INDR 372 Homework Assignment 1

Sarp Çağan Kelleci 79482, Tan Karahasanoğlu 79136

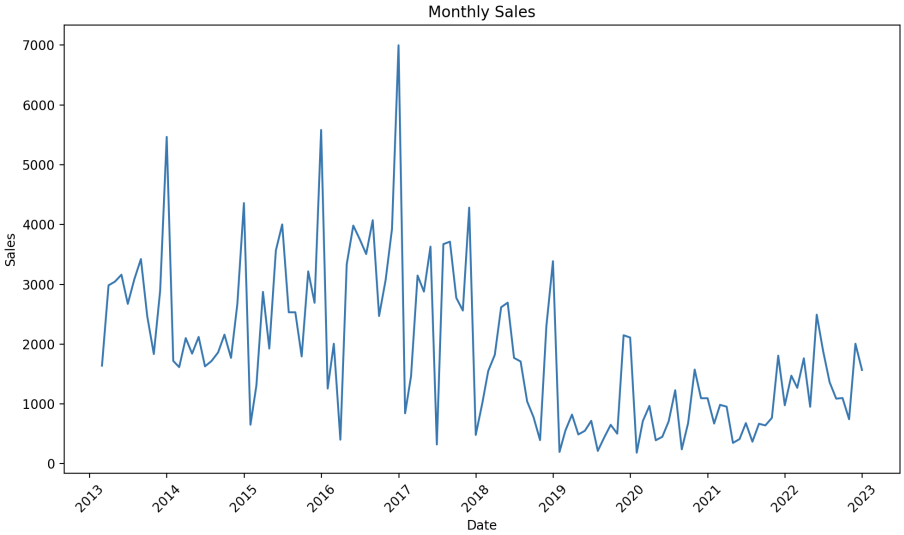
March 22, 2024

(Assistance was taken from ChatGPT 3.5 during the writing of certain codes)

Q1)

a)

```python
import pandas as pd
import matplotlib.pyplot as plt


df = pd.read_csv('renault.csv', skiprows=1, header=None)
df = df.iloc[:, :2]

df.columns = ['Date', 'Sales']

df['Date'] = pd.to_datetime(df['Date'])

df['Sales'] = pd.to_numeric(df['Sales'], errors='coerce')

df_monthly = df.resample('M', on='Date').sum()

plt.figure(figsize=(10, 6))

plt.plot(df_monthly.index, df_monthly['Sales'], linestyle='-')

plt.title('Monthly Sales')
plt.xlabel('Date')
plt.ylabel('Sales')
plt.xticks(rotation=45)
plt.tight_layout()

plt.show()
```



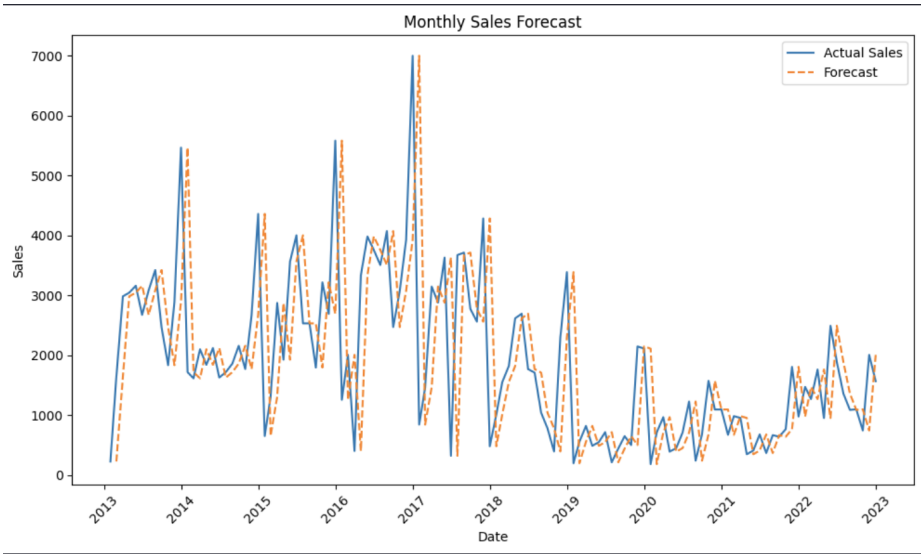
Analysis: As can be seen above, there isn't a significant upwards or downward shift across the graph. This means that there is not a trend observable. That being said, when specific periods such as between $2017 - 2020$ are observed we can see a general downward shift. Whether this specific period indicates a trend is questionable. In terms of seasonality, while there are fluctuating peaks and burrows for the data, these do not occur in reoccurring periods, therefore there is also no seasonality. These peaks and burrows can be attributed to outside interference in the car market such as cheaper purchasing due to outside financing or sales rather than a seasonal happenstance.

(b)

The following will be the code, graph, and requested values of MAE, MAPE and RMSE, for the forecast when $\tau = 1$

Tau = 1

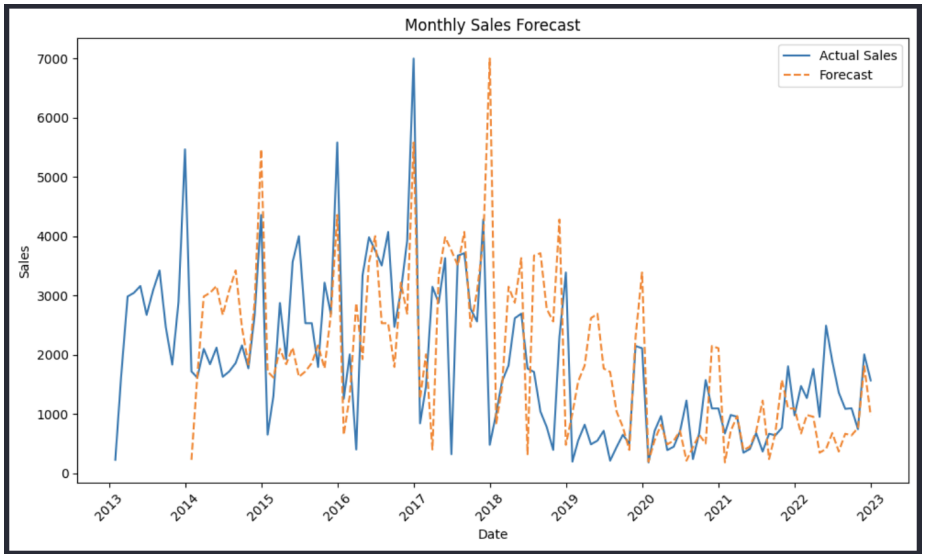
```
df_monthly = df.resample('M', on='Date').sum()

for i in range(1, len(df_monthly)):
    df_monthly.loc[df_monthly.index[i], 'Forecast'] = df_monthly.loc[df_monthly.index[i - 1], 'Sales']

df_monthly['Error'] = df_monthly['Sales'] - df_monthly['Forecast']


start_index = df[df['Date'] == '2019-01-01'].index[0]

mae = (df_monthly['Sales'][start_index:] - df_monthly['Forecast'][start_index:]).abs().mean()
mape = ((df_monthly['Sales'][start_index:] - df_monthly['Forecast'][start_index:]) / df_monthly['Sales'][start_index:]).abs().mean() * 100
rmse = ((df_monthly['Sales'][start_index:] - df_monthly['Forecast'][start_index:]) ** 2).mean() ** 0.5

print("Mean Absolute Error (MAE):", mae)
print("Mean Absolute Percentage Error (MAPE):", mape)
print("Root Mean Squared Error (RMSE):", rmse)

plt.figure(figsize=(10, 6))

plt.plot(df_monthly.index, df_monthly['Sales'], label='Actual Sales', linestyle='-')

plt.plot(df_monthly.index[1:], df_monthly['Forecast'][1:], label='Forecast', linestyle='--')

plt.title('Monthly Sales Forecast')
plt.xlabel('Date')
plt.ylabel('Sales')
plt.xticks(rotation=45)
plt.legend()
plt.tight_layout()

plt.show()
```




```
Mean Absolute Error (MAE): 529.4166666666666
Mean Absolute Percentage Error (MAPE): 109.72726757103048
Root Mean Squared Error (RMSE): 782.5992216113345
```

The following will be the code, graph, and requested values of MAE, MAPE and RMSE for the forecast when $\tau = 12$

```python
df_monthly = df.resample('M', on='Date').sum()

for i in range(1, len(df_monthly)):
    df_monthly.loc[df_monthly.index[i], 'Forecast'] = df_monthly.loc[df_monthly.index[i - 12], 'Sales']

df_monthly['Error'] = df_monthly['Sales'] - df_monthly['Forecast']


start_index = df[df['Date'] == '2019-01-01'].index[0]

mae = (df_monthly['Sales'][start_index:] - df_monthly['Forecast'][start_index:]).abs().mean()
mape = ((df_monthly['Sales'][start_index:] - df_monthly['Forecast'][start_index:]) / df_monthly['Sales'][start_index:]).abs().mean() * 100
rmse = ((df_monthly['Sales'][start_index:] - df_monthly['Forecast'][start_index:]) ** 2).mean() ** 0.5

print("Mean Absolute Error (MAE):", mae)
print("Mean Absolute Percentage Error (MAPE):", mape)
print("Root Mean Squared Error (RMSE):", rmse)

plt.figure(figsize=(10, 6))

plt.plot(df_monthly.index, df_monthly['Sales'], label='Actual Sales', linestyle='-')

plt.plot(df_monthly.index[12:], df_monthly['Forecast'][12:], label='Forecast', linestyle='--')

plt.title('Monthly Sales Forecast')
plt.xlabel('Date')
plt.ylabel('Sales')
plt.xticks(rotation=45)
plt.legend()
plt.tight_layout()

plt.show()
```

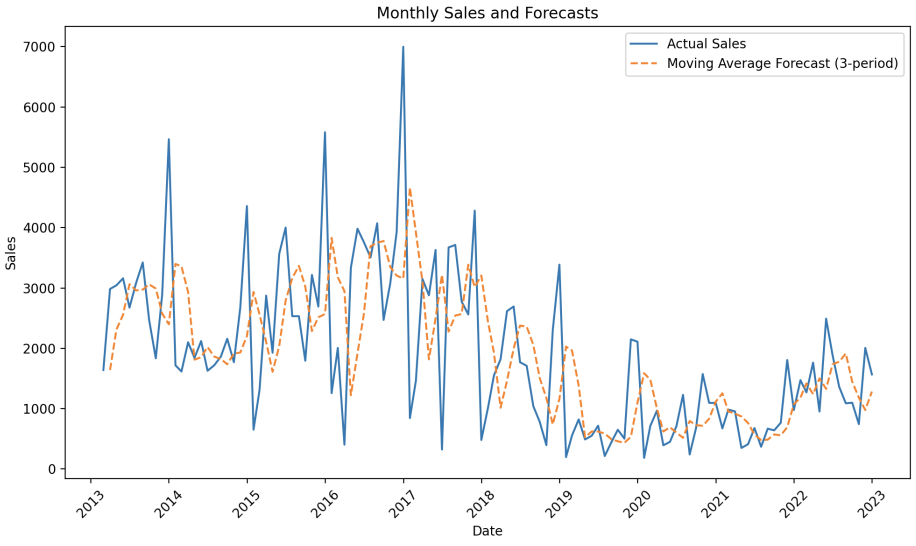



```
Mean Absolute Error (MAE): 606.7708333333334
Mean Absolute Percentage Error (MAPE): 88.86049187264734
Root Mean Squared Error (RMSE): 832.7095457000598
```

(c)

We have adjusted our previous do me more conducive to mathematical operations and silenced the graphing function for the previous two naive forecasts as we have already provided them.

```python
df_monthly = df.resample('M', on='Date').sum()

for i in range(1, len(df_monthly)):
    df_monthly.loc[df_monthly.index[i], 'Forecast'] = df_monthly.loc[df_monthly.index[i - 12], 'Sales']

df_monthly['Error'] = df_monthly['Sales'] - df_monthly['Forecast']


start_index = df[df['Date'] == '2019-01-01'].index[0]

mae = (df_monthly['Sales'][start_index:] - df_monthly['Forecast'][start_index:]).abs().mean()
mape = ((df_monthly['Sales'][start_index:] - df_monthly['Forecast'][start_index:]) / df_monthly['Sales'][start_index:]).abs().mean() * 100
rmse = ((df_monthly['Sales'][start_index:] - df_monthly['Forecast'][start_index:]) ** 2).mean() ** 0.5

print("Mean Absolute Error (MAE):", mae)
print("Mean Absolute Percentage Error (MAPE):", mape)
print("Root Mean Squared Error (RMSE):", rmse)

plt.figure(figsize=(10, 6))

plt.plot(df_monthly.index, df_monthly['Sales'], label='Actual Sales', linestyle='-')

plt.plot(df_monthly.index[12:], df_monthly['Forecast'][12:], label='Forecast', linestyle='--')

plt.title('Monthly Sales Forecast')
plt.xlabel('Date')
plt.ylabel('Sales')
plt.xticks(rotation=45)
plt.legend()
plt.tight_layout()

plt.show()
```



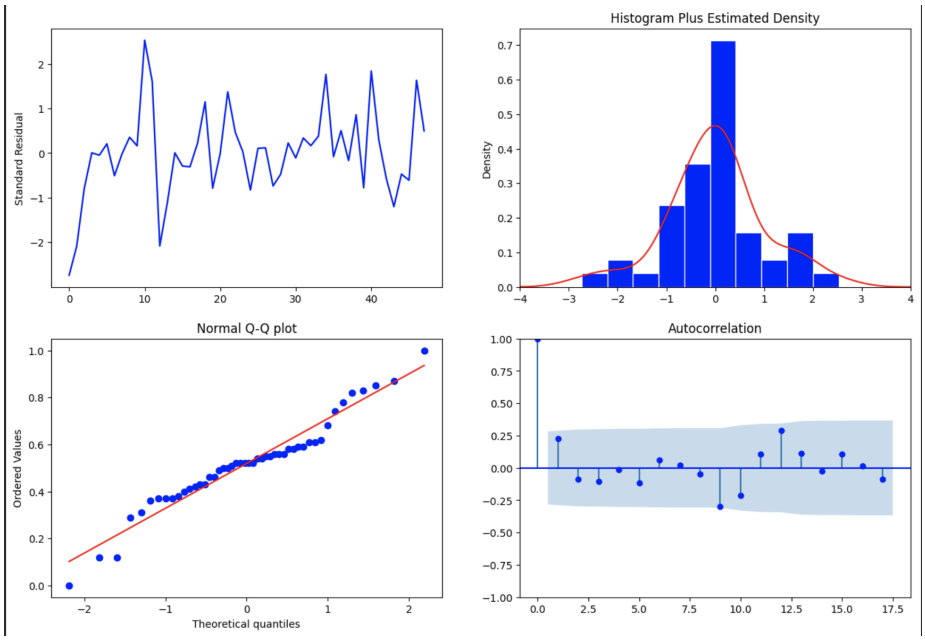
```
Naive Forecast 1 (Ft = Dt-1) Errors:
MAE: 529.4166666666666
MAPE: 109.72726757103048
RMSE: 782.5992216113345

Naive Forecast 2 (Ft = Dt-12) Errors:
MAE: 606.7708333333334
MAPE: 88.86049187264734
RMSE: 832.7095457000598

Moving Average Forecast (3-period) Errors:
MAE: 460.8680555555555
MAPE: 82.30538388339791
RMSE: 649.3309038427058

Prediction Intervals for 2022:
Date                    Lower Bound                  Upper Bound
2022-01-31 00:00:00     -1280.9538424247253     3648.9538424247253
2022-02-28 00:00:00     -1045.620509091392      3884.2871757580588
2022-03-31 00:00:00     -1224.620509091392      3705.2871757580588
2022-04-30 00:00:00     -962.9538424247253      3966.9538424247253
2022-05-31 00:00:00     -1136.2871757580585     3793.620509091392
2022-06-30 00:00:00     -728.620509091392       4201.287175758059
2022-07-31 00:00:00     -686.2871757580585      4243.620509091392
2022-08-31 00:00:00     -549.620509091392       4380.287175758059
2022-09-30 00:00:00     -1017.620509091392      3912.2871757580588
2022-10-31 00:00:00     -1281.2871757580585     3648.620509091392
2022-11-30 00:00:00     -1487.2871757580588     3442.620509091392
```

We can see that the moving average provides considerably lower error values (MAE, MAPE, and RMSE) than the previous naive forecasts, implying it's a better fit as a forecasting method for this data set

(d)

Here is our data as well as our various graphs for residuals.

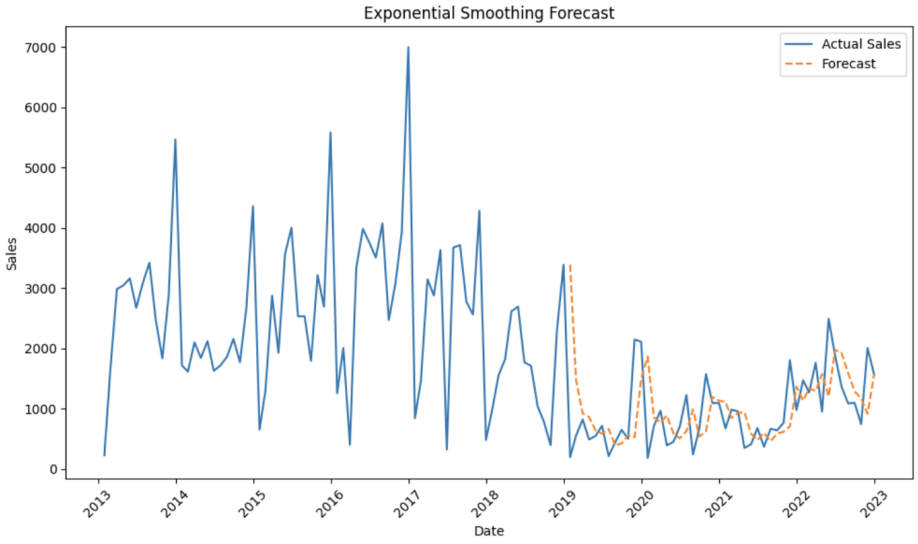
```
Mean of Residual: 42.9942
S.D. of Residual: 1187.0257
MAE : 891.2725
MAPE: 79.6540
MSE : 1398626.0918
RMSE: 1182.6352
Mean of Residual / Mean of Sales: 2.29%
```



Beyond this, our distribution on the normal line has frequent outliers, implying that the values aren't distributed normally. The auto-correlation doesn't indicate that there is a statistically significant correlation within the data, therefore independence can be implied.

e)

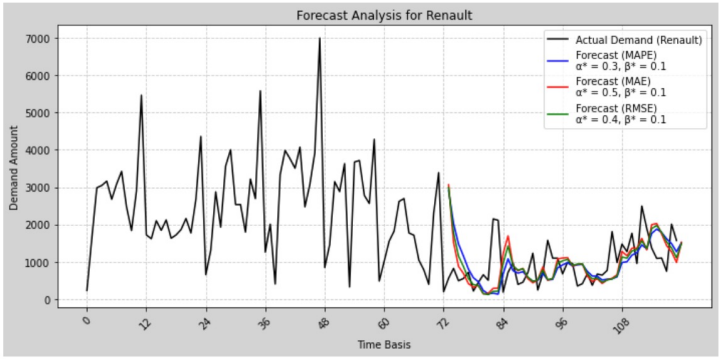
```
Best alpha: 0.6
MAE: 480.7988204017501
MAPE: 102.05588125633409
RMSE: 743.7248995079555
```



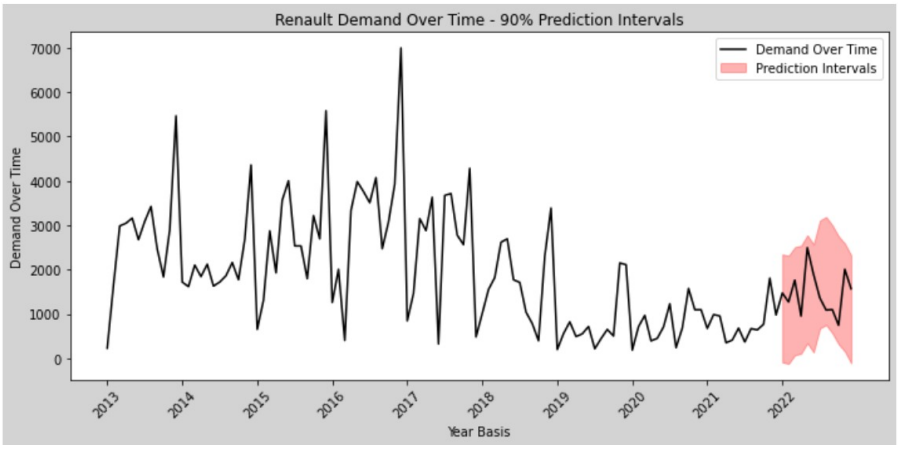
Based on our values, which prioritize MAE in determining best α value, $\alpha = 0.6$. MA-3 better error values, which generally could indicate that compared to the exponential smoothing models, the last 3 values can have a higher impact on the determination of the upcoming value in this model.

f)

```
Best MAPE (Renault): 69.80290423154662
Alpha (MAPE): 0.3
Beta (MAPE): 0.1
Best RMSE (Renault): 740.3038392206994
Alpha (RMSE): 0.4
Beta (RMSE): 0.1
Best MAE (Renault): 541.8243398590141
Alpha (MAE): 0.5
Beta (MAE): 0.1
```




```
90% Confidence Prediction Intervals for Renault:
Forecast: None, Interval: (-91.07941076419593, 2344.303499212338)
Forecast: None, Interval: (-122.64741343605988, 2312.735496540474)
Forecast: None, Interval: (71.53002329873334, 2506.9129332752673)
Forecast: None, Interval: (106.06762620812924, 2541.450536184663)
Forecast: None, Interval: (341.55982470591084, 2776.942734682445)
Forecast: None, Interval: (134.60509261681273, 2569.9880025933467)
Forecast: None, Interval: (672.0603914591509, 3107.4433014356846)
Forecast: None, Interval: (753.343496906657, 3188.726406883191)
Forecast: None, Interval: (566.9919620993635, 3002.3748720758977)
Forecast: None, Interval: (317.7537045314825, 2753.1366145080165)
Forecast: None, Interval: (154.75294360996372, 2590.1358535864974)
Forecast: None, Interval: (-109.74528888687655, 2325.6376210896574)
```



We have provided errors values, prediction intervals as well as error metrics above. The behaviour and error percentage characteristics of this method is not too different than the previous forecasting models. With RMSE characterizing the generally high prediction interval seen in the last graph.

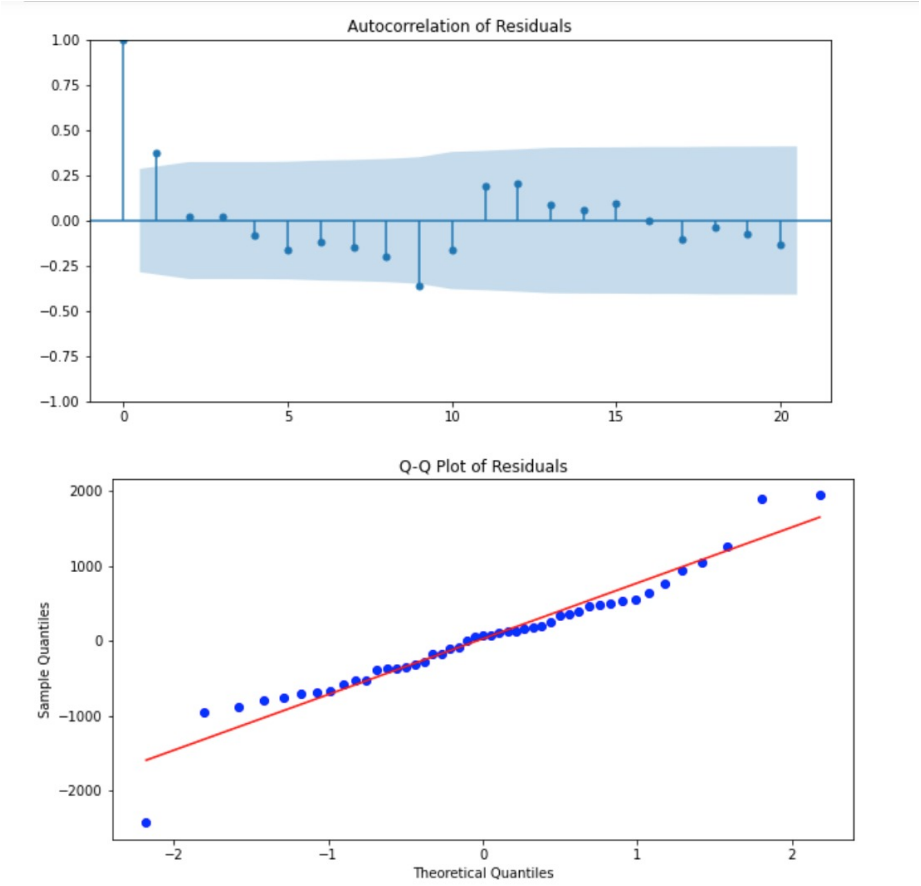
g)

To evaluate the residual diagnostics for independence and normality, we need to analyze the behavior of the residuals obtained from the double exponential smoothing forecast.

Regarding the comparison to previous forecasts, we can observe how the new forecasts based on the optimal alpha and beta values perform compared to the previous forecasts. We should compare the accuracy metrics (MAPE, RMSE, MAE) of the new forecasts with those of the previous forecasts. If the new forecasts have lower error metrics, it indicates that they are more accurate and provide better forecasting.

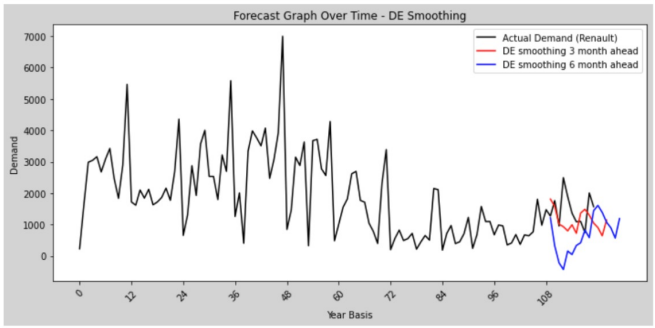
A drawback of the double exponential smoothing forecast concerning this data is that it might not capture more complex patterns or seasonality present in the demand data. Furthermore, and more glaringly, it has trouble adapting to large changes in a manner that doesn't endanger the forecast. Ultimately, double exponential smoothing is a simple forecasting method that assumes a linear trend, which might not be suitable for data with more complex patterns.

We can comment on the residual diagnostics of our data based on the following graphs:



In regards to normality, since our Q-Q plot shows several noticeable outliers, and is generally not a good fit. Therefore, it would be a stretch to attach the label of normal distribution here. In regards to independence, the autocorrelation graph has a significant statistical correlation between the various points at lag(1). It should therefore not be assumed that there is independence.

h)




```
Performance Metrics for 3 Month Lookahead Forecast
Mean Absolute Percentage Error (MAPE): 100.00085557484313
Mean Absolute Error (MAE): 627.4603634457803
Root Mean Square Error (RMSE): 836.6886985891067
Updated Alpha: 0.4
Updated Beta: 0.1
Performance Metrics for 6 Month Lookahead Forecast
Mean Absolute Percentage Error (MAPE): 114.98525810713855
Mean Absolute Error (MAE): 733.1378055395888
Root Mean Square Error (RMSE): 933.88812383681
Updated Alpha: 0.4
Updated Beta: 0.1
```

i)

```python
# Find the alpha values corresponding to minimum error metrics
min_rmse_alpha = (rmse_errors.index(min(rmse_errors)) + 1) / 10
min_mape_alpha = (mape_errors.index(min(mape_errors)) + 1) / 10
min_mae_alpha = (mae_errors.index(min(mae_errors)) + 1) / 10

# Find the minimum error metrics
min_rmse = min(rmse_errors)
min_mape = min(mape_errors)
min_mae = min(mae_errors)

# Print the results
print("Minimum RMSE:", min_rmse, "- Alpha:", min_rmse_alpha)
print("Minimum MAPE:", min_mape, "- Alpha:", min_mape_alpha)
print("Minimum MAE:", min_mae, "- Alpha:", min_mae_alpha)

# Set alpha_star to the alpha corresponding to minimum MAE
alpha_star = min_mae_alpha
```

```
Minimum RMSE: nan - Alpha: 0.1
Minimum MAPE: nan - Alpha: 0.1
Minimum MAE: nan - Alpha: 0.1
```

```
/var/folders/_q/yh72yh5138lfhr1n5jh_2x0c0000gn/T/ipykernel_78661/3726012199.py:45: FutureWarning: The default dtype
for empty Series will be 'object' instead of 'float64' in a future version. Specify a dtype explicitly to silence t
his warning.
  smoothed_series = pd.Series(index=series.index)
```

```python
# Function to transform demand data to remove seasonality
def transform_demand_to_U(demand_data):
    transformed_data = []
    for t in range(len(demand_data)):
        if t < 12:
            transformed_data.append(None)
        else:
            transformed_data.append(demand_data.loc[t] - demand_data.loc[t - 12])
    return pd.DataFrame(transformed_data).loc[:,0]

# Apply transformation to demand data
U_style = transform_demand_to_U(demand_renault)

# Plot the transformed series
plt.figure(figsize=(10, 5))
plt.plot(U_style)
plt.xticks(range(0, len(df['Year']), 12), df['Year'].unique(), rotation=45)
plt.title('Transformed Demand Over Time (2013-2022)')
plt.xlabel('Year')
plt.ylabel('Demand')
plt.tight_layout()
plt.show()
```

```python
# Define the functions for error calculation
def calculate_mean_abs_error(demand_data, forecast_data, start_yr, end_yr, question=1):
    total_error = 0
    data_count = 0
    for t_idx in range(len(demand_data)):
        if forecast_data[t_idx] is None or np.isnan(forecast_data[t_idx]):
            continue
        if check_valid_time(start_yr, end_yr, t_idx, question):
            continue
        error = abs(demand_data[t_idx] - forecast_data[t_idx])
        total_error += error
        data_count += 1
    if data_count == 0:
        return np.nan  # Return NaN if count is zero to avoid division by zero
    return total_error / data_count

def calculate_mean_abs_percentage_error(demand_data, forecast_data, start_yr, end_yr, question=1):
    total_error = 0
    data_count = 0
    for t_idx in range(len(demand_data)):
        if forecast_data[t_idx] is None or np.isnan(forecast_data[t_idx]):
            continue
        if check_valid_time(start_yr, end_yr, t_idx, question):
            continue
        error = demand_data[t_idx] - forecast_data[t_idx]
        if demand_data[t_idx] != 0:  # Check if demand value is not zero to avoid division by zero
            total_error += abs(error / demand_data[t_idx])
            data_count += 1
    if data_count == 0:
        return np.nan  # Return NaN if count is zero to avoid division by zero
    return 100 * total_error / data_count

def calculate_root_mean_squared_error(demand_data, forecast_data, start_yr, end_yr, question=1):
    residuals = compute_residuals(demand_data, forecast_data, start_yr, end_yr, question)
    total_squared_error = 0
    data_count = 0
    for error in residuals:
        data_count += 1
        total_squared_error += error ** 2
    if data_count == 0:
        return np.nan  # or any other value to indicate that RMSE cannot be calculated
    return np.sqrt(total_squared_error / data_count)
```

```python
def simple_exponential_smoothing(series, alpha_val):
    smoothed_series = pd.Series(index=series.index)
    smoothed_series[0] = series[0]
    for t_idx in range(1, len(series)):
        smoothed_series[t_idx] = alpha_val * series[t_idx] + (1 - alpha_val) * smoothed_series[t_idx - 1]
    return smoothed_series

def check_valid_time(start_yr, end_yr, t_idx, question):
    """
    Check if the time index is within the valid range.
    """
    # Placeholder logic, modify as needed
    return False

def compute_residuals(demand_data, forecast_data, start_yr, end_yr, question=1):
    """
    Compute residuals for error calculation.
    """
    # Placeholder logic, modify as needed
    return []

# Initialize lists to store error metrics
rmse_errors = []
mape_errors = []
mae_errors = []

# Iterate over alpha values from 0.1 to 1.0
for i, alpha in enumerate(range(1, 11)):
    alpha /= 10  # Convert alpha to float
    smoothed_series = simple_exponential_smoothing(U_style, alpha)

    # Calculate error metrics for the current alpha
    rmse = calculate_root_mean_squared_error(U_style, smoothed_series, 2019, 2022)
    mape = calculate_mean_abs_percentage_error(U_style, smoothed_series, 2019, 2022)
    mae = calculate_mean_abs_error(U_style, smoothed_series, 2019, 2022)

    # Append error metrics to respective lists
    rmse_errors.append(rmse)
    mape_errors.append(mape)
    mae_errors.append(mae)

# Find the alpha values corresponding to minimum error metrics
min_rmse_alpha = (rmse_errors.index(min(rmse_errors)) + 1) / 10
min_mape_alpha = (mape_errors.index(min(mape_errors)) + 1) / 10
min_mae_alpha = (mae_errors.index(min(mae_errors)) + 1) / 10

# Find the minimum error metrics
min_rmse = min(rmse_errors)
min_mape = min(mape_errors)
min_mae = min(mae_errors)
```

```python
def simple_exponential_smoothing(series, alpha_val):
    smoothed_series = pd.Series(index=series.index)
    smoothed_series[0] = series[0]
    for t_idx in range(1, len(series)):
        smoothed_series[t_idx] = alpha_val * series[t_idx] + (1 - alpha_val) * smoothed_series[t_idx - 1]
    return smoothed_series

def check_valid_time(start_yr, end_yr, t_idx, question):
    """
    Check if the time index is within the valid range.
    """
    # Placeholder logic, modify as needed
    return False

def compute_residuals(demand_data, forecast_data, start_yr, end_yr, question=1):
    """
    Compute residuals for error calculation.
    """
    # Placeholder logic, modify as needed
    return []

# Initialize lists to store error metrics
rmse_errors = []
mape_errors = []
mae_errors = []

# Iterate over alpha values from 0.1 to 1.0
for i, alpha in enumerate(range(1, 11)):
    alpha /= 10  # Convert alpha to float
    smoothed_series = simple_exponential_smoothing(U_style, alpha)

    # Calculate error metrics for the current alpha
    rmse = calculate_root_mean_squared_error(U_style, smoothed_series, 2019, 2022)
    mape = calculate_mean_abs_percentage_error(U_style, smoothed_series, 2019, 2022)
    mae = calculate_mean_abs_error(U_style, smoothed_series, 2019, 2022)

    # Append error metrics to respective lists
    rmse_errors.append(rmse)
    mape_errors.append(mape)
    mae_errors.append(mae)

# Find the alpha values corresponding to minimum error metrics
min_rmse_alpha = (rmse_errors.index(min(rmse_errors)) + 1) / 10
min_mape_alpha = (mape_errors.index(min(mape_errors)) + 1) / 10
min_mae_alpha = (mae_errors.index(min(mae_errors)) + 1) / 10

# Find the minimum error metrics
min_rmse = min(rmse_errors)
min_mape = min(mape_errors)
min_mae = min(mae_errors)
```

We believe that this code should be sufficient in applying triple exponential smoothing, however, have been unable to get it running without errors.Therefore we are attaching our code and proceeding forward. We make the assumption that since the data has no seasonality, trend or any other factor usually fixed by applying triple exponential smoothing, there won't be any differences in our answer to part j.

j)

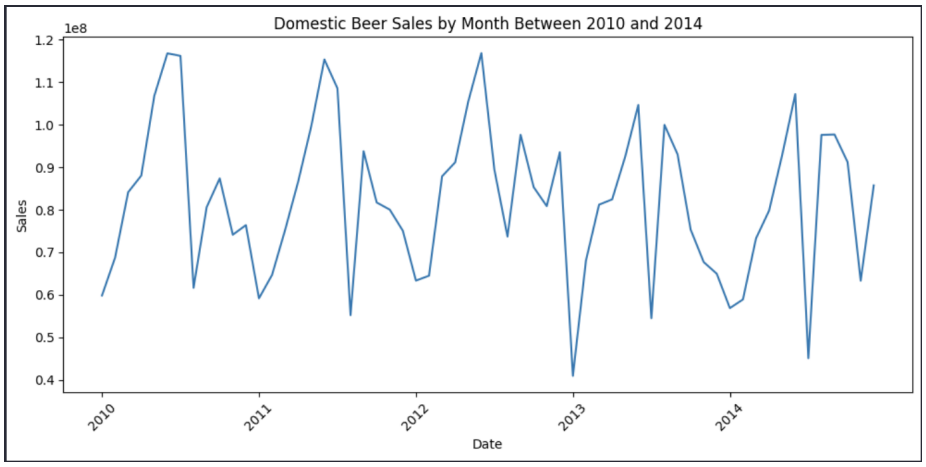
Method	Specification	RMSE	MAPE
Benchmark-1	tau =1	782.6	109.73%
Benchmark-2	tau =12	832.7	88.86%
MA-3		649.33	82.31%
ES	alpha=0.6	743.72	102.06%
DES	alpha* = 0.5 beta* =0.1	725.3	95.17%
Seasonal			

As our code for question 1 i) kept producing errors we decided not to enter it. That being said, within our gathered data, MA-3 has the lowest MAPE,

indicating that it is the best forecasting model for the available data. However, this is still generally a high percentage and it is backed by high values of MAE and RMSE. Lack of any trend, seasonality and the general randomness of the data could be the reason for this. Exponential smoothing and Naive Forecast performed especially poorly,the reasons for which could respectively be the lack of a trend and the general weakness of the model.

Question 2

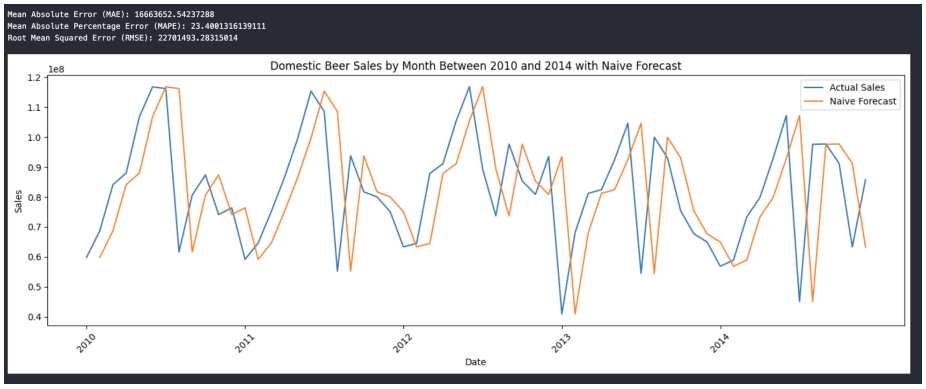
a



The graph shows no trend. There is a strong seasonality that provides yearly (every 12 months) repetition of data patterns.

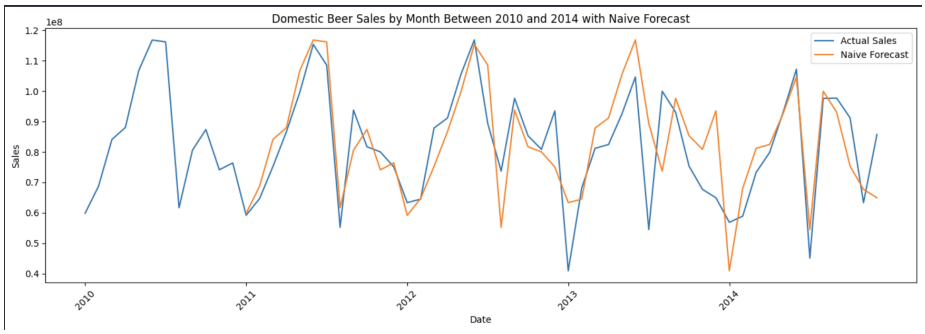
b)

for $\tau = 1$ (the error data is attached with the graph)



```
Mean Absolute Error (MAE): 16663652.54237288
Mean Absolute Percentage Error (MAPE): 23.4001316139111
Root Mean Squared Error (RMSE): 22701493.28315014
```

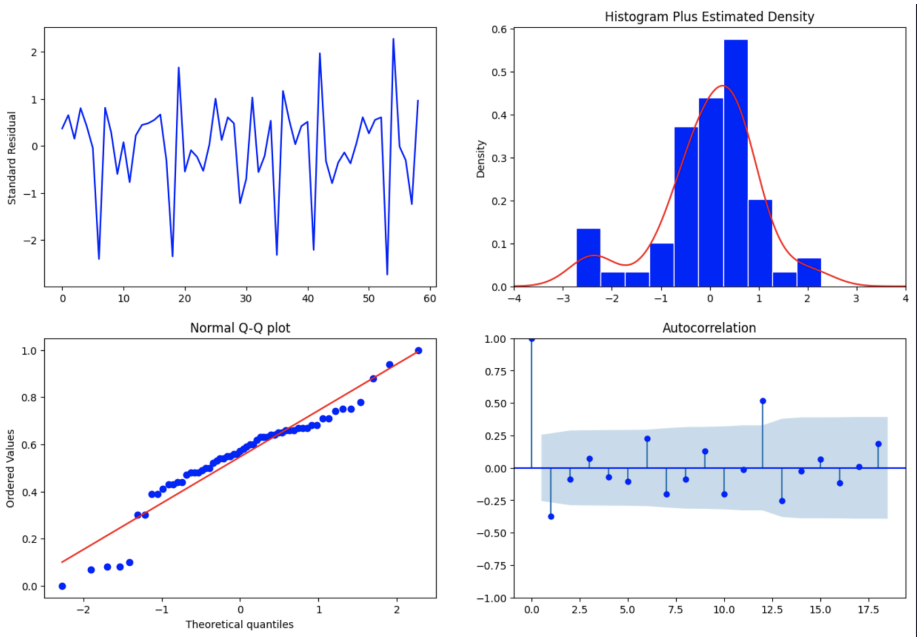
for $\tau = 12$




```
Mean Absolute Error (MAE): 9104424.75
Mean Absolute Percentage Error (MAPE): 12.501022374181908
Root Mean Squared Error (RMSE): 12085064.922952536
```

c)

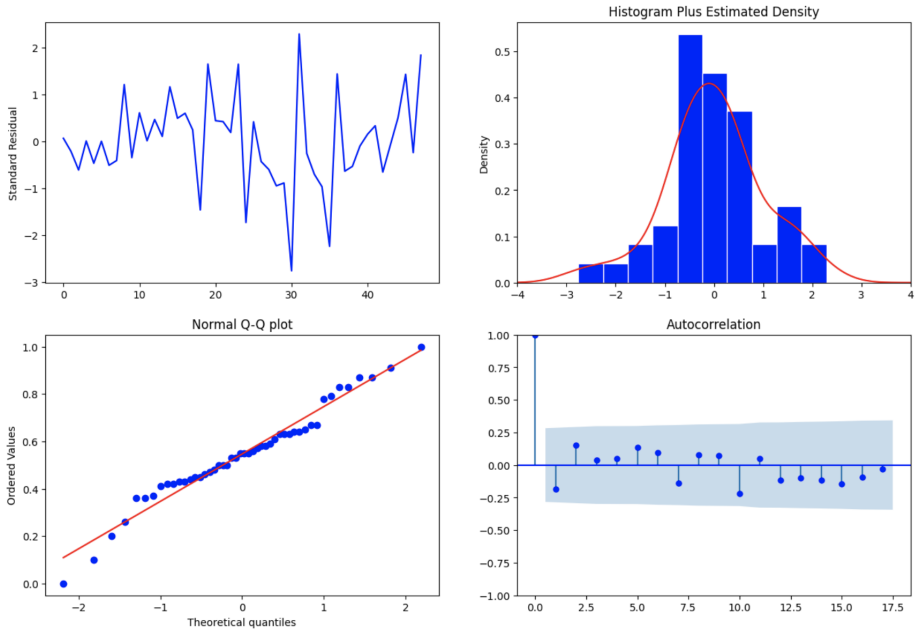
For $\tau = 1$ the residuals are provided below



The normal distribution plot shows that there isn't a normal distribution.

The autocorrelation plot is indicative of the seasonal nature of the data as lag(1)and lag(12) present outliers with significant statistical correlation between them.

The occurrence observed in the autocorrelation provides the drawback of this method, as the seasonality ensures that residuals observed will reoccur with t = 12 long reoccurrence periods.
For $\tau = 12$ the residuals are provided below



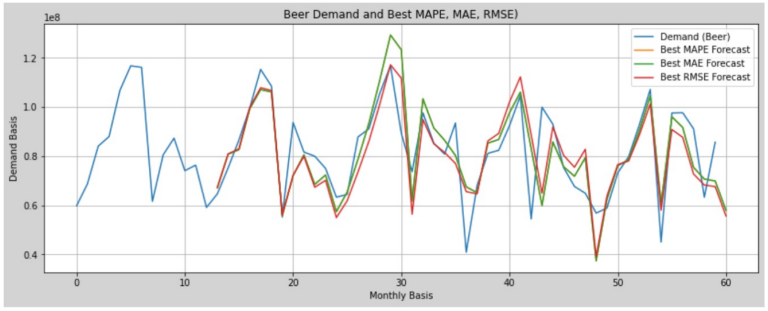
As expected of data said with a seasonality that reoccurs with 12 monthly intervals, this data provides far fewer outliers in the autocorrelation plot, indicating that the data is independent.

Normality plot again is indicative that normal distribution is not occurring.

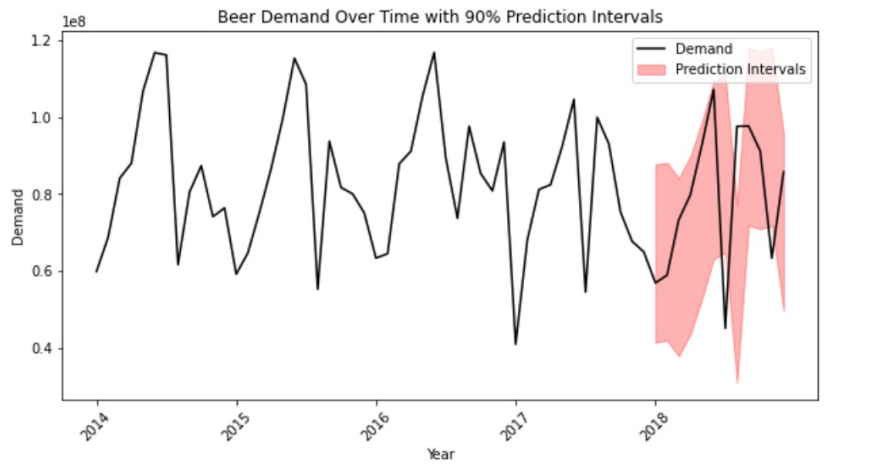
This method of forecasting fits the data well, so there are no clear drawbacks.

d)

```
Best MAPE (Beer): 13.439903421307337
Alpha (MAPE): 0.1
Beta (MAPE): 0.4
Gamma (MAPE): 0.9
Best RMSE (Beer): 13397575.39660953
Alpha (RMSE): 0.1
Beta (RMSE): 0.1
Gamma (RMSE): 0.9
Best MAE (Beer): 9619281.001392981
Alpha (MAE): 0.1
Beta (MAE): 0.4
Gamma (MAE): 0.9
```




```
90% Confidence Prediction Intervals for Beer:
Forecast: 59832432, Interval: (41420460.01267411, 87715947.39059667)
Forecast: 68727007, Interval: (41880943.13034581, 88176430.50826837)
Forecast: 84126559, Interval: (37821429.05419558, 84116916.43211813)
Forecast: 88028614, Interval: (43586537.45637606, 89882024.83429861)
Forecast: 106798913, Interval: (52760548.37329979, 99056035.75122236)
Forecast: 116806052, Interval: (63105032.58396992, 109400519.96189249)
Forecast: 116201586, Interval: (64740681.91089176, 111036169.28881432)
Forecast: 61627436, Interval: (30954468.02736136, 77249955.40528391)
Forecast: 80596102, Interval: (71873369.4909176, 118168856.86884016)
Forecast: 87376160, Interval: (70895883.22586642, 117191370.60378899)
Forecast: 74128665, Interval: (71802230.89663805, 118097718.27456062)
Forecast: 76384870, Interval: (49600002.90563309, 95895490.28355566)
```



We have provided α,β, and γ values determined to be ideal based on the different error metrics above. It should be noted that excluding a minor outlier (that isn't surprising considering the way sudden value data shifts can be a weakness of the exponential smoothing models) all of our observed demand is within our prediction interval

e)

(We will be attaching the MAPE, MAE, and RMSE values in written form as the Python terminal output was extremely difficult to read when screenshot, and any attempts to fix this and make it more legible caused errors.)

Here we can see the demand versus forecast for tau=3 and tau=6. The best alpha, beta, and gamma values found in the previous question are used.
$\alpha = 0.1$
$\beta = 0.4$
$\gamma = 0.9$

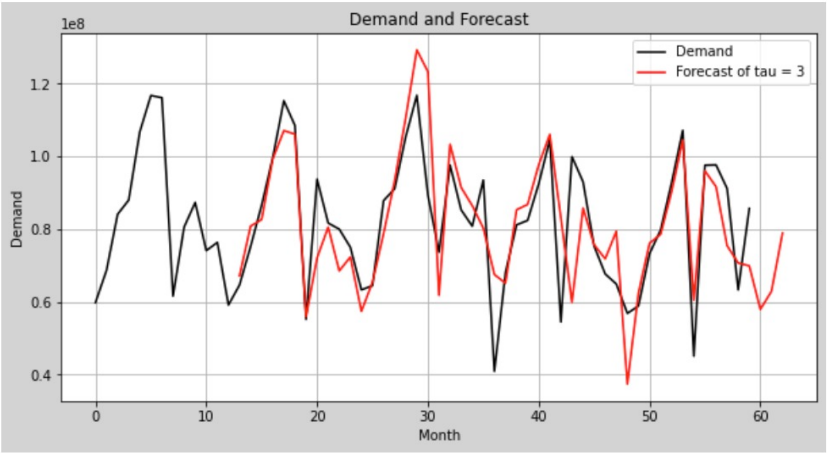
Tau 3 Forecast means 3 Months ahead TES
Tau 6 Forecast means 6 Months ahead TES

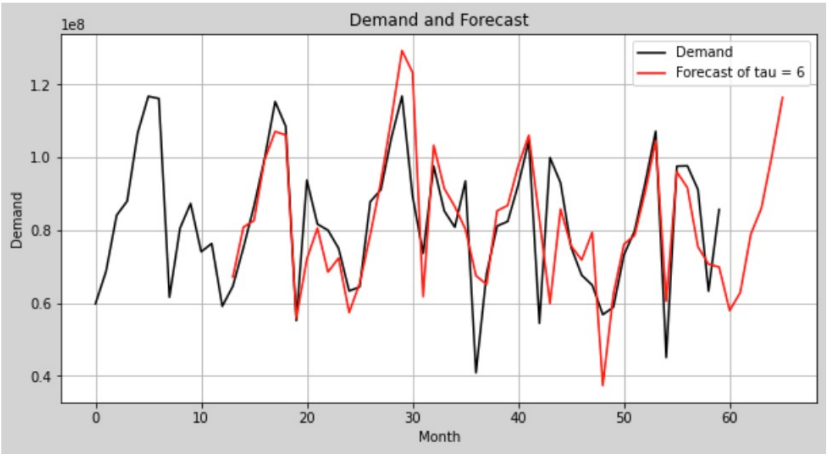
Forecasts are similar to each other from 0 to 60 months. It is because of the triple exponential smoothing process which uses seasonality. Looking at less than 12 months periods would give us similar forecasts. That's why 3-month and 6-month forecasts are similar.

Error Metrics for 3 Months:
MAPE: 13.44
RMSE: 13547209.42

MAE: 9619281.00

Error Metrics for 6 Months:
MAPE: 13.44
RMSE: 13547209.42
MAE: 9619281.00





f)

Method	Specification	RMSE	MAPE
Benchmark-1	tau =1	22701493.28	23.4%
Benchmark-2	tau =12	12085064.92	12.5%
TES	$\alpha = 0.1, \beta = 0.4, \gamma = 0.9$	13397575.39	13.44%

Considering the 12 monthly seasonality of the data, the 12-month naive forecast having the lowest MAPE is not a massive shock. However, the fact that it outperformed triple exponential smoothing is interesting. The reason why this generally unreliable model has outperformed the much more complex triple exponential model could be that unlike triple exponential smoothing the naive forecast doesn't try to account for (in this case nonexisting) trends, which could have skewed the forecasts of the TES to having higher errors.This would also explain why the difference between their MAPE's are so little.