Project 4: Time Series Forecasting

Elvessa Tatum

Problem Statement

- In this project, we aim to forecast future retail sales using historical sales data from product families sold at Favorita stores located in Ecuador.
- Accurately predicting future sales is critical for effective inventory management, staffing, marketing campaigns, and strategic planning.
- By applying time series forecasting models such as SARIMA, Prophet, and simple moving averages, we seek to identify patterns, seasonality, and trends that influence sales behavior over time.
- The goal is to build robust models that minimize forecasting error and provide actionable insights for business decision-making.

About the Data

- The dataset for this project comes from the Kaggle "Store Sales" competition. It contains detailed sales information from a major Ecuadorian grocery retailer across multiple stores and product categories.
- Key features include:
- date: The date of the sales transaction.
- store_nbr: Unique identifier for each store.
- family: Product category (e.g., beverages, dairy, produce).
- sales: Total sales amount for the given date, store, and product family.
- onpromotion: Number of products on promotion on that date.
- The data spans several years and captures important sales patterns such as:
- Seasonality (weekly and yearly shopping cycles), Holidays and promotions effects, Store-specific trends
- For this project, we focus primarily on aggregated daily sales across all stores and product families to model overall retail sales trends and forecast future performance.

```
df.head()
```

	id	date	store_nbr	family	sales	onpromotion
0	0	2013-01-01	1	AUTOMOTIVE	0.0	0
1	1	2013-01-01	1	BABY CARE	0.0	0
2	2	2013-01-01	1	BEAUTY	0.0	0
3	3	2013-01-01	1	BEVERAGES	0.0	0
4	4	2013-01-01	1	BOOKS	0.0	0

```
df.info()
df.describe()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3000888 entries, 0 to 3000887
Data columns (total 6 columns):
 #   Column       Dtype
---  ------       -----
 0   id           int64
 1   date         datetime64[ns]
 2   store_nbr    int64
 3   family       object
 4   sales        float64
 5   onpromotion  int64
dtypes: datetime64[ns](1), float64(1), int64(3), object(1)
memory usage: 137.4+ MB
```

	id	date	store_nbr	sales	onpromotion
count	3.000888e+06	3000888	3.000888e+06	3.000888e+06	3.000888e+06
mean	1.500444e+06	2015-04-24 08:27:04.703088384	2.750000e+01	3.577757e+02	2.602770e+00
min	0.000000e+00	2013-01-01 00:00:00	1.000000e+00	0.000000e+00	0.000000e+00
25%	7.502218e+05	2014-02-26 18:00:00	1.400000e+01	0.000000e+00	0.000000e+00
50%	1.500444e+06	2015-04-24 12:00:00	2.750000e+01	1.100000e+01	0.000000e+00
75%	2.250665e+06	2016-06-19 06:00:00	4.100000e+01	1.958473e+02	0.000000e+00
max	3.000887e+06	2017-08-15 00:00:00	5.400000e+01	1.247170e+05	7.410000e+02
std	8.662819e+05	NaN	1.558579e+01	1.101998e+03	1.221888e+01

Importing Data

Let’s start by importing the data and taking a overview at it.

Hypothesis

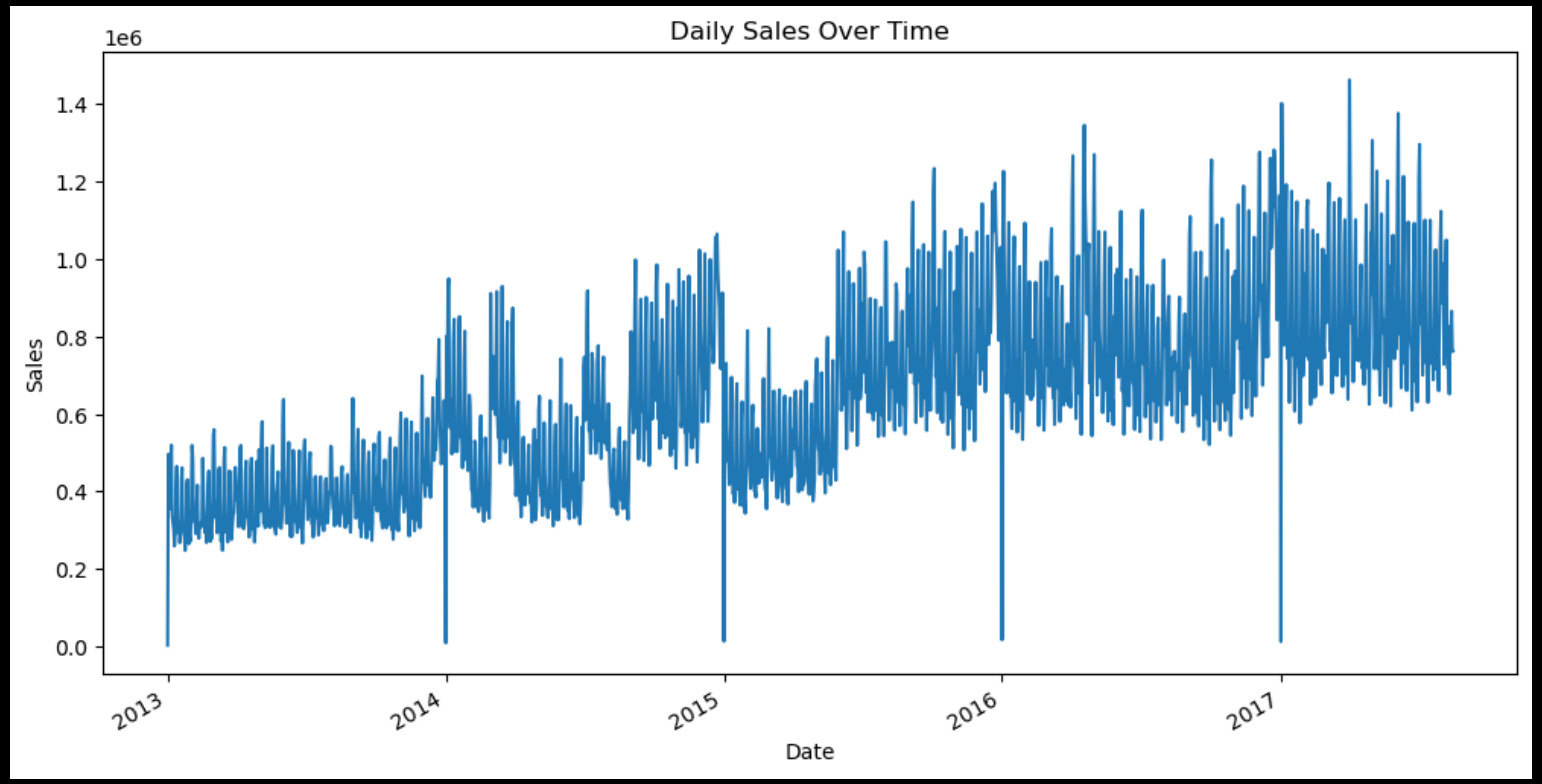
- I predict that more people will be buying on the weekends rather than the weekdays.
- This means more buys on Sunday and Saturday compared to the other 5 days of the week.

Pre-Processing Data

- After seeing that there are no missing values, I decided to move on and extract time based features.

```
# check for missing values
print(df.isnull().sum())
```

```
id              0
date            0
store_nbr       0
family          0
sales           0
onpromotion     0
dtype: int64
```



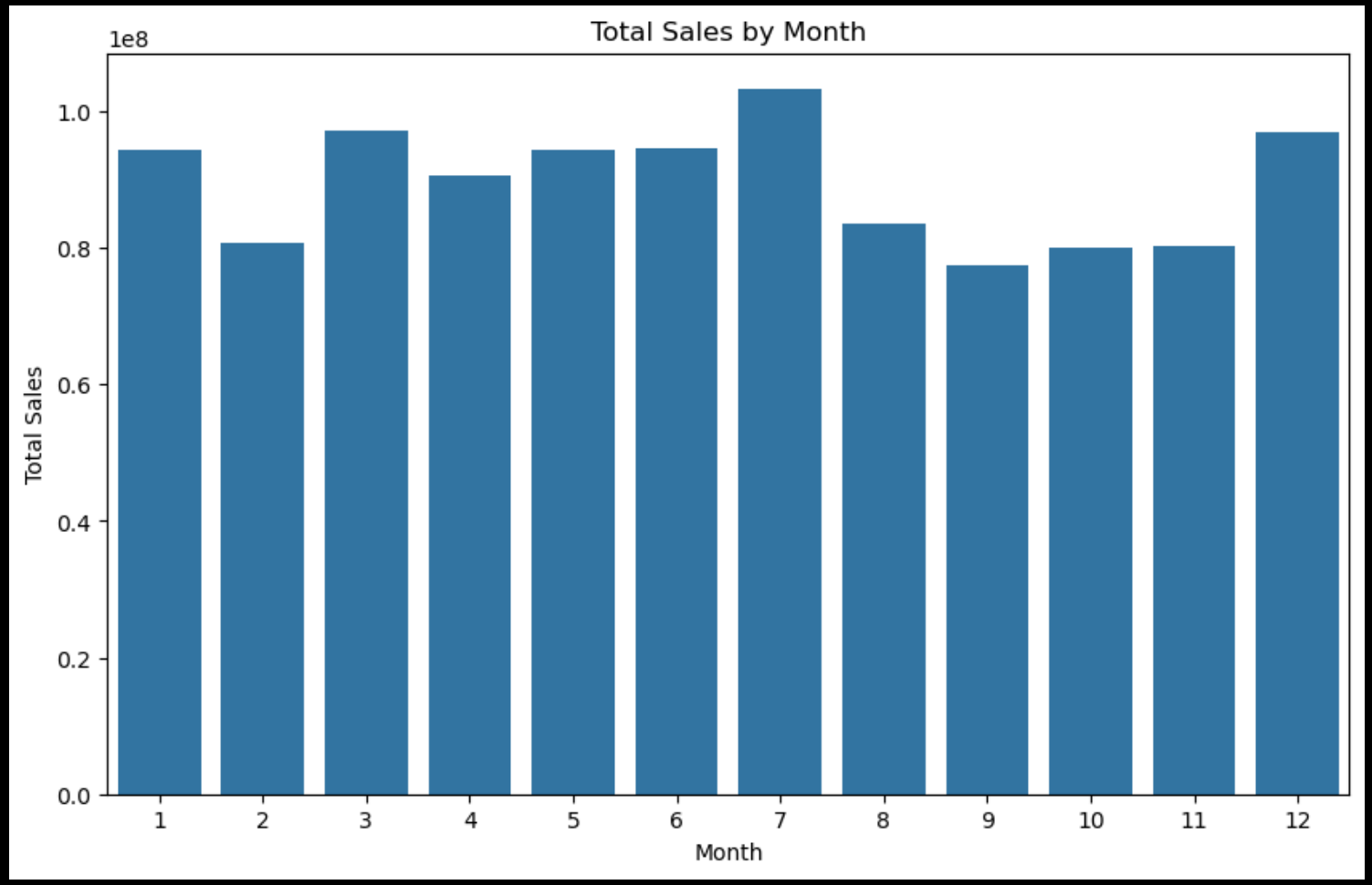
Data Exploration

- Let’s take a look at the data a bit more. First, let’s take a look at the daily sales over time.
- Clearly, the sales have gone up over time, reaching a peak in 2017.

Data Exploration continued

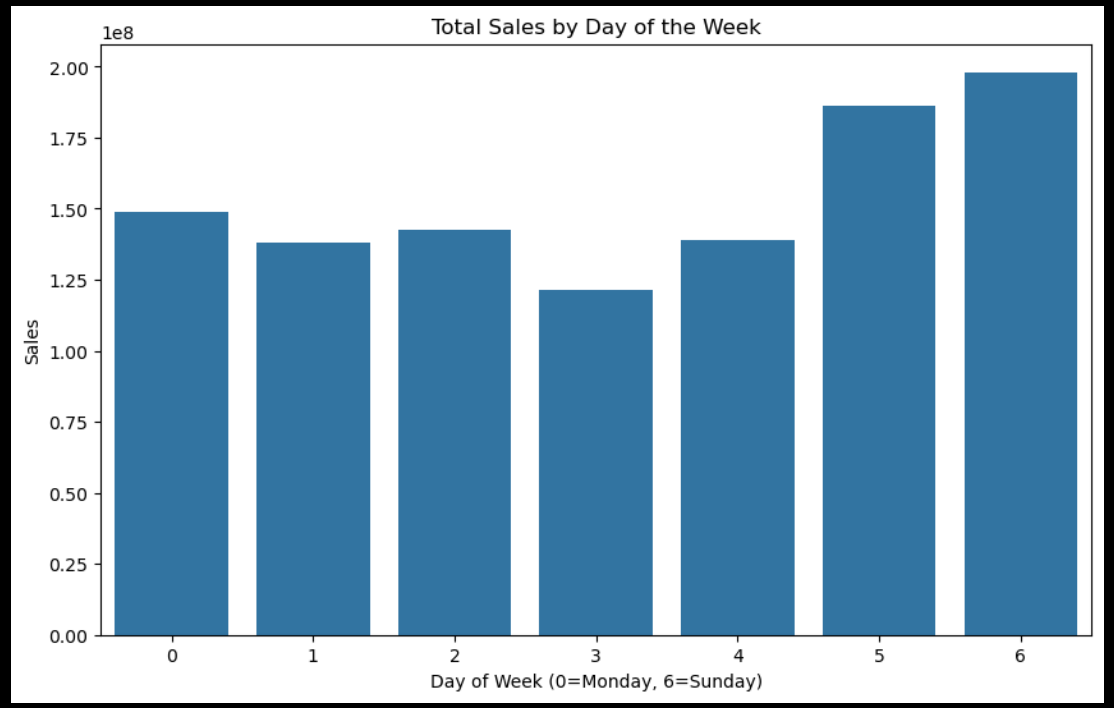
- Next, let's check the total sales by month

- Clearly, July is the highest, followed by December. Overall, the highest time period for sales seems to be in Spring and Summer (March-July)



Data Exploration continued

- Next, let's see if my hypothesis was correct with the sales by days of the week.
- Clearly, it was. The sales on Saturday and Sunday outdid the sales throughout the rest of the week.



Creating our SARIMA Model

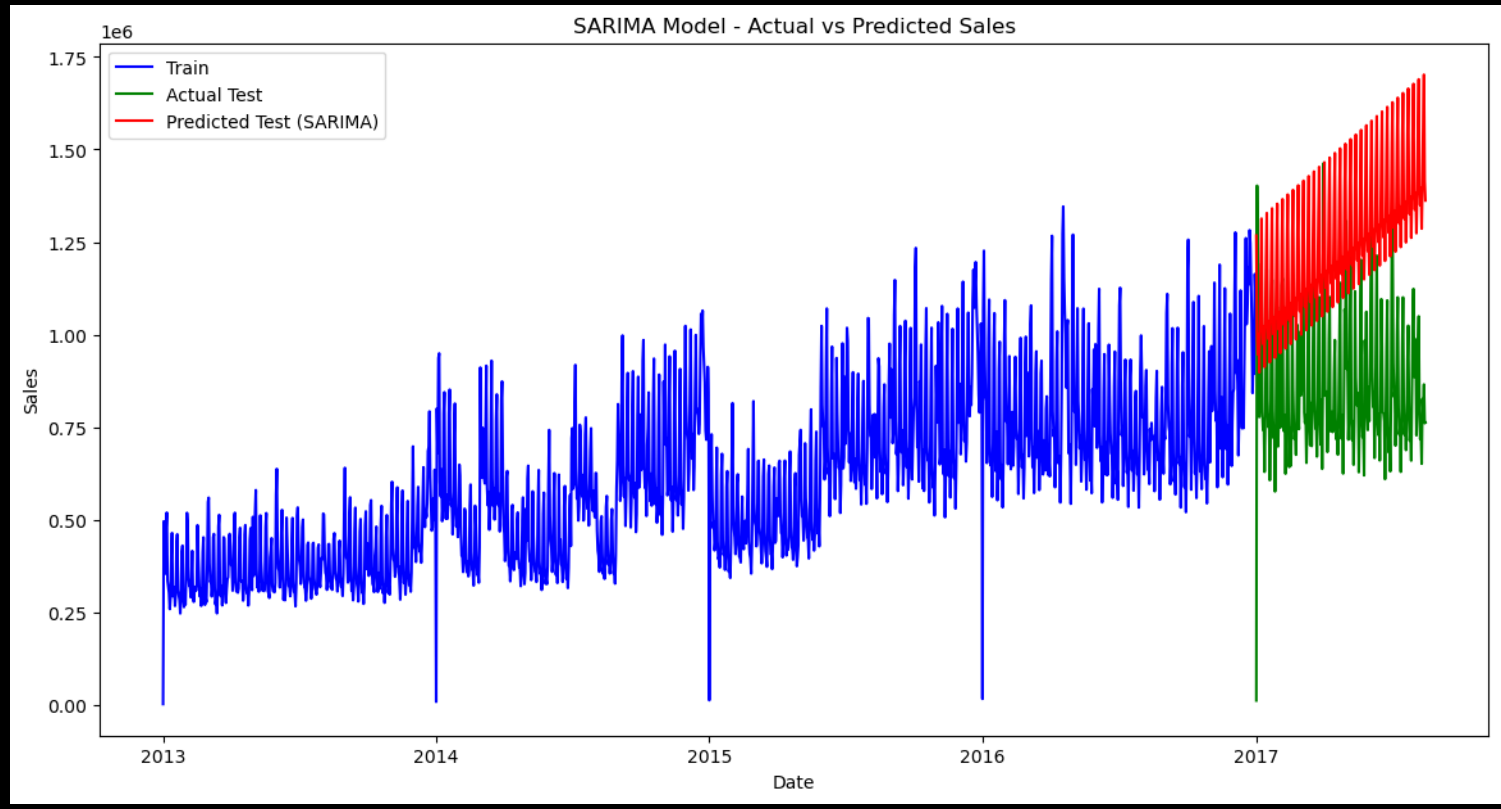
- Now that we've explored the data, let's make the SARIMA Model and view it.

```
# define cutoff date for training and testing, and make training and testing sets
cutoff_date = '2017-01-01'
train = df[df['date'] < cutoff_date]
test = df[df['date'] >= cutoff_date]
```

```
from statsmodels.tsa.statespace.sarimax import SARIMAX

# aggregate sales by date
train_sales = train.groupby('date')['sales'].sum()

# set frequency to daily
train_sales = train_sales.asfreq('D')

# now fit SARIMA model
model = SARIMAX(train_sales, order=(1, 1, 1), seasonal_order=(1, 1, 1, 7))
results = model.fit()

# forecast
forecast = results.get_forecast(steps=len(test['date'].unique()))
predicted_mean = forecast.predicted_mean
```

Creating our SARIMA Model Continued

- Our Model predicted for sales to be way higher than they ended up being for the test sales.
- This isn't good, as the model should be more accurate to the testing set.



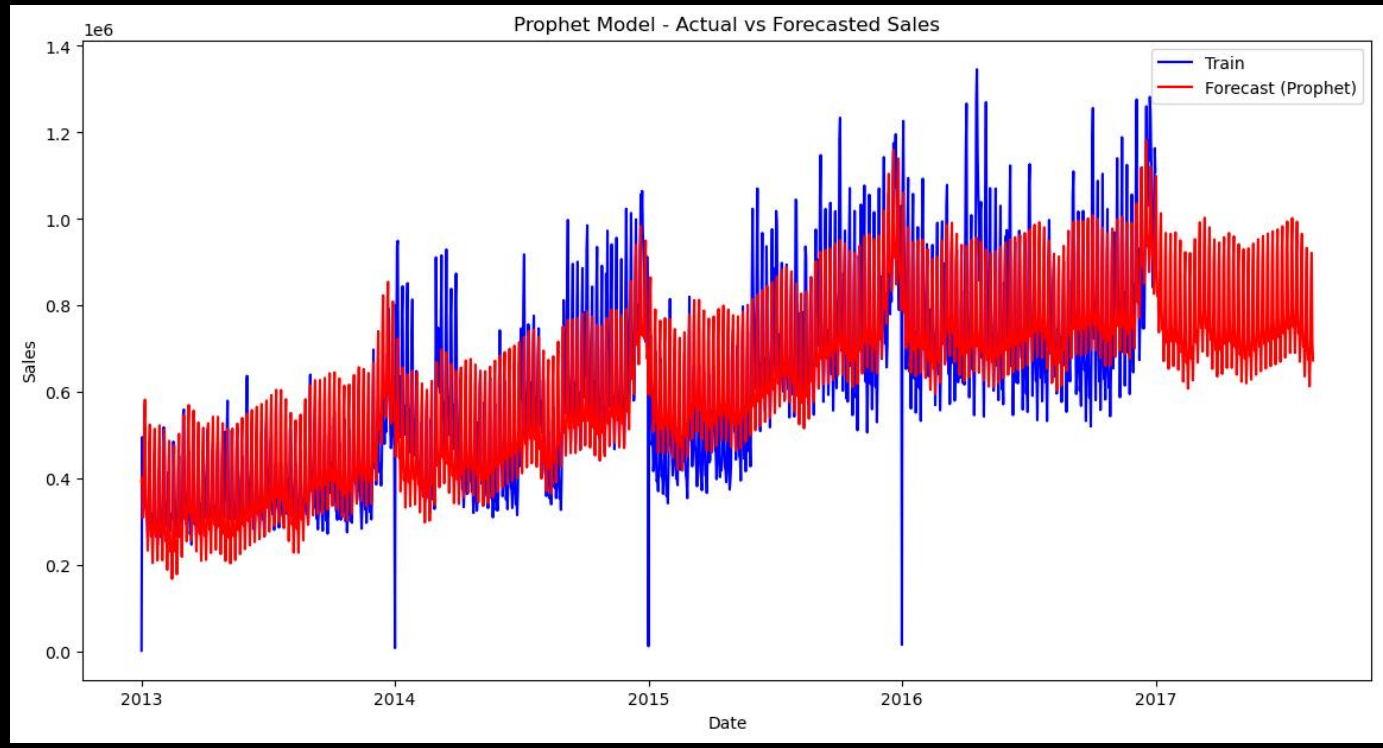
```python
from prophet import Prophet

# prepare data for Prophet
prophet_df = train.groupby('date')['sales'].sum().reset_index()
prophet_df.columns = ['ds', 'y']

# initialize and fit Prophet model
prophet_model = Prophet()
prophet_model.fit(prophet_df)

# create future dataframe
future = prophet_model.make_future_dataframe(periods=len(test['date'].unique()))
forecast = prophet_model.predict(future)
```

Creating Prophet Model

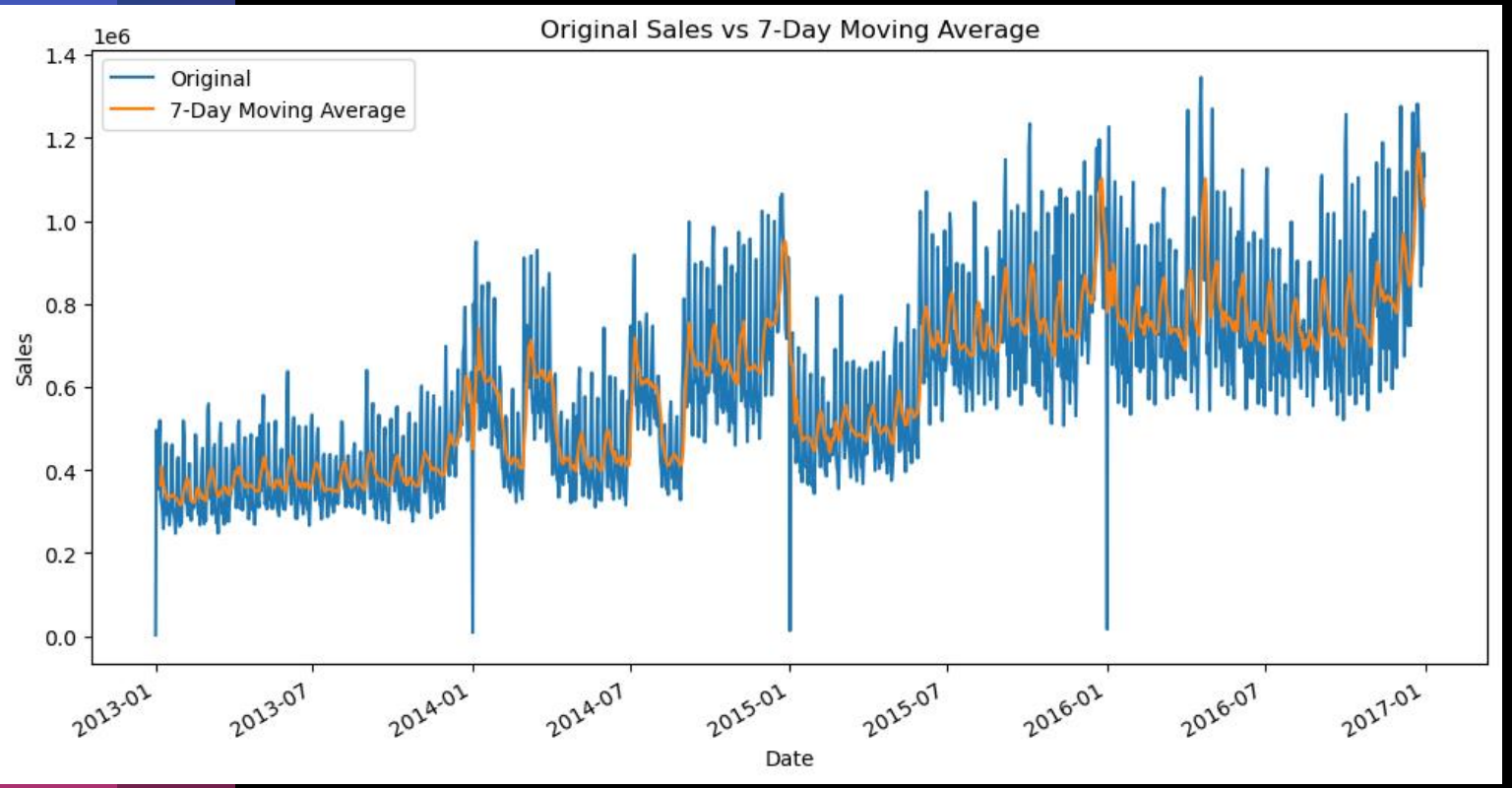
- Next, let's make the Prophet Model.

Creating Prophet Model Continued

- Let’s take a look at the Prophet Model.
- This model did much better, landing fairly close to the training data, and also having forecasts similar to the actual testing data we saw in the previous graph.



Moving averages



- Finally, let's take a look at the 7 day moving averages.
- From here, we can see a general upward trend, reaching over a million dollars in sales.
- However, there are some dips, particularly in January each year. This could just be seasonal slowdowns that happen each year.

Evaluating models

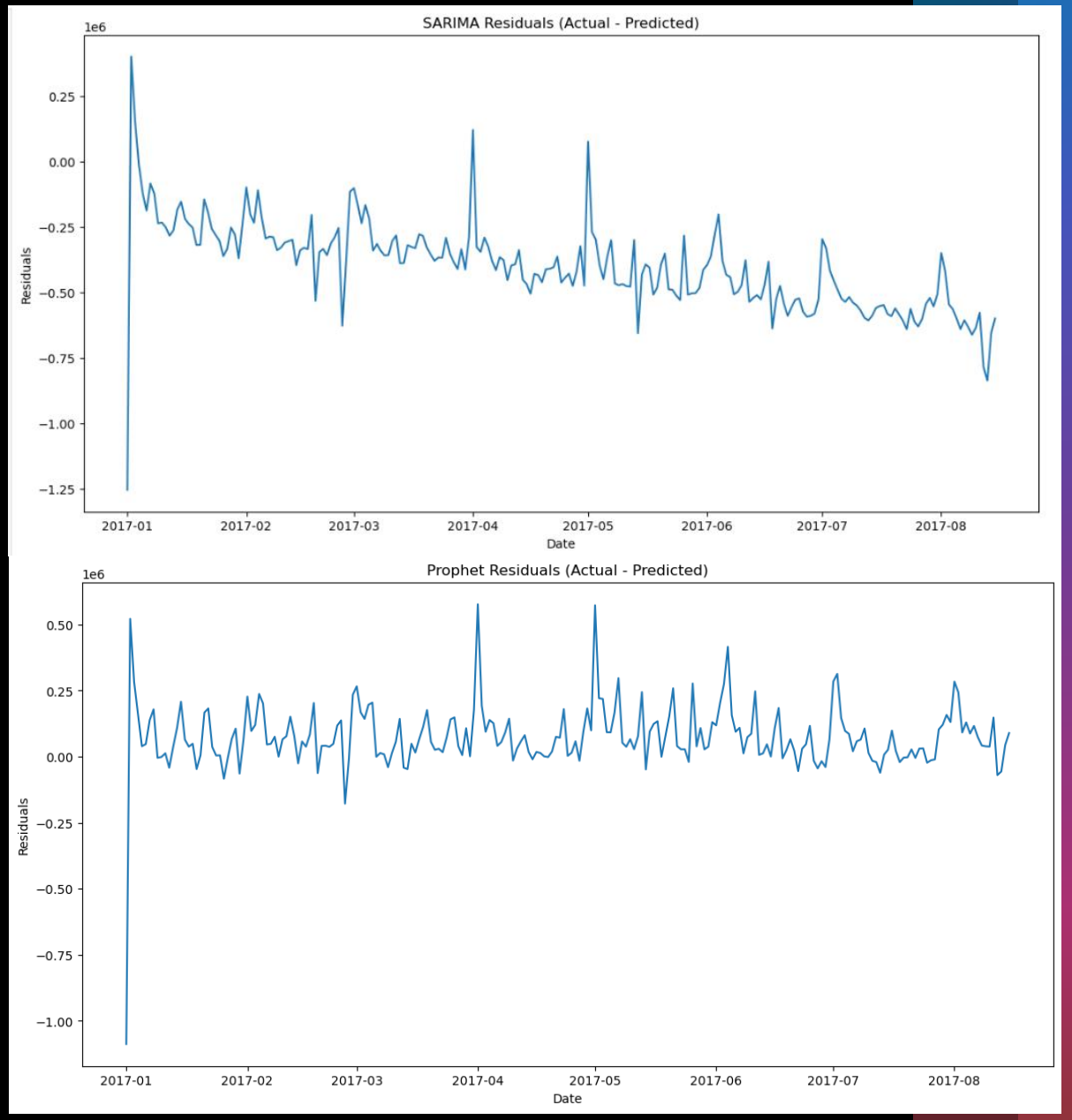
```
SARIMA RMSE: 187864310676.90134
SARIMA MAE: 404867.8403458795
SARIMA MAPE: 94.98297005595363%
```

```
Prophet RMSE: 22334269406.310436
Prophet MAE: 96327.96515666434
Prophet MAPE: 49.223125930579535%
```

- Let's evaluate the models now, first the SARIMA model then the Prophet model. We'll do this by calculating RMSE, MAE, and MAPE for each model using the test set.
- Here, Prophet has MUCH lower RMSE. This means that Prophet predicts closer on average.
- Again, Prophet has much lower MAE. This means that Prophet has smaller average errors.
- Finally, SARIMA had a 94% MAPE, while Prophet had s 49% MAPE.
- Clearly, Prophet was the better model by far.

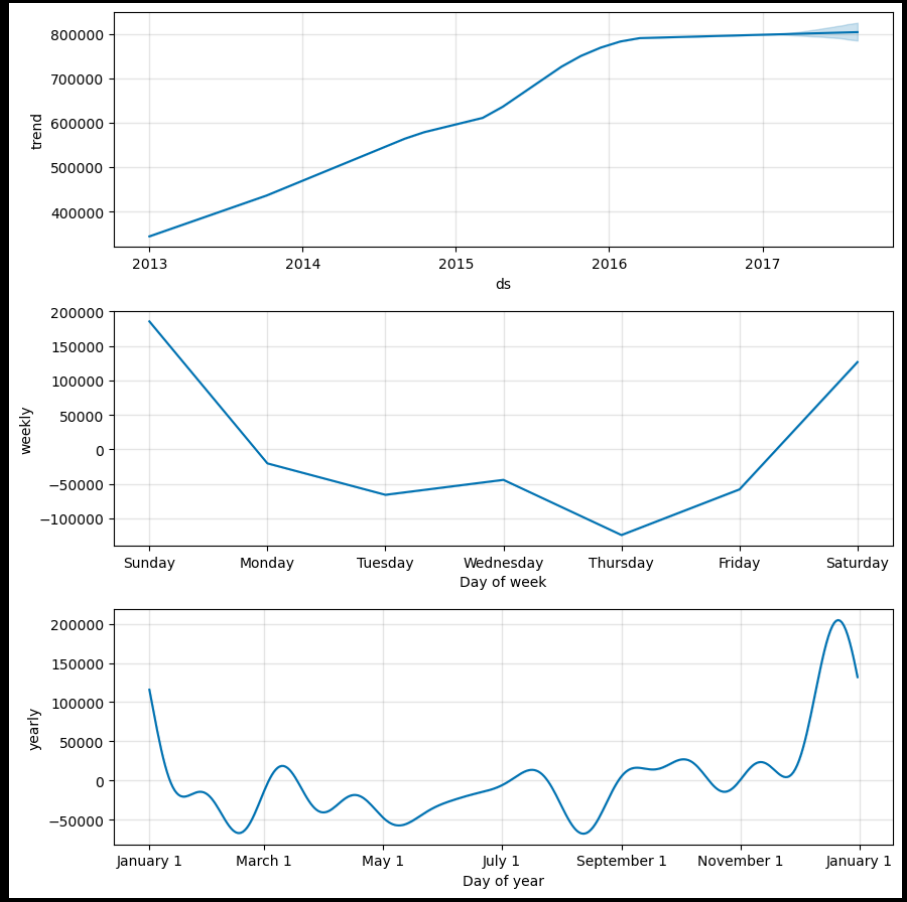
Evaluating Models Continued

- Now, we'll take a look at the residuals for the two models
- Prophet had some spikes, but overall wasn't too bad.
- Meanwhile, SARIMA was constantly going down, meaning it consistently overestimated the actual sales values



Evaluating Models Continued

- Finally, let's look at the seasonality for the prophet model.
- The lowest day of the week seems to be Thursday, while the best day is Sunday.
- Meanwhile, The worst month seems to be September, which is supported by the total sales by month graph we saw earlier.



Conclusion

- After training and evaluation, the Prophet model significantly outperformed the SARIMA model across all evaluation metrics:
- Prophet achieved a lower RMSE, MAE, and MAPE compared to SARIMA.
- The SARIMA model showed high forecasting errors, with a MAPE of approximately 95%, indicating it struggled to capture the underlying patterns in the sales data.
- In contrast, the Prophet model achieved a MAPE of approximately 49%, suggesting much better predictive performance, although there is still room for improvement.
- We were also able to see that the best day was on Sunday, while the best month was July. This information can be used to make more sales around those time periods.