xgboost-for-sales-forecasting

April 22, 2024

1 XGBoost

50 2013 ~2017 2018
One-Hot-Encoding

[ ]:
```
import xgboost as xgb
import seaborn as sns
import numpy as np # linear algebra
import matplotlib.pyplot as plt
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
from matplotlib.pyplot import figure
```

[ ]:
```
train_data = pd.read_csv("/kaggle/input/demand-forecasting-kernels-only/train.
 ↪csv")
test_data = pd.read_csv("/kaggle/input/demand-forecasting-kernels-only/test.
 ↪csv")
df = pd.concat([train_data,test_data],sort=False)
df.tail(10)
```

[ ]:
```
             date  store  item  sales       id
44990  2018-03-22     10    50    NaN  44990.0
44991  2018-03-23     10    50    NaN  44991.0
44992  2018-03-24     10    50    NaN  44992.0
44993  2018-03-25     10    50    NaN  44993.0
44994  2018-03-26     10    50    NaN  44994.0
44995  2018-03-27     10    50    NaN  44995.0
44996  2018-03-28     10    50    NaN  44996.0
44997  2018-03-29     10    50    NaN  44997.0
44998  2018-03-30     10    50    NaN  44998.0
44999  2018-03-31     10    50    NaN  44999.0
```

2

2.1 Descriptive Statistics

```
[ ]: #
     statistic_sheet_s = df.groupby(["store"]).agg({"sales": ["count","sum", "mean",␣
      ↪"median", "std", "min", "max"]})
     statistic_sheet_s.head(2)
```

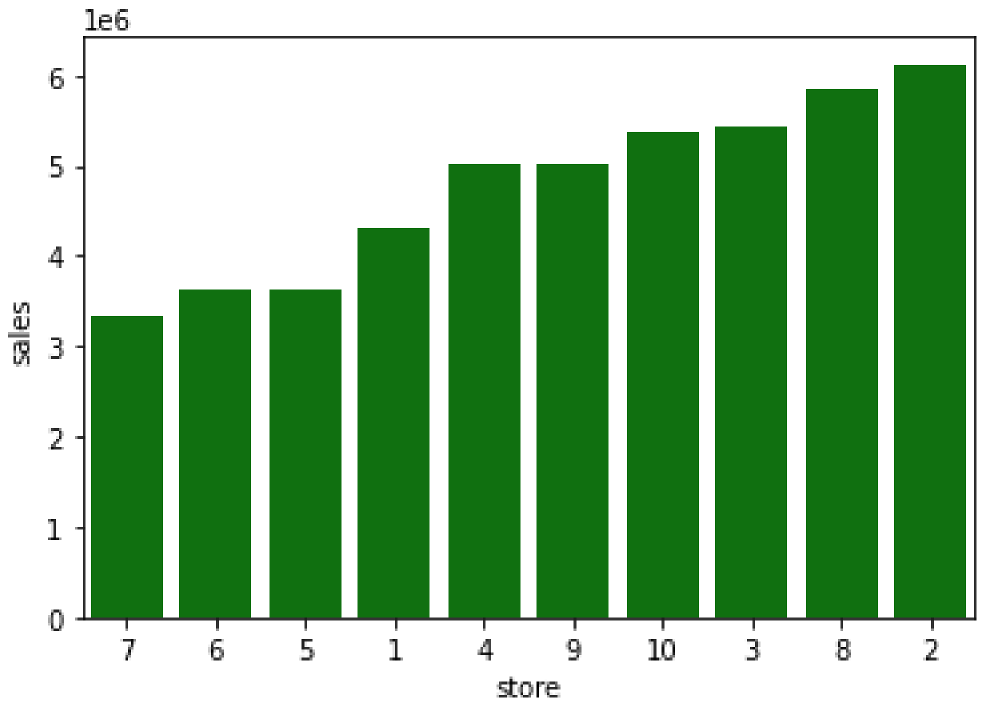
```
[ ]:        sales
            count        sum       mean median        std  min    max
     store
     1      91300  4315603.0  47.268379   44.0  24.006252  1.0  155.0
     2      91300  6120128.0  67.033165   62.0  33.595810  3.0  231.0
```

```
[ ]: #
     statistic_sheet_i = df.groupby(["item"]).agg({"sales": ["count","sum", "mean",␣
      ↪"median", "std", "min", "max"]})
     statistic_sheet_i.head(2)
```

```
[ ]:       sales
           count        sum       mean median        std  min    max
     item
     1     18260   401384.0  21.981599   21.0   8.468922  1.0   59.0
     2     18260  1069564.0  58.574151   56.0  20.093015  9.0  150.0
```

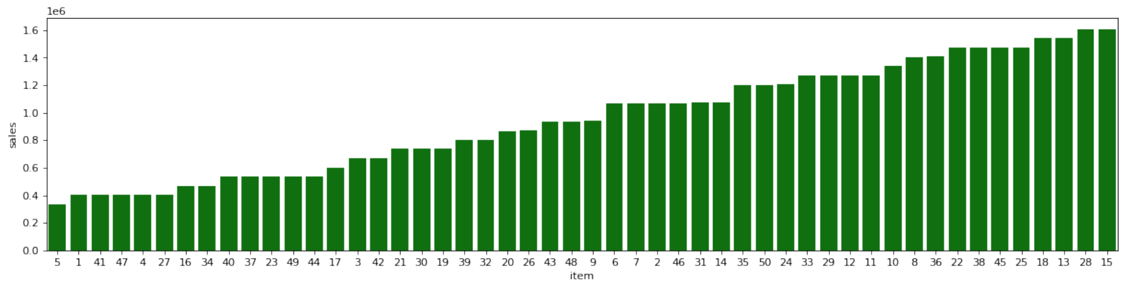
```
[ ]: #
     stores_sum = df.groupby(["store"],as_index=False).agg({"sales": "sum"}).
      ↪sort_values(by="sales",ascending=False)
     sns.barplot(data=stores_sum,x='store',y='sales',color="green",order=stores_sum.
      ↪sort_values('sales').store)
```

```
[ ]: <AxesSubplot:xlabel='store', ylabel='sales'>
```



[ ]:
```
#
figure(figsize=(18, 4), dpi=80)
item_sum = df.groupby(["item"],as_index=False).agg({"sales": "sum"}).
↪sort_values(by="sales",ascending=False)
sns.barplot(data=item_sum,x='item',y='sales',color="green",order=item_sum.
↪sort_values('sales').item)
```

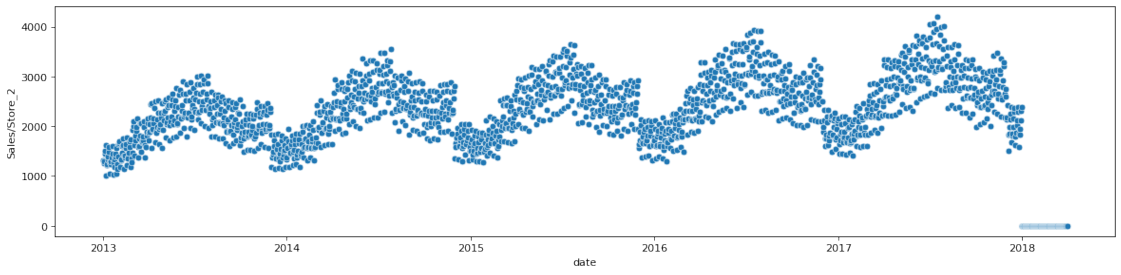
[ ]:
```
<AxesSubplot:xlabel='item', ylabel='sales'>
```



[ ]:
```
#     ,
figure(figsize=(18, 4), dpi=80)
store_daily = df.groupby(["date","store"],as_index=False).agg({"sales":"sum"})
```

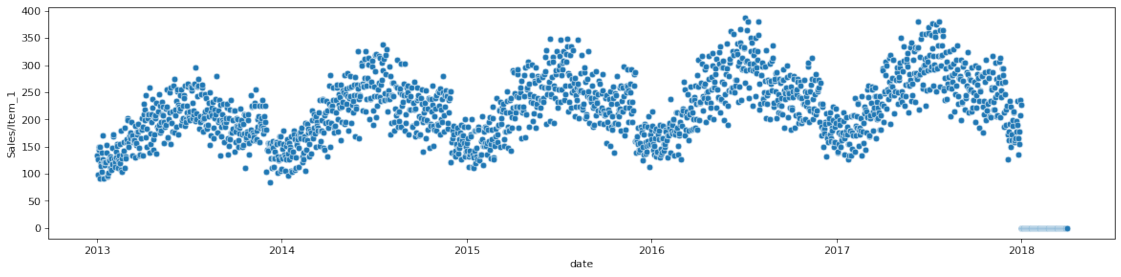
```
#
store_daily['date'] = pd.to_datetime(store_daily.date, format='%Y/%m/%d')
store_1 = store_daily[store_daily['store']==1]
ax_1 = sns.scatterplot(data=store_1,x='date',y='sales')
ax_1.set_ylabel("Sales/Store_2")
```

```
[ ]: Text(0, 0.5, 'Sales/Store_2')
```



```
[ ]: #     ,
figure(figsize=(18, 4), dpi=80)
item_daily = df.groupby(["date","item"],as_index=False).agg({"sales":"sum"})
#
item_daily['date'] = pd.to_datetime(item_daily.date, format='%Y/%m/%d')
item_1 = item_daily[item_daily['item']==1]
ax_2 = sns.scatterplot(data=item_1,x='date',y='sales')
ax_2.set_ylabel("Sales/Item_1")
```

```
[ ]: Text(0, 0.5, 'Sales/Item_1')
```



3

3.1 Feature Engineering

```
[ ]: def generate_timeline_features(data):
         data = data.copy()
         date_format = pd.to_datetime(data.date)
         data['year'] = date_format.dt.year
         data['month'] = date_format.dt.month
         #    Q1-Q4
         data['quarter'] = date_format.dt.quarter
         #
         data['dayofweek'] = date_format.dt.dayofweek
         data['dayofyear'] = date_format.dt.dayofyear
         #    0: Winter  - 1: Spring  - 2: Summer  - 3: Fall
         data["season"] = np.where(data.month.isin([12,1,2]), 0, 1)
         data["season"] = np.where(data.month.isin([6,7,8]), 2, data["season"])
         data["season"] = np.where(data.month.isin([9, 10, 11]), 3, data["season"])
         return data
     new_df = generate_timeline_features(df)
```

```
[ ]: new_df.groupby([ "year", "month","store", "item"]).agg({"sales": ["sum",␣
      ↪"mean", "median", "std"]}).tail(5)
```

```
[ ]:                       sales
                             sum mean median std
     year month store item
     2018 3     10    46     0.0  NaN     NaN NaN
                      47     0.0  NaN     NaN NaN
                      48     0.0  NaN     NaN NaN
                      49     0.0  NaN     NaN NaN
                      50     0.0  NaN     NaN NaN
```

Lag features

```
[ ]: # lags   = [91, 98, 105, 112, 180, 270, 365, 546, 728]
     lags   = [91, 180, 365, 546]
     def lag_features(df, lags):
         for lag in lags:
             value = df.groupby(["store", "item"])['sales'].transform(lambda x: x.
      ↪shift(lag))
             df['sales_lag_' + str(lag)] = value
         return df

     new_df= lag_features(new_df, lags)
```

4

4.1 Rolling Mean Features

```
[ ]: def roll_mean_features(df, windows):
         for window in windows:
             value = df.groupby(["store", "item"])['sales'].transform(lambda x: x.
       ↪shift(1).rolling(window=window, min_periods=10, win_type="triang").mean())
             df['sales_roll_mean_' + str(window)] = value
         return df
     new_df= roll_mean_features(new_df, [365, 546])
```

5

5.1 Exponentially Weighted Mean Features

```
[ ]: def ewm_features(dataframe, alphas, lags):
         for alpha in alphas:
             for lag in lags:
                 dataframe['sales_ewm_alpha_' + str(alpha).replace(".", "") +␣
       ↪"_lag_" + str(lag)] = \
                     dataframe.groupby(["store", "item"])['sales'].transform(lambda␣
       ↪x: x.shift(lag).ewm(alpha=alpha).mean())
         return dataframe
     alphas = [0.9, 0.8, 0.7, 0.5]
     new_df= ewm_features(new_df, alphas, lags)
     new_df.tail(2)
```

```
[ ]:              date  store  item  sales       id  year  month  quarter  \
     44998  2018-03-30     10    50    NaN  44998.0  2018      3        1
     44999  2018-03-31     10    50    NaN  44999.0  2018      3        1

            dayofweek  dayofyear  …  sales_ewm_alpha_08_lag_365  \
     44998          4         89  …                   68.550876
     44999          5         90  …                   68.910175

            sales_ewm_alpha_08_lag_546  sales_ewm_alpha_07_lag_91  \
     44998                   82.314892                  69.403475
     44999                   94.062978                  64.221042

            sales_ewm_alpha_07_lag_180  sales_ewm_alpha_07_lag_365  \
     44998                   98.791375                   68.602440
     44999                   79.337413                   68.880732

            sales_ewm_alpha_07_lag_546  sales_ewm_alpha_05_lag_91  \
     44998                   83.485707                  66.038719
     44999                   92.945712                  64.019360
```

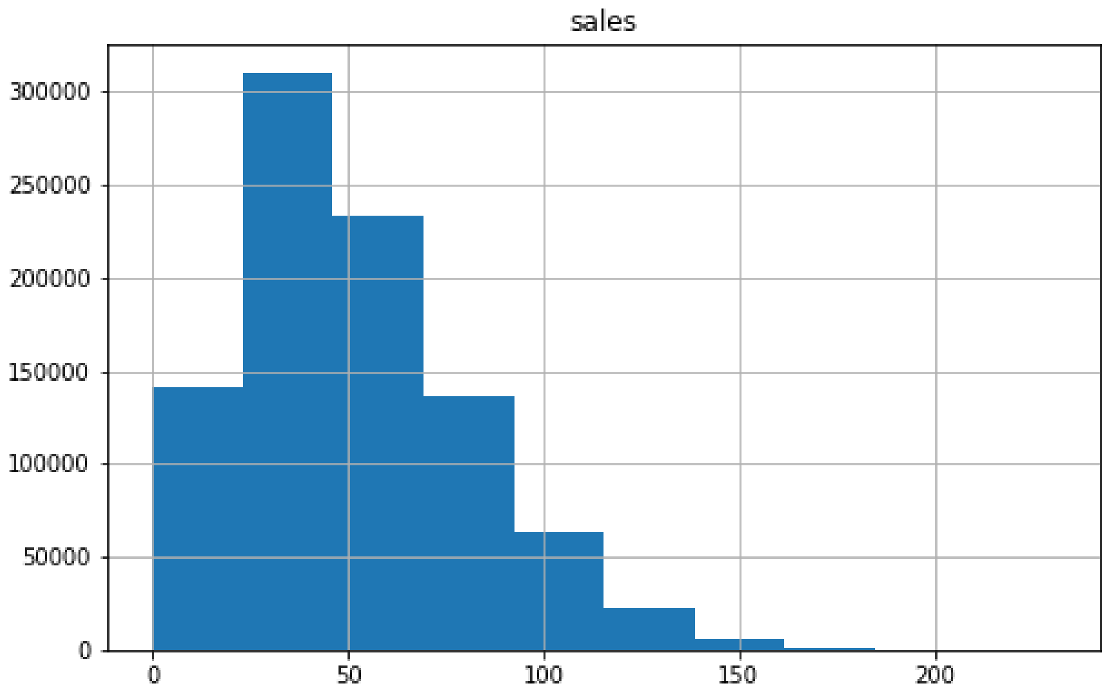
```
       sales_ewm_alpha_05_lag_180  sales_ewm_alpha_05_lag_365  \
44998                   96.603586                   68.716870
44999                   83.801793                   68.858435

       sales_ewm_alpha_05_lag_546
44998                   84.936127
44999                   90.968063

[2 rows x 33 columns]
```

6

6.1 Logarithmic Transformation for Sales Data

[ ]:
```
new_df.hist('sales',figsize=(8,5))
```

[ ]:
```
array([[<AxesSubplot:title={'center':'sales'}>]], dtype=object)
```

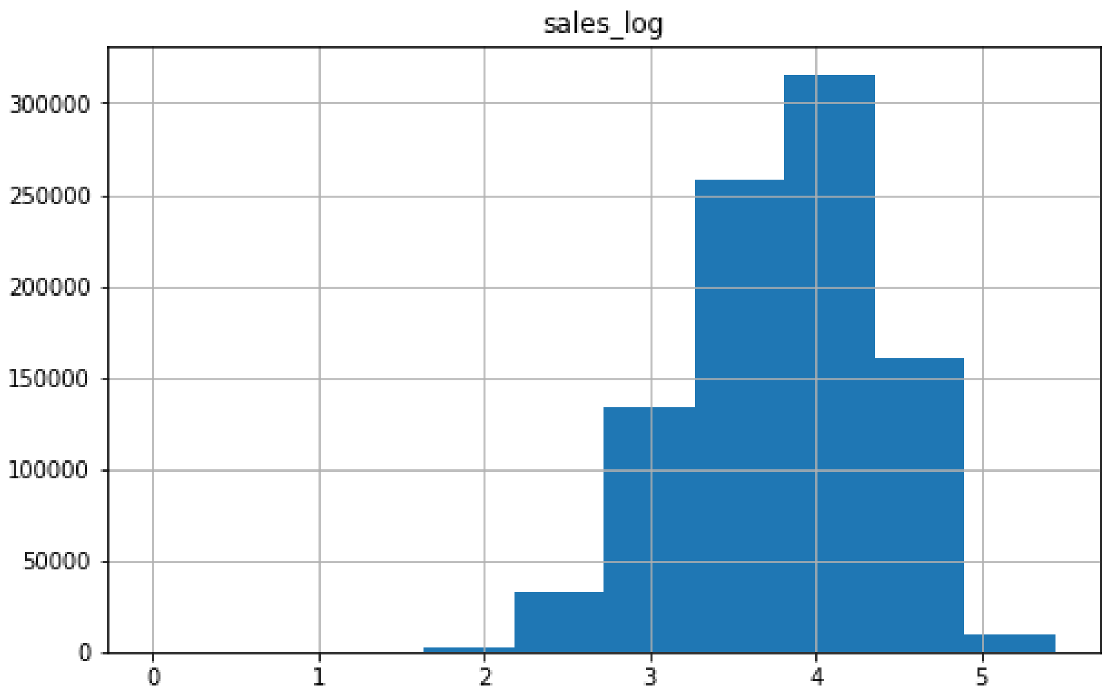


50

[ ]:
```
#
new_df['sales_log']= np.log1p(new_df["sales"].values)
```

```
new_df.hist('sales_log',figsize=(8,5))
```

[ ]:
```
array([[<AxesSubplot:title={'center':'sales_log'}>]], dtype=object)
```



7

[ ]:
```
#                        --One-Hot-Encoding
#    store 10  50   7  4   4  6   12
#                 1
# status_features = ['store', 'item', 'dayofweek', "quarter", 'month', "year",␣
↪"season"]
status_features = ['store', 'item', 'dayofweek', "season"]
#
df_discrete = pd.get_dummies(new_df, columns=status_features)
#
tem = df_discrete.copy()
#
df_discrete_sorted = tem.sort_values("date").reset_index(drop = True)
#  2017
```

```
train_df = df_discrete_sorted.loc[(df_discrete_sorted["date"] < "2015-01-01"), :
 ↪]
#  2017
valid_df = df_discrete_sorted.loc[(df_discrete_sorted["date"] >= "2015-01-01")&␣
 ↪(df_discrete_sorted["date"] < "2015-04-01"),:]
#
cols = [col for col in train_df.columns if col not in ['date', 'id', "sales",␣
 ↪"year"]]
#
X_train = train_df[cols]
#
X_valid = valid_df[cols]
#
y_train = train_df['sales']
#
y_valid = valid_df['sales']
```

8 XGBoost

1. **Explained Variance Score** 0-1, 1 2. **Mean Absolute Error(MAE)** 3. **R2 Score** Explained Variance Score 4. **Roor Mean Squared Error RMSE**

[ ]:
```
from sklearn.metrics import explained_variance_score,mean_absolute_error,␣
 ↪mean_squared_error, r2_score

# SMAPE: Symmetric mean absolute percentage error (adjusted MAPE)
def smape(preds, target):
    n = len(preds)
    masked_arr = ~((preds == 0) & (target == 0))
    preds, target = preds[masked_arr], target[masked_arr]
    num = np.abs(preds-target)
    denom = np.abs(preds)+np.abs(target)
    smape_val = (200*np.sum(num/denom))/n
    return smape_val

def xgb_smape(y_pred, y_true):
    smape_val = smape(np.expm1(preds), np.expm1(y_true))
    return 'SMAPE', smape_val, False
#
xgb_model= xgb.XGBRegressor()
#
first_model= xgb_model.fit(X_train, y_train,
                           eval_metric= lambda y_pred, y_true: [xgb_smape(y_pred,␣
 ↪y_true)])
```

```
print("VALID SMAPE:", smape(np.expm1(first_model.predict(X_valid)), np.
  ↪expm1(y_valid)))

#print("\tExplained variance:", explained_variance_score(y_valid, first_model.
  ↪predict(X_valid)))
print("\tMean absolute error (MAE):", mean_absolute_error(y_valid, first_model.
  ↪predict(X_valid)))
#print("\tRoot Mean squared error (RMSE):",  np.
  ↪sqrt(mean_squared_error(y_valid, first_model.predict(X_valid))))
#print("\tR2 score:", r2_score(y_valid, first_model.predict(X_valid)))
```

```
/opt/conda/lib/python3.7/site-packages/xgboost/sklearn.py:797: UserWarning:
`eval_metric` in `fit` method is deprecated for better compatibility with
scikit-learn, use `eval_metric` in constructor or`set_params` instead.
  UserWarning,
```

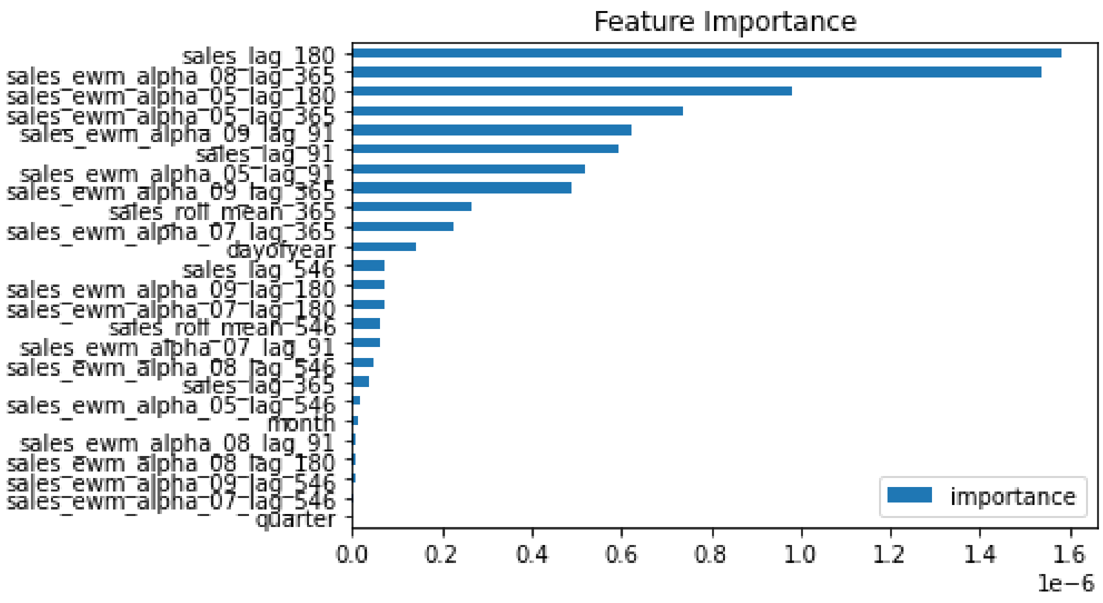
```
VALID SMAPE: 0.005818615765527391
        Mean absolute error (MAE): 7.355786429511176e-05
```

```
/opt/conda/lib/python3.7/site-packages/ipykernel_launcher.py:22: RuntimeWarning:
overflow encountered in expm1
```

9 XGBoost

[ ]:
```
feature_importance= pd.DataFrame(data = first_model.feature_importances_,
                 index  = first_model.feature_names_in_,
                 columns= ['importance'])
feature_importance.sort_values('importance', ascending=False).head(25)
figure(figsize=(28, 24), dpi=80)
feature_importance.head(25).sort_values('importance').plot(kind='barh',␣
  ↪title='Feature Importance')
plt.legend(loc='lower right')
plt.show()
```

```
<Figure size 2240x1920 with 0 Axes>
```



10

10.1 Predication for Sales

```
[ ]: from time import time

     data= df_discrete_sorted.copy()
     # Sales
     train = data.loc[~data.sales.isna()]
     # Sales
     test = data.loc[data.sales.isna()]
     X_train = train[cols]
     X_test = test[cols]

     Y_train = train['sales']

     start = time()
     xgb_params= {"colsample_bytree": 0.3,
                  "learning_rate": 0.1,
                  "max_depth": 3,
                  "n_estimators": 100,
                  "verbose": 30,
                  "num_boost_round": xgb_model.best_iteration}

     xgbtrain_all= xgb.DMatrix(data=X_train, label=Y_train)

     test_model= xgb.train(xgb_params, xgbtrain_all,
                           num_boost_round=xgb_model.best_iteration)

     train_time = time() - start
     start = time()
     test_preds = test_model.predict(xgb.DMatrix(X_test))
     predict_time = time()-start
     test_preds
```

```
[15:51:32] WARNING: ../src/learner.cc:627:
Parameters: { "n_estimators", "num_boost_round", "verbose" } might not be used.

  This could be a false alarm, with some parameters getting used by language
bindings but
  then being mistakenly passed down to XGBoost core, or some parameter actually
being used
  but getting flagged wrongly here. Please open an issue if you find any such
cases.
```

```
[ ]: array([16.07758 , 28.052837, 14.076222, …, 45.081093, 15.289815,
            41.9718  ], dtype=float32)
```

11

11.1 Submmition

Kaggle

```
[ ]: #     id sales
     submission_df = test.loc[:, ['id', 'sales']]
     #
     submission_df['sales'] = np.expm1(test_preds)
     # id
     submission_df['id'] = submission_df.id.astype(int)
     submission_df.to_csv('submission.csv',index=False)
```