

A PROJECT REPORT ON

Predict Future Sales

A project report submitted in fulfillment for the Diploma Degree in AI & ML

Under

Applied Roots with University of Hyderabad



Project submitted by

Bhawik Jain

Under the Guidance of

Mentor: Harichandhana Kalikiri

Approved by : Mentor : Harichandhana Kalikiri



University of Hyderabad

Declaration of Authorship

We hereby declare that this thesis titled “Predict Future Sales” and the work presented by the undersigned candidate, as part of Diploma Degree in AI & ML.

All information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name: Harichandhana Kalikiri

Thesis Title: Predict Future Sales

CERTIFICATE OF RECOMMENDATION

We hereby recommend that the thesis entitled “Predict Future Sales” prepared under my supervision and guidance by Bhawik Jain be accepted in fulfillment of the requirement for awarding the degree of Diploma in AI & ML Under applied roots with University of Hyderabad. The project, in our opinion, is worthy of its acceptance.

Mentor: Harichandhana Kalikiri

Under Applied roots with



University of Hyderabad

ACKNOWLEDGEMENT

First and foremost, I have to thank my project mentor, Harichandhana Kalikiri. Without their assistance and dedicated involvement in every step throughout the process, this thesis would have never been accomplished. I would like to thank you very much for your support and understanding over these past few months.

I sincerely appreciate the inspiration; support and guidance of all those people, including my classmates and family, who have been instrumental in making this project a success. I, Bhawik Jain student of applied roots, is extremely grateful to mentors for the confidence bestowed in me and entrusting my project entitled "Predict Future Sales" with special reference.

At this juncture, I express my sincere thanks to applied roots for making the resources available at the right time and providing valuable insights leading to the successful completion of our project who even assisted me in completing the project.

Name: Bhawik Jain

Abstract:

Forecasting provides utmost value in any company. It has always been an area with direct application to various business problems. We are discussing a similar sales forecasting problem. With the sales data of one of the largest Russian software firms - 1C Company, Kaggle has a running competition in the name of "Predict Future Sales" where we are provided with daily historical sales data. The task is to forecast the total amount of products sold in every shop for the test set. This is a challenging problem as the list of shops and products slightly changes every month. Creating a robust model that can handle such situations is part of the challenge.

1. Business Problem

Predicting sales is crucial for any business, be it a small or large company. It is the most important element in business planning as it helps in financial planning, strategic planning and in making better decisions. It enables better inventory planning, timely promotional planning, competitive pricing, etc. For retailers, it helps them to take necessary measures to effectively allocate their resources and plan their budgets or investments in a time period among different product categories and items. Thereby, upscaling their business by completing customer demand. Since sales forecasting has far-reaching impacts, it's critical that the forecast information is as accurate as possible. Difficult and interesting sales forecasting problems are very common these days and one of them is provided by Kaggle as a running competition titled "Predict Future Sales". Given historical sales data (which slightly changes every month), we have to predict total sales for every product and store in the next month.

2. Dataset

2.1 Source of the dataset:

This data is taken from a running Kaggle Competition named "Predict Future Sales". This sales data is of one of the largest Russian software firms - 1C Company.

Link

<https://www.kaggle.com/c/competitive-data-science-predict-future-sales>

2.2 Dataset properties and features:

The size of the total data is 101.61MB. Since the data provided is not too big, it can be processed smoothly. We can download this dataset and upload it on our google drive. We can use Google Colab as a running environment after mounting our google drive on it. We are given 11 features in 4 different files and a test file.

3. Key Matric (KPI) to optimize

There are different types of evaluation metrics used in machine learning depending on the model used and the results generated. Since this data is a time series data, there are different evaluation metrics used to measure the performance of a time-series forecasting model.

Root Mean Square Error (RMSE) is a standard way to measure the error of a model in predicting quantitative data. A metric that tells us how far apart the predicted values are from the observed values in a dataset, on average. The lower the RMSE, the better a model fits a dataset. This measure is defined as the square root of mean square error. When the spread is important and bigger values need to be penalized, RMSE is useful.

$$\text{RMSE} = \sqrt{\frac{1}{n} \sum_{i=1}^n (Y_i - \hat{Y}_i)^2}$$

Where,

Y_i = Actual Value

\hat{Y}_i = Predicted Value

n = total number of values in test data

Other evaluation metrics are:

- R-Squared
- Mean Absolute Error(MAE)
- Mean Absolute Percentage Error(MAPE)
- Mean Squared Error(MSE)

When compared to MSE, RMSE is easier to interpret because the RMSE number is on the same scale as the projected values. Since we don't need Absolute Percentage Error in this problem and R^2 tells us how well a model can predict the value of the response variable in percentage terms while RMSE tells us in absolute terms, we choose RMSE as our Key Performance Indicator.

This statistic can mask issues with low data volume. Therefore, it should not be used with low data volume.

4. Real world challenges and constraints

This dataset is a time-series challenging dataset as the list of shops and products slightly changes every month. Creating a robust model that can handle such situations is part of the challenge.

It is very important to forecast sales as accurately as possible which indeed is a constraint for this problem for now.

5. Similar problem solved

This is a sales predicting problem and to solve this, we can use different Machine Learning techniques/models for solving :

- Linear Regression
- Random Forest Regression
- XGBoost
- LSTMs
- ARIMA Time Series Forecasting

After researching top kaggle solutions and comments, we have realized that applying XGBoost with lagged features did work out pretty well for many people in the past. Also, we have found that Autoregressive Integrated Moving Average (ARIMA) and LSTM based networks have provided quality results over similar forecasting tasks but not for this.

6. Exploratory Data Analysis

EDA is a significant statistical approach to understand what data we have in our hands. It is a crucial step as analyzing the data alone can give valuable insights to solve the problem. Understanding the structure of the dataset given is the initial stage in this process followed by the data preprocessing and data cleaning. Since we have time series data in our hands, we will look for trends, seasonality and relationship between various variables.

6.1 - Let us have a look at all the dataframes given.

(84, 2)

	item_category_name	item_category_id
0	PC - Гарнитуры/Наушники	0
1	Аксессуары - PS2	1
2	Аксессуары - PS3	2
3	Аксессуары - PS4	3
4	Аксессуары - PSP	4

(60, 2)

	shop_name	shop_id
0	!Якутск Орджоникидзе, 56 фран	0
1	!Якутск ТЦ "Центральный" фран	1
2	Адыгея ТЦ "Мера"	2
3	Балашиха ТРК "Октябрь-Киномир"	3
4	Волжский ТЦ "Волга Молл"	4

(2935849, 6)

	date	date_block_num	shop_id	item_id	item_price	item_cnt_day
0	02.01.2013	0	59	22154	999.00	1.0
1	03.01.2013	0	25	2552	899.00	1.0
2	05.01.2013	0	25	2552	899.00	-1.0
3	06.01.2013	0	25	2554	1709.05	1.0
4	15.01.2013	0	25	2555	1099.00	1.0

There are a total of 22,170 items, 84 categories and 60 shops and a total of 29,35,849 sales data.

---Missing value---

```
date          0
date_block_num 0
shop_id        0
item_id        0
item_price     0
item_cnt_day   0
dtype: int64
```

---Null value---

```
date          0
date_block_num 0
shop_id        0
item_id        0
item_price     0
item_cnt_day   0
dtype: int64
```

There are no missing values and null values in the train data. Dataset seems quite legitimate.

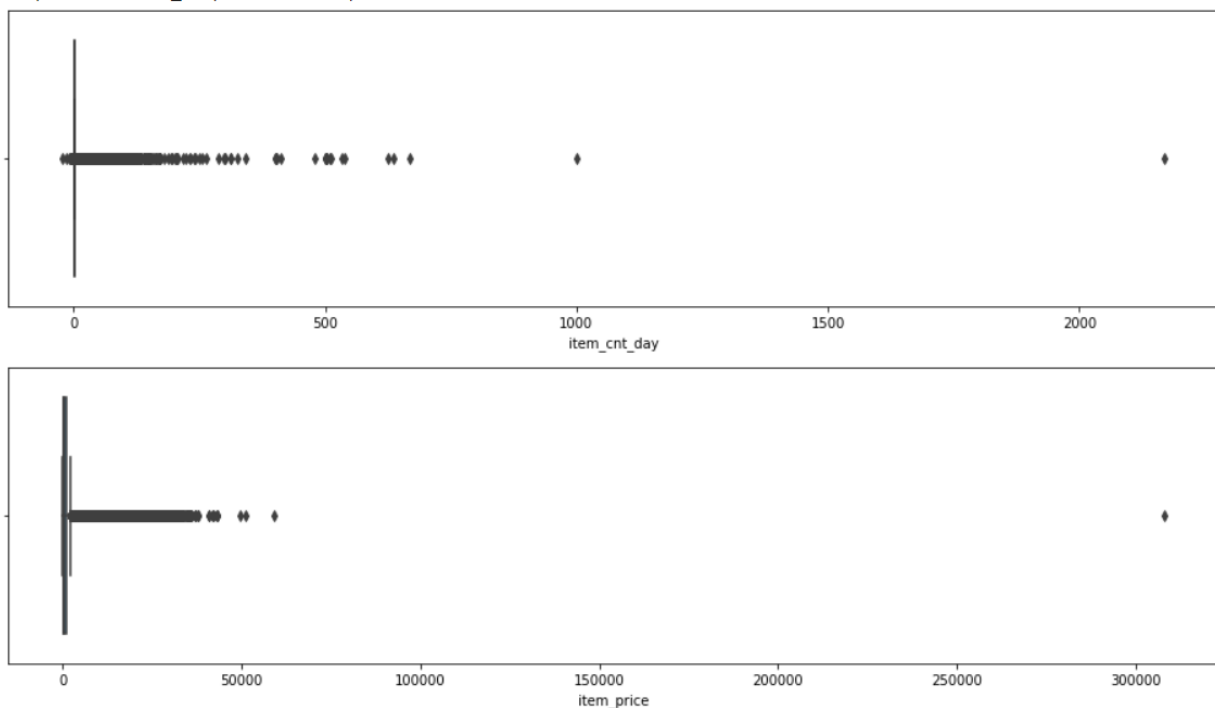
Description of train data

	date_block_num	shop_id	item_id	item_price	item_cnt_day
count	2.935849e+06	2.935849e+06	2.935849e+06	2.935849e+06	2.935849e+06
mean	1.456991e+01	3.300173e+01	1.019723e+04	8.908532e+02	1.242641e+00
std	9.422988e+00	1.622697e+01	6.324297e+03	1.729800e+03	2.618834e+00
min	0.000000e+00	0.000000e+00	0.000000e+00	-1.000000e+00	-2.200000e+01
25%	7.000000e+00	2.200000e+01	4.476000e+03	2.490000e+02	1.000000e+00
50%	1.400000e+01	3.100000e+01	9.343000e+03	3.990000e+02	1.000000e+00
75%	2.300000e+01	4.700000e+01	1.568400e+04	9.990000e+02	1.000000e+00
max	3.300000e+01	5.900000e+01	2.216900e+04	3.079800e+05	2.169000e+03

Minimum value for item_price and item_cnt_day is negative.

6.2 - Exploring item_cnt_day and item_price columns

Boxplot for item_cnt_day and item_price



- Anomalies in data - 7356 Negative value of item_cnt_day and 1 Negative value item_price.
- Outliers in data - Observed 1 Strange value of item_cnt_day (above 2000) and 1 extremely high item_price

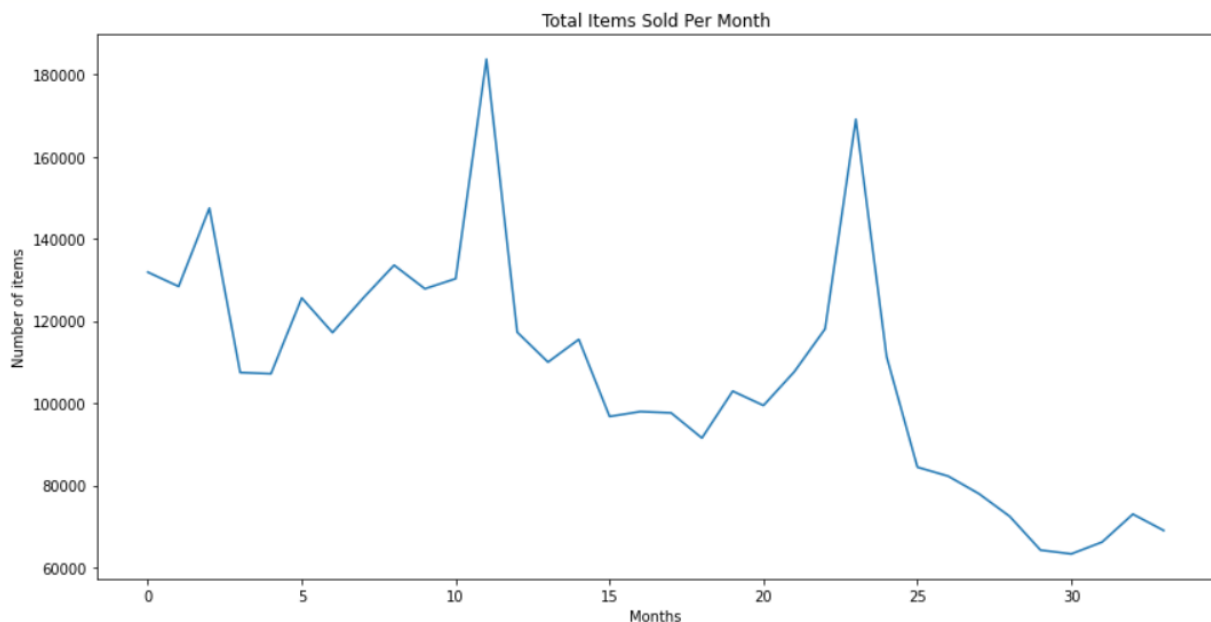
Negative value for item_cnt_day indicates that they are returned items, so we will set them to zero. Because no item was sold on that date and that returned item is included in previous sales.

```
#Updating negative value of item_cnt_day to 0.
train.loc[train.item_cnt_day < 0, 'item_cnt_day'] = 0
#Removing outliers
train = train[train.item_cnt_day < 1001]
train = train.loc[train['item_price'] < 100000]
```

Replaced the negative value of item_price with the mean of the other two item prices from the same month and from the same shop.

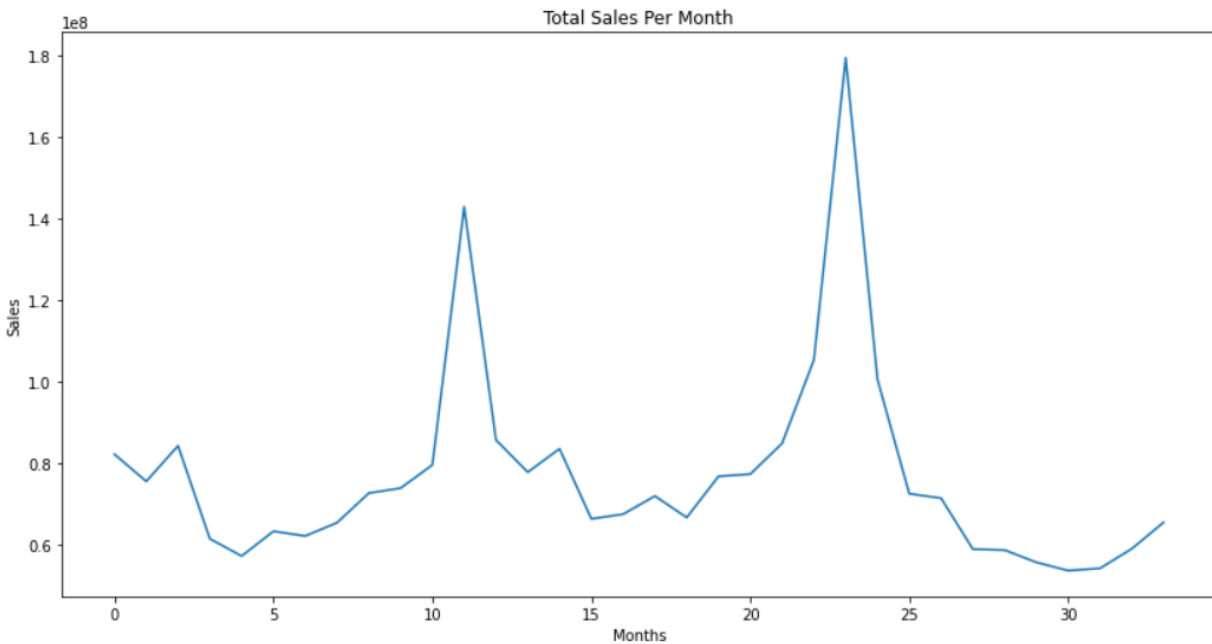
```
mean = train.loc[ [484682,484684], 'item_price' ].mean(axis=0)
train.loc[484683, 'item_price'] = mean
```

6.3 - Plotting items sold per month



- Seasonality : We can see two peaks around the end of the year(Dec 2013 and Dec 2014), therefore seasonality is there.
- Trend : Total number of items sold per month has a decreasing trend.

6.4 - Plotting total sales per month



- Seasonality : We can see that sales peaks towards the end of the year(Dec 2013 and Dec 2014) and lowers in mid-year. Therefore seasonality is there.
- Trend : Total sales per month trend looks relatively flat.

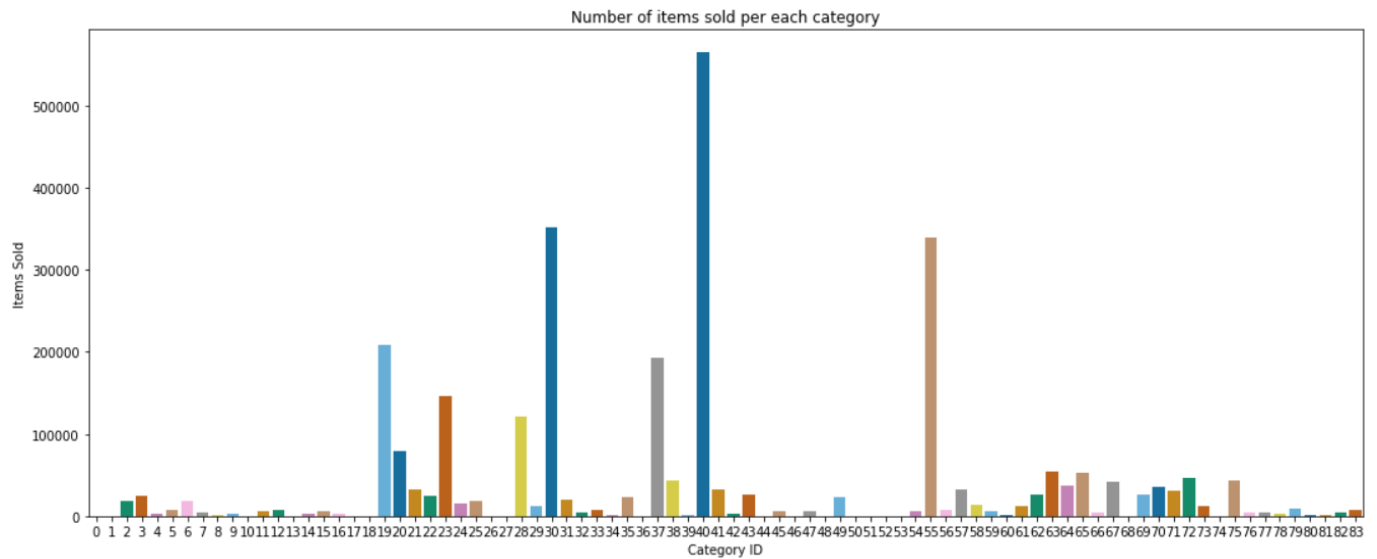
6.5 - Checking number of unique shops and unique items in train and test data.

The number of shops and items are significantly lower in test data than in train data. This could be a problem as we have to make predictions given a shop_id and item_id. After investigating we found two things :

- There are 363 unique items in the test dataset that are not represented in the train dataset. This also means that we don't have a price for these items.
- We have all unique shops of test dataset present in our train dataset.

This gives an intuition that we can use categories of items to make some predictions for these missing items.

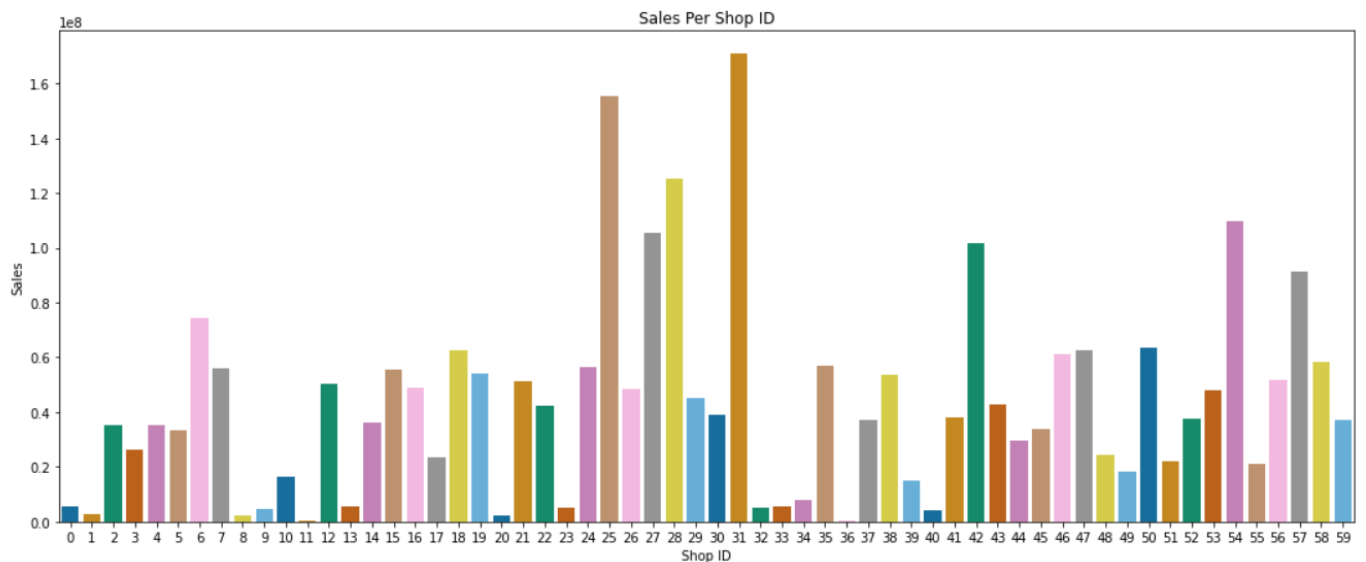
6.6 - Plotting number of items sold per each category



.Most of the categories have uniform sales.

- Category ID 40 has registered most sales
- Category ID 30 and 55 also have sales greater than other categories.

6.7 - Plotting total sales per shop ID



It is clear from the plot that shop_id = 31 has registered most sales followed by shop_id = 25.

7. Data Preprocessing and Feature Extraction

The test set consists of combinations of 42 shops and 5100 items = 214200 pairs. Since the task is to make a monthly prediction, so to make a training set which approximates the test set, we first created a training dataframe with every combination of item and shop for each month and then aggregated the train data to a monthly level.

We also added a Monthly Revenue Feature to accurately calculate prices after aggregation.

For 363 new items in test data, we didn't have sales data.. Hence, we replaced revenue and item_cnt_day as 0 for all NaN values..

Following, we concatenated the test data of the 34th month to the training dataframe and merged it with other datasets. We also dropped item_name, shop_name and item_category_name columns from respective dataframe as we won't be able to extract much useful features in order to predict sales given shop_id and item_id.

The final dataset looked like this.

	shop_id	item_id	date_block_num	item_cnt_month	revenue	item_category_id
0	59	22154	0	1.0	999.0	37
1	59	2552	0	0.0	0.0	58
2	59	2554	0	0.0	0.0	58
3	59	2555	0	0.0	0.0	56
4	59	2564	0	0.0	0.0	59
...
11127999	45	18454	34	0.0	0.0	55
11128000	45	16188	34	0.0	0.0	64
11128001	45	15757	34	0.0	0.0	55
11128002	45	19648	34	0.0	0.0	40
11128003	45	969	34	0.0	0.0	37

11128004 rows × 6 columns

8. Feature Engineering

Feature engineering plays a vital role in many of the data modeling tasks. This is simply a process that defines important features of the data using which a model can enhance its performance. In time series modeling, feature engineering works in a different way because it is sequential data and it gets formed using the changes in any values according to the time. We will generate new features and their lagged columns.

8.1 - First step is to create a function that would return lag features of different lag periods of newly created features that we will see in the following steps. Also, the original newly created columns will be deleted when they are no longer needed to avoid data leakage.

- lag1 shows the value of the prior month
- lag2 shows the value two months prior
- lag3 shows the value three months prior

```
#This function creates lag features of different lag periods.
def create_lags(data, column, lags):
    temp = data[['date_block_num', 'shop_id', 'item_id', column]]
    for i in lags:
        shift = temp.copy()
        shift.columns = ['date_block_num', 'shop_id', 'item_id', column + '_lag' + str(i)]
        shift['date_block_num'] += i
        #merge
        data = pd.merge(data, shift, on=['date_block_num', 'shop_id', 'item_id'], how='left').fillna(0)
        data[column + '_lag' + str(i)] = data[column + '_lag' + str(i)].astype(np.float32)
    return data
```

8.2 - Revenue related features

- Adding revenue of each shop per month by grouping ['shop_id', 'date_block_num']
- Adding average monthly shop revenue of each shop by grouping ['shop_id', 'date_block_num']
- Adding monthly revenue of each category of item of shop for every month by grouping ['shop_id', 'item_category_id', 'date_block_num']

8.3 - Item count related features

- Item-average item count as a feature. Averaging all item_cnt_month for each item within a month by grouping ['date_block_num']
- Item-average item count as a feature. Averaging all item_cnt_month for each item within a month by grouping ['item_id', 'date_block_num']
- Shop-average item count as a feature. Averaging all item_cnt_month for each shop within a month by grouping ['item_category_id', 'date_block_num']
- ShopCat-average item count as a feature. Averaging all item_cnt_month for each shop and category within a month by grouping ['shop_id', 'item_category_id', 'date_block_num']

8.4 - Price related features

- Adding average item price for a whole time frame of 34 months by grouping ['item_id']
- Adding monthly average item price by grouping ['item_id','date_block_num']
- Adding monthly average item price for every shop by grouping ['item_id','shop_id','date_block_num']

8.5 - Date-time related features

- Extracting Day, month , year separately from date timestamp by converting date column into pandas datetime object.

8.6 - Extracting Lags of these columns

- Lags of all the numeric features were calculated for the past 1-3 months as stated above in the first step.

We then combined all the features generated in above steps and created a dataframe with every combination of (month, shop, and item) of increasing month order.

```
data.columns

Index(['date_block_num', 'shop_id', 'item_id', 'item_cnt_month', 'revenue',
      'item_category_id', 'item_cnt_month_lag1', 'item_cnt_month_lag2',
      'item_cnt_month_lag3', 'revenue_shop_lag1', 'revenue_shop_lag3',
      'revenue_shop_avg_lag1', 'revenue_shop_cat_lag1',
      'revenue_shop_cat_lag2', 'revenue_shop_cat_lag3',
      'item_cnt_month_Avg_lag1', 'item_cnt_month_ItemAvg_lag1',
      'item_cnt_month_ItemAvg_lag2', 'item_cnt_month_ItemAvg_lag3',
      'item_cnt_month_ShopAvg_lag1', 'item_cnt_month_ShopAvg_lag2',
      'item_cnt_month_ShopAvg_lag3', 'item_cnt_month_CatAvg_lag1',
      'item_cnt_month_ShopCatAvg_lag1', 'item_overall_price_avg_lag1',
      'item_overall_price_avg_lag3', 'item_monthly_price_avg_lag1',
      'item_monthly_price_avg_lag2', 'item_shop_monthly_price_avg_lag1',
      'item_shop_monthly_price_avg_lag2', 'day', 'month', 'year'],
      dtype='object')
```

After that we deleted the first 3 months' data instances from the dataframe since they don't have lag values and saved this dataframe in the disk by using pickle module.

```
[ ] data = data[data["date_block_num"] > 2]
    data.fillna(0, inplace=True)
    data.to_pickle('/content/drive/MyDrive/Project 27/Pickle files/data_for_model.pkl')
```

9. Experiments (Different Models)

We concluded the data engineering part by splitting the data sets into training set and validation set and testing set, where we split the first 32 months for training and the 33rd month data was sliced out for validation and the last month for testing.

9.1 - Linear Regression Model

Linear Regression is one of the starting algorithm that we chose. Linear Regression modelling is usually the baseline model for every regression task. It is mostly used for finding out the relationship between variables and forecasting.

We used scikit learn to import the linear regression model. We fitted the model on the training data. Now, let us see the Intercept value and Coefficient values for all 32 columns.

```
[ ] #Importing libraries
    from sklearn.linear_model import LinearRegression
    from sklearn.metrics import mean_squared_error

    #Training Linear Regression Model on X_train and Y_train
    LR = LinearRegression()
    LR.fit(X_train, Y_train)
```

```
LinearRegression()
```

```
#Lets see the Intercept value and Coefficient values for all 32 columns
print("Intercept value is {}".format(LR.intercept_))
print("Coefficient Value is {}".format(LR.coef_))
```

```
Intercept value is 0.9921002293162108
Coefficient Value is [-1.43757198e-03  3.09553029e-03 -2.51180905e-06  2.13788656e-04
 -5.73812596e-03  2.68324417e-01  6.94745624e-02  1.40604949e-01
 -1.19739523e+05 -3.27427367e-08  1.19739523e+05 -2.95928036e-06
 -2.47438460e-07 -2.17444250e-07 -6.83553194e-01  2.06838642e-02
 -1.08785203e-02 -7.87104343e-03  5.87634195e-01 -2.80245934e-01
 -2.27762337e-01  3.62860049e-02  4.64223531e-01 -2.24060352e-03
  3.03987190e-04  1.28346045e-03  6.00705586e-04 -1.49718725e-03
 -3.01841628e-04  1.54881878e-02  2.31555384e-01  7.33677285e-04]
```

Then we predicted cross validation values and evaluated Root Mean Square Error of CV prediction and learned that we got RMSE of 4.171224909663225. Then we predicted test values and clipped them in the range 0 to 20 as it was required while submitting the result in kaggle. In kaggle, we got a score of 1.1668 which is a decent score as a baseline model.

YOUR RECENT SUBMISSION



LinearRegression_1.csv

Submitted by Bhawik Jain · Submitted just now

Score: 1.16668

Private score:

↓ Jump to your leaderboard position

Now, before going to the next model, In regression analysis, the features are estimated using coefficients while modelling. Also, if the estimates can be restricted, or shrunk or regularized towards zero, then the impact of insignificant features might be reduced and would prevent models from high variance with a stable fit. In simple words, it avoids overfitting by penalizing the regression coefficients of high value.

Regularization is the most used technique as it enhances the performance of models for new inputs. There are various regularization techniques, some well-known techniques are L1 and L2.

9.2 - Linear Regression Model with L1 Regularization.

It adds an L1 penalty that is equal to the absolute value of the magnitude of coefficient, or simply restricting the size of coefficients. For example, Lasso regression implements this method.


We imported Lasso from `sklearn.linear_model` and then fitted the train data with a different value of `alpha`, which is a hyper parameter. `Alpha` is called the regularization parameter and `alpha > 0` is manually tuned. We found the best `alpha` for Lasso by iterating the `alpha` values between 0.1 to 1 and fitting the train data and calculating the RMSE of the cross validation set. The best value of `alpha` is 0.9.

```
#Now that we have found the best alpha value, let's train the model
L1 = Lasso(alpha=best_alpha).fit(X_train,Y_train)
#Predict cross-validation values
Y_cv_pred= L1.predict(X_cv)
#Evaluating Root Mean Square Error of CV prediction
rmse = mean_squared_error(Y_cv, Y_cv_pred,squared=False)
print('RMSE : ', rmse)
```

```
RMSE : 4.04280560015707
```

After getting RMSE of 4.04 for cross validation dataset, we predict our Test values and submit the result on kaggle.

YOUR RECENT SUBMISSION



LinearRegression_with_L1.csv

Submitted by Bhawik Jain · Submitted just now

Score: 1.15104
Private score:

↓

Jump to your leaderboard position

With this model, we got a score of 1.15104 which is better than the baseline linear regression without any regularization.

9.3 - Linear Regression Model with L2 Regularization.


It adds an L2 penalty which is equal to the square of the magnitude of coefficients. For example, Ridge regression and SVM implement this method. But we will use Ridge regression as it is widely used.

L2 regularization can deal with the multicollinearity (independent variables are highly correlated) problems through constricting the coefficient and by keeping all the variables.

We imported Ridge from `sklearn.linear_model` and then fitted the train data with a different value of alpha, which is a hyper parameter. Alpha is called the same regularization parameter but plays a different role and $\alpha > 0$ is manually tuned. We found the best alpha for Ridge by iterating the alpha values between 1 to 100 and fitting the train data and calculating the RMSE of the cross validation set. The best value of alpha is 99. Then we predicted the test set.

With this model, we got a score of 1.16746.

YOUR RECENT SUBMISSION



LinearRegression_with_L2.csv

Submitted by Bhawik Jain · Submitted just now

Score: 1.16746
Private score:

↓

Jump to your leaderboard position

9.4 - Random Forest Regressor

The random forest regression algorithm takes advantage of the 'wisdom of the crowds'. It takes multiple (but different) regression decision trees and makes them 'vote'. Each tree needs to predict the expected price of the real estate based on the decision criteria it picked. Random forest regression then calculates the average of all of the predictions to generate a great estimate of what the expected price for a real estate should be.

We will use the sklearn module for training our random forest regression model, specifically the RandomForestRegressor function. The RandomForestRegressor has various different parameters that we can select for our model. But with limited time and computational resources, we will only hypertune 'max_depth' and 'n_estimators' with RandomSearchCV. We take a random sample of data with fraction 0.10 because RAM of google colab is getting full resulting in 'session expired' and crashing.

```
#Loading the actual data that we saved previously again.
data = pd.read_pickle('/content/drive/MyDrive/Project 27/Pickle files/data_for_model.pkl')

X_test = data[data.date_block_num == 34].drop(['item_cnt_month'], axis=1)

#Taking a random sample of data with fraction 0.10 because RAM of google colab is getting full resulting in 'session expired'.
new_data = data[data.date_block_num < 34]
X_train = new_data[data.date_block_num < 34].drop(['item_cnt_month'], axis=1)
Y_train = new_data[data.date_block_num < 34]['item_cnt_month']
sample_data = new_data.sample(frac=0.10)
X_train_sample = sample_data[sample_data.date_block_num < 34].drop(['item_cnt_month'], axis=1)
Y_train_sample = sample_data[sample_data.date_block_num < 34]['item_cnt_month']
```

Training and Tuning hyperparameters

Now, let us take both the parameters and set different values for hyperparameter tuning.

- max_depth — this sets the maximum possible depth of each tree
- n_estimators — the number of decision trees you will be running in the model

```
#Let's set our Parameters for Random Search
parameters = {'max_depth' : [5,6],
              'n_estimators' : [10,50,100]
              }
```

```
#Model fitting
Random_Search_CV.fit(X_train_sample, Y_train_sample)

Fitting 2 folds for each of 5 candidates, totalling 10 fits
RandomizedSearchCV(cv=2, estimator=RandomForestRegressor(), n_iter=5, n_jobs=-1,
                  param_distributions={'max_depth': [5, 6],
                                      'n_estimators': [10, 50, 100]},
                  return_train_score=True, verbose=4)
```

```
print(" Randomized Search CV Result ")
print("\n Best estimator:\n",Random_Search_CV.best_estimator_)
print("\n Best score:\n",Random_Search_CV.best_score_)
print("\n Best parameters:\n",Random_Search_CV.best_params_)
```

```
Randomized Search CV Result


Best estimator:
RandomForestRegressor(max_depth=6)

Best score:
1.0

Best parameters:
{'n_estimators': 100, 'max_depth': 6}
```

Since we got out the best parameters, let us now fit the model on train data and predict the test values and submit on kaggle.

YOUR RECENT SUBMISSION



RFR.csv

Submitted by Bhawik Jain · Submitted just now

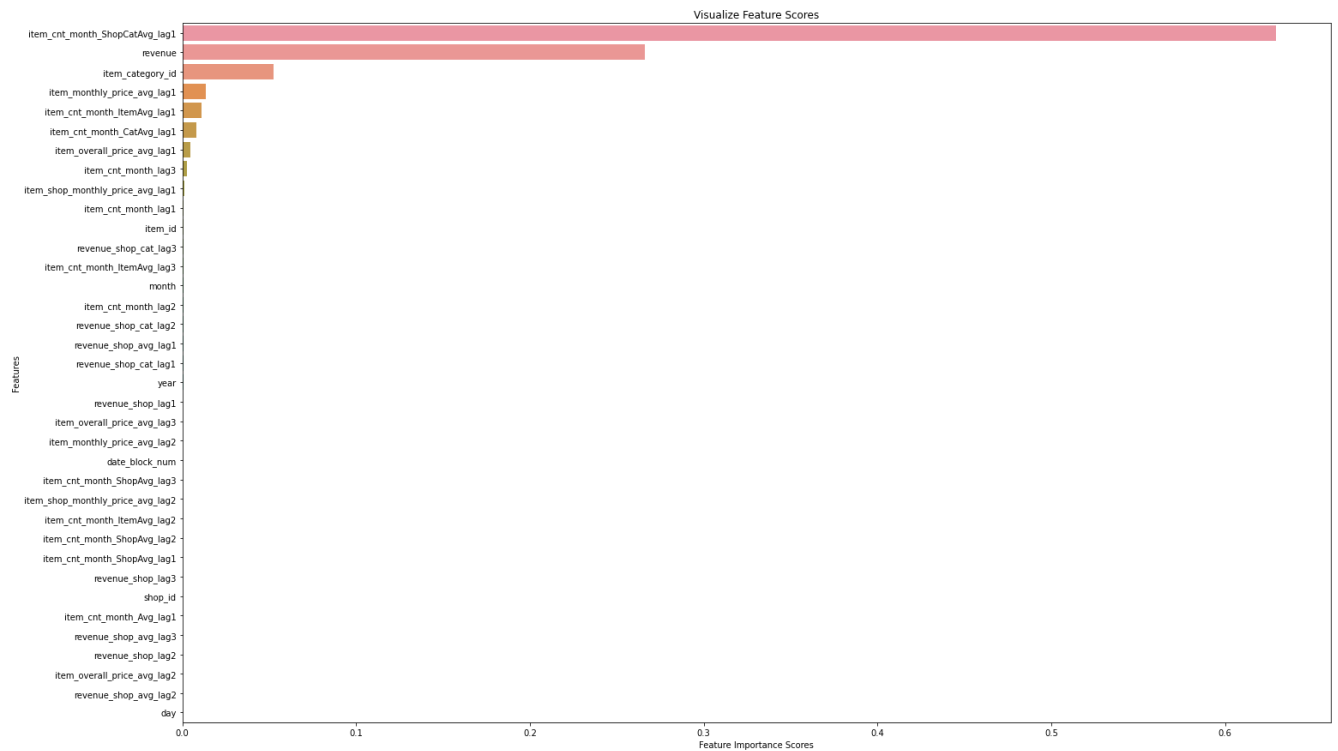
Score: 1.20513

Private score:

↓ Jump to your leaderboard position

We got a score of 1.20513 with Random Forest Regressor. It is comparatively high because we were unable to fully train our model and even with 2 parameters, we took 0.10 fraction of sample.

Feature Importance



- We can clearly see that the model found some features like `item_cnt_month_ShopCatAvg_lag1`, `revenue`, `item_category_id` very useful, in trying to predict the target.
- While some features like `day`, `item_cnt_month_Avg_lag1`, `revenue_shop_avg_lag3`, `revenue_shop_lag2`, `item_overall_price_avg_lag2`, `revenue_shop_avg_lag2` were not very useful.

9.5 - eXtreme Gradient Boosting Regressor

The last and advanced model that we are going to use is the eXtreme Gradient Boosting Regressor aka XGBRegressor. It is particularly popular because it has been the winning algorithm in several recent Kaggle competitions. It is an ensemble learning method that will create a final model by combining several weak models.

As described in the name, gradient boosting machine use gradient descent and boosting method. Boosting method adopts the iterative procedure to adaptively change the distribution of training data by focusing more on previously misclassified records to

build the base learners. It can be considered as adding the models sequentially until no further improvement is made. A gradient descent algorithm is used to minimize the loss. In gradient boosting where the predictions of multiple models are combined the gradient is used to optimize the boosted model prediction in each boosting round.

XGBoost is a special implementation of a gradient boosting machine that uses more accurate approximations to find the best model. It improves upon gradient boosting machine framework through systems optimization and algorithmic enhancements. Some of its improvements are computing second-order gradients which need fewer steps to coverage to the optimum and regularization terms which improve model generalization.

It also handles different types of sparsity patterns in the data efficiently. Now, let's train the model using train data and tune hyperparameters using the validation set.

```
#Loading the data we saved previously
data = pd.read_pickle('/content/drive/MyDrive/Project 27/Pickle files/data_for_model.pkl')

#Splitting the data into Train , cross-validation, test dataset. We are using first 33 months for training.
X_train = data[data.date_block_num < 33].drop(['item_cnt_month'], axis=1)
Y_train = data[data.date_block_num < 33]['item_cnt_month']
X_cv = data[data.date_block_num == 33].drop(['item_cnt_month'], axis=1)
Y_cv = data[data.date_block_num == 33]['item_cnt_month']
X_test = data[data.date_block_num == 34].drop(['item_cnt_month'], axis=1)

#Deleting data that we loaded as it has no use now.
del data
```

Training and Tuning hyperparameters

We created a function for hyperparameter tuning with different parameters. I have chosen 6 of the hyperparameters that are usually having a high impact on performance; “n_estimators”, “max_depth”, “min_child_weight”, “subsample”, “cosample_bytree”, and “alpha”. The evaluation metric used to train the model is the root mean squared error (RMSE).

- n_estimators: corresponds to the maximum number of boosting rounds or number of gradient boosted trees to build. Usually set it to a large value and together with early_stopping_rounds threshold to find the optimal number of rounds before reaching it. A set of 2 candidate values {500, 1000}

- eta : parameter that control the learning rate. If the learning rate is high, it is difficult to converge. On the other hand, if the learning rate is too low it will take more boosting round to converge. A set of 2 candidate values {0.1, 0.3}
- max_depth: maximum tree depth for base learners. The deeper the tree, the more complex the model. However, if the tree is too deep, the split becomes less relevant and will cause the model to overfit. A set of 3 candidate values {5, 8, 10}
- min_child_weight: corresponds to a minimum number of instances needed to be in each node. A smaller min_child_weight will lead to a child node with fewer samples and thus allowing a more complex tree. A set of 3 candidate values {3, 5, 10}
- subsample: subsampling ratio of the training instances. It will occur once in every boosting iteration. Subsample ratio = 0.5 means that the algorithm would randomly sample half of the training data prior to growing trees. A set of 2 candidate values {0.5, 0.6}
- colsample_bytree: subsample ratio of columns or features when constructing each tree. Similarly, it occurs once in every boosting iteration. Colsample_bytree = 1 means that we will use all features. A set of 2 candidate values {0.5, 0.6}

```

def HyperParaTuning(X_train, y_train):
    parameters = {
        'learning_rate': [0.1, 0.3],
        'max_depth': [5, 8, 10],
        'min_child_weight': [3, 5, 10],
        'subsample': [0.5, 0.6],
        'colsample_bytree': [0.5, 0.6],
        'n_estimators': [500, 1000],
        'objective': ['reg:squarederror']
    }

    model = XGBRegressor()

    Grid_Search_CV = GridSearchCV(estimator = model,
                                   param_grid = parameters,
                                   scoring = 'neg_mean_squared_error',
                                   cv = 2,
                                   verbose = 4)

    Grid_Search_CV.fit(X_train, Y_train)

    return Grid_Search_CV.best_params_

HyperParaTuning(X_train, Y_train)

```

The total combinations using the above hyperparameters will need to run $2 \times 3 \times 3 \times 2 \times 2 \times 2 = 360$ different models. With limited time and computational resources, I have decided to run these hyperparameters in pairs, i.e. first performed hyperparameter tuning using a combination of `max_depth` and `min_child_weight`, second performed using combinations of `subsample` and `cosample_bytree`, third performed on `eta`, and finally performed on `n_estimators`.


The table below enumerates the hyperparameters we have tuned over the validation set using `GridSearchCV`.

Parameters	Optimal Value
eta	0.1
n_estimators	1000
max_depth	10
min_child_weight	5
subsample	0.6
cosample_bytree	0.6

Table

Now, after fitting on a train dataset with optimal values of all hyperparameters, we predict the Test set and submit the results on kaggle.

YOUR RECENT SUBMISSION



xgb_submission.csv

Submitted by Bhawik Jain · Submitted 4 minutes ago

Score: 0.91600

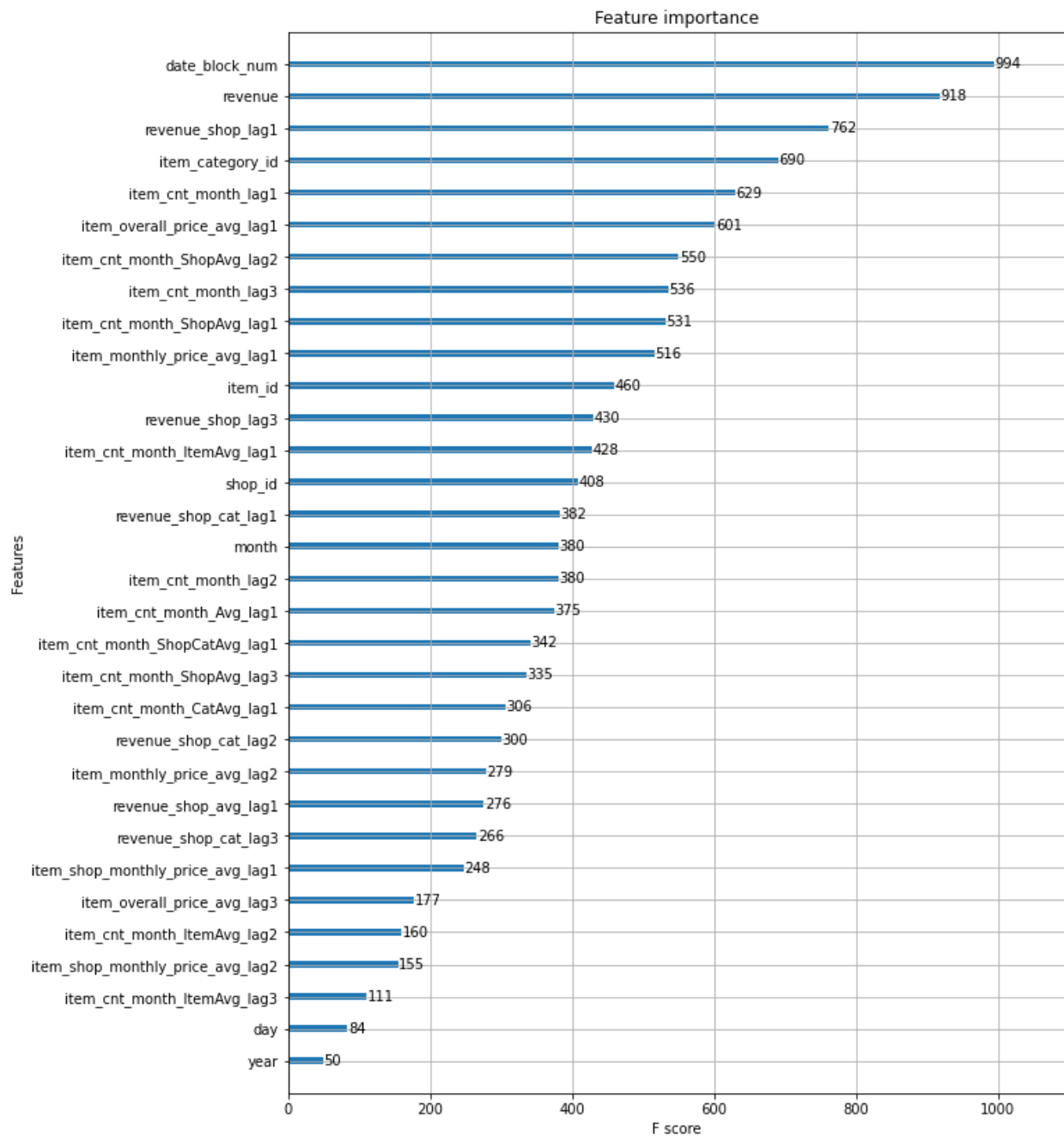
Private score:

↓

Jump to your leaderboard position

XGBRegressor GAVE THE BEST SCORE TILL NOW : 0.91600

Feature Importance



We can see that lag features have contributed fairly well and the top 5 features are date_block_num, revenue, revenue_shop_lag_1, item_category_id and item_cnt_month_lag1.

10. Deployment and Productionization

Now we will deploy the whole machine learning pipeline into a production system, into a real-time scenario. This is done in order to put research available to end users for use. The model will be deployed in a real world scenario where it takes raw input from the real world and predicts the output. Logistic Regression, Random Forest Regressor , XGBRegressor were trained using the trained dataset and as we have seen, XGBoostRegressor gives the best performance. We will develop our web application using flask and deploy it on AWS EC2.

10.1 - Developing Web Application

We will use Flask for developing our web application using python, implemented on Werkzeug and Jinja2. Flask is an open- source python web framework for building web apps for Machine Learning and Data Science This means Flask provides us with tools, libraries and technologies that will allow us to build a web application. Flask allows you to write an app the same way you write a python code. The file will be stored in .py format.

- We started by importing the Flask class.
- We then make an instance of this class. The ‘__name__’ argument is passed which is the name of the application’s module or package. Flask needs this to know where to look for resources like templates and static files.
- The route() decorator is used to inform Flask which URL should activate method.
- This method returns the message that should be shown in the user’s browser.
- POST Method is used to send data to a server to update or create a resource.

```
1 from flask import Flask, request, render_template
2 import pickle
3 import xgboost
4 import pandas as pd
5
6 app = Flask(__name__)
7
8 # Loading our model
9 with open('XGBR.pkl', 'rb') as file:
10     model = pickle.load(file)
11 # Loading our dataset
12 data = pickle.load(open('Test_data.pkl', 'rb'))
13
14
15 @app.route('/')
16 def home():...
17
18
19
20 @app.route('/predict', methods=['POST'])
21 def predict():...
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36 if __name__ == "__main__":
37     app.run(host='0.0.0.0', port=8080)
```

We will create a web app with the title as Predict Future Sales and ask the user to input Shop ID and Item ID for which we have to predict sales for. This is done in the index.html file. We have also made a style.css file that describes how HTML elements are to be displayed on screen.

```
<body>
  <div class="main">
    <h1>Predict Future Sales<br></h1>
    <!-- Main Input For Receiving Query to our ML -->
    <form action="{ url_for('predict') }}" method="post">
      Enter SHOP ID<input type="text" name="SHOPID" placeholder="Example : 5" required="required" /><br>
      Enter ITEM ID<input type="text" name="ITEMID" placeholder="Example : 5037" required="required" /><br>
      <button type="submit" class="btn btn-primary btn-block btn-large">PREDICT</button>
    </form>
    <br>
    <h5>
      <a href="https://www.kaggle.com/competitions/competitive-data-science-predict-future-sales/data?select=test.csv"
        target="_blank" color="red">Click here to know every SHOP ID and ITEM ID</a>
    </h5><br><br>
    {{ prediction_text }}
  </div>
</body>
```

After receiving the input from the user, we then check if it is in our test dataset or not. If it is not present, we will simply tell the user to enter a valid shop id and item id and if it is present, we will predict the next month's sale by the XGBRegressor model that we have made.

```
@app.route('/predict', methods=['POST'])
def predict():
    shop = int(request.form.get("SHOPID"))
    item = int(request.form.get("ITEMID"))
    query = data.loc[(data['shop_id'] == shop) & (data['item_id'] == item)]
    if query.empty:
        return render_template('index.html', prediction_text='Please enter valid SHOP ID and ITEM ID.')
    else:
        prediction = model.predict(query)
        output = round(prediction[0], 3)

        return render_template('index.html', prediction_text='Predicted next month sale for SHOP ID : {}'
                                ' and ITEM ID : {} is {}'.format(shop, item, output))
```

Let's run this and it will start a built-in server, which is used for testing but probably not suitable for production usage, our console showed the following output.

```
Run: app x
"C:\Users\Bhawik Jain\AppData\Local\Programs\Python\Python38\python.exe" "C:\Users\Bhawik Jain\Desktop\Predict Future
[10:13:47] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is
* Serving Flask app 'app'
* Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on all addresses (0.0.0.0)
* Running on http://127.0.0.1:8080
* Running on http://192.168.1.11:8080
```

Predict Future Sales

Enter SHOP ID

Enter ITEM ID

[Click here to know every SHOP ID and ITEM ID](#)

PREDICT

Once we add valid Shop Id and Item Id and click on the predict button, our model will be called, and we get the model prediction for the input data as output.

Predict Future Sales

Enter SHOP ID

5

Enter ITEM ID

5037

PREDICT

[Click here to know every SHOP ID and ITEM ID](#)

Predicted next month sale for SHOP ID : 5
and ITEM ID : 5037 is
0.1509999930858612

10.2 - Deployment

Deploying machine learning model using AWS EC2 :

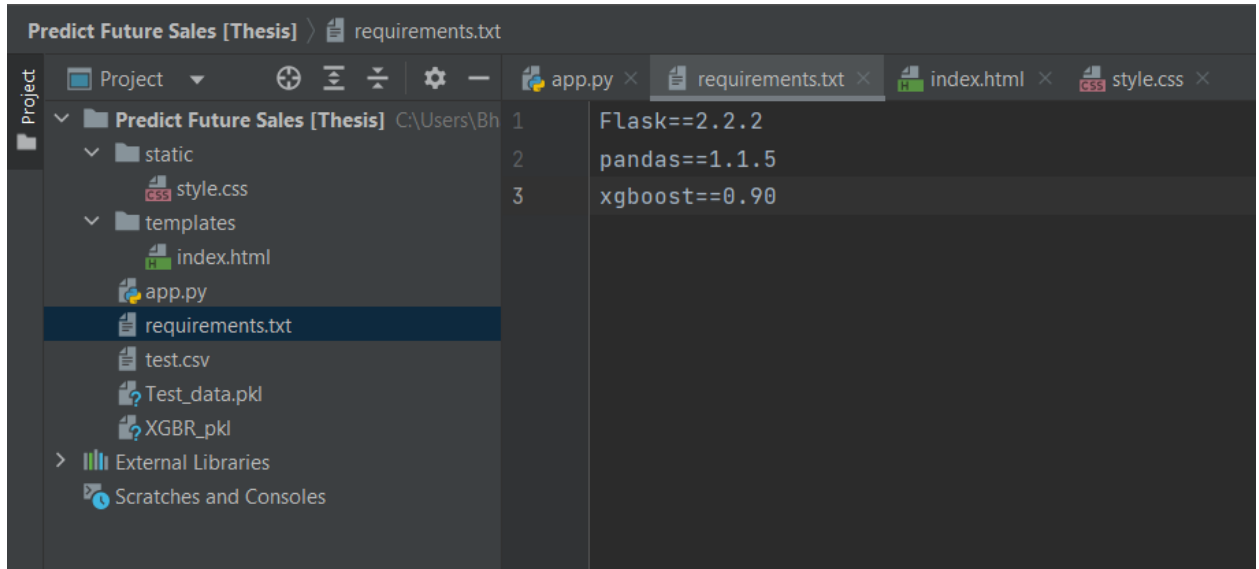
Deploying over the internet increases the accessibility of your application. After deployment applications can be accessed from mobile, computer from anywhere in the world.

Amazon Web Services(AWS) is a cloud service from Amazon, which provides services in the form of building blocks, these building blocks can be used to create and deploy any type of application in the cloud. These services or building blocks are designed to work with each other, and result in applications which are sophisticated and highly scalable.

Now let us see how to deploy an ML model app in Amazon EC2 service.

- As we have built our model on our local machine, we will add dependencies all move all files into a folder.

Adding a requirement.txt file for python dependencies.



- We will now create an account on AWS and login to the console to launch a micro instance on AWS. Following are the configurations that we have selected to launch instance.

Number of instances [Info](#)

1

[Software Image \(AMI\)](#)

Ubuntu Server 18.04 LTS (HVM), SSD Volume Type
ami-01ebac1e76247c6f2

[Virtual server type \(instance type\)](#)

t2.micro

[Firewall \(security group\)](#)

New security group

[Storage \(volumes\)](#)

1 volume(s) - 8 GiB

Cancel [Launch instance](#)

- Connect to the AWS instance.

Instances (1/1) Info								
<input type="text" value="Find instance by attribute or tag (case-sensitive)"/> < 1 >								
<input checked="" type="checkbox"/>	Name ▾	Instance ID	Instance state ▾	Instance type ▾	Status check	Alarm status	Availability Zone ▾	Public IPv4 D
<input checked="" type="checkbox"/>	Predict Future ...	i-09e721fa5efab82b2	Running	t2.micro	2/2 checks passed	No alarms +	ap-northeast-1c	ec2-54-199-1

As we can see that Instance state is in running mode which means that our instance is launched successfully. Let us connect by clicking on the Connect button at the top.

Connect to instance [Info](#)

Connect to your instance i-09e721fa5efab82b2 (Predict Future Sales) using any of these options

EC2 Instance Connect

Session Manager

SSH client

EC2 serial console

Instance ID

i-09e721fa5efab82b2 (Predict Future Sales)

1. Open an SSH client.
2. Locate your private key file. The key used to launch this instance is ec2-passkey.pem
3. Run this command, if necessary, to ensure your key is not publicly viewable.
 - `chmod 400 ec2-passkey.pem`
4. Connect to your instance using its Public DNS:
 - `ec2-54-199-118-213.ap-northeast-1.compute.amazonaws.com`

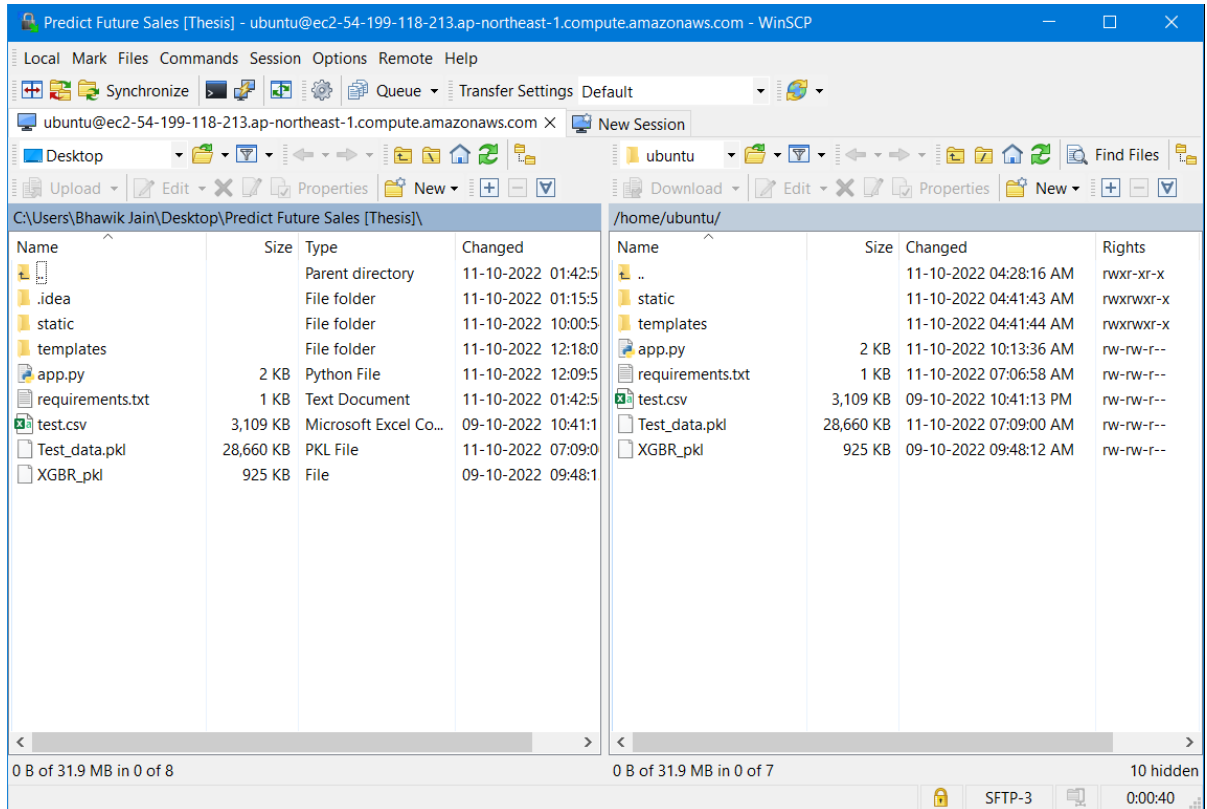
Example:

`ssh -i "ec2-passkey.pem" ubuntu@ec2-54-199-118-213.ap-northeast-1.compute.amazonaws.com`

Note: In most cases, the guessed user name is correct. However, read your AMI usage instructions to check if the AMI owner has changed the default AMI user name.

- Move the files to an AWS-EC2 instance

We have used WinSCP to transfer files. It is a free and open-source SSH File Transfer Protocol and secure copy protocol client for Microsoft Windows. Its main function is secure file transfer between a local computer and a remote server.



- Installing all packages needed on the AWS instance.

PutTY is used to interact with the server directly. Putty is just a command line interface to your server. We install python, and all the packages and modules in the amazon ec2 instance.

- Running app.py on the AWS box.

```
ubuntu@ip-172-31-9-89: ~
[screen is terminating]
ubuntu@ip-172-31-9-89:~$ python3 app.py
[08:48:04] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
* Serving Flask app 'app' (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: off
* Running on all addresses.
  WARNING: This is a development server. Do not use it in a production deployment.
* Running on http://172.31.9.89:8080/ (Press CTRL+C to quit)
223.190.80.121 - - [11/Oct/2022 08:48:06] "GET / HTTP/1.1" 200 -
223.190.80.121 - - [11/Oct/2022 08:48:06] "GET /static/style.css HTTP/1.1" 304 -
```

- Checking the output in the browser.

Not secure | ec2-54-199-118-213.ap-northeast-1.compute.amazonaws.com:8080

Predict Future Sales

Enter SHOP ID
Example : 5

Enter ITEM ID
Example : 5037

PREDICT

[Click here to know every SHOP ID and ITEM ID](#)

URL for accessing the app is :

<https://ec2-54-199-118-213.ap-northeast-1.compute.amazonaws.com:8080>

11. References

- <https://www.kaggle.com/competitions/competitive-data-science-predict-future-sales/data>
- <https://towardsdatascience.com/5-machine-learning-techniques-for-sales-forecasting-598e4984b109>
- <https://arxiv.org/pdf/2008.07779.pdf>
- <https://analyticsindiamag.com/a-guide-to-different-evaluation-metrics-for-time-series-forecasting-models/>
- <https://www.linkedin.com/learning/sales-forecasting-2022>
- <https://www.kaggle.com/code/dlarionov/feature-engineering-xgboost>
- <https://www.kaggle.com/code/kyakovlev/1st-place-solution-part-1-hands-on-data-notebook>