DEMAND FORECASTING

image.png

Demand Forecasting is the process in which historical sales data is used to develop an estimate of an expected forecast of customer demand. To businesses, Demand Forecasting provides an estimate of the amount of goods and services that its customers will purchase in the foreseeable future.

1.0 Import Library

```
import pandas as pd
import numpy as np
from datetime import datetime
import matplotlib.pyplot as plt
%matplotlib inline
import plotly.offline as py
py.init_notebook_mode(connected=True)
import plotly.graph_objs as go
import os
import seaborn as sns
import gc
from sklearn.metrics import mean_squared_error
import statsmodels.api as sm
import lightgbm as lgb
plt.style.use('ggplot')
seed = 433
from sklearn.metrics import mean_squared_log_error
from sklearn.metrics import mean_squared_error
```

1.2 Load the datasets

```
from google.colab import drive
drive.mount('/content/drive/')
#os.chdir("C://Users//rohan//Desktop//Supply-chain//Dataset")
train_df = pd.read_csv('/content/drive/My Drive/Dataset/train.csv')
# First let us load the datasets into different Dataframes
#train_df = pd.read_csv('train.csv')

# Dimensions
print('Train shape:', train_df.shape)
# Set of features we have are: date, store, and item
display(train_df.sample(10))
```

```
Drive already mounted at /content/drive/; to attempt to forcibly remount, call drive.mount("/content/drive/", force_rem
Train shape: (913000, 4)
```

	date	store	item	sales
363977	2014-08-27	10	20	41
748992	2013-11-29	1	42	36
571491	2017-11-15	3	32	48
48975	2017-02-08	7	3	19
355658	2016-11-15	5	20	28
252031	2013-02-13	9	14	35
328333	2017-01-19	10	18	63
742885	2017-03-10	7	41	20
900277	2013-03-01	4	50	53
132157	2014-11-17	3	8	69

```
#import os
#os.chdir("C://Users//rohan//Desktop//Supply-chain//Dataset")
#pd.read_csv('/content/drive/My Drive/Dataset/train.csv')
train = pd.read_csv('/content/drive/My Drive/Dataset/train.csv',parse_dates=[0],nrows=None)
test = pd.read_csv('/content/drive/My Drive/Dataset/test.csv',parse_dates=[1], nrows=None )
print('Number of rows and columns in train dataset are:',train.shape)
print('Number of rows and columns in test dataset are:', test.shape)
```

```
Number of rows and columns in train dataset are: (913000, 4)
Number of rows and columns in test dataset are: (45000, 4)
```

1.3 Useful function

```
def basic_details(df):
    """Find number of missing value,dtyeps, unique value in
    dataset"""
    k = pd.DataFrame()
    k['Missing value'] = df.isnull().sum()
    k['% Missing value'] = df.isnull().sum()/df.shape[0]
    k['dtype'] = df.dtypes
    k['N unique'] = df.nunique()
    return k
```

```
def agg_stats(df,statistics,groupby_column):
    """Aggregate a column by unit sales statistics such as
    'mean','sum','min','max', 'var', 'std',"""
    f,ax = plt.subplots(3,2,figsize=(14,8))
    ax =ax.ravel()
    for i,s in enumerate(statistics):
        tmp = (df
         .groupby(groupby_column)
         .agg({'sales':s})
         )
```

```
        tmp.columns = ['sales_{}'.format(s)]
        sns.lineplot(x=tmp.index, y = tmp.iloc[:,0],color='blue',ax=ax[i])
        ax[i].set_xticks(tmp.index)
        for ticks in ax[i].get_xticklabels(): ticks.set_rotation(90)
        #plt.xticks(rotation=90)
        ax[i].set_title('sales_{}'.format(s))
        ax[i].set_ylabel('')
    plt.tight_layout()


### date_time_feat
def date_time_feat(df,column):
    "Extract date time feature"
    df['day'] = df[column].dt.day
    df['dayofweek'] = df[column].dt.dayofweek
    df['month'] = df[column].dt.month
    df['year'] = df[column].dt.year

    df['is_month_end'] = df[column].dt.is_month_end.astype('int8')
    df['is_month_start'] = df[column].dt.is_month_start.astype('int8')
    df['weekofyear'] = df[column].dt.weekofyear
    # conver to category
    #df['dayofweek'] = pd.Categorical(df['dayofweek'],
     #       categories=['Monday','Tuesday', 'Wednesday', 'Thursday', 'Friday','Saturday', 'Sunday',])



# Reduce memory of dataset
def reduce_memory_usage(df):
    """ The function will reduce memory of dataframe """
    intial_memory = df.memory_usage().sum()/1024**2
    print('Intial memory usage:',intial_memory,'MB')
    for col in df.columns:
        mn = df[col].min()
        mx = df[col].max()
        if df[col].dtype != object:
            if df[col].dtype == int:
                if mn >=0:
                    if mx < np.iinfo(np.uint8).max:
                        df[col] = df[col].astype(np.uint8)
```

```
                        df[col] = df[col].astype(np.uint8)
                    elif mx < np.iinfo(np.uint16).max:
                        df[col] = df[col].astype(np.uint16)
                    elif mx < np.iinfo(np.uint32).max:
                        df[col] = df[col].astype(np.uint32)
                    elif mx < np.iinfo(np.uint64).max:
                        df[col] = df[col].astype(np.uint64)
                else:
                    if mn > np.iinfo(np.int8).min and mx < np.iinfo(np.int8).max:
                        df[col] = df[col].astype(np.int8)
                    elif mn > np.iinfo(np.int16).min and mx < np.iinfo(np.int16).max:
                        df[col] = df[col].astype(np.int16)
                    elif mn > np.iinfo(np.int32).min and mx < np.iinfo(np.int32).max:
                        df[col] = df[col].astype(np.int32)
                    elif mn > np.iinfo(np.int64).min and mx < np.iinfo(np.int64).max:
                        df[col] = df[col].astype(np.int64)
            if df[col].dtype == float:
                df[col] =df[col].astype(np.float32)

    red_memory = df.memory_usage().sum()/1024**2
    print('Memory usage after complition: ',red_memory,'MB')
```

2.0 Exploratory data analysis

Glimpse dataset

```
train.head()
```

date	store	item	sales

```
test.head()
```

	id	date	store	item
0	0	2018-01-01	1	1
1	1	2018-01-02	1	1
2	2	2018-01-03	1	1
3	3	2018-01-04	1	1
4	4	2018-01-05	1	1

The test dataset contains id column but train dataset does not contains id column. While importing dataset parse_date is assigned with perticular column index.

```
basic_details(test) # test dataset
```

	Missing value	% Missing value	dtype	N unique
id	0	0.0	int64	45000
date	0	0.0	datetime64[ns]	90
store	0	0.0	int64	10
item	0	0.0	int64	50

```
train.describe() # descriptive statistics about features
```

	store	item	sales
count	913000.000000	913000.000000	913000.000000
mean	5.500000	25.500000	52.250287
std	2.872283	14.430878	28.801144
min	1.000000	1.000000	0.000000
25%	3.000000	13.000000	30.000000

There are 50 diffirent item in 10 diffirent stores. The maximum number of items sold is 231 and average item sold is 52.25.

2.1 Date

Let's extract day, week, month, year from date feature

```
print('Time series start time: "{}" and end time: "{}"'.format(train['date'].min(), train['date'].max()))
print('Time series start time: "{}" and end time: "{}"'.format(test['date'].min(), test['date'].max()))
```

```
Time series start time: "2013-01-01 00:00:00" and end time: "2017-12-31 00:00:00"
Time series start time: "2018-01-01 00:00:00" and end time: "2018-03-31 00:00:00"
```

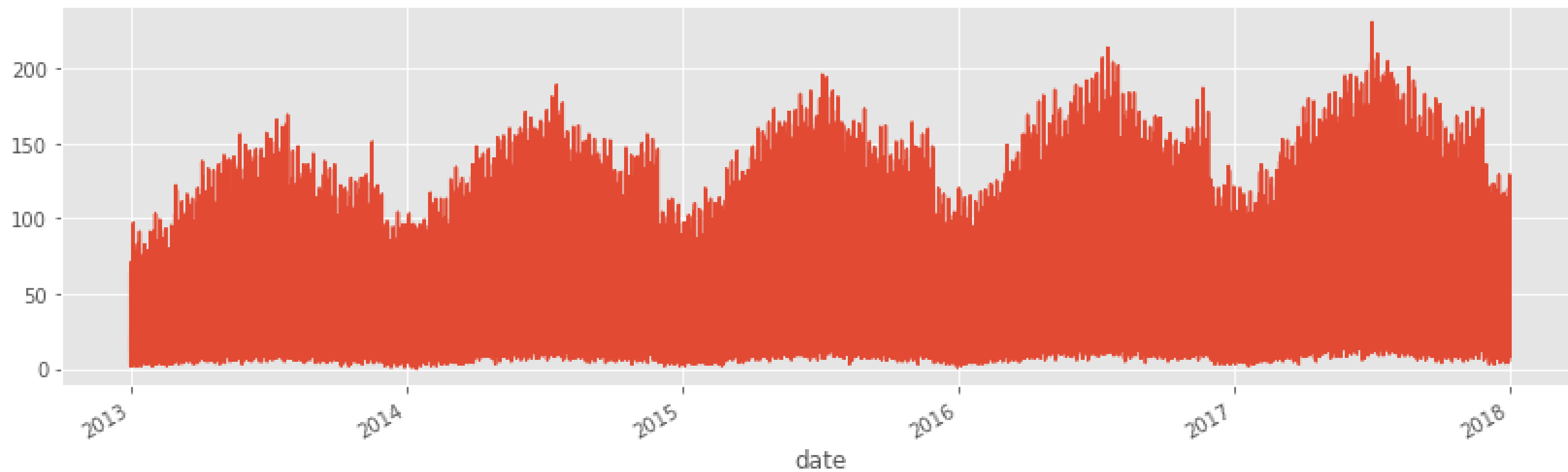
```
# Generate date time feature
date_time_feat(train,'date')
date_time_feat(test,'date')
train.head()
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:11: FutureWarning:

Series.dt.weekofyear and Series.dt.week have been deprecated.  Please use Series.dt.isocalendar().week instead.
```

	date	store	item	sales	day	dayofweek	month	year	is_month_end	is_month_start	weekofyear

```
plt.figure(figsize=(14,4))
train.set_index('date')['sales'].plot(kind='line')
```

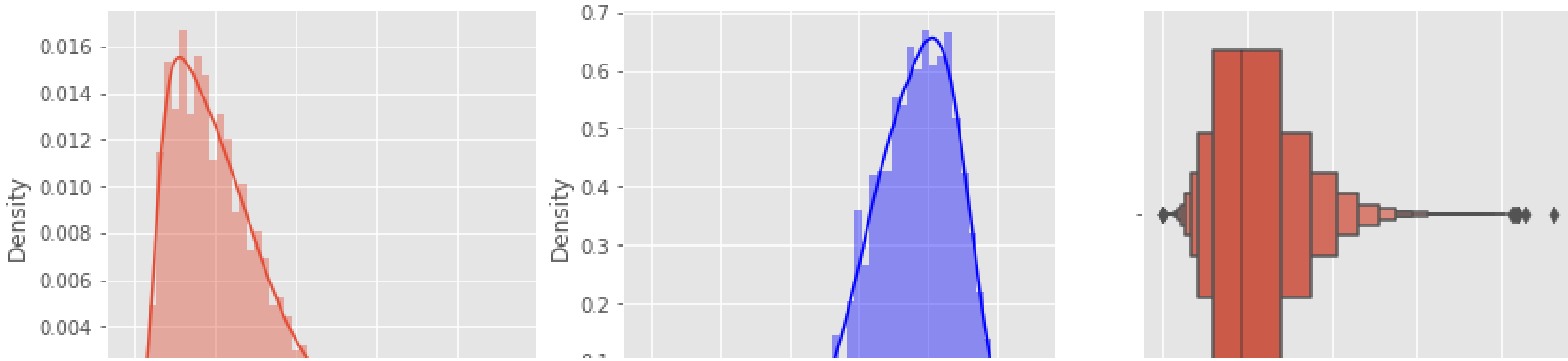
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f6fced79910>
```



2.1 Sales

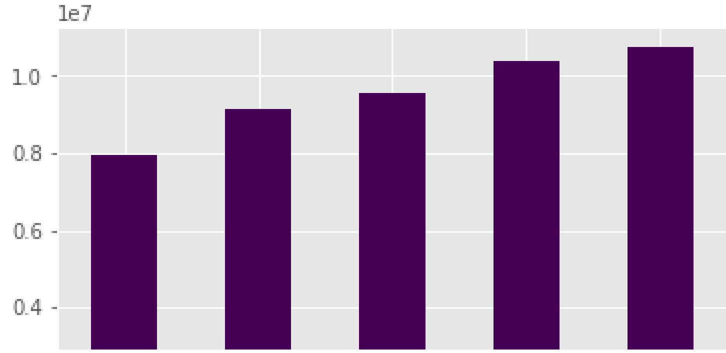
```
f,ax = plt.subplots(1,3,figsize=(14,4))
sns.distplot(train['sales'],ax =ax[0])
sns.distplot(np.log(train['sales']+1),ax=ax[1], color='b')
sns.boxenplot(train['sales'],ax =ax[2])
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2557: FutureWarning:

`distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `disp

/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2557: FutureWarning:

`distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `disp

/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning:

Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`,

<matplotlib.axes._subplots.AxesSubplot at 0x7f6faf1935d0>
```



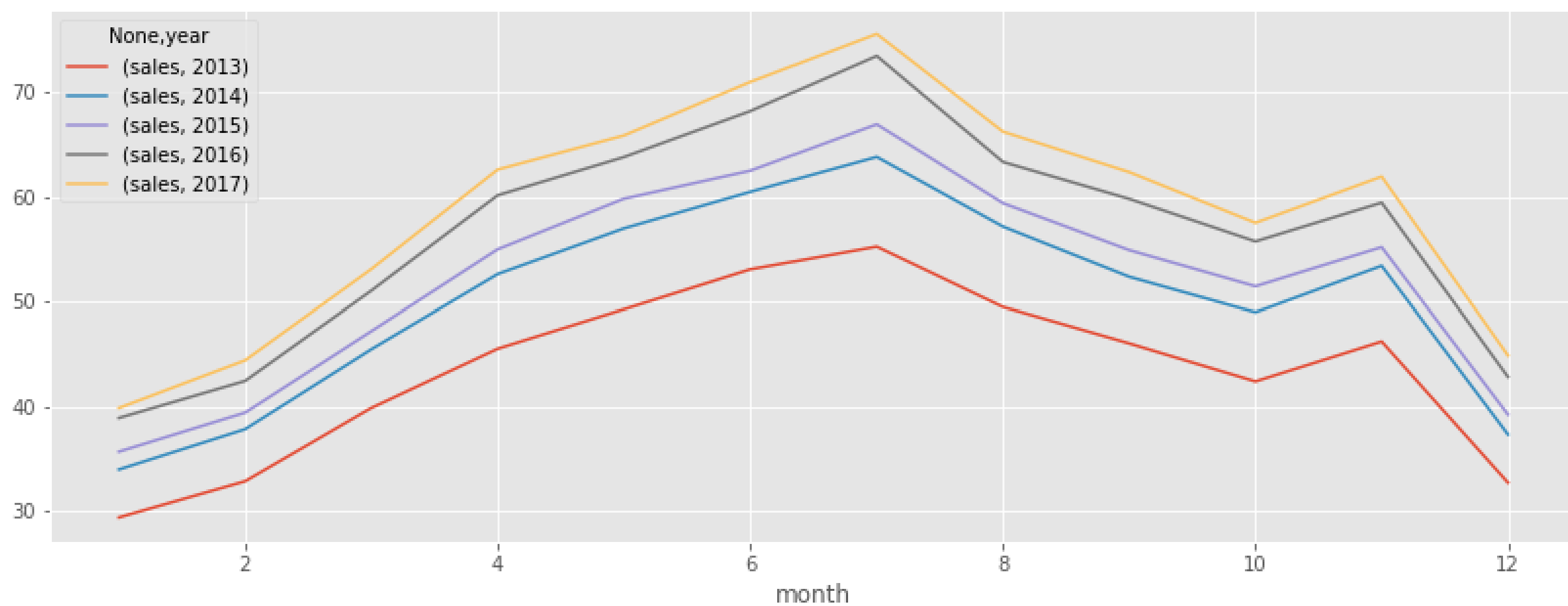

```
(train
 .groupby(['year',])
 .agg({'sales':['sum',]})
 .unstack()
 .plot(kind='bar',cmap='viridis'))
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f6faf0b8dd0>
```




```
(train
 .groupby(['month','year'])
 .agg({'sales':'mean'})
 .unstack()
 .plot(figsize=(14,5)))
```

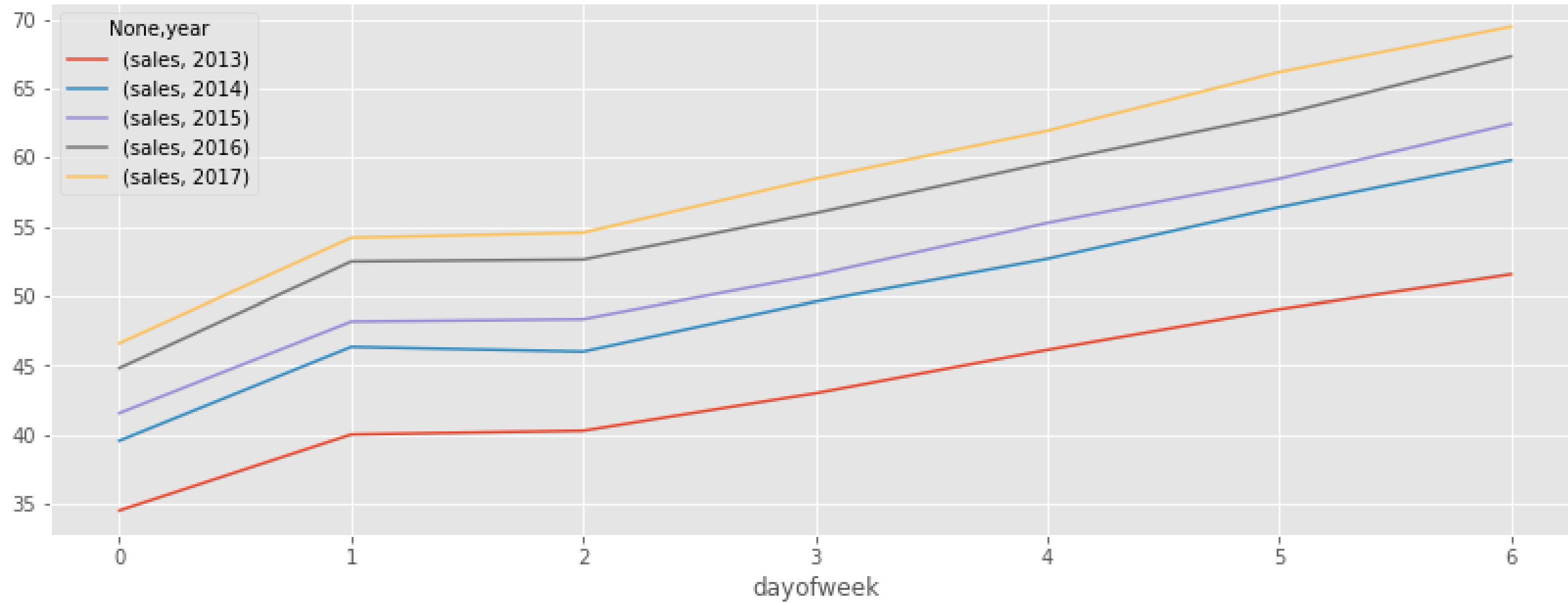
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f6faf05e550>
```




```
(train
 .groupby(['dayofweek','year'])
```

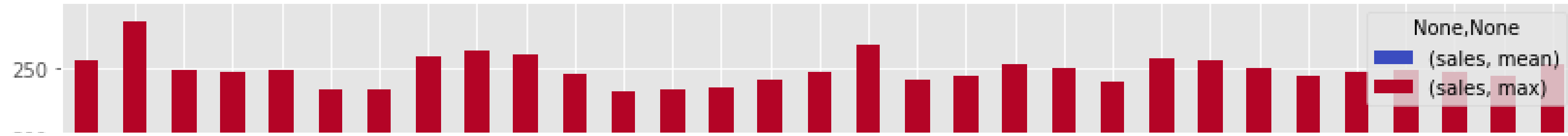
```
.agg({'sales':'mean'})
.unstack()
.plot(figsize=(14,5)))
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f6faeff2fd0>
```



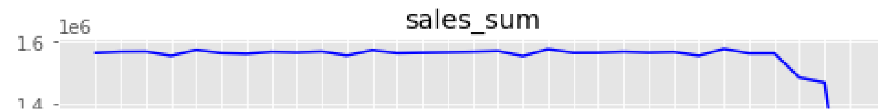

```
(train
.groupby(['day'])
.agg({'sales':['mean','max']})
.plot(figsize=(14,4),kind='bar',stacked=True,cmap='coolwarm'))
```

<matplotlib.axes._subplots.AxesSubplot at 0x7f6faef23bd0>



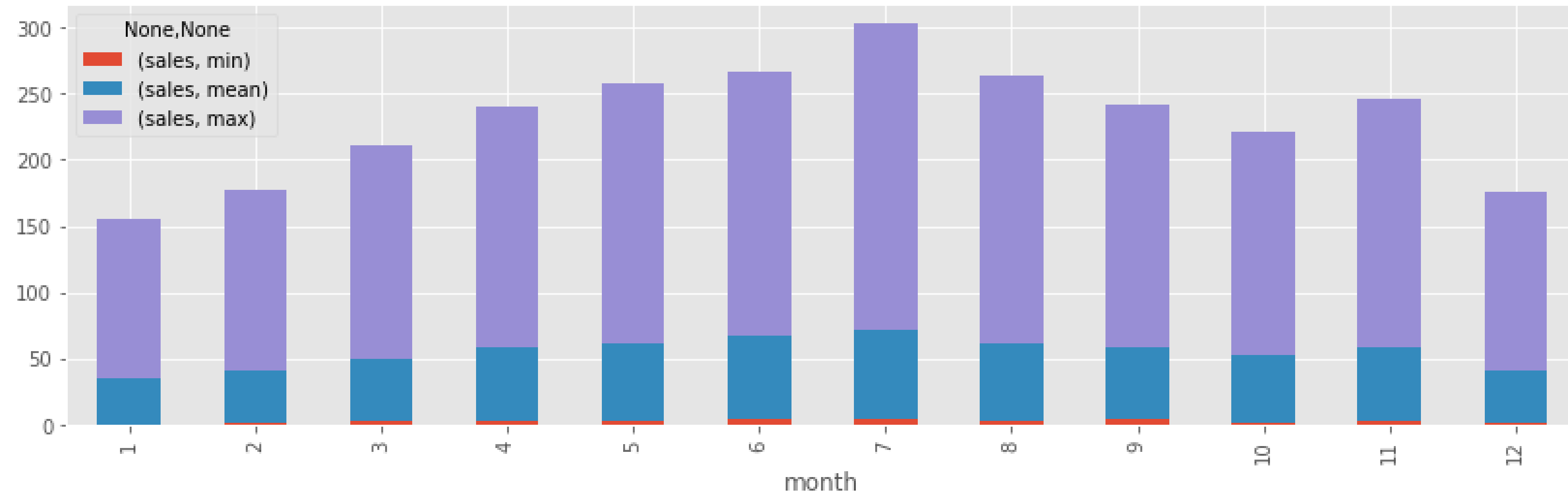
2.2 Aggregate sales statistics by day

```
agg_stats(train,statistics=['mean','sum','min','max', 'var', 'count'],groupby_column=['day'])
```

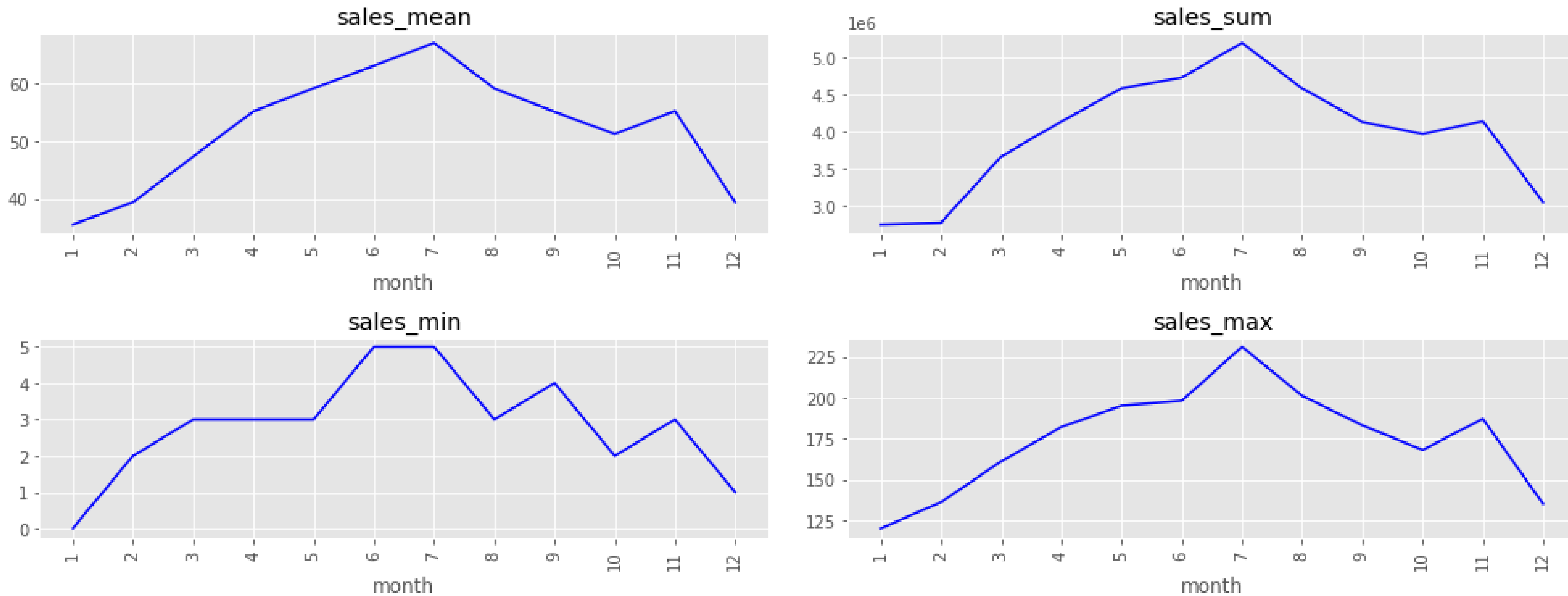



```
(train.groupby('month')
.agg({'sales':['min','mean','max']})
 .plot(figsize=(14,4),kind='bar',stacked=True))
```

<matplotlib.axes._subplots.AxesSubplot at 0x7f6fae9b1910>

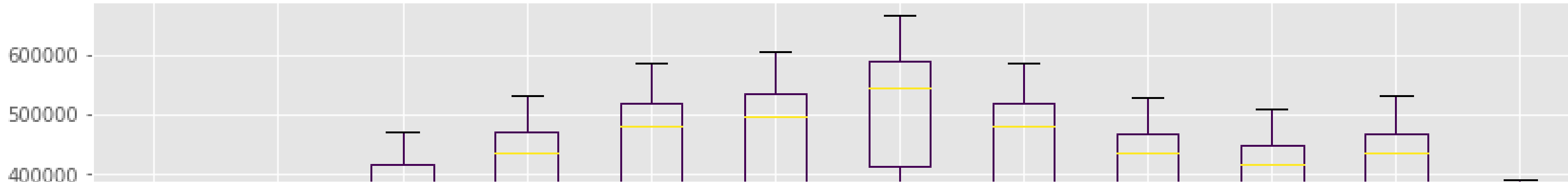


```
agg_stats(train,statistics=['mean','sum','min','max', 'var', 'count'],groupby_column=['month'])
```

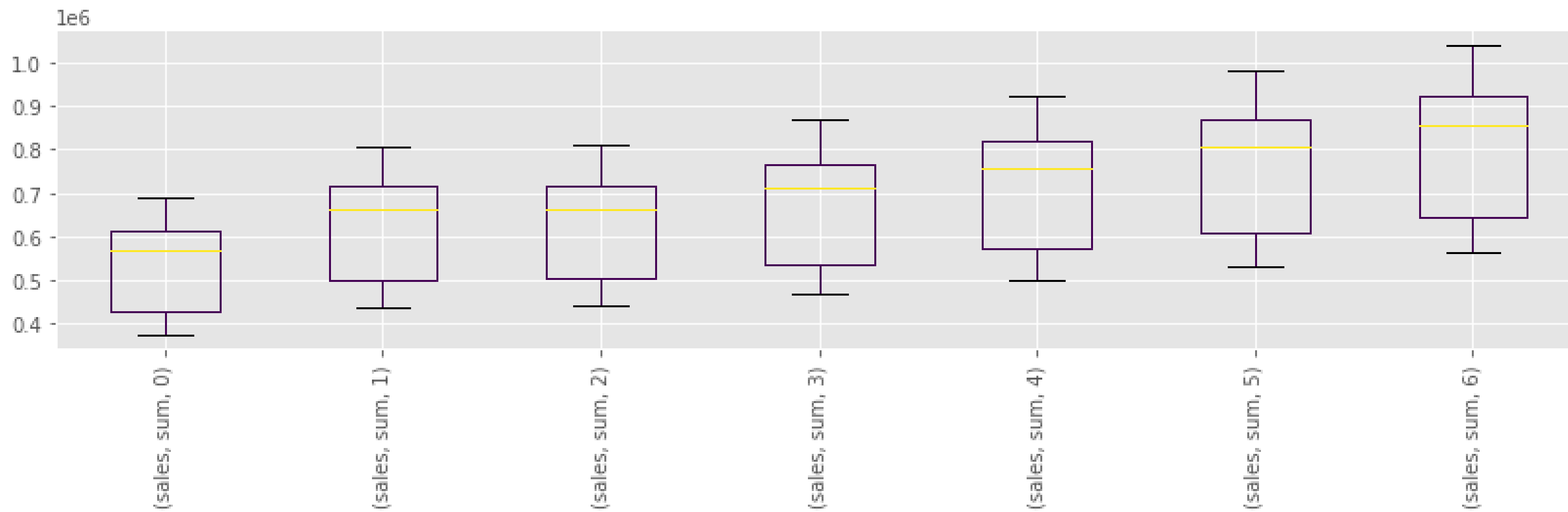


2.3 Store

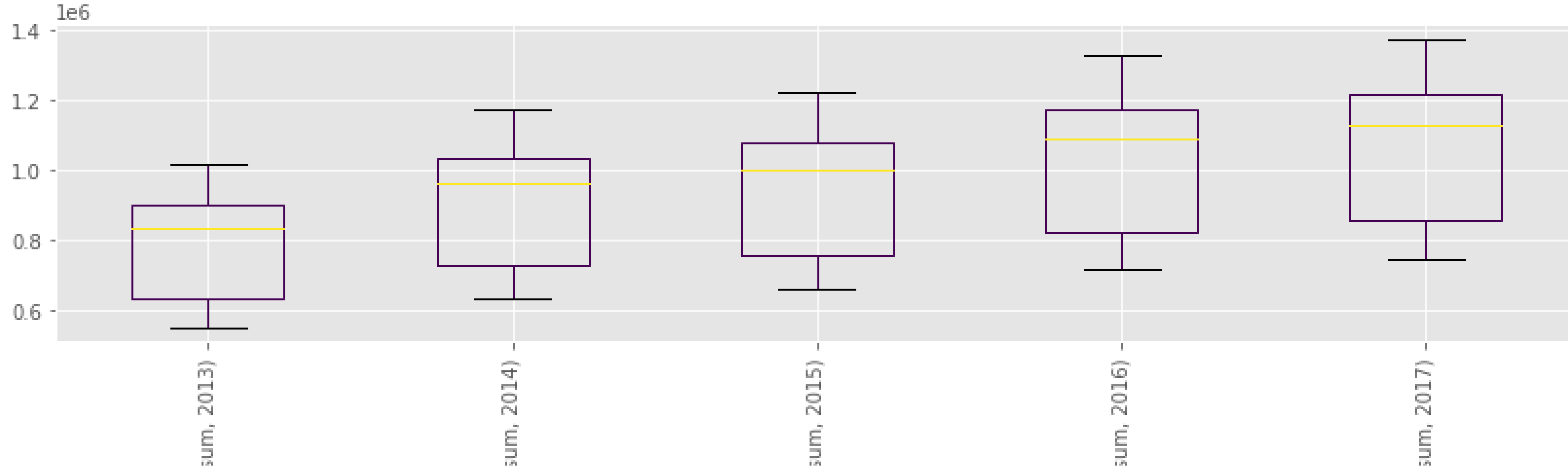
```
(train
 .groupby(['store','month'])
 .agg({'sales':['sum']})
 .unstack()
 .plot(figsize=(14,3),kind='box',stacked=True,cmap='viridis'))
plt.xticks(rotation=90);
```




```
(train
 .groupby(['store','dayofweek'])
 .agg({'sales':['sum']})
 .unstack()
 .plot(figsize=(14,3),kind='box',stacked=True,cmap='viridis'))
plt.xticks(rotation=90);
```

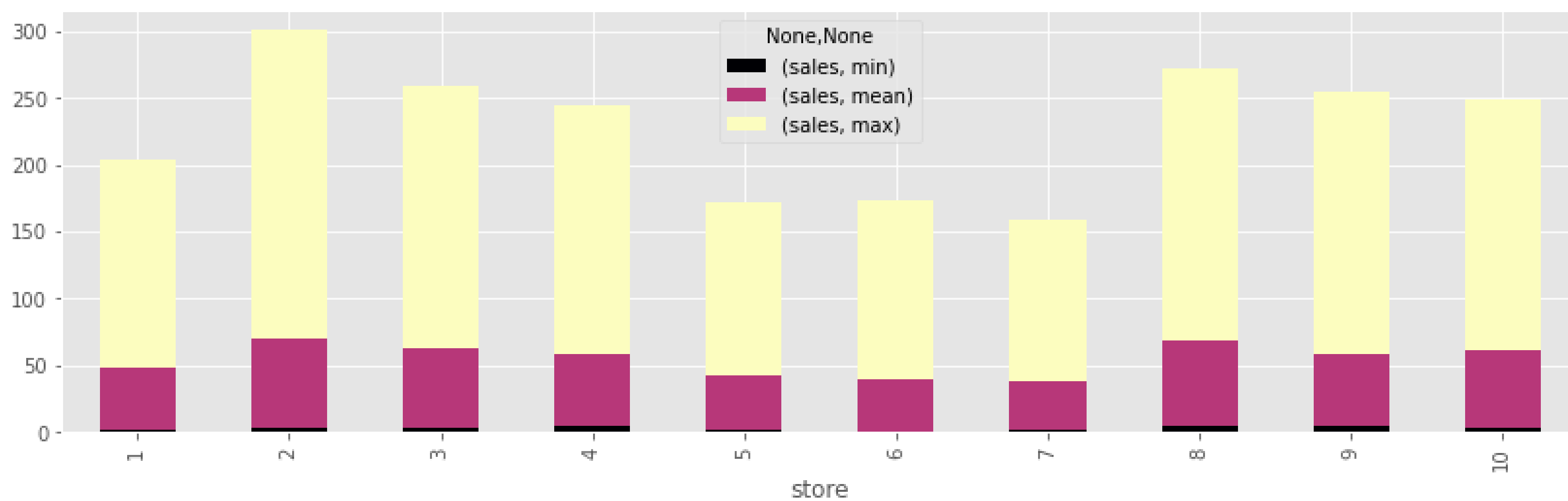



```
(train
 .groupby(['store','year'])
 .agg({'sales':['sum']})
 .unstack()
 .plot(figsize=(14,3),kind='box',stacked=True,cmap='viridis'))
plt.xticks(rotation=90);
```

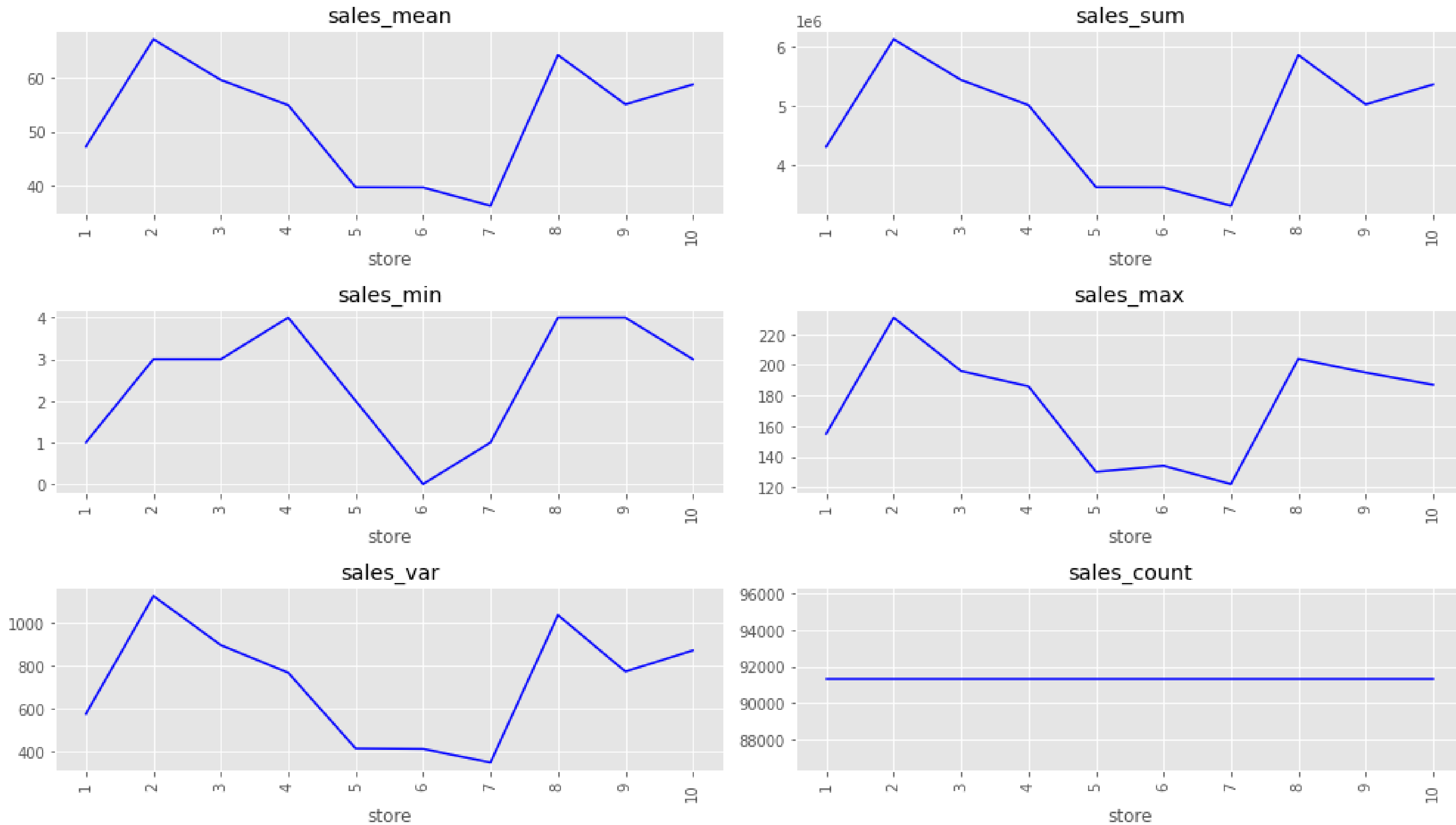



```
(train
 .groupby('store')
 .agg({'sales':['min','mean','max']})
 .plot(figsize=(14,4),kind='bar',stacked=True,cmap='magma'))
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f6fae29e850>
```



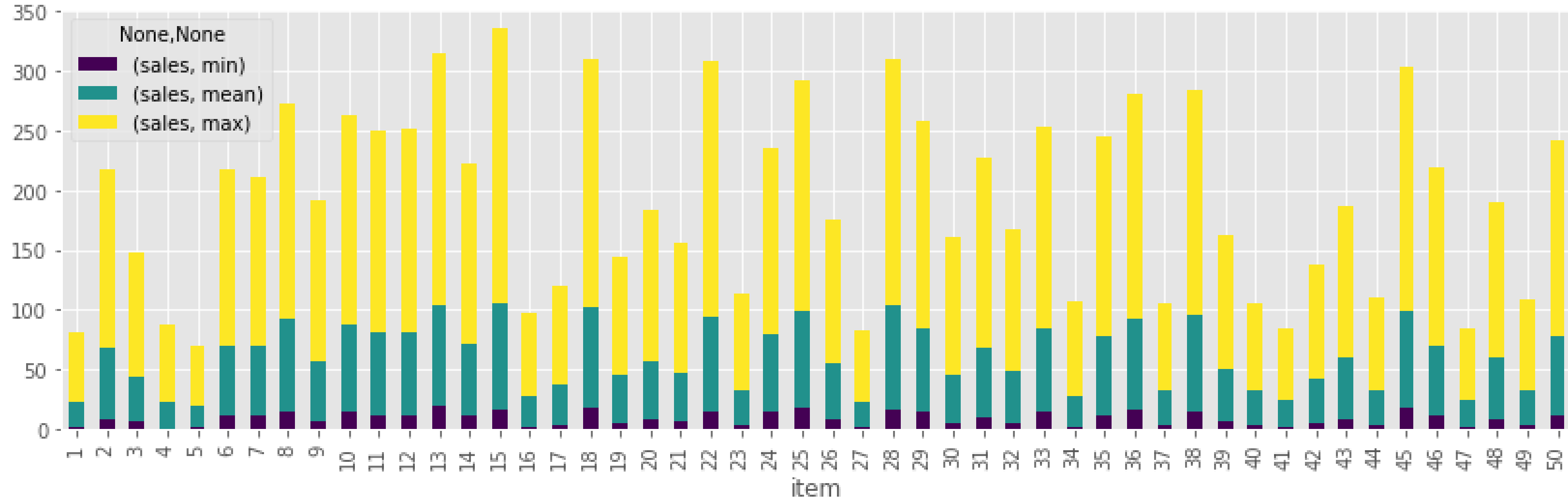

```
agg_stats(train,statistics=['mean','sum','min','max', 'var', 'count'],groupby_column=['store'])
```



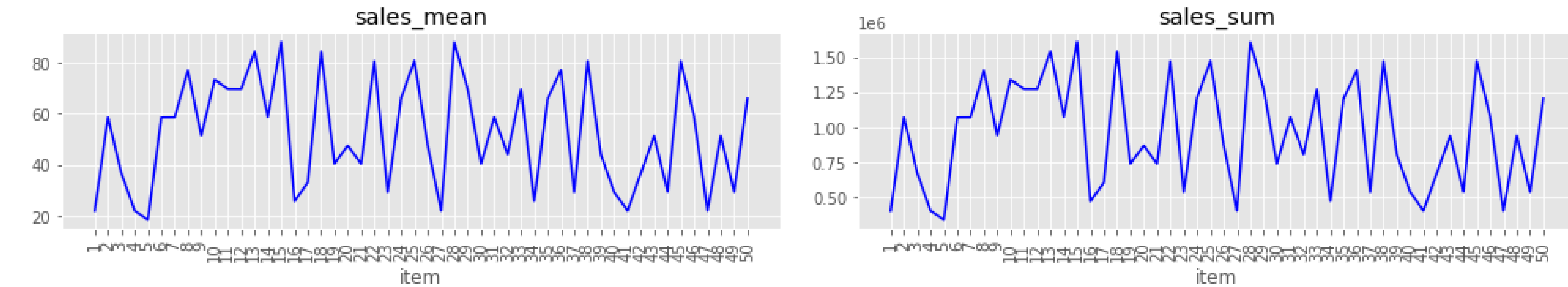
2.4 item

```
(train
 .groupby('item')
 .agg({'sales':['min','mean','max']})
 .plot(figsize=(14,4),kind='bar',stacked=True,cmap='viridis'))
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f6fae00e510>
```

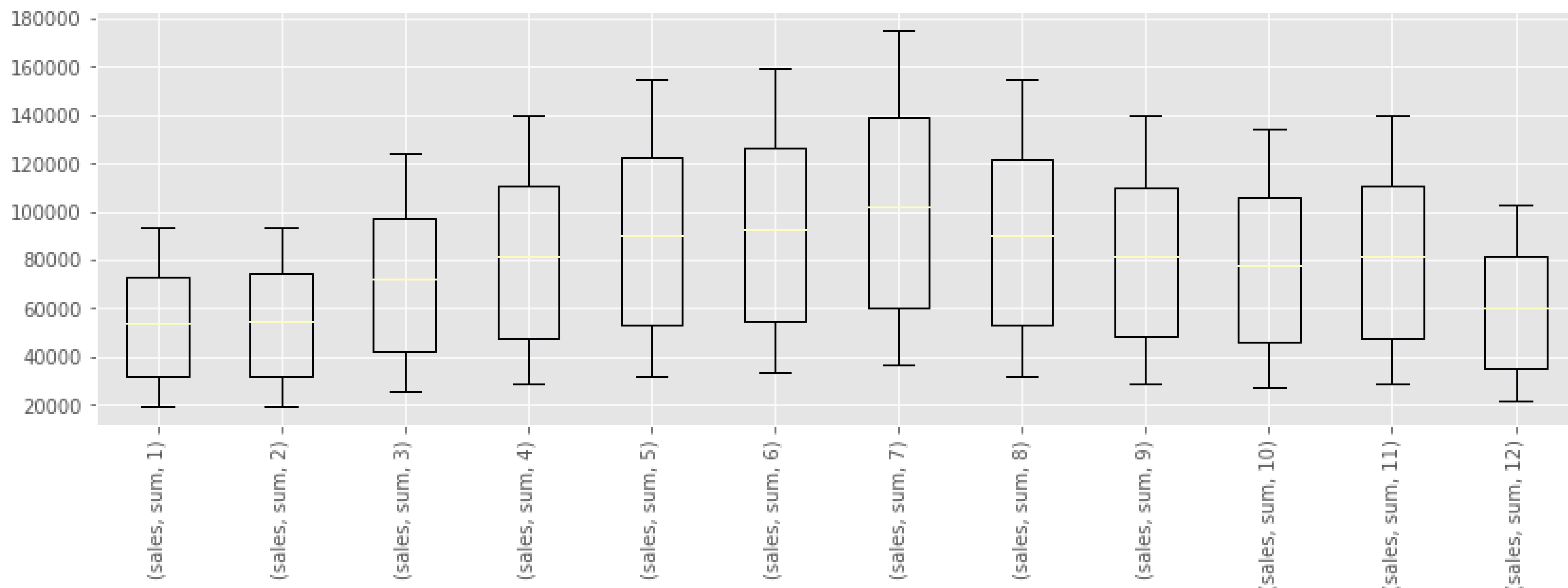



```
agg_stats(train,statistics=['mean','sum','min','max', 'var', 'count'],groupby_column=['item'])
```



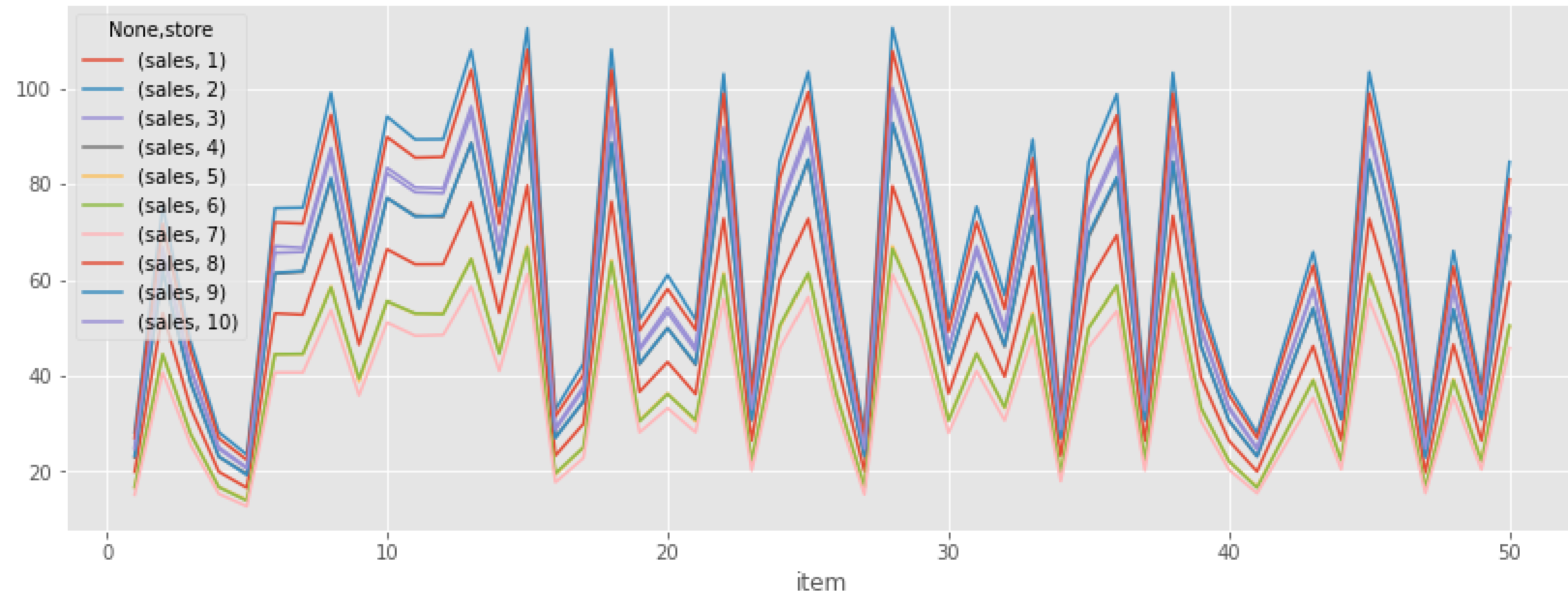
sales min sales max

```
(train
 .groupby(['item','month'])
 .agg({'sales':['sum']})
 .unstack()
 .plot(figsize=(14,4),kind='box',stacked=True,cmap='magma'))
plt.xticks(rotation=90);
```

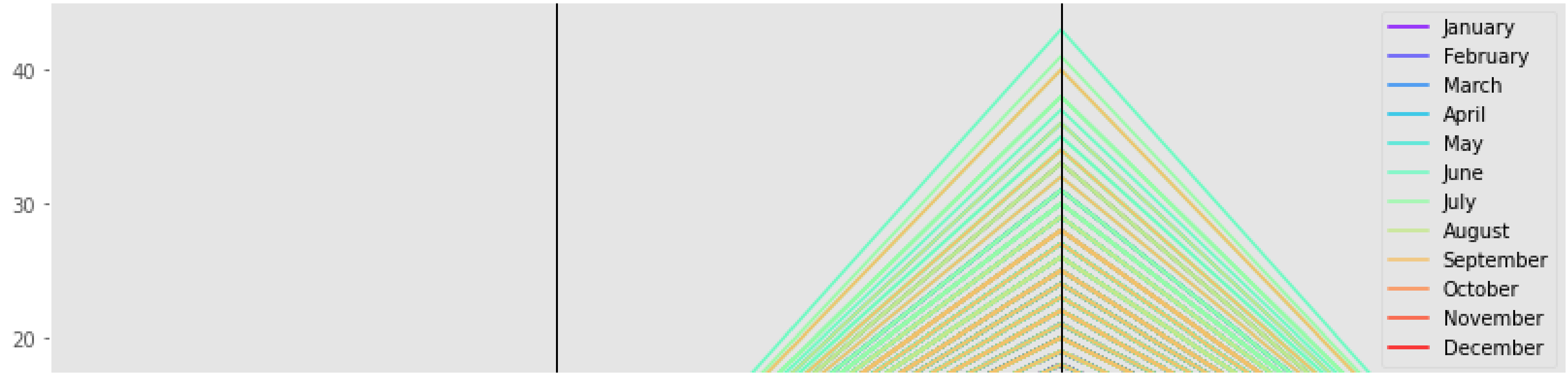



```
(train
 .groupby(['item','store'])
 .agg({'sales':'mean'})
```

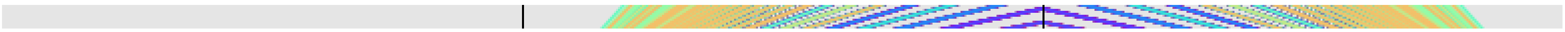
```
 .agg({'sales':'mean'})
 .unstack()
 .plot(figsize=(14,5),kind='line'))
plt.savefig('agg.png')
```




```
train1 =train.copy()
train1['month'] = train1['date'].dt.month_name()
plt.figure(figsize=(14,6))
pd.plotting.parallel_coordinates(train1[['dayofweek','store','sales','item','month']][:1000]
                                 ,'month',colormap='rainbow')
del train1
```

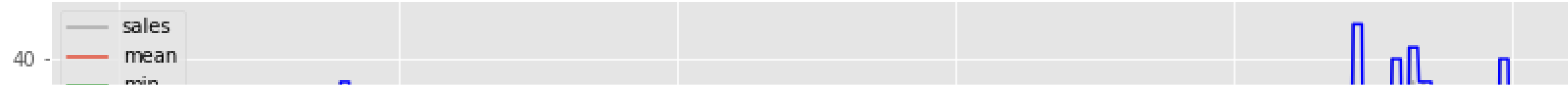


2.5 Rolling window



```
plt.figure(figsize=(14,5))
train['sales'].head(1000).plot(color='darkgray')
train['sales'].head(1000).rolling(window=12).mean().plot(label='mean')
#train['sales'].head(1000).rolling(window=12).median().plot(label='median')
train['sales'].head(1000).rolling(window=7).min().plot(label='min',color='g')
train['sales'].head(1000).rolling(window=7).max().plot(label='max',color='b')
train['sales'].head(1000).rolling(window=7).std().plot(label='std',color='yellow')
plt.legend()
#plt.savefig('Rolling window.png')
```

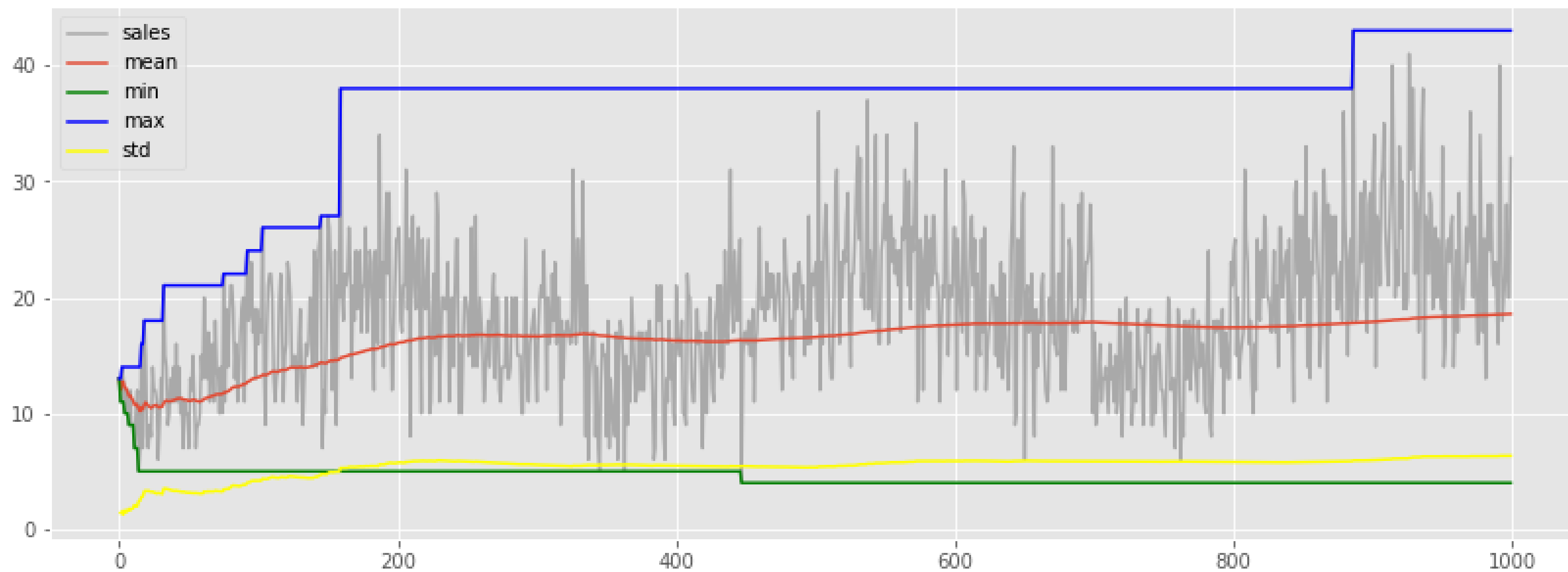
```
<matplotlib.legend.Legend at 0x7f6fae5a4110>
```



2.6 Expanding window

```
# Expanding window
plt.figure(figsize=(14,5))
train['sales'].head(1000).plot(color='darkgray')
train['sales'].head(1000).expanding().mean().plot(label='mean')
#train['sales'].head(1000).rolling(window=12).median().plot(label='median')
train['sales'].head(1000).expanding().min().plot(label='min',color='g')
train['sales'].head(1000).expanding().max().plot(label='max',color='b')
train['sales'].head(1000).expanding().std().plot(label='std',color='yellow')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7f6faa8a2ad0>
```



3.0 Data preprocessing

3.0 Aggregate / Rolling function

```
# Claculate groupby statics for lag date
def calc_stats(df, end,window,groupby=None,aggregates='mean',value='sales'):

    # dates
    last_date = pd.to_datetime(end) - pd.Timedelta(days=1)
    first_date = pd.to_datetime(end) - pd.Timedelta(days= window)
    # Aggregate
    df1 = df[(df.date >=first_date) & (df.date<= last_date) ]
    df_agg = df1.groupby(groupby)[value].agg(aggregates)
    # Change name of columns
    df_agg.name =  str(end).split(' ')[0]+'_' + '_'.join(groupby)+'_'+aggregates+'_'+ str(window)
    return df_agg.reset_index()

#sales_by_store_item
def sales_by_store_item(df, end, aggregates='mean', value='sales'):

    print('Adding sales by store item')
    data = calc_stats(df,end, window=1,aggregates=aggregates,
                      groupby=['store','item'], value=value)
    print('window 1 added')

    for window in  [3,7,14,28,90,180,365]:
        agg = calc_stats(df,end, window=window, aggregates=aggregates,
                         groupby=['store','item'], value=value )
        data = pd.merge(data,agg)
        print('window %d added'% window)
    return data

# sales by store item dayofweek
def sales_by_store_item_dayofweek(df, end, aggregates='mean', value='sales'):

    print('Adding sales by store item dayofweek')
    data = calc_stats(df,end, window=7, aggregates=aggregates,
                      groupby = ['store','item','dayofweek'], value=value)
```

```
    print('window 7 added')

    for window in  [14,28,28*2,28*3,28*6,28*12]:
        agg = calc_stats(df,end, window=window, aggregates=aggregates,
                          groupby=['store','item','dayofweek'], value=value )
        data = pd.merge(data,agg)
        print('window %d added'% window)
    return data

# sales_by_store_item_day
def sales_by_store_item_day(df, end, aggregates='mean', value='sales'):

    print('Adding sales by store item day')
    data = calc_stats(df,end, window=365, aggregates=aggregates,
                      groupby = ['store','item','day'], value=value)
    print('window 365 added')

    return data

# Sales by item
def sales_by_item(df, end, aggregates='mean', value='sales'):

    print('Adding sales by item ')
    data = calc_stats(df,end, window=7, aggregates=aggregates,
                      groupby = ['item'], value=value)
    print('window 7 added')

    for window in  [14,28,28*2]:
        agg = calc_stats(df,end, window=window, aggregates=aggregates,
                          groupby=['item'], value=value )
        data = pd.merge(data,agg)
        print('window %d added'% window)
    return data


def calc_roll_stat(df,end,groupby=None,window=1,aggregate='mean'):
    # Rolling statistics method
    last_date = pd.to_datetime(end) - pd.Timedelta(days=1)
    first_date = pd.to_datetime(end) - pd.Timedelta(days=window)
```

```
        df1 = df[(df.date >= first_date) & (df.date <= last_date)]

        dfPivot = df1.set_index(['date']+groupby)['sales'].unstack().unstack()
        dfPivot = dfPivot.rolling(window=window).mean().fillna(method='bfill')
        return dfPivot.stack().stack().rename(aggregate+str(window))

    def calc_expand_stat(df,end,window=1,aggregate='mean'):
        # Expanding statistics method
        last_date = pd.to_datetime(end) - pd.Timedelta(days=1)
        first_date = pd.to_datetime(end) - pd.Timedelta(days=window)
        df1 = df[(df.date >= first_date) & (df.date <= last_date)]

        dfPivot = df1.set_index(['date','store','item'])['sales'].unstack().unstack()
        dfPivot = dfPivot.expanding(min_periods=window).mean().fillna(method='bfill')
        dfPivot = dfPivot.stack().stack().rename(aggregate+'_'+str(window)).reset_index()
        return dfPivot

    def sales_by_store_item_expading(df,end,aggregate = 'mean', value = 'sales'):
        print('Adding sales by expanding')
        data =calc_expand_stat(df,end,window=3, aggregate='mean')
        return data
    # https://stackoverflow.com/questions/25917287/pandas-groupby-expanding-mean-by-column-value


    def create_data1(sales,test,date):

        # Date input
        for i in range(2):
            end = pd.to_datetime(date) - pd.Timedelta(days=7*i+1)
            print(end)

            # Rolling feature
            #for aggregates in ['mean','min','max','sum','std']:
            for aggregates in ['mean','sum']:

                # store/item
                print('-'*20+'Aggregate by '+aggregates+'-'*20)
                data = sales_by_store_item(sales,end, aggregates=aggregates,value='sales')
                sales = pd.merge(sales,data,on=['store','item'],how='left')
                test = pd.merge(test,data,on=['store','item'], how='left')
```

```
        test = pd.merge(test,data,on=['store','item'], how='left')

        # store/item/dayofweek
        df = sales_by_store_item_dayofweek(sales,end, aggregates=aggregates,value='sales')
        #data = pd.merge(data,df,)
        sales = pd.merge(sales,df,on=['store','item','dayofweek'],how='left')
        test = pd.merge(test,df,on=['store','item','dayofweek'], how='left')

        # store/item/day
        df = sales_by_store_item_day(sales,end, aggregates=aggregates,value='sales')
        #data = pd.merge(data,df)
        sales = pd.merge(sales,df,on=['store','item','day'],how='left')
        test = pd.merge(test,df,on=['store','item','day'], how='left')

        # sales/item
        df = sales_by_item(sales,end, aggregates=aggregates, value='sales')
        data = pd.merge(data,df)
        #data = pd.merge(sales,data)
        sales = pd.merge(sales,df, on=['item'],how='left')
        test = pd.merge(test,df, on=['item'], how='left')

    return sales,test


#Time series start time: "2013-01-01 00:00:00" and end time: "2017-12-31 00:00:00"
#Time series start time: "2018-01-01 00:00:00" and end time: "2018-03-31 00:00:00"
tes_start = '2018-01-01'


# Rolling aggregation or lag feature for diffirend window size
train1,test1 = create_data1(train,test,tes_start)
```

```
window 56 added
window 84 added
window 168 added
window 336 added
Adding sales by store item day
window 365 added
Adding sales by item
window 7 added
window 14 added
window 28 added
```

```
window 28 added
window 56 added
2017-12-24 00:00:00

--------------------Aggregate by mean--------------------
Adding sales by store item
window 1 added
window 3 added
window 7 added
window 14 added
window 28 added
window 90 added
window 180 added
window 365 added
Adding sales by store item dayofweek
window 7 added
window 14 added
window 28 added
window 56 added
window 84 added
window 168 added
window 336 added
Adding sales by store item day
window 365 added
Adding sales by item
window 7 added
window 14 added
window 28 added
window 56 added
--------------------Aggregate by sum--------------------
Adding sales by store item
window 1 added
window 3 added
window 7 added
window 14 added
window 28 added
window 90 added
window 180 added
window 365 added
Adding sales by store item dayofweek
window 7 added
window 14 added
window 28 added
window 56 added
window 84 added
```

```
window 84 added
window 168 added
window 336 added

Adding sales by store item day
window 365 added
Adding sales by item
window 7 added
```

3.1 One hot encoding

```
train1['id'] = np.nan
train1['is_train'] = True
test1['is_train'] = False
test1['sales'] = np.nan

# concat train,test
train_test = pd.concat([train1,test1],axis=0)

#Log transform
train_test['sales_log'] = np.log(train_test['sales']+1)
gc.collect()
train_test.shape

def one_hot_encoding(df,columns):
    print('Original shape',df.shape)
    df = pd.get_dummies(df,drop_first=True,columns=columns)
    print('After OHE', df.shape)
    return df


gc.collect()
train_test = one_hot_encoding(train_test,columns=['month','dayofweek'])
```

```
Original shape (958000, 94)
After OHE (958000, 109)
```

```
reduce_memory_usage(train_test)
```

```
Intial memory usage: 676.0787963867188 MB
Memory usage after complition:  252.15911865234375 MB
```

```
#plt.figure(figsize=(14,10))
#sns.heatmap(train_test1.corr(), cmap='coolwarm', annot=True,fmt='.2f')
```

4.0 Model selection

```
# Model
col_drop = ['id','is_train','sales','sales_log']
X = train_test[train_test['is_train'] == True].drop(col_drop, axis=1)
y = train_test[train_test['is_train'] == True]['sales_log']
test_new = train_test[train_test['is_train'] == False].drop(col_drop +['date'],axis=1)

# Time series based split
#Time series start time: "2013-01-01 00:00:00" and end time: "2017-12-31 00:00:00"
#Time series start time: "2018-01-01 00:00:00" and end time: "2018-03-31 00:00:00"
tra_start, tra_end = '2013-01-01','2016-12-31'
val_start, val_end = '2017-01-01','2017-12-31'
tes_start = '2018-01-01'

X_train = X[X.date.isin(pd.date_range(tra_start,tra_end))].drop(['date'],axis=1)
X_valid = X[X.date.isin(pd.date_range(val_start, val_end))].drop(['date'],axis=1)
y_train = y[X.date.isin(pd.date_range(tra_start,tra_end))]
y_valid = y[X.date.isin(pd.date_range(val_start, val_end))]
gc.collect()
X.shape,test_new.shape
```

```
((913000, 105), (45000, 104))
```

```
# SMAPE Systematic mean absolute Persent error
def smape(y_true,y_pred):

    n = len(y_pred)
    masked_arr = ~((y_pred==0)&(y_true==0))
```

```
    y_pred, y_true = y_pred[masked_arr], y_true[masked_arr]
    nom = np.abs(y_true - y_pred)
    denom = np.abs(y_true) + np.abs(y_pred)
    smape = 200/n * np.sum(nom/denom)
    return smape
def lgb_smape(pred,train_data):
    '''
    Custom evaluvation function
    '''
    label = train_data.get_label()
    smape_val = smape(np.expm1(pred), np.expm1(label))
    return 'SMAPE',smape_val, False
```

```
import sklearn
from sklearn.metrics import r2_score
def rscore(y_true,y_pred):
    return sklearn.metrics.r2_score(y_true, y_pred)

def lgb_rscore(pred,train_data):
    '''
    Custom evaluvation function
    '''
    label = train_data.get_label()
    rscore_val = rscore(np.expm1(pred), np.expm1(label))
    return 'RSCORE',rscore_val, False
```

```
import sklearn
from sklearn.metrics import r2_score
```

5.0 Models

a) Linear regression

```
from sklearn.linear_model import LinearRegression
model1 = LinearRegression()
model1.fit(X_train, y_train)
predict1 = model1.predict(X_valid)
```

```
y_pred_new=test_predict = model1.predict(test_new)
y_pred = model1.predict(X_valid)
print("rscore is",sklearn.metrics.r2_score(y_valid, y_pred))
```

```
rscore is 0.6558538493142778
```

b) XGBoost

```
from sklearn.model_selection import train_test_split
from xgboost import XGBRegressor
from sklearn.metrics import mean_absolute_error
#from sklearn.preprocessing import Imputer
```

```
#define model
my_model = XGBRegressor()
# Add silent=True to avoid printing out updates with each cycle
my_model.fit(X_train, y_train, verbose=False)
# make predictions
predictions = my_model.predict(X_valid)
print("Mean Absolute Error : " + str(mean_absolute_error(predictions,y_valid)))
```

```
[11:25:43] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:square
Mean Absolute Error : 0.12240535
```

```
y_pred_new=test_predict = my_model.predict(test_new)
y_pred = my_model.predict(X_valid)
print("rscore is",sklearn.metrics.r2_score(y_valid, y_pred))
```

```
rscore is 0.9233099242965672
```

```
from sklearn.model_selection import train_test_split
from xgboost import XGBRegressor
from sklearn.metrics import mean_absolute_error
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_log_error
from sklearn.metrics import mean_squared_error
```

c) Decision Tree Regressor¶

```
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_log_error
from sklearn.metrics import mean_squared_error

dec_reg_model = DecisionTreeRegressor(random_state=1)
dec_reg_model.fit(X_train, y_train)
```

```
DecisionTreeRegressor(ccp_alpha=0.0, criterion='mse', max_depth=None,
                      max_features=None, max_leaf_nodes=None,
                      min_impurity_decrease=0.0, min_impurity_split=None,
                      min_samples_leaf=1, min_samples_split=2,
                      min_weight_fraction_leaf=0.0, presort='deprecated',
                      random_state=1, splitter='best')
```

```
predictions = dec_reg_model.predict(X_valid)
```

```
print("Mean Absolute Error : " + str(mean_absolute_error(predictions,y_valid)))
```

```
Mean Absolute Error : 0.1710087944488003
```

```
## prediction on test data spliting of metadata
x_pred_dec = dec_reg_model.predict(X_valid)
print("Mean Squared Log Error is ", mean_squared_log_error(predictions,y_valid))
```

```
Mean Squared Log Error is  0.002518574556977172
```

```
print("Root Mean Squared Error is ", mean_squared_error(y_valid,predictions)**(0.5))
```

```
Root Mean Squared Error is  0.22365719975270779
```

```
y_pred_new=test_predict = dec_reg_model.predict(test_new)
y_pred = dec_reg_model.predict(X_valid)
print("rscore is",sklearn.metrics.r2_score(y_valid, y_pred))
```

```
rscore is 0.8446189984842214
```

d) Random Forest Regressor

```
from sklearn.ensemble import RandomForestRegressor
##RandomForest Regressor
ran_reg_model = RandomForestRegressor(random_state=1)
ran_reg_model.fit(X_train, y_train)
```

```
RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                      max_depth=None, max_features='auto', max_leaf_nodes=None,
                      max_samples=None, min_impurity_decrease=0.0,
                      min_impurity_split=None, min_samples_leaf=1,
                      min_samples_split=2, min_weight_fraction_leaf=0.0,
```

```
                           n_estimators=100, n_jobs=None, oob_score=False,
                           random_state=1, verbose=0, warm_start=False)
```

```
redictions = ran_reg_model.predict(X_valid)
print("Mean Absolute Error : " + str(mean_absolute_error(predictions,y_valid)))
```

```
    Mean Absolute Error : 0.1710087944488003
```

```
## prediction on test data spliting of metadata
x_pred_dec = ran_reg_model.predict(X_valid)
print("Mean Squared Log Error is ", mean_squared_log_error(predictions,y_valid))
```

```
    Mean Squared Log Error is  0.002518574556977172
```

```
print("Root Mean Squared Error is ", mean_squared_error(y_valid,predictions)**(0.5))
```

```
    Root Mean Squared Error is  0.22365719975270779
```

```
y_pred_new=test_predict = ran_reg_model.predict(test_new)
y_pred = ran_reg_model.predict(X_valid)
print("rscore is",sklearn.metrics.r2_score(y_valid, y_pred))
```

```
    rscore is 0.9092196878110362
```

b) Time Series Analysis

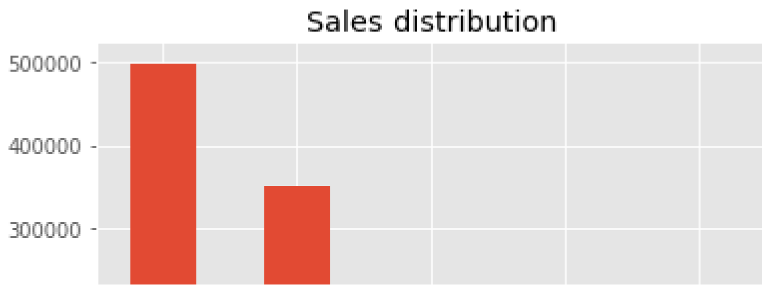
Distribution of sales

Now let us understand how the sales varies across all the items in all the stores

```
# Sales distribution across the train data
sales_df = train_df.copy(deep=True)
sales_df['sales_bins'] = pd.cut(sales_df.sales, [0, 50, 100, 150, 200, 250])
print('Max sale:', sales_df.sales.max())
print('Min sale:', sales_df.sales.min())
print('Avg sale:', sales_df.
sales.mean())
print()

# Total number of data points
total_points = pd.value_counts(sales_df.sales_bins).sum()
print('Sales bucket v/s Total percentage:')
display(pd.value_counts(sales_df.sales_bins).apply(lambda s: (s/total_points)*100))
```

```
Max sale: 231
Min sale: 0
Avg sale: 52.250286966046005

Sales bucket v/s Total percentage:
(0, 50]       54.591407
(50, 100]     38.388322
(100, 150]     6.709974
(150, 200]     0.308544
(200, 250]     0.001752
Name: sales_bins, dtype: float64
```

```
# Let us visualize the same
pd.value_counts(sales_df.sales_bins).plot(kind='bar', title='Sales distribution');
```



As we can see, almost 92% of sales are less than 100. Max, min and average sales are 231, 0 and 52.25 respectively.
So any prediction model has to deal with the skewness in the data appropriately.

How does sales vary across stores

Let us get a overview of sales distribution in the whole data.

```
# Let us understand the sales data distribution across the stores
store_df = train_df.copy()
sales_pivoted_df = pd.pivot_table(store_df, index='store', values=['sales','date'], columns='item', aggfunc=np.mean)
# Pivoted dataframe
display(sales_pivoted_df)
```

	sales										
item	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
store											
1	19.971522	53.148959	33.208105	19.956188	16.612815	53.060789	52.783680	69.472070	46.504929	66.354326	63.217
4	22.938664	61.715225	38.548193	23.086528	19.525192	61.270537	61.625411	80.819825	54.043812	77.047645	73.400

This pivoted dataframe has average sales per each store per each item.

Let use this dataframe and produce some interesting visualizations!

```
# Let us calculate the average sales of all the items by each store
sales_across_store_df = sales_pivoted_df.copy()
sales_across_store_df['avg_sale'] = sales_across_store_df.apply(lambda r: r.mean(), axis=1)
```

```
# Scatter plot of average sales per store
sales_store_data = go.Scatter(
    y = sales_across_store_df.avg_sale.values,
    mode='markers',
    marker=dict(
        size = sales_across_store_df.avg_sale.values,
        color = sales_across_store_df.avg_sale.values,
        colorscale='Viridis',
        showscale=True
    ),
    text = sales_across_store_df.index.values
)
data = [sales_store_data]

sales_store_layout = go.Layout(
    autosize= True,
    title= 'Scatter plot of avg sales per store',
    hovermode= 'closest',
    xaxis= dict(
        title= 'Stores',
        ticklen= 10,
        zeroline= False,
        gridwidth= 1,
```

```
        ),
        yaxis=dict(
            title= 'Avg Sales',
            ticklen= 10,
            zeroline= False,
            gridwidth= 1,
        ),
        showlegend= False
    )
    fig = go.Figure(data=data, layout=sales_store_layout)
    py.iplot(fig,filename='scatter_sales_store')
```

From the visualization, it is clear that the stores with ID 2 and 8 have higher average sales than the remaining stores and is a clear indication that they are doing good money!

Whereas store with ID 7 has very poor performance in terms of average sales.

How does sales vary across items

```
# Let us calculate the average sales of each of the item across all the stores
sales_across_item_df = sales_pivoted_df.copy()
# Aggregate the sales per item and add it as a new row in the same dataframe
sales_across_item_df.loc[11] = sales_across_item_df.apply(lambda r: r.mean(), axis=0)
# Note the 11th index row, which is the average sale of each of the item across all the stores
#display(sales_across_item_df.loc[11:])
avg_sales_per_item_across_stores_df = pd.DataFrame(data=[[i+1,a] for i,a in enumerate(sales_across_item_df.loc[11:].values[0
# And finally, sort by avg sale
avg_sales_per_item_across_stores_df.sort_values(by='avg_sale', ascending=False, inplace=True)
# Display the top 10 rows
display(avg_sales_per_item_across_stores_df.head())
```

	item	avg_sale
14	15	88.030778
27	28	87.881325
12	13	84.316594
17	18	84.275794
24	25	80.686418

Great! Let us visualize these average sales per item!

```
avg_sales_per_item_across_stores_sorted = avg_sales_per_item_across_stores_df.avg_sale.values
# Scatter plot of average sales per item
```

```
# Scatter plot of average sales per item…
sales_item_data = go.Bar(
    x=[i for i in range(0, 50)],
    y=avg_sales_per_item_across_stores_sorted,
    marker=dict(
        color=avg_sales_per_item_across_stores_sorted,
        colorscale='Blackbody',
        showscale=True
    ),
    text = avg_sales_per_item_across_stores_df.item.values
)
data = [sales_item_data]

sales_item_layout = go.Layout(
    autosize= True,
    title= 'Scatter plot of avg sales per item',
    hovermode= 'closest',
    xaxis= dict(
        title= 'Items',
        ticklen= 55,
        zeroline= False,
        gridwidth= 1,
    ),
    yaxis=dict(
        title= 'Avg Sales',
        ticklen= 10,
        zeroline= False,
        gridwidth= 1,
    ),
    showlegend= False
)
fig = go.Figure(data=data, layout=sales_item_layout)
py.iplot(fig,filename='scatter_sales_item')
```

Amazing! The sales is uniformly distributed across all the items.

Top items with highest average sale are 15, 28, 13, 18 and with least average sales are 5, 1, 41 and so on.

Time-series visualization of the sales

Let us see how sales of a given item in a given store varies in a span of 5 years.

```
store_item_df = train_df.copy()
# First, let us filterout the required data
store_id = 10   # Some store
item_id = 40    # Some item
print('Before filter:', store_item_df.shape)
store_item_df = store_item_df[store_item_df.store == store_id]
store_item_df = store_item_df[store_item_df.item == item_id]
```

```
print('After filter:', store_item_df.shape)
#display(store_item_df.head())

# Let us plot this now
store_item_ts_data = [go.Scatter(
    x=store_item_df.date,
    y=store_item_df.sales)]
py.iplot(store_item_ts_data)
```

```
Before filter: (913000, 4)
```

Woww! Clearly there is a pattern here! Feel free to play around with different store and item IDs.
Almost all the items and store combination has this pattern!

The sales go high in June, July and August months. The sales will be lowest in December, January and February months. That's something!!

Let us make it more interesting. What if we aggregate the sales on a montly basis and compare different items and stores.
This should help us understand how different item sales behave at a high level.

```
multi_store_item_df = train_df.copy()
# First, let us filterout the required data
store_ids = [1, 1, 1, 1]   # Some stores
item_ids = [10, 20, 30, 40]    # Some items
print('Before filter:', multi_store_item_df.shape)
multi_store_item_df = multi_store_item_df[multi_store_item_df.store.isin(store_ids)]
multi_store_item_df = multi_store_item_df[multi_store_item_df.item.isin(item_ids)]
print('After filter:', multi_store_item_df.shape)
#display(multi_store_item_df)
# TODO Monthly avg sales

# Let us plot this now
multi_store_item_ts_data = []
for st,it in zip(store_ids, item_ids):
    flt = multi_store_item_df[multi_store_item_df.store == st]
    flt = flt[flt.item == it]
    multi_store_item_ts_data.append(go.Scatter(x=flt.date, y=flt.sales, name = "Store:" + str(st) + ",Item:" + str(it)))
py.iplot(multi_store_item_ts_data)
```

```
Before filter: (913000, 4)
After filter: (7304, 4)
```

Interesting!!
Though the pattern remains same across different stores and items combinations, the **actual sale value consitently varies with the same scale**.

As we can see in the visualization, item 10 has consistently highest sales through out the span of 5 years!
This is an interesting behaviour that can be seen across almost all the items.

```
test_new
```

	store	item	day	year	is_month_end	is_month_start	weekofyear	2017-12-31_store_item_mean_1	2017-12-31_store_item_mean_3	31
0	1	1	1	2018	0	1	1	27	20.333334	
1	1	1	2	2018	0	0	1	27	20.333334	
2	1	1	3	2018	0	0	1	27	20.333334	
3	1	1	4	2018	0	0	1	27	20.333334	
4	1	1	5	2018	0	0	1	27	20.333334	
...	...	...	...	...	...	...	...	...	...	
44995	10	50	27	2018	0	0	13	62	65.000000	
44996	10	50	28	2018	0	0	13	62	65.000000	
44997	10	50	29	2018	0	0	13	62	65.000000	
44998	10	50	30	2018	0	0	13	62	65.000000	
44999	10	50	31	2018	1	0	13	62	65.000000	

45000 rows × 104 columns

5.0 Model

```
def lgb_model(X_train, X_valid, y_valid, y_test,test_new):
    lgb_param = {}
    lgb_param['boosting_type'] ='gbdt'
    lgb_param['max_depth'] = 7
    lgb_param['num_leaves'] = 2**7
    lgb_param['learning_rate'] = 0.05
    #lgb_param['n_estimators'] = 3000
    lgb_param['feature_fraction'] = 0.9
    lgb_param['bagging_fraction'] = 0.9
    lgb_param['lambda_l1'] = 0.06
    lgb_param['lambda_l2'] =  0.1
    lgb_param['random_state'] = seed
    lgb_param['n_jobs'] = 4
    lgb_param['silent'] = -1
    lgb_param['verbose'] = -1
    lgb_param['metric'] = 'mae'

    model = lgb.LGBMRegressor(**lgb_param)
    lgb_train = lgb.Dataset(X_train,y_train)
    lgb_valid = lgb.Dataset(X_valid,y_valid)
    valid_set = [lgb_train,lgb_valid]
    #model = lgb.train(params=lgb_param,train_set=lgb_train,valid_sets=valid_set,num_boost_round= 300,
    #                  feval=lgb_rscore,early_stopping_rounds=20,)
    model = lgb.train(params=lgb_param,train_set=lgb_train,valid_sets=valid_set,num_boost_round= 300,
                     feval=lgb_smape,early_stopping_rounds=20,)
    print('-'*10,'*'*20,'-'*10)
    #model.fit(X_train,y_train, eval_set= [(X_train,y_train),(X_valid,y_valid)],
    #          eval_metric ='rmse',early_stopping_rounds=20,verbose=100)

    y_pred = model.predict(X_valid)
    print('Root mean_squared_error','-'*20 ,np.sqrt(mean_squared_error(y_valid, y_pred)))
    print("rscore is",sklearn.metrics.r2_score(y_valid, y_pred))
    y_pred_new = model.predict(test_new)
    #print("rscore is",sklearn.metrics.r2_score(y_valid, y_pred_new))
```

```
    return y_pred_new, model
```

```
# Model training
y_pred_new, model = lgb_model(X_train, X_valid, y_valid, y_valid,test_new)
```

```
[178]   training's l1: 0.12378  training's SMAPE: 12.7153      valid_1's l1: 0.120151  valid_1's SMAPE: 12.2895
[179]   training's l1: 0.123772 training's SMAPE: 12.7145      valid_1's l1: 0.120146  valid_1's SMAPE: 12.2891
[180]   training's l1: 0.123766 training's SMAPE: 12.7139      valid_1's l1: 0.120145  valid_1's SMAPE: 12.289
[181]   training's l1: 0.123759 training's SMAPE: 12.7132      valid_1's l1: 0.120143  valid_1's SMAPE: 12.2888
[182]   training's l1: 0.123755 training's SMAPE: 12.7128      valid_1's l1: 0.120133  valid_1's SMAPE: 12.2877
[183]   training's l1: 0.123748 training's SMAPE: 12.7121      valid_1's l1: 0.120133  valid_1's SMAPE: 12.2877
[184]   training's l1: 0.123739 training's SMAPE: 12.7111      valid_1's l1: 0.120135  valid_1's SMAPE: 12.288
[185]   training's l1: 0.123733 training's SMAPE: 12.7105      valid_1's l1: 0.120135  valid_1's SMAPE: 12.288
[186]   training's l1: 0.123725 training's SMAPE: 12.7097      valid_1's l1: 0.120135  valid_1's SMAPE: 12.2879
[187]   training's l1: 0.123719 training's SMAPE: 12.7091      valid_1's l1: 0.120132  valid_1's SMAPE: 12.2876
[188]   training's l1: 0.123712 training's SMAPE: 12.7085      valid_1's l1: 0.120128  valid_1's SMAPE: 12.2873
[189]   training's l1: 0.123705 training's SMAPE: 12.7077      valid_1's l1: 0.120123  valid_1's SMAPE: 12.2867
[190]   training's l1: 0.123696 training's SMAPE: 12.7068      valid_1's l1: 0.120123  valid_1's SMAPE: 12.2868
[191]   training's l1: 0.123689 training's SMAPE: 12.7061      valid_1's l1: 0.120115  valid_1's SMAPE: 12.2859
[192]   training's l1: 0.123684 training's SMAPE: 12.7055      valid_1's l1: 0.120113  valid_1's SMAPE: 12.2857
[193]   training's l1: 0.123675 training's SMAPE: 12.7046      valid_1's l1: 0.120113  valid_1's SMAPE: 12.2857
[194]   training's l1: 0.123666 training's SMAPE: 12.7037      valid_1's l1: 0.120114  valid_1's SMAPE: 12.2859
[195]   training's l1: 0.123658 training's SMAPE: 12.7029      valid_1's l1: 0.120117  valid_1's SMAPE: 12.2862
[196]   training's l1: 0.123651 training's SMAPE: 12.7022      valid_1's l1: 0.120117  valid_1's SMAPE: 12.2862
[197]   training's l1: 0.123645 training's SMAPE: 12.7016      valid_1's l1: 0.120118  valid_1's SMAPE: 12.2863
[198]   training's l1: 0.123639 training's SMAPE: 12.701       valid_1's l1: 0.120117  valid_1's SMAPE: 12.2861
[199]   training's l1: 0.123632 training's SMAPE: 12.7003      valid_1's l1: 0.120116  valid_1's SMAPE: 12.286
[200]   training's l1: 0.123627 training's SMAPE: 12.6998      valid_1's l1: 0.120116  valid_1's SMAPE: 12.2861

[201]   training's l1: 0.123621 training's SMAPE: 12.6991      valid_1's l1: 0.120108  valid_1's SMAPE: 12.2852
[202]   training's l1: 0.123611 training's SMAPE: 12.6981      valid_1's l1: 0.120109  valid_1's SMAPE: 12.2853
[203]   training's l1: 0.123602 training's SMAPE: 12.6972      valid_1's l1: 0.120109  valid_1's SMAPE: 12.2854
[204]   training's l1: 0.123593 training's SMAPE: 12.6963      valid_1's l1: 0.12011   valid_1's SMAPE: 12.2854
[205]   training's l1: 0.123586 training's SMAPE: 12.6956      valid_1's l1: 0.12011   valid_1's SMAPE: 12.2854
[206]   training's l1: 0.12358  training's SMAPE: 12.695       valid_1's l1: 0.12011   valid_1's SMAPE: 12.2855
[207]   training's l1: 0.123571 training's SMAPE: 12.694       valid_1's l1: 0.120109  valid_1's SMAPE: 12.2854
[208]   training's l1: 0.123563 training's SMAPE: 12.6933      valid_1's l1: 0.120106  valid_1's SMAPE: 12.2851
[209]   training's l1: 0.123554 training's SMAPE: 12.6924      valid_1's l1: 0.120108  valid_1's SMAPE: 12.2852
[210]   training's l1: 0.12355  training's SMAPE: 12.6919      valid_1's l1: 0.120109  valid_1's SMAPE: 12.2854
[211]   training's l1: 0.123544 training's SMAPE: 12.6914      valid_1's l1: 0.12011   valid_1's SMAPE: 12.2855
[212]   training's l1: 0.123541 training's SMAPE: 12.691       valid_1's l1: 0.120104  valid_1's SMAPE: 12.2849
```

```
[213]	training's l1: 0.123532	training's SMAPE: 12.6901	valid_1's l1: 0.120105	valid_1's SMAPE: 12.285
[214]	training's l1: 0.123526	training's SMAPE: 12.6895	valid_1's l1: 0.120107	valid_1's SMAPE: 12.2852
[215]	training's l1: 0.123521	training's SMAPE: 12.689	valid_1's l1: 0.120108	valid_1's SMAPE: 12.2853
[216]	training's l1: 0.123516	training's SMAPE: 12.6884	valid_1's l1: 0.12011	valid_1's SMAPE: 12.2855
[217]	training's l1: 0.123511	training's SMAPE: 12.6879	valid_1's l1: 0.120105	valid_1's SMAPE: 12.285
[218]	training's l1: 0.123507	training's SMAPE: 12.6876	valid_1's l1: 0.120098	valid_1's SMAPE: 12.2843
[219]	training's l1: 0.123499	training's SMAPE: 12.6868	valid_1's l1: 0.120094	valid_1's SMAPE: 12.2839
[220]	training's l1: 0.123493	training's SMAPE: 12.6861	valid_1's l1: 0.120095	valid_1's SMAPE: 12.284
[221]	training's l1: 0.123486	training's SMAPE: 12.6854	valid_1's l1: 0.120095	valid_1's SMAPE: 12.284
[222]	training's l1: 0.123478	training's SMAPE: 12.6846	valid_1's l1: 0.120094	valid_1's SMAPE: 12.2839
[223]	training's l1: 0.12347	training's SMAPE: 12.6838	valid_1's l1: 0.120094	valid_1's SMAPE: 12.2839
[224]	training's l1: 0.123463	training's SMAPE: 12.6831	valid_1's l1: 0.120094	valid_1's SMAPE: 12.2839
[225]	training's l1: 0.123458	training's SMAPE: 12.6826	valid_1's l1: 0.120092	valid_1's SMAPE: 12.2837
[226]	training's l1: 0.123451	training's SMAPE: 12.6819	valid_1's l1: 0.120093	valid_1's SMAPE: 12.2838
[227]	training's l1: 0.123444	training's SMAPE: 12.6812	valid_1's l1: 0.120095	valid_1's SMAPE: 12.284
[228]	training's l1: 0.123435	training's SMAPE: 12.6802	valid_1's l1: 0.120098	valid_1's SMAPE: 12.2843
[229]	training's l1: 0.123427	training's SMAPE: 12.6794	valid_1's l1: 0.1201	valid_1's SMAPE: 12.2845
[230]	training's l1: 0.12342	training's SMAPE: 12.6787	valid_1's l1: 0.120097	valid_1's SMAPE: 12.2841
[231]	training's l1: 0.123415	training's SMAPE: 12.6782	valid_1's l1: 0.120097	valid_1's SMAPE: 12.2842
[232]	training's l1: 0.123406	training's SMAPE: 12.6773	valid_1's l1: 0.120096	valid_1's SMAPE: 12.2841
[233]	training's l1: 0.123401	training's SMAPE: 12.6768	valid_1's l1: 0.120097	valid_1's SMAPE: 12.2842
[234]	training's l1: 0.123395	training's SMAPE: 12.6762	valid_1's l1: 0.120094	valid_1's SMAPE: 12.2839
[235]	training's l1: 0.12339	training's SMAPE: 12.6757	valid_1's l1: 0.120096	valid_1's SMAPE: 12.2841
[236]	training's l1: 0.123385	training's SMAPE: 12.6752	valid_1's l1: 0.120096	valid_1's SMAPE: 12.2841
[237]	training's l1: 0.123379	training's SMAPE: 12.6746	valid_1's l1: 0.120096	valid_1's SMAPE: 12.2841
```

6.0 Model evaluation

```
#print('Root mean_squared_error',np.sqrt(mean_squared_error(y_test, y_pred)))
```

```
#182500, 45000]
```

```
#len(y_valid)
#len(X_valid)
```

```
#len(X_valid)
```

```
#len(test_new)
```

```
#len(test1)
```

```
#len(y_valid)
```

```
#print(y_valid.shape)
```

```
#print(y_pred_new.shape)
```

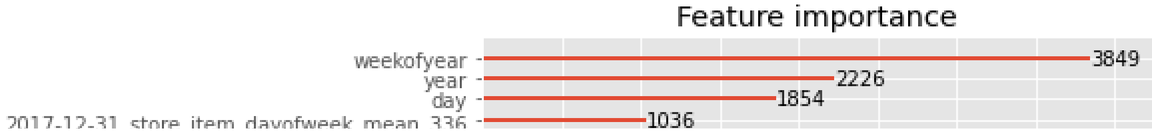
```
#len(y_pred)
```

```
#len(y_pred_new)#
```

```
#len(X_train)
```

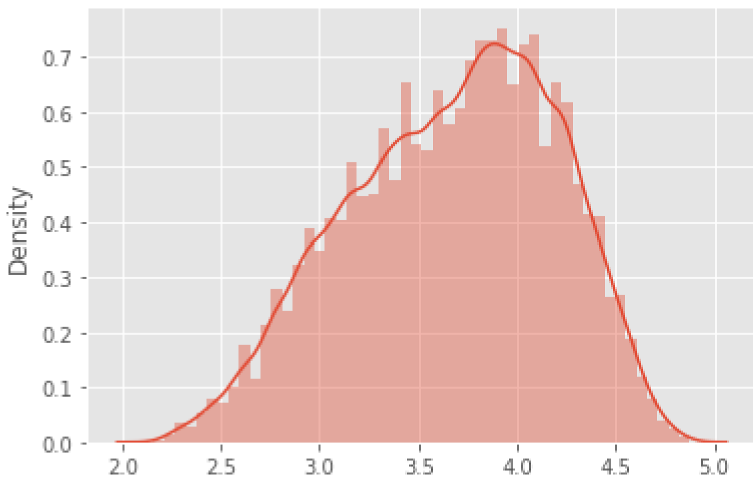
```
#print("rscore is",sklearn.metrics.r2_score(y_valid,y_pred_new))
```

```
# Feature importance
lgb.plot_importance(model,max_num_features=20);
```




```
sns.distplot(y_pred_new)
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2557: FutureWarning:
`distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `disp
<matplotlib.axes._subplots.AxesSubplot at 0x7f6fac361050>
```



END OF LGBM MODEL

##

ARIMA MODEL

Import the packages

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt  # Matlab-style plotting
%matplotlib inline
import seaborn as sns
import warnings
warnings.filterwarnings('ignore') #to ignore if any warnings takes place during the run time.
#import statsmodels.api as sm
import os
#os.chdir("/content/drive/My Drive/Dataset/Dataset")
```

```
#read the data
df=pd.read_csv('/content/drive/My Drive/Dataset/train.csv')
df.head()
```

	date	store	item	sales
0	2013-01-01	1	1	13
1	2013-01-02	1	1	11
2	2013-01-03	1	1	14
3	2013-01-04	1	1	13
4	2013-01-05	1	1	10

```
#check for missing values in train data
df.isnull().sum()
 #No missing valuues
```

```
date     0
store    0
item     0
sales    0
dtype: int64
```

Here for better understanding of the data, We can eloborate as month and weekday wise.

```
df['date'] = pd.to_datetime(df['date'], format="%Y-%m-%d") #If need extract year, month and day to new columns:

# per 1 store, 1 item
train_df = df[df['store']==1]
train_df = train_df[df['item']==1]

# train_df = train_df.set_index('date')
train_df['year'] = df['date'].dt.year
train_df['month'] = df['date'].dt.month
train_df['day'] = df['date'].dt.dayofyear
train_df['weekday'] = df['date'].dt.weekday

train_df.head()
```

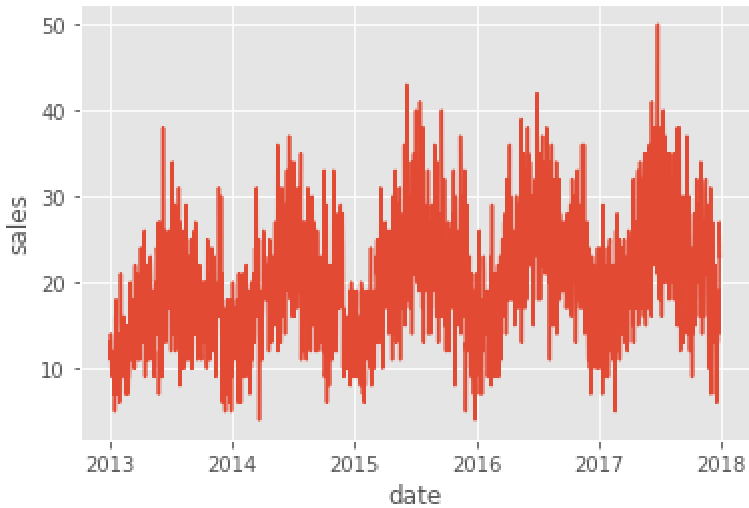
	date	store	item	sales	year	month	day	weekday
0	2013-01-01	1	1	13	2013	1	1	1
1	2013-01-02	1	1	11	2013	1	2	2
2	2013-01-03	1	1	14	2013	1	3	3
3	2013-01-04	1	1	13	2013	1	4	4
4	2013-01-05	1	1	10	2013	1	5	5

Below plots are for checking the seasonality, trends and outliers.

```
sns.lineplot(x="date", y="sales",legend = 'full' , data=train_df)
```

```
sns.lineplot(x="date", y="sales",legend = 'full' , data=train_df)
```

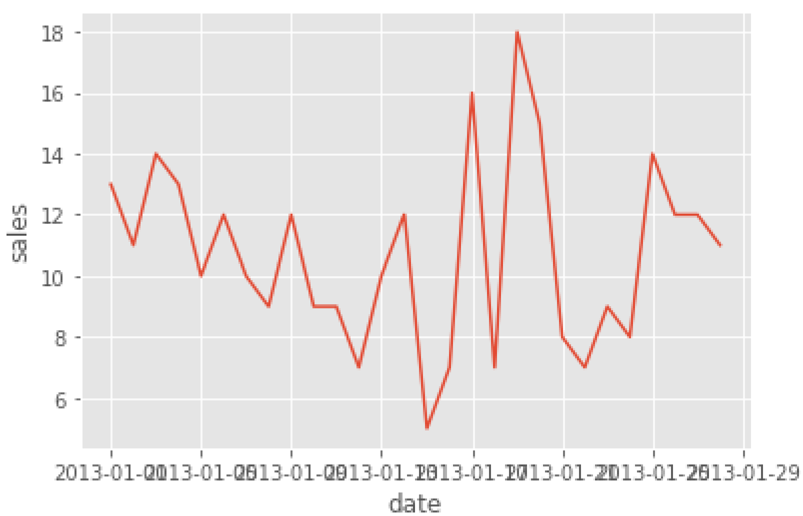
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f6fac3de810>
```



Double-click (or enter) to edit

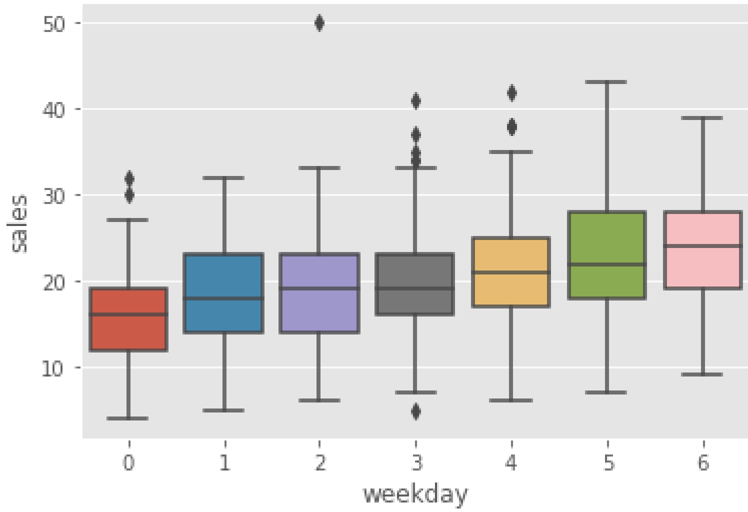
```
sns.lineplot(x="date", y="sales",legend = 'full' , data=train_df[:28])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f6fad732d10>
```



```
sns.boxplot(x="weekday", y="sales", data=train_df)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f6faed19390>
```




```
train_df = train_df.set_index('date')
train_df['sales'] = train_df['sales'].astype(float)
train_df.head()
```

date	store	item	sales	year	month	day	weekday
2013-01-01	1	1	13.0	2013	1	1	1
2013-01-02	1	1	11.0	2013	1	2	2
2013-01-03	1	1	14.0	2013	1	3	3
2013-01-04	1	1	13.0	2013	1	4	4
2013-01-05	1	1	10.0	2013	1	5	5

Time series decomposition

Think of the time series ytyt as consisting of three components: a seasonal component, a trend-cycle component (containing both trend and cycle), and a remainder component (containing anything else in the time series).

1. Additive model
2. Multiplicative model

The additive model is most appropriate if the magnitude of the seasonal fluctuations or the variation around the trend-cycle does not vary with the level of the time series.

When the variation in the seasonal pattern, or the variation around the trend-cycle, appears to be proportional to the level of the time series, then a multiplicative model is more appropriate.

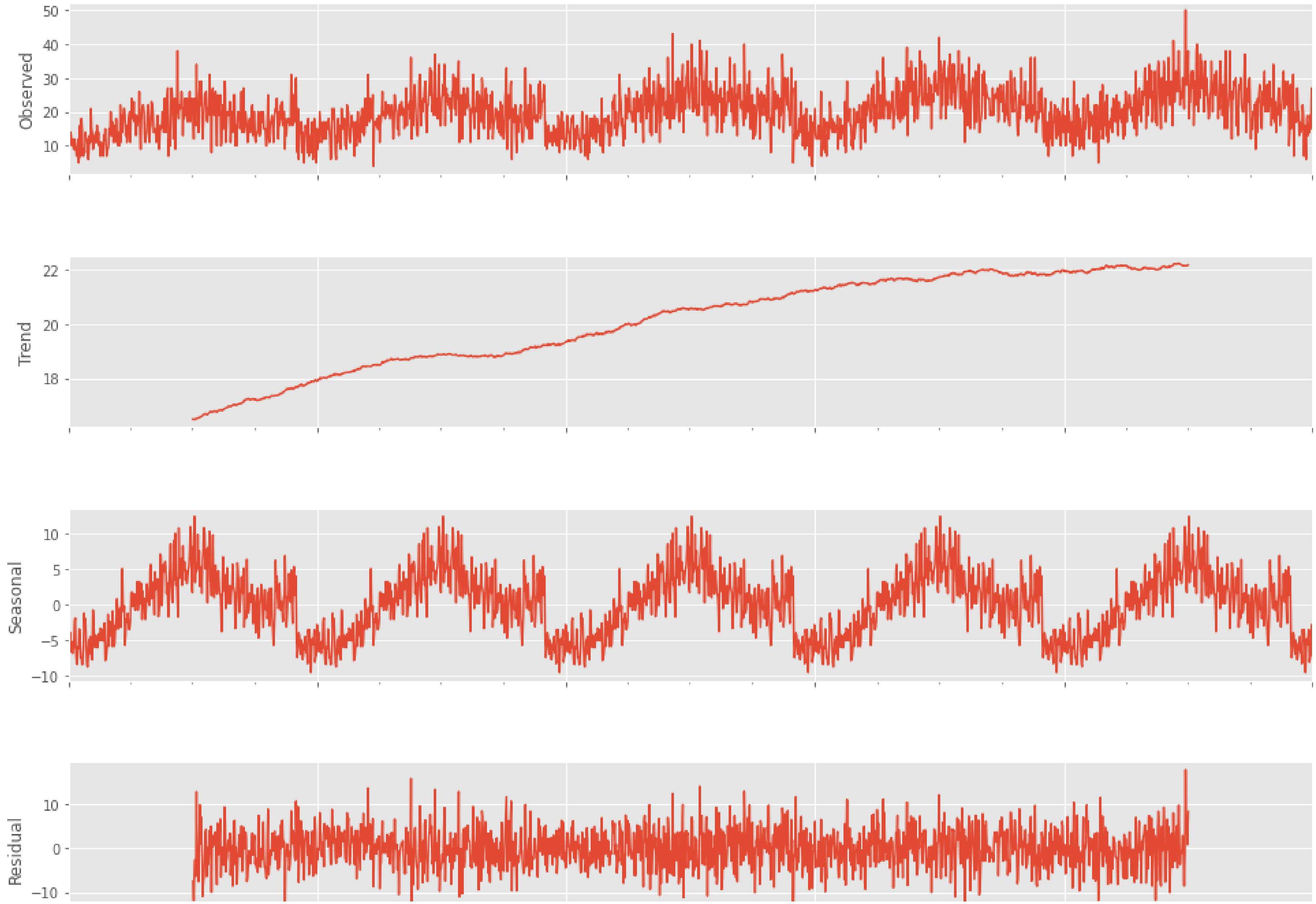
play this quiz you will come familiar with additive or multiplicative https://kourentzes.com/forecasting/2014/11/09/additive-and-multiplicative-seasonality/

Should I use an additive model or a multiplicative model?

Choose the multiplicative model when the magnitude of the seasonal pattern in the data depends on the magnitude of the data. In other words, the magnitude of the seasonal pattern increases as the data values increase, and decreases as the data values decrease. Choose the additive model when the magnitude of the seasonal pattern in the data does not depend on the magnitude of the data. In other words, the magnitude of the seasonal pattern does not change as the series goes up or down. If the pattern in the data is not very obvious, and you have trouble choosing between the additive and multiplicative procedures, you can try both and

```
from statsmodels.tsa.seasonal import seasonal_decompose
result = seasonal_decompose(train_df['sales'], model='additive', freq=365)
fig = plt.figure()
fig = result.plot()
fig.set_size_inches(15, 12)
```

```
<Figure size 432x288 with 0 Axes>
```



The yearly pattern is very obvious. and also we can see a upwards trend. Which means this data is not stationary.

Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit

```
from statsmodels.tsa.stattools import adfuller
def test_stationarity(timeseries, window = 12, cutoff = 0.01):

  #Determing rolling statistics
    rolmean = timeseries.rolling(window).mean()
    rolstd = timeseries.rolling(window).std()

    fig= plt.figure(figsize=(12,8))
    orig = plt.plot(timeseries, color='orange',label='Original')
    mean = plt.plot(rolmean, color='red', label='Rolling Mean')
    std = plt.plot(rolstd, color='blue', label = 'Rolling Std')
    plt.legend(loc='best')
    plt.title('Rolling Mean & Standard Deviation')
    plt.show()


      #Perform Dickey-Fuller test:
    print('Results of Dickey-Fuller Test:')
    dftest = adfuller(timeseries, autolag='AIC', maxlag = 20 )
    dfoutput = pd.Series(dftest[0:4], index=['Test Statistic','p-value','#Lags Used','Number of Observations Used'])
    for key,value in dftest[4].items():
        dfoutput['Critical Value (%s)'%key] = value
    pvalue = dftest[1]
    if pvalue < cutoff:
        print('p-value = %.4f. The series is likely stationary.' % pvalue)
    else:
        print('p-value = %.4f. The series is likely non-stationary.' % pvalue)

    print(dfoutput)
```

```
    print(dfoutput)
```

```
test_stationarity(train_df['sales'])
```

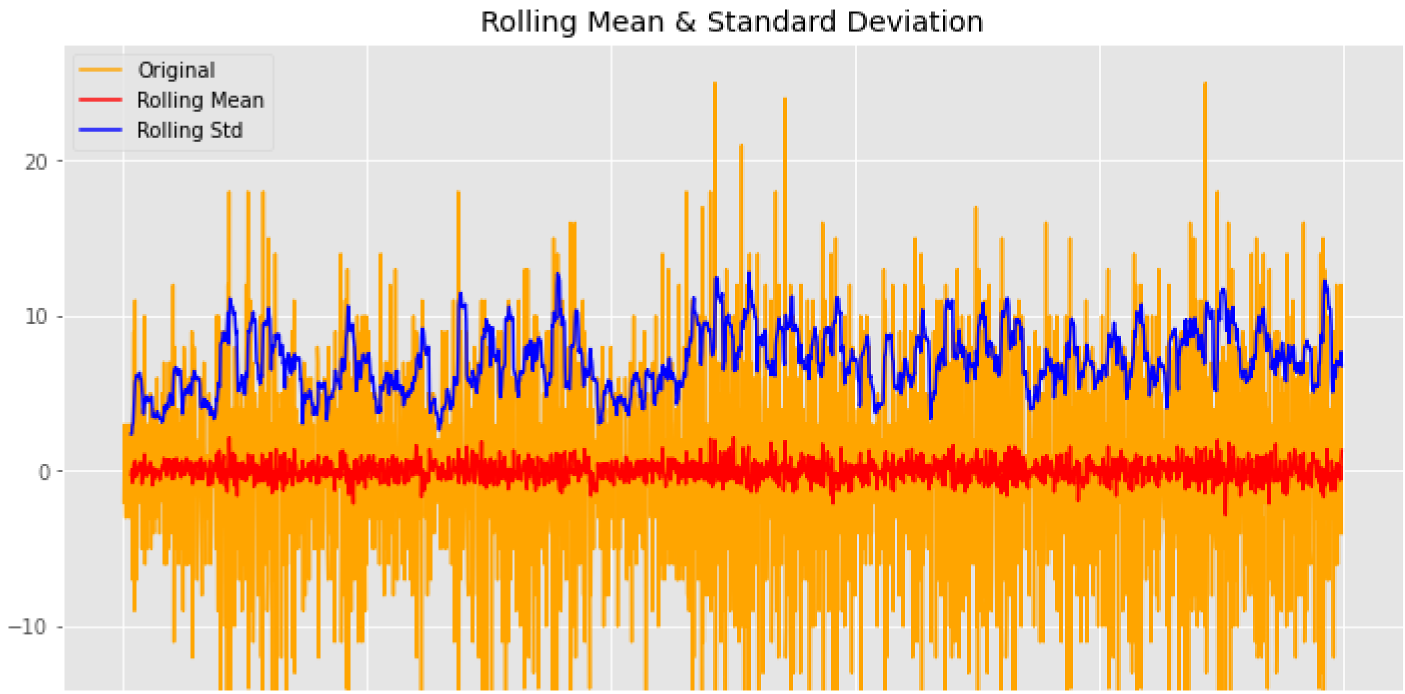
How to find whether our data is stationary or not ?

the smaller p-value, the more likely it's stationary. Here our p-value is 0.036. It's actually not bad, if we use a 5% Critical Value(CV), this series would be considered stationary. But as we just visually found an upward trend, we want to be more strict, we use 1% CV. To get a stationary data, there's many techiniques. We can use log, differencing etc...

NOTE

If the **p-value** is less than 5%(significance level) or If the **Test Static** value is greater than than the **Critical value** than our data is stationary

```
#this is for reducing trend and seasonality
first_diff = train_df.sales - train_df.sales.shift(1)
first_diff = first_diff.dropna(inplace = False)
test_stationarity(first_diff, window = 12)
```



ACF (Auto Corelation Function) and (Partial Auto Corelation Function)

What is ACF ?

For Instance today stock price we predicted based on yesterday stock price the ACF will tell how much strongly they are corelated, If todays value is depended on day before yesterday than ACF will tell how strong they are and how many days required to predict the todays value.

What is PACF?

If we want to calculate the corelation between today and yesterday we have to take the corelation of day before yesterday because todays value depends upon the yesterday time spot. So this is the reason we use PACF.

** PACF- AR model**

ACF- MA model

image.png

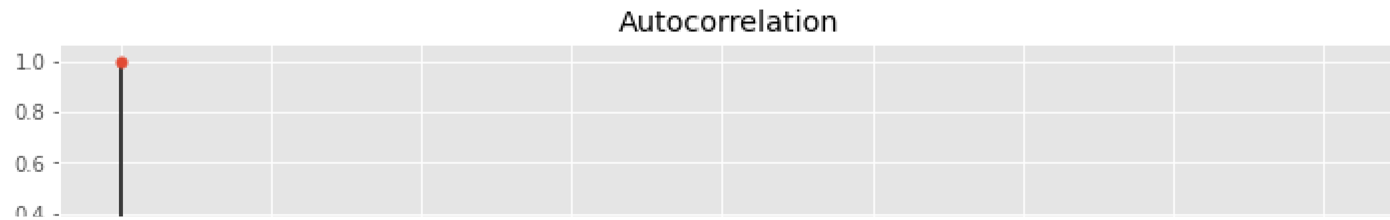
by the above images we can observe that, the lines which crosses the blue dotted lines in PACF and ACF those lines are considered to be that many days are required to predict the todays value. **For example in above PACF plot that has only three lines which crossed the blue dotted lines so last three days values are required to predict the todays value**, similarly ACF plot also but for model we should not consider ACF-MA model because many lines crossed the blue threshold line, so it will create the model complex. So we should select only PACF-AR model to predict

if you want to know more about PACF and ACF go through this link https://www.youtube.com/watch?v=5Q5p6eVM7zM

```
import statsmodels.api as sm

fig = plt.figure(figsize=(12,8))
ax1 = fig.add_subplot(211)
fig = sm.graphics.tsa.plot_acf(train_df.sales, lags=40, ax=ax1)
ax2 = fig.add_subplot(212)
fig = sm.graphics.tsa.plot_pacf(train_df.sales, lags=40, ax=ax2)      #lags=40
```

By seeing the above plots, there are lots of significant plots so as previously i explained if more lines crossed the blue line, than the model will get complex so go for the first difference.

```
fig = plt.figure(figsize=(12,8))
ax1 = fig.add_subplot(211)
fig = sm.graphics.tsa.plot_acf(first_diff, lags=40, ax=ax1)
ax2 = fig.add_subplot(212)
fig = sm.graphics.tsa.plot_pacf(first_diff, lags=40, ax=ax2)

# Here we can see the acf and pacf both has a recurring pattern every 7 periods. Indicating a weekly pattern exists.
# Any time you see a regular pattern like that in one of these plots, you should suspect that there is some sort of
# significant seasonal thing going on. Then we should start to consider SARIMA to take seasonality into accuont
```



How to determin p, d, q

It's easy to determin I. In our case, we see the first order differencing make the **ts stationary. I = 1.**

In our case, it's clearly that within 6 lags the AR is significant. Which means, we can use **AR = 6 (6 lines are crossed the blue lines so 6past days are required to predict)**

To avoid the potential for incorrectly specifying the MA order (in the case where the MA is first tried then the MA order is being set to 0), it may often make sense to extend the lag observed from the last significant term in the PACF.

What is interesting is that when the AR model is appropriately specified, the the residuals from this model can be used to directly observe the uncorrelated error. This residual can be used to further investigate alternative MA and ARMA model specifications directly by regression.

Assuming an AR(s) model were computed, then I would suggest that the next step in identification is to estimate an MA model with s-1 lags in the uncorrelated errors derived from the regression. The parsimonious MA specification might be considered and this might be compared with a more parsimonious AR specification. Then ARMA models might also be analysed.

0 5 10 15 20 25 30 35 40

```
arima_mod6 = sm.tsa.ARIMA(train_df.sales, (6,1,0)).fit(disp=False)
print(arima_mod6.summary())
```

```
                             ARIMA Model Results
==============================================================================
Dep. Variable:                D.sales   No. Observations:                 1825
Model:                 ARIMA(6, 1, 0)   Log Likelihood               -5597.668
Method:                       css-mle   S.D. of innovations              5.195
Date:                Tue, 25 May 2021   AIC                          11211.335
Time:                        12:39:32   BIC                          11255.410
Sample:                    01-02-2013   HQIC                         11227.594
                         - 12-31-2017
=================================================================================
                    coef    std err          z      P>|z|      [0.025      0.975]
```

```
------------------------------------------------------------------------------
const              0.0039      0.025      0.152      0.879      -0.046       0.054
ar.L1.D.sales     -0.8174      0.022    -37.921      0.000      -0.860      -0.775
ar.L2.D.sales     -0.7497      0.026    -28.728      0.000      -0.801      -0.699
ar.L3.D.sales     -0.6900      0.028    -24.665      0.000      -0.745      -0.635
ar.L4.D.sales     -0.6138      0.028    -21.950      0.000      -0.669      -0.559
ar.L5.D.sales     -0.5247      0.026    -20.132      0.000      -0.576      -0.474
ar.L6.D.sales     -0.3892      0.022    -18.064      0.000      -0.431      -0.347
                                    Roots
=============================================================================
                  Real          Imaginary           Modulus         Frequency
-----------------------------------------------------------------------------
AR.1            0.6842           -0.8982j            1.1292           -0.1464
AR.2            0.6842           +0.8982j            1.1292            0.1464
AR.3           -1.0869           -0.5171j            1.2037           -0.4293
AR.4           -1.0869           +0.5171j            1.2037            0.4293
AR.5           -0.2714           -1.1477j            1.1794           -0.2870
AR.6           -0.2714           +1.1477j            1.1794            0.2870
-----------------------------------------------------------------------------
```

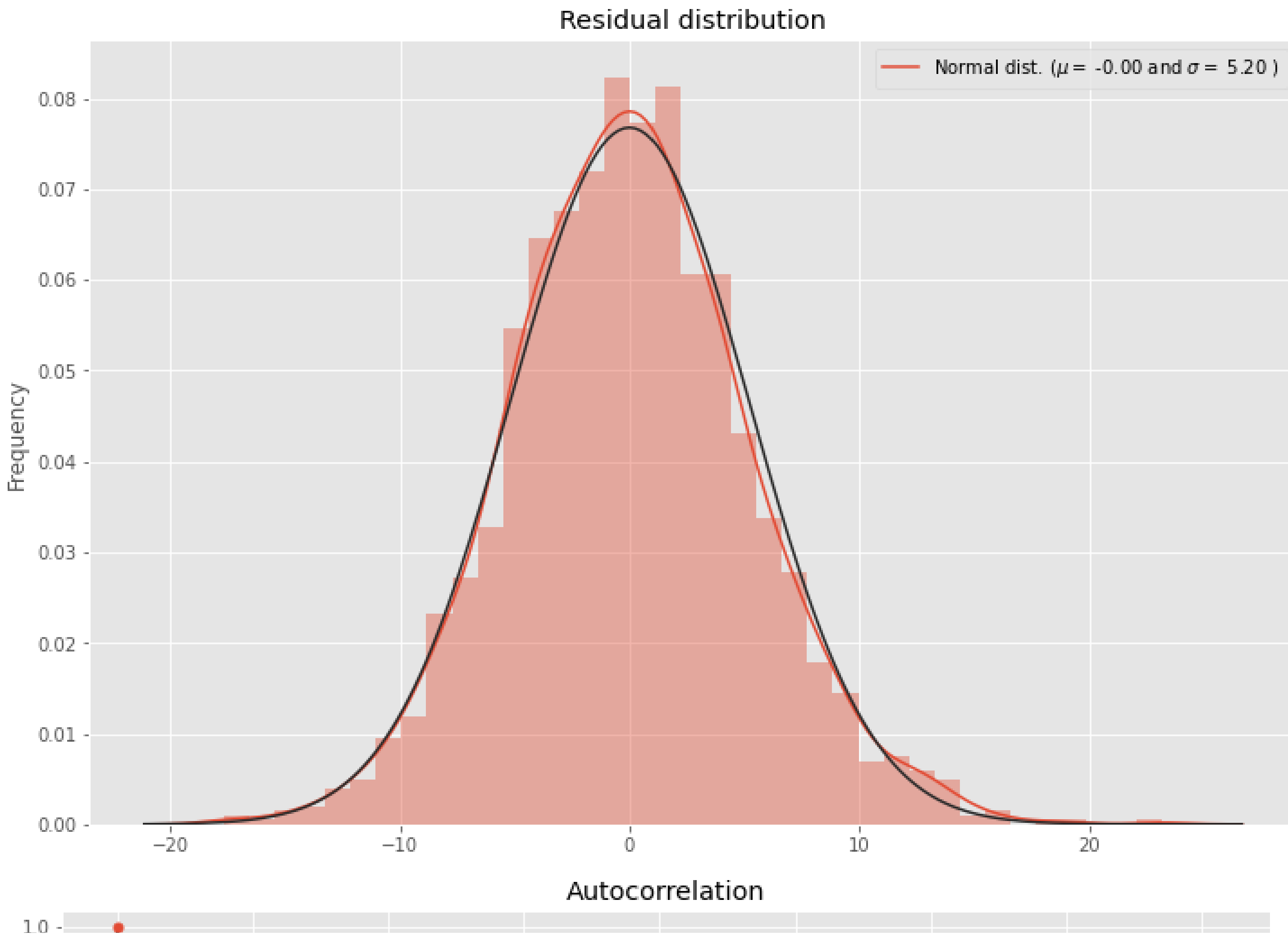
Analyze the result

To see how our first model perform, we can plot the residual distribution. See if it's normal dist. And the ACF and PACF. For a good model, we want to see the residual is normal distribution. And ACF, PACF has not significant terms.

```
from scipy import stats
from scipy.stats import normaltest

resid = arima_mod6.resid
print(normaltest(resid))
# returns a 2-tuple of the chi-squared statistic, and the associated p-value. the p-value is very small, meaning
# the residual is not a normal distribution

fig = plt.figure(figsize=(12,8))
ax0 = fig.add_subplot(111)

sns.distplot(resid ,fit = stats.norm, ax = ax0) # need to import scipy.stats

# Get the fitted parameters used by the function
```

```
    (mu, sigma) = stats.norm.fit(resid)

    #Now plot the distribution using
    plt.legend(['Normal dist. ($\mu=$ {:.2f} and $\sigma=$ {:.2f} )'.format(mu, sigma)], loc='best')
    plt.ylabel('Frequency')
    plt.title('Residual distribution')


    # ACF and PACF
    fig = plt.figure(figsize=(12,8))
    ax1 = fig.add_subplot(211)
    fig = sm.graphics.tsa.plot_acf(arima_mod6.resid, lags=40, ax=ax1)
    ax2 = fig.add_subplot(212)
    fig = sm.graphics.tsa.plot_pacf(arima_mod6.resid, lags=40, ax=ax2)
```

```
NormaltestResult(statistic=16.426387689817304, pvalue=0.0002710536340863827)
```



Although the graph looks very like a normal distribution. But it failed the test. Also we see a recurring correlation exists in both ACF and PACF. So we need to deal with seasonality.

When the plots of ACF and PACF are similar or any sesaonality is present between them than we need to apply **SARIMA** model, which it is extended model of **ARIMA**

What is SARIMA and what is the use of it ?

ARIMA, is one of the most widely used forecasting methods for univariate time series data forecasting, but it does not support time series with a seasonal component. The ARIMA model is extended (SARIMA) to support the seasonal component of the series. SARIMA (Seasonal Autoregressive Integrated Moving Average), method for time series forecasting is used on univariate data containing trends and seasonality. SARIMA is composed of trend and seasonal elements of the series.

Some of the parameters that are same as ARIMA model are: p: Trend autoregression order. d: Trend difference order. q: Trend moving average order There are four seasonal elements that are not part of ARIMA are: P: Seasonal autoregressive order. D: Seasonal difference order. Q: Seasonal moving average order. m: The number of time steps for a single seasonal period. Thus SARIMA model can be specified as: SARIMA (p, d, q) (P,D,Q) m

If m is 12, it specifies monthly data suggests a yearly seasonal cycle. SARIMA time series models can also be combined with spatial and event based models to yield ensemble models that solves multi-dimensional ML problems. Such a ML model can be designed to predict cell load in cellular networks at different times of the day round the year as illustrated below in the sample figure Autocorrelation, trend, and seasonality (weekday , weekend effects) from time series analysis can be used to interpret temporal influence. Regional and cell wise load distribution can be used to predict sparse and over loaded cells in varying intervals of time. Events (holidays, special mass gatherings and others) can be predicted using decision trees.

Reference :

https://towardsdatascience.com/arima-sarima-vs-lstm-with-ensemble-learning-insights-for-time-series-data-509a5d87f20a

```
sarima_mod6 = sm.tsa.statespace.SARIMAX(train_df.sales, trend='n', order=(6,1,0)).fit()
print(sarima_mod6.summary())
```

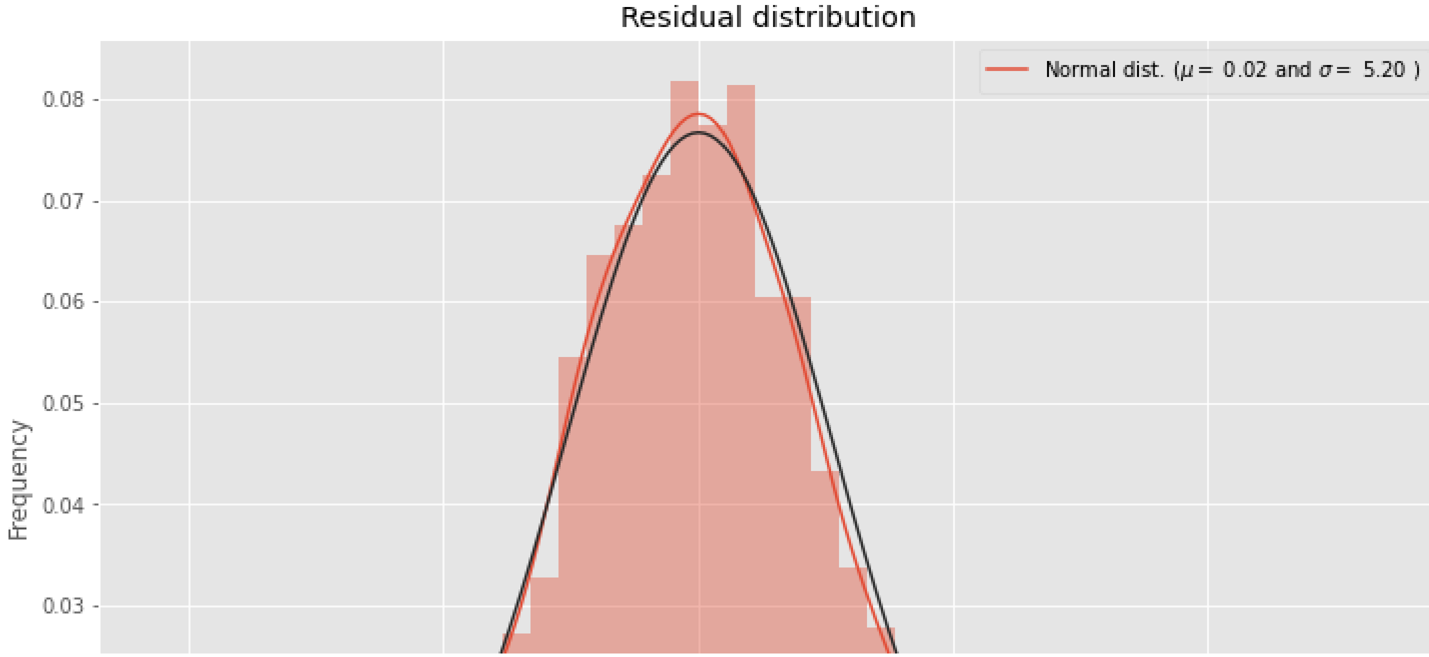
```
                           Statespace Model Results
==============================================================================
Dep. Variable:                  sales   No. Observations:                 1826
Model:               SARIMAX(6, 1, 0)   Log Likelihood               -5597.679
Date:                Tue, 25 May 2021   AIC                          11209.359
Time:                        12:39:36   BIC                          11247.924
Sample:                    01-01-2013   HQIC                         11223.585
                         - 12-31-2017
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
```

```
------------------------------------------------------------------------------
ar.L1         -0.8174      0.021    -39.063      0.000      -0.858      -0.776
ar.L2         -0.7497      0.025    -30.480      0.000      -0.798      -0.702
ar.L3         -0.6900      0.026    -26.686      0.000      -0.741      -0.639
ar.L4         -0.6138      0.027    -22.743      0.000      -0.667      -0.561
ar.L5         -0.5247      0.025    -21.199      0.000      -0.573      -0.476
ar.L6         -0.3892      0.021    -18.819      0.000      -0.430      -0.349
sigma2        26.9896      0.817     33.037      0.000      25.388      28.591
===================================================================================
Ljung-Box (Q):                      205.88   Jarque-Bera (JB):                19.53
Prob(Q):                              0.00   Prob(JB):                         0.00
Heteroskedasticity (H):               1.41   Skew:                             0.15
Prob(H) (two-sided):                  0.00   Kurtosis:                         3.40
===================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
```

```
resid = sarima_mod6.resid
print(normaltest(resid))

fig = plt.figure(figsize=(12,8))
ax0 = fig.add_subplot(111)

sns.distplot(resid ,fit = stats.norm, ax = ax0) # need to import scipy.stats

# Get the fitted parameters used by the function
(mu, sigma) = stats.norm.fit(resid)

#Now plot the distribution using
plt.legend(['Normal dist. ($\mu=$ {:.2f} and $\sigma=$ {:.2f} )'.format(mu, sigma)], loc='best')
plt.ylabel('Frequency')
plt.title('Residual distribution')


# ACF and PACF
fig = plt.figure(figsize=(12,8))
```

```
ax1 = fig.add_subplot(211)
fig = sm.graphics.tsa.plot_acf(arima_mod6.resid, lags=40, ax=ax1)
ax2 = fig.add_subplot(212)
fig = sm.graphics.tsa.plot_pacf(arima_mod6.resid, lags=40, ax=ax2)
```

```
NormaltestResult(statistic=16.742690143436878, pvalue=0.00023140408921805145)
```

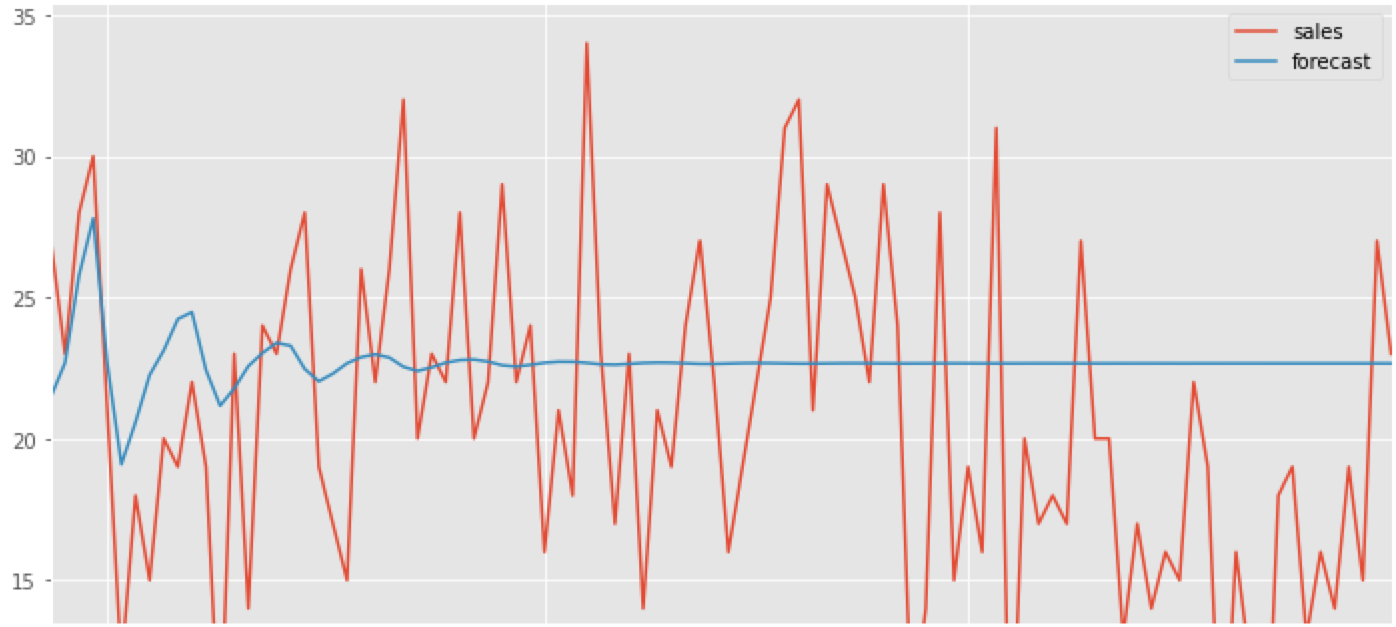


Make prediction and evaluation

Take the last 30 days in training set as validation data

```
start_index = 1730
end_index = 1826
train_df['forecast'] = sarima_mod6.predict(start = start_index, end= end_index, dynamic= True)
train_df[start_index:end_index][['sales', 'forecast']].plot(figsize=(12, 8))
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f6fac879610>
```



Evaluations of the model

```
def smape_kun(y_true, y_pred):
    mape = np.mean(abs((y_true-y_pred)/y_true))*100
    smape = np.mean((np.abs(y_pred - y_true) * 200/ (np.abs(y_pred) + np.abs(y_true))).fillna(0))
    print('MAPE: %.2f %% \nSMAPE: %.2f'% (mape,smape), "%")
```

```
smape_kun(train_df[1730:1825]['sales'],train_df[1730:1825]['forecast'])
```

```
MAPE: 33.01 %
SMAPE: 25.07 %
```

Conclusion

The study concludes with some case studies why specific machine learning methods perform so poorly in practice, given their impressive performance in other areas of artificial intelligence. The challenge leaves it open to evaluate reasons of poor performance for ARIMA/SARIMA and LSTM models, and devise mechanisms to improve model's poor performance and accuracy. Some of the areas of application of the models and their performance is listed below:

ARIMA yields better results in forecasting short term, whereas LSTM yields better results for long term modeling. Traditional time series forecasting methods (ARIMA) focus on univariate data with linear relationships and fixed and manually-diagnosed temporal dependence. Machine learning problems with substantial dataset, its found that the average reduction in error rates obtained by LSTM is between 84–87 percent when compared to ARIMA indicating the superiority of LSTM to ARIMA.

The number of training times, known as "epoch" in deep learning, has no effect on the performance of the trained forecast model and it exhibits a truly random behavior.

LSTMs when compared to simpler NNs like RNN and MLP appear to be more suited at fitting or overfitting the training dataset rather than forecasting it.

Neural networks (LSTMs and other deep learning methods) with huge datasets offer ways to divide it into several smaller batches and train the network in multiple stages. The batch size/each chunk size refers to the total number of training data used. The term iteration is used to represent number of batches needed to complete training a model using the entire dataset.

LSTM is undoubtedly more complicated and difficult to train and in most cases do not exceed the performance of a simple ARIMA

part B

SOURCING AND OUTSOURCING

```
import math
```

```
def distance(origin, destination):
    lat1, lon1 = origin
    lat2, lon2 = destination
    radius = 6371 # km

    dlat = math.radians(lat2-lat1)
    dlon = math.radians(lon2-lon1)
    a = math.sin(dlat/2) * math.sin(dlat/2) + math.cos(math.radians(lat1)) \
        * math.cos(math.radians(lat2)) * math.sin(dlon/2) * math.sin(dlon/2)
    c = 2 * math.atan2(math.sqrt(a), math.sqrt(1-a))
    d = radius * c

    return d

lat1 = 40.5; lat2 = 42; long1 = -90; long2 = -93
print( distance((lat1, long1), (lat2, long2)) )
```

```
301.17000641409464
```

```
!pip install calmap
```

```
Collecting calmap
  Downloading https://files.pythonhosted.org/packages/aa/2e/2fa4e527047261256b8e2d40bf9ed84e7e0c315ab904754c8ab2ce6f88¢
Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-packages (from calmap) (1.19.5)
Requirement already satisfied: pandas in /usr/local/lib/python3.7/dist-packages (from calmap) (1.1.5)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.7/dist-packages (from calmap) (3.2.2)
Requirement already satisfied: python-dateutil>=2.7.3 in /usr/local/lib/python3.7/dist-packages (from pandas->calmap) (
Requirement already satisfied: pytz>=2017.2 in /usr/local/lib/python3.7/dist-packages (from pandas->calmap) (2018.9)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.7/dist-packages (from matplotlib->calmap) (0.10.€
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in /usr/local/lib/python3.7/dist-packages (from
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.7/dist-packages (from matplotlib->calmap) (1
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.7/dist-packages (from python-dateutil>=2.7.3->pandas-
Installing collected packages: calmap
Successfully installed calmap-0.0.9
```

```
#Data Visualization libraries
import matplotlib.pyplot as plt
```

```
%matplotlib inline
import seaborn as sns
import plotly.express as px
import plotly.graph_objs as go
import plotly.figure_factory as ff
import folium
import calmap
from plotly.subplots import make_subplots
import plotly.io as pio
pio.templates.default = "plotly_dark"

#Some styling
sns.set_style("whitegrid")
plt.style.use("fivethirtyeight")

#displaying markdown
from IPython.display import Markdown
def bold(string):
    display(Markdown(string))

#Web scraping tools
#REQUESTS --> to fetch data from website
import requests
import json

#BEAUTIFULSOUP  -->parse HTML content
from bs4 import BeautifulSoup

#Showing full path of datasets
#import os
#for dirname, _, filenames in os.walk('/kaggle/input'):
#    for filename in filenames:
#        print(os.path.join(dirname, filename))

# Disable warnings
import warnings
warnings.filterwarnings('ignore')
```

```
import pandas  as  pd
import  numpy  as  np
import  seaborn  as  sns
import  matplotlib.pyplot  as  plt
%matplotlib inline
from  datetime  import  datetime
from  time  import  time
from  datetime  import  timedelta
import os
import itertools
```

```
#os.chdir("C://Users//rohan//Desktop//Supply-chain//Dataset")
```

```
df = pd.read_csv('/content/drive/My Drive/Dataset/available_products.csv')
df.head()
```

	Pune	Delhi
0	brakes	tyre
1	fuel_pump	brakes
2	tyre	v4

```
part = input("enter a product name ")
```

```
enter a product name brakes
```

```
def search(part):
    if part in df.values :
        return df.columns
    else :
        print("\nThis value does not exists in Dataframe")
```

```
a=search(part)
print(a)
```

```
Index(['Pune', 'Delhi'], dtype='object')
```

```
def search(part):
    lst=[]
    for (columnName, columnData) in df.iteritems():
        if part in columnData.values :
            lst.append(columnName)
    return(lst)
```

```
a=search(part)
print(a)
print(part)
```

```
['Pune', 'Delhi']
brakes
```

```
distance=pd.read_csv('/content/drive/My Drive/Dataset/district wise centroids.csv')
distance.head()
```

	State	District	Latitude	Longitude
0	Andaman and Nicobar	Andaman Islands	12.382571	92.822911
1	Andaman and Nicobar	Nicobar Islands	7.835291	93.511601
2	Andhra Pradesh	Adilabad	19.284514	78.813212
3	Andhra Pradesh	Anantapur	14.312066	77.460158
4	Andhra Pradesh	Chittoor	13.331093	78.927639

```
distance['Longitude']
```

```
0      92.822911
1      93.511601
2      78.813212
3      77.460158
4      78.927639
         ...
589    88.877940
590    86.396853
591    88.445370
592    88.235952
593    87.231014
Name: Longitude, Length: 594, dtype: float64
```

```
distance[distance["District"]=='Pune']
```

	State	District	Latitude	Longitude
328	Maharashtra	Pune	18.516962	74.129229

```
#distance.loc(distance[distance['District'] == location])
```

```
e=pd.DataFrame()
latitudes=[]
longitudes=[]
locations=[]
```

```
for i in range(len(a)):
    location=a[i]
    e=distance[distance['District'] == location]
    latitudes.append(list(e["Latitude"]))
    longitudes.append(list(e["Longitude"]))
    locations.append(list(e["District"]))
```

```
#c['Latitude']
```

```
longitudes
```

```
[[74.12922881632646], [77.1280451754386]]
```

```
latitudes
```

```
[[18.51696171428573], [28.64594429824561]]
```

```
latitude = latitudes

# output list
latitudes = []

# function used for removing nested
# lists in python.
def reemovNestings(latitude):
    for i in latitude:
        if type(i) == list:
            reemovNestings(i)
        else:
            latitudes.append(i)

# Driver code
print ('The original list: ', latitude)
reemovNestings(latitude)
print ('The list after removing nesting: ', latitudes)
```

```
The original list:  [[18.51696171428573], [28.64594429824561]]
The list after removing nesting:  [18.51696171428573, 28.64594429824561]
```

```
longitude = longitudes

# output list
```

```
longitudes = []

# function used for removing nested
# lists in python.
def reemovNestings(longitude):
    for i in longitude:
        if type(i) == list:
            reemovNestings(i)
        else:
            longitudes.append(i)

# Driver code
print ('The original list: ', longitude)
reemovNestings(longitude)
print ('The list after removing nesting: ', longitudes)
```

```
The original list:  [[74.12922881632646], [77.1280451754386]]
The list after removing nesting:  [74.12922881632646, 77.1280451754386]
```

```
locations
```

```
[['Pune'], ['Delhi']]
```

```
location = locations

# output list
locations = []

# function used for removing nested
# lists in python.
def reemovNestings(location):
    for i in location:
        if type(i) == list:
            reemovNestings(i)
        else:
            locations.append(i)

# Driver code
```

```
print ('The original list: ', location)
reemovNestings(location)
print ('The list after removing nesting: ', locations)
```

```
The original list:  [['Pune'], ['Delhi']]
The list after removing nesting:  ['Pune', 'Delhi']
```

```
longitudes
```

```
[74.12922881632646, 77.1280451754386]
```

```
latitudes
```

```
[18.51696171428573, 28.64594429824561]
```

```
def distance(origin, destination):
    lat1, lon1 = origin
    lat2, lon2 = destination
    radius = 6371 # km

    dlat = math.radians(lat2-lat1)
    dlon = math.radians(lon2-lon1)
    a = math.sin(dlat/2) * math.sin(dlat/2) + math.cos(math.radians(lat1)) \
        * math.cos(math.radians(lat2)) * math.sin(dlon/2) * math.sin(dlon/2)
    c = 2 * math.atan2(math.sqrt(a), math.sqrt(1-a))
    d = radius * c
    print(d)

    return d
```

```
sleepy=[]
j=0
lat1 = 27.1767; lat2 = 0; long1 = 78.0081; long2 = 0
for i in range(len(latitudes)):
        j=i
        lat2=float(latitudes[i])
        long2=float(longitudes[j])
        aa=distance((lat1, long1), (lat2, long2))
        sleepy.append(aa)


minpos = sleepy.index(min(sleepy))
print("closest_location_is",locations[minpos])
```

```
1041.502317603526
184.84472203899054
closest_location_is Delhi
```

```
#Creating Empty Map
radius = 0

m=folium.Map(location=[20.5937, 78.9629], zoom_start=14,max_zoom=4,min_zoom=3,tiles="Stamen Toner",
            height = 600,width = '70%')

for i in range(0,len(latitudes)):
    folium.Circle(location=[latitudes[i],longitudes[i]],
                 color="crimson",
                 radius=int(1000*50),
                  tooltip='<li><bold>District: '+str(locations[i])+
                  '<li><bold>Available part : '+str(part),
                  fill=True
                 ).add_to(m)

folium.Marker(location=[12.9716,77.5946],tooltip='<li><bold>WAREHOUSE LOCATION: '+str('BANGLORE'),icon=folium.Icon(color="re

#for i in range(0,len(latitudes)):
#    folium.Marker(location=[latitudes[i],longitudes[i]],
#                  icon=folium.Icon(color="red",icon="fa-hamburger", prefix='fa')).add_to(m)
```

```
m
```

```
m = folium.Map(location=[28,77], zoom_start=4)

# I can add marker one by one on the map
for i in range(0,len(latitudes)):
    folium.Circle(location=[latitudes[i],longitudes[i]],
                 color="crimson",
                  radius=int(1000*50),
                  tooltip='<li><bold>District: '+str(locations[i])+
                  '<li><bold>Available part : '+str(part),
                  fill=True, fill_color='crimson'
                 ).add_to(m)

folium.Marker(location=[12.9716,77.5946],
              tooltip='<li><bold>WAREHOUSE LOCATION: '+str('BANGLORE'),
              icon=folium.Icon(color="red",icon="fa-hamburger", prefix='fa')).add_to(m)

m
```


```
help(folium.Icon)
```

```
 |
 |  Methods defined here:
 |
 |  __init__(self, color='blue', icon_color='white', icon='info-sign', angle=0, prefix='glyphicon')
 |      Initialize self.  See help(type(self)) for accurate signature.
 |
 |  ----------------------------------------------------------------------
 |  Methods inherited from branca.element.MacroElement:
 |
 |  render(self, **kwargs)
 |      Renders the HTML representation of the element.
 |
 |  ----------------------------------------------------------------------
 |  Methods inherited from branca.element.Element:
 |
 |  add_child(self, child, name=None, index=None)
 |      Add a child.
 |
 |  add_children(self, child, name=None, index=None)
```

```
 |      Add a child.
 |  
 |  add_to(self, parent, name=None, index=None)
 |      Add element to a parent.
 |  
 |  get_bounds(self)
 |      Computes the bounds of the object and all it's children
 |      in the form [[lat_min, lon_min], [lat_max, lon_max]].
 |  
 |  get_name(self)
 |      Returns a string representation of the object.
 |      This string has to be unique and to be a python and
 |      javascript-compatible
 |      variable name.
 |  
 |  get_root(self)
 |      Returns the root of the elements tree.
 |  
 |  save(self, outfile, close_file=True, **kwargs)
 |      Saves an Element into a file.
 |      
 |      Parameters
 |      ----------
 |      outfile : str or file object
 |          The file (or filename) where you want to output the html.
 |      close_file : bool, default True
 |          Whether the file has to be closed after write.
 |  
 |  to_dict(self, depth=-1, ordered=True, **kwargs)
 |      Returns a dict representation of the object.
 |  
 |  to_json(self, depth=-1, **kwargs)
 |      Returns a JSON representation of the object.
 |  
 |  ----------------------------------------------------------------------
 |  Data descriptors inherited from branca.element.Element:
 |  
 |  __dict__
 |      dictionary for instance variables (if defined)
 |  
 |  __weakref__
```

Considering constant speed for now i.e 40km/hr

```
speed=40
for i in range(len(sleepy)):
    time_taken=sleepy[i]/speed
    print("time to deliver from",locations[i],"is",int(time_taken),"HOURS")
```

```
time to deliver from Pune is 26 HOURS
time to deliver from Delhi is 4 HOURS
```

Now lets predict the trip duration using Machine Learning Techniques

Trip Duration Prediction

The purpose of this modelling is to accurately predict the trip duration of taxi's. To make predictions we will use several algorithms, tune the corresponding parameters of the algorithm by analysisng each parameter against RMSE and predict the trip duration. To make our prediction we use RandomForest Regressor, LinearSVR and LinearRegression.

How does the pipeline look

1. Loading the data 2. Cleaning the data 3. Training the model 4. Making Predictions 5. Tuning the hyper Parameters to increase Confidence

```
import pandas as pd
import datetime as dt
import numpy as np
import matplotlib.pyplot as plt
from math import sqrt
from sklearn.metrics import mean_squared_error
from sklearn.ensemble import RandomForestRegressor
from sklearn import preprocessing, svm
from sklearn.svm import LinearSVR
```

```
from sklearn.linear_model import LinearRegression, SGDRegressor, Ridge
from sklearn.cluster import KMeans
from matplotlib import style
import pickle
style.use('ggplot')


from google.colab import drive
drive.mount('/content/drive/')
```

```
Drive already mounted at /content/drive/; to attempt to forcibly remount, call drive.mount("/content/drive/", force_rem
```

```
import os
#os.chdir("C://Users//rohan//Desktop//Supply-chain//Dataset")
#path = "random_2015_cleaned.csv"
#'/content/drive/My Drive/Tranferlearning-indian currency/dataset/xxxfile'
df = pd.read_csv('/content/drive/My Drive/Dataset/random_2015_cleaned.csv')
df.dropna(inplace=True)
df.head(10)
```

Unnamed: 0	tpep_pickup_datetime	tpep_dropoff_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropof

```
#Getting attributes for EDA
df = df[['tpep_pickup_datetime', 'tpep_dropoff_datetime', 'pickup_longitude', 'pickup_latitude', 'dropoff_longitude', 'dropo
df['tpep_pickup_datetime'] = pd.to_datetime(df['tpep_pickup_datetime'])
df['tpep_dropoff_datetime'] = pd.to_datetime(df['tpep_dropoff_datetime'])
df['pickup_hrs'] = df['tpep_pickup_datetime'].dt.hour
df['dropoff_hrs'] = df['tpep_dropoff_datetime'].dt.hour
df['day_week'] = df['tpep_pickup_datetime'].dt.weekday
df['tpep_pickup_timestamp'] = (df['tpep_pickup_datetime'] - dt.datetime(1970, 1, 1)).dt.total_seconds()
df['tpep_dropoff_timestamp'] = (df['tpep_dropoff_datetime'] - dt.datetime(1970, 1, 1)).dt.total_seconds()
df['duration'] = df['tpep_dropoff_timestamp'] - df['tpep_pickup_timestamp']
df['speed'] = (df['trip_distance'] * 3600)//df['duration']
```

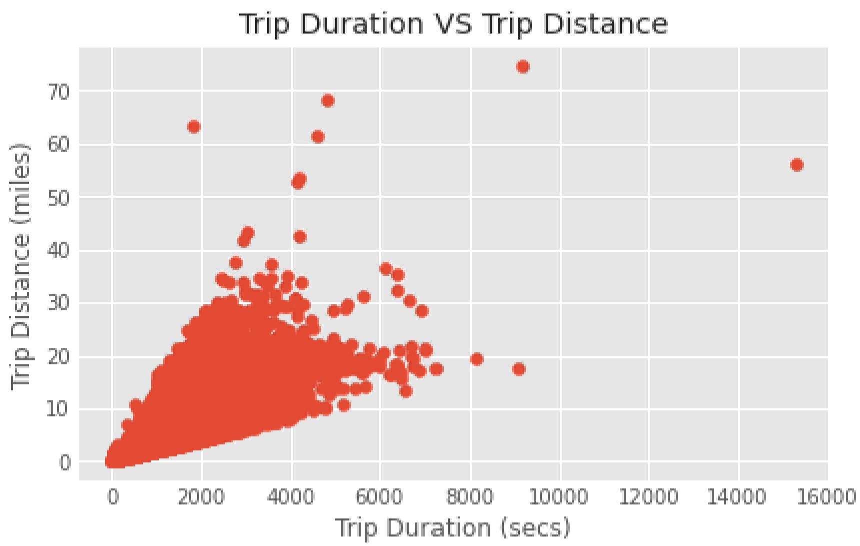
7	7	2015-01-08 18:18:30	2015-01-08 18:23:16	-74.004387	40.747833	-73.999710

```
#cleaning for EDA, removing outliers
df = df[ (df['duration'] > 0)]
df = df[ (df['speed'] > 6.0)]
df = df[ (df['speed'] < 140.0)]
df = df[ (df['pickup_longitude'] != 0)]
df = df[ (df['dropoff_longitude'] != 0)]
df = df[ (df['pickup_latitude'] > 38)]
df = df[ (df['pickup_latitude'] < 45)]
```

```
df.head()
```

tpep_pickup_datetime	tpep_dropoff_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude

```
#EDA
plt.scatter(df['duration'], df['trip_distance'])
plt.title('Trip Duration VS Trip Distance')
plt.xlabel('Trip Duration (secs)')
plt.ylabel('Trip Distance (miles)')
plt.show()
```




```
#clustering pickup and dropoff locations
n = len(df)
kmeans_pickup = KMeans(n_clusters = 15, random_state = 2).fit(df[['pickup_latitude', 'pickup_longitude']])
df['kmeans_pickup'] = kmeans_pickup.predict(df[['pickup_latitude','pickup_longitude']])
plt.scatter(df.pickup_longitude[:n],
            df.pickup_latitude[:n],
            cmap = 'viridis',
            c = df.kmeans_pickup[:n])
plt.title('Pickup Location Clustering')
plt.xlabel('Longitude')
plt.ylabel('Latitude')
plt.show()

kmeans_dropoff = KMeans(n_clusters = 15, random_state = 2).fit(df[['dropoff_latitude', 'dropoff_longitude']])
```

```
kmeans_dropoff = KMeans(n_clusters = 15, random_state = 2).fit(df[['dropoff_latitude','dropoff_longitude']])
df['kmeans_dropoff'] = kmeans_dropoff.predict(df[['dropoff_latitude','dropoff_longitude']])
plt.scatter(df.dropoff_longitude[:n],
            df.dropoff_latitude[:n],
            cmap = 'viridis',
            c = df.kmeans_dropoff[:n])
plt.title('Dropoff Location Clustering')
plt.xlabel('Longitude')
plt.ylabel('Latitude')
plt.show()
```

Pickup Location Clustering

```
#creating dummy variables/one hot encoding, adding features
df = pd.concat([df, pd.get_dummies(df['pickup_hrs'], prefix = 'hrs')], axis = 1)
df = pd.concat([df, pd.get_dummies(df['day_week'], prefix = 'day')], axis = 1)
df['pickup_dropoff_cluster'] = df['kmeans_pickup'].map(str) + 'to' +  df['kmeans_dropoff'].map(str)
df = pd.concat([df, pd.get_dummies(df['pickup_dropoff_cluster'], prefix = 'route')], axis = 1)
```

```
##cleaninig df for training containig only features
df.drop(df.columns[[0, 1, 2, 3, 4, 5, 7, 8, 9, 10, 11, 13, 14, 15, 47]], axis = 1, inplace = True)
```

```
#writing cleaned data to file for post prediction analysis and tuning hyperparameyres for Randomforest
df.to_csv('post_analysis_data.csv')
```

Dropoff Location Clustering

MODELS

```
from sklearn.model_selection import train_test_split
X = np.array(df.drop(['duration'], 1))
y = np.array(df['duration'])
```

```
#from sklearn.preprocessing import MinMaxScaler
#Scaler = MinMaxScaler(feature_range = (0,1))
#X = Scaler.fit_transform(X)
#X = pd.DataFrame(X)
#print(X.head(5))
#from sklearn.preprocessing import StandardScaler
#sc = StandardScaler()
#X= sc.fit_transform(X)
#X = sc.transform(X)
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2)
```

```
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_squared_log_error
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
from sklearn import *
```

a) Linear Regression

```
from sklearn.linear_model import LinearRegression
lr = LinearRegression()
lr.fit(X_train, y_train)
y_pred_lr = lr.predict(X_test)
```

```
accuracy = lr.score(X_test, y_test)
accuracy
```

```
0.7805781859765831
```

```
predictions = lr.predict(X_test)
print("Mean Absolute Error : " + str(mean_absolute_error(predictions,y_test)))
```

```
Mean Absolute Error : 170.7867109199458
```

```
print("Root Mean Squared Error is ", mean_squared_error(y_test,predictions)**(0.5))
```

```
Root Mean Squared Error is  259.4134724830809
```

```
y_pred = lr.predict(X_test)
print("r2score is",sklearn.metrics.r2_score(y_test,y_pred))
```

```
print("rscore is",sklearn.metrics.r2_score(y_test,y_pred))
```

```
rscore is 0.7805781859765831
```

b) Support vector regression (SVRs)

```
from sklearn import svm
svm = svm.SVR()
svm.fit(X_train, y_train)
y_pred = svm.predict(X_test)
```

```
accuracy = svm.score(X_test, y_test)
accuracy
```

```
predictions = svm.predict(X_test)
print("Mean Absolute Error : " + str(mean_absolute_error(predictions,y_test)))
```

```
print("Root Mean Squared Error is ", mean_squared_error(y_test,predictions)**(0.5))
```

```
y_pred = svm.predict(X_test)
print("rscore is",sklearn.metrics.r2_score(y_test,y_pred))
```

c) Bayesian regression

```
from sklearn.linear_model import BayesianRidge
# Creating and training model
modelb = BayesianRidge()
modelb.fit(X_train, y_train)
```

```
BayesianRidge(alpha_1=1e-06, alpha_2=1e-06, alpha_init=None,
              compute_score=False, copy_X=True, fit_intercept=True,
              lambda_1=1e-06, lambda_2=1e-06, lambda_init=None, n_iter=300,
              normalize=False, tol=0.001, verbose=False)
```

```
accuracy = modelb.score(X_test, y_test)
accuracy
```

```
0.780553392432325
```

```
predictions = modelb.predict(X_test)
print("Mean Absolute Error : " + str(mean_absolute_error(predictions,y_test)))
```

```
Mean Absolute Error : 170.94139274017988
```

```
print("Root Mean Squared Error is ", mean_squared_error(y_test,predictions)**(0.5))
```

```
Root Mean Squared Error is  259.42812826778606
```

```
y_pred = modelb.predict(X_test)
print("rscore is",sklearn.metrics.r2_score(y_test,y_pred))
```

```
rscore is 0.780553392432325
```

d) Decision Tree Regression

```
#from sklearn import tree
#d3 = tree.DecisionTreeClassifier()
##d3.fit(X_train,y_train)
#d3_pred=d3.predict(X_test)
```

```
from sklearn.tree import DecisionTreeRegressor
d3= DecisionTreeRegressor()
d3.fit(X_train, y_train)
```

```
DecisionTreeRegressor(ccp_alpha=0.0, criterion='mse', max_depth=None,
                      max_features=None, max_leaf_nodes=None,
                      min_impurity_decrease=0.0, min_impurity_split=None,
                      min_samples_leaf=1, min_samples_split=2,
                      min_weight_fraction_leaf=0.0, presort='deprecated',
                      random_state=None, splitter='best')
```

```
accuracy = d3.score(X_test, y_test)
accuracy
```

```
0.7072930124263475
```

```
predictions = d3.predict(X_test)
print("Mean Absolute Error : " + str(mean_absolute_error(predictions,y_test)))
```

```
Mean Absolute Error : 179.71852584942945
```

```
print("Root Mean Squared Error is ", mean_squared_error(y_test,predictions)**(0.5))
```

```
Root Mean Squared Error is  299.6188768169315
```

```
y_pred = d3.predict(X_test)
print("rscore is",sklearn.metrics.r2_score(y_test,y_pred))
```

```
rscore is 0.7072930124263475
```

e) KNN regression

```
from sklearn.neighbors import KNeighborsRegressor
neigh = KNeighborsRegressor(n_neighbors=2)
neigh.fit(X_train, y_train)
knn_pred=neigh.predict(X_test)
```

```
accuracy = neigh.score(X_test, y_test)
accuracy
```

```
0.7509285383054066
```

```
predictions = neigh.predict(X_test)
print("Mean Absolute Error : " + str(mean_absolute_error(predictions,y_test)))
```

```
Mean Absolute Error : 168.74404283801874
```

```
print("Root Mean Squared Error is ", mean_squared_error(y_test,predictions)**(0.5))
```

```
Root Mean Squared Error is  276.385090392524
```

```
y_pred = neigh.predict(X_test)
print("rscore is",sklearn.metrics.r2_score(y_test,y_pred))
```

```
rscore is 0.7509285383054066
```

F) Random forest Regressor

```
## Training the model
```

```
clf = RandomForestRegressor(n_estimators = 50, n_jobs = -1)
clf.fit(X_train, y_train)
```

```
RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                      max_depth=None, max_features='auto', max_leaf_nodes=None,
                      max_samples=None, min_impurity_decrease=0.0,
                      min_impurity_split=None, min_samples_leaf=1,
                      min_samples_split=2, min_weight_fraction_leaf=0.0,
                      n_estimators=50, n_jobs=-1, oob_score=False,
                      random_state=None, verbose=0, warm_start=False)
```

```
##Making Predictions
```

```
accuracy = clf.score(X_test, y_test)
```

```
accuracy
```

```
0.8264000477212416
```

```
predictions = clf.predict(X_test)
print("Mean Absolute Error : " + str(mean_absolute_error(predictions,y_test)))
```

```
Mean Absolute Error : 142.01376535875664
```

```
print("Root Mean Squared Error is ", mean_squared_error(y_test,predictions)**(0.5))
```

```
Root Mean Squared Error is  230.74241157859484
```

```
y_pred = clf.predict(X_test)
print("rscore is",sklearn.metrics.r2_score(y_test,y_pred))
```

```
rscore is 0.8264000477212416
```

```
#Tuning/Analysing the hyperparameters to improve confidence
```

```
#Analysing the required number of trees for RandomForest
a = np.array([[10, 247]])
for i in range(20, 60, 10):
    clf = RandomForestRegressor(n_estimators = i)
    clf.fit(X_train, y_train)
    y_actual = y_test
    y_pred = clf.predict(X_test)
    rms = sqrt(mean_squared_error(y_actual, y_pred))
    a = np.append(a, [[i, rms]], axis = 0)
```

```
plt.plot(a[:, 0], a[:, 1], linewidth = 2.0)
plt.title('RMSE VS No. of Trees')
plt.xlabel('No. of Trees')
plt.ylabel('RMSE')
plt.show()
```

0s completed at 7:17 PM