In [1]:

```
import numpy as np
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_log_error
from statsmodels.tsa.deterministic import DeterministicProcess
```

In [2]:

```
# path to the dataset in Kaggle's notebook
path = '../input/store-sales-time-series-forecasting/'
```

## 1. Compute Moving Average of Oil Prices

In [3]:

```
# read oil price
data_oil = pd.read_csv(path + 'oil.csv', parse_dates=['date'], infer_datetime_format=True, index_col='date')

########################################################################################################################
# TODO: compute data_oil['ma_oil'] as the moving average of data_oil['dcoilwtico'] with window size 7
# Hint: check the documentation of .rolling() method of pandas.DataFrame
########################################################################################################################
data_oil['ma_oil'] = data_oil['dcoilwtico'].rolling(7).mean()


# Create continguous moving average of oil prices
calendar = pd.DataFrame(index=pd.date_range('2013-01-01', '2017-08-31'))

########################################################################################################################
# TODO 1: merge two DataFrame instances (data_oil and calendar) such that the merged instances has the same indexes
# as calendar.
# TODO 2: replace each NaN in data_oil['ma_oil'] by the first non-null value before it.
# Hint: check the documentation of .merge() and .fillna() methods of pandas.DataFrame
########################################################################################################################
calendar = pd.merge(calendar, data_oil, left_index=True, how='left', right_index=True)
calendar.fillna(method='bfill', inplace=True)
calendar.head(15) # display some entries of calendar
```

Out[3]:

| | dcoilwtico | ma_oil |
|---|---|---|
| **2013-01-01** | 93.14 | 93.218571 |
| **2013-01-02** | 93.14 | 93.218571 |
| **2013-01-03** | 92.97 | 93.218571 |
| **2013-01-04** | 93.12 | 93.218571 |
| **2013-01-05** | 93.20 | 93.218571 |
| **2013-01-06** | 93.20 | 93.218571 |
| **2013-01-07** | 93.20 | 93.218571 |
| **2013-01-08** | 93.21 | 93.218571 |
| **2013-01-09** | 93.08 | 93.218571 |
| **2013-01-10** | 93.81 | 93.218571 |
| **2013-01-11** | 93.60 | 93.284286 |
| **2013-01-12** | 94.27 | 93.470000 |
| **2013-01-13** | 94.27 | 93.470000 |
| **2013-01-14** | 94.27 | 93.470000 |
| **2013-01-15** | 93.26 | 93.490000 |

## 2. Create Workday Feature

In [4]:

```
##########################################################################################################################
# TODO: create a True/False feature calendar['wd'] to indicate whether each date is a workday (Monday-Friday) or not.
# Hint: check documentation of pandas.DatetimeIndex.dayofweek
##########################################################################################################################
calendar['wd'] = calendar.index.day_of_week <=4

calendar.head(15) # display some entries of calendar
```

Out[4]:

| | dcoilwtico | ma_oil | wd |
|---|---|---|---|
| **2013-01-01** | 93.14 | 93.218571 | True |
| **2013-01-02** | 93.14 | 93.218571 | True |
| **2013-01-03** | 92.97 | 93.218571 | True |
| **2013-01-04** | 93.12 | 93.218571 | True |
| **2013-01-05** | 93.20 | 93.218571 | False |
| **2013-01-06** | 93.20 | 93.218571 | False |
| **2013-01-07** | 93.20 | 93.218571 | True |
| **2013-01-08** | 93.21 | 93.218571 | True |
| **2013-01-09** | 93.08 | 93.218571 | True |
| **2013-01-10** | 93.81 | 93.218571 | True |
| **2013-01-11** | 93.60 | 93.284286 | True |
| **2013-01-12** | 94.27 | 93.470000 | False |
| **2013-01-13** | 94.27 | 93.470000 | False |
| **2013-01-14** | 94.27 | 93.470000 | True |
| **2013-01-15** | 93.26 | 93.490000 | True |

## 3. Read Train and Test Data

In [5]:

```
df_train = pd.read_csv(path + 'train.csv',
                       usecols=['store_nbr', 'family', 'date', 'sales'],
                       dtype={'store_nbr': 'category', 'family': 'category', 'sales': 'float32'},
                       parse_dates=['date'], infer_datetime_format=True)

df_train.date = df_train.date.dt.to_period('D')
df_train = df_train.set_index(['store_nbr', 'family', 'date']).sort_index()

df_train.head(15) # display some entries of the training data
```

Out[5]:

| | | | sales |
|---|---|---|---|
| **store_nbr** | **family** | **date** | |
| | | **2013-01-01** | 0.0 |
| | | **2013-01-02** | 2.0 |
| | | **2013-01-03** | 3.0 |
| | | **2013-01-04** | 3.0 |
| | | **2013-01-05** | 5.0 |
| | | **2013-01-06** | 2.0 |
| | | **2013-01-07** | 0.0 |
| **1** | **AUTOMOTIVE** | **2013-01-08** | 2.0 |
| | | **2013-01-09** | 2.0 |
| | | **2013-01-10** | 2.0 |
| | | **2013-01-11** | 3.0 |
| | | **2013-01-12** | 2.0 |
| | | **2013-01-13** | 2.0 |
| | | **2013-01-14** | 2.0 |
| | | **2013-01-15** | 1.0 |

In [6]:

```
df_test = pd.read_csv(path + 'test.csv',
                      usecols=['store_nbr', 'family', 'date'],
                      dtype={'store_nbr': 'category', 'family': 'category'},
                      parse_dates=['date'], infer_datetime_format=True)

df_test.date = df_test.date.dt.to_period('D')
df_test = df_test.set_index(['store_nbr', 'family', 'date']).sort_index()

df_test.head(15) # display some entries of the testing data
```

Out[6]:

| store_nbr | family | date |
|---|---|---|
| | | **2017-08-16** |
| | | **2017-08-17** |
| | | **2017-08-18** |
| | | **2017-08-19** |
| | | **2017-08-20** |
| | | **2017-08-21** |
| | | **2017-08-22** |
| **1** | **AUTOMOTIVE** | **2017-08-23** |
| | | **2017-08-24** |
| | | **2017-08-25** |
| | | **2017-08-26** |
| | | **2017-08-27** |
| | | **2017-08-28** |
| | | **2017-08-29** |
| | | **2017-08-30** |

In [7]:

```
# set the range of data used in training
sdate = '2017-04-01'
edate = '2017-08-15'

# we will train a model that takes feature of a date as input and predicts the sales for each store and family of goods on that d
y = df_train.unstack(['store_nbr', 'family']).loc[sdate:edate]


########################################################################################################################
# TODO: create the trend feature X: the value for sdate is 1, the value for the next day of sdate is 2, etc.
# Hint: check the documentation of DeterministicProcess, or this tutorial: https://www.kaggle.com/code/ryanholbrook/trend.
########################################################################################################################
dp = DeterministicProcess(
    index=y.index,  # dates from the training data
    constant=True,       # dummy feature for the bias (y_intercept)
    order=1,             # the time dummy (trend)
    drop=True,           # drop terms if necessary to avoid collinearity
)
X = dp.in_sample()

# Extentions
X['oil']  = calendar.loc[sdate:edate]['ma_oil'].values
X['wd']   = calendar.loc[sdate:edate]['wd'].values

X.head(15)
```

Out[7]:

| | const | trend | oil | wd |
|---|---|---|---|---|
| **date** | | | | |
| **2017-04-01** | 1.0 | 1.0 | 49.034286 | False |
| **2017-04-02** | 1.0 | 2.0 | 49.034286 | False |
| **2017-04-03** | 1.0 | 3.0 | 49.034286 | True |
| **2017-04-04** | 1.0 | 4.0 | 49.561429 | True |
| **2017-04-05** | 1.0 | 5.0 | 50.150000 | True |
| **2017-04-06** | 1.0 | 6.0 | 50.625714 | True |
| **2017-04-07** | 1.0 | 7.0 | 51.022857 | True |
| **2017-04-08** | 1.0 | 8.0 | 51.417143 | False |
| **2017-04-09** | 1.0 | 9.0 | 51.417143 | False |
| **2017-04-10** | 1.0 | 10.0 | 51.417143 | True |
| **2017-04-11** | 1.0 | 11.0 | 51.822857 | True |
| **2017-04-12** | 1.0 | 12.0 | 52.232857 | True |
| **2017-04-13** | 1.0 | 13.0 | 52.547143 | True |
| **2017-04-14** | 1.0 | 14.0 | 50.512857 | True |
| **2017-04-15** | 1.0 | 15.0 | 50.512857 | False |

## 4. Train Model!

In [8]:

```
model = LinearRegression()
model.fit(X, y)
y_pred = pd.DataFrame(model.predict(X), index=X.index, columns=y.columns)
```

In [9]:

```
# Results on the training set

y_pred   = y_pred.stack(['store_nbr', 'family']).reset_index()
y_target = y.stack(['store_nbr', 'family']).reset_index().copy()

y_target['sales_pred'] = y_pred['sales'].clip(0.) # Sales should be >= 0

########################################################################################################################
# TODO: show the training loss for each type of product.
# Hint: check the documentation of DataFrame.groupby() and GroupBy.apply().
########################################################################################################################
```

In [10]:

```
y_target.groupby('family')[['sales', 'sales_pred']].apply(lambda Y: mean_squared_log_error(Y.sales, Y.sales_pred))
```

Out[10]:

```
family
AUTOMOTIVE                    0.274819
BABY CARE                     0.070619
BEAUTY                        0.284931
BEVERAGES                     0.203731
BOOKS                         0.028635
BREAD/BAKERY                  0.135292
CELEBRATION                   0.339155
CLEANING                      0.212288
DAIRY                         0.147564
DELI                          0.119779
EGGS                          0.202576
FROZEN FOODS                  0.185090
GROCERY I                     0.214652
GROCERY II                    0.371296
HARDWARE                      0.291139
HOME AND KITCHEN I            0.278487
HOME AND KITCHEN II           0.232872
HOME APPLIANCES               0.162094
HOME CARE                     0.131746
LADIESWEAR                    0.277219
LAWN AND GARDEN               0.295717
LINGERIE                      0.421013
LIQUOR,WINE,BEER              0.791481
MAGAZINES                     0.274324
MEATS                         0.174464
PERSONAL CARE                 0.150410
PET SUPPLIES                  0.222304
PLAYERS AND ELECTRONICS       0.237393
POULTRY                       0.156695
PREPARED FOODS                0.136247
PRODUCE                       0.229127
SCHOOL AND OFFICE SUPPLIES    1.388927
SEAFOOD                       0.286115
dtype: float64
```

In [11]:

```
# Test predictions

stest = '2017-08-16'
etest = '2017-08-31'

########################################################################################################
# TODO: create the feature matrix of test data.
# Hint: check the documentation of DeterministicProcess.
########################################################################################################
# X_test = None # change 'None' to your answer
# dp = DeterministicProcess(
#     index=df_test.unstack(['store_nbr', 'family']).index,  # dates from the training data
#     constant=True,       # dummy feature for the bias (y_intercept)
#     order=1,             # the time dummy (trend)
#     drop=True,           # drop terms if necessary to avoid collinearity
# )
# X_test = dp.in_sample()
X_test = dp.out_of_sample(steps=16)
# Extentions
X_test['oil']  = calendar.loc[stest:etest]['ma_oil'].values
X_test['wd']   = calendar.loc[stest:etest]['wd'].values


print(X_test)

sales_pred = pd.DataFrame(model.predict(X_test), index=X_test.index, columns=y.columns)
sales_pred = sales_pred.stack(['store_nbr', 'family'])

sales_pred[sales_pred < 0] = 0. # Sales should be >= 0
```

```
            const  trend        oil     wd
2017-08-16    1.0  138.0  48.281429   True
2017-08-17    1.0  139.0  47.995714   True
2017-08-18    1.0  140.0  47.852857   True
2017-08-19    1.0  141.0  47.688571  False
2017-08-20    1.0  142.0  47.688571  False
2017-08-21    1.0  143.0  47.688571   True
2017-08-22    1.0  144.0  47.522857   True
2017-08-23    1.0  145.0  47.645714   True
2017-08-24    1.0  146.0  47.598571   True
2017-08-25    1.0  147.0  47.720000   True
2017-08-26    1.0  148.0  47.624286  False
2017-08-27    1.0  149.0  47.624286  False
2017-08-28    1.0  150.0  47.624286   True
2017-08-29    1.0  151.0  47.320000   True
2017-08-30    1.0  152.0  47.115714   True
2017-08-31    1.0  153.0  47.060000   True
```

In [12]:

```
sales_pred
```

Out[12]:

| | | | sales |
|---|---|---|---|
| | **store_nbr** | **family** | |
| | | **AUTOMOTIVE** | 4.922374 |
| | | **BABY CARE** | 0.000000 |
| **2017-08-16** | **1** | **BEAUTY** | 3.543350 |
| | | **BEVERAGES** | 2192.004787 |
| | | **BOOKS** | 0.140162 |
| **...** | **...** | **...** | ... |
| | | **POULTRY** | 377.218940 |
| | | **PREPARED FOODS** | 91.890624 |
| **2017-08-31** | **9** | **PRODUCE** | 1477.958478 |
| | | **SCHOOL AND OFFICE SUPPLIES** | 100.100853 |
| | | **SEAFOOD** | 14.306982 |

28512 rows × 1 columns

In [13]:

```
# Create submission

df_sub = pd.read_csv(path + 'sample_submission.csv', index_col='id')
df_sub.sales = sales_pred.values
df_sub.to_csv('submission.csv', index=True)
```

In [ ]:

```

```