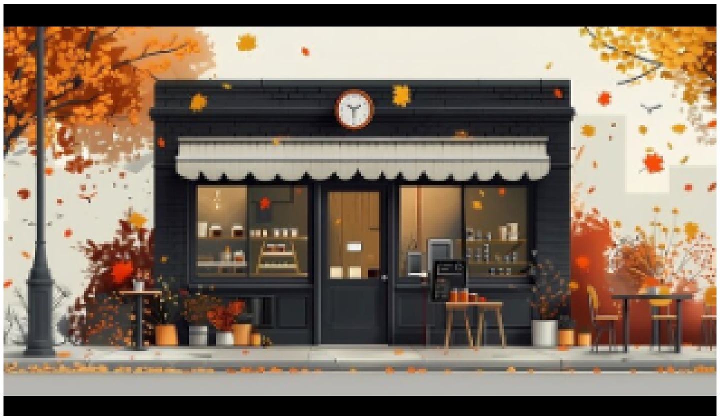
Capstone Three: Final Report



Introduction

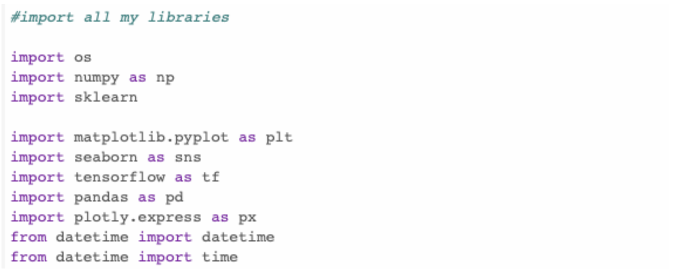
Coffee can make or break a morning. The roast and refreshing sip of a hot cup of coffee can set the tone for the rest of your day. This morning's cup inspired me to analyze some coffee data. I decided to use data from Kaggle on the sales of different coffee shops. From there, I would wrangle, explore, and predict sales for the coffee shop business. My goal was to wrangle the information of coffee shops so that I can optimize their menus to balance profitability, improve revenue, and meet customer demand. I will provide engaging visuals and analyze ways to expand the business. Now, we will go through the process of organizing, analyzing, and modeling a multivariate time series DataFrame. This will assist with proposals to improve business and understand consistent sales patterns that the coffee shop has.

Data: https://www.kaggle.com/code/ahmedabbas757/coffee-shop-sales/input

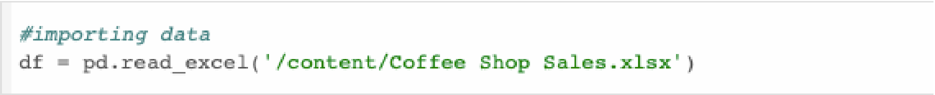
Table of Contents

1. Loading Data

When I got the idea to analyze a coffee shop to propose improvement to their business, I roamed through Kaggle trying to find the best data for my proposal. I found a dataset that had transactions data over a span of 7 months for multiple coffee ship and the data ranged from



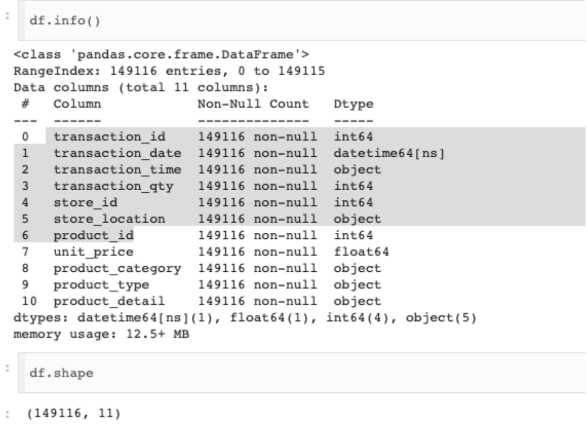
```
#import all my libraries

import os
import numpy as np
import sklearn

import matplotlib.pyplot as plt
import seaborn as sns
import tensorflow as tf
import pandas as pd
import plotly.express as px
from datetime import datetime
from datetime import time
```



```
#importing data
df = pd.read_excel('/content/Coffee Shop Sales.xlsx')
```

2. Wrangle the Data

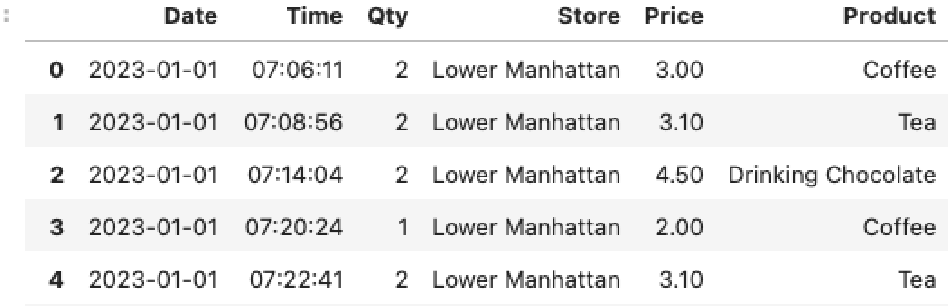
Now that the data is loaded, it's time to analyze it to determine how we'll approach it. Within the data, there are multiple columns that won't be needed, we need to check for any missing data within the columns and inspect the shape of the data. We will be streamlining the data within the DataFrame to explore and later provide some forecasting and modeling.



```
:  df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 149116 entries, 0 to 149115
Data columns (total 11 columns):
 #   Column            Non-Null Count   Dtype
---  ------            --------------   -----
 0   transaction_id    149116 non-null  int64
 1   transaction_date  149116 non-null  datetime64[ns]
 2   transaction_time  149116 non-null  object
 3   transaction_qty   149116 non-null  int64
 4   store_id          149116 non-null  int64
 5   store_location    149116 non-null  object
 6   product_id        149116 non-null  int64
 7   unit_price        149116 non-null  float64
 8   product_category  149116 non-null  object
 9   product_type      149116 non-null  object
 10  product_detail    149116 non-null  object
dtypes: datetime64[ns](1), float64(1), int64(4), object(5)
memory usage: 12.5+ MB

:  df.shape

:  (149116, 11)
```

Through this process, I determined that it was important to analyze a single shop before comparing the shops to each other. I come from the theory that I need to clean up my house first before I start comparing my house to another, so we will do that with this DataFrame. Now we will rename and drop the data that is no longer needed.

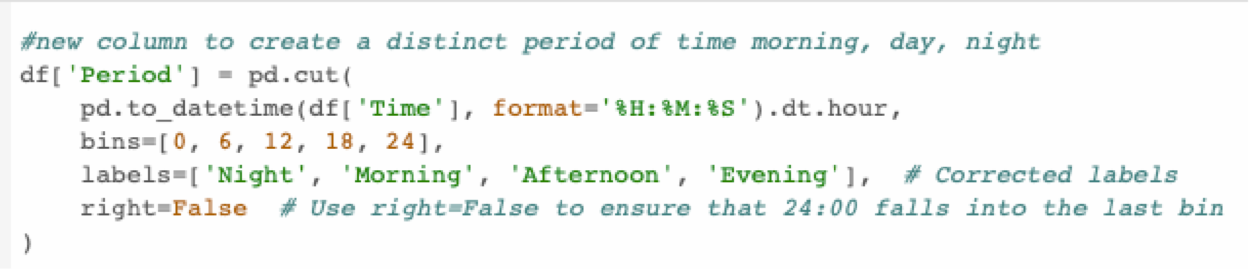
Renamed: 'unit_price': 'Price','transaction_qty': 'Qty','transaction_date': 'Date', 'transaction_time': 'Time', 'store_location': 'Store','product_category': 'Product'
Dropped: 'product_type', 'transaction_id','store_id', 'product_id', 'product_detail'



	Date	Time	Qty	Store	Price	Product
0	2023-01-01	07:06:11	2	Lower Manhattan	3.00	Coffee
1	2023-01-01	07:08:56	2	Lower Manhattan	3.10	Tea
2	2023-01-01	07:14:04	2	Lower Manhattan	4.50	Drinking Chocolate
3	2023-01-01	07:20:24	1	Lower Manhattan	2.00	Coffee
4	2023-01-01	07:22:41	2	Lower Manhattan	3.10	Tea

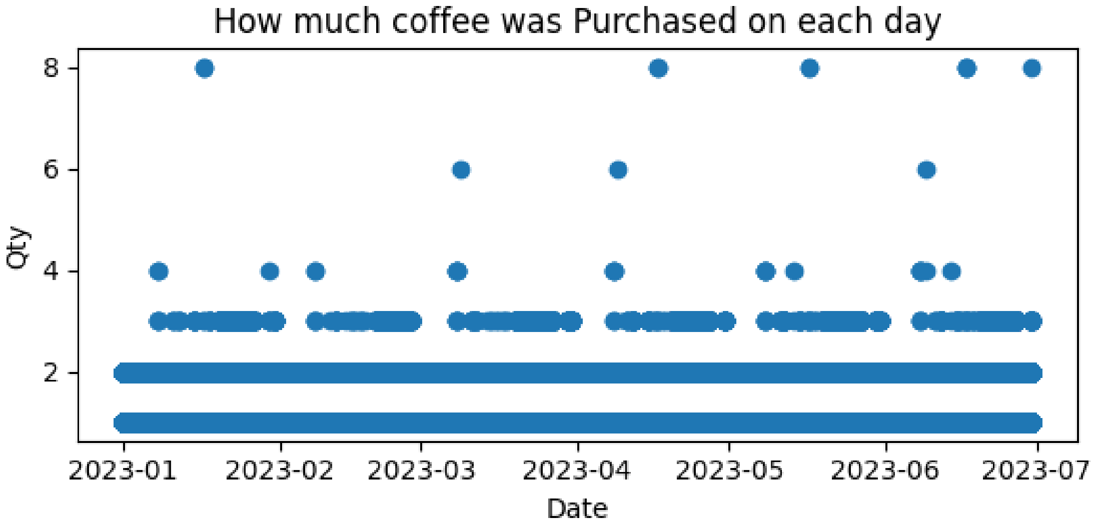
3. EDA

Now that the DataFrame is truly tailored, it's time to explore all the interesting time frames of each different transactions in the DataFrame. This would include comparing morning to evening and weekend vs weekday. This would allow for a full understanding of when sales are most popular for the coffee shops in New York City.

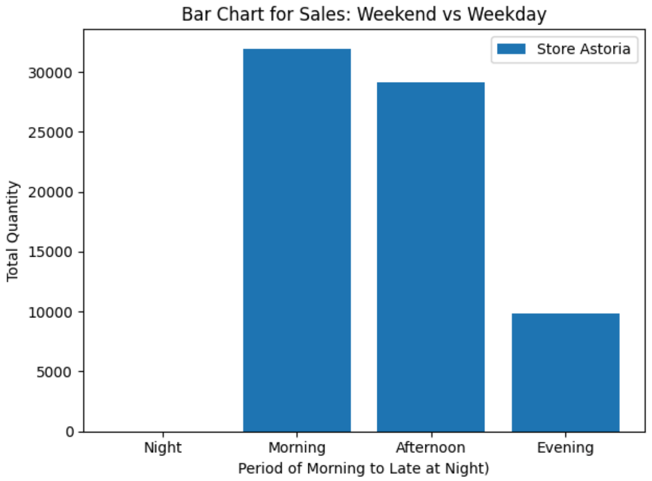


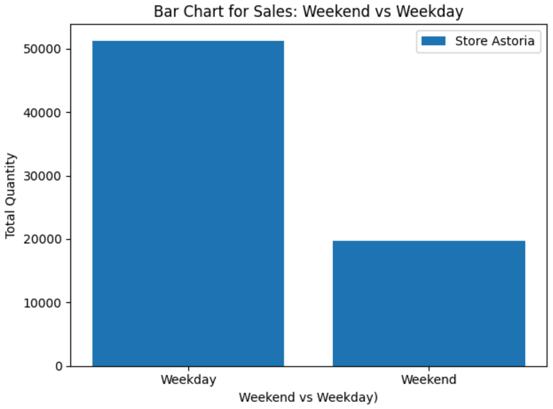
```
#new column to create a distinct period of time morning, day, night
df['Period'] = pd.cut(
    pd.to_datetime(df['Time'], format='%H:%M:%S').dt.hour,
    bins=[0, 6, 12, 18, 24],
    labels=['Night', 'Morning', 'Afternoon', 'Evening'],  # Corrected labels
    right=False  # Use right=False to ensure that 24:00 falls into the last bin
)
```

The DataFrame has more organized columns to analyze and we've inspected duplicates and verified that there weren't any true duplicates. Now that I have some columns formed analyzing different portions of the days / weeks sales, we will get into some visualization on how the sales are.



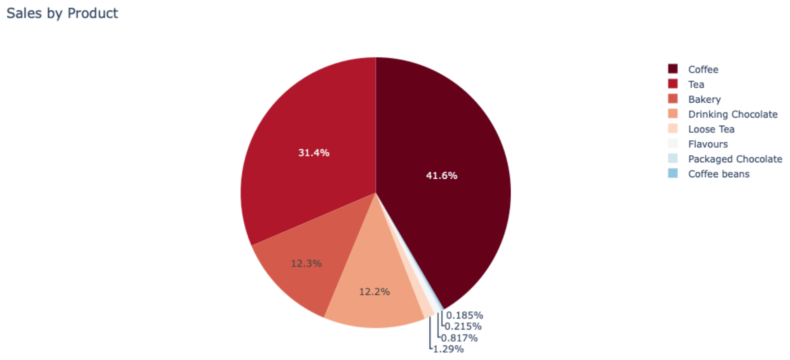
The coffee shops show that there are consistent sales on a regular basis. We can now analyze what time of day most of the sales occur. Knowing the consistent sales, we can narrow down our data to the singular coffee shop in Astoria. Furthermore, we see that people really need their fix on weekdays compared to the weekends.



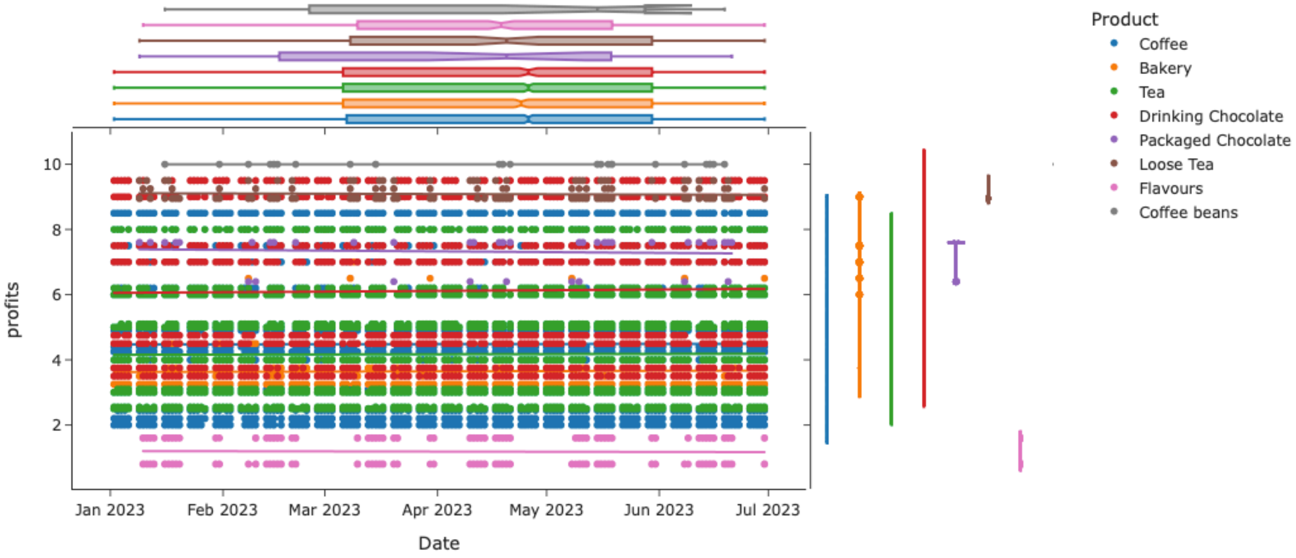


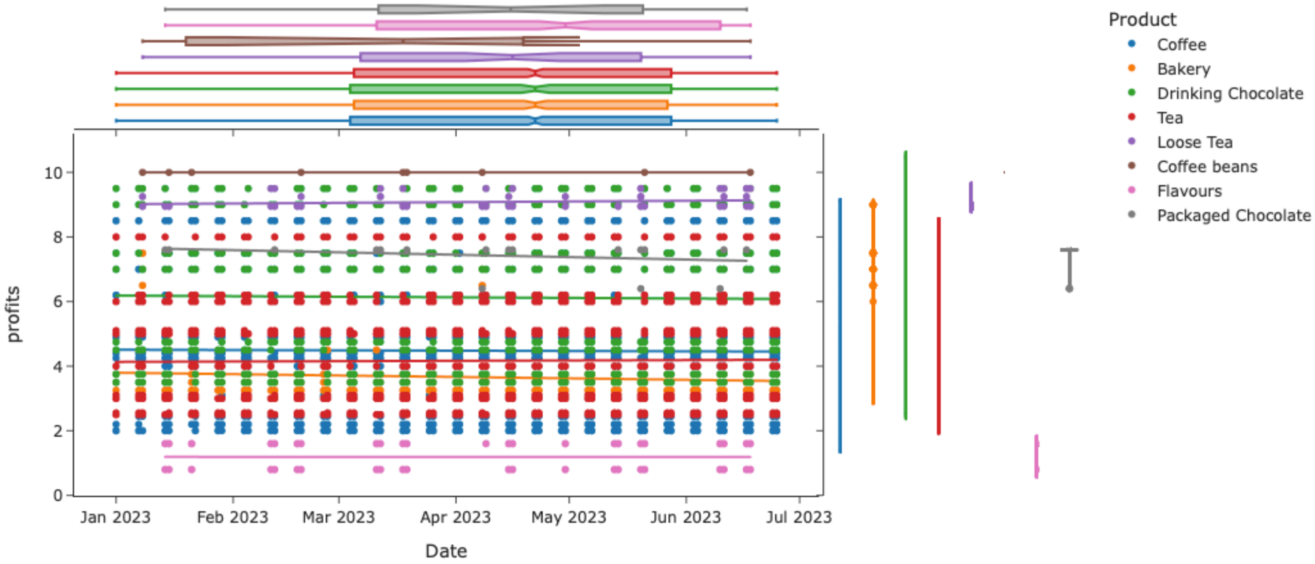
With the bar chart above we can see the difference in sales in Astoria on the weekdays vs the weekends

All the Data has nicely been organized I located all the outlier that were low bound and upper bound in the transactions. After examining the outliers, I eliminated them from the DataFrame and examine the revenue sales over the span of 7 months



I find the evening sales interesting because I would have expected a more dramatic shift toward customers purchasing either chocolate or tea during that time. However, 41% of sales being attributed to coffee still represents a significant portion of evening profits. The difference in sales between the morning and evening is notable, as morning profits more than double those of the evening. This demonstrates that Astoria's most consistent customers visit during the morning hours. Then I moved forward and compare the profits to examine the differences between weekends and weekdays for Astoria.

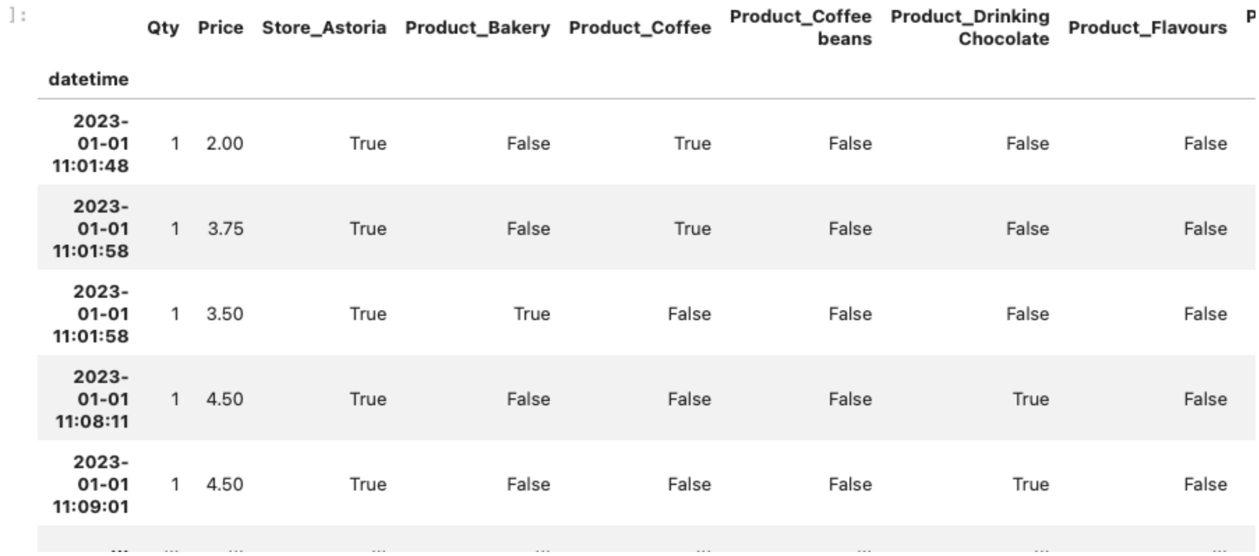




With the graphs above, see that there is consistent sales that Astoria has from weekday to weekend, especially for coffee and tea which provides the greatest revenue. Through careful analysis, I can see that the time periods with the lowest profits are evenings and weekends. Additionally, the products offered by Astoria that generate the lowest profits include branded items, loose tea, flavored products, and packaged chocolate.

4. Prep for Split

I used date time module to organize the time series of the data, made sure there are no missing values, dropped the columns 'Date' / 'Time' and used pd.get_dummies creates new columns for each unique value in the category columns. I noticed that there were columns 'Price' and 'profits', I dropped 'profits' because it was an unnecessary column in the process of modeling/forcasting.

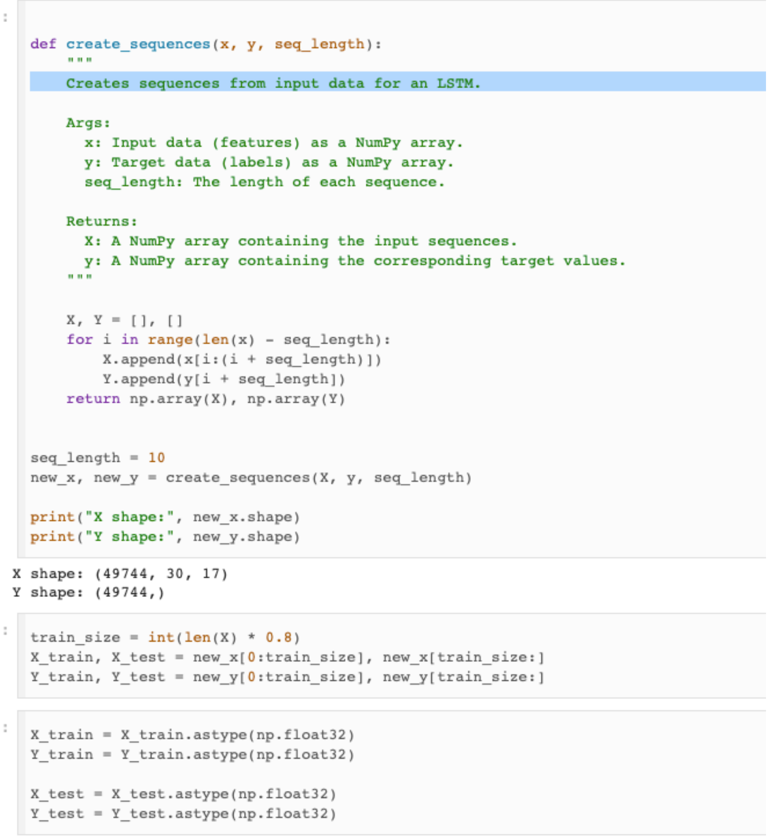


datetime	Qty	Price	Store_Astoria	Product_Bakery	Product_Coffee	Product_Coffee beans	Product_Drinking Chocolate	Product_Flavours	P
2023-01-01 11:01:48	1	2.00	True	False	True	False	False	False	
2023-01-01 11:01:58	1	3.75	True	False	True	False	False	False	
2023-01-01 11:01:58	1	3.50	True	True	False	False	False	False	
2023-01-01 11:08:11	1	4.50	True	False	False	False	True	False	
2023-01-01 11:09:01	1	4.50	True	False	False	False	True	False	
...	...	...	...	...	...	...	...	...	

Now we've got our DataFrame prepped and it's time to create our X and y variable for the Train/Test split. For each modeling I had to treat the arrays differently because some modeling needed a 3D array and other's I needed a 2D array.

5. Train Test Split

The sequences were created for the input data to prepare the data to go through the Long Short Term Memory Model because it needs sequential data.



```
def create_sequences(x, y, seq_length):
    """
    Creates sequences from input data for an LSTM.

    Args:
      x: Input data (features) as a NumPy array.
      y: Target data (labels) as a NumPy array.
      seq_length: The length of each sequence.

    Returns:
      X: A NumPy array containing the input sequences.
      y: A NumPy array containing the corresponding target values.
    """

    X, Y = [], []
    for i in range(len(x) - seq_length):
        X.append(x[i:(i + seq_length)])
        Y.append(y[i + seq_length])
    return np.array(X), np.array(Y)


seq_length = 10
new_x, new_y = create_sequences(X, y, seq_length)

print("X shape:", new_x.shape)
print("Y shape:", new_y.shape)
```

```
X shape: (49744, 30, 17)
Y shape: (49744,)
```

```
train_size = int(len(X) * 0.8)
X_train, X_test = new_x[0:train_size], new_x[train_size:]
Y_train, Y_test = new_y[0:train_size], new_y[train_size:]
```

```
X_train = X_train.astype(np.float32)
Y_train = Y_train.astype(np.float32)

X_test = X_test.astype(np.float32)
Y_test = Y_test.astype(np.float32)
```

This allows the LSTM model to train on the sequences created. I can adjust the sequence lengths and parameters to my specific needs if I wanted. This framing also helps with other modeling we will be doing with GRU which is Grated Recurrent Unit.

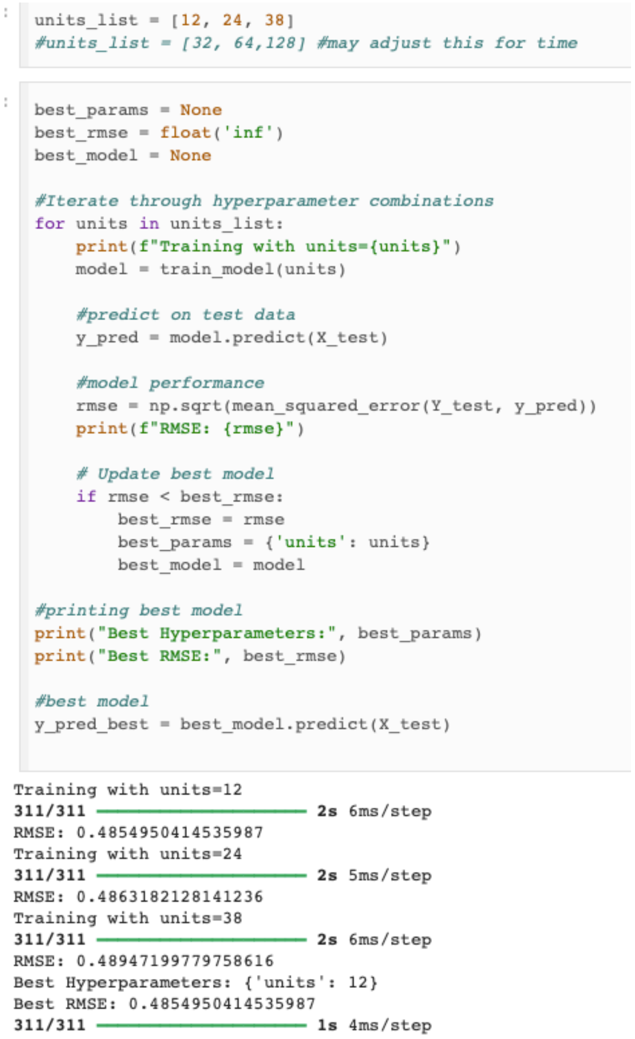
To layman's terms a sequence is a ordered list of data points and captures the temporal or sequential dependencies within the data.

Train/test split is a part of the stamp to prepare the data for machine learning and forecasting, Modeling with learn in the training set using 80% of the data and then we will evaluate the performance in the remaining in test. Since this is done it will lead into our next step, modeling.

6. Modeling

The models I use to do a deeper dive in the data is mainly; LSTM, GRU, XGBoost and a bit of VAR.

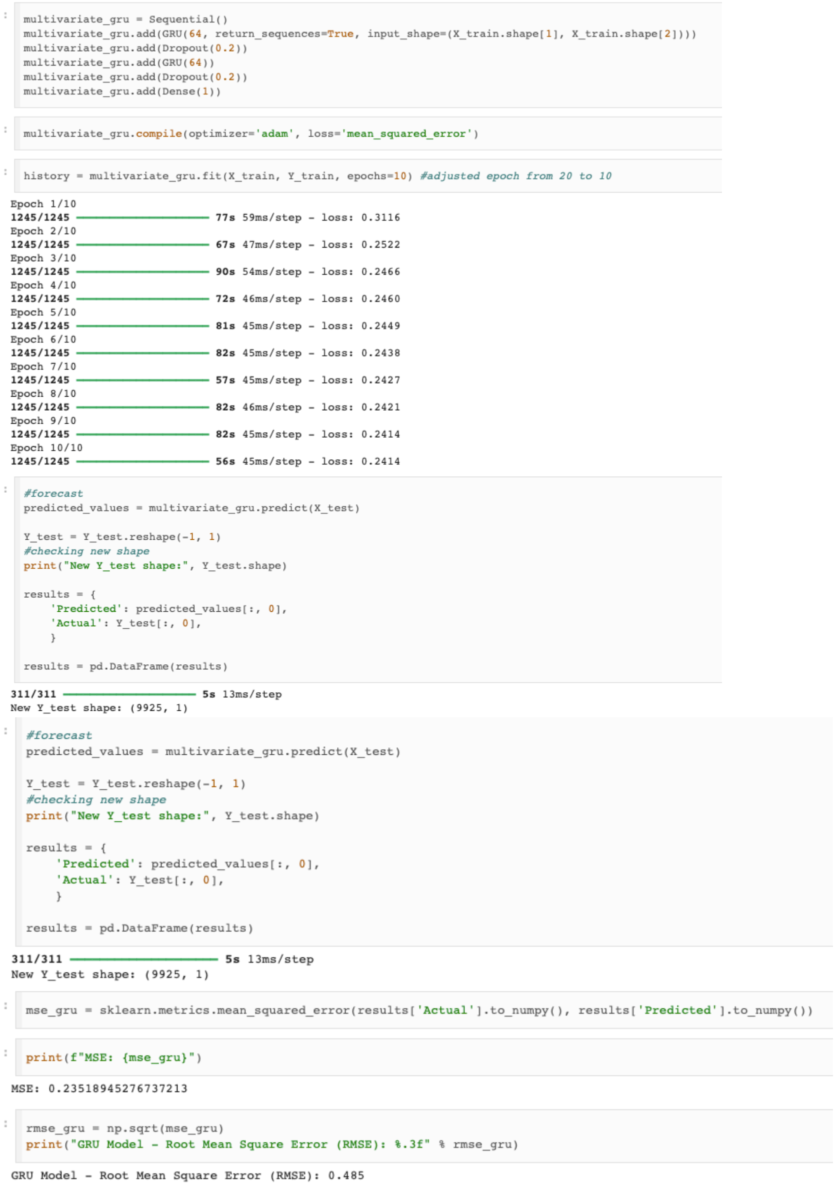
I begin my model examining LSTM which is, Long term Short term memory, this modeling uses neural network and can be used to analyze long term dependencies and was designed to prevent the problems of those dependencies. This is normally used as a time series forecasting. Normally this model is used to predict stop price. While GRU, Gated Recurrent Unit, this uses gates to control the flow of information that passes through the network of gates to learn each long term dependency and this also uses sequential data just like LSTM. This model will hone in on the relevant data. Below we are going to look at how I used both models to examine the coffee data set of Astoria. The left displays LSTM and right shows GRU.



```
units_list = [12, 24, 38]
#units_list = [32, 64,128] #may adjust this for time
```

```
best_params = None
best_rmse = float('inf')
best_model = None

#Iterate through hyperparameter combinations
for units in units_list:
    print(f"Training with units={units}")
    model = train_model(units)

    #predict on test data
    y_pred = model.predict(X_test)

    #model performance
    rmse = np.sqrt(mean_squared_error(Y_test, y_pred))
    print(f"RMSE: {rmse}")

    # Update best model
    if rmse < best_rmse:
        best_rmse = rmse
        best_params = {'units': units}
        best_model = model

#printing best model
print("Best Hyperparameters:", best_params)
print("Best RMSE:", best_rmse)

#best model
y_pred_best = best_model.predict(X_test)
```

```
Training with units=12
311/311 ━━━━━━━━━━━━━━━━━━━━ 2s 6ms/step
RMSE: 0.48549504145359887
Training with units=24
311/311 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step
RMSE: 0.4863182128141236
Training with units=38
311/311 ━━━━━━━━━━━━━━━━━━━━ 2s 6ms/step
RMSE: 0.48947199779758616
Best Hyperparameters: {'units': 12}
Best RMSE: 0.48549504145359887
311/311 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step
```



```
multivariate_gru = Sequential()
multivariate_gru.add(GRU(64, return_sequences=True, input_shape=(X_train.shape[1], X_train.shape[2])))
multivariate_gru.add(Dropout(0.2))
multivariate_gru.add(GRU(64))
multivariate_gru.add(Dropout(0.2))
multivariate_gru.add(Dense(1))
```

```
multivariate_gru.compile(optimizer='adam', loss='mean_squared_error')
```

```
history = multivariate_gru.fit(X_train, Y_train, epochs=10) #adjusted epoch from 20 to 10
```

```
Epoch 1/10
1245/1245 ━━━━━━━━━━━━━━━━━━━━ 77s 59ms/step - loss: 0.3116
Epoch 2/10
1245/1245 ━━━━━━━━━━━━━━━━━━━━ 67s 47ms/step - loss: 0.2522
Epoch 3/10
1245/1245 ━━━━━━━━━━━━━━━━━━━━ 90s 54ms/step - loss: 0.2466
Epoch 4/10
1245/1245 ━━━━━━━━━━━━━━━━━━━━ 72s 46ms/step - loss: 0.2460
Epoch 5/10
1245/1245 ━━━━━━━━━━━━━━━━━━━━ 81s 45ms/step - loss: 0.2449
Epoch 6/10
1245/1245 ━━━━━━━━━━━━━━━━━━━━ 82s 45ms/step - loss: 0.2438
Epoch 7/10
1245/1245 ━━━━━━━━━━━━━━━━━━━━ 57s 45ms/step - loss: 0.2427
Epoch 8/10
1245/1245 ━━━━━━━━━━━━━━━━━━━━ 82s 46ms/step - loss: 0.2421
Epoch 9/10
1245/1245 ━━━━━━━━━━━━━━━━━━━━ 82s 45ms/step - loss: 0.2414
Epoch 10/10
1245/1245 ━━━━━━━━━━━━━━━━━━━━ 56s 45ms/step - loss: 0.2414
```

```
#forecast
predicted_values = multivariate_gru.predict(X_test)

Y_test = Y_test.reshape(-1, 1)
#checking new shape
print("New Y_test shape:", Y_test.shape)

results = {
    'Predicted': predicted_values[:, 0],
    'Actual': Y_test[:, 0],
    }

results = pd.DataFrame(results)
```

```
311/311 ━━━━━━━━━━━━━━━━━━━━ 5s 13ms/step
New Y_test shape: (9925, 1)
```

```
#forecast
predicted_values = multivariate_gru.predict(X_test)

Y_test = Y_test.reshape(-1, 1)
#checking new shape
print("New Y_test shape:", Y_test.shape)

results = {
    'Predicted': predicted_values[:, 0],
    'Actual': Y_test[:, 0],
    }

results = pd.DataFrame(results)
```

```
311/311 ━━━━━━━━━━━━━━━━━━━━ 5s 13ms/step
New Y_test shape: (9925, 1)
```

```
mse_gru = sklearn.metrics.mean_squared_error(results['Actual'].to_numpy(), results['Predicted'].to_numpy())
```

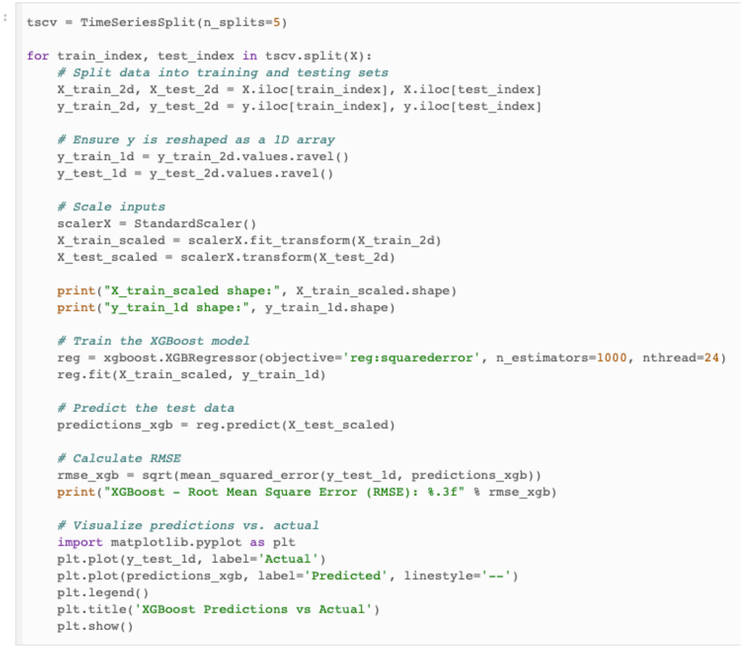
```
print(f"MSE: {mse_gru}")
```

```
MSE: 0.23518945276737213
```

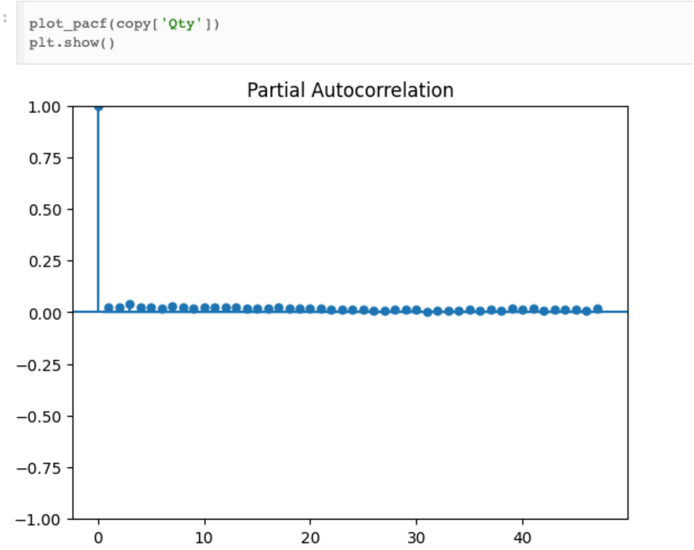
```
rmse_gru = np.sqrt(mse_gru)
print("GRU Model - Root Mean Square Error (RMSE): %.3f" % rmse_gru)
```

```
GRU Model - Root Mean Square Error (RMSE): 0.485
```

Now that we've walked through the first two models we go into XGBoost and VAR. XGBoost is, Extreme Gradient Boosting, which can be used for none linear data and can capture intricate patterns in time-series data. This is really useful to me because I will be analyzing sales in coffeeshops over a 7 month period. While VAR is, Vector Auto Regression, this model analyzes multivariate data to make predictions of future values. This model takes into account for lags, it describes each variable as a function with it's own lagged value and I decided to research whether I should use this model because it's normally used for multivariate time series. I decided to graph a Partial Autocorrelation, there weren't any good negative values to indicate max lags for the parameters for the model so I opted not to do the modeling for VAR. The XGBoost is on the left and the graphing that aided my decision on VAR is on the right.



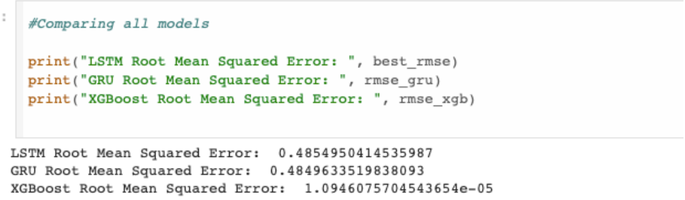
```
tscv = TimeSeriesSplit(n_splits=5)

for train_index, test_index in tscv.split(X):
    # Split data into training and testing sets
    X_train_2d, X_test_2d = X.iloc[train_index], X.iloc[test_index]
    y_train_2d, y_test_2d = y.iloc[train_index], y.iloc[test_index]

    # Ensure y is reshaped as a 1D array
    y_train_1d = y_train_2d.values.ravel()
    y_test_1d = y_test_2d.values.ravel()

    # Scale inputs
    scalerX = StandardScaler()
    X_train_scaled = scalerX.fit_transform(X_train_2d)
    X_test_scaled = scalerX.transform(X_test_2d)

    print("X_train_scaled shape:", X_train_scaled.shape)
    print("y_train_1d shape:", y_train_1d.shape)

    # Train the XGBoost model
    reg = xgboost.XGBRegressor(objective='reg:squarederror', n_estimators=1000, nthread=24)
    reg.fit(X_train_scaled, y_train_1d)

    # Predict the test data
    predictions_xgb = reg.predict(X_test_scaled)

    # Calculate RMSE
    rmse_xgb = sqrt(mean_squared_error(y_test_1d, predictions_xgb))
    print("XGBoost - Root Mean Square Error (RMSE): %.3f" % rmse_xgb)

    # Visualize predictions vs. actual
    import matplotlib.pyplot as plt
    plt.plot(y_test_1d, label='Actual')
    plt.plot(predictions_xgb, label='Predicted', linestyle='--')
    plt.legend()
    plt.title('XGBoost Predictions vs Actual')
    plt.show()
```



```
plot_pacf(copy['Qty'])
plt.show()
```

7. Comparing the Models

Now that all the models have been trained I compared each model using Root Mean Squared Error. Due to the closes value to 1 is XGBoost. I would determine that this provides the best modeling.



```
#Comparing all models

print("LSTM Root Mean Squared Error: ", best_rmse)
print("GRU Root Mean Squared Error: ", rmse_gru)
print("XGBoost Root Mean Squared Error: ", rmse_xgb)
```

```
LSTM Root Mean Squared Error:  0.4854950414535987
GRU Root Mean Squared Error:  0.4849633519838093
XGBoost Root Mean Squared Error:  1.0946075704543654e-05
```

8. Conclusion

Through careful analysis, I can see that the time periods with the lowest profits are evenings and weekends. Additionally, the products offered by Astoria that generate the lowest profits include branded items, loose tea, flavored products, and packaged chocolate. It is also worth noting that shelf items, which may be displayed at the front of the coffee shop, tend to sell the least but have the highest outliers in terms of revenue. This insight suggests a potential low-effort opportunity to increase sales. The analysis further shows that coffee and tea have consistent sales throughout the day but experience a significant boost in the morning.

Sales decline in the evening, which may present a hidden opportunity to host events such as tastings or to inspire customers to purchase more shelf items that yield higher revenue. Another small change that can be made is opening Astoria at 6 am instead of 7 am. This would accommodate customers who work early hours during the cold mornings and desire a nice cup of coffee. We can verify this by finding a dataset that identifies when most people go to work in New York City. This can help us determine if it would be worth it for Astoria to open one hour earlier.

Now that we have some understanding and exploration of our data and have completed the modeling for the multivariate time series, we see that Astoria has consistent sales and that we can build upon that. Next, I would like to explore my proposals to improve Astoria's business and even compare them to the other coffee shops in the original dataset.