit','op_area','checkout_price','price_last_curr_diff','pricediff_
cat=['emailer_for_promotion','homepage_featured','category','cuisine','center_type','price_increase','profit/loss','city_region'
#x[illegible]
xfeat=nfeat+cat
x=df_train[xfeat]
y=df_train['num_orders_log1p']
```

Train-Test split:

```
In [25]: from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size=0.02,
                                                    shuffle=False)
```

Modelling

Random forest

```
In [26]: from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_log_error
regressor = RandomForestRegressor(n_estimators = 10, random_state = 0,verbose=2,n_jobs=-1,min_samples_leaf=5,max_depth=20,min_sa
regressor.fit(X_train,y_train)
ypred=regressor.predict(X_test)
print("Accuracy:",100*mean_squared_log_error(np.expm1(y_test),np.expm1(ypred)))

building tree 1 of 10building tree 2 of 10building tree 3 of 10

building tree 4 of 10

building tree 5 of 10
building tree 6 of 10
building tree 7 of 10
building tree 8 of 10
building tree 9 of 10
building tree 10 of 10

[Parallel(n_jobs=-1)]: Done  10 out of  10 | elapsed:   11.5s finished

Accuracy: 41.53[illegible]

/[illegible]/lib/python3.7/site-packages/sklearn/ensemble/forest.py:[illegible]: UserWarning: Some inputs do not have OOB scores. This
probably means too few trees were used to compute any reliable oob estimates.
  warn("Some inputs do not have OOB scores. "
[Parallel(n_jobs=4)]: Using backend ThreadingBackend with 4 concurrent workers.
[Parallel(n_jobs=4)]: Done  10 out of  10 | elapsed:    0.0s finished
```

Catboost Regressor

```
In [48]: import catboost as cb
         cat_rgr = cb.CatBoostRegressor(iterations=625, learning_rate=0.05, depth=8, l2_leaf_reg=10, loss_function='RMSE', random_seed=30
```

```
In [50]: cat_rgr.fit(X_train, y_train, cat_features=cat, verbose=True)
         ...
         624:    learn: 0.1867644        total: 2m 17s   remaining: 0us

Out[50]: <catboost.core.CatBoostRegressor at 0x1a1e3e6668>
```

```
In [51]: pred = cat_rgr.predict(X_test)
         print("Accuracy:",100*mean_squared_log_error(np.expm1(y_test),np.expm1(pred)))

         Accuracy: 32.68259645488088
```

LightGBM

```
In [32]: from lightgbm import LGBMRegressor

         g = {'colsample_bytree': 0.4,
              'min_child_samples': 5,
              'num_leaves': 275}

         estimator = LGBMRegressor(learning_rate=0.008,n_jobs=-1,max_depth=10,
                                   n_estimators=10000,
                                   silent=False,
                                   **g)

         fit_params = {"early_stopping_rounds": 1000,
                       "feature_name": xfeat,
                       'categorical_feature': cat,
                       "eval_set": [(X_train, y_train), (X_test, y_test)]}

         estimator.fit(X_train, y_train, **fit_params)
```

```
In [35]: from sklearn.metrics import mean_squared_log_error
         pred = estimator.predict(X_test)
         print("Accuracy:",100*mean_squared_log_error(np.expm1(y_test),np.expm1(pred)))

         Accuracy: 30.993407955456927
```

XGBoost

RESULTS:

Serial No	Model	Test Accuracy (RMSLE*100)	Submission Score(RMSLE* 100)
1.	Catboost	32.48	0.53
2.	Random Forest	41.53	0.68
3.	XGBoost	41.39	0.65
4.	LightGBM	30.99	0.514

CONCLUSION:

Multiple models were fit with varying features and parameter tuning resulting in different levels of performance each time. The best performances achieved were as follows: Random forest gave us an RMSLE of 0.68. Catboost resulted in an RMSLE of 0.53 while LightGBM produced the best results with an RMSLE of 0.5144.

Finally, the model that placed us in the 50th position of the competition is LightGBM producing a RMSLE score of 0.5144.

Further hyper parameter tuning and model enhancements could lead to potential performance improvements.