چالش سوم: M5 Forecasting - Accuracy

فراهم‌آورنده: سینا حیدری

زمستان ۹۸

برای مشاهده‌ی جزئیات این چالش در کگل به لینک زیر مراجعه کنید.

https://www.kaggle.com/c/m5-forecasting-accuracy

۱ در مورد این چالش

نکته: این رقابت یکی از سه رقابتی است که چالش M5 forcasting را تشکیل می‌دهد. آیا می‌توانید با دقت بالا، تعداد واحدهایی از هر محصول، که توسط وال‌مارت در آمریکا به‌فروش خواهد رفت را پیش‌بینی کنید؟
یک فروشگاه، چه تعداد ابزار چادرزنی را، در هر ماه از سال خواهد فروخت؟ برای کسانی که دانش پیشین ندارند، محاسبه‌ی فروش‌ها در این سطح به‌اندازه‌ی پیش‌بینی آب‌وهوا سخت جلوه می‌کند. هر دو نوع پیش‌بینی[۱]، وابسته به دانش و داده‌های مبتنی بر تاریخ است. یک پیش‌بینی اشتباه آب‌وهوا شاید باعث شود چترتان را در یک روز آفتابی بدون استفاده با خود حمل کنید؛ اما یک پیش‌بینی اشتباه در کسب‌وکار ممکن است موجب از دست رفتن فرصت‌های واقعی شود. در این رقابت، علاوه بر روش‌های سنتی پیش‌بینی، از **یادگیری ماشین** هم برای بالا بردن دقت پیش‌بینی‌ها بهره می‌بریم.
مرکز پیش‌بینی ماکریداکیس (MOFC) در دانشگاه نیکوزیا تحقیقات پیشرفته‌یی را در زمینه‌ی پیش‌بینی هدایت کرده و آموزش پیش‌بینی کسب‌وکار را فراهم می‌آورد. این مرکز به‌شرکت‌ها کمک می‌کند تا به پیشبینی‌های دقیق دست یافته؛ درجات عدم‌قطعیت را تقریب زده، از اشتباهات هزینه‌بر دوری کرده، و بهترین روش‌های پیش‌بینی را در کسب‌وکارشان اعمال کنند. MOFC به‌دلیل رقابت‌های Makridakis-که اولین رقابت‌اش در سال ۱۹۸۰ برگذار شد-در دنیا به خوبی شناخته شده است.
در این رقابت از داده‌ی سلسله‌مراتبی فروش‌های **وال‌مارت**-که از بزرگ‌ترین شرکت‌های دنیا است-برای پیش‌بینی روزانه‌ی فروش‌ها، در ۲۸ روز آینده استفاده کرده و برای این پیش‌بینی‌ها عدم‌قطعیت را تقریب بزنیم. مجموعه‌داده، فروشگاه‌های داخل سه ایالت آمریکا (کالیفرنیا، تگزاس و ویسکانسین) را در بر گرفته و شامل درجه‌ی آیتم، دپارتمان، دسته‌بندی محصول و جزئیات فروشگاه می‌شود. علاوه بر این، شامل مقادیر توضیحی، همچون؛ قیمت، تبلیغات، روز هفته و رخدادهای خاص می‌شود. از تمام این‌ها می‌توان استفاده کرد تا دقت پیش‌بینی را افزایش داد.
اگر موفق باشیم، کارمان در مسیر پیشرفت تئوری و عملی پیش‌بینی جلو خواهد رفت. روش‌های استفاده شده را می‌توان برای راه‌اندازی مناسب انبارها یا درجه‌های سرویس‌دهی به‌کار برد. MOFC از طریق حمایت و آموزش به توزیع ابزارها و دانش کمک می‌کند تا بقیه هم بتوانند به‌دقت بالاتر و درجه‌بندی بهتر پیش‌بینی دست یافته، ضایعات را کاهش دهند و آمادگی پذیرفتن ریسک‌های عدم‌قطعیت را داشته باشند.

---

[۱] forcasting

۲ در مورد این گزارش

در این گزارش، ساختار داده را به‌طور خلاصه توصیف می‌کنیم. سپس، داده را با استفاده از دو کتابخانه‌ی Matplotlib و Plotly مصورسازی می‌کنیم. در نهایت، نشان خواهیم چه رویکردهایی به این مسئله از طریق الگوریتم‌های پیش‌بینی وجود دارد.

۳ بررسی اکتشافی داده

داده‌ی ما شامل پنج فایل می‌شود:

- **calendar.csv** - Contains the dates on which products are sold. The dates are in a yyyy/dd/mm format.
- **sales_train_validation.csv** - Contains the historical daily unit sales data per product and store [d_1 - d_1913].
- **submission.csv** - Demonstrates the correct format for submission to the competition.
- **sell_prices.csv** - Contains information about the price of the products sold per store and date.
- **sales_train_evaluation.csv** - Available one month before the competition deadline. It will include sales for [d_1 - d_1941].

در این رقابت ما باید فروش‌ها را برای [d_1942 - d_1969] پیش‌بینی کنیم. این سطرها مجموعه‌داده‌ی آموزش ما را تشکیل داده و سطرهای بازمانده، داده‌ی آموزشی ما را تشکیل می‌دهد. حال که مجموعه‌داده را می‌شناسیم و می‌دانیم چه چیزی را باید پیش‌بینی کنیم، بیایید مجموعه‌داده را مصورسازی کنیم.

۱.۳ بارگذاری کتابخانه‌های مورد نیاز

```
import os
import gc
import time
import math
import datetime
from math import log, floor
from sklearn.neighbors import KDTree

import numpy as np
import pandas as pd
```

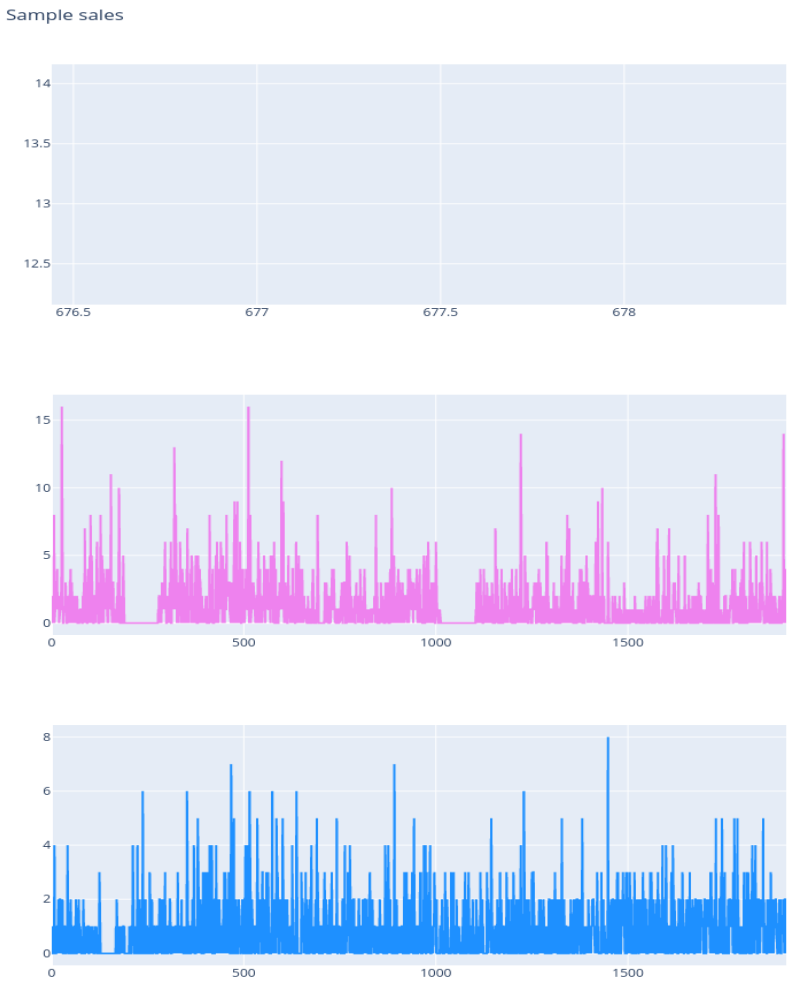
```
from pathlib import Path
from sklearn.utils import shuffle
from tqdm.notebook import tqdm as tqdm

import seaborn as sns
from matplotlib import colors
import matplotlib.pyplot as plt
from matplotlib.colors import Normalize

import plotly.express as px
import plotly.graph_objects as go
import plotly.figure_factory as ff
from plotly.subplots import make_subplots

import pywt
from statsmodels.robust import mad

import scipy
import statsmodels
from scipy import signal
import statsmodels.api as sm
from fbprophet import Prophet
from scipy.signal import butter, deconvolve
from statsmodels.tsa.arima_model import ARIMA
from statsmodels.tsa.api import ExponentialSmoothing,
    SimpleExpSmoothing, Holt

import warnings
warnings.filterwarnings("ignore")
```

نمونه

نمونه‌هایی از مجموعه‌داده فروش‌ها:

```
ids = sorted(list(set(sales_train_val['id'])))
d_cols = [c for c in sales_train_val.columns if 'd_' in c]
x_1 = sales_train_val.loc[sales_train_val['id'] == ids[2]].set_index('
    id')[d_cols].values[0]
x_2 = sales_train_val.loc[sales_train_val['id'] == ids[1]].set_index('
    id')[d_cols].values[0]
x_3 = sales_train_val.loc[sales_train_val['id'] == ids[17]].set_index('
    id')[d_cols].values[0]

fig = make_subplots(rows=3, cols=1)

fig.add_trace(go.Scatter(x=np.arange(len(x_1)), y=x_1, showlegend=False
    ,
mode='lines', name="First sample",
marker=dict(color="mediumseagreen")),
row=1, col=1)

fig.add_trace(go.Scatter(x=np.arange(len(x_2)), y=x_2, showlegend=False
    ,
mode='lines', name="Second sample",
marker=dict(color="violet")),
row=2, col=1)
```

```
18
19 fig.add_trace(go.Scatter(x=np.arange(len(x_3)), y=x_3, showlegend=False
       ,
20 mode='lines', name="Third sample",
21 marker=dict(color="dodgerblue")),
22 row=3, col=1)
23
24 fig.update_layout(height=1200, width=800, title_text="Sample sales")
25 fig.show()
```



این نمونه‌ها، شامل رکورد فروش‌های فروشگاه‌هایی است که به‌طور تصادفی از مجموعه‌داده انتخاب شده است. همانطور که انتظار می‌رود، داده فروش‌ها بسیار نامنظم است، چراکه عوامل زیادی روی فروش‌های یک روز مشخص تاثیرگذار است. در روزهای مشخصی تعداد فروش صفر است؛ و دلیل آن در دسترس نبودن محصول در آن روزهاست.

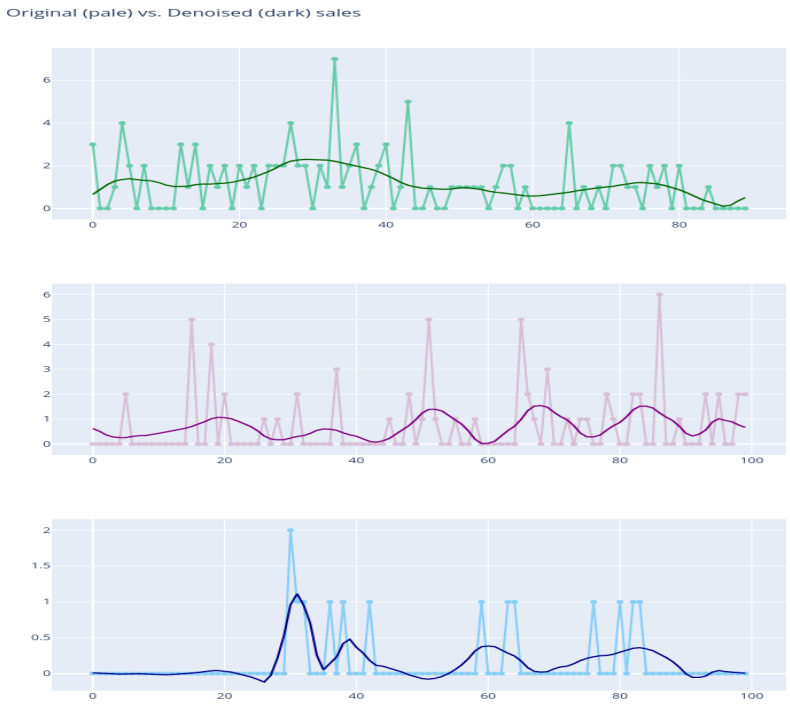
Denoising

حالا نشان خواهیم داد چگونه می‌توان نویز این فروش‌های متغیر را از بین برد، تا بتوانیم ترندهای اساسی را استخراج کنیم. این روش موجب از دست دادن اطلاعاتی از سری زمانی اصلی می‌شود، اما استخراج ویژگی‌های مشخص در جهت نشان دادن ترندها در سری زمانی مفید است.

Wavelet denoising

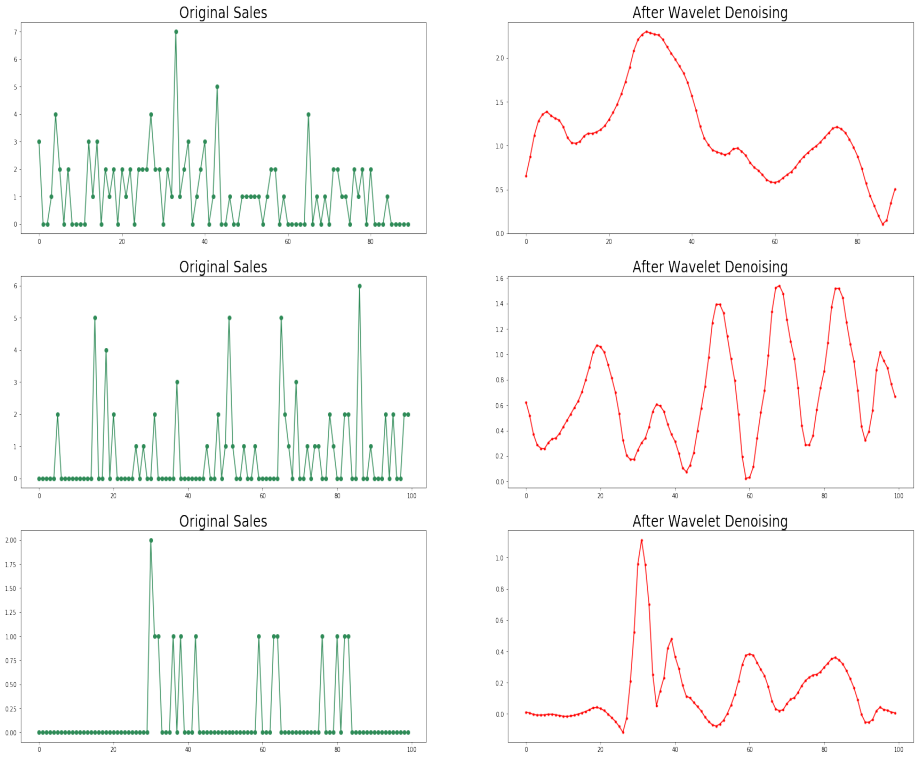
Wavelet denoising یک روش برای حذف نویز اضافی از سری زمانی است. این روش ضرایبی که به‌آن wavelet coefficients می‌گویند را محسابه می‌کند. این ضرایب تعیین می‌کند که کدام بخش‌های داده را نگاه داریم (سیگنال) و کدام بخش‌ها را دور بریزیم (نویز).
ما از MAD[۲] جهت پی‌بردن به تصادفی بودن فروش‌ها بهره می‌بریم که بر اساس آن minimum threshold برای ضرایب در سری زمانی تعیین می‌شود. ما ضرایبی که مقدار پایین‌تری دارند را فیلتر کرده سپس، داده فروش‌ها را بازسازی می‌کنیم. همین است!–با موفقیت نویز را از داده فروش‌ها حذف کردیم.

```
def maddest(d, axis=None):
return np.mean(np.absolute(d - np.mean(d, axis)), axis)

def denoise_signal(x, wavelet='db4', level=1):
coeff = pywt.wavedec(x, wavelet, mode="per")
sigma = (1/0.6745) * maddest(coeff[-level])

uthresh = sigma * np.sqrt(2*np.log(len(x)))
coeff[1:] = (pywt.threshold(i, value=uthresh, mode='hard') for i in
    coeff[1:])

return pywt.waverec(coeff, wavelet, mode='per')
```



[۲] mean absolute deviation

در گراف‌های بالا، lineplot-هایی که رنگ تیره‌تر دارند، نشان‌دهنده داده فروش‌های denoised است؛ و لاین‌پلات‌هایی که رنگ روشن‌تر دارند فروش‌های اصلی را نشان می‌دهد. مشاهده می‌کنید که Wavelet denoising قادر است با موفقیت general trend را در داده فروش‌ها پیدا کند. پیدا کردن این الگوها در فروش‌ها می‌تواند در تولید ویژگی‌ها برای آموزش یک مدل مفید باشد.
دیاگرام زیر، گراف‌ها را کنار هم نشان می‌دهد:



Average smoothing

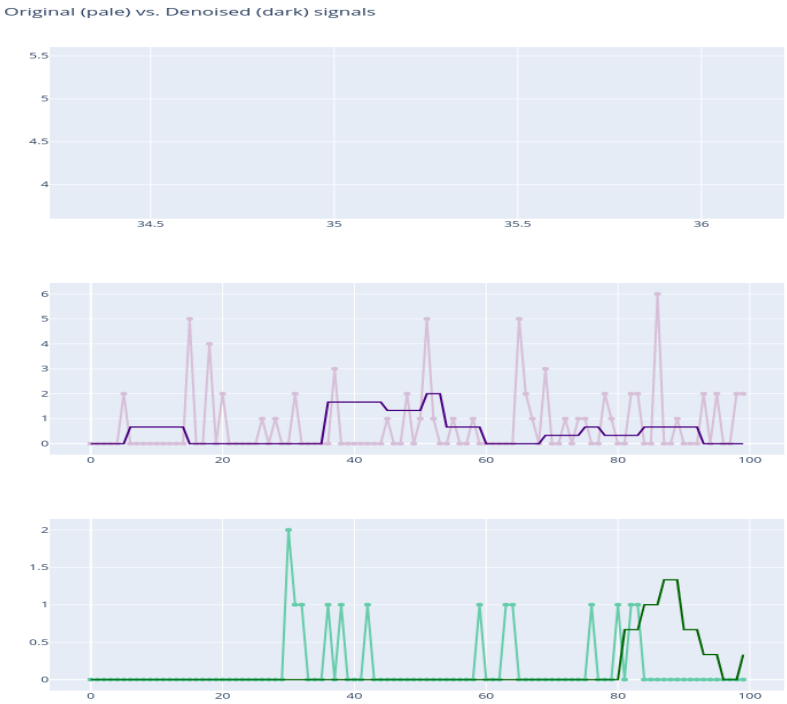
Average smoothing یک روش نسبتا ساده برای denoise کردن داده سری زمانی است. در این روش، ما یک پنجره با اندازه ثابت (مثلا ۱۰) در نظر می‌گیریم. ابتدا پنجره را در اول سری زمانی قرار می‌دهیم (ده سطر اول) و میانه این بخش را محاسبه می‌کنیم. حال بر اساس یک گام مشخص، پنجره را تا انتهای سری جلو می‌بریم. برای تمام پنجره‌ها میانه محاسبه خواهد شد؛ سپس، با میانه‌های به‌دست آمده، سری زمانی جدید ساخته می‌شود که داده denoised را تشکیل می‌دهد.

```
def average_smoothing(signal, kernel_size=3, stride=1):
  sample = []
  start = 0
  end = kernel_size
  while end <= len(signal):
    start = start + stride
    end = end + stride
```

```
    sample.extend(np.ones(end - start)*np.mean(signal[start:end]))
  return np.array(sample)
```

مقایسه سیگنال‌های اصلی (کم‌رنگ) با سیگنال‌های denoised شده (پررنگ) :

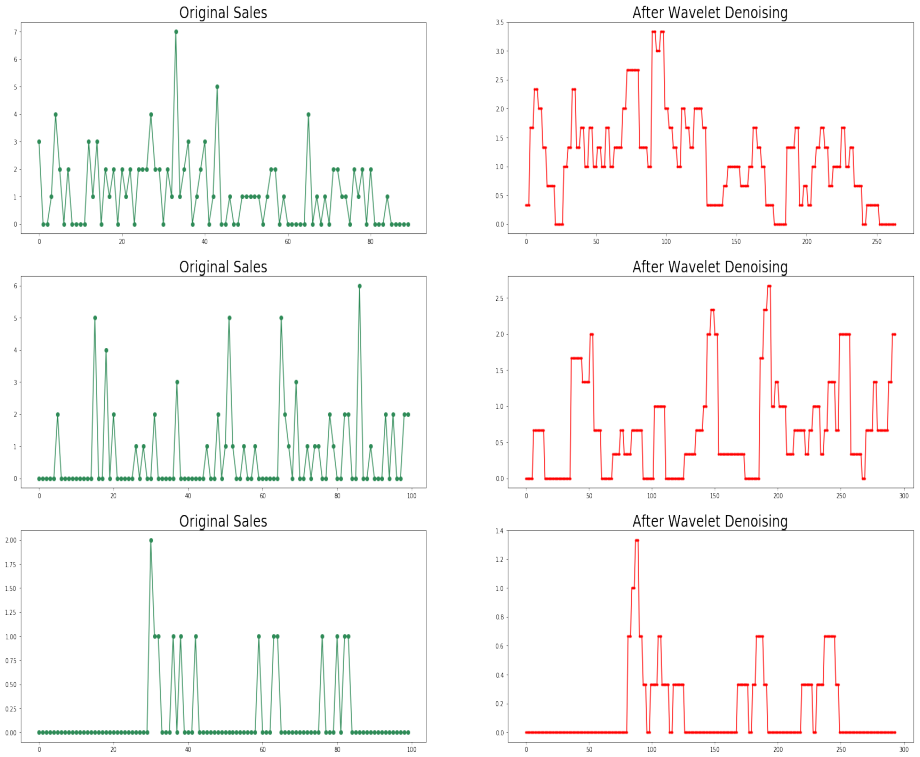
```
y_a1 = average_smoothing(x_1)
y_a2 = average_smoothing(x_2)
y_a3 = average_smoothing(x_3)

fig = make_subplots(rows=3, cols=1)

fig.add_trace(
go.Scatter(x=np.arange(len(x_1)), mode='lines+markers', y=x_1, marker=dict(color="lightskyblue"), showlegend=False,
name="Original sales"),
row=1, col=1
)

fig.add_trace(
go.Scatter(x=np.arange(len(x_1)), y=y_a1, mode='lines', marker=dict(color="navy"), showlegend=False,
name="Denoised sales"),
row=1, col=1
)

fig.add_trace(
go.Scatter(x=np.arange(len(x_2)), mode='lines+markers', y=x_2, marker=dict(color="thistle"), showlegend=False),
row=2, col=1
)

fig.add_trace(
go.Scatter(x=np.arange(len(x_2)), y=y_a2, mode='lines', marker=dict(color="indigo"), showlegend=False),
row=2, col=1
)

fig.add_trace(
go.Scatter(x=np.arange(len(x_3)), mode='lines+markers', y=x_3, marker=dict(color="mediumaquamarine"), showlegend=False),
row=3, col=1
)

fig.add_trace(
go.Scatter(x=np.arange(len(x_3)), y=y_a3, mode='lines', marker=dict(color="darkgreen"), showlegend=False),
row=3, col=1
)

fig.update_layout(height=1200, width=800, title_text="Original (pale) vs. Denoised (dark) signals")
fig.show()
```



می‌توان دید average smoothing به‌اندازه Wavelet denoising، در پیدا کردن ترندهای ماکروسکوپی و الگوها موثر نیست. بیشتر نویز داده حتی بعد denoise کردن پایدار است. از این رو، Wavelet denoising به وضوح برای پیدا کردن ترندها بهتر است؛ هر چند، average smoothing و rolling mean هم برای محاسبه ویژگی‌های مفید، قابل استفاده است.

دیاگرام زیر این گراف‌ها را در کنار هم نشان می‌دهد:

```
fig, ax = plt.subplots(nrows=3, ncols=2, figsize=(30, 20))

ax[0, 0].plot(x_1, color='seagreen', marker='o')
ax[0, 0].set_title('Original Sales', fontsize=24)
ax[0, 1].plot(y_a1, color='red', marker='.')
ax[0, 1].set_title('After Wavelet Denoising', fontsize=24)

ax[1, 0].plot(x_2, color='seagreen', marker='o')
ax[1, 0].set_title('Original Sales', fontsize=24)
ax[1, 1].plot(y_a2, color='red', marker='.')
ax[1, 1].set_title('After Wavelet Denoising', fontsize=24)

ax[2, 0].plot(x_3, color='seagreen', marker='o')
ax[2, 0].set_title('Original Sales', fontsize=24)
ax[2, 1].plot(y_a3, color='red', marker='.')
ax[2, 1].set_title('After Wavelet Denoising', fontsize=24)

plt.show()
```



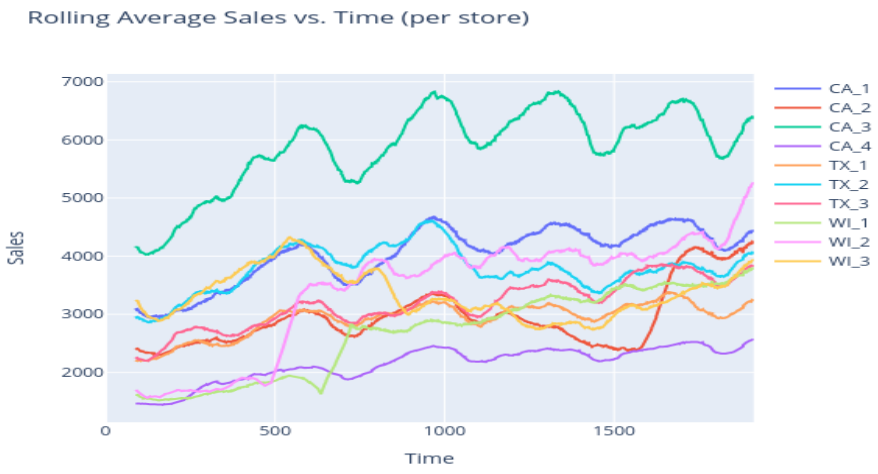
فروشگاه‌ها و ایالت‌ها

حالا برای کسب اطلاعات مفید، می‌خواهیم نگاهی به فروش‌ها در فروشگاه‌ها و ایالت‌های متفاوت بی‌اندازیم.

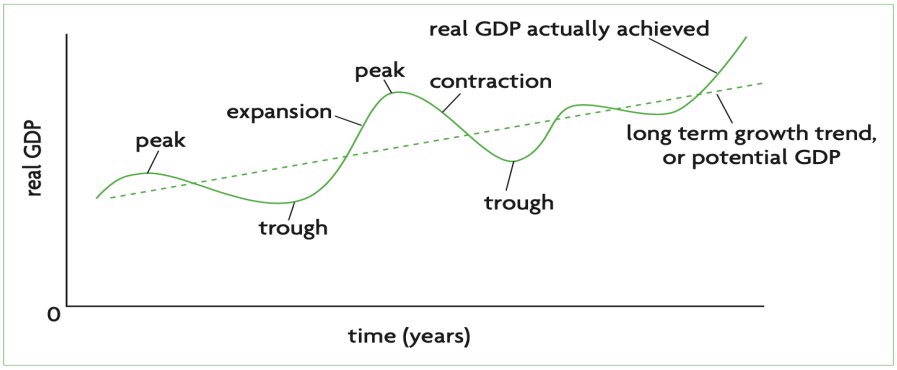
Rolling Average Price vs. Time (per store)

```
past_sales = sales_train_val.set_index('id')[d_cols] \
.T \
.merge(calendar.set_index('d')['date'],
left_index=True,
right_index=True,
validate='1:1') \
.set_index('date')

store_list = selling_prices['store_id'].unique()
means = []
fig = go.Figure()
for s in store_list:
  store_items = [c for c in past_sales.columns if s in c]
  data = past_sales[store_items].sum(axis=1).rolling(90).mean()
```

```
  means.append(np.mean(past_sales[store_items].sum(axis=1)))
  fig.add_trace(go.Scatter(x=np.arange(len(data)), y=data, name=s))

fig.update_layout(yaxis_title="Sales", xaxis_title="Time", title="
    Rolling Average Sales vs. Time (per store)")
```

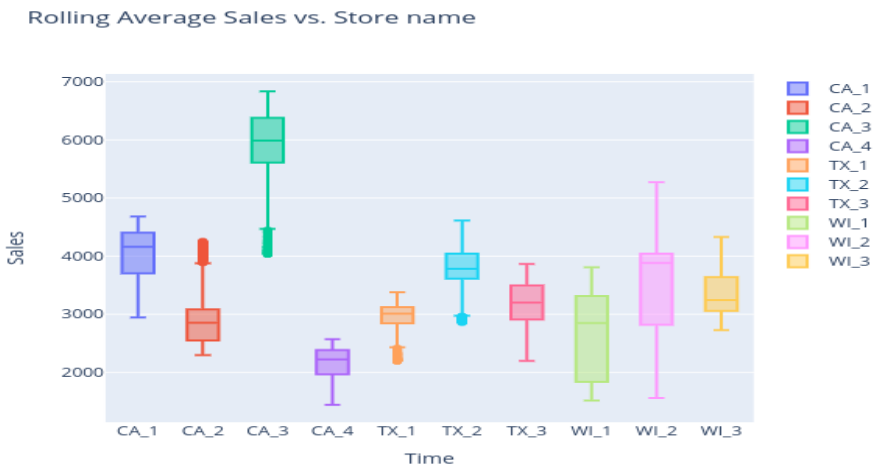


در گراف بالا، ما rolling sales را برای تمام فروشگاه‌ها ترسیم کرده‌ایم. تقریباً تمام منحنی‌های فروش در پلات بالا، دارای نوسان خطی هستند. فروش‌ها مانند یک موج سینوسی با میانگین مشخص در نوسان هستند، اما این میانگین ترند خطی رو به‌بالا دارد. این نشان می‌دهد فروش‌ها هر چند ماه با سطح بالاتر نوسان می‌کنند.
این ترند یادآور business cycle است، که در آن نوسانات کوتاه‌مدت وجود دارد اما دارای یک رشد خطی در بلندمدت است. شاید چنین ترندهای مقیاس‌پایین، در سطح فروشگاه‌ها، باعث درک بهتر ترندهای macroeconomic شوند.
در پایین نمایشی از این مفاهیم را مشاهده می‌کنید:




```
fig = go.Figure()
```

```
2
3  for i, s in enumerate(store_list):
4    store_items = [c for c in past_sales.columns if s in c]
5    data = past_sales[store_items].sum(axis=1).rolling(90).mean()
6    fig.add_trace(go.Box(x=[s]*len(data), y=data, name=s))
7
8  fig.update_layout(yaxis_title="Sales", xaxis_title="Time", title="
       Rolling Average Sales vs. Store name ")
```



پلات بالا توزیع فروش‌ها برای هر فروشگاه در مجموعه‌داده را نشان می‌دهد. فروشگاه‌هایی که در کالیفرنیا هستند، به‌نظر دارای بیشترین واریانس در فروش‌ها هستند که نشان می‌دهد بعضی مکان‌ها در کالیفرنیا به‌طور قابل ملاحضه بیشتر از مکان‌های دیگر رشد دارند. از دید دیگر، فروش‌های ویسکانسین و تگزاس به‌نظر در میان خودشان بدون واریانس زیاد، ثابت‌قدم هستند. این می‌تواند نشان‌دهنده توسعه یک‌دست در این ایالت‌ها باشد. علاوه بر این، کالیفرنیا دارای بیشتر میانگین فروش‌ها است.

```
df = pd.DataFrame(np.transpose([means, store_list]))
df.columns = ["Mean sales", "Store name"]
px.bar(df, y="Mean sales", x="Store name", color="Store name", title="
    Mean sales vs. Store name")
```

۴ مدلینگ

در این بخش از روش‌های مختلفی برای پیش‌بینی سری زمانی استفاده برده‌ایم. این روش‌ها عبارت‌اند از:

- naive approach
- moving average
- Holt linear
- exponential smoothing
- ARIMA
- Prophet

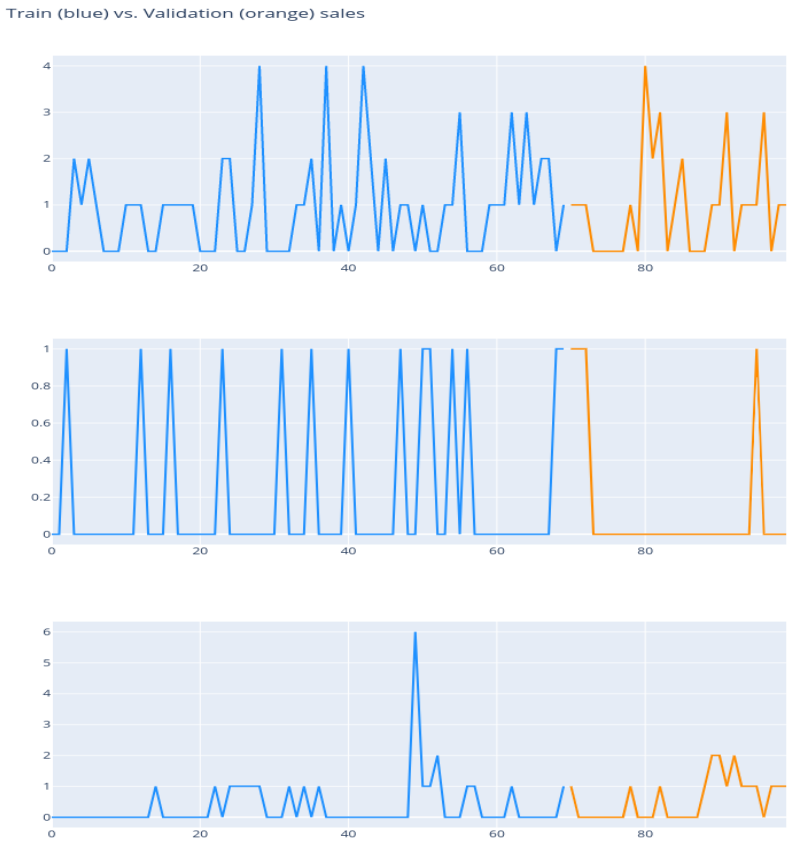
Train/Val split

ابتدا دو مجموعه‌داده کوچک آموزشی و آزمایشی را جدا می‌کنیم تا مدل‌ها را ارزیابی کنیم. فروش‌های ۳۰ روز پایانی برای داده آزمایشی و فروش‌های ۷۰ روز قبل‌تر برای داده آموزشی انتخاب شده است. باید فروش‌های داده آزمایشی را با استفاده از داده آموزشی پیش‌بینی کنیم.

```
train_dataset = sales_train_val[d_cols[-100:-30]]
val_dataset = sales_train_val[d_cols[-30:]]
```

در پلات پایین، فروش‌های سه نقطه نمونه را مشاهده می‌کنید. ما از این نمونه‌ها برای نشان دادن عملکرد مدل‌ها بهره خواهیم برد.

```
fig = make_subplots(rows=3, cols=1)

fig.add_trace(
go.Scatter(x=np.arange(70), mode='lines', y=train_dataset.loc[0].values, marker=dict(color="dodgerblue"), showlegend=False,
name="Original signal"),
row=1, col=1
)

fig.add_trace(
go.Scatter(x=np.arange(70, 100), y=val_dataset.loc[0].values, mode='lines', marker=dict(color="darkorange"), showlegend=False,
name="Denoised signal"),
row=1, col=1
)

fig.add_trace(
go.Scatter(x=np.arange(70), mode='lines', y=train_dataset.loc[1].values, marker=dict(color="dodgerblue"), showlegend=False),
row=2, col=1
)

fig.add_trace(
go.Scatter(x=np.arange(70, 100), y=val_dataset.loc[1].values, mode='lines', marker=dict(color="darkorange"), showlegend=False),
row=2, col=1
)

fig.add_trace(
go.Scatter(x=np.arange(70), mode='lines', y=train_dataset.loc[2].values, marker=dict(color="dodgerblue"), showlegend=False),
row=3, col=1
)

fig.add_trace(
go.Scatter(x=np.arange(70, 100), y=val_dataset.loc[2].values, mode='lines', marker=dict(color="darkorange"), showlegend=False),
row=3, col=1
)

fig.update_layout(height=1200, width=800, title_text="Train (blue) vs. Validation (orange) sales")
fig.show()
```



Naive approach

رویکرد اول رویکرد ساده‌یی است که فروش‌های روز بعد را برابر فروش‌های امروز قرار می‌دهد. این مدل را می‌توان به‌صورت زیر خلاصه‌سازی کرد:

$$\hat{y}_{t+1} = y_t$$

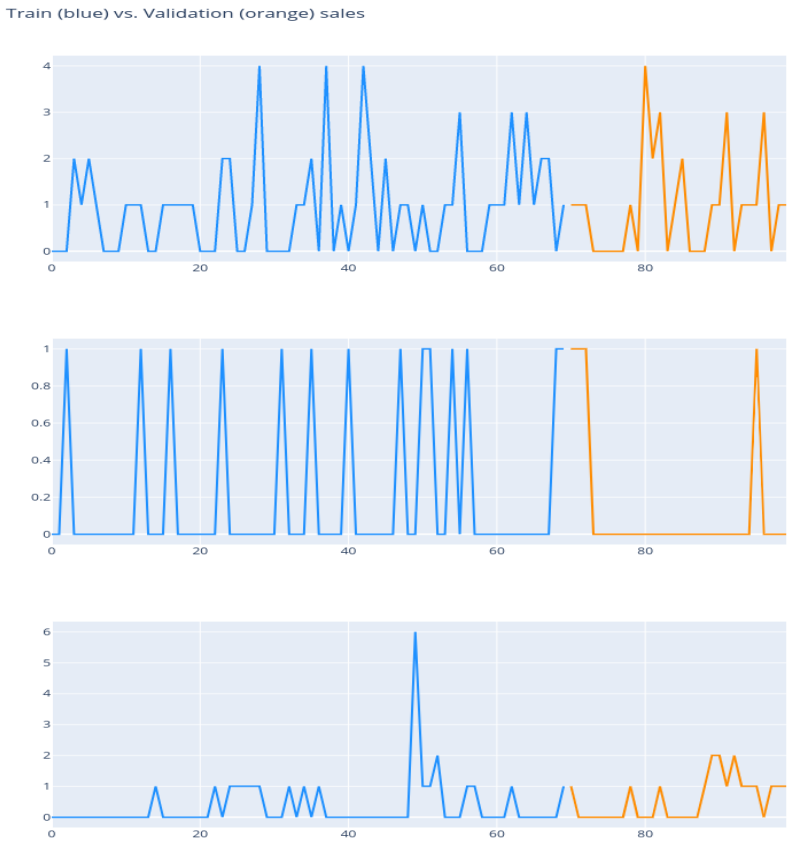
بیایید عملکرد مدل naive approach را بر روی نمونه‌ها بررسی کنیم:

```
predictions = []
for i in range(len(val_dataset.columns)):
  if i == 0:
    predictions.append(train_dataset[train_dataset.columns[-1]].values)
  else:
```

```
    predictions.append(val_dataset[val_dataset.columns[i-1]].values)

predictions = np.transpose(np.array([row.tolist() for row in
    predictions]))
error_naive = np.linalg.norm(predictions[:3] - val_dataset.values[:3])/
    len(predictions[0])
```

```
pred_1 = predictions[0]
pred_2 = predictions[1]
pred_3 = predictions[2]

fig = make_subplots(rows=3, cols=1)

fig.add_trace(
go.Scatter(x=np.arange(70), mode='lines', y=train_dataset.loc[0].values
    , marker=dict(color="dodgerblue"),
name="Train"),
row=1, col=1
)

fig.add_trace(
go.Scatter(x=np.arange(70, 100), y=val_dataset.loc[0].values, mode='
    lines', marker=dict(color="darkorange"),
name="Val"),
row=1, col=1
)

fig.add_trace(
go.Scatter(x=np.arange(70, 100), y=pred_1, mode='lines', marker=dict(
    color="seagreen"),
name="Pred"),
row=1, col=1
)

fig.add_trace(
go.Scatter(x=np.arange(70), mode='lines', y=train_dataset.loc[1].values
    , marker=dict(color="dodgerblue"), showlegend=False),
row=2, col=1
)

fig.add_trace(
go.Scatter(x=np.arange(70, 100), y=val_dataset.loc[1].values, mode='
    lines', marker=dict(color="darkorange"), showlegend=False),
row=2, col=1
)

fig.add_trace(
go.Scatter(x=np.arange(70, 100), y=pred_2, mode='lines', marker=dict(
    color="seagreen"), showlegend=False,
name="Denoised signal"),
row=2, col=1
)

fig.add_trace(
go.Scatter(x=np.arange(70), mode='lines', y=train_dataset.loc[2].values
    , marker=dict(color="dodgerblue"), showlegend=False),
row=3, col=1
)
```

```
45
46 fig.add_trace(
47 go.Scatter(x=np.arange(70, 100), y=val_dataset.loc[2].values, mode='
       lines', marker=dict(color="darkorange"), showlegend=False),
48 row=3, col=1
49 )
50
51 fig.add_trace(
52 go.Scatter(x=np.arange(70, 100), y=pred_3, mode='lines', marker=dict(
       color="seagreen"), showlegend=False,
53 name="Denoised signal"),
54 row=3, col=1
55 )
56
57 fig.update_layout(height=1200, width=800, title_text="Naive approach")
58 fig.show()
```



مشاهده می‌کنید، پیش‌بینی‌های انجام شده توسط رویکرد ساده دقیق نیست که از این مدل ساده همین انتظار می‌رود.

Moving average

روش moving average پیچیده‌تراز روش ساده بالاست. این روش، میانگین فروش‌های ۳۰-یا هر چند- روز را به‌عنوان فروش‌های روز بعد محاسبه می‌کند. در این روش ۳۰ گام زمانی در نظر گرفته می‌شود، به این علت کمتر به نوسانات کوتاه مدت تمایل نشان می‌دهد. این مدل را می‌توان به‌صورت زیر خلاصه‌سازی کرد:

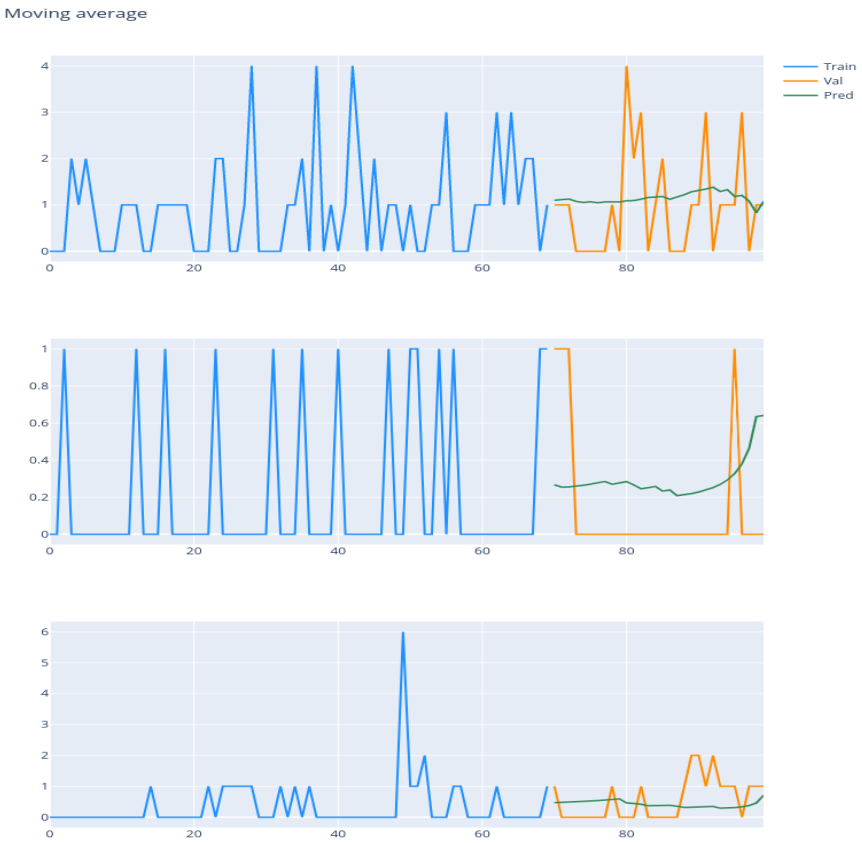
$$\hat{y}_{t+1} = \frac{1}{30} \cdot \sum_{t-30}^{t} y_n$$

بیایید عملکرد مدل moving average را بر روی نمونه‌ها بررسی کنیم:

```
predictions = []
for i in range(len(val_dataset.columns)):
  if i == 0:
    predictions.append(np.mean(train_dataset[train_dataset.columns
    [-30:]].values, axis=1))
  if i < 31 and i > 0:
    predictions.append(0.5 * (np.mean(train_dataset[train_dataset.
    columns[-30+i:]].values, axis=1 + \ np.mean(predictions[:i], axis
    =0)))
  if i > 31:
    predictions.append(np.mean([predictions[:i]], axis=1))

predictions = np.transpose(np.array([row.tolist() for row in
    predictions]))
error_avg = np.linalg.norm(predictions[:3] - val_dataset.values[:3])/
    len(predictions[0])
```

```
pred_1 = predictions[0]
pred_2 = predictions[1]
pred_3 = predictions[2]

fig = make_subplots(rows=3, cols=1)

fig.add_trace(
go.Scatter(x=np.arange(70), mode='lines', y=train_dataset.loc[0].values
    , marker=dict(color="dodgerblue"),
name="Train"),
row=1, col=1
)

fig.add_trace(
go.Scatter(x=np.arange(70, 100), y=val_dataset.loc[0].values, mode='
    lines', marker=dict(color="darkorange"),
name="Val"),
row=1, col=1
)

fig.add_trace(
go.Scatter(x=np.arange(70, 100), y=pred_1, mode='lines', marker=dict(
    color="seagreen"),
name="Pred"),
row=1, col=1
)

fig.add_trace(
```

```
go.Scatter(x=np.arange(70), mode='lines', y=train_dataset.loc[1].values
    , marker=dict(color="dodgerblue"), showlegend=False),
row=2, col=1
)

fig.add_trace(
go.Scatter(x=np.arange(70, 100), y=val_dataset.loc[1].values, mode='
    lines', marker=dict(color="darkorange"), showlegend=False),
row=2, col=1
)

fig.add_trace(
go.Scatter(x=np.arange(70, 100), y=pred_2, mode='lines', marker=dict(
    color="seagreen"), showlegend=False,
name="Denoised signal"),
row=2, col=1
)

fig.add_trace(
go.Scatter(x=np.arange(70), mode='lines', y=train_dataset.loc[2].values
    , marker=dict(color="dodgerblue"), showlegend=False),
row=3, col=1
)

fig.add_trace(
go.Scatter(x=np.arange(70, 100), y=val_dataset.loc[2].values, mode='
    lines', marker=dict(color="darkorange"), showlegend=False),
row=3, col=1
)

fig.add_trace(
go.Scatter(x=np.arange(70, 100), y=pred_3, mode='lines', marker=dict(
    color="seagreen"), showlegend=False,
name="Denoised signal"),
row=3, col=1
)

fig.update_layout(height=1200, width=800, title_text="Moving average")
fig.show()
```



می‌توان دید این مدل بهتر از روش ساده عمل می‌کند و کمتر مستعد پذیرش نوسان‌های روزانه فروش‌ها است. مدل moving average ترندها را با دقت اندکی بیشتر پیدا می‌کند. هر چند توانایی پیدا کردن ترندهای مهم را ندارد.

Holt linear

این روش کاملا متفاوت با دو روش اول است. Holt linear تلاش می‌کند ترندهای سطح بالا را با بهره بردن از یک تابع خطی شناسایی کند. این روش را می‌توان به‌صورت زیر خلاصه سازی کرد:

$$\hat{y}_{t+h} = l_t + h \cdot b_t$$

$$l_t = a \cdot y_t + (1-\alpha) \cdot (l_{t-1} + b_{t-1})$$

$$b_t = \beta \cdot (l_t - l_{t-1}) + (1-\beta) \cdot b_{t-1}$$

در معادله‌های بالا، آلفا و بتا ثابت‌هایی هستند که قابل تغییر است. l_t نشان‌دهنده مقدار سطح و b_t نشان‌دهنده مقدار ترند است. مقدار ترند برابر شیب تابع پیش‌بینی خطی است و مقدار سطح، نقطه تلاقی محور عمودی و تابع پیش‌بینی خطی است. شیب و نقطه تلاقی به‌صورت مداوم با معادله دوم و سوم به‌روز می‌شوند. در پایان، از شیب و نقطه تلاقی برای پیش‌بینی l_{t+1} (در معادله اول) استفاده می‌شود. در این معادله h تعداد گام‌های زمانی بعد از گام زمانی فعلی است.

بیایید عملکرد مدل Holt linear را بر روی نمونه‌ها بررسی کنیم:

```
predictions = []
for row in tqdm(train_dataset[train_dataset.columns[-30:]].values[:3]):
  fit = Holt(row).fit(smoothing_level = 0.3, smoothing_slope = 0.01)
  predictions.append(fit.forecast(30))
predictions = np.array(predictions).reshape((-1, 30))
error_holt = np.linalg.norm(predictions - val_dataset.values[:len(
    predictions)])/len(predictions[0])
```

```
pred_1 = predictions[0]
pred_2 = predictions[1]
pred_3 = predictions[2]

fig = make_subplots(rows=3, cols=1)

fig.add_trace(
go.Scatter(x=np.arange(70), mode='lines', y=train_dataset.loc[0].values
    , marker=dict(color="dodgerblue"),
name="Train"),
row=1, col=1
)

fig.add_trace(
go.Scatter(x=np.arange(70, 100), y=val_dataset.loc[0].values, mode='
    lines', marker=dict(color="darkorange"),
name="Val"),
row=1, col=1
)

fig.add_trace(
go.Scatter(x=np.arange(70, 100), y=pred_1, mode='lines', marker=dict(
    color="seagreen"),
name="Pred"),
row=1, col=1
)

fig.add_trace(
go.Scatter(x=np.arange(70), mode='lines', y=train_dataset.loc[1].values
    , marker=dict(color="dodgerblue"), showlegend=False),
row=2, col=1
)

fig.add_trace(
go.Scatter(x=np.arange(70, 100), y=val_dataset.loc[1].values, mode='
    lines', marker=dict(color="darkorange"), showlegend=False),
row=2, col=1
)

fig.add_trace(
go.Scatter(x=np.arange(70, 100), y=pred_2, mode='lines', marker=dict(
    color="seagreen"), showlegend=False,
```

```
name="Denoised signal"),
row=2, col=1
)

fig.add_trace(
go.Scatter(x=np.arange(70), mode='lines', y=train_dataset.loc[2].values
    , marker=dict(color="dodgerblue"), showlegend=False),
row=3, col=1
)

fig.add_trace(
go.Scatter(x=np.arange(70, 100), y=val_dataset.loc[2].values, mode='
    lines', marker=dict(color="darkorange"), showlegend=False),
row=3, col=1
)

fig.add_trace(
go.Scatter(x=np.arange(70, 100), y=pred_3, mode='lines', marker=dict(
    color="seagreen"), showlegend=False,
name="Denoised signal"),
row=3, col=1
)

fig.update_layout(height=1200, width=800, title_text="Holt linear")
fig.show()
```

می‌توان دید Holt linear ترندهای سطح بالا را همواره پیش‌بینی می‌کند. اما این مدل، توانایی شناسایی نوسان‌های کوتاه‌مدت را همچون روش‌های قبلی ندارد.

Exponential smoothing

در این روش از یک نوع smoothing متفاوت استفاده می‌شود که با average smoothing متفاوت است. گام‌های زمانی قبل از روز هدف را به‌صورت نمایی وزن‌دهی کرده و سپس با هم جمع می‌کنیم تا پیش‌بینی انجام پذیرد. این مدل می‌توان به‌صورت زیر خلاصه‌سازی کرد:

$$\hat{y}_{t+1} = \alpha \cdot y_t + \alpha \cdot (1-\alpha) \cdot y_{t-1} + \alpha \cdot (1-\alpha)^2 \cdot y_{t-2} + \dots$$

$$\hat{y}_{t+1} = \alpha \cdot y_t + (1-\alpha) \cdot \hat{y}_t$$

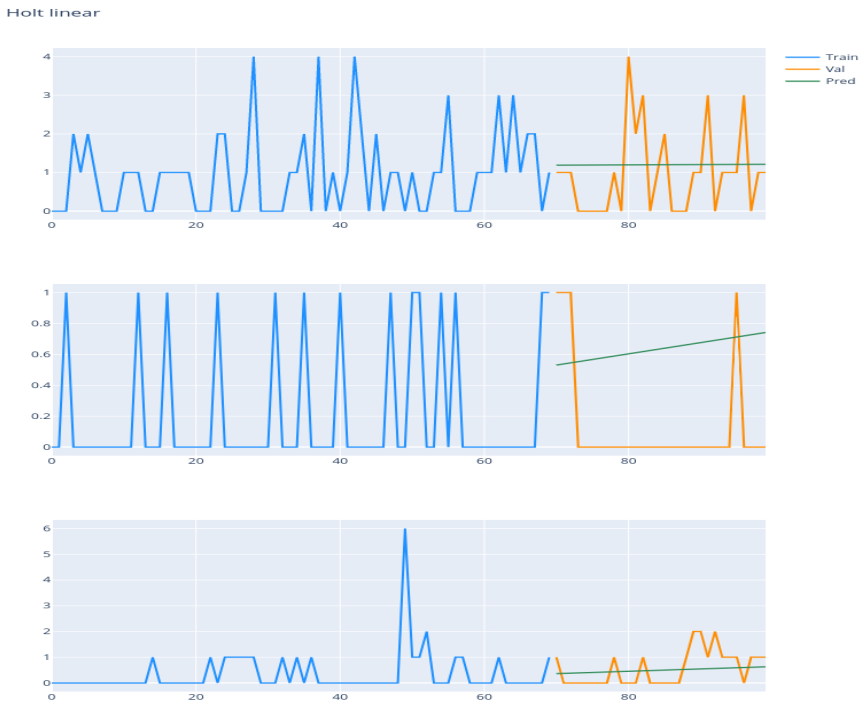
در معادله‌های بالا، آلفا پارامتر smoothing است. پیش‌بینی y_{t+1} یک میانگین از بررسی‌های سری زمانی است.

نرخ زوال وزن‌ها با آلفا کنترل می‌شود. این روش وزن‌های متفاوتی به‌هر گام زمانی اختصاص می‌دهد (بر عکس روش میانگین). این نوع وزن‌دهی به‌ما اطمینان می‌دهد که فروش‌های اخیر اهمیت بیشتری نسبت به فروش‌های قدیمی داشته باشند.
حال بیایید عملکرد این مدل را بررسی کنیم:

```
predictions = []
for row in tqdm(train_dataset[train_dataset.columns[-30:]].values[:3]):
  fit = ExponentialSmoothing(row, seasonal_periods=3).fit()
  predictions.append(fit.forecast(30))
predictions = np.array(predictions).reshape((-1, 30))
error_exponential = np.linalg.norm(predictions[:3] - val_dataset.values
    [:3])/len(predictions[0])
```

```
pred_1 = predictions[0]
pred_2 = predictions[1]
pred_3 = predictions[2]

fig = make_subplots(rows=3, cols=1)

fig.add_trace(
go.Scatter(x=np.arange(70), mode='lines', y=train_dataset.loc[0].values
    , marker=dict(color="dodgerblue"),
name="Train"),
row=1, col=1
)

fig.add_trace(
go.Scatter(x=np.arange(70, 100), y=val_dataset.loc[0].values, mode='
    lines', marker=dict(color="darkorange"),
name="Val"),
row=1, col=1
)

fig.add_trace(
go.Scatter(x=np.arange(70, 100), y=pred_1, mode='lines', marker=dict(
    color="seagreen"),
name="Pred"),
row=1, col=1
)

fig.add_trace(
go.Scatter(x=np.arange(70), mode='lines', y=train_dataset.loc[1].values
    , marker=dict(color="dodgerblue"), showlegend=False),
row=2, col=1
)

fig.add_trace(
go.Scatter(x=np.arange(70, 100), y=val_dataset.loc[1].values, mode='
    lines', marker=dict(color="darkorange"), showlegend=False),
row=2, col=1
)

fig.add_trace(
go.Scatter(x=np.arange(70, 100), y=pred_2, mode='lines', marker=dict(
    color="seagreen"), showlegend=False,
name="Denoised signal"),
row=2, col=1
```

```
)

fig.add_trace(
go.Scatter(x=np.arange(70), mode='lines', y=train_dataset.loc[2].values
    , marker=dict(color="dodgerblue"), showlegend=False),
row=3, col=1
)

fig.add_trace(
go.Scatter(x=np.arange(70, 100), y=val_dataset.loc[2].values, mode='
    lines', marker=dict(color="darkorange"), showlegend=False),
row=3, col=1
)

fig.add_trace(
go.Scatter(x=np.arange(70, 100), y=pred_3, mode='lines', marker=dict(
    color="seagreen"), showlegend=False,
name="Denoised signal"),
row=3, col=1
)

fig.update_layout(height=1200, width=800, title_text="Exponential
    smoothing")
fig.show()
```



مدل‌های ARIMA[۳]

ARIMA در واقع یک کلاس از مدل‌هاست که یک سری زمانی را بر اساس مقادیر قبلی خود تشریح می‌کند. هر سری زمانی که الگویی داشته باشد و white noise تصادفی نباشد با ARIMA قابل مدل شدن است. هر مدل ARIMA با سه شرط مشخص می‌شود:

- p is the order of the AR term
- q is the order of the MA term
- d is the number of differencing required to make the time series stationary

اولین گام برای ساختن یک مدل ARIMA، ایستا کردن سری زمانی است. ar به‌معنی auto regressive می‌باشد و نشان می‌دهد که یک مدل رگرسیون خطی است که از خطای خودش برای پیش‌بینی استفاده می‌کند. همانطور که می‌دانید، رگرسیون خطی وقتی predictor-ها به‌هم مرتبط نیستند بهتر عمل می‌کند. رویکرد اصلی کم کردن مقدار قبلی از مقدار فعلی‌ست. با توجه به پیچیدگی سری زمانی، شاید اختلاف‌های بیشتری مورد نیاز باشد. مقدار d در این صورت برابر: کمینه تعداد اختلاف‌های مورد نیاز برای ایستا کردن سری زمانی است. اگر سری زمانی ایستا باشد، مقدار d برابر صفر خواهد بود. p برابر تعداد lag-های Y است که به‌عنوان predictor استفاده خواهد شد. q برابر تعداد خطاهای پیش‌بینی lag-دار است که باید وارد مدل ARIMA شود. یک مدل ARIMA مدلی‌ست که سری زمانی در آن حداقل یک بار اختلاف‌گیری شده تا به‌وضعیت ایستا برسد. اگر ترم‌های AR و MA را ترکیب کنیم به معادله زیر می‌رسیم.

$$Y_t = \alpha + \beta_1 Y_{t-1} + \beta_2 Y_{t-2} + .. + \beta_p Y_{t-p} \epsilon_t + \phi_1 \epsilon_{t-1} + \phi_2 \epsilon_{t-2} + .. + \phi_q \epsilon_{t-q}$$

ARIMA model in words:
Predicted Yt = Constant + Linear combination Lags of Y (upto p lags) + Linear Combination of Lagged forecast errors (upto q lags)

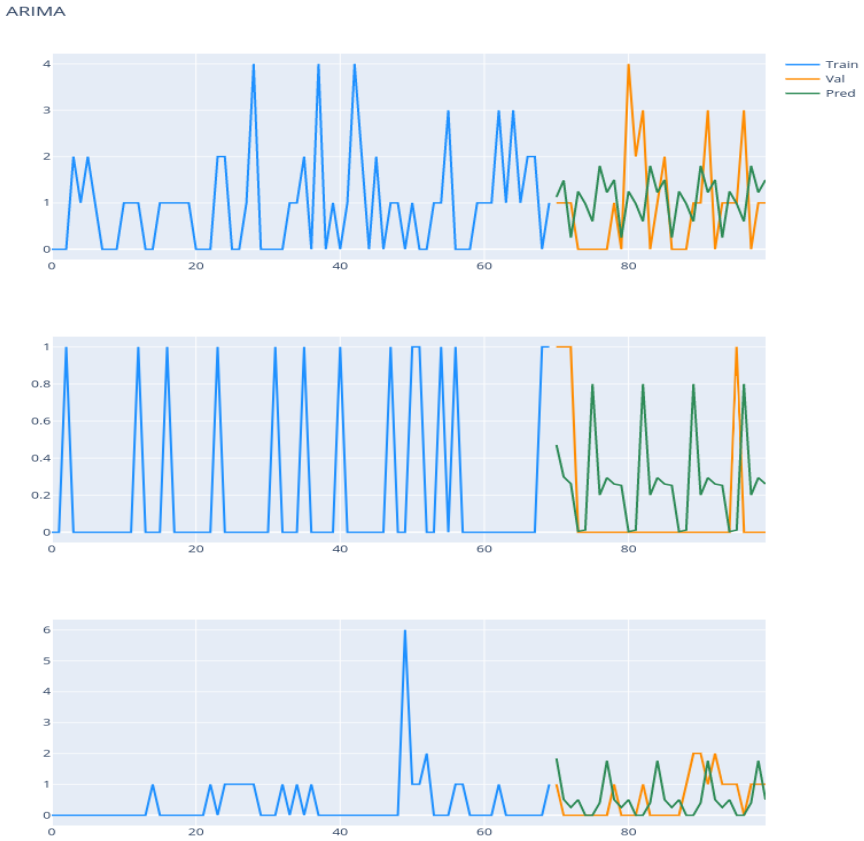
حال بیایید عملکرد این مدل را بررسی کنیم:

```
predictions = []
for row in tqdm(train_dataset[train_dataset.columns[-30:]].values[:3]):
  fit = sm.tsa.statespace.SARIMAX(row, seasonal_order=(0, 1, 1, 7)).fit
    ()
  predictions.append(fit.forecast(30))
predictions = np.array(predictions).reshape((-1, 30))
error_arima = np.linalg.norm(predictions[:3] - val_dataset.values[:3])/
    len(predictions[0])
```

[۳] Auto Regressive Integrated Moving Average

```
pred_1 = predictions[0]
pred_2 = predictions[1]
pred_3 = predictions[2]

fig = make_subplots(rows=3, cols=1)

fig.add_trace(
go.Scatter(x=np.arange(70), mode='lines', y=train_dataset.loc[0].values
    , marker=dict(color="dodgerblue"),
name="Train"),
row=1, col=1
)

fig.add_trace(
go.Scatter(x=np.arange(70, 100), y=val_dataset.loc[0].values, mode='
    lines', marker=dict(color="darkorange"),
name="Val"),
row=1, col=1
)

fig.add_trace(
go.Scatter(x=np.arange(70, 100), y=pred_1, mode='lines', marker=dict(
    color="seagreen"),
name="Pred"),
row=1, col=1
)

fig.add_trace(
go.Scatter(x=np.arange(70), mode='lines', y=train_dataset.loc[1].values
    , marker=dict(color="dodgerblue"), showlegend=False),
row=2, col=1
)

fig.add_trace(
go.Scatter(x=np.arange(70, 100), y=val_dataset.loc[1].values, mode='
    lines', marker=dict(color="darkorange"), showlegend=False),
row=2, col=1
)

fig.add_trace(
go.Scatter(x=np.arange(70, 100), y=pred_2, mode='lines', marker=dict(
    color="seagreen"), showlegend=False,
name="Denoised signal"),
row=2, col=1
)

fig.add_trace(
go.Scatter(x=np.arange(70), mode='lines', y=train_dataset.loc[2].values
    , marker=dict(color="dodgerblue"), showlegend=False),
row=3, col=1
)

fig.add_trace(
go.Scatter(x=np.arange(70, 100), y=val_dataset.loc[2].values, mode='
    lines', marker=dict(color="darkorange"), showlegend=False),
row=3, col=1
)
```

```
50
51 fig.add_trace(
52 go.Scatter(x=np.arange(70, 100), y=pred_3, mode='lines', marker=dict(
       color="seagreen"), showlegend=False,
53 name="Denoised signal"),
54 row=3, col=1
55 )
56
57 fig.update_layout(height=1200, width=800, title_text="ARIMA")
58 fig.show()
```



References

[1] Time Series ARIMA Models

[2] M5 Forecasting - Starter Data Exploration

[3] How to Create an ARIMA Model for Time Series Forecasting in Python

[4] 7 methods to perform Time Series forecasting (with Python codes)

[5] Economics for the IB Diploma