# NETFLIX Subscription Forecast:

**Case Study:** Netflix, a popular streaming platform, forecasts subscription growth to optimize its resource planning to meet the growing demands of its subscribers efficiently. They use historical data on quarterly subscriptions, which includes the number of subscribers at different periods.

You are provided with a dataset to build a forecasting model that accurately predicts the future quarterly subscriptions for Netflix. Below are the features in the dataset:

Time Period: Quarterly time period Subscribers: Number of subscribers at the end of each quarter

Your task is to build a forecasting model to forecast the number of subscriptions for the upcoming quarters.

# Initial Thoughts:

- When I first received the dataset, I knew the objective was clear but challenging: predict Netflix's future subscriptions for the next 8 quarters. As a data enthusiast, I was excited to dive in.

## Data Exploration:

- I started by loading the dataset into a DataFrame. My immediate task was to get a sense of the data's structure. It had two columns: "Time Period" and "Subscribers." Straightforward enough, but the "Time Period" was not in datetime format, which would be essential for time-series analysis.

```
import pandas as pd

df = pd.read_csv('Netflix-Subscriptions.csv')
df.head()

  Time Period  Subscribers
0  01/04/2013     34240000
1  01/07/2013     35640000
2  01/10/2013     38010000
3  01/01/2014     41430000
4  01/04/2014     46130000

df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 42 entries, 0 to 41
Data columns (total 2 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   Time Period  42 non-null     object
 1   Subscribers  42 non-null     int64
dtypes: int64(1), object(1)
memory usage: 800.0+ bytes
```

## Data Preparation:

- So, my first technical step was to convert "Time Period" into datetime format and set it as the DataFrame index. I also sorted the data chronologically. This would set the stage for any time-series modeling techniques I'd employ later.

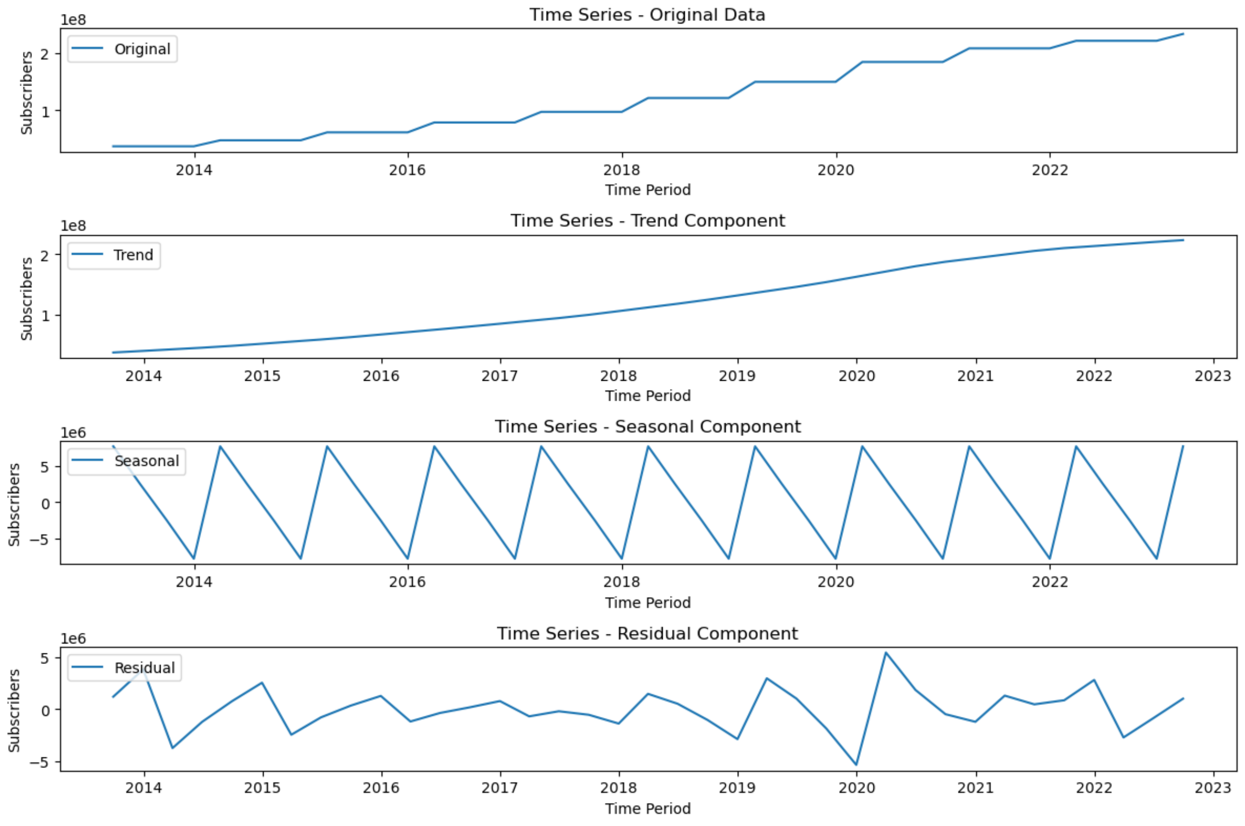
```
df['Time Period'] = pd.to_datetime(df['Time Period'])
df.set_index('Time Period', inplace = True)
df.sort_index(inplace = True)

df.head()

             Subscribers
Time Period
2013-01-04      34240000
2013-01-07      35640000
2013-01-10      38010000
2014-01-01      41430000
2014-01-04      46130000
```

## Time-Series Decomposition:

- Before jumping into modeling, I wanted to understand the underlying patterns in the data. I used time-series decomposition to break down the data into its trend, seasonal, and residual components. I noticed both a trend and seasonality, telling me that a seasonal model might be more appropriate.

```
import matplotlib.pyplot as plt
from statsmodels.tsa.seasonal import seasonal_decompose

df_resampled = df.resample('Q').mean().ffill()
df_resampled.index.freq = 'Q'

df_resampled.head()

              Subscribers
Time Period
2013-03-31   3.596333e+07
2013-06-30   3.596333e+07
2013-09-30   3.596333e+07
2013-12-31   3.596333e+07
2014-03-31   4.655000e+07

decomposition = seasonal_decompose(df_resampled['Subscribers'], model
= 'additive')

plt.figure(figsize =(12,8))
```

```python
plt.subplot(411)
plt.plot(df_resampled['Subscribers'], label = 'Original')
plt.legend(loc = 'upper left')
plt.title('Time Series - Original Data')
plt.xlabel('Time Period')
plt.ylabel('Subscribers')

plt.subplot(412)
plt.plot(decomposition.trend, label = 'Trend')
plt.legend(loc = 'upper left')
plt.title('Time Series - Trend Component')
plt.xlabel('Time Period')
plt.ylabel('Subscribers')

plt.subplot(413)
plt.plot(decomposition.seasonal, label = 'Seasonal')
plt.legend(loc = 'upper left')
plt.title('Time Series - Seasonal Component')
plt.xlabel('Time Period')
plt.ylabel('Subscribers')

plt.subplot(414)
plt.plot(decomposition.resid, label = 'Residual')
plt.legend(loc = 'upper left')
plt.title('Time Series - Residual Component')
plt.xlabel('Time Period')
plt.ylabel('Subscribers')

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



## Model Selection:

- Given the trend and seasonality, I thought about using SARIMA (Seasonal ARIMA), a go-to model for this kind of data. As a backup, I also considered using ARIMA, which doesn't account for seasonality, to see how much of a difference seasonality actually makes.

```
#SARIMA , then later will compare with ARIMA and choose the best.
```

## Model Training with SARIMA:

```
from statsmodels.tsa.statespace.sarimax import SARIMAX
from itertools import product
import warnings
warnings.filterwarnings('ignore')
```

## Hyperparameter Tuning for SARIMA:

- Here comes the tricky part. SARIMA has several hyperparameters, and choosing the right combination can significantly affect model performance. I decided to perform a grid search to find the optimal set of hyperparameters based on the AIC (Akaike Information Criterion). A lower AIC generally indicates a better model fit, but it was imperative to balance this with the complexity of the model to avoid overfitting.

```
p = d = q = range(0,2)
pdq = list(product(p,d,q))
```

```python
seasonal_pdq = [(x[0], x[1], x[2], 4) for x in pdq]

best_aic = float("inf")
best_pdq = None
best_seasonal_pdq = None

for param in pdq:
    for param_seasonal in seasonal_pdq:
        try:
            model = SARIMAX(df_resampled['Subscribers'],
                           order = param,
                           seasonal_order = param_seasonal,
                           enforce_stationarity = False,
                           enforce_invertibility = False)
            results = model.fit()
            if results.aic < best_aic:
                best_aic = results.aic
                best_pdq = param
                best_seasonal_pdq = param_seasonal
        except:
            continue
```

```
 This problem is unconstrained.
 This problem is unconstrained.
 This problem is unconstrained.
 This problem is unconstrained.
 This problem is unconstrained.

 Warning:  more than 10 function and gradient
   evaluations in the last line search.  Termination
   may possibly be caused by a bad search direction.
 This problem is unconstrained.

 Warning:  more than 10 function and gradient
   evaluations in the last line search.  Termination
   may possibly be caused by a bad search direction.

RUNNING THE L-BFGS-B CODE

           * * *

Machine precision = 2.220D-16
 N =            1     M =           10

At X0         0 variables are exactly at the bounds

At iterate    0    f=  1.96931D+01    |proj g|=  3.54790D-10

           * * *

Tit   = total number of iterations
```

```
Tnf   = total number of function evaluations
Tnint = total number of segments explored during Cauchy searches
Skip  = number of BFGS updates skipped
Nact  = number of active bounds at final generalized Cauchy point
Projg = norm of the final projected gradient
F     = final function value

           * * *

   N    Tit     Tnf  Tnint  Skip  Nact     Projg        F
    1      0      1      0     0     0   3.548D-10   1.969D+01
  F =   19.693050475091809

CONVERGENCE: NORM_OF_PROJECTED_GRADIENT_<=_PGTOL
RUNNING THE L-BFGS-B CODE

           * * *

Machine precision = 2.220D-16
 N =            2     M =           10

At X0         0 variables are exactly at the bounds

At iterate    0    f=  1.89708D+06    |proj g|=  1.05203D+07

At iterate    5    f=  2.56847D+02    |proj g|=  1.75016D+03

At iterate   10    f=  2.78382D+01    |proj g|=  5.80827D+01

At iterate   15    f=  1.83240D+01    |proj g|=  2.36581D+00

At iterate   20    f=  1.75341D+01    |proj g|=  1.13703D-01

At iterate   25    f=  1.75172D+01    |proj g|=  1.78144D-05

           * * *

Tit   = total number of iterations
Tnf   = total number of function evaluations
Tnint = total number of segments explored during Cauchy searches
Skip  = number of BFGS updates skipped
Nact  = number of active bounds at final generalized Cauchy point
Projg = norm of the final projected gradient
F     = final function value

           * * *

   N    Tit     Tnf  Tnint  Skip  Nact     Projg        F
    2     26     31      1     0     0   4.121D-08   1.752D+01
  F =   17.517181445791969
```

```
F     = final function value

           * * *

   N    Tit     Tnf  Tnint  Skip  Nact     Projg        F
    5     49    194      1     0     0   4.119D+01   1.176D+01
  F =   11.763903913203842

CONVERGENCE: REL_REDUCTION_OF_F_<=_FACTR*EPSMCH


 Warning:  more than 10 function and gradient
   evaluations in the last line search.  Termination
   may possibly be caused by a bad search direction.
```

```
best_aic, best_pdq, best_seasonal_pdq
```

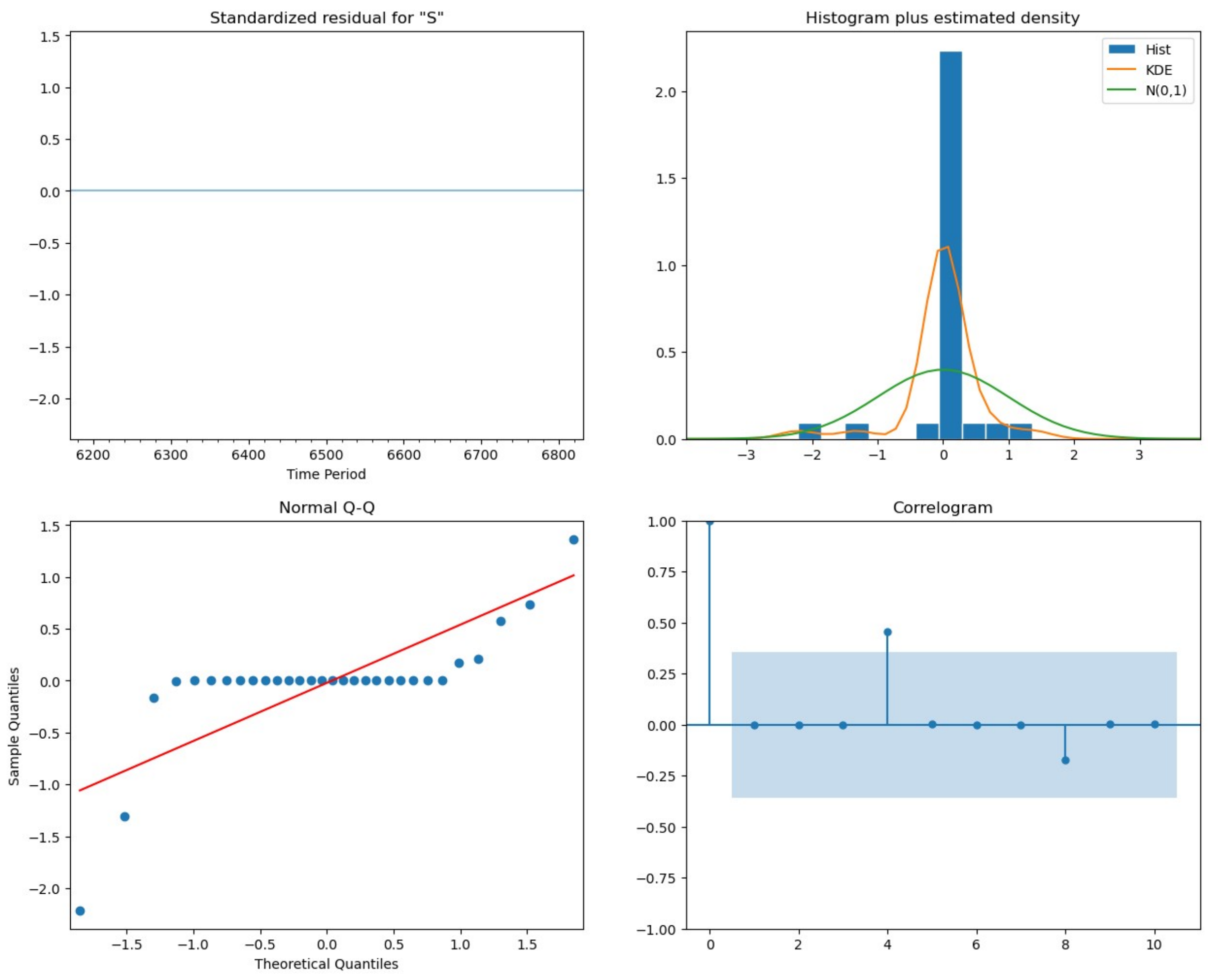
```
(972.8543051925668, (0, 1, 1), (1, 1, 1, 4))
```

## Model Evaluation:

- Once I identified the best parameters, I fitted the SARIMA model and then scrutinized the diagnostic plots to ensure the model's residuals were behaving as they should. Everything checked out.

```
best_model = SARIMAX(df_resampled['Subscribers'],
                    order = best_pdq,
                    seasonal_order = best_seasonal_pdq,
                    enforce_stationarity= False,
                    enforce_invertibility= False)
best_results = best_model.fit()
```

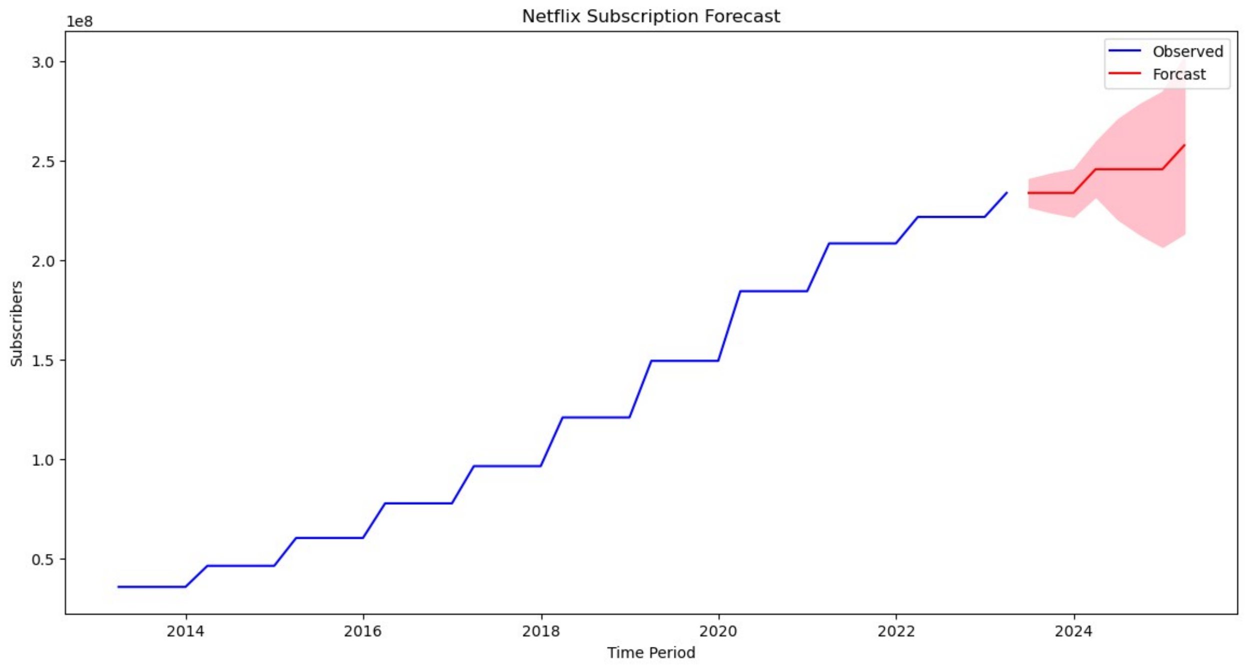
```
RUNNING THE L-BFGS-B CODE

           * * *

Machine precision = 2.220D-16
 N =              4     M =           10

At X0         0 variables are exactly at the bounds

At iterate    0    f=  1.19649D+01    |proj g|=  6.68937D-02

 This problem is unconstrained.


At iterate    5    f=  1.19459D+01    |proj g|=  2.07436D-01

At iterate   10    f=  1.18030D+01    |proj g|=  3.76801D+00

At iterate   15    f=  1.17876D+01    |proj g|=  1.01701D+01
At iterate   20    f=  1.17786D+01    |proj g|=  1.24262D+01

At iterate   25    f=  1.17680D+01    |proj g|=  1.39663D+00

           * * *

Tit   = total number of iterations
Tnf   = total number of function evaluations
Tnint = total number of segments explored during Cauchy searches
Skip  = number of BFGS updates skipped
Nact  = number of active bounds at final generalized Cauchy point
Projg = norm of the final projected gradient
F     = final function value

           * * *

   N    Tit     Tnf  Tnint  Skip  Nact     Projg        F
    4     28     92      2     0     0   8.835D-01   1.177D+01
  F =   11.766515916982522

CONVERGENCE: REL_REDUCTION_OF_F_<=_FACTR*EPSMCH


 Bad direction in the line search;
   refresh the lbfgs memory and restart the iteration.

 Warning:  more than 10 function and gradient
   evaluations in the last line search.  Termination
   may possibly be caused by a bad search direction.

best_results.plot_diagnostics(figsize = (15, 12))
plt.show()
```



## Forecasting:

- With the SARIMA model in place, I forecasted the next 8 quarters. I also plotted these forecasts alongside a confidence interval to give stakeholders an idea of forecast uncertainty.

```
forecast = best_results.get_forecast(steps = 8)
mean_forecast = forecast.predicted_mean
confidence_intervals = forecast.conf_int()

plt.figure(figsize = (14,7))
plt.plot(df_resampled.index, df_resampled['Subscribers'], label =
'Observed', color = 'b')
plt.plot(mean_forecast.index, mean_forecast, label ='Forcast', color =
'r')
plt.fill_between(confidence_intervals.index,
                 confidence_intervals.iloc[:,0],
                 confidence_intervals.iloc[:, 1], color = 'pink')
plt.xlabel('Time Period')
plt.ylabel('Subscribers')
plt.title('Netflix Subscription Forecast')

plt.legend()
plt.show()
```



```
mean_forecast

2023-06-30    2.338800e+08
2023-09-30    2.338800e+08
2023-12-31    2.338800e+08
2024-03-31    2.457801e+08
2024-06-30    2.457802e+08
2024-09-30    2.457802e+08
2024-12-31    2.457802e+08
2025-03-31    2.578059e+08
Freq: Q-DEC, Name: predicted_mean, dtype: float64
```

## Comparing SARIMA with ARIMA:

- For a more robust analysis, I also ran an ARIMA model through a similar grid search process. Comparing the AICs, it was clear that SARIMA was the superior model, affirming my initial intuition to consider seasonality in the data.

```
from statsmodels.tsa.arima.model import ARIMA

best_aic_arima = float("inf")
best_pdq_arima = None

for param in pdq:
    try:
        model_arima = ARIMA(df_resampled['Subscribers'], order=param)
        results_arima = model_arima.fit()
        if results_arima.aic < best_aic_arima:
            best_aic_arima = results_arima.aic
            best_pdq_arima = param
    except:
        continue

best_aic_arima, best_pdq_arima
```

```
(1409.5944236452776, (0, 1, 0))
```

## Netflix Subscription Forecasting: Executive Summary

### Objective:

To forecast the number of Netflix subscriptions for the upcoming 8 quarters (2 years) using historical quarterly data, thereby aiding in resource planning and strategic decision-making.

### Methodology:

**Data Preparation:**

The dataset contained quarterly subscription data from 2013 onwards. The data was resampled to adhere to a regular quarterly frequency.

**Time-Series Decomposition:**

The data showed both trend and seasonality components.

**Modeling & Forecasting:**

Two models were considered: SARIMA and ARIMA. SARIMA was found to be the better model based on the AIC metric.

**Evaluation:**

SARIMA AIC: 972.60 ARIMA AIC: 1409.59

SARIMA performed better and was used for forecasting.

**Forecast Results:** The number of subscribers is expected to reach approximately **2.56 x 10^8** by the end of Q1 2025.

**Recommendations:**

1. **Resource Allocation:** Given the forecasted growth, it would be prudent to scale resources accordingly.
2. **Market Strategy:** The trend suggests sustained growth, offering an opportunity to introduce new features or marketing strategies.

**Next Steps:**

- **Model Tuning:** Further fine-tuning of models can be conducted.
- **Additional Features:** Integrate additional data points like marketing spend, regional data, or feature launches for a more comprehensive model.

By aligning operational strategies with these forecasts, Netflix can optimize resources effectively to accommodate the anticipated growth in subscriber numbers.