Batch :- D-2 **Roll No. :-** 16010122151

Experiment :- 06

TITLE : To perform time series analysis on health care

AIM: To perform forecasting using time series analysis

Expected OUTCOME of Experiment:

CO4: Perform Time series Analytics and forecasting

Books/ Journals/ Websites referred:

Students have to list.

Pre Lab/ Prior Concepts:

Students should have a basic understanding of: Time series Analytics and forecasting

Implementation details:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1821 entries, 0 to 1820
Data columns (total 6 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   DATE     1821 non-null   object
 1   MinTemp  1821 non-null   float64
 2   MaxTemp  1821 non-null   float64
 3   AvgTemp  1821 non-null   float64
 4   Sunrise  1821 non-null   int64
 5   Sunset   1821 non-null   int64
dtypes: float64(3), int64(2), object(1)
memory usage: 85.5+ KB
```

```
df.head()
```

	DATE	MinTemp	MaxTemp	AvgTemp	Sunrise	Sunset
0	1/1/2014	33.0	46.0	40.0	657	1756
1	1/2/2014	35.0	50.0	43.0	657	1756
2	1/3/2014	36.0	45.0	41.0	657	1757
3	1/4/2014	32.0	41.0	37.0	658	1757
4	1/5/2014	24.0	38.0	31.0	658	1758

\

Next steps: Generate code with df | View recommended plots | New interactive sheet

```
df=df[["DATE","AvgTemp"]]
df.head()
```

	DATE	AvgTemp
0	1/1/2014	40.0
1	1/2/2014	43.0
2	1/3/2014	41.0
3	1/4/2014	37.0
4	1/5/2014	31.0

Next steps: Generate code with df | View recommended plots | New interactive sheet

0

Change Column Names for FB Prophet

```
[7] df.columns = ['ds','y']
```

```
df['ds'] = pd.to_datetime(df['ds'])
df.tail()
```

	ds	y
1816	2018-12-26	40.0
1817	2018-12-27	39.0
1818	2018-12-28	40.0
1819	2018-12-29	42.0
1820	2018-12-30	46.0

2182	2019-12-27	47.282039	39.145026	47.723931	46.386911	48.133276	-3.964542	-3.964542	-3.964542	0.313683
2183	2019-12-28	47.283639	38.710468	47.834916	46.383824	48.139061	-4.263419	-4.263419	-4.263419	0.022770
2184	2019-12-29	47.285239	38.891326	46.957960	46.380738	48.144700	-4.383681	-4.383681	-4.383681	-0.115247
2185	2019-12-30	47.286838	38.748876	47.286797	46.377652	48.149515	-4.282685	-4.282685	-4.282685	-0.057084

[15]
```
forecast[['ds', 'yhat', 'yhat_lower', 'yhat_upper']].tail()
```

	ds	yhat	yhat_lower	yhat_upper
2181	2019-12-26	43.081677	38.828041	47.121860
2182	2019-12-27	43.317497	39.145026	47.723931
2183	2019-12-28	43.020220	38.710468	47.834916
2184	2019-12-29	42.901558	38.891326	46.957960
2185	2019-12-30	43.004153	38.748876	47.286797

```
df.tail()
```

	ds	y
1816	2018-12-26	40.0
1817	2018-12-27	39.0
1818	2018-12-28	40.0
1819	2018-12-29	42.0
1820	2018-12-30	46.0

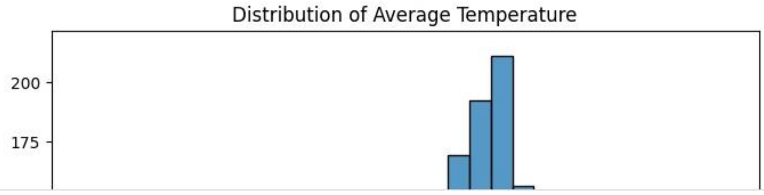
[13]
```
m = Prophet()
m.fit(df)
future = m.make_future_dataframe(periods=365) #MS for monthly, H for hourly
forecast = m.predict(future)
```

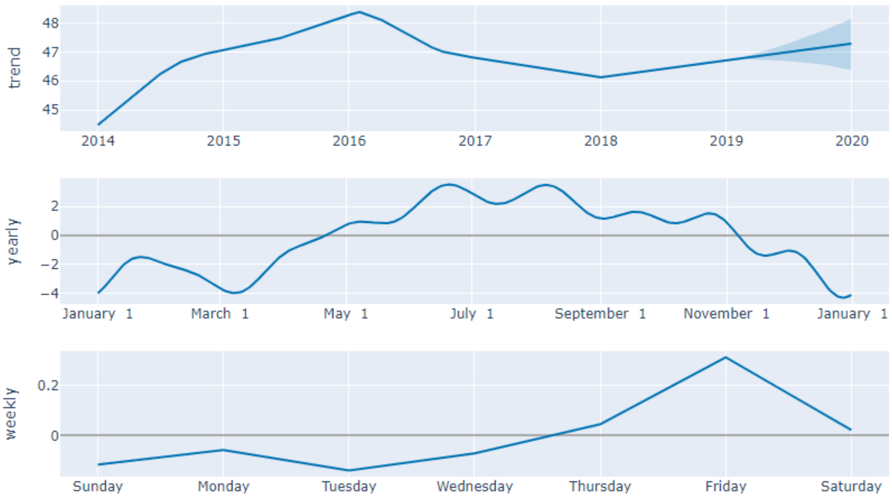
```
INFO:prophet:Disabling daily seasonality. Run prophet with daily_seasonality=True to override this.
DEBUG:cmdstanpy:input tempfile: /tmp/tmpscq3ocnp/iubeft1g.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpscq3ocnp/xbghidbm.json
DEBUG:cmdstanpy:idx 0
DEBUG:cmdstanpy:running CmdStan, num_threads: None
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.10/dist-packages/prophet/stan_model/prophet_model.bin', 'random', 'seed=99390', 'data', 'file=/tmp/tmpscq3ocnp/iubeft1g.json', 'init=/tmp/t
11:24:09 - cmdstanpy - INFO - Chain [1] start processing
INFO:cmdstanpy:Chain [1] start processing
11:24:09 - cmdstanpy - INFO - Chain [1] done processing
INFO:cmdstanpy:Chain [1] done processing
```

```
forecast.tail()
```

	ds	trend	yhat_lower	yhat_upper	trend_lower	trend_upper	additive_terms	additive_terms_lower	additive_terms_upper	weekly	weekly_lower	weekly_upper	yearly	yearly_lower
2181	2019-12-26	47.280439	38.828041	47.121860	46.389895	48.126045	-4.198762	-4.198762	-4.198762	0.045622	0.045622	0.045622	-4.244385	-4.244385
2182	2019-12-27	47.282039	39.145026	47.723931	46.386911	48.133276	-3.964542	-3.964542	-3.964542	0.313683	0.313683	0.313683	-4.278224	-4.278224
2183	2019-12-28	47.283639	38.710468	47.834916	46.383824	48.139061	-4.263419	-4.263419	-4.263419	0.022770	0.022770	0.022770	-4.286189	-4.286189
2184	2019-12-29	47.285239	38.891326	46.957960	46.380738	48.144700	-4.383681	-4.383681	-4.383681	-0.115247	-0.115247	-0.115247	-4.268433	-4.268433
2185	2019-12-30	47.286838	38.748876	47.286797	46.377652	48.149515	-4.282685	-4.282685	-4.282685	-0.057084	-0.057084	-0.057084	-4.225601	-4.225601

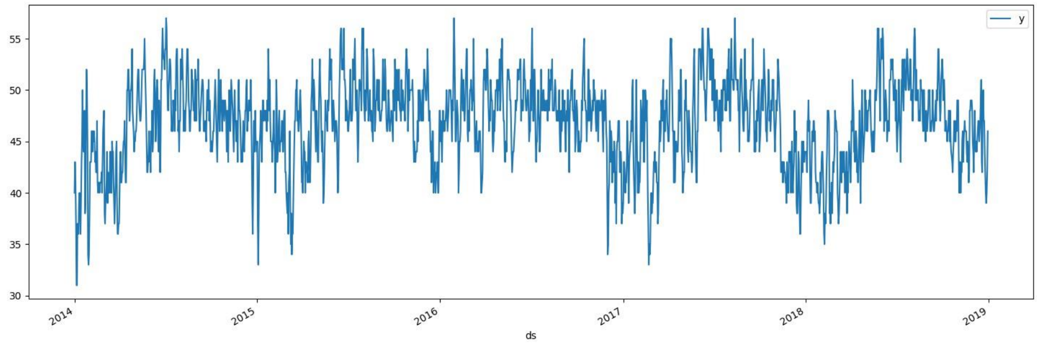
```
                              ds            y
count                       1821  1821.000000
mean   2016-06-30 20:15:25.205930752    46.818781
min              2014-01-01 00:00:00    31.000000
25%              2015-04-02 00:00:00    44.000000
50%              2016-07-01 00:00:00    47.000000
75%              2017-09-30 00:00:00    50.000000
max              2018-12-30 00:00:00    57.000000
std                              NaN     4.143192
ds    0
y     0
dtype: int64
ds    datetime64[ns]
y            float64
dtype: object
          ds         y
ds  1.000000  0.021973
y   0.021973  1.000000
```

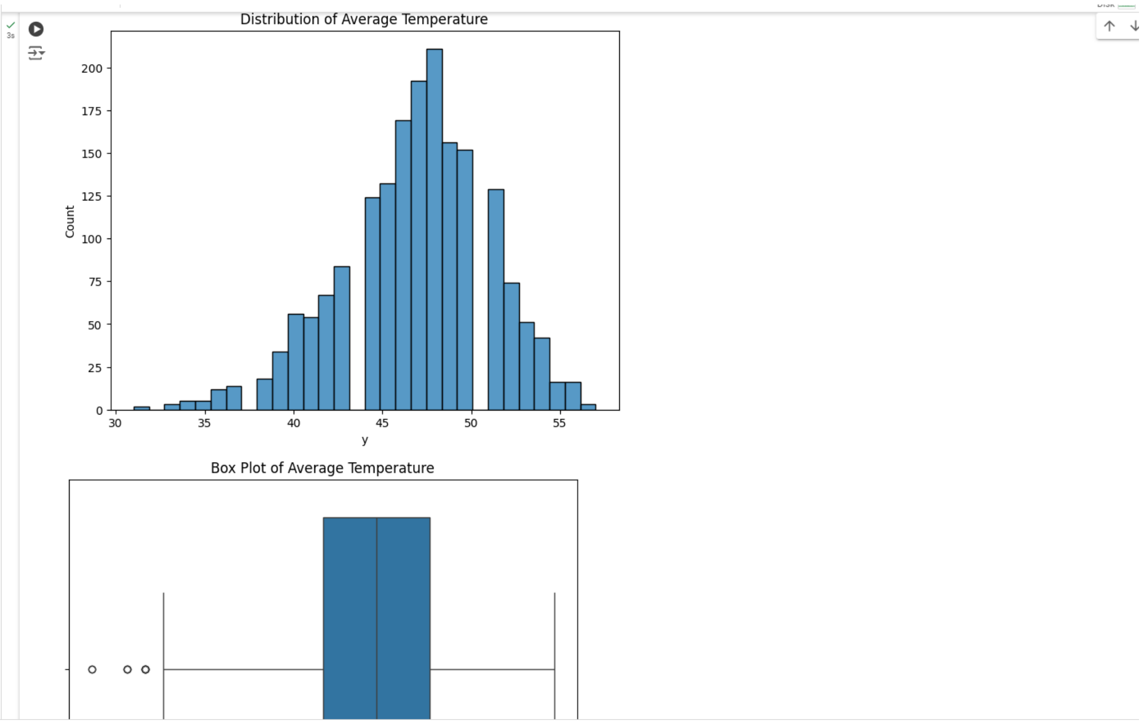


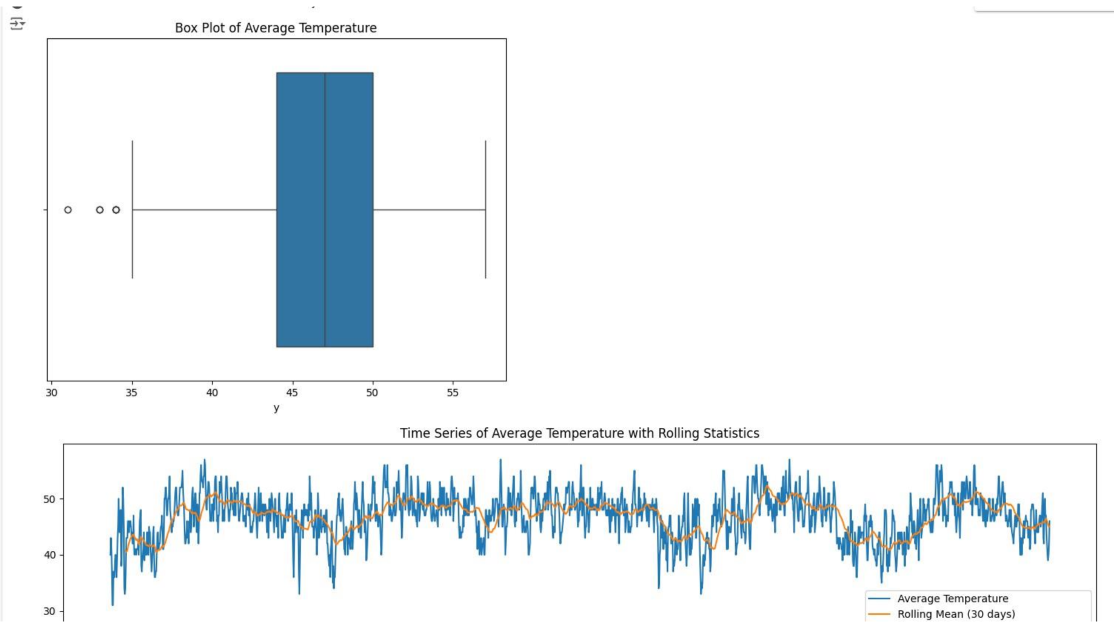


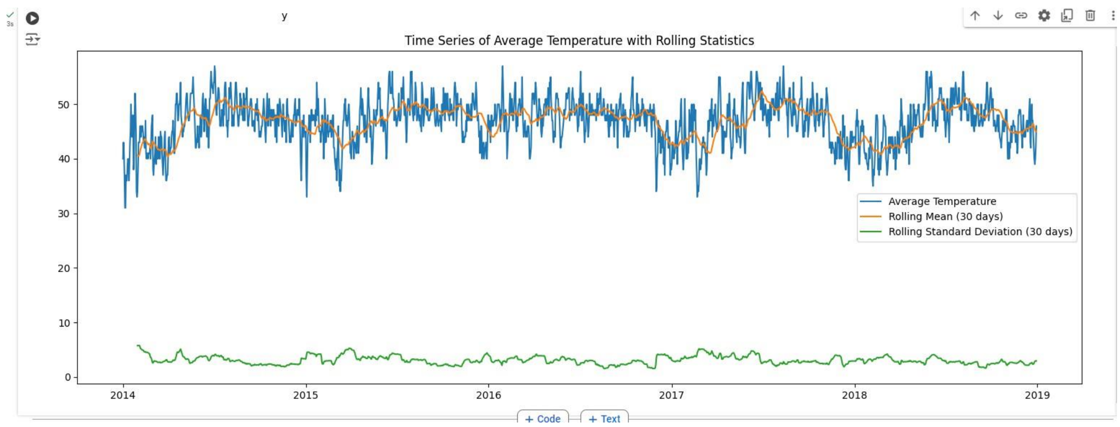
```
df.plot(x='ds',y='y',figsize=(18,6))
```

```
<Axes: xlabel='ds'>
```




Distribution of Average Temperature
Count
y
Box Plot of Average Temperature


Box Plot of Average Temperature
y
Time Series of Average Temperature with Rolling Statistics
Average Temperature
Rolling Mean (30 days)



Output:

Date: _____________ **Signature of faculty in-charge**

Post Lab Descriptive Questions:

1. Explain the components of time series?

Time series data can typically be broken down into four main components:

- **Trend**: This represents the long-term movement in the data. It shows the overall direction (upward or downward) over a period.
- **Seasonality**: These are regular, periodic fluctuations that occur at specific intervals, such as daily, monthly, or yearly. For example, retail sales often spike during the holiday season.
- **Cyclic Patterns**: Unlike seasonality, which is consistent, cyclic patterns occur over irregular intervals and are usually linked to economic or business cycles.
- **Irregular or Random Component**: This includes random noise or unexpected events that don't fit into the trend, seasonality, or cyclic components. It can be thought of as the "error" term in time series forecasting.

2. How do you handle seasonality in time series data? What methods or transformations can you apply?

To manage seasonality in time series data, you can use several methods:

- **Decomposition**: Break down the time series into its components (trend, seasonality, and residual) using techniques like STL (Seasonal and Trend Decomposition using Loess).
- **Differencing**: Subtract the value from a previous season (e.g., last year's data) to remove seasonal effects. Seasonal differencing can be effective in stabilizing the series.
- **Transformation**: Applying transformations such as logarithmic or square root can help stabilize variance, especially if seasonality affects the amplitude of the time series.
- **Seasonal Dummy Variables**: Create dummy variables for seasonal periods (e.g., months or quarters) and include them in regression models to account for their effects.
- **Seasonal ARIMA (SARIMA)**: Use specialized ARIMA models that incorporate seasonal terms to explicitly model and forecast seasonality.

3. What are some common metrics for evaluating forecasting models (e.g., MAE, RMSE, MAPE)?
4. \

Several metrics are commonly used to assess the accuracy of forecasting models:

- **Mean Absolute Error (MAE)**: The average of the absolute differences between predicted and actual values. It provides a straightforward measure of forecast accuracy.
- **Root Mean Squared Error (RMSE)**: The square root of the average of the squared differences between predicted and actual values. It gives higher weight to larger errors and is sensitive to outliers.
- **Mean Absolute Percentage Error (MAPE)**: The average of the absolute percentage errors. It expresses accuracy as a percentage, making it easier to interpret across different scales.
- **Mean Squared Error (MSE)**: The average of the squared differences between predicted and actual values. Like RMSE, it emphasizes larger errors.
- **R-squared**: Indicates the proportion of the variance in the dependent variable that can be predicted from the independent variables. Though not specifically for time series, it can provide insight into model fit.