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

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.r
```

import Library

```
import numpy as np
import pandas as pd
from sklearn import preprocessing, linear_model, metrics
import matplotlib.pyplot as plt
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
```

Readind Data and Processing Data

```
dtypes = {'id':'int64', 'item_nbr':'int32', 'store_nbr':'int8', 'onpromotion':str}
data = {
    'tra': pd.read_csv('/content/drive/My Drive/extracted/train.csv', dtype=dtypes, parse_
    'tes': pd.read_csv('/content/drive/My Drive/extracted/test.csv', dtype=dtypes, parse_d
    'ite': pd.read_csv('/content/drive/My Drive/extracted/items.csv'),
    'sto': pd.read_csv('/content/drive/My Drive/extracted/stores.csv'),
    'trn': pd.read_csv('/content/drive/My Drive/extracted/transactions.csv', parse_dates=[
    'hol': pd.read_csv('/content/drive/My Drive/extracted/holidays_events.csv', dtype={'tr
    'oil': pd.read_csv('/content/drive/My Drive/extracted/oil.csv', parse_dates=['date']),
    }
```

```
train = data['tra'][(data['tra']['date'].dt.month == 8) & (data['tra']['date'].dt.day > 15
test = data['tes'][(data['tes']['date'].dt.month == 8) & (data['tes']['date'].dt.day > 15)
strain = train.sample(frac =.5)
stest = test.sample(frac =.5)
print(strain.shape,stest.shape)
```

```
(2229506, 6) (1685232, 5)
```

```
target = strain['unit_sales'].values
target[target < 0.] = 0.
strain['unit_sales'] = target
```

```
strain = pd.merge(strain, data['ite'], how='left', on=['item_nbr'])
strain = pd.merge(strain, data['sto'], how='left', on=['store_nbr'])
data_h_1 = data['hol'][data['hol']['locale'] == 'National'][['date','transferred']]
data_h_1['transferred'] = data_h_1['transferred'].map({'False': 0, 'True': 1})
strain = pd.merge(strain, data_h_1, how='left', on=['date'])
strain = pd.merge(strain, data['oil'], how='left', on=['date'])
```

```
stest = pd.merge(stest, data['ite'], how='left', on=['item_nbr'])
stest = pd.merge(stest, data['sto'], how='left', on=['store_nbr'])
data_h_t = data['hol'][data['hol']['locale'] == 'National'][['date','transferred']]
data_h_t['transferred'] = data_h_t['transferred'].map({'False': 0, 'True': 1})
stest = pd.merge(stest, data_h_t, how='left', on=['date'])
stest = pd.merge(stest, data['oil'], how='left', on=['date'])
```

```
from sklearn import preprocessing
def df_transform(df):
    df['date'] = pd.to_datetime(df['date'])
    df['yea'] = df['date'].dt.year
    df['mon'] = df['date'].dt.month
    df['day'] = df['date'].dt.day
    df['dayofweek'] = df['date'].dt.dayofweek
    df['onpromotion'] = df['onpromotion'].map({'False': 1, 'True': 2})
    df['perishable'] = df['perishable'].map({0:1.0, 1:1.25})
    df = df.fillna(0)
    return df
def df_lbl_enc(df):
    for c in df.columns:
        if df[c].dtype == 'object':
            lbl = preprocessing.LabelEncoder()
            df[c] = lbl.fit_transform(df[c])
            print(c)
    return df
```

```
strain_t = df_transform(strain)
strain_t_e = df_lbl_enc(strain_t)

stest_t = df_transform(stest)
stest_t_e = df_lbl_enc(stest_t)
```

```
family
city
state
type
family
city
state
type
```

```
strain_t_e_dateIndex = strain_t_e.set_index('date')
stest_t_e_dateIndex = stest_t_e.set_index('date')
```

```
col =[c for c in strain_t_e_dateIndex if c not in ['id','item_nbr','mon','class','city','c
print(col)
train_features = strain_t_e_dateIndex[col]
target = np.log1p(strain_t_e_dateIndex[['unit_sales']])

col =[c for c in stest_t_e_dateIndex if c not in ['id','item_nbr','mon','class','city','cl
features = stest_t_e_dateIndex[col]
```

```
['store_nbr', 'onpromotion', 'family', 'perishable', 'state', 'type', 'transferred',
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(train_features, target, test_size=0.20
```

```
W_train = X_train['perishable']#.map({0:1.0, 1:1.25})
W_test = X_test['perishable']
```

Random Forest Regression

```
rf = RandomForestRegressor(max_features = "auto", random_state =50 )
```

```
rf.fit(X_train, y_train)
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:1: DataConversionWarning
  """Entry point for launching an IPython kernel.
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=50, verbose=0, warm_start=False)
```

```
print ('RF accuracy: TRAINING', rf.score(X_train,y_train,W_train))
print ('RF accuracy: TESTING', rf.score(X_test,y_test,W_test))
print("feature Importance",rf.feature_importances_)
```

```
from statsmodels.tsa.seasonal import seasonal_decompose
from plotly.offline import download_plotlyjs, init_notebook_mode, plot, iplot
import plotly.graph_objs as go
result = seasonal_decompose(np.log1p(strain_t_e.unit_sales.values),freq=12)
trace1=go.Scatter(x=pd.to_datetime(strain_t_e.date),y=result.trend,name="trend")
trace2 = go.Scatter(
    x = pd.to_datetime(strain_t_e.date),y = result.seasonal,
    name = 'Seasonal'
)
dat=[trace1,trace2]
plot(dat)
plt.plot(result.trend)
plt.plot(result.seasonal)
```

```
result = seasonal_decompose(np.log1p(strain_t_e.unit_sales.values),freq=6)
trace1=go.Scatter(x=pd.to_datetime(strain_t_e.date),y=result.trend,name="trend")
trace2 = go.Scatter(
    x = pd.to_datetime(strain_t_e.date),y = result.seasonal,
```

```
    name = 'Seasonal'
)
dat=[trace1,trace2]
plot(dat)
plt.plot(result.trend)
plt.plot(result.seasonal)
```

Polynomial regression

```
from sklearn.preprocessing import PolynomialFeatures
polynomial_features= PolynomialFeatures(degree=3)
x_train_poly = polynomial_features.fit_transform(X_train)
x_test_poly = polynomial_features.fit_transform(X_test)
```

```
from sklearn.linear_model import LinearRegression
model = LinearRegression()
model.fit(x_train_poly, y_train)
y_test_pred=model.predict(x_test_poly)
y_train_pred=model.predict(x_train_poly)
```

```
from sklearn.metrics import mean_squared_error
from math import sqrt
from sklearn.metrics import r2_score
print("rmse value of linear regression using sklearn = ",np.sqrt(np.mean((y_test-y_test_pr
sse=np.sum((y_test-y_test_pred)**2)
print("sum of squared error value =",sse)
r2=r2_score(y_test,y_test_pred)
print("r2_score",r2)
```

Displaying trend and Seasonality

```
from statsmodels.tsa.seasonal import seasonal_decompose
series = strain_t_e.unit_sales.values
result = seasonal_decompose(series, model='additive',freq=12,two_sided = False)
plt.plot(result.trend)
```

```
plt.plot(result.seasonal)
```

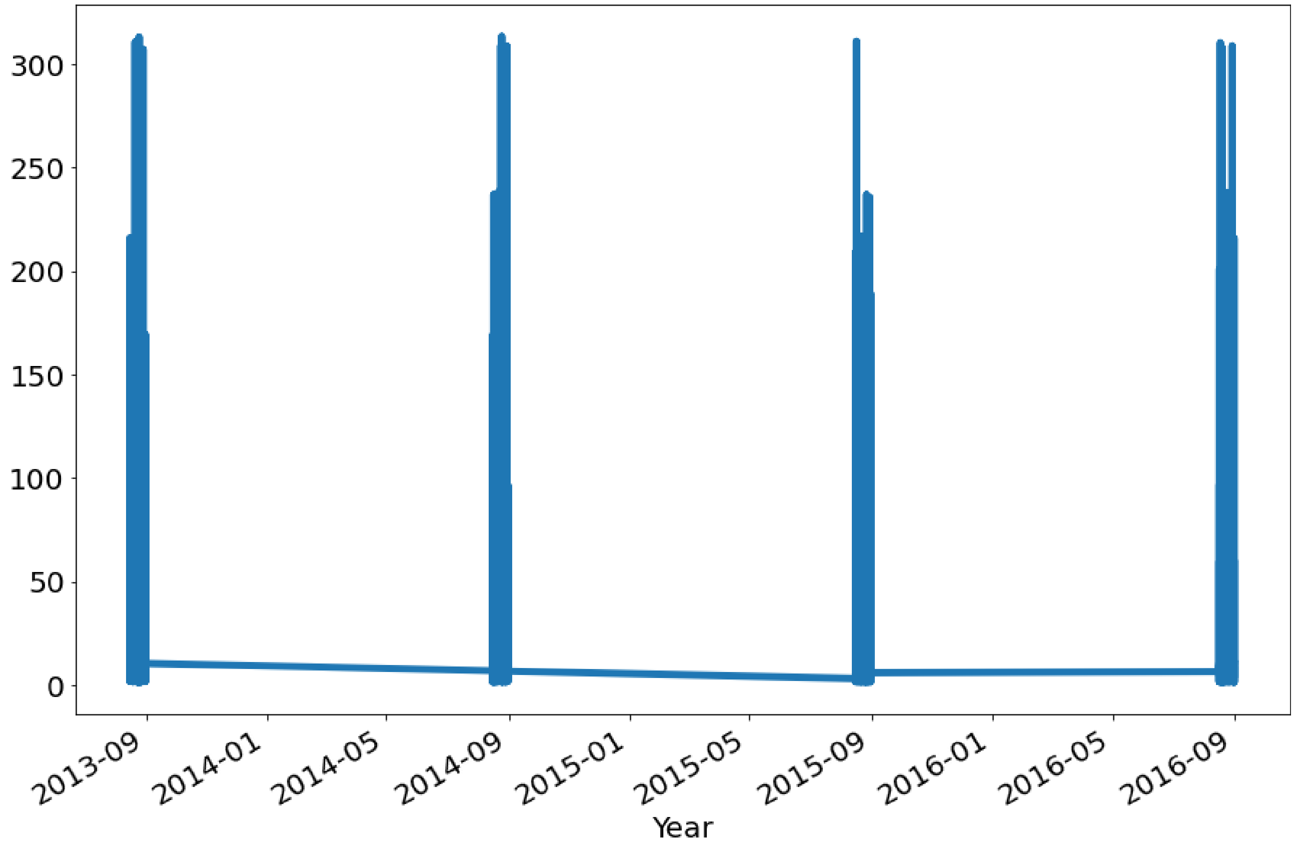
```
strain_t_e_dateIndex['unit_sales'].plot(figsize=(20,10), linewidth=5, fontsize=20)
plt.xlabel('Date', fontsize=20);
```

Printing Trend Using Rolling Average

```
from statsmodels.tsa.stattools import adfuller
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.tsa.arima_model import ARIMA
from pandas.plotting import register_matplotlib_converters
register_matplotlib_converters()
```

```
/usr/local/lib/python3.6/dist-packages/statsmodels/tools/_testing.py:19: FutureWarnir
  import pandas.util.testing as tm
```

```
strain_t_e_dateIndex['unit_sales'].rolling(12).mean().plot(x=strain_t_e_dateIndex['yea'],f
plt.xlabel('Year', fontsize=20);
```




```
rolling_mean = strain_t_e_dateIndex.rolling(window = 12).mean()
rolling_std = strain_t_e_dateIndex.rolling(window = 12).std()
```

```
rolling_mean.shape()
```

```
plt.rcParams['agg.path.chunksize'] = 10000000000000000000000
plt.plot(strain_t_e_dateIndex, color = 'blue', label = 'Original')
plt.plot(rolling_mean, color = 'red', label = 'Rolling Mean')
plt.plot(rolling_std, color = 'black', label = 'Rolling Std')
plt.legend(loc = 'best')
plt.title('Rolling Mean & Rolling Standard Deviation')
plt.show()
```

```
result = adfuller(strain_t_e_dateIndex['unit_sales'])
print('ADF Statistic: {}'.format(result[0]))
print('p-value: {}'.format(result[1]))
print('Critical Values:')
for key, value in result[4].items():
    print('\t{}: {}'.format(key, value))
```

Auto Regressive model

```
from statsmodels.tsa.ar_model import AR
model = AR(X_train)
model_fitted = model.fit()
```

```
print('The lag value chose is: %s' % model_fitted.k_ar)
print('The coefficients of the model are:\n %s' % model_fitted.params)
```

```
predictions = model_fit.predict(start=len(train), end=len(train)+len(test)-1, dynamic=Fals
```

Navie bias

```
stest_t_e_dateIndex.head()
```

	id	store_nbr	item_nbr	onpromotion	family	class	perishable	city
date								

```
dd= np.asarray(strain_t_e_dateIndex.unit_sales)
y_hat =stest_t_e_dateIndex.copy()
y_hat['naive'] = dd[len(dd)-1]
plt.figure(figsize=(12,8))
plt.plot(strain_t_e_dateIndex.index, strain_t_e_dateIndex['unit_sales'], label='Train')
plt.plot(stest_t_e_dateIndex.index,stest_t_e_dateIndex['unit_sales'], label='Test')
plt.plot(y_hat.index,y_hat['naive'], label='Naive Forecast')
plt.legend(loc='best')
plt.title("Naive Forecast")
plt.show()
```

2017-

ARIMA model

```
df_log = np.log(strain_t_e_dateIndex)
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:1: RuntimeWarning: divi
  """Entry point for launching an IPython kernel.
```

```
decomposition = seasonal_decompose(df_log)
model = ARIMA(df_log, order=(2,1,2))
results = model.fit(disp=-1)
plt.plot(results.fittedvalues, color='red')
```

2013-08-27	16.145275	1.098612	13.038556	0.693147	-inf	2.397895	6.990257
2013-08-18	16.108216	3.850148	12.904640	2.708050	-inf	2.397895	6.999422
2014-08-27	17.207557	2.079442	13.277528	2.457193	0.0	3.135494	7.741534
2013-08-30	16.159747	3.178054	12.788933	3.784190	-inf	1.791759	8.018955
...	...	...	...	...	...	...	...
2016-08-24	18.303537	3.258097	13.034467	1.694330	0.0	3.295837	7.789869
2013-08-16	16.097206	3.465736	13.034467	1.837848	-inf	3.295837	7.789869
2014-08-22	17.199123	3.496508	13.055982	2.197225	0.0	2.397895	6.981006
2014-08-17	17.189522	2.397895	12.661343	0.693147	0.0	2.397895	6.971669