Q1

```
%pip install pandas
%pip install prophet
%pip install -U scikit-learn

import numpy as np
import pandas as pd
import prophet
import matplotlib.pyplot as plt
import scipy.optimize as opt
```

```
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packages (2.0.3)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.10/dist-packages (from pandas) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas) (2023.4)
Requirement already satisfied: tzdata>=2022.1 in /usr/local/lib/python3.10/dist-packages (from pandas) (2024.1)
Requirement already satisfied: numpy>=1.21.0 in /usr/local/lib/python3.10/dist-packages (from pandas) (1.25.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.8.2->pandas) (1.16.0)
Requirement already satisfied: prophet in /usr/local/lib/python3.10/dist-packages (1.1.5)
Requirement already satisfied: cmdstanpy>=1.0.4 in /usr/local/lib/python3.10/dist-packages (from prophet) (1.2.2)
Requirement already satisfied: numpy>=1.15.4 in /usr/local/lib/python3.10/dist-packages (from prophet) (1.25.2)
Requirement already satisfied: matplotlib>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from prophet) (3.7.1)
Requirement already satisfied: pandas>=1.0.4 in /usr/local/lib/python3.10/dist-packages (from prophet) (2.0.3)
Requirement already satisfied: holidays>=0.25 in /usr/local/lib/python3.10/dist-packages (from prophet) (0.49)
Requirement already satisfied: tqdm>=4.36.1 in /usr/local/lib/python3.10/dist-packages (from prophet) (4.66.4)
Requirement already satisfied: importlib-resources in /usr/local/lib/python3.10/dist-packages (from prophet) (6.4.0)
Requirement already satisfied: stanio<2.0.0,>=0.4.0 in /usr/local/lib/python3.10/dist-packages (from cmdstanpy>=1.0.4->prophet) (0.5.0)
Requirement already satisfied: python-dateutil in /usr/local/lib/python3.10/dist-packages (from holidays>=0.25->prophet) (2.8.2)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=2.0.0->prophet) (1.2.1)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=2.0.0->prophet) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=2.0.0->prophet) (4.51.0)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=2.0.0->prophet) (1.4.5)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=2.0.0->prophet) (24.0)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=2.0.0->prophet) (9.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=2.0.0->prophet) (3.1.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=1.0.4->prophet) (2023.4)
Requirement already satisfied: tzdata>=2022.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=1.0.4->prophet) (2024.1)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil->holidays>=0.25->prophet) (1.16.0)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.10/dist-packages (1.2.2)
Collecting scikit-learn
  Downloading scikit_learn-1.5.0-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (13.3 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 13.3/13.3 MB 58.5 MB/s eta 0:00:00
Requirement already satisfied: numpy>=1.19.5 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (1.25.2)
Requirement already satisfied: scipy>=1.6.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (1.11.4)
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (1.4.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (3.5.0)
Installing collected packages: scikit-learn
  Attempting uninstall: scikit-learn
    Found existing installation: scikit-learn 1.2.2
    Uninstalling scikit-learn-1.2.2:
      Successfully uninstalled scikit-learn-1.2.2
Successfully installed scikit-learn-1.5.0
```

```
#Regression function

from sklearn import linear_model
reg = linear_model.LinearRegression()

#data = pd.read_csv('/SmallBusiness.csv"')
#years = data['Year']
#sales = data['Sales']

years = np.array((range(1,21)))
sales = np.array ([283, 288, 336, 388, 406, 412, 416, 435, 428, 435, 462, 452, 474, 476, 497, 487, 523, 528, 532, 552])

df = pd.DataFrame.from_dict({'Year':years,'Sales':sales})
print(df)

reg = reg.fit(df[['Year']], df[['Sales']])
print('intercept:', reg.intercept_)#b0
print('coefficient:', reg.coef_)#b1
print(reg.score(df[['Year']], df['Sales']))#R^2
```

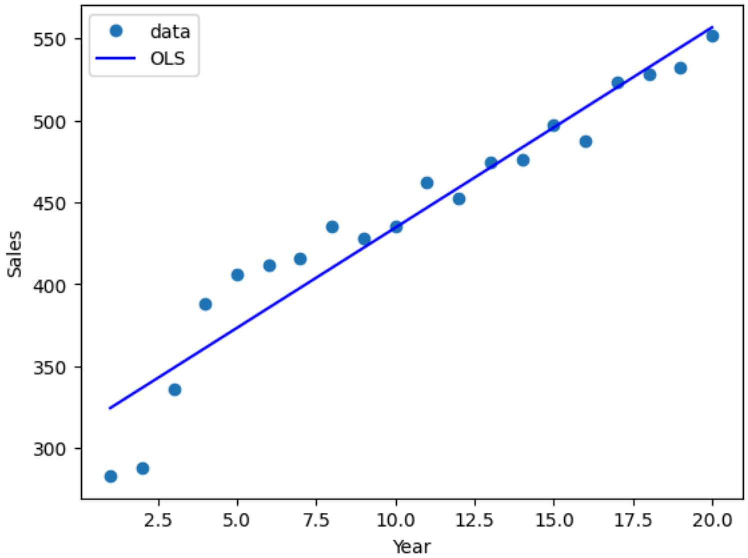
```
   Year  Sales
0     1    283
1     2    288
2     3    336
3     4    388
4     5    406
5     6    412
6     7    416
7     8    435
8     9    428
```

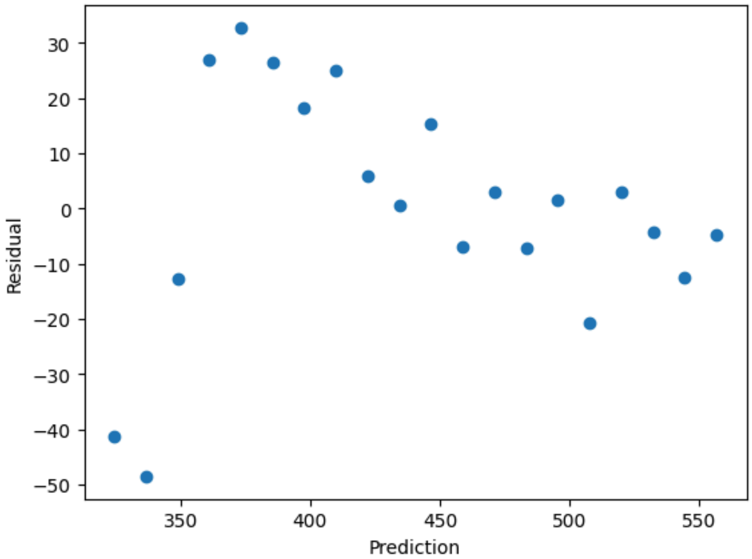
```
9     10    435
10    11    462
11    12    452
12    13    474
13    14    476
14    15    497
15    16    487
16    17    523
17    18    528
18    19    532
19    20    552
intercept: [312.1]
coefficient: [[12.22857143]]
0.9194295594103281
```

```
#calculate the prediction (use the function above)
df['Prediction']=reg.predict(df[['Year']])
#calculate error (actual value - predict value)
df['Error']=df['Sales']-df['Prediction']
print(df)
```

```
    Year  Sales  Prediction      Error
0      1    283  324.328571 -41.328571
1      2    288  336.557143 -48.557143
2      3    336  348.785714 -12.785714
3      4    388  361.014286  26.985714
4      5    406  373.242857  32.757143
5      6    412  385.471429  26.528571
6      7    416  397.700000  18.300000
7      8    435  409.928571  25.071429
8      9    428  422.157143   5.842857
9     10    435  434.385714   0.614286
10    11    462  446.614286  15.385714
11    12    452  458.842857  -6.842857
12    13    474  471.071429   2.928571
13    14    476  483.300000  -7.300000
14    15    497  495.528571   1.471429
15    16    487  507.757143 -20.757143
16    17    523  519.985714   3.014286
17    18    528  532.214286  -4.214286
18    19    532  544.442857 -12.442857
19    20    552  556.671429  -4.671429
```

```
plt.plot (df['Year'], df['Sales'], "o", label="data")
plt.plot (df['Year'], df['Prediction'], "b-", label="OLS") #Ordinary Least Squares
plt.legend (loc="best")
plt.xlabel ('Year')
plt.ylabel ('Sales')
plt.show ()

plt.plot (df['Prediction'], df['Error'],"o")
plt.xlabel ("Prediction")
plt.ylabel ("Residual") #The errors
plt.show ()
```






```
#Exponential smoothing
#from scipy.optimize import minimize

def exp(sales, extra_periods=1, alpha = 0.5):

    cols = len(sales)
    sales = np.append(sales,[np.nan]*extra_periods)

    f = np.full(cols+extra_periods,np.nan)
    f[1] = sales[0]

    for t in range(2, cols+1):
        f[t] = alpha*sales[t-1] + (1-alpha)*f[t-1]

    # forecast
    for t in range(cols+1, cols+extra_periods):
        f[t] = f[t-1]

    df = pd.DataFrame.from_dict({'Sales':sales,'Forecast':f,'ES_Error':sales-f})

    return df

def exp_smooth_opti(sales, extra_periods = 6):

    params = []  # contains all the different parameter sets
    KPIs = []    # contains the results of each model
    dfs = []   # contains all the dataframes returned by the different models

    for alpha in [0.05, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]:

        df = exp(sales,extra_periods=extra_periods,alpha=alpha)
        params.append(f'Simple Smoothing, alpha: {alpha}')
        dfs.append(df)
        MAE = df['ES_Error'].abs().mean()
        KPIs.append(MAE)
```

```
        mini = np.argmin(KPIs)
        print(f'Best solution found for {params[mini]} MAE of',round(KPIs[mini],2))

        return dfs[mini]

df2 = exp_smooth_opti(sales)

print (df2)
```

```
Best solution found for Simple Smoothing, alpha: 0.9 MAE of 17.73
    Sales    Forecast   ES_Error
0   283.0         NaN        NaN
1   288.0  283.000000   5.000000
2   336.0  287.500000  48.500000
3   388.0  331.150000  56.850000
4   406.0  382.315000  23.685000
5   412.0  403.631500   8.368500
6   416.0  411.163150   4.836850
7   435.0  415.516315  19.483685
8   428.0  433.051631  -5.051631
9   435.0  428.505163   6.494837
10  462.0  434.350516  27.649484
11  452.0  459.235052  -7.235052
12  474.0  452.723505  21.276495
13  476.0  471.872351   4.127649
14  497.0  475.587235  21.412765
15  487.0  494.858724  -7.858724
16  523.0  487.785872  35.214128
17  528.0  519.478587   8.521413
18  532.0  527.147859   4.852141
19  552.0  531.514786  20.485214
20    NaN  549.951479        NaN
21    NaN  549.951479        NaN
22    NaN  549.951479        NaN
23    NaN  549.951479        NaN
24    NaN  549.951479        NaN
25    NaN  549.951479        NaN
```

```
SSE_reg = (df['Error']**2).sum()
SSE_ES = (df2['ES_Error']**2).sum()

print(SSE_reg)
print(SSE_ES)
```

```
8714.257142857146
10273.563718144005
```

Answer

(a) According to the above code and result, b0 (intercept = 312.1) and b1 (coefficient = 12.22857143), so the estimated regression function will be : (hat)Y = 312.1 + 12.22857143*x

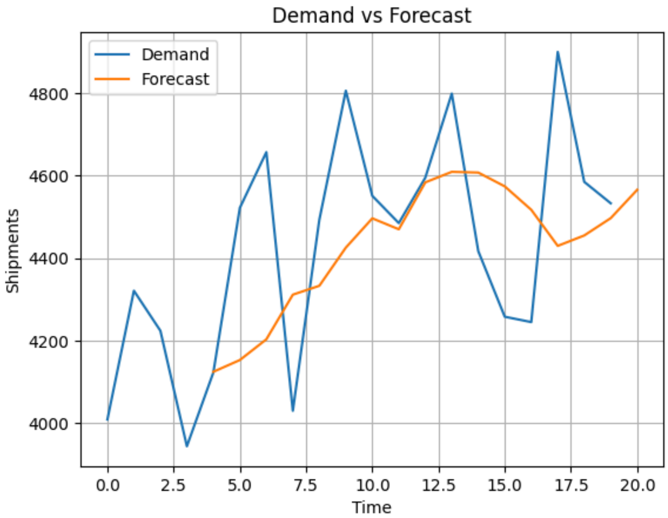
(b) R^2 is the value which we use to measure how well the regression line approximate the real data in statistical. It ranges from 0 to 1. If the value are close to 1, which indicates that this regression line can fits the data better. Otherwise, the regression can not explain the variables of the response data around its mean at all. In this case, the value of R^2 is 0.9194295594103281 (according to the above code). It means that about 91.94% of the variance in this data can be explained by the time value (x:years).

(c) First, indentify the value of alpha (smoothing constant). We use the optimal alpha (0.9), which we found in the code. Then, adopt the first sales value in actual data as the initial forecast (F1). For each subsequent period, the function is : Ft = a*At-1 + (1-a)*Ft-1, use this function to calculate prediction in each year (like above code and results). For example, the next prediction of F1 is 283, and the second one is 287.5. In linear trend model, we can captures a consistent trend over time. It's useful when data show a linear pattern, and also can see the relation whithin each points. If we use expoential smoothing model, we can adapt quickly to new trend. In this case, SSE of linear trend model is smaller then expoential smoothing model, which means the prediction is closer to its actual value. As a result, we can consider to adopt linear trend model.

2-1

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

data = {
    'Quarter': [
        'Q1-1985', 'Q2-1985', 'Q3-1985', 'Q4-1985',
        'Q1-1986', 'Q2-1986', 'Q3-1986', 'Q4-1986',
        'Q1-1987', 'Q2-1987', 'Q3-1987', 'Q4-1987',
        'Q1-1988', 'Q2-1988', 'Q3-1988', 'Q4-1988',
        'Q1-1989', 'Q2-1989', 'Q3-1989', 'Q4-1989'
    ],
    'Shipments': [
        4009, 4321, 4224, 3944,
        4123, 4522, 4657, 4030,
        4493, 4806, 4551, 4485,
        4595, 4799, 4417, 4258,
        4245, 4900, 4585, 4533
    ]
}

df = pd.DataFrame(data)

def moving_average(d, extra_periods=1, n=3):
    cols = len(d)
    d = np.append(d, [np.nan]*extra_periods)
    f = np.full(cols+extra_periods, np.nan)
    for t in range(n, cols):
        f[t] = np.mean(d[t-n:t])
    f[t+1:] = np.mean(d[t-n+1:t+1])
    df = pd.DataFrame.from_dict({'Demand':d, 'Forecast':f, 'Error':d-f})
    return df

result = moving_average(df['Shipments'].values, extra_periods=1, n=4)

print(result)

result[['Demand', 'Forecast']].plot()
plt.title('Demand vs Forecast')
plt.xlabel('Time')
plt.ylabel('Shipments')
plt.grid(True)
plt.show()
```

```
    Demand  Forecast   Error
0   4009.0       NaN     NaN
1   4321.0       NaN     NaN
2   4224.0       NaN     NaN
3   3944.0       NaN     NaN
4   4123.0   4124.50   -1.50
5   4522.0   4153.00  369.00
6   4657.0   4203.25  453.75
7   4030.0   4311.50 -281.50
8   4493.0   4333.00  160.00
9   4806.0   4425.50  380.50
10  4551.0   4496.50   54.50
11  4485.0   4470.00   15.00
12  4595.0   4583.75   11.25
13  4799.0   4609.25  189.75
14  4417.0   4607.50 -190.50
15  4258.0   4574.00 -316.00
16  4245.0   4517.25 -272.25
17  4900.0   4429.75  470.25
18  4585.0   4455.00  130.00
19  4533.0   4497.00   36.00
20     NaN   4565.75     NaN
```



在1989 Q4 shipment有下降的趨勢，但我們計算出的shipment卻有上升的徵兆，因此有可能會overestimate 1990 Q1的shipment，理由主要是因為利用moving average的算法算出的預測值通常無法及時反映當下的數值，像是1989 Q4 shipment有下降的趨勢，但我們的預測可能要再晚一期的時間才能反應過來。

2-2

```
import pandas as pd
import statsmodels.api as sm

data = {
    'Year': ['1985', '1985', '1985', '1985', '1986', '1986', '1986', '1986', '1987', '1987', '1987', '1987', '1988', '1988', '1988', '1988',
    'Quarter': ['Q1', 'Q2', 'Q3', 'Q4', 'Q1', 'Q2', 'Q3', 'Q4', 'Q1', 'Q2', 'Q3', 'Q4', 'Q1', 'Q2', 'Q3', 'Q4', 'Q1', 'Q2', 'Q3', 'Q4'],
    'Shipments': [4009, 4321, 4224, 3944, 4123, 4522, 4657, 4030, 4493, 4806, 4551, 4485, 4595, 4799, 4417, 4258, 4245, 4900, 4585, 4533]
}
df = pd.DataFrame(data)
df['Year'] = df['Year'].astype(int)
df['Time'] = range(1, len(df) + 1)

seasonal_dummies = pd.get_dummies(df['Quarter']).astype(int)
df = pd.concat([df, seasonal_dummies], axis=1)

train = df[df['Year'] != 1989]
test = df[df['Year'] == 1989]
X_train = sm.add_constant(train[['Time', 'Q1', 'Q2', 'Q3']])
y_train = train['Shipments']
X_test = sm.add_constant(test[['Time', 'Q1', 'Q2', 'Q3']])
y_test = test['Shipments']

model = sm.OLS(y_train, X_train).fit()

predictions = model.predict(X_test)
print(predictions)

print(model.summary())
```

```
16    4662.25
17    4969.25
18    4819.50
19    4536.50
dtype: float64
                            OLS Regression Results
==============================================================================
```

```
Dep. Variable:              Shipments   R-squared:                       0.749
Model:                            OLS   Adj. R-squared:                  0.658
Method:                 Least Squares   F-statistic:                     8.208
Date:                Tue, 28 May 2024   Prob (F-statistic):            0.00255
Time:                        15:04:56   Log-Likelihood:                -100.84
No. Observations:                  16   AIC:                             211.7
Df Residuals:                      11   BIC:                             215.5
Df Model:                           4
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const       3822.0000    119.462     31.994      0.000    3559.067    4084.933
Time          35.7250      8.904      4.012      0.002      16.127      55.323
Q1           232.9250    115.754      2.012      0.069     -21.848     487.698
Q2           504.2000    114.029      4.422      0.001     253.224     755.176
Q3           318.7250    112.981      2.821      0.017      70.055     567.395
==============================================================================
Omnibus:                        0.690   Durbin-Watson:                   1.504
Prob(Omnibus):                  0.708   Jarque-Bera (JB):                0.345
Skew:                           0.345   Prob(JB):                        0.841
Kurtosis:                       2.799   Cond. No.                         45.9
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
/usr/local/lib/python3.10/dist-packages/scipy/stats/_stats_py.py:1806: UserWarning: kurtosistest only valid for n>=20 ... continuing anywa
  warnings.warn("kurtosistest only valid for n>=20 ... continuing "
```

```
df
```

	Year	Quarter	Shipments	Time	Q1	Q2	Q3	Q4
0	1985	Q1	4009	1	1	0	0	0
1	1985	Q2	4321	2	0	1	0	0
2	1985	Q3	4224	3	0	0	1	0
3	1985	Q4	3944	4	0	0	0	1
4	1986	Q1	4123	5	1	0	0	0
5	1986	Q2	4522	6	0	1	0	0
6	1986	Q3	4657	7	0	0	1	0
7	1986	Q4	4030	8	0	0	0	1
8	1987	Q1	4493	9	1	0	0	0
9	1987	Q2	4806	10	0	1	0	0
10	1987	Q3	4551	11	0	0	1	0
11	1987	Q4	4485	12	0	0	0	1
12	1988	Q1	4595	13	1	0	0	0
13	1988	Q2	4799	14	0	1	0	0
14	1988	Q3	4417	15	0	0	1	0
15	1988	Q4	4258	16	0	0	0	1
16	1989	Q1	4245	17	1	0	0	0
17	1989	Q2	4900	18	0	1	0	0
18	1989	Q3	4585	19	0	0	1	0
19	1989	Q4	4533	20	0	0	0	1

```
import numpy as np
import pandas as pd

def holt_winters_multiplicative(d, slen=4, extra_periods=1, alpha=0.4, beta=0.4, gamma=0.3):
    cols = len(d)
    d = np.append(d, [np.nan] * extra_periods)

    f, E, T, S = np.full((4, cols + extra_periods), np.nan)

    for i in range(slen):
        E[i] = d[:slen].mean()  # Initial average
        T[i] = 0  # Initial trend
        S[i] = d[i] / E[i]  # Initial seasonality index

    # t+1 forecast
    for t in range(slen, cols):
        f[t] = (E[t-1] + T[t-1]) * S[t-slen]
        E[t] = alpha * (d[t] / S[t-slen]) + (1-alpha) * (E[t-1] + T[t-1])
        T[t] = beta * (E[t] - E[t-1]) + (1-beta) * T[t-1]
        S[t] = gamma * (d[t] / E[t]) + (1-gamma) * S[t-slen]

    for t in range(cols, cols + extra_periods):
        f[t] = (E[cols-1] + (t-(cols-1)) * T[cols-1]) * S[t-slen]

    df = pd.DataFrame.from_dict({'Demand': d, 'Forecast': f, 'Level': E, 'Trend': T, 'Season': S, 'Error': d - f})

    return df

data = {
```

```
data = {
    'Year': ['1985', '1985', '1985', '1985', '1986', '1986', '1986', '1986', '1987', '1987', '1987', '1987', '1988', '1988', '1988', '1988', '1
    'Quarter': ['Q1', 'Q2', 'Q3', 'Q4', 'Q1', 'Q2', 'Q3', 'Q4', 'Q1', 'Q2', 'Q3', 'Q4', 'Q1', 'Q2', 'Q3', 'Q4', 'Q1', 'Q2', 'Q3', 'Q4'],
    'Shipments': [4009, 4321, 4224, 3944, 4123, 4522, 4657, 4030, 4493, 4806, 4551, 4485, 4595, 4799, 4417, 4258, 4245, 4900, 4585, 4533]
}

shipments = data['Shipments']
print(type(shipments))
result = holt_winters_multiplicative(shipments, slen=4, extra_periods=4, alpha=0.4, beta=0.4, gamma=0.3)
print(result[['Demand', 'Forecast', 'Level', 'Trend', 'Season', 'Error']])
```

```
<class 'list'>
    Demand     Forecast        Level      Trend    Season       Error
0   4009.0          NaN  4124.500000   0.000000  0.971997         NaN
1   4321.0          NaN  4124.500000   0.000000  1.047642         NaN
2   4224.0          NaN  4124.500000   0.000000  1.024124         NaN
3   3944.0          NaN  4124.500000   0.000000  0.956237         NaN
4   4123.0  4009.000000  4171.413744  18.765498  0.976916  114.000000
5   4522.0  4389.808341  4240.651308  38.954324  1.053253  132.191659
6   4657.0  4382.847422  4386.683502  81.785472  1.035374  274.152578
7   4030.0  4272.915902  4366.855726  41.140173  0.946224 -242.915902
8   4493.0  4306.240845  4484.464783  71.727727  0.984412  186.759155
9   4806.0  4798.824280  4558.917674  72.817792  1.053537    7.175720
10  4551.0  4795.576231  4537.247351  35.022546  1.025671 -244.576231
11  4485.0  4326.392544  4639.318468  61.841975  0.952378  158.607456
12  4595.0  4627.878973  4687.800601  56.498038  0.983150  -32.878973
13  4799.0  4998.291812  4668.632799  26.231702  1.045853 -199.291812
14  4417.0  4815.385322  4539.498733 -35.914606  1.009874 -398.385322
15  4258.0  4289.114467  4490.516010 -41.141852  0.951131  -31.114467
16  4245.0  4374.400312  4396.726905 -62.200754  0.977852 -129.400312
17  4900.0  4533.276196  4474.784441  -6.097438  1.060604  366.723804
18  4585.0  4512.810909  4497.280309   5.339885  1.012763   72.189091
19  4533.0  4282.580737  4607.934526  47.465617  0.960913  250.419263
20     NaN  4552.292271          NaN        NaN       NaN         NaN
21     NaN  4987.880050          NaN        NaN       NaN         NaN
22     NaN  4810.961426          NaN        NaN       NaN         NaN
23     NaN  4610.265214          NaN        NaN       NaN         NaN
```

2-3

2-3

```
result_for_hw_model = result['Forecast'][-4:].values
validation = shipments[16:]
hw_mape = np.mean(np.abs((np.array(validation) - result_for_hw_model) / np.array(validation))) * 100
print(f'Holt-Winters MAPE: {hw_mape:.2f}%')
```

```
Holt-Winters MAPE: 3.92%
```

```
linear_mape = np.mean(np.abs((y_test.values - predictions) / y_test.values)) * 100
print(f'Linear Trend and Seasonal Model MAPE: {linear_mape:.2f}%')
```

```
Linear Trend and Seasonal Model MAPE: 4.11%
```

```
if hw_mape < linear_mape:
    print("Holt-Winters 模型更適合預測 Q1-1990 的 shipment")
else:
    print("線性模型更適合預測 Q1-1990 的 shipment")
```

```
Holt-Winters 模型更適合預測 Q1-1990 的 shipment
```

```
import numpy as np
import pandas as pd
import prophet
from google.colab import files
import io
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
import matplotlib.pyplot as plt
import statsmodels.api as sm
import scipy.stats as stats
```

```
# Upload and read the CSV file
uploaded = files.upload()
```

選擇檔案 Duque.csv

- **Duque.csv**(text/csv) - 400 bytes, last modified: 2024/5/29 - 100% done

```
Saving Duque.csv to Duque.csv
```

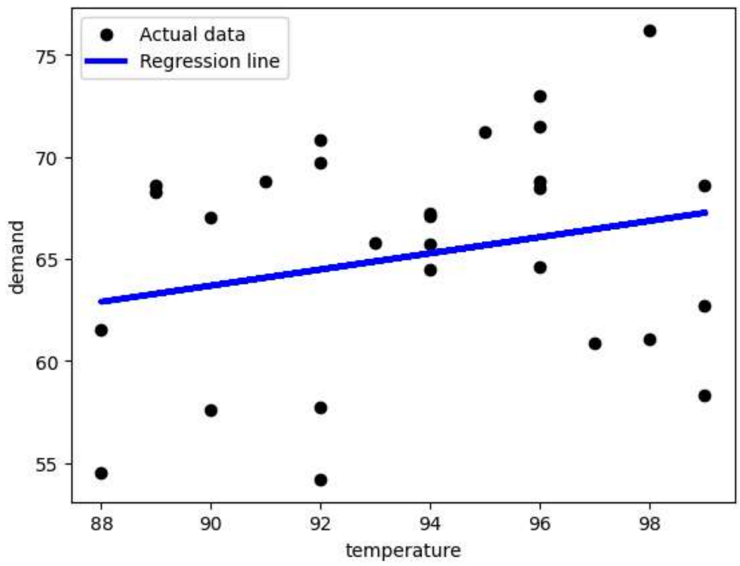
```
df1 = pd.read_csv(io.BytesIO(uploaded['Duque.csv']))
```

```
# Extract temperature and demand data
demand_data = df1.iloc[:, 3].values
temperature_data=df1.iloc[:, 2].values.reshape(-1, 1)
```

Q3(a)

```
X_train, X_test, y_train, y_test = train_test_split(temperature_data, demand_data, test_size=0.2, random_state=0)
simple_regressor = LinearRegression()
simple_regressor.fit(X_train, y_train)
y_pred_simple = simple_regressor.predict(temperature_data)


slope = simple_regressor.coef_[0]
intercept = simple_regressor.intercept_

print(f'Estimated regression equation: y = {slope:.4f} * x + {intercept:.4f}')

plt.scatter(temperature_data, demand_data, color='black', label='Actual data')
plt.plot(temperature_data, y_pred_simple, color='blue', linewidth=3, label='Regression line')
plt.xlabel('temperature')
plt.ylabel('demand')
plt.legend()
plt.show()
```

```
Estimated regression equation: y = 0.3962 * x + 28.0319
```



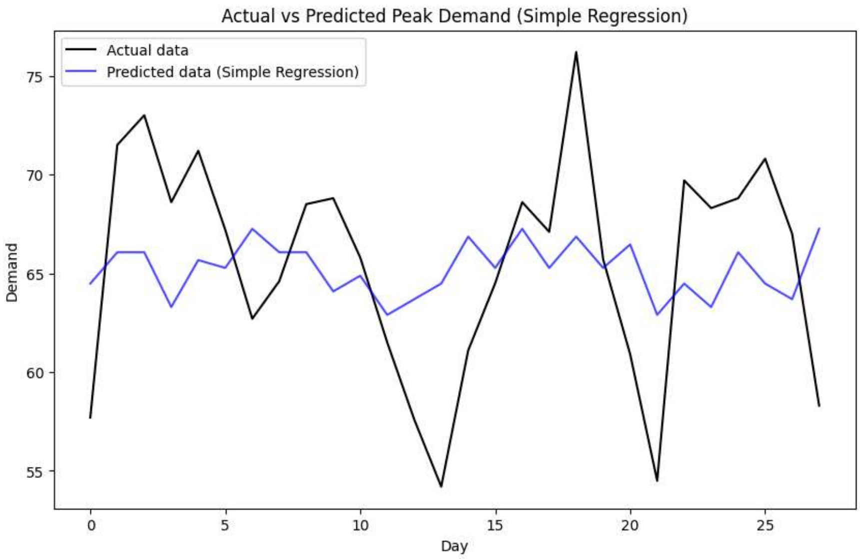
Q3(b)

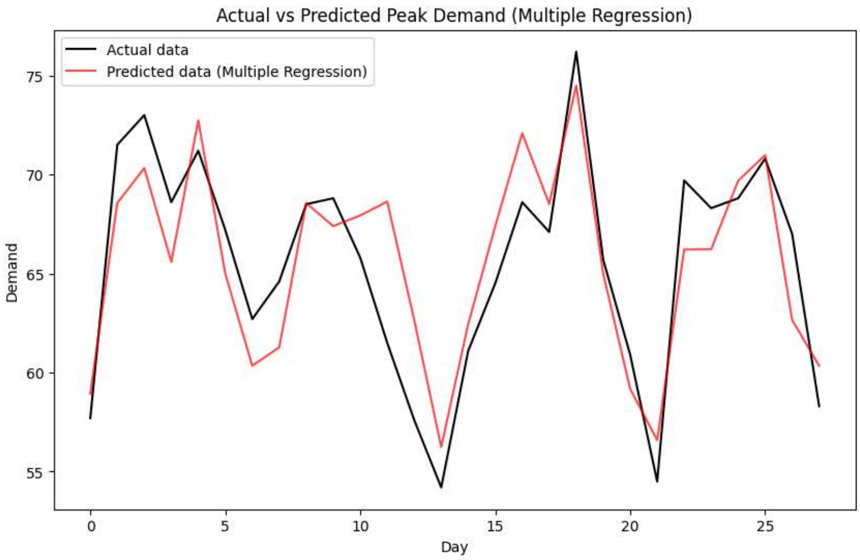
```
# Create dummy variables for the day of the week
df2 = pd.get_dummies(df1, columns=['Day'], drop_first=True)

# Define feature columns for multiple regression
feature_columns = ['Temp'] + [col for col in df2.columns if col.startswith('Day_')]

# Ensure feature columns are correct
print("Feature columns: ", feature_columns)

# Prepare the data for multiple linear regression
X = df2[feature_columns].values
y = df2['Demand'].values

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

# Create and train the multiple regression model
multiple_regressor = LinearRegression()
multiple_regressor.fit(X_train, y_train)

# Estimated regression model
coefficients = multiple_regressor.coef_
intercept = multiple_regressor.intercept_
equation = f"y = {intercept:.4f} + " + " + ".join([f"{coeff:.4f}*{name}" for coeff, name in zip(coefficients, feature_columns)])
print('Estimated regression equation:', equation)

# Interpret the coefficients
for feature, coeff in zip(feature_columns, coefficients):
    print(f'Coefficient for {feature}: {coeff:.4f}')
```

```
Feature columns:  ['Temp', 'Day_2', 'Day_3', 'Day_4', 'Day_5', 'Day_6', 'Day_7']
Estimated regression equation: y = 5.0303 + 0.5859*Temp + 7.2803*Day_2 + 9.0474*Day_3 + 8.4094*Day_4 + 12.0423*Day_5 + 4.8906*Day_6 + -2.6914*Da
Coefficient for Temp: 0.5859
Coefficient for Day_2: 7.2803
Coefficient for Day_3: 9.0474
Coefficient for Day_4: 8.4094
Coefficient for Day_5: 12.0423
Coefficient for Day_6: 4.8906
Coefficient for Day_7: -2.6914
```

Q3(c)

```
y_pred_multiple = multiple_regressor.predict(X)

# Plot the actual vs predicted values for simple regression
plt.figure(figsize=(10, 6))
plt.plot(df1.index, demand_data, color='black', label='Actual data')
plt.plot(df1.index, y_pred_simple, color='blue', label='Predicted data (Simple Regression)', alpha=0.7)
plt.xlabel('Day')
plt.ylabel('Demand')
plt.legend()
plt.title('Actual vs Predicted Peak Demand (Simple Regression)')
plt.show()

# Plot the actual vs predicted values for multiple regression
plt.figure(figsize=(10, 6))
plt.plot(df1.index, demand_data, color='black', label='Actual data')
plt.plot(df1.index, y_pred_multiple, color='red', label='Predicted data (Multiple Regression)', alpha=0.7)
plt.xlabel('Day')
plt.ylabel('Demand')
plt.legend()
plt.title('Actual vs Predicted Peak Demand (Multiple Regression)')
plt.show()

# Evaluate model performance
from sklearn.metrics import mean_absolute_error, mean_squared_error

mae_simple = mean_absolute_error(demand_data, y_pred_simple)
mae_multiple = mean_absolute_error(demand_data, y_pred_multiple)

mse_simple = mean_squared_error(demand_data, y_pred_simple)
mse_multiple = mean_squared_error(demand_data, y_pred_multiple)

rmse_simple = np.sqrt(mse_simple)
rmse_multiple = np.sqrt(mse_multiple)

print(f'Simple Model - MAE: {mae_simple}, MSE: {mse_simple}, RMSE: {rmse_simple}')
print(f'Multiple Model - MAE: {mae_multiple}, MSE: {mse_multiple}, RMSE: {rmse_multiple}')
```






```
Simple Model - MAE: 4.598403138048843, MSE: 28.61984969829599, RMSE: 5.34975230251793
Multiple Model - MAE: 2.338567275094857, MSE: 7.596797956025045, RMSE: 2.75622893752€
```

Q3(d)

```
#跑出回歸式
df_with_dummies = pd.get_dummies(df1, columns=['Day'], drop_first=True)

feature_columns = ['Temp'] + [col for col in df_with_dummies.columns if col.startswith('Day_')]

print("Feature columns: ", feature_columns)

X = df_with_dummies[feature_columns].values
y = df_with_dummies['Demand'].values

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

multiple_regressor = LinearRegression()
multiple_regressor.fit(X_train, y_train)
```

```
coefficients = multiple_regressor.coef_
intercept = multiple_regressor.intercept_
equation = f"y = {intercept:.4f} + " + " + ".join([f"{coeff:.4f}*{name}" for coeff, name in zip(coefficients, feature_columns)])
print('Estimated regression equation:', equation)

#進行新預測
X_new = np.array([[94, 0, 1, 0, 0, 0, 0]]) #溫度94，Day_3的預測
y_pred = multiple_regressor.predict(X_new)[0]
residuals = y_train - multiple_regressor.predict(X_train)
sigma = np.sqrt(np.sum(residuals**2) / (len(y_train) - len(coefficients) - 1))

X_new_with_intercept = np.insert(X_new, 0, 1, axis=1)
X_train_with_intercept = np.insert(X_train, 0, 1, axis=1)
X_train_with_intercept = X_train_with_intercept.astype('float64')
se_pred = np.sqrt(sigma**2 + X_new_with_intercept @ np.linalg.inv(X_train_with_intercept.T @ X_train_with_intercept) @ X_new_with_intercept.T

t_value = stats.t.ppf(0.975, df=len(y_train) - len(coefficients) - 1)
prediction_interval = (y_pred - t_value * se_pred, y_pred + t_value * se_pred)

print(f"預測值: {y_pred:.2f}")

print("範圍：",prediction_interval[0][0],"到",prediction_interval[1][0])
```

```
Feature columns: ['Temp', 'Day_2', 'Day_3', 'Day_4', 'Day_5', 'Day_6', 'Day_7']
Estimated regression equation: y = 5.0303 + 0.5859*Temp + 7.2803*Day_2 + 9.0474*Day_3 + 8.4094*Day_4 + 12.0423*Day_5 + 4.8906*Day_6 + -2.6914*Day
預測值: 69.15
範圍： [62.4041984] 到 [75.90094069]
```

有95%的機率，用電高峰會落在62.404到75.9之間(正負一個標準差)。