Introduction

Welcome to the "Wal Transparency, Accuracy" competition! In this competition, contestants are challenged to forecast future sales at Walmart. Accurate forecasting is important for Walmart, as it helps the company to manage its inventory, plan its staffing, and make other critical operational decisions. Accurate forecasting is important for businesses, and it is important because it allows companies to make guided decisions based on data and to avoid costly mistakes.

In this kernel, I primarily explore the structure of whatever There, I will visualize the dataset using Matplotlib and Plotly. And finally, I will demonstrate how the problem can be approached with a series of forecasting approaches.

Please upvote this kernel if you like it. It motivates me to produce more quality content :)

To get started, take a look at numerous video clips of Walmart that are available on YouTube.

```
from IPython.display import YouTube
YouTube('https://www.youtube.com/watch?v=XOJ1Ni_Ydfo&ab_channel=Walmart%27sCommunityGardening')
```

Acknowledgements

- EDA Examples - Grocery sales forecasting - by Rob Mulla
- Time Series Forecasting - by Jason Brownlee
- How to Make an ARIMA Model for Time Series Forecasting in Python - by Marco Cerliani
- Multivariate Time Series Data Modeling and Forecasting with Prophet - by Anderson Santiago
- Time Series Forecasting with Prophet - by Vitor Cerqueira
- Prophet - by Facebook

Contents

- Time Series
- EDA
  - Importing the dataset
  - Data Cleaning
  - Statistics
  - Exploratory analysis
- Modeling
  - Train/test split
  - Moving average
  - Moving average smoothing
  - Exponential smoothing
  - ARIMA
  - Prophet
  - Facebook Prophet model
- Testing
- Conclusion
- Exploring Stores

The dataset

The dataset consists of five .csv files.

- `calendar.csv` - Contains information about the dates on which products are sold. The dates are in `d_[number]` format.
- `sales_train_validation.csv` - Contains the historical daily unit sales data per product and store `d_1` - `d_1913`
- `sample_submission.csv` - Demonstrates the correct format for submission to the competition.
- `sell_prices.csv` - Contains information about the sell prices for each product on a given store and date.
- `sell_prices_validation.csv` - Available once month before the competition deadline. It will include sales for `d_1914` - `d_1941`

In this competition, we need to forecast the sales for `d_1942` - `d_1941`. These sales form the evaluation set. The data `d_1914` - `d_1941` form the validation set, and the remaining data form the training set. Now, once we understand the dataset and know what to do, let's move on to examine the dataset.

EDA

Now, I will try to visualize the sales data and gain some insights from it.

Preparing the ground

Import libraries

```
import os
import gc
import sys
import psutil
import datetime
import numpy as np
from ipykernel.zmqstream import IOStream
import pandas as pd
import matplotlib as mpl
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.offline as py
import plotly.graph_objs as go
import plotly.tools as tls
import warnings
from scipy import stats
from statsmodels import tsa
from statsmobile import *
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.tsa.stattools import adfuller, acf, pacf
from statsmodels import tsa.arima_model
from statsmodels.tsa.arima_model.ARIMA import ARIMA
from statsmodels.tsa.arima_model.ARMA import ARMA
from statsmodels.tsa.arima_model.ARIMAResults import ARIMAResults
from statsmodels.tsa.arima_model.ARMAResults import ARMAResults
from statsmodels.tsa.stattools import pacf_yw
from statsmodels.tsa.stattools import pacf_ols
from statsmodels import sandbox
from sklearn.preprocessing import LabelEncoder, MinMaxScaler
from sklearn.metrics import mean_squared_error
from sklearn import preprocessing, metrics
from fbprophet import Prophet
from fbprophet import diagnostics
from fbprophet.plot import plot_cross_validation_metric
from IPython.display import display, HTML
warnings = warnings.filterwarnings('ignore')
%matplotlib inline
%pylab inline
%matplotlib notebook
%load_ext rpy2.ipython
%load_ext google.colaboratory.extensions
%load_ext google.colaboratory.auto_eda
%load_ext google.colaboratory.auto_plot
%load_ext google.colaboratory.auto_ml
%load_ext google.colaboratory.auto_viz
%load_ext google.colaboratory.auto_nlp
%load_ext google.colaboratory.auto_cv
%load_ext google.colaboratory.auto_ai
%load_ext google.colaboratory.auto_gcp
%load_ext google.colaboratory.auto_azure
%load_ext google.colaboratory.auto_aws
%load_ext google.colaboratory.auto_oracle
%load_ext google.colaboratory.auto_ibm
%load_ext google.colaboratory.auto_sap
%load_ext google.colaboratory.auto_salesforce
%load_ext google.colaboratory.auto_twitter
%load_ext google.colaboratory.auto_facebook
%load_ext google.colaboratory.auto_google
%load_ext google.colaboratory.auto_microsoft
%load_ext google.colaboratory.auto_amazon
%load_ext google.colaboratory.auto_apple
%load_ext google.colaboratory.auto_samsung
%load_ext google.colaboratory.auto_huawei
%load_ext google.colaboratory.auto_xiaomi
%load_ext google.colaboratory.auto_oppo
%load_ext google.colaboratory.auto_vivo
%load_ext google.colaboratory.auto_realme
%load_ext google.colaboratory.auto_oneplus
%load_ext google.colaboratory.auto_nokia
%load_ext google.colaboratory.auto_sony
%load_ext google.colaboratory.auto_lg
%load_ext google.colaboratory.auto_htc
%load_ext google.colaboratory.auto_motorola
%load_ext google.colaboratory.auto_lenovo
%load_ext google.colaboratory.auto_asus
%load_ext google.colaboratory.auto_acer
%load_ext google.colaboratory.auto_dell
%load_ext google.colaboratory.auto_hp
%load_ext google.colaboratory.auto_ibm
%load_ext google.colaboratory.auto_intel
%load_ext google.colaboratory.auto_amd
%load_ext google.colaboratory.auto_nvidia
%load_ext google.colaboratory.auto_qualcomm
%load_ext google.colaboratory.auto_mediatek
%load_ext google.colaboratory.auto_broadcom
%load_ext google.colaboratory.auto_marvell
%load_ext google.colaboratory.auto_texas_instruments
%load_ext google.colaboratory.auto_analog_devices
%load_ext google.colaboratory.auto_maxim_integrated
%load_ext google.colaboratory.auto_cypress_semiconductor
%load_ext google.colaboratory.auto_silicon_labs
%load_ext google.colaboratory.auto_microchip
%load_ext google.colaboratory.auto_renesas
%load_ext google.colaboratory.auto_rohm
%load_ext google.colaboratory.auto_semtech
%load_ext google.colaboratory.auto_ti
%load_ext google.colaboratory.auto_st
%load_ext google.colaboratory.auto_nxp
%load_ext google.colaboratory.auto_infineon
%load_ext google.colaboratory.auto_microsemi
%load_ext google.colaboratory.auto_vishay
%load_ext google.colaboratory.auto_ixys
%load_ext google.colaboratory.auto_littelfuse
%load_ext google.colaboratory.auto_ixys
%load_ext google.colaboratory.auto_ixys
%load_ext google.colaboratory.auto_ixys
```

```
[In]: sales_pivot = sales.sort_values(sort_sales_column)[['Id',
                                                          'Store name',
                                                          'Sales Value',
                                                          'Sales Time',
                                                          'Year',
                                                          'Month',
                                                          'Day',
                                                          'Hour',
                                                          'Minute',
                                                          'Second']].pivot(index='Id',
                                                                           columns='Sales Time')
      sales_pivot.reset_index(inplace=True)
      sales_pivot.dropna(inplace=True)
      sales_pivot.drop(['Year', 'Month', 'Day', 'Hour', 'Minute', 'Second'],
                       axis=1, inplace=True)
      sales_pivot.columns = ['Id', 'Store name', 'Sales Value', 'Sales Time']
      sales_pivot.sort_values('Sales Time', inplace=True)
      sales_pivot.reset_index(drop=True)
      sales_pivot.head()
```

```
[Out]:
```

	Id	Store name	Sales Value	Sales Time
0	1	Store Two	1	2015-01-01 00:00:00
1	2	Store Two	1	2015-01-01 00:00:00
2	3	Store Two	1	2015-01-01 00:00:00
3	4	Store Two	1	2015-01-01 00:00:00
4	5	Store Two	1	2015-01-01 00:00:00

```
[In]: fig, ax = plt.figure()
      for i, j in enumerate(stores):
          sns.lineplot(data=sales_pivot[sales_pivot['Store name'] == j],
                       x='Sales Time',
                       y='Sales Value',
                       ax=ax,
                       ci=False,
                       legend='full',
                       lw=3,
                       alpha=0.8,
                       marker='o',
                       markersize=5)
      plt.xlabel('Time', fontsize=15)
      plt.ylabel('Sales Value', fontsize=15)
      plt.title('Rolling Average Sales vs. Time (per store)', fontsize=20)
      plt.legend(stores, fontsize=15, frameon=False, loc='upper left')
      plt.show()
```

In the above graph, I have plotted rolling sales across all stores in the dataset. Almost every sales curve has "three oscillations" trend at the microscopic level. Basically, the dependence the curve has since about a certain point onwards, is the mirror image of an upside-down version. This implies that the sales are oscillating at a higher and higher level every two months.

In the above graph, I have plotted rolling sales across all stores in the dataset. Almost every sales curve has "three oscillations" trend at the microscopic level. Basically, the dependence the curve has since about a certain point onwards, is the mirror image of an upside-down version. This implies that the sales are oscillating at a higher and higher level every two months.

The above plot contains the sales distribution for each store in the dataset. The stores in California seem to have the highest variance in sales, which might indicate that some stores in California grow significantly faster than others. Or, there is development ongoing. On the other hand, the stores in Texas seem to have the lowest variance in sales, which might indicate that the stores in Texas are not developing much or that some outlets in Texas develop much faster. On the other hand, the outlets in Texas also seem to have the highest overall mean sales.

```
[In]: fig, ax = plt.figure()
      sns.boxplot(data=sales_pivot, x='Store name', y='Sales Value')
      plt.xlabel('Store name', fontsize=15)
      plt.ylabel('Sales Value', fontsize=15)
      plt.title('Sales vs. Store name', fontsize=20)
      plt.show()
```

Mean sales vs. Store name

From the above graph, we can see the same trends. California stores have the highest variance and mean sales among all the stores in the dataset.

Rolling Average Price vs. Time (CA)

```
[In]: stores = ['California', 'Texas', 'Nevada', 'Canada']
      sales = pd.read_csv('sales.csv')
      sales.dropna(inplace=True)
      sales.reset_index(drop=True)
      sales['Sales Time'] = pd.to_datetime(sales['Year'],
                                           sales['Month'],
                                           sales['Day'],
                                           sales['Hour'],
                                           sales['Minute'],
                                           sales['Second'])
      for i, j in enumerate(stores):
          sales['Rolling Average Price vs. Time ({})'.format(j)] = sales[sales['Store state'] == j].\
              sort_values('Sales Time')['Sales Value'].\
              rolling(window=365).\
              mean()
      sales.head()
```

```
[Out]:
```

	Id	Store name	Sales Value	Sales Time	Year	Month	Day	Hour	Minute	Second	Rolling Average Price vs. Time (CA)	Rolling Average Price vs. Time (TX)	Rolling Average Price vs. Time (NV)
0	1	Store Two	1	2015-01-01 00:00:00	2015	1	1	0	0	0	NaN	NaN	NaN
1	2	Store Two	1	2015-01-01 00:00:00	2015	1	1	0	0	0	NaN	NaN	NaN
2	3	Store Two	1	2015-01-01 00:00:00	2015	1	1	0	0	0	NaN	NaN	NaN
3	4	Store Two	1	2015-01-01 00:00:00	2015	1	1	0	0	0	NaN	NaN	NaN
4	5	Store Two	1	2015-01-01 00:00:00	2015	1	1	0	0	0	NaN	NaN	NaN

```
[In]: fig, ax = plt.figure()
      for i, j in enumerate(stores):
          sns.lineplot(data=sales[sales['Store state'] == j],
                       x='Sales Time',
                       y='Rolling Average Price vs. Time ({})'.format(j),
                       ax=ax,
                       ci=False,
                       legend='full',
                       lw=3,
                       alpha=0.8,
                       marker='o',
                       markersize=5)
      plt.xlabel('Time', fontsize=15)
      plt.ylabel('Rolling Average Sales Vs. Time (CA)', fontsize=15)
      plt.title('Rolling Average Sales vs. Time (per state)', fontsize=20)
      plt.legend(stores, fontsize=15, frameon=False, loc='upper left')
      plt.show()
```

Rolling Average Sales vs. Time (CA)

In the above graph, we can see the trends clearly in sales among California stores. The sales curve shows never-ceased upward drive. The big outlook that this sales "trend" of development in California stands on the average over time, that what sales always around the average sales, the sales curve always "oscillates" around the average. In the above graph, the sales curve "oscillates" around the average. The above $\{1, 2, 3, 4, 5\}$ $\{6, 7, 8, 9, 10\}$ $\{11, 12, 13, 14, 15\}$ $\{16, 17, 18, 19, 20\}$ $\{21, 22, 23, 24, 25\}$ $\{26, 27, 28, 29, 30\}$ $\{31, 32, 33, 34, 35\}$ $\{36, 37, 38, 39, 40\}$

```
In [29]: score_f = pd.DataFrame()
         for i, (name, df) in enumerate(dfs.items()):
             if i > 0:
                 continue
             fig = px.bar(df[df[target] > cutoff],
                          x='Sensor',
                          y='Anomaly Count',
                          color='Sensor',
                          facet_row='Date',
                          category_orders={'Sensor': [f'Sensor{s}' for s in range(1, 26)], 'Date': [pd.Timestamp('2020-04-
                          height=1000,
                          title=f'{name}',
                          color_discrete_sequence=px.colors.qualitative.Plotly*25,
                          show_legend=False,
                          facet_row_spacing=0.01)
             fig.update_layout(
                 xaxis=dict(tickmode='array',
                            tickvalues=[f'Sensor{s}' for s in range(1, 26)],
                            ticktext=[f'{s}' for s in range(1, 26)],
                            tickangle=90,
                            tickfont=dict(size=10),
                            gridcolor='lightgrey',
                            gridwidth=0.5),
                 yaxis=dict(gridcolor='lightgrey',
                            gridwidth=0.5),
                 plot_bgcolor='white',
                 paper_bgcolor='white',
                 margin=dict(l=50, r=20, t=80, b=100),
                 font=dict(family="Arial, sans-serif"),
                 bargap=0.15,
                 bargroup=0.1,
                 facet_row_spacing=0.01
             )
             fig.update_traces(marker_line_width=0.5, marker_line_color='rgba(0,0,0,0.3)')
             fig.update_xaxes(showgrid=True)
             fig.update_yaxes(showgrid=True)
             fig.show()
```

Naive approach

Sensor
Sensor
Sensor

```
In [48]: sns.set(rc={'figure.figure.size':(11,4)})
         sns.set_style('white')
         sns.set_palette('muted')
         %matplotlib inline

         ts = ts[ts_begin:ts_end].resample('D').mean()

         plot_df = ts[ts_begin:ts_end]
         for c in range(0, len(ts.columns)):
             ax = plot_df[ts.columns[c]].plot(marker='.', alpha=0.5, linewidth=0.5, figsize=(11,4), label='Raw Data')
             ax.set_ylabel('Visits')
             ax.set_xlabel('')
             ax.legend(['Raw Data'],
                       loc='upper left');

         plot_df = ts[ts_begin:ts_end]
         for c in range(0, len(ts.columns)):
             ax = plot_df[ts.columns[c]].rolling(7, center=True, win_type='gaussian', min_periods=4).sum(std=2).plot(marker='.', alpha=0.5, linewidth=0.5, figsize=(11,4), label='7-Day Gaussian')
             ax.set_ylabel('Visits')
             ax.set_xlabel('')
             ax.legend(['7-Day Gaussian'],
                       loc='upper left');

         plot_df = ts[ts_begin:ts_end]
         for c in range(0, len(ts.columns)):
             ax = plot_df[ts.columns[c]].rolling(365, center=True, win_type='gaussian', min_periods=183).sum(std=2).plot(marker='.', alpha=0.5, linewidth=0.5, figsize=(11,4), label='Yearly Gaussian')
             ax.set_ylabel('Visits')
             ax.set_xlabel('')
             ax.legend(['Yearly Gaussian'],
                       loc='upper left');

         plot_df = ts[ts_begin:ts_end]
         for c in range(0, len(ts.columns)):
             ax = plot_df[ts.columns[c]].rolling(7, center=True, win_type='boxcar', min_periods=4).sum(std=2).plot(marker='.', alpha=0.5, linewidth=0.5, figsize=(11,4), label='7-Day Boxcar')
             ax.set_ylabel('Visits')
             ax.set_xlabel('')
             ax.legend(['7-Day Boxcar'],
                       loc='upper left');

         plot_df = ts[ts_begin:ts_end]
         for c in range(0, len(ts.columns)):
             ax = plot_df[ts.columns[c]].rolling(365, center=True, win_type='boxcar', min_periods=183).sum(std=2).plot(marker='.', alpha=0.5, linewidth=0.5, figsize=(11,4), label='Yearly Boxcar')
             ax.set_ylabel('Visits')
             ax.set_xlabel('')
             ax.legend(['Yearly Boxcar'],
                       loc='upper left');

         ts.to_pickle('./ts_pickle_filter_test', protocol=4)
         ts = pd.read_pickle('./ts_pickle_filter_test')
         ts.to_csv('./ts_filter_test.csv')
         ts[ts_begin:ts_end].head(10)
```

```
In [49]: sns.set(rc={'figure.figure.size':(11,4)})
         sns.set_style('white')
         sns.set_palette('muted')
         %matplotlib inline

         ts = ts[ts_begin:ts_end].resample('D').mean()

         plot_df = ts[ts_begin:ts_end]
         for c in range(0, len(ts.columns)):
             ax = plot_df[ts.columns[c]].plot(marker='.', alpha=0.5, linewidth=0.5, figsize=(11,4), label='Raw Data')
             ax.set_ylabel('Visits')
             ax.set_xlabel('')
             ax.legend(['Raw Data'],
                       loc='upper left');

         plot_df = ts[ts_begin:ts_end]
         for c in range(0, len(ts.columns)):
             ax = plot_df[ts.columns[c]].rolling(7, center=True, win_type='gaussian', min_periods=4).sum(std=2).plot(marker='.', alpha=0.5, linewidth=0.5, figsize=(11,4), label='7-Day Gaussian')
             ax.set_ylabel('Visits')
             ax.set_xlabel('')
             ax.legend(['7-Day Gaussian'],
                       loc='upper left');

         plot_df = ts[ts_begin:ts_end]
         for c in range(0, len(ts.columns)):
             ax = plot_df[ts.columns[c]].rolling(365, center=True, win_type='gaussian', min_periods=183).sum(std=2).plot(marker='.', alpha=0.5, linewidth=0.5, figsize=(11,4), label='Yearly Gaussian')
             ax.set_ylabel('Visits')
             ax.set_xlabel('')
             ax.legend(['Yearly Gaussian'],
                       loc='upper left');

         plot_df = ts[ts_begin:ts_end]
         for c in range(0, len(ts.columns)):
             ax = plot_df[ts.columns[c]].rolling(7, center=True, win_type='boxcar', min_periods=4).sum(std=2).plot(marker='.', alpha=0.5, linewidth=0.5, figsize=(11,4), label='7-Day Boxcar')
             ax.set_ylabel('Visits')
             ax.set_xlabel('')
             ax.legend(['7-Day Boxcar'],
                       loc='upper left');

         plot_df = ts[ts_begin:ts_end]
         for c in range(0, len(ts.columns)):
             ax = plot_df[ts.columns[c]].rolling(365, center=True, win_type='boxcar', min_periods=183).sum(std=2).plot(marker='.', alpha=0.5, linewidth=0.5, figsize=(11,4), label='Yearly Boxcar')
             ax.set_ylabel('Visits')
             ax.set_xlabel('')
             ax.legend(['Yearly Boxcar'],
                       loc='upper left');

         ts.to_pickle('./ts_pickle_filter_test', protocol=4)
         ts = pd.read_pickle('./ts_pickle_filter_test')
         ts.to_csv('./ts_filter_test.csv')
         ts[ts_begin:ts_end].head(10)
```

```
In [50]: sns.set(rc={'figure.figure.size':(11,4)})
         sns.set_style('white')
         sns.set_palette('muted')
         %matplotlib inline

         ts = ts[ts_begin:ts_end].resample('D').mean()

         plot_df = ts[ts_begin:ts_end]
         for c in range(0, len(ts.columns)):
             ax = plot_df[ts.columns[c]].plot(marker='.', alpha=0.5, linewidth=0.5, figsize=(11,4), label='Raw Data')
             ax.set_ylabel('Visits')
             ax.set_xlabel('')
             ax.legend(['Raw Data'],
                       loc='upper left');

         plot_df = ts[ts_begin:ts_end]
         for c in range(0, len(ts.columns)):
             ax = plot_df[ts.columns[c]].rolling(7, center=True, win_type='gaussian', min_periods=4).sum(std=2).plot(marker='.', alpha=0.5, linewidth=0.5, figsize=(11,4), label='7-Day Gaussian')
             ax.set_ylabel('Visits')
             ax.set_xlabel('')
             ax.legend(['7-Day Gaussian'],
                       loc='upper left');

         plot_df = ts[ts_begin:ts_end]
         for c in range(0, len(ts.columns)):
             ax = plot_df[ts.columns[c]].rolling(365, center=True, win_type='gaussian', min_periods=183).sum(std=2).plot(marker='.', alpha=0.5, linewidth=0.5, figsize=(11,4), label='Yearly Gaussian')
             ax.set_ylabel('Visits')
             ax.set_xlabel('')
             ax.legend(['Yearly Gaussian'],
                       loc='upper left');

         plot_df = ts[ts_begin:ts_end]
         for c in range(0, len(ts.columns)):
             ax = plot_df[ts.columns[c]].rolling(7, center=True, win_type='boxcar', min_periods=4).sum(std=2).plot(marker='.', alpha=0.5, linewidth=0.5, figsize=(11,4), label='7-Day Boxcar')
             ax.set_ylabel('Visits')
             ax.set_xlabel('')
             ax.legend(['7-Day Boxcar'],
                       loc='upper left');

         plot_df = ts[ts_begin:ts_end]
         for c in range(0, len(ts.columns)):
             ax = plot_df[ts.columns[c]].rolling(365, center=True, win_type='boxcar', min_periods=183).sum(std=2).plot(marker='.', alpha=0.5, linewidth=0.5, figsize=(11,4), label='Yearly Boxcar')
             ax.set_ylabel('Visits')
             ax.set_xlabel('')
             ax.legend(['Yearly Boxcar'],
                       loc='upper left');

         ts.to_pickle('./ts_pickle_filter_test', protocol=4)
         ts = pd.read_pickle('./ts_pickle_filter_test')
         ts.to_csv('./ts_filter_test.csv')
         ts[ts_begin:ts_end].head(10)
```

```
In [51]: %matplotlib inline
         %load_ext rpy2.ipynb
         %Rpush ts_filter_test ts_begin ts_end
         %R ts_filter_test = read.csv('./ts_filter_test.csv', header=TRUE)
         %R ts_filter_test$Date = as.Date(ts_filter_test$X, format='%Y-%m-%d')
         %R ts_filter_test$X = NULL
         %R ts_filter_test = ts_filter_test[ts_filter_test$Date >= ts_begin & ts_filter_test$Date <= ts_end,]
         %R ts_filter_test$Date = NULL
         %R ts_filter_test = ts_filter_test[,c(1:4,6,5)]
         %R ts_filter_test = ts_filter_test[,c(1:5,7,6)]
         %R ts_filter_test = ts_filter_test[,c(1:6,8,7)]
         %R ts_filter_test = ts_filter_test[,c(1:7,9,8)]
         %R ts_filter_test = ts_filter_test[,c(1:8,10,9)]
         %R ts_filter_test = ts_filter_test[,c(1:9,11,10)]
         %R ts_filter_test = ts_filter_test[,c(1:10,12,11)]
         %R ts_filter_test = ts_filter_test[,c(1:11,13,12)]
         %R ts_filter_test = ts_filter_test[,c(1:12,14,13)]
         %R ts_filter_test = ts_filter_test[,c(1:13,15,14)]
         %R ts_filter_test =
```