Import libraries

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings

from scipy.stats import skew,norm,zscore
from scipy.signal import periodogram

from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from statsmodels.tsa.deterministic import DeterministicProcess, CalendarFourier

from sklearn.model_selection import train_test_split, cross_val_score, TimeSeriesSplit, GridSearchCV, cross_validate
from sklearn.metrics import mean_squared_error, make_scorer, mean_squared_log_error, mean_absolute_error, mean_absolute_percentag
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import Ridge
from sklearn.neural_network import MLPRegressor
from sklearn.decomposition import PCA
from random import shuffle
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.metrics import mean_squared_log_error as msle
from tqdm import tqdm

from xgboost import XGBRegressor
from lightgbm import LGBMRegressor
from catboost import Pool, CatBoostRegressor

import optuna
```

Loading the dataset

In [2]:

```
holidays_events = pd.read_csv("/kaggle/input/store-sales-time-series-forecasting/holidays_events.csv", parse_dates=['date'])
oil = pd.read_csv("/kaggle/input/store-sales-time-series-forecasting/oil.csv", parse_dates=['date'])
stores = pd.read_csv("/kaggle/input/store-sales-time-series-forecasting/stores.csv")
transactions = pd.read_csv("/kaggle/input/store-sales-time-series-forecasting/transactions.csv", parse_dates=['date'])

test = pd.read_csv("/kaggle/input/store-sales-time-series-forecasting/test.csv", parse_dates=['date'])
train = pd.read_csv("/kaggle/input/store-sales-time-series-forecasting/train.csv", parse_dates=['date'])
```

In [3]:

```
important_dates = {
    'train_start_date': '2013-01-01',
    'train_end_date': '2017-08-15',
    'test_start_date': '2017-08-16',
    'test_end_date': '2017-08-31',
    'forest_start_date': '2016-06-01'
}
```

Prepare Data

In [4]:

```
def add_store_details(main_df, train, test):
    df = main_df.copy()

    df['uniquestore'] = df.city.apply(lambda x: 0 if x in ['Quito', 'Guayaquil', 'Santo Domingo', 'Cuenca', 'Manta', 'Machala', '
    df['newstore'] = df.store_nbr.apply(lambda x: 1 if x in [19, 20, 21, 28, 35, 41, 51, 52] else 0)

    df = pd.concat([train, test], axis=0).merge(df, on=['store_nbr'], how='left')
    df = df.rename(columns={'type' : 'store'})

    return df
```

In [5]:

```
final_df = add_store_details(stores, train, test)
final_df
```

Out[5]:

	id	date	store_nbr	family	sales	onpromotion	city	state	store	cluster	uniquestore	newstore
0	0	2013-01-01	1	AUTOMOTIVE	0.0	0	Quito	Pichincha	D	13	0	0
1	1	2013-01-01	1	BABY CARE	0.0	0	Quito	Pichincha	D	13	0	0
2	2	2013-01-01	1	BEAUTY	0.0	0	Quito	Pichincha	D	13	0	0
3	3	2013-01-01	1	BEVERAGES	0.0	0	Quito	Pichincha	D	13	0	0
4	4	2013-01-01	1	BOOKS	0.0	0	Quito	Pichincha	D	13	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...
3029395	3029395	2017-08-31	9	POULTRY	NaN	1	Quito	Pichincha	B	6	0	0
3029396	3029396	2017-08-31	9	PREPARED FOODS	NaN	0	Quito	Pichincha	B	6	0	0
3029397	3029397	2017-08-31	9	PRODUCE	NaN	1	Quito	Pichincha	B	6	0	0
3029398	3029398	2017-08-31	9	SCHOOL AND OFFICE SUPPLIES	NaN	9	Quito	Pichincha	B	6	0	0
3029399	3029399	2017-08-31	9	SEAFOOD	NaN	0	Quito	Pichincha	B	6	0	0

3029400 rows × 12 columns

In [6]:

```
def add_holiday_details(main_df):

    df = main_df.copy()

    df.loc[297, 'transferred'] = df.loc[297, 'transferred'] = False
    df = df.query("transferred!=True")

    df = df.drop(index=holidays_events[holidays_events[['date', 'locale_name']].duplicated()].index.values)

    df.loc[df.type=='Event', 'type'] = df.description.apply(lambda x: x[0:7])

    nat_df = df.query("locale=='National'")
    loc_df = df.query("locale=='Local'")
    reg_df = df.query("locale=='Regional'")

    df = final_df.merge(nat_df, left_on=['date'], right_on=['date'], how='left')
    df = df.merge(loc_df, left_on=['date', 'city'], right_on=['date', 'locale_name'], how='left')
    df = df.merge(reg_df, left_on=['date', 'state'], right_on=['date', 'locale_name'], how='left')

    df['firstday'] = df.description_x.apply(lambda x: 1 if x=='Primer dia del ano' else 0)

    df = df.drop(columns=['locale_x', 'locale_name_x', 'description_x', 'transferred_x',
                          'locale_y', 'locale_name_y', 'description_y', 'transferred_y',
                          'locale', 'locale_name', 'description', 'transferred'])
    df.loc[~df.type_x.isnull(), 'event_type'] = df.type_x.apply(lambda x: x)
    df.loc[~df.type_y.isnull(), 'event_type'] = df.type_y.apply(lambda x: x)
    df.loc[~df.type.isnull(), 'event_type'] = df.type.apply(lambda x: x)
    df.loc[df.event_type.isnull(), 'event_type'] = df.event_type.apply(lambda x: 'norm')
    df = df.drop(columns=['type_x', 'type_y', 'type'])

    df['isevent'] = df.event_type.apply(lambda x: 'y' if x!='norm' else 'n')

    df.loc[df.date.isin(['2017-04-16', '2016-03-27', '2015-04-05', '2014-04-20', '2013-03-31']), 'isevent'] = df.isevent.apply(la
    df.loc[df.date.isin(['2017-04-16', '2016-03-27', '2015-04-05', '2014-04-20', '2013-03-31']), 'event_type'] = df.event_type.ap

    df['isclosed'] = df.groupby(by=['date', 'store_nbr'])['sales'].transform(lambda x: 1 if x.sum()==0 else 0)
    df.loc[(df.date.dt.year==2017) & (df.date.dt.month==8) & (df.date.dt.day>=16) , 'isclosed'] = df.isclosed.apply(lambda x: 0)
    df.loc[df.date.isin(['2017-01-01']), 'isevent'] = df.isevent.apply(lambda x: 'n')

    return df
```

In [7]:

```
final_df = add_holiday_details(holidays_events)
final_df
```

Out[7]:

	id	date	store_nbr	family	sales	onpromotion	city	state	store	cluster	uniquestore	newstore	firstday	event_t
0	0	2013-01-01	1	AUTOMOTIVE	0.0	0	Quito	Pichincha	D	13	0	0	1	Holi
1	1	2013-01-01	1	BABY CARE	0.0	0	Quito	Pichincha	D	13	0	0	1	Holi
2	2	2013-01-01	1	BEAUTY	0.0	0	Quito	Pichincha	D	13	0	0	1	Holi
3	3	2013-01-01	1	BEVERAGES	0.0	0	Quito	Pichincha	D	13	0	0	1	Holi
4	4	2013-01-01	1	BOOKS	0.0	0	Quito	Pichincha	D	13	0	0	1	Holi
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
3029395	3029395	2017-08-31	9	POULTRY	NaN	1	Quito	Pichincha	B	6	0	0	0	n
3029396	3029396	2017-08-31	9	PREPARED FOODS	NaN	0	Quito	Pichincha	B	6	0	0	0	n
3029397	3029397	2017-08-31	9	PRODUCE	NaN	1	Quito	Pichincha	B	6	0	0	0	n
3029398	3029398	2017-08-31	9	SCHOOL AND OFFICE SUPPLIES	NaN	9	Quito	Pichincha	B	6	0	0	0	n
3029399	3029399	2017-08-31	9	SEAFOOD	NaN	0	Quito	Pichincha	B	6	0	0	0	n

3029400 rows × 16 columns

In [8]:

```
print(f"Total null entries in oil: \n{oil.set_index('date').resample('D').mean().isnull().sum()}")
```

```
Total null entries in oil:
dcoilwtico    529
dtype: int64
```

In [9]:

```
def add_oil_details(main_df):

    df = main_df.copy()

    df = df.set_index('date').resample("D").mean().interpolate(limit_direction='backward').reset_index()

    for i in [1, 2, 3, 4, 5, 6, 7, 10, 14, 21, 30, 60, 90]:
        df['lagoil_' + str(i) + '_dcoilwtico'] = df['dcoilwtico'].shift(i)

    df['oil_week_avg'] = df['dcoilwtico'].rolling(7).mean()
    df['oil_2weeks_avg'] = df['dcoilwtico'].rolling(14).mean()
    df['oil_month_avg'] = df['dcoilwtico'].rolling(30).mean()

    df.dropna(inplace = True)

    df = final_df.merge(df, on=['date'], how='left')

    return df
```

In [10]:

```
final_df = add_oil_details(oil)
final_df
```

Out[10]:

	id	date	store_nbr	family	sales	onpromotion	city	state	store	cluster	...	lagoil_7_dcoilwtico	lagoil_10_dcoilwt
0	0	2013-01-01	1	AUTOMOTIVE	0.0	0	Quito	Pichincha	D	13	...	NaN	N
1	1	2013-01-01	1	BABY CARE	0.0	0	Quito	Pichincha	D	13	...	NaN	N
2	2	2013-01-01	1	BEAUTY	0.0	0	Quito	Pichincha	D	13	...	NaN	N
3	3	2013-01-01	1	BEVERAGES	0.0	0	Quito	Pichincha	D	13	...	NaN	N
4	4	2013-01-01	1	BOOKS	0.0	0	Quito	Pichincha	D	13	...	NaN	N
...	...	...	...	...	...	...	...	...	...	...	...	...	
3029395	3029395	2017-08-31	9	POULTRY	NaN	1	Quito	Pichincha	B	6	...	47.24	47
3029396	3029396	2017-08-31	9	PREPARED FOODS	NaN	0	Quito	Pichincha	B	6	...	47.24	47
3029397	3029397	2017-08-31	9	PRODUCE	NaN	1	Quito	Pichincha	B	6	...	47.24	47
3029398	3029398	2017-08-31	9	SCHOOL AND OFFICE SUPPLIES	NaN	9	Quito	Pichincha	B	6	...	47.24	47
3029399	3029399	2017-08-31	9	SEAFOOD	NaN	0	Quito	Pichincha	B	6	...	47.24	47

3029400 rows × 33 columns

In [11]:

```
def add_transaction_details(main_df):

    df = main_df.copy()

    df = final_df.merge(df, on=['date', 'store_nbr'], how='left')

    df.loc[(df.transactions.isnull()) & (df.isclosed==1), 'transactions'] = df.transactions.apply(lambda x: 0)
    group_df = df.groupby(by=['store_nbr', 'date']).transactions.first().reset_index()
    group_df['avg_tra'] = group_df.transactions.rolling(15, min_periods=10).mean()
    group_df['16_tra'] = group_df.transactions.shift(16)
    group_df['21_tra'] = group_df.transactions.shift(21)
    group_df['30_tra'] = group_df.transactions.shift(30)
    group_df['60_tra'] = group_df.transactions.shift(60)
    group_df.drop(columns='transactions', inplace=True)
    df = df.merge(group_df, on=['date', 'store_nbr'], how='left')
    df.loc[(df.transactions.isnull()) & (df.isclosed==0), 'transactions'] = df.avg_tra
    df.drop(columns='avg_tra', inplace=True)
    df.loc[(df.date.dt.year==2017) & (df.date.dt.month==8) & (df.date.dt.day>=16) , 'transactions'] = df.transactions.apply(lambd

    df['tot_store_day_onprom'] = df.groupby(by=['date', 'store_nbr']).onpromotion.transform(lambda x: x.sum())

    return df
```

In [12]:

```
final_df = add_transaction_details(transactions)
final_df
```

Out[12]:

	id	date	store_nbr	family	sales	onpromotion	city	state	store	cluster	...	lagoil_90_dcoilwtico	oil_week_avg
0	0	2013-01-01	1	AUTOMOTIVE	0.0	0	Quito	Pichincha	D	13	...	NaN	NaN
1	1	2013-01-01	1	BABY CARE	0.0	0	Quito	Pichincha	D	13	...	NaN	NaN
2	2	2013-01-01	1	BEAUTY	0.0	0	Quito	Pichincha	D	13	...	NaN	NaN
3	3	2013-01-01	1	BEVERAGES	0.0	0	Quito	Pichincha	D	13	...	NaN	NaN
4	4	2013-01-01	1	BOOKS	0.0	0	Quito	Pichincha	D	13	...	NaN	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...	...
3029395	3029395	2017-08-31	9	POULTRY	NaN	1	Quito	Pichincha	B	6	...	47.68	46.825714
3029396	3029396	2017-08-31	9	PREPARED FOODS	NaN	0	Quito	Pichincha	B	6	...	47.68	46.825714
3029397	3029397	2017-08-31	9	PRODUCE	NaN	1	Quito	Pichincha	B	6	...	47.68	46.825714
3029398	3029398	2017-08-31	9	SCHOOL AND OFFICE SUPPLIES	NaN	9	Quito	Pichincha	B	6	...	47.68	46.825714
3029399	3029399	2017-08-31	9	SEAFOOD	NaN	0	Quito	Pichincha	B	6	...	47.68	46.825714

3029400 rows × 39 columns

In [13]:

```
def add_time_features (main_df):

    df = main_df.copy()

    df['year'] = df.index.year.astype('int')
    df['quarter'] = df.index.quarter.astype('int')
    df['month'] = df.index.month.astype('int')
    df['day'] = df.index.day.astype('int')
    df['dayofweek'] = df.index.day_of_week.astype('int')
    df['weekofyear'] = df.index.week.astype('int')
    df['isweekend'] = df.dayofweek.apply(lambda x: 1 if x in (5,6) else 0)
    df['startschool'] = df.month.apply(lambda x: 1 if x in (4,5,8,9) else 0)

    df['daysinmonth'] = df.index.days_in_month.astype('int')

    df = pd.get_dummies(df, columns=['year'], drop_first=True)
    df = pd.get_dummies(df, columns=['quarter'], drop_first=True)
    df = pd.get_dummies(df, columns=['dayofweek'], drop_first=True)
    df = pd.get_dummies(df, columns=['store'], drop_first=True)
    df = pd.get_dummies(df, columns=['event_type'], drop_first=True)
    df = pd.get_dummies(df, columns=['isevent'], drop_first=True)
    df = pd.get_dummies(df, columns=['state'], drop_first=True)

    # DeterministicProcess
    fourierA = CalendarFourier(freq='A', order=5)
    fourierM = CalendarFourier(freq='M', order=2)
    fourierW = CalendarFourier(freq='W', order=4)

    dp = DeterministicProcess(index=df.index,
                              order=1,
                              seasonal=False,
                              constant=False,
                              additional_terms=[fourierA, fourierM, fourierW],
                              drop=True)
    dp_df = dp.in_sample()
    df = pd.concat([df, dp_df], axis=1)

    df['outliers'] = df.sales.apply(lambda x: 1 if x>30000 else 0)

    df.drop(columns=['daysinmonth', 'month', 'city'], inplace=True)

    return df
```

In [14]:

```
final_df = final_df.set_index('date').loc[important_dates['forest_start_date']:,:]
final_df
```

Out[14]:

	id	store_nbr	family	sales	onpromotion	city	state	store	cluster	uniquestore	...	lagoil_90_dcoilwtico	oil_week_a
date													
2016-06-01	2216808	1	AUTOMOTIVE	3.0	0	Quito	Pichincha	D	13	0	...	34.56	49.1742
2016-06-01	2216809	1	BABY CARE	0.0	0	Quito	Pichincha	D	13	0	...	34.56	49.1742
2016-06-01	2216810	1	BEAUTY	4.0	0	Quito	Pichincha	D	13	0	...	34.56	49.1742
2016-06-01	2216811	1	BEVERAGES	2199.0	37	Quito	Pichincha	D	13	0	...	34.56	49.1742
2016-06-01	2216812	1	BOOKS	0.0	0	Quito	Pichincha	D	13	0	...	34.56	49.1742
...	...	...	...	...	...	...	...	...	...	...	...	...	
2017-08-31	3029395	9	POULTRY	NaN	1	Quito	Pichincha	B	6	0	...	47.68	46.8257
2017-08-31	3029396	9	PREPARED FOODS	NaN	0	Quito	Pichincha	B	6	0	...	47.68	46.8257
2017-08-31	3029397	9	PRODUCE	NaN	1	Quito	Pichincha	B	6	0	...	47.68	46.8257
2017-08-31	3029398	9	SCHOOL AND OFFICE SUPPLIES	NaN	9	Quito	Pichincha	B	6	0	...	47.68	46.8257
2017-08-31	3029399	9	SEAFOOD	NaN	0	Quito	Pichincha	B	6	0	...	47.68	46.8257

812592 rows × 38 columns

In [15]:

```
df = add_time_features(final_df).loc[:important_dates['test_end_date'],:].reset_index().set_index(['store_nbr', 'family', 'date']
df['16_tra'] = df['16_tra'].fillna(0)
df['21_tra'] = df['21_tra'].fillna(0)
df['30_tra'] = df['30_tra'].fillna(0)
df['60_tra'] = df['60_tra'].fillna(0)
df
```

```
/opt/conda/lib/python3.7/site-packages/ipykernel_launcher.py:10: FutureWarning: weekofyear and week have been depre
cated, please use DatetimeIndex.isocalendar().week instead, which returns a Series.  To exactly reproduce the behav
ior of week and weekofyear and return an Index, you may call pd.Int64Index(idx.isocalendar().week)
  # Remove the CWD from sys.path while we load stuff.
```

Out[15]:

			id	sales	onpromotion	cluster	uniquestore	newstore	firstday	isclosed	dcoilwtico	lagoil_1_dcoilwtico
store_nbr	**family**	**date**										
1	**AUTOMOTIVE**	**2016-06-01**	2216808	3.0	0	13	0	0	0	0	49.070000	49.100000
		2016-06-02	2218590	1.0	0	13	0	0	0	0	49.140000	49.070000
		2016-06-03	2220372	4.0	0	13	0	0	0	0	48.690000	49.140000
		2016-06-04	2222154	9.0	0	13	0	0	0	0	49.030000	48.690000
		2016-06-05	2223936	2.0	0	13	0	0	0	0	49.370000	49.030000
...	**...**	**...**	...	...	...	...	...	...	...	...	...	...
54	**SEAFOOD**	**2017-08-27**	3022139	NaN	0	3	1	0	0	0	46.816667	47.233333
		2017-08-28	3023921	NaN	0	3	1	0	0	0	46.400000	46.816667
		2017-08-29	3025703	NaN	0	3	1	0	0	0	46.460000	46.400000
		2017-08-30	3027485	NaN	0	3	1	0	0	0	45.960000	46.460000
		2017-08-31	3029267	NaN	0	3	1	0	0	0	47.260000	45.960000

812592 rows × 95 columns

Modeling

In [16]:

```
def train_val_split_function(df):
    train, val = train_test_split(df, test_size=0.1, shuffle=False)
    return [train.drop(columns=['sales']), val.drop(columns=['sales']), train['sales'], val['sales']]

def get_model(name):
    RF_param = {
        'criterion': 'squared_error',
        'bootstrap': 'False',
        'max_depth': 9733,
        'max_features': 'auto',
        'max_leaf_nodes': 4730,
        'n_estimators': 159,
        'min_samples_split': 3,
        'min_samples_leaf': 8
    }
    switcher = {
        "GB": GradientBoostingRegressor(n_estimators=20,min_samples_split=30, subsample=0.3),
        "RF": RandomForestRegressor(**RF_param, random_state=0),
        "Ridge": Ridge(alpha=0.5),
        "LR": LinearRegression(),
        "CB": CatBoostRegressor(),
        "XGB": XGBRegressor()
    }

    return switcher.get(name, switcher.get("RF"))

def get_predicitions(final_predictions):
    sample_submission = pd.read_csv('../input/store-sales-time-series-forecasting/sample_submission.csv')
    sample_submission['sales'] = sample_submission['id'].map(final_predictions)
    sample_submission['sales'] = np.exp(sample_submission['sales']) - 1

    return sample_submission
```

In [17]:

```
def train(pca=None):
    train_error = val_error = count = 0
    final_predictions = {}

    for i in tqdm(df.index.get_level_values(0).unique()):
        for j in df.index.get_level_values(1).unique():
            current_df = df.loc[(i, j)]
            test_id = current_df[current_df['sales'].isna()]['id']
            current_df = current_df.drop(columns=['id', 'transactions'])

            train = current_df[~current_df['sales'].isna()]
            X_test = current_df[current_df['sales'].isna()].drop(columns=['sales'])

            X_train, X_val, y_train, y_val = train_val_split_function(train)

            if pca:
                pca.fit(X_train)
                X_train = pca.transform(X_train)
                X_val = pca.transform(X_val)
                X_test = pca.transform(X_test)

            y_train = np.log1p(y_train)
            y_val = np.log1p(y_val)

            model = get_model("RF")
            model.fit(X_train, y_train)

            train_pred =  model.predict(X_train).clip(0.0)
            val_pred = model.predict(X_val).clip(0.0)

            train_error += msle(np.exp(y_train) - 1, (np.exp(train_pred) - 1).clip(0))
            val_error += msle(np.exp(y_val) - 1, (np.exp(val_pred) - 1).clip(0))
            count += 1

            test_preds = model.predict(X_test).clip(0.0)

            for q in range(test_preds.shape[0]):
                final_predictions[test_id[q]] = test_preds[q]

        print(f"Train Performance: {(train_error / count)**0.5}; Val Performance: {(val_error / count)**0.5}")

    return final_predictions
```

Train without PCA

In [20]:

```
predictions = train()
predictions = get_predicitions(predictions)
predictions.to_csv('/kaggle/working/without_pca.csv',index=False)
```

With PCA

In [21]:

```
predictions = train(pca=PCA(n_components=10))
predictions = get_predicitions(predictions)
predictions.to_csv('/kaggle/working/with_pca.csv',index=False)
```

```
  2%|▏         | 1/54 [00:12<11:23, 12.90s/it]

Train Performance: 0.376240411156621; Val Performance: 0.38727815021678813

  4%|▎         | 2/54 [00:25<11:09, 12.88s/it]

Train Performance: 0.3727576897575212; Val Performance: 0.37559824498430233

  6%|▌         | 3/54 [00:38<10:55, 12.85s/it]

Train Performance: 0.37124555352408073; Val Performance: 0.3660436352747978

  7%|▋         | 4/54 [00:51<10:44, 12.89s/it]

Train Performance: 0.3698769529214687; Val Performance: 0.3629608728637732

  9%|▉         | 5/54 [01:04<10:32, 12.92s/it]

Train Performance: 0.36777165090428204; Val Performance: 0.3767625829677819

 11%|█         | 6/54 [01:16<10:11, 12.74s/it]
```

With Extra features from: [artemchistyakov (https://www.kaggle.com/code/artemchistyakov/store-sales-eda-rf)](https://www.kaggle.com/code/artemchistyakov/store-sales-eda-rf)

In [22]:

```
def tags_to_dict():
    tags = {
     'AUTOMOTIVE': [4, 7, 30, 10, 'family'],
     'BABY CARE':  [-8, 2, 25, 5, 'family'],
     'BEAUTY': [-8, 7, 25, 5, 'other'],
     'BEVERAGES': [0, 0, 40, 40, 'food'],
     'BOOKS': [0, 0, 55, 15, 'other'],
     'BREAD/BAKERY': [-3, 0, 30, 30, 'food'],
     'CELEBRATION': [-5, 5, 50, 20, 'family'],
     'CLEANING': [-8, 3, 40, 20, 'food'],
     'DAIRY': [-4, 0, 40, 40, 'food'],
     'DELI': [3, 6, 40, 20, 'food'],
     'EGGS': [-4, -5, 40, 20, 'food'],
     'FROZEN FOODS': [-4, -3, 40, 20, 'food'],
     'GROCERY I': [-4, 3, 40, 20, 'food'],
     'GROCERY II': [-4, 3, 40, 20, 'food'],
     'HARDWARE': [10, 10, 30, 20, 'other'],
     'HOME AND KITCHEN I': [-10, 4, 40, 20, 'family'],
     'HOME AND KITCHEN II': [-10, 4, 40, 20, 'family'],
     'HOME APPLIANCES': [0, 4, 40, 20, 'family'],
     'HOME CARE': [-10, 4, 40, 20, 'family'],
     'LADIESWEAR': [-10, 4, 40, 20, 'other'],
     'LAWN AND GARDEN': [-10, 4, 40, 20, 'family'],
     'LINGERIE': [-10, 4, 40, 2, 'other'],
     'LIQUOR,WINE,BEER': [4, 8, 40, 20, 'food'],
     'MAGAZINES': [-6, -7, 50, 20, 'other'],
     'MEATS': [-4, 5, 40, 20, 'food'],
     'PERSONAL CARE': [-5, 5, 40, 20, 'family'],
     'PET SUPPLIES': [-5, 0, 40, 20, 'family'],
     'PLAYERS AND ELECTRONICS': [5, 5, 25, 10, 'other'],
     'POULTRY': [-7, -4, 40, 20, 'food'],
     'PREPARED FOODS': [0, 6, 30, 10, 'food'],
     'PRODUCE': [0, 0, 40, 40, 'other'],
     'SCHOOL AND OFFICE SUPPLIES': [3, 3, 25, 15, 'family'],
     'SEAFOOD': [-5, 8, 40, 20, 'food']
    }

    sex_dict = {}
    luxury_dict = {}
    age_mean_dict = {}
    age_var_dict = {}
    type_dict = {}
    for i in tags.keys():
        sex_dict[i] = tags[i][0]
        luxury_dict[i] = tags[i][1]
        age_mean_dict[i] = tags[i][2]
        age_var_dict[i] = tags[i][3]
        type_dict[i] = tags[i][4]
    return [sex_dict, luxury_dict, age_mean_dict, age_var_dict, type_dict]

def get_oil_dict(oil):
    # estimate price of gaps (market don't work on weekends and holidays)
    price_estim = [-1] * (oil['days_from_2013'][oil.shape[0] - 1] + 1)
    price_estim[0] = 93.14
    for i in range(1, oil.shape[0]):
        price_estim[oil['days_from_2013'][i]] = oil['dcoilwtico'][i]

    for i in range (len(price_estim)):
        if price_estim[i] == -1 or math.isnan(price_estim[i]):
            tj = -1
            for j in range(i + 1, len(price_estim)):
                if price_estim[j] != -1 and (not math.isnan(price_estim[j])):
                    tj = j
                    break

            for j in range(i, tj):
                price_estim[j] = ((tj - j) * price_estim[i - 1] + (j - i) * price_estim[tj]) / (tj - i)

            i = tj

    oil_dict = dict(zip(np.arange(len(price_estim)), price_estim))
    return oil_dict


import math

def add_custom_features(df):
    # read
    train = pd.read_csv('../input/store-sales-time-series-forecasting/train.csv')
    oil = pd.read_csv('../input/store-sales-time-series-forecasting/oil.csv')
    trans = pd.read_csv('../input/store-sales-time-series-forecasting/transactions.csv')

    # add 'days_from_2013' for easy shifting
    df['days_from_2013'] = (pd.to_datetime(df.index.get_level_values(2)) - pd.to_datetime('2013-01-01')).days
```

```python
    train['days_from_2013'] = (pd.to_datetime(train['date']) - pd.to_datetime('2013-01-01')).dt.days
    oil['days_from_2013'] = (pd.to_datetime(oil['date']) - pd.to_datetime('2013-01-01')).dt.days
    trans['days_from_2013'] = (pd.to_datetime(trans['date']) - pd.to_datetime('2013-01-01')).dt.days

    # groupby features
    gr_day = train.groupby('days_from_2013')['sales'].mean()
    gr_store = train.groupby('store_nbr')['sales'].mean()
    gr_family = train.groupby('family')['sales'].mean()

    days = [16, 18, 20, 21, 25, 28, 30, 35, 42, 60, 90, 120, 180, 365]
    for i in days:
        df['days_' + str(i)] = df['days_from_2013'] - i
        df['days_lagged' + str(i)] = df['days_' + str(i)].map(gr_day).fillna(0)
        df = df.drop(columns=['days_' + str(i)])

    df['store_gb'] = df.index.get_level_values(0).map(gr_store)
    df['family_gb'] = df.index.get_level_values(1).map(gr_family)

    oil_dict = get_oil_dict(oil)

    # lagged oil
    days = [0, 1, 2, 3, 4, 5, 6, 7, 10, 14, 21, 30, 60, 90, 120, 180, 360]
    for i in days:
        df['days_' + str(i)] = df['days_from_2013'] - i
        df['oil_lagged' + str(i)] = df['days_' + str(i)].map(oil_dict)
        df = df.drop(columns=['days_' + str(i)])

    # lagged transactions
    # # fill trans dict
    trans_dict = {}
    for ii in range(trans.shape[0]):
        i = trans.loc[ii]
        trans_dict[tuple([i['store_nbr'], i['days_from_2013']])] = i['transactions']

    def transaction_get_value(a, b):
        try:
            return trans_dict[tuple([a, (pd.to_datetime(b) - pd.to_datetime('2013-01-01').dt.days)])]
        except:
            return 0

    days = [16, 18, 20, 21, 25, 28, 30, 35, 42, 60, 90, 120, 180, 365]
    for i in days:
        df['days_' + str(i)] = df['days_from_2013'] - i
        df['oil_lagged' + str(i)] = df['days_' + str(i)].map(oil_dict)
        df['trans_lagged' + str(i)] = [transaction_get_value(*a) for a in tuple(zip(df.index.get_level_values(0),
                                                                  df.index.get_level_values(2)))]
        df = df.drop(columns=['days_' + str(i)])

    sex_dict, luxury_dict, age_mean_dict, age_var_dict, type_dict = tags_to_dict()
    df['tag_sex'] = df.index.get_level_values(1).map(sex_dict)
    df['tag_luxury'] = df.index.get_level_values(1).map(luxury_dict)
    df['tag_age_mean'] = df.index.get_level_values(1).map(age_mean_dict)
    df['tag_age_var'] = df.index.get_level_values(1).map(age_var_dict)
    df['tag_type'] = df.index.get_level_values(1).map(type_dict)
    df = pd.get_dummies(df, columns=['tag_type'])

    df['tag_age_min'] = df['tag_age_mean'] - df['tag_age_var']
    df['tag_age_max'] = df['tag_age_mean'] + df['tag_age_var']
    return df

def custom_split_function(main_df, train_start_date='2013-01-01', train_end_date='2017-08-30', val_start_date='2017-09-01', val_e
    train_start_date = (pd.to_datetime(train_start_date) - pd.to_datetime('2013-01-01')).days
    train_end_date = (pd.to_datetime(train_end_date) - pd.to_datetime('2013-01-01')).days
    val_start_date = (pd.to_datetime(val_start_date) - pd.to_datetime('2013-01-01')).days
    val_end_date = (pd.to_datetime(val_end_date) - pd.to_datetime('2013-01-01')).days

    train = main_df[(main_df['days_from_2013'] >= train_start_date) & (main_df['days_from_2013'] <= train_end_date)]
    val = main_df[(main_df['days_from_2013'] >= val_start_date) & (main_df['days_from_2013'] <= val_end_date)]
    return [train.drop(columns=['sales']), val.drop(columns=['sales']), train['sales'], val['sales']]

def get_weights_distribution(tp, dates):
    if tp == 1:
        return np.ones(dates.shape)
    if tp == 2:
        return np.exp((400 - (pd.to_datetime('2017-08-16') - pd.to_datetime(dates)).days) / 100)
    if tp == 3:
        return np.exp((400 - (pd.to_datetime('2017-08-16') - pd.to_datetime(dates)).days) / 200)
    if tp == 4:
        return np.exp((400 - (pd.to_datetime('2017-08-16') - pd.to_datetime(dates)).days) / 300)
    if tp == 5:
        return np.exp((400 - (pd.to_datetime('2017-08-16') - pd.to_datetime(dates)).days) / 400)
```

In [23]:

```
df = add_custom_features(df)
df
```

Out[23]:

store_nbr	family	date	id	sales	onpromotion	cluster	uniquestore	newstore	firstday	isclosed	dcoilwtico	lagoil_1_dcoilwtico
1	AUTOMOTIVE	2016-06-01	2216808	3.0	0	13	0	0	0	0	49.070000	49.100000
		2016-06-02	2218590	1.0	0	13	0	0	0	0	49.140000	49.070000
		2016-06-03	2220372	4.0	0	13	0	0	0	0	48.690000	49.140000
		2016-06-04	2222154	9.0	0	13	0	0	0	0	49.030000	48.690000
		2016-06-05	2223936	2.0	0	13	0	0	0	0	49.370000	49.030000
...	...	...	...	...	...	...	...	...	...	...	...	...
54	SEAFOOD	2017-08-27	3022139	NaN	0	3	1	0	0	0	46.816667	47.233333
		2017-08-28	3023921	NaN	0	3	1	0	0	0	46.400000	46.816667
		2017-08-29	3025703	NaN	0	3	1	0	0	0	46.460000	46.400000
		2017-08-30	3027485	NaN	0	3	1	0	0	0	45.960000	46.460000
		2017-08-31	3029267	NaN	0	3	1	0	0	0	47.260000	45.960000

812592 rows × 160 columns

In [24]:

```
def train_best():
    train_error = val_error = count = 0
    final_predictions = {}

    for i in tqdm(df.index.get_level_values(0).unique()):
        for j in df.index.get_level_values(1).unique():
            current_df = df.loc[(i, j)]
            test_id = current_df[current_df['sales'].isna()]['id']
            current_df = current_df.drop(columns=['id', 'transactions'])

            train = current_df[~current_df['sales'].isna()]
            X_test = current_df[current_df['sales'].isna()].drop(columns=['sales'])

            X_train, X_val, y_train, y_val = custom_split_function(train)

            y_train = np.log1p(y_train)
#             y_val = np.log1p(y_val)

            model = get_model("RF")
            weights = get_weights_distribution(5, X_train.index)
            model.fit(X_train, y_train, sample_weight=weights)

            train_pred =  model.predict(X_train).clip(0.0)
#             val_pred = model.predict(X_val).clip(0.0)

            train_error += msle(np.exp(y_train) - 1, (np.exp(train_pred) - 1).clip(0))
#             val_error += msle(np.exp(y_val) - 1, (np.exp(val_pred) - 1).clip(0))
            count += 1

            test_preds = model.predict(X_test).clip(0.0)

            for q in range(test_preds.shape[0]):
                final_predictions[test_id[q]] = test_preds[q]

        print(f"Train Performance: {(train_error / count)**0.5}; Val Performance: {(val_error / count)**0.5}")

    return final_predictions
```

Submission

In [25]:

```
final_preds = train_best()
best_submission = get_predicitions(final_preds)
best_submission.to_csv('/kaggle/working/best.csv', index=False)
best_submission
```

Out[25]:

	id	sales
0	3000888	3.218919
1	3000889	0.000000
2	3000890	4.031745
3	3000891	2420.409059
4	3000892	0.115015
...	...	...
28507	3029395	340.372360
28508	3029396	113.848349
28509	3029397	1272.751190
28510	3029398	116.628746
28511	3029399	13.446688

In [ ]:

In [ ]: