Water Metric Steps

1. Importing the required Libraries.

```
import pandas as pd
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn import metrics
import requests
import io
import warnings
warnings.filterwarnings('ignore')
```

2. Reading the generaed data, and printing the first five rows.

```
url=  'https://raw.githubusercontent.com/Duaa14/Meter/main/Meter.csv'
meters = pd.read_csv(url)
meters.head()
```

	User ID	family member	seasons	weekly	monthly	by cycle
0	11666	8	Spring	7.11	33.14	83.61
1	75296	2	Summer	2.07	8.05	25.89
2	20350	4	Summer	4.70	15.89	55.92
3	94396	8	Winter	4.13	14.74	44.70
4	19178	9	Spring	8.63	30.77	102.08

3. Calling the *describe()* function, which computes a summary of statistics pertaining to the DataFrame columns. And gives the mean, std, IQR values, and a summary about numeric columns.

```
meters.describe()
```

	User ID	family member	weekly	monthly	by cycle
count	99999.000000	99999.000000	99999.000000	99999.000000	99999.000000
mean	54671.739957	5.502275	4.181349	17.918232	53.791900
std	25876.880026	2.270426	2.104152	9.019419	27.106391
min	10012.000000	2.000000	0.810000	3.450000	10.370000
25%	32540.000000	4.000000	2.470000	10.630000	31.830000
50%	54262.000000	6.000000	3.900000	16.690000	50.140000
75%	77165.500000	7.000000	5.600000	23.940000	71.870000
max	99994.000000	9.000000	10.840000	46.460000	139.360000

4. Replacing the categorical column with a numeric values, in order to use it in prediction.

```
meters.seasons [meters.seasons == 'Winter']= 1
meters.seasons [meters.seasons == 'Autumn']= 2
meters.seasons [meters.seasons == 'Spring']= 3
meters.seasons [meters.seasons == 'Summer']= 4
meters.head()
```

	User ID	family member	seasons	weekly	monthly	by cycle
0	11666	8	3	7.11	33.14	83.61
1	75296	2	4	2.07	8.05	25.89
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]

5. Dropping the 'User ID' column since it's not useful in prediction.
6. Determining the independent variables in X -> [family member, seasons, weekly, monthly].
7. Determining the dependent variable in y -> [by cycle].

```
meters= meters.drop(('User ID'), axis=1)
X = meters.drop(["by cycle"], axis=1)
y = meters["by cycle"]
```

```
# from sklearn.preprocessing import MinMaxScaler
# scaling = MinMaxScaler(feature_range=(-1,1)).fit(x_train)
# x_train = scaling.transform(x_train)
# x_test = scaling.transform(x_test)
```

8. Splitting the data into a training and test sets, by using Scikit-Learn's built-in train_test_split() method. Where 80% for the trining data, and 20% for the test set.

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test = train_test_split(X,y,random_state=0, test_size = 0.2)
```

Multiple Linear Regression

The term "linearity" in algebra refers to a linear relationship between two or more variables.

`In this regression, we will predict the water consumption (every 3 months) that a user is expected to consume based upon the number of their family, the current season, weekly and monthly consumption. This is a multiple linear regression as it involves five variables.`

---

9. Calling the LinearRegression class, and calling the fit () method along with the training data.

```
linear = LinearRegression()
linear.fit(x_train,y_train)
```

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

`The linear regression model basically finds the best value for the intercept and slope, which results in a line that best fits the data. We will excute the following two code to see the intercept and slop value calculated by the linear regression algorithm for our dataset.`

---

10. The following code is to retrieve the intercept:

```
print(linear.intercept_)
```

```
-7.315860848893642
```

11. The following code is for retrieving the slope (coefficients of x):

```
NOTE:

* These values tell us that if the family member increases by 1, the water consumption increases by 2.481368.

*   And if the season changed from one to another, the water consumption increases by 2.795319.
```

```
* Also if the weekly consumption increased by 1 miter, the water consumption increases by 4.785608.

* Finally if the monthly consumption increased by 1 miter, the water consumption increases by 1.142562.
```

These numbers show how the predicted water consumption will be affected by these coefficients.

```
coeff_df = pd.DataFrame(linear.coef_, X.columns, columns=['Coefficient'])
coeff_df
```

	Coefficient
family member	2.481368
seasons	2.795319
weekly	4.785608
monthly	1.142562

Making Predictions

---

12. Now we will make some predictions, after we have trained our model. To do so, we will use our test data and see how accurately our model predicts the percentage score.

```
y_pred = linear.predict(x_test)
y_pred
```

```
array([50.9776673 , 33.00338133, 61.04538261, ..., 86.4924522 ,
       68.56766884, 47.39003399])
```

```
# predict the monthly consumption when the family members = 2, and we are in Autumn.
month = linear.predict([[3,1,1.45,6.46]])
month
```

```
array([17.24364622])
```

```
meters.loc[(meters['family member'] == 4) & (meters['seasons'] == 1) & (meters['weekly']== 1.45) & (meters['monthly']==6.46)]
```

family member	seasons	weekly	monthly	by cycle

Evaluating the Model

---

13. We will evaluate the performance of model. This step is particularly important to compare how well different models perform on this dataset. For regression algorithms, three evaluation metrics are commonly used:

- The mean absolute error (MAE) turns out to be 2.119037. This tells us that the average difference between the actual data value and the value predicted by the linear model is 2.119037. It is calculated as:

$$MAE = \sum_{i=1}^{n} |Actual - Predicted|$$

- The mean square error is the average of the square of the difference between the observed and predicted values of the monthly consumption.

$$MSE = \sum_{i=1}^{n} |Actual - Predicted|^2$$

- Root Mean Square Error is the square root of the average of the squared differences between the estimated and the actual value of monthly consumption.

$$RMSE = \sum_{i=1}^{n} \sqrt{|Actual - Predicted|^2}$$

```
from sklearn import metrics
```

```
from sklearn import metrics

e1 = metrics.mean_absolute_error(y_test, y_pred)
e2 = metrics.mean_squared_error(y_test, y_pred)
e3 = np.sqrt(metrics.mean_squared_error(y_test, y_pred))

print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
```

```
Mean Absolute Error: 4.425943080289606
Mean Squared Error: 33.96530732367923
Root Mean Squared Error: 5.827976263136221
```

14. Calculating the R squared

`Note: R squared is a regression error metric to evaluate the accuracy and efficiency of a model on the data values that it would be applied to.`

```
linear_r2 = metrics.r2_score(y_test, y_pred)
```

Support Vector Regression

Support vector regression (SVR) is a statistical method that examines the linear relationship between two continuous variables.

15. Calling the SVR class, and calling the fit () method along with the training data.

```
from sklearn.svm import SVR
regressor = SVR(kernel='linear')
regressor.fit(x_train,y_train)
```

```
array([49.92413135, 33.21541999, 60.75493383, ..., 85.54830426,
       68.02214039, 47.3401441 ])
```

16. Making predictions

```
predicted = regressor.predict(x_test)
predicted
```

Evaluating the model

17. As explained before, we will evaluate the SVR model using MAE, MSE, and RMSE.

```
s1 =  metrics.mean_absolute_error(y_test, predicted)
s2 =  metrics.mean_squared_error(y_test, predicted)
s3 =  np.sqrt(metrics.mean_squared_error(y_test, predicted))

print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, predicted))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, predicted))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, predicted)))
```

```
Mean Absolute Error: 4.374001994850912
Mean Squared Error: 34.4423639699528
Root Mean Squared Error: 5.868761706693568
```

18. Calculating the R squared for the SVR model.

```
SVR_r2 = metrics.r2_score(y_test, predicted)
```

19. Comparing the actual output values for X_test with the predicted values in both the linear and SVR model.

> Though our models is not very precise, the predicted percentages are close to the actual ones.

```
df = pd.DataFrame({'Actual': y_test, 'Linear Predictections': y_pred, 'SVR Predictions':predicted})
df
```

	Actual	Linear Predictections	SVR Predictions
3582	46.85	50.977667	49.924131
60498	33.54	33.003381	33.215420
53227	61.73	61.045383	60.754934
21333	41.39	36.624006	36.630083
3885	102.83	109.366599	109.368195
...	...	...	...
60116	112.13	110.547960	109.696026
2415	28.74	30.893243	30.498945
43763	81.41	86.492452	85.548304
71344	66.69	68.567669	68.022140
82259	48.74	47.390034	47.340144

20000 rows × 3 columns

20. Showing the difference in accuracy and the mean error in both the linear and the SVR model.

```
List = {'Linear': [linear_r2,e1,e2,e3],
        'SVR': [SVR_r2,s1,s2,s3]}
comp = pd.DataFrame(List,index=['R Squared','Mean Absolute Error','Mean Squared Error','Root Mean Squared Error'])
comp
```

	Linear	SVR
R Squared	0.952905	0.952243
Mean Absolute Error	4.425943	4.374002
Mean Squared Error	33.965307	34.442364
Root Mean Squared Error	5.827976	5.868762