Name: MD Ekram Ullah
Batch: ML-DS 102

1. Problem Statement:

The problem is about to find the salinity of ocean water

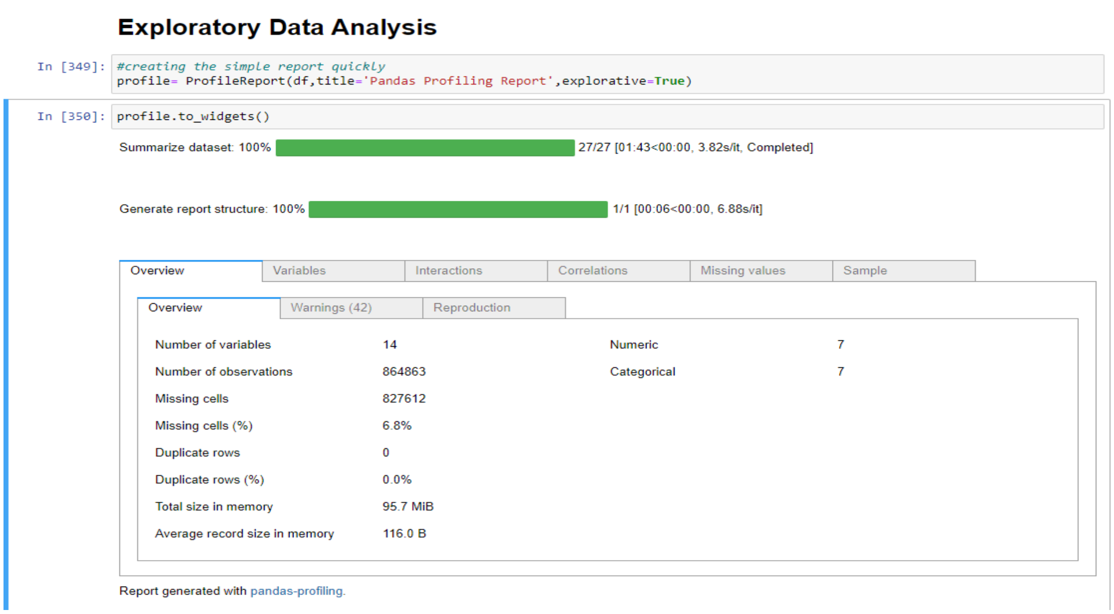
2. Appproach:

First of all, I loaded the data in jupyter notebook and put a detailed look inti the data. I dropped the feature which seemed unnecessary for the determination of the result. I calculated the missing percentage of each feature and dropped the features which have more than 30% missing values. Then I label encoded the categorical features. Later, I calculated the correlation of the features on the basis of the Label data and removed the features that are highly correlated to each other. Then, I dropped all the rows with Nan values out there. Then I did the feature scaling and train test split and Lastly I trained the regression models.

3. Phases:

i) Explorartory Data Analysis:

Using the ProfileReport module from pandas , I did my EDA task. I got the full visualization of the features , the whole overview of each feature, the interaction between features, correlations, the amount of missing values and sample data also. From the correlation heatmap I got the understanding of the nullity correlation of the features.



Exploratory Data Analysis

In [349]:
```python
#creating the simple report quickly
profile= ProfileReport(df,title='Pandas Profiling Report',explorative=True)
```

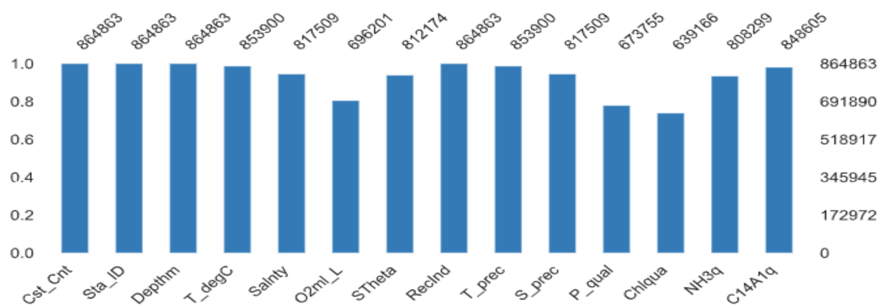
In [350]:
```python
profile.to_widgets()
```

Summarize dataset: 100% 27/27 [01:43<00:00, 3.82s/it, Completed]

Generate report structure: 100% 1/1 [00:06<00:00, 6.88s/it]

Overview | Variables | Interactions | Correlations | Missing values | Sample

Count | Matrix | Heatmap | Dendrogram



A simple visualization of nullity by column.

Report generated with pandas-profiling.

Exploratory Data Analysis

In [349]:
```python
#creating the simple report quickly
profile= ProfileReport(df,title='Pandas Profiling Report',explorative=True)
```

In [350]:
```python
profile.to_widgets()
```

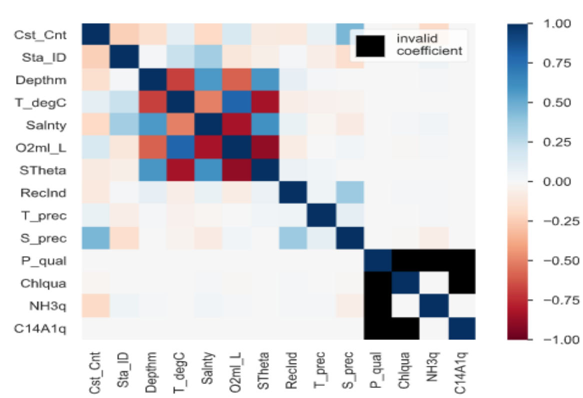
Summarize dataset: 100% 27/27 [01:43<00:00, 3.82s/it, Completed]

Generate report structure: 100% 1/1 [00:06<00:00, 6.88s/it]

Overview | Variables | Interactions | Correlations | Missing values | Sample

Toggle correlation descriptions

Pearson's r | Spearman's ρ | Kendall's τ | Phik (φk) | Cramér's V (φc)



ii) Data Preprocessing:

In this stage , I cleaned the dataset as much as I can . First of all , I dropped the unnecessary features from the dataset. Then I calculated the missing percentage of each feature and deleted the features that have missing percentage above 30%.
I did this because I observed that the features that have more than 30% of missing values are the one that have the largest portion of missing values throughout the dataset.

I dropped all the rows that have null values in it as I have got a huge bunch of data . I did that because, in spite of removing the rows with null values, still I have a huge amount of data

dropping the rows with null values

In [135]:
```
df2=df.dropna(thresh=14) #dropping all rows with NaN values
```

In [136]:
```
df2
```

Out[136]:

	Cst_Cnt	Sta_ID	Depthm	T_degC	Salnty	O2ml_L	STheta	RecInd	T_prec	S_prec	P_qual	Chlqua	NH3q	C14A1q
2160	71	162	0	10.30	33.030	5.90	25.364	3	1.0	2.0	9.0	9.0	9.0	9.0
2161	71	162	6	18.46	32.920	6.02	23.568	3	2.0	2.0	9.0	9.0	9.0	9.0
2162	71	162	10	10.29	32.951	6.04	25.304	7	2.0	3.0	9.0	9.0	9.0	9.0
2163	71	162	15	10.29	32.990	6.06	25.335	3	2.0	2.0	9.0	9.0	9.0	9.0
2164	71	162	20	10.33	33.005	6.04	25.339	7	2.0	3.0	9.0	9.0	9.0	9.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
817823	32582	575	250	7.83	34.088	1.51	26.587	7	2.0	3.0	9.0	9.0	9.0	9.0
817825	32582	575	300	7.09	34.115	1.18	26.713	7	2.0	3.0	9.0	9.0	9.0	9.0
817828	32582	575	400	6.50	34.196	0.62	26.858	7	2.0	3.0	9.0	9.0	9.0	9.0
817830	32582	575	500	5.84	34.252	0.40	26.987	7	2.0	3.0	9.0	9.0	9.0	9.0
830772	33073	1324	0	18.05	33.586	6.40	24.178	7	2.0	3.0	9.0	9.0	9.0	9.0

386376 rows × 14 columns

iii) Handling Categorical Features:

Luckily , there was only one categorical feature out there after the removal of features in the preprocessing step. I used Label Encoding to transform the feature into numerical data.
I did that because, if I use one hot encoding, there will be a huge number of columns there.

Handling Catagorical values

In [118]:
```
df['Sta_ID'].nunique()
```
Out[118]:
```
2634
```

In [119]:
```
from sklearn.preprocessing import LabelEncoder
```

In [120]:
```
labelencoder = LabelEncoder()
```

In [121]:
```
df['Sta_ID'] = labelencoder.fit_transform(df['Sta_ID'])
```

In [123]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 864863 entries, 0 to 864862
Data columns (total 31 columns):
 #   Column       Non-Null Count   Dtype
---  ------       --------------   -----
 0   Cst_Cnt      864863 non-null  int64
 1   Btl_Cnt      864863 non-null  int64
 2   Sta_ID       864863 non-null  int32
 3   Depthm       864863 non-null  int64
 4   T_degC       853900 non-null  float64
 5   Salnty       817509 non-null  float64
 6   O2ml_L       696201 non-null  float64
 7   STheta       812174 non-null  float64
 8   O2Sat        661274 non-null  float64
 9   Oxy_µmol/Kg  661268 non-null  float64
 10  RecInd       864863 non-null  int64
 11  T_prec       853900 non-null  float64
 12  S_prec       817509 non-null  float64
 13  P_qual       673755 non-null  float64
 14  Chlqua       639166 non-null  float64
 15  Phaqua       639170 non-null  float64
 16  NH3q         808299 non-null  float64
 17  C14A1q       848605 non-null  float64
 18  C14A2q       848623 non-null  float64
 19  DarkAq       840440 non-null  float64
 20  MeanAq       840439 non-null  float64
 21  R_Depth      864863 non-null  float64
 22  R_TEMP       853900 non-null  float64
 23  R_POTEMP     818816 non-null  float64
 24  R_SALINITY   817509 non-null  float64
 25  R_SIGMA      812007 non-null  float64
 26  R_SVA        812092 non-null  float64
 27  R_DYNHT      818206 non-null  float64
 28  R_O2         696201 non-null  float64
 29  R_O2Sat      666448 non-null  float64
 30  R_PRES       864863 non-null  int64
dtypes: float64(25), int32(1), int64(5)
memory usage: 201.3 MB
```

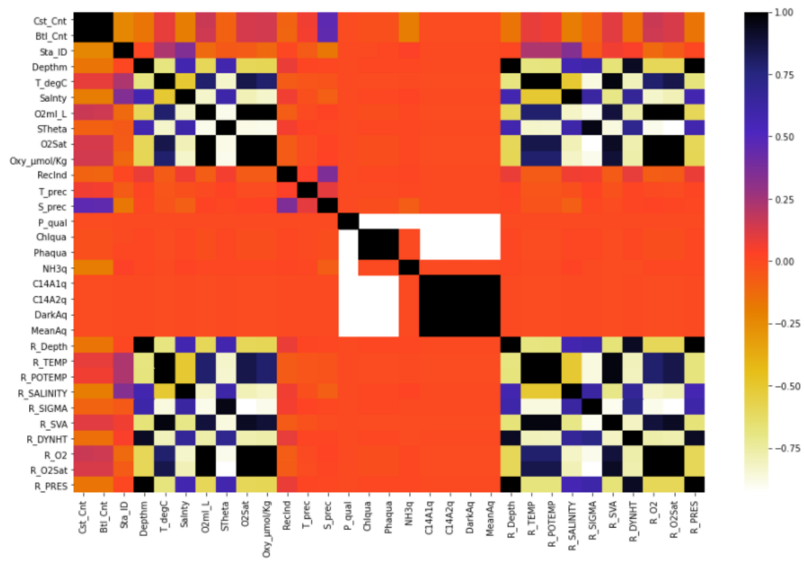
iv) Correlation:

I calculated correlation on the basis of the output feature 'Salnty'.I also plot the correlation plot using seaborn. I observed there are many features that are highly correlated with each other. So, I removed the highly correlated features from the dataset. Because they act like same feature.

```
In [125]: corr = df.corr()
```

```
In [126]: plt.figure(figsize=[16,10])
          sns.heatmap(corr,cmap=plt.cm.CMRmap_r)
```

Out[126]: <AxesSubplot:>



v) Train Test Split:

I divided the dataset into 3:1 ratio. Where 75% was training data and 25% for test data. The training dataset contained 289782 number of rows and 13 columns. The test dataset contains 96594 number of rows and 13 rows in it.

I did that because there are a huge number of datas in the dataset. So, I took 25% in the test data.

Train Test split

```
In [138]: x= df2.drop('Salnty',axis=1)
          y= df2['Salnty']

          from sklearn.model_selection import train_test_split

          x_train,x_test,y_train,y_test= train_test_split(x,y,test_size=.25,random_state=5)
```

```
In [139]: x_train.shape
```

Out[139]: (289782, 13)

```
In [140]: x_test.shape
```

Out[140]: (96594, 13)

vi) Feature Scaling:

I used MinMaxScaler method to scale the features . Because there was no negative values in my dataset.

Feature Scaling

In [35]:
```
# data normalization with sklearn
from sklearn.preprocessing import MinMaxScaler

# fit scaler on training data
norm = MinMaxScaler().fit(x_train)

# transform training data
x_train = norm.transform(x_train)

# transform testing dataabs
x_test = norm.transform(x_test)

x_train
x_test
```

Out[35]:
```
array([[6.34264590e-01, 6.54006836e-01, 1.12689217e-01, ...,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00],
       [3.97248652e-02, 3.09532852e-01, 1.73051766e-01, ...,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00],
       [3.31252651e-01, 1.86099506e-01, 1.86880957e-04, ...,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00],
       ...,
       [3.89461245e-01, 6.24762628e-01, 9.34404784e-02, ...,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00],
       [2.54439125e-01, 1.64830991e-01, 2.80321435e-02, ...,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00],
       [6.69656384e-03, 2.23699202e-01, 0.00000000e+00, ...,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00]])
```

4. Modelling:

i) Linear regression:

Definition:

Linear regression is an attractive model because the representation is so simple.

The representation is a linear equation that combines a specific set of input values (x) the solution to which is the predicted output for that set of input values (y). As such, both the input values (x) and the output value are numeric.

The linear equation assigns one scale factor to each input value or column, called a coefficient and represented by the capital Greek letter Beta (B). One additional coefficient is also added, giving the line an additional degree of freedom (e.g. moving up and down on a two-dimensional plot) and is often called the intercept or the bias coefficient.

For example, in a simple regression problem (a single x and a single y), the form of the model would be:

$$y = B0 + B1*x$$

Performance Metrics:

Mean Absolute Error(MAE): This metric calculates the sum of the average of the absolute error between the predicted values and the true values which does not consider direction. The cons of this metric is that it is unable to give information about the model overshooting or undershooting, so the smaller it is, the better the model.

My application of this metric: 0.06755557585955091

Mean Squared Error(MSE): This metric is the average of the squared difference between the target value and the value predicted by the regression model. The con to this metric is that it is more sensitive to outliers present in the dataset.

My application of this metric: 0.010194830565404062

Root Mean Square Error(RMSE): This metric is the square root of the mean square error that estimates the standard deviation of the residuals, describing the spread of the residuals from the line of best fit and the noise in the model. A low RMSE postulates that the error made by the model has a small deviation from the true values.

My application of this metric: 0.10096945362536168

Finally, I checked the model prediction score using the r2_score library an d got a 0.9508505380654608 accuracy.

Screenshot of the output:

1. Linear Regression

In [254]:
```
import sklearn
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score
from sklearn import metrics
import math
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error

regressor_1 = LinearRegression(normalize=True)
regressor_1.fit(x_train,y_train)
y_pred_1 = regressor_1.predict(x_test)
```

r2_score, MAE, MSE, RMSE

In [256]:
```
r2_score_LR=r2_score(y_test,y_pred_1)
r2_score_LR
```

Out[256]: 0.9508505380654608

In [257]:
```
MSE_LR=np.mean((regressor_1.predict(x_test) - y_test) ** 2)
MSE_LR
```

Out[257]: 0.010194830565404062

In [258]:
```
RMSE_LR = math.sqrt(MSE_LR)
RMSE_LR
```

Out[258]: 0.10096945362536168

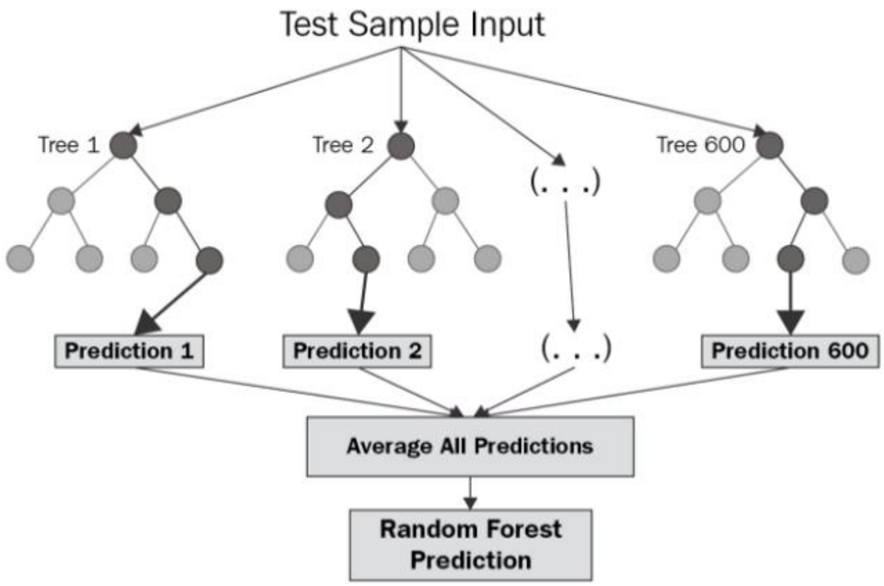
In [259]:
```
MAE_LR = mean_absolute_error(y_test, y_pred_1)
MAE_LR
```

Out[259]: 0.06755557585955091

ii) Random Forest Regression:

Definition:

Random Forest Regression is a supervised learning algorithm that uses **ensemble learning** method for regression. Ensemble learning method is a technique that combines predictions from multiple machine learning algorithms to make a more accurate prediction than a single model.



The diagram above shows the structure of a Random Forest. You can notice that the trees run in parallel with no interaction amongst them. A Random Forest operates by constructing several decision trees during training time and outputting the mean of the classes as the prediction of all the trees.

Performance Metrics:

Mean Absolute Error(MAE):

My application of this metric: 0.006602247896694698

Mean Squared Error(MSE):

My application of this metric: 0.00031516410020866335

Root Mean Square Error(RMSE):

My application of this metric: 0.017752861747015982

Finally, I checked the model prediction score using the r2_score library and got a `0.9984805881915385` accuracy.

Tuning Details:

i) n_estimators=100, random_state=42, max_features=4 ,

R2_score= .9928, MAE= .0215, MSE=.0014, RMSE=.0386

ii) n_estimators=120, random_state=50, max_features=10,

R2_score= .9981, MAE= .0084, MSE=.0038, RMSE=.0195

iii) n_estimators=150, random_state=50, max_features=13 ,

R2_score= .9984, MAE= .0066, MSE=.0003, RMSE=.0177

Screenshot of the output:

2. Random Forest Regression

In [290]:
```
from sklearn.ensemble import RandomForestRegressor

regressor_2 = RandomForestRegressor( n_estimators = 150, random_state = 50,max_features=13,criterion='r
regressor_2.fit(x_train,y_train)
y_pred_2 = regressor_2.predict(x_test)
```

r2_score, MAE, MSE, RMSE

In [291]:
```
r2_score_RFR=r2_score(y_test,y_pred_2)

r2_score_RFR
```

Out[291]: 0.9984805881915385

In [292]:
```
MSE_RFR= np.mean((regressor_2.predict(x_test) - y_test) ** 2)
MSE_RFR
```

Out[292]: 0.00031516410020866335

In [293]:
```
RMSE_RFR = math.sqrt(MSE_RFR)
RMSE_RFR
```

Out[293]: 0.017752861747015982

In [294]:
```
MAE_RFR = mean_absolute_error(y_test, y_pred_2)
MAE_RFR
```

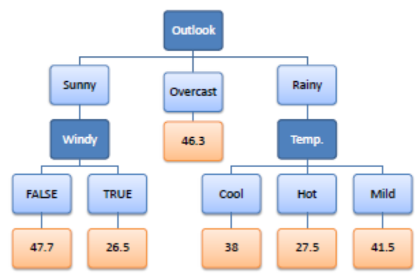
Out[294]: 0.006602247896694698

iii) Decision Tree Regression:

Definition:

Decision tree builds regression or classification models in the form of a tree structure. It breaks down a dataset into smaller and smaller subsets while at the same time an associated decision tree is incrementally developed. The final result is a tree with **decision nodes** and **leaf nodes**. A decision node (e.g., Outlook) has two or more branches (e.g., Sunny, Overcast and Rainy), each representing values for the attribute tested. Leaf node (e.g., Hours Played) represents a decision on the numerical target. The topmost decision node in a tree which corresponds to the best predictor called **root node**. Decision trees can handle both categorical and numerical data.

Predictors				Target
Outlook	Temp.	Humidity	Windy	Hours Played
Rainy	Hot	High	False	26
Rainy	Hot	High	True	30
Overcast	Hot	High	False	48
Sunny	Mild	High	False	46
Sunny	Cool	Normal	False	62
Sunny	Cool	Normal	True	23
Overcast	Cool	Normal	True	43
Rainy	Mild	High	False	36
Rainy	Cool	Normal	False	38
Sunny	Mild	Normal	False	48
Rainy	Mild	Normal	True	48
Overcast	Mild	High	True	62
Overcast	Hot	Normal	False	44
Sunny	Mild	High	True	30



Performance Metrics:

Mean Absolute Error(MAE):

Applied this principle in my model and this is the outcome

```
0.013246257531523706
```

Mean Squared Error(MSE):

My application of this metric

```
0.0009528147400459523
```

Root Mean Square Error(RMSE):

My application of this metric

```
0.030867697355746383
```

Finally, I checked the model prediction score using the r2_score libr
ary
and got a `0.9954064629621727` accuracy.

Tuning Details:

i) random_state=50, max_features=4 ,

R2_score= .9732, MAE= .0377, MSE=.0055, RMSE=.0745

ii) random_state=75, max_features=9,

R2_score= .9888, MAE= .0205, MSE=.0023, RMSE=.0481

iii) random_state=50, max_features=13 ,

R2_score= .9954, MAE= .0132, MSE=.0009, RMSE=.0308

Screenshot of the output:

3. Decision Tree Regression

In [305]:
```
from sklearn.tree import DecisionTreeRegressor

# create a regressor object
regressor_3 = DecisionTreeRegressor(random_state = 100,criterion='mse',max_features=13)

# fit the regressor with X and Y data
regressor_3.fit(x_train,y_train)
y_pred_3 = regressor_3.predict(x_test)
```

r2_score, MAE, MSE, RMSE

In [306]:
```
r2_score_DTR=r2_score(y_test,y_pred_3)

r2_score_DTR
```

Out[306]: 0.9954064629621727

In [307]:
```
MSE_DTR= np.mean((regressor_3.predict(x_test) - y_test) ** 2)
MSE_DTR
```

Out[307]: 0.0009528147400459523

In [308]:
```
RMSE_DTR = math.sqrt(MSE_DTR)
RMSE_DTR
```

Out[308]: 0.030867697355746383

In [309]:
```
MAE_DTR = mean_absolute_error(y_test, y_pred_3)
MAE_DTR
```

Out[309]: 0.013246257531523706

iv) Bayesian Regression:

Definition:

In the Bayesian viewpoint, we formulate linear regression using probability distributions rather than point estimates. The response, y, is not estimated as a single value, but is assumed to be drawn from a probability distribution. The model for Bayesian Linear Regression with the response sampled from a normal distribution is:

$$y \sim N(\beta^T X, \sigma^2 I)$$

The output, y is generated from a normal (Gaussian) Distribution characterized by a mean and variance. The mean for linear regression is the transpose of the weight matrix multiplied by the predictor matrix. The variance is the square of the standard deviation σ (multiplied by the Identity matrix because this is a multi-dimensional formulation of the model).

The aim of Bayesian Linear Regression is not to find the single "best" value of the model parameters, but rather to determine the posterior distribution for the model parameters.

Performance Metrics:

Mean Absolute Error(MAE):

Applied this principle in my model and this is the outcome

```
0.06755556958280429
```

Mean Squared Error(MSE):

My application of this metric `0.010194828019839494`

Root Mean Square Error(RMSE):

My application of this metric

```
0.10096944101974366
```

Finally, I checked the model prediction score using the r2_score libr
ary
and got a `0.9508505503376731` accuracy.

Tuning Details:

i) n_iter=300, alpha_1=1e-06, alpha_2=1e-06, lambda_1=1e-06, lambda_2=1e-06

ii) n_iter=200, alpha_1=1e-05, alpha_2=1e-05, lambda_1=1e-05, lambda_2=1e-05

iii) n_iter=100, alpha_1=1e-04, alpha_2=1e-04, lambda_1=1e-04, lambda_2=1e-04

Screenshot of the output:

4. Bayesian regression

In [330]:
```
from sklearn.linear_model import BayesianRidge

regressor_4 = BayesianRidge(n_iter=40,alpha_1=1e-01,
    alpha_2=1e-01,
    lambda_1=1e-01,
    lambda_2=1e-01)
regressor_4.fit(x_train, y_train)

y_pred_4 = regressor_4 .predict(x_test)
```

r2_score, MAE, MSE, RMSE

In [331]:
```
r2_score_BR=r2_score(y_test,y_pred_4)

r2_score_BR
```

Out[331]: 0.9508505503376731

In [332]:
```
MSE_BR= np.mean((regressor_4.predict(x_test) - y_test) ** 2)
MSE_BR
```

Out[332]: 0.010194828019839494

In [333]:
```
RMSE_BR = math.sqrt(MSE_BR)
RMSE_BR
```

Out[333]: 0.10096944101974366

In [334]:
```
MAE_BR = mean_absolute_error(y_test, y_pred_4)
MAE_BR
```

Out[334]: 0.06755556958280429

v) Neural Network Regression:

Definition:

Although neural networks are widely known for use in deep learning and modeling complex problems such as image recognition, they are easily adapted to regression problems. Any class of statistical models can be termed a neural network if they use adaptive weights and can approximate non-linear functions of their inputs. Thus neural network regression is suited to problems where a more traditional regression model cannot fit a solution.

Neural network regression is a supervised learning method, and therefore requires a *tagged dataset*, which includes a label column. Because a regression model predicts a numerical value, the label column must be a numerical data type.

Performance Metrics:

Mean Absolute Error(MAE):

My application of this metric: 0.04703519865870476

Mean Squared Error(MSE):

My application of this metric: 0.005472843069583178

Root Mean Square Error(RMSE):

My application of this metric: 0.13853444159030914

Tuning Details:

i) Hidden_layers=2 , Neurons= 39

MSE= .00038, MAE= .0147, MAPE= .0433

ii) Hidden_layers= 3, Neurons= 43

MSE= .0055, MAE= .0502, MAPE= .1480

iii) Hidden_layers= 1, Neurons= 34,

MSE= .0055, MAE= .0470, MAPE= .1385

Screenshot of the output:

r2_score, MAE, MSE, RMSE

```
In [343]: test_loss
Out[343]: [0.005472843069583178,
           0.04703519865870476,
           0.005472843069583178,
           0.13853444159030914]
```