Statistical Forecasting

```python
import pandas as pd
import numpy as np
from sklearn.metrics import mean_absolute_error, mean_squared_error
```

```python
data = pd.read_csv("NaturalGas.csv")
data.head()
```

	Time	Gas Demand (bcf)	Forecast
0	Jan-10	2210.162	NaN
1	Feb-10	2047.815	NaN
2	Mar-10	2276.546	NaN
3	Apr-10	2190.270	NaN
4	May-10	2236.507	2181.19825

```python
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 156 entries, 0 to 155
Data columns (total 3 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   Time              156 non-null    object 
 1   Gas Demand (bcf)  151 non-null    float64
 2   Forecast          147 non-null    float64
dtypes: float64(2), object(1)
memory usage: 3.8+ KB
```

Defining KPIs

```python
def kpi(df):
    dem_ave = df.loc[df['Error'].notnull(),'Demand'].mean()
    bias_abs = df['Error'].mean()
    bias_rel = bias_abs / dem_ave
    print('Bias: {:0.2f}, {:.2%}'.format(bias_abs,bias_rel))

    MAE_abs = df['Error'].abs().mean()
    MAE_rel = MAE_abs / dem_ave
    print('MAE: {:0.2f}, {:.2%}'.format(MAE_abs,MAE_rel))

    RMSE_abs = np.sqrt((df['Error']**2).mean())
    RMSE_rel = RMSE_abs / dem_ave
    print('RMSE: {:0.2f}, {:.2%}'.format(RMSE_abs,RMSE_rel))
```

1. Moving Average

```python
def moving_average(d, extra_periods = 6, n = 3):
    cols = len(d)
    demand = np.append(d,[np.nan]*extra_periods)
    forecast = np.full(cols+extra_periods, np.nan)
    for t in range(n, cols):
        forecast[t] = np.mean(demand[t-n:t])

    forecast[t+1:] = np.mean(d[t-n+1:t+1])
    df = pd.DataFrame.from_dict({'Demand':demand,'Forecast':forecast,'Error':forecast-demand})
    return df
```

```
d = data.iloc[:,[1]]
df = moving_average(d, n = 4)
df.to_csv("MA_forecast.csv")
kpi(df)
```

```
Bias: -24.39, -0.85%
MAE: 85.60, 2.99%
RMSE: 114.10, 3.98%
/usr/local/lib/python3.10/dist-packages/numpy/core/fromnumeric.py:3472: FutureWarning: In a futur
  return mean(axis=axis, dtype=dtype, out=out, **kwargs)
```

2. Simple Exponential Smoothing

```
def simple_exp_smooth(d, extra_periods=1, alpha=0.3):
    cols = len(d)
    d = np.append(d,[np.nan]*extra_periods)
    f = np.full(cols+extra_periods,np.nan)
    f[1] = d[0]
    for t in range(2,cols+1):
        f[t] = alpha*d[t-1]+(1-alpha)*f[t-1]
    for t in range(cols+1,cols+extra_periods):
        f[t] = f[t-1]
    df = pd.DataFrame.from_dict({'Demand':d,'Forecast':f,'Error':d-f})
    return df
```

```
df1 = simple_exp_smooth(d)
df1.to_csv("SES_forecast.csv")
kpi(df1)
```

```
Bias: 30.77, 1.08%
MAE: 91.24, 3.20%
RMSE: 116.22, 4.08%
```

3. Double Exponential Smoothing

```
def double_exp_smooth(d, extra_periods=1, alpha=0.3, beta=0.3):
    cols = len(d)
    d = np.append(d,[np.nan]*extra_periods)
    f = np.full(cols+extra_periods,np.nan)
    at = np.full(cols+extra_periods,np.nan)
    bt = np.full(cols+extra_periods,np.nan)
    at[0] = d[0]
    bt[0] = d[1] - d[0]
    f[1] = at[0] + bt[0]
    for t in range(1,cols+1):
        at[t] = alpha*d[t]+(1-alpha)*(at[t-1]+bt[t-1])
        bt[t] = beta*(at[t]-at[t-1])+(1-beta)*bt[t-1]
        f[t] = at[t-1]+bt[t-1]
    for t in range(cols+1,cols+extra_periods):
        f[t] = f[t-1]
    df = pd.DataFrame.from_dict({'Demand':d,'at':at,'bt':bt,'Forecast':f,'Error':d-f})
    return df
```

```
df2 = double_exp_smooth(d)
df2.to_csv("DES_forecast.csv")
kpi(df2)
```

```
Bias: 13.07, 0.46%
MAE: 95.04, 3.33%
RMSE: 131.88, 4.63%
```

Colab paid products - Cancel contracts here

✓ 0s completed at 12:24 AM