In [0]:

```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
import seaborn as sns
import lightgbm as lgb
from statsmodels.tsa.holtwinters import SimpleExpSmoothing
from sklearn.metrics import mean_absolute_error
from statsmodels.tsa.holtwinters import ExponentialSmoothing
from statsmodels.tsa.statespace.sarimax import SARIMAX
from statsmodels.tsa.arima_model import ARIMA
from statsmodels.tsa.seasonal import seasonal_decompose
import statsmodels.api as sm
import itertools

import warnings
pd.set_option('display.max_columns', None)
pd.set_option('display.width', 500)
warnings.filterwarnings('ignore')
```

In [0]:

```
train = pd.read_csv('data/train.csv', parse_dates=['date'])
test = pd.read_csv('data/test.csv', parse_dates=['date'])
df = pd.concat([train, test], sort=False)
df.head()
```

Out[0]:

	date	store	item	sales	id
0	2013-01-01	1	1	13.0	NaN
1	2013-01-02	1	1	11.0	NaN
2	2013-01-03	1	1	14.0	NaN
3	2013-01-04	1	1	13.0	NaN
4	2013-01-05	1	1	10.0	NaN

In [0]:

```
print("Train setinin boyutu:",train.shape)
print("Test setinin boyutu:",test.shape)
```

```
Train setinin boyutu: (913000, 4)
Test setinin boyutu: (45000, 4)
```

In [0]:

```
df.shape
```

Out[0]:

```
(958000, 5)
```

In [0]:

```
df.quantile([0, 0.05, 0.25, 0.50, 0.75, 0.95, 0.99, 1]).T
```

Out[0]:

	0.00	0.05	0.25	0.50	0.75	0.95	0.99	1.00
store	1.0	1.00	3.00	5.5	8.00	10.00	10.00	10.0
item	1.0	3.00	13.00	25.5	38.00	48.00	50.00	50.0
sales	0.0	16.00	30.00	47.0	70.00	107.00	135.00	231.0

	0.00	0.05	0.25	0.50	0.75	0.95	0.99	1.00
id	0.0	2249.95	11249.75	22499.5	33749.25	42749.05	44549.01	44999.0

In [0]:

```
df["date"].min()
```

Out[0]:

```
Timestamp('2013-01-01 00:00:00')
```

In [0]:

```
df["date"].max()
```

Out[0]:

```
Timestamp('2018-03-31 00:00:00')
```

In [0]:

```
df["sales"].describe([0.10, 0.30, 0.50, 0.70, 0.80, 0.90, 0.95, 0.99])
```

Out[0]:

```
count    913000.000000
mean         52.250287
std          28.801144
min           0.000000
10%          20.000000
30%          33.000000
50%          47.000000
70%          64.000000
80%          76.000000
90%          93.000000
95%         107.000000
99%         135.000000
max         231.000000
Name: sales, dtype: float64
```

In [0]:

```
df["store"].nunique()
```

Out[0]:

```
10
```

In [0]:

```
df["item"].nunique()
```

Out[0]:

```
50
```

In [0]:

```
df.groupby(["store"])["item"].nunique()
```

Out[0]:

```
store
1      50
2      50
3      50
4      50
5      50
6      50
7      50
8      50
9      50
10     50
Name: item, dtype: int64
```

In [0]:

```
df.groupby(["store", "item"]).agg({"sales": ["sum", "mean", "median", "std"]})
```

Out[0]:

		sales			
		sum	mean	median	std
store	item				
1	1	36468.0	19.971522	19.0	6.741022
	2	97050.0	53.148959	52.0	15.005779
	3	60638.0	33.208105	33.0	10.072529
	4	36440.0	19.956188	20.0	6.640618
	5	30335.0	16.612815	16.0	5.672102
...	...	...	...	...	...
10	46	120601.0	66.046550	65.0	18.114991
	47	45204.0	24.755750	24.0	7.924820
	48	105570.0	57.814896	57.0	15.898538
	49	60317.0	33.032311	32.0	10.091610
	50	135192.0	74.037240	73.0	19.937566

500 rows × 4 columns

In [0]:

```
df['month'] = df.date.dt.month
df['day_of_month'] = df.date.dt.day
df['day_of_year'] = df.date.dt.dayofyear
df['week_of_year'] = df.date.dt.weekofyear
df['day_of_week'] = df.date.dt.dayofweek
df['year'] = df.date.dt.year
df["is_wknd"] = df.date.dt.weekday // 4
df['is_month_start'] = df.date.dt.is_month_start.astype(int)
df['is_month_end'] = df.date.dt.is_month_end.astype(int)
```

In [0]:

```
df.head()
```

Out[0]:

	date	store	item	sales	id	month	day_of_month	day_of_year	week_of_year	day_of_week	year	is_wknd	is_month_s
0	2013-01-01	1	1	13.0	NaN	1	1	1	1	1	2013	0	
1	2013-01-02	1	1	11.0	NaN	1	2	2	1	2	2013	0	
2	2013-01-03	1	1	14.0	NaN	1	3	3	1	3	2013	0	
3	2013-01-04	1	1	13.0	NaN	1	4	4	1	4	2013	1	
4	2013-01-05	1	1	10.0	NaN	1	5	5	1	5	2013	1	

In [0]:

```
df.groupby(["store", "item", "month"]).agg({"sales": ["sum", "mean", "median", "std"]})
```

Out[0]:

			sales			
			sum	mean	median	std
store	item	month				
1	1	1	2125.0	13.709677	13.0	4.397413
		2	2063.0	14.631206	14.0	4.668146
		3	2728.0	17.600000	17.0	4.545013
		4	3118.0	20.786667	20.0	4.894301
		5	3448.0	22.245161	22.0	6.564705
...	...	...	...	...	...	...
10	50	8	13108.0	84.567742	85.0	15.676527
		9	11831.0	78.873333	79.0	15.207423
		10	11322.0	73.045161	72.0	14.209171
		11	11549.0	76.993333	77.0	16.253651
		12	8724.0	56.283871	56.0	11.782529

6000 rows × 4 columns

In [0]:

```
def random_noise(dataframe):
    return np.random.normal(scale=1.6, size=(len(dataframe),))
```

In [0]:

```
df.sort_values(by=['store', 'item', 'date'], axis=0, inplace=True)
df.head()
```

Out[0]:

	date	store	item	sales	id	month	day_of_month	day_of_year	week_of_year	day_of_week	year	is_wknd	is_month_s
0	2013-01-01	1	1	13.0	NaN	1	1	1	1	1	2013	0	
1	2013-01-02	1	1	11.0	NaN	1	2	2	1	2	2013	0	
2	2013-01-03	1	1	14.0	NaN	1	3	3	1	3	2013	0	
3	2013-01-04	1	1	13.0	NaN	1	4	4	1	4	2013	1	
4	2013-01-05	1	1	10.0	NaN	1	5	5	1	5	2013	1	

In [0]:

```
def lag_features(dataframe, lags):
    for lag in lags:
        dataframe['sales_lag_' + str(lag)] = dataframe.groupby(["store", "item"])['sales
'].transform(
            lambda x: x.shift(lag)) + random_noise(dataframe)
    return dataframe

df = lag_features(df, [91, 98, 105, 112, 119, 126, 182, 364, 546, 728])
```

In [0]:

```
def roll_mean_features(dataframe, windows):
    for window in windows:
```

```
        dataframe['sales_roll_mean_' + str(window)] = dataframe.groupby(["store", "item"
])['sales']. \
                                                          transform(
            lambda x: x.shift(1).rolling(window=window, min_periods=10, win_type="triang
").mean()) + random_noise(
            dataframe)
    return dataframe


df = roll_mean_features(df, [365, 546, 730])
```

In [0]:

```
def ewm_features(dataframe, alphas, lags):
    for alpha in alphas:
        for lag in lags:
            dataframe['sales_ewm_alpha_' + str(alpha).replace(".", "") + "_lag_" + str(l
ag)] = \
                dataframe.groupby(["store", "item"])['sales'].transform(lambda x: x.shif
t(lag).ewm(alpha=alpha).mean())
    return dataframe


alphas = [0.99, 0.95, 0.9, 0.8, 0.7, 0.5]
lags = [91, 98, 105, 112, 180, 270, 365, 546, 728]

df = ewm_features(df, alphas, lags)
df.tail()
```

Out[0]:

	date	store	item	sales	id	month	day_of_month	day_of_year	week_of_year	day_of_week	year	is_wknd	is_
44995	2018-03-27	10	50	NaN	44995.0	3	27	86	13	1	2018	0	
44996	2018-03-28	10	50	NaN	44996.0	3	28	87	13	2	2018	0	
44997	2018-03-29	10	50	NaN	44997.0	3	29	88	13	3	2018	0	
44998	2018-03-30	10	50	NaN	44998.0	3	30	89	13	4	2018	1	
44999	2018-03-31	10	50	NaN	44999.0	3	31	90	13	5	2018	1	

In [0]:

```
df = pd.get_dummies(df, columns=['day_of_week', 'month'])
```

In [0]:

```
df['sales'] = np.log1p(df["sales"].values)
```

In [0]:

```
train = df.loc[(df["date"] < "2017-01-01"), :]

val = df.loc[(df["date"] >= "2017-01-01") & (df["date"] < "2017-04-01"), :]

cols = [col for col in train.columns if col not in ['date', 'id', "sales", "year"]]
```

In [0]:

```
Y_train = train['sales']

X_train = train[cols]

Y_val = val['sales']
```

```
X_val = val[cols]

Y_train.shape, X_train.shape, Y_val.shape, X_val.shape
```

Out[0]:

```
((730500,), (730500, 94), (45000,), (45000, 94))
```

In [0]:

```
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 lgbm_smape(preds, train_data):
    labels = train_data.get_label()
    smape_val = smape(np.expm1(preds), np.expm1(labels))
    return 'SMAPE', smape_val, False
```

In [0]:

```
# LightGBM parameters
lgb_params = {'metric': {'mae'},
              'num_leaves': 10,
              'learning_rate': 0.02,
              'feature_fraction': 0.8,
              'max_depth': 5,
              'verbose': 0,
              'num_boost_round': 2000,
              'early_stopping_rounds': 200,
              'nthread': -1}
```

In [0]:

```
lgbtrain = lgb.Dataset(data=X_train, label=Y_train, feature_name=cols)
lgbval = lgb.Dataset(data=X_val, label=Y_val, reference=lgbtrain, feature_name=cols)

model = lgb.train(lgb_params, lgbtrain,
                  valid_sets=[lgbtrain, lgbval],
                  num_boost_round=lgb_params['num_boost_round'],
                  early_stopping_rounds=lgb_params['early_stopping_rounds'],
                  feval=lgbm_smape,
                  verbose_eval=100)

y_pred_val = model.predict(X_val, num_iteration=model.best_iteration)

smape(np.expm1(y_pred_val), np.expm1(Y_val))
```

```
[LightGBM] [Warning] Auto-choosing col-wise multi-threading, the overhead of testing was
0.445511 seconds.
You can set `force_col_wise=true` to remove the overhead.
Training until validation scores don't improve for 200 rounds
[100] training's l1: 0.171539 training's SMAPE: 17.4956 valid_1's l1: 0.170797 valid_1's
SMAPE: 17.458
[200] training's l1: 0.141262 training's SMAPE: 14.4691 valid_1's l1: 0.14515 valid_1's S
MAPE: 14.8904
[300] training's l1: 0.135658 training's SMAPE: 13.9075 valid_1's l1: 0.140024 valid_1's
SMAPE: 14.376
[400] training's l1: 0.13356 training's SMAPE: 13.6978 valid_1's l1: 0.138344 valid_1's S
MAPE: 14.2078
[500] training's l1: 0.132368 training's SMAPE: 13.5784 valid_1's l1: 0.137064 valid_1's
SMAPE: 14.0792
[600] training's l1: 0.131549 training's SMAPE: 13.4962 valid_1's l1: 0.135993 valid_1's
SMAPE: 13.9714
[700] training's l1: 0.130932 training's SMAPE: 13.4344 valid_1's l1: 0.135259 valid_1's
```

```
SMAPE: 13.8975
[800] training's l1: 0.130454 training's SMAPE: 13.3865 valid_1's l1: 0.134711 valid_1's
SMAPE: 13.8423
[900] training's l1: 0.130056 training's SMAPE: 13.3467 valid_1's l1: 0.134309 valid_1's
SMAPE: 13.8018
[1000] training's l1: 0.12973 training's SMAPE: 13.3141 valid_1's l1: 0.133989 valid_1's
SMAPE: 13.7696
[1100] training's l1: 0.129432 training's SMAPE: 13.2843 valid_1's l1: 0.133728 valid_1's
SMAPE: 13.7432
[1200] training's l1: 0.129176 training's SMAPE: 13.2587 valid_1's l1: 0.133529 valid_1's
SMAPE: 13.7232
[1300] training's l1: 0.128961 training's SMAPE: 13.2372 valid_1's l1: 0.133355 valid_1's
SMAPE: 13.7056
[1400] training's l1: 0.128761 training's SMAPE: 13.2172 valid_1's l1: 0.133181 valid_1's
SMAPE: 13.6882
[1500] training's l1: 0.128579 training's SMAPE: 13.199 valid_1's l1: 0.133043 valid_1's
SMAPE: 13.6742
[1600] training's l1: 0.128425 training's SMAPE: 13.1835 valid_1's l1: 0.132922 valid_1's
SMAPE: 13.662
[1700] training's l1: 0.128281 training's SMAPE: 13.169 valid_1's l1: 0.132798 valid_1's
SMAPE: 13.6496
[1800] training's l1: 0.12815 training's SMAPE: 13.1559 valid_1's l1: 0.132686 valid_1's
SMAPE: 13.6383
[1900] training's l1: 0.128035 training's SMAPE: 13.1444 valid_1's l1: 0.132595 valid_1's
SMAPE: 13.629
[2000] training's l1: 0.127919 training's SMAPE: 13.1328 valid_1's l1: 0.132506 valid_1's
SMAPE: 13.6201
Did not meet early stopping. Best iteration is:
[2000] training's l1: 0.127919 training's SMAPE: 13.1328 valid_1's l1: 0.132506 valid_1's
SMAPE: 13.6201
```

Out[0]:

```
13.62007233782973
```

In [0]:

```
#Final Model

train = df.loc[~df.sales.isna()]
Y_train = train['sales']
X_train = train[cols]

test = df.loc[df.sales.isna()]
X_test = test[cols]
```

In [0]:

```
lgb_params = {'metric': {'mae'},
              'num_leaves': 10,
              'learning_rate': 0.02,
              'feature_fraction': 0.8,
              'max_depth': 5,
              'verbose': 0,
              'nthread': -1,
              "num_boost_round": model.best_iteration}

# LightGBM dataset
lgbtrain_all = lgb.Dataset(data=X_train, label=Y_train, feature_name=cols)

model = lgb.train(lgb_params, lgbtrain_all, num_boost_round=model.best_iteration)
test_preds = model.predict(X_test, num_iteration=model.best_iteration)
```

```
[LightGBM] [Warning] Auto-choosing col-wise multi-threading, the overhead of testing was
0.540060 seconds.
You can set `force_col_wise=true` to remove the overhead.
```

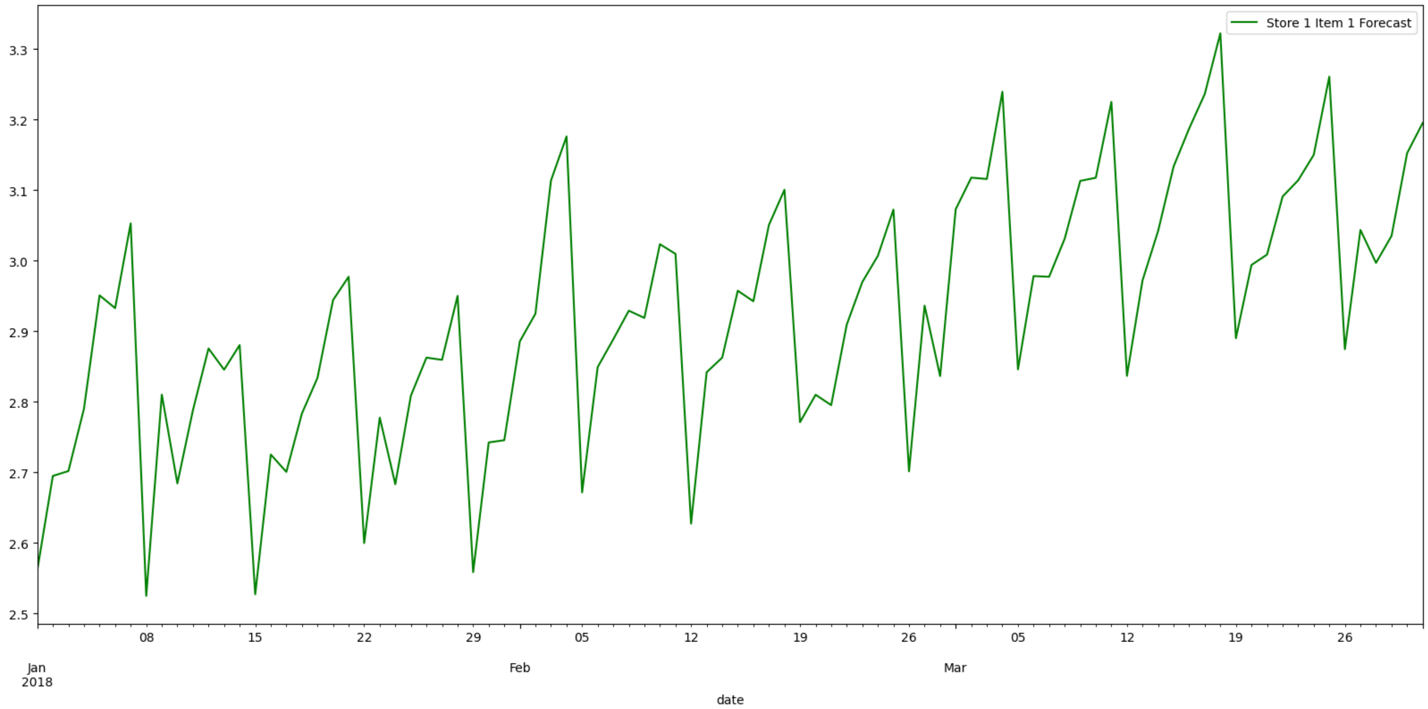
In [0]:

```
forecast = pd.DataFrame({"date":test["date"],
                         "store":test["store"],
                         "item":test["item"],
```

```
                              "sales":test_preds
                              })

forecast[(forecast.store == 1) & (forecast.item == 1)].set_index("date").sales.plot(colo
r = "green",
                                                                                     fig
size = (20,9),
                                                                                     leg
end=True, label = "Store 1 Item 1 Forecast");
```

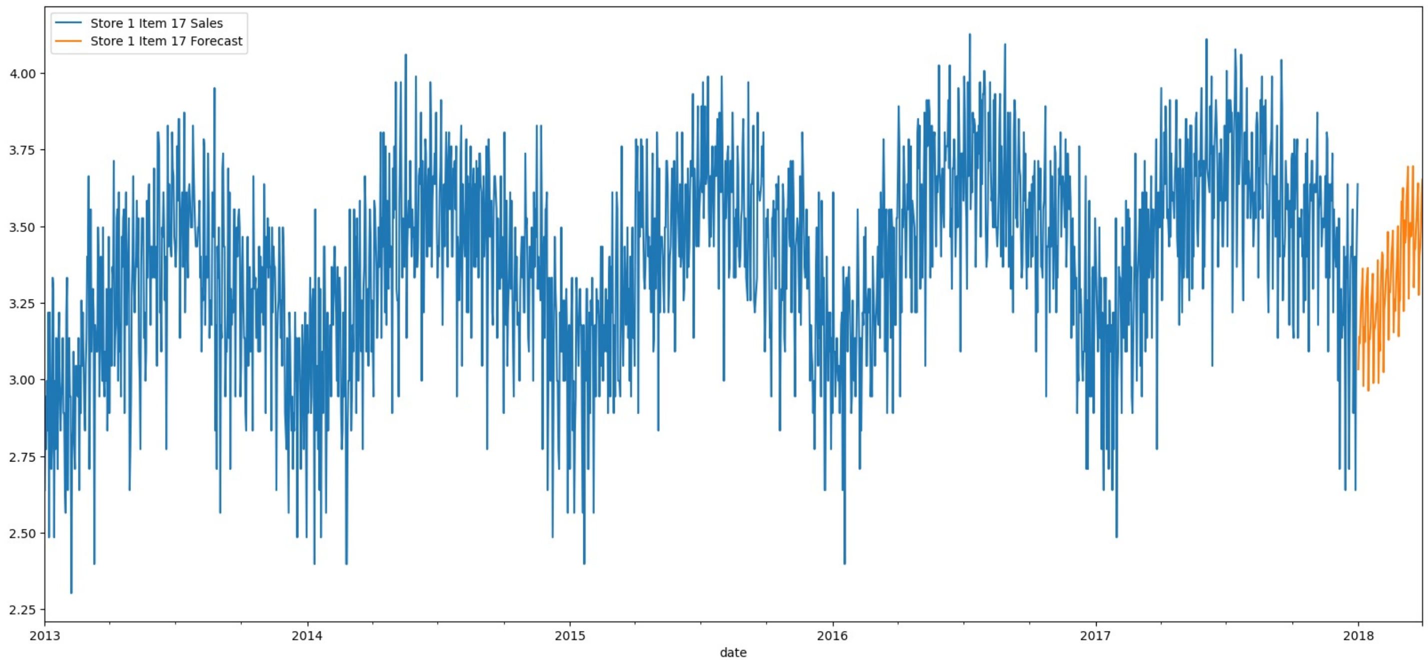
Out[0]:



In [0]:

```
train[(train.store == 1) & (train.item == 17)].set_index("date").sales.plot(figsize = (2
0,9),legend=True, label = "Store 1 Item 17 Sales")
forecast[(forecast.store == 1) & (forecast.item == 17)].set_index("date").sales.plot(leg
end=True, label = "Store 1 Item 17 Forecast");
```

Out[0]:



In [0]:

```
df.shape
```

Out[0]:

```
(958000, 98)
```