- Scrum Master for Next Week
    - Gustavo A Hernandez
- List at least 5 things the team did well and will continue doing
    - Working in the Python code
    - Looking forward to developing visualizations in Tableau
    - Sharing ideas
    - Developing code
    - Working with Trello
- List at least 3 things the team did poorly and how you will mitigate them next sprint
    - Time management
    - Workspace
    - Weekly meeting
- List shout-outs to any team members for excelling in any way
    - Chelsea Miller – Taking care of Trello for this week
- What did you learn as a team this week?
    - How to share ideas
- What did you learn as an individual this week?
    - How to work while life is being busy

CODE

Libraries

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib.pylab import rcParams
rcParams['figure.figsize'] = (10,5)
import seaborn as sns
sns.set_style('darkgrid')

from statsmodels.graphics.tsaplots import plot_acf,plot_pacf
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.tsa.stattools import adfuller
from statsmodels.tsa.statespace.sarimax import SARIMAX

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

Loading Data

```
coffee_raw = pd.read_csv('../Data/coffee.csv')
```

Data Wrangling

```
coffee_raw.Date = pd.to_datetime(coffee_raw.Date, yearfirst=True)
coffee_raw.set_index('Date', inplace = True)
coffee = coffee_raw.asfreq('b', 'ffill')
```

Exploratory Analysis

```
fig,axes = plt.subplots(2,2,figsize=[15,7])
fig.suptitle('Coffee Price',size=24)

## Resampling to Daily freq (Original Data)
axes[0,0].plot(coffee.Close)
axes[0,0].set_title("Daily",size=16)

## Resampling to Monthly freq
axes[0,1].plot(coffee.Close.resample('M').mean())
axes[0,1].set_title("Monthly",size=16)

## Resmapling to Quarterly freq
axes[1,0].plot(coffee.Close.resample('Q').mean())
axes[1,0].set_title('Quarterly',size=16)

## Resampling to Annualy freq
axes[1,1].plot(coffee.Close.resample('A').mean())
axes[1,1].set_title('Annualy',size=16)

plt.tight_layout()
plt.show()
```

Using statsmodels

```
data_close_price = coffee.Close.resample('Q').mean()
```

```
decompose_result = seasonal_decompose(data_close_price, model = 'additive')

## Systematic Components
trend = decompose_result.trend
seasonal = decompose_result.seasonal

## Non-Systematic Components
residual = decompose_result.resid
decompose_result.plot();
```

Stationarity

```
def plot_rolling_stats(series, window=1):
    ## Calculating the Rolling Mean and Rolling Standard Deviation
    rol_mean = series.rolling(window).mean()
    rol_std  = series.rolling(window).std()

    ## ploting the results along side the original data
    fig = plt.figure(figsize=(10,5))
    orig = plt.plot(series,color='blue',label='Original')
    mean = plt.plot(rol_mean,color='red',label='Rolling mean')
    std  = plt.plot(rol_std,color='black',label='Rolling std')


    plt.title('Rolling Mean/Standard Deviation',size=20)
    plt.legend(loc='best')
    plt.show(block=False)
```

```
def stationarity_check(series):
    print('Results of Dickey Fuller Test:')
    coffee_test = adfuller(series, autolag='AIC')

    coffee_output = pd.Series(coffee_test[0:4], index=['Test Statistic','p-value',
                                        '#Lags Used','Number of Observations Used'])
    for key,value in coffee_test[4].items():
        coffee_output['Critical Value (%s)'%key] = value

    print(coffee_output)
```

```
plot_rolling_stats(data_close_price,4)
stationarity_check(data_close_price)
```

```
## Regular Differentiation
plot_rolling_stats(data_close_price.diff()[1:],4)
stationarity_check(data_close_price.diff()[1:])
```

Autocorrelation and Partial Correlation

```
fig = plt.figure(figsize=(20,5))
ax_1 = fig.add_subplot(121)
plot_pacf(data_close_price,lags=12,zero=False,ax=ax_1)

ax_2 = fig.add_subplot(122)
plot_acf(data_close_price,lags=12,zero=False,ax=ax_2);
```

Time Series Modeling

```
size = 0.8 ## train size
train, test = data_close_price.iloc[:int(size *len(data_close_price))],
data_close_price.iloc[int(size *len(data_close_price)):]
```

SARIMAX

```
model = SARIMAX(train,order=(2,1,2),seasonal_order=(1,1,1,4)).fit(disp=-1)
model.summary()
```

```
model.plot_diagnostics(figsize=(20,10))
plt.show()
```

Predictions

```
predictions = model.get_prediction(start='2000-03-31',end='2022-06-30')
conf = predictions.conf_int()
test_conf = conf.loc[test.index[0]:]
## ploting results
plt.plot(predictions.predicted_mean[1:],color='red',label='predictions')
plt.plot(train,color='blue',label='original')
plt.plot(test,color='green',label='test')
plt.fill_between(test_conf.index, test_conf.iloc[:,0], test_conf.iloc[:,1], color='gray', alpha=.2,label='95% confidence')
plt.title('Original vs Predictions',size=20)
plt.legend(loc='best');
```

Accuracy Metrics

```
print(f"Mean Absolute Error: {mean_absolute_error(data_close_price[1:],predictions.predicted_mean[1:])}")
print(f"Mean Absolute Percentage Error:
{mean_absolute_percentage_error(data_close_price[1:],predictions.predicted_mean[1:])}")
```