Spark Mllib

We will cover:

- Introduction to Spark MLlib
- Code walkthrough:
  - ML Algorithm implementation – Linear Regression

- Spark MLlib (Machine Learning library) is being developed with the goal of making machine learning easy and scalable.
- It makes available several utilities and common machine learning algorithms that include regression, classification, clustering, collaborative filtering.
- MLlib includes tools for operations such as:
  - Featurization: feature extraction, selection and transformation
  - Pipelines: APIs for constructing, evaluating, and tuning ML Pipelines
  - Persistence: saving & loading algorithms and models
- It is divided into two packages:
  - `spark.mllib` contains the original API built on top of RDDs.
  - `spark.ml` provides higher-level API built on top of DataFrames for constructing ML pipelines.
- As of Spark 2.0, the RDD-based APIs i.e. those in the `spark.mllib` package have entered maintenance mode.
- Going forward the primary Machine Learning API library for Spark is the DataFrame-based `spark.ml` package.
- Let us take a use case of an ML algorithm – Linear Regression with a data set and understand how Spark MLlib is used to develop and implement machine learning algorithms.

- Linear Regression is a machine learning algorithm that comes under Supervised Learning.
- A supervised learning technique takes historical data of inputs and output, learns the relationship between the inputs & output and builds the model.
- Then, for a given new input it predicts the output by applying the model.
- Linear Regression is an approach to modelling the relationship between the input variables and the output as a linear function.
- In Spark MLlib the input variables are referred to as *features* and the output variable as *label.* And the predicted value of new input is simply called prediction.
- We will use Combined Cycle Power Plant Dataset available at UCI Machine Learning Repository for this exercise.
- The dataset contains 9568 data points collected from a Combined Cycle Power Plant over 6 years (2006-2011), when the power plant was set to work with full load.
  - A combined-cycle power plant uses both a gas turbine and a steam turbine together to produce up to 50 percent more electricity from the same amount of fuel than a traditional simple-cycle plant.

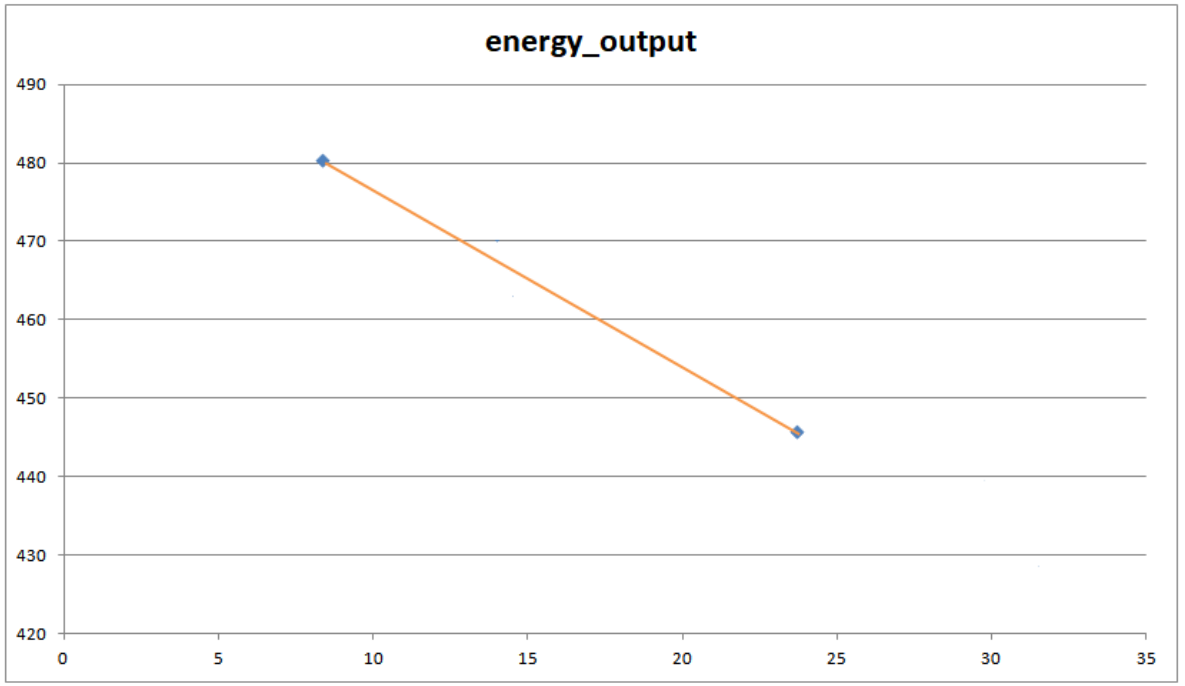
- *Features* in our dataset consist of hourly average ambient variables `Temperature, Exhaust Vacuum, Ambient Pressure` and `Relative Humidity.`
- The column `Energy Output` which is the net hourly electrical energy is the *Label*.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
1	temperature	exhaust_vacuum	ambient_pressure	relative_humidity	energy_output													
2	8.34	40.77	1010.84	90.01	480.48													
3	23.64	58.49	1011.4	74.2	445.75													
4	29.74	56.9	1007.15	41.91	438.76													
5	19.07	49.69	1007.22	76.79	453.09													
6	11.8	40.66	1017.13	97.2	464.43													
7	13.97	39.16	1016.05	84.6	470.96													
8	22.1	71.29	1008.2	75.38	442.35													
9	14.47	41.76	1021.98	78.41	464													
10	31.25	69.51	1010.25	36.83	428.77													
11	6.77	38.18	1017.8	81.13	484.31													
12	28.28	68.67	1006.36	69.9	435.29													
13	22.99	46.93	1014.15	49.42	451.41													
14	29.3	70.04	1010.95	61.23	426.25													
15	8.14	37.49	1009.04	80.33	480.66													
16	16.92	44.6	1017.34	58.75	460.17													
17	22.72	64.15	1021.14	60.34	453.13													
18	18.14	43.56	1012.83	47.1	461.71													
19	11.49	44.63	1020.44	86.04	471.08													
20	9.94	40.46	1018.9	68.51	473.74													
21	23.54	41.1	1002.05	38.05	448.56													
22	14.9	52.05	1015.11	77.33	464.82													
23	33.8	64.96	1004.88	49.37	427.28													
24	25.37	68.31	1011.12	70.99	441.76													
25	7.29	41.04	1024.06	89.19	474.71													
26	13.55	40.71	1019.13	75.44	467.21													
27	6.39	35.57	1025.53	77.23	487.69													
28	26.64	62.44	1011.81	72.46	438.67													
29	7.84	41.39	1018.21	91.92	485.66													
30	21.82	58.66	1011.71	64.37	452.16													
31	27.17	67.45	1015.67	49.03	429.87													
32	13.42	41.23	994.17	95.79	468.82													
33	20.77	56.85	1012.4	83.63	442.85													
34	8.29	36.08	1020.38	81.53	483.26													
35	30.98	67.45	1015.18	45.4	433.59													

- Let us take one input variable (or *feature*) – `Temperature` & the output variable (i.e. *label*) `Energy Output` and plot the points on X-Axis and Y-Axis respectively. To begin with, let us plot only 2 such data points.
- We can draw a line through these points which can be expressed as the function: $y = ax + b$

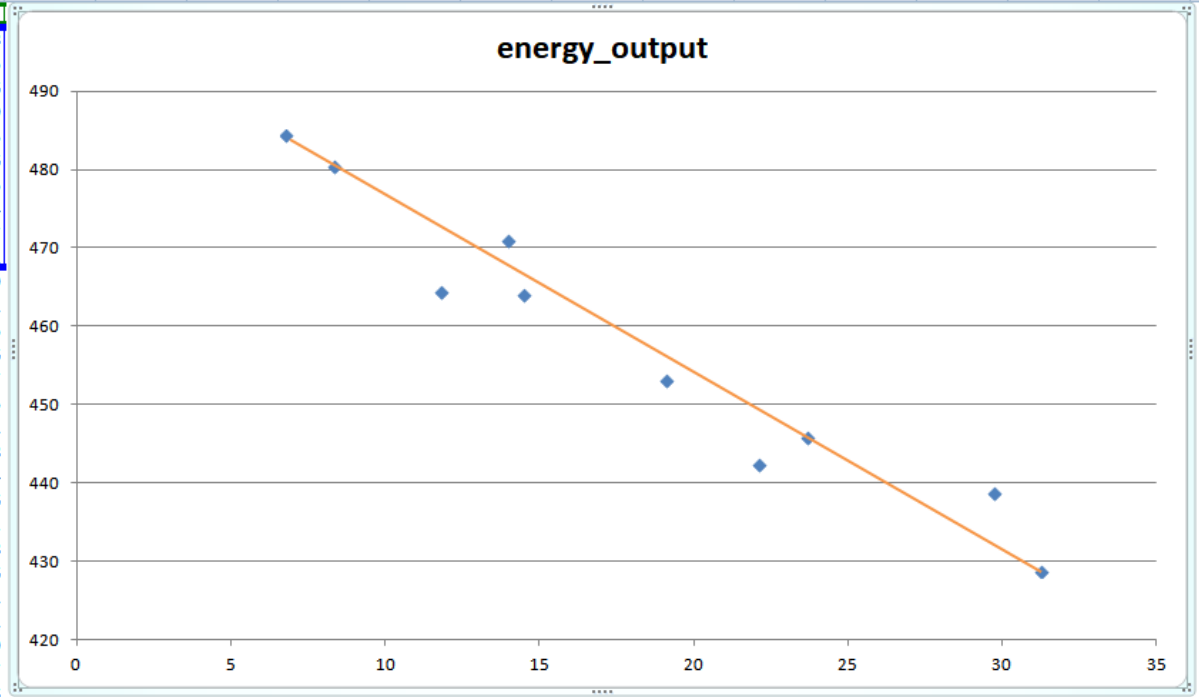
  where the coefficient $a = -2.27$ (which is the slope of the line) and the coefficient $b = 499.41$ (which is the intercept of the line on Y-Axisin) this case.

	A	B	C	D	E
1	temperature	exhaust_vacuum	ambient_pressure	relative_humidity	energy_output
2	8.34	40.77	1010.84	90.01	480.48
3	23.64	58.49	1011.4	74.2	445.75
4	29.74	56.9	1007.15	41.91	438.76
5	19.07	49.69	1007.22	76.79	453.09
6	11.8	40.66	1017.13	97.2	464.43
7	13.97	39.16	1016.05	84.6	470.96
8	22.1	71.29	1008.2	75.38	442.35
9	14.47	41.76	1021.98	78.41	464
10	31.25	69.51	1010.25	36.83	428.77
11	6.77	38.18	1017.8	81.13	484.31
12	28.28	68.67	1006.36	69.9	435.29
13	22.99	46.93	1014.15	49.42	451.41
14	29.3	70.04	1010.95	61.23	426.25
15	8.14	37.49	1009.04	80.33	480.66
16	16.92	44.6	1017.34	58.75	460.17
17	22.72	64.15	1021.14	60.34	453.13
18	18.14	43.56	1012.83	47.1	461.71
19	11.49	44.63	1020.44	86.04	471.08
20	9.94	40.46	1018.9	68.51	473.74
21	23.54	41.1	1002.05	38.05	448.56
22	14.9	52.05	1015.11	77.33	464.82
23	33.8	64.96	1004.88	49.37	427.28
24	25.37	68.31	1011.12	70.99	441.76
25	7.29	41.04	1024.06	89.19	474.71
26	13.55	40.71	1019.13	75.44	467.21
27	6.39	35.57	1025.53	77.23	487.69
28	26.64	62.44	1011.81	72.46	438.67
29	7.84	41.39	1018.21	91.92	485.66
30	21.82	58.66	1011.71	64.37	452.16
31	27.17	67.45	1015.67	49.03	429.87
32	13.42	41.23	994.17	95.79	468.82
33	20.77	56.85	1012.4	83.63	442.85
34	8.29	36.08	1020.38	81.53	483.26
35	30.98	67.45	1015.18	45.4	433.59



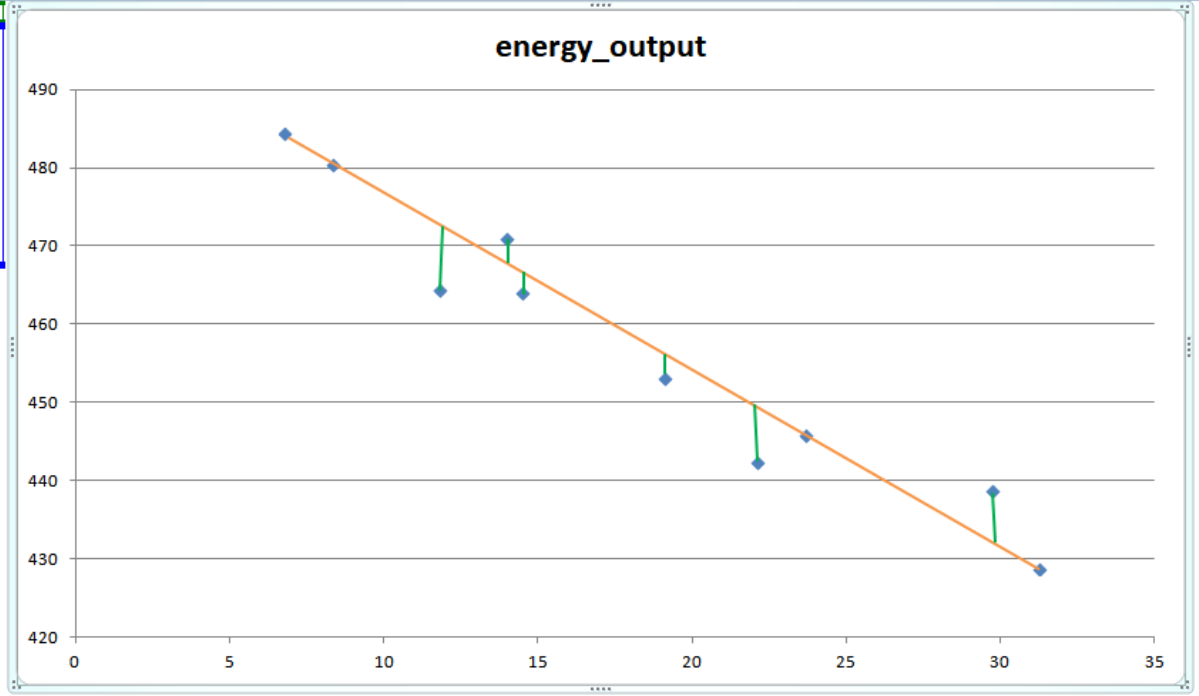
- However when we plot more data points from the dataset most of the points fall *around* our line with only few falling exactly on the line.
- So with the larger dataset we have to find the **best fit**.
- And the best fit is the line that is as close as possible to all the points.

	A	B	C	D	E
1	temperature	exhaust_vacuum	ambient_pressure	relative_humidity	energy_output
2	8.34	40.77	1010.84	90.01	480.48
3	23.64	58.49	1011.4	74.2	445.75
4	29.74	56.9	1007.15	41.91	438.76
5	19.07	49.69	1007.22	76.79	453.09
6	11.8	40.66	1017.13	97.2	464.43
7	13.97	39.16	1016.05	84.6	470.96
8	22.1	71.29	1008.2	75.38	442.35
9	14.47	41.76	1021.98	78.41	464
10	31.25	69.51	1010.25	36.83	428.77
11	6.77	38.18	1017.8	81.13	484.31
12	28.28	68.67	1006.36	69.9	435.29
13	22.99	46.93	1014.15	49.42	451.41
14	29.3	70.04	1010.95	61.23	426.25
15	8.14	37.49	1009.04	80.33	480.66
16	16.92	44.6	1017.34	58.75	460.17
17	22.72	64.15	1021.14	60.34	453.13
18	18.14	43.56	1012.83	47.1	461.71
19	11.49	44.63	1020.44	86.04	471.08
20	9.94	40.46	1018.9	68.51	473.74
21	23.54	41.1	1002.05	38.05	448.56
22	14.9	52.05	1015.11	77.33	464.82
23	33.8	64.96	1004.88	49.37	427.28
24	25.37	68.31	1011.12	70.99	441.76
25	7.29	41.04	1024.06	89.19	474.71
26	13.55	40.71	1019.13	75.44	467.21
27	6.39	35.57	1025.53	77.23	487.69
28	26.64	62.44	1011.81	72.46	438.67
29	7.84	41.39	1018.21	91.92	485.66
30	21.82	58.66	1011.71	64.37	452.16
31	27.17	67.45	1015.67	49.03	429.87
32	13.42	41.23	994.17	95.79	468.82
33	20.77	56.85	1012.4	83.63	442.85
34	8.29	36.08	1020.38	81.53	483.26
35	30.98	67.45	1015.18	45.4	433.59



- Now let us see what is meant by *as close as possible*.
- It means that the distance between the data points and the line (shown in green below) should be minimal.
- The sum of squares of these residual distances (Residual Sum of Squares – RSS) is taken as the standard measure for this purpose.
- So the learning process involves arriving at the coefficients (***a*** and ***b*** in our example) such that this quantity – RSS, is minimized.

	A	B	C	D	E
1	temperature	exhaust_vacuum	ambient_pressure	relative_humidity	energy_output
2	8.34	40.77	1010.84	90.01	480.48
3	23.64	58.49	1011.4	74.2	445.75
4	29.74	56.9	1007.15	41.91	438.76
5	19.07	49.69	1007.22	76.79	453.09
6	11.8	40.66	1017.13	97.2	464.43
7	13.97	39.16	1016.05	84.6	470.96
8	22.1	71.29	1008.2	75.38	442.35
9	14.47	41.76	1021.98	78.41	464
10	31.25	69.51	1010.25	36.83	428.77
11	6.77	38.18	1017.8	81.13	484.31
12	28.28	68.67	1006.36	69.9	435.29
13	22.99	46.93	1014.15	49.42	451.41
14	29.3	70.04	1010.95	61.23	426.25
15	8.14	37.49	1009.04	80.33	480.66
16	16.92	44.6	1017.34	58.75	460.17
17	22.72	64.15	1021.14	60.34	453.13
18	18.14	43.56	1012.83	47.1	461.71
19	11.49	44.63	1020.44	86.04	471.08
20	9.94	40.46	1018.9	68.51	473.74
21	23.54	41.1	1002.05	38.05	448.56
22	14.9	52.05	1015.11	77.33	464.82
23	33.8	64.96	1004.88	49.37	427.28
24	25.37	68.31	1011.12	70.99	441.76
25	7.29	41.04	1024.06	89.19	474.71
26	13.55	40.71	1019.13	75.44	467.21
27	6.39	35.57	1025.53	77.23	487.69
28	26.64	62.44	1011.81	72.46	438.67
29	7.84	41.39	1018.21	91.92	485.66
30	21.82	58.66	1011.71	64.37	452.16
31	27.17	67.45	1015.67	49.03	429.87
32	13.42	41.23	994.17	95.79	468.82
33	20.77	56.85	1012.4	83.63	442.85
34	8.29	36.08	1020.38	81.53	483.26
35	30.98	67.45	1015.18	45.4	433.59



- As we have more than one input parameter in the dataset, we will have a vector of values instead of a single value in the function. And the relationship to be learned will become:

  $y = ax1 + bx2 + cx3 + . . . + k$

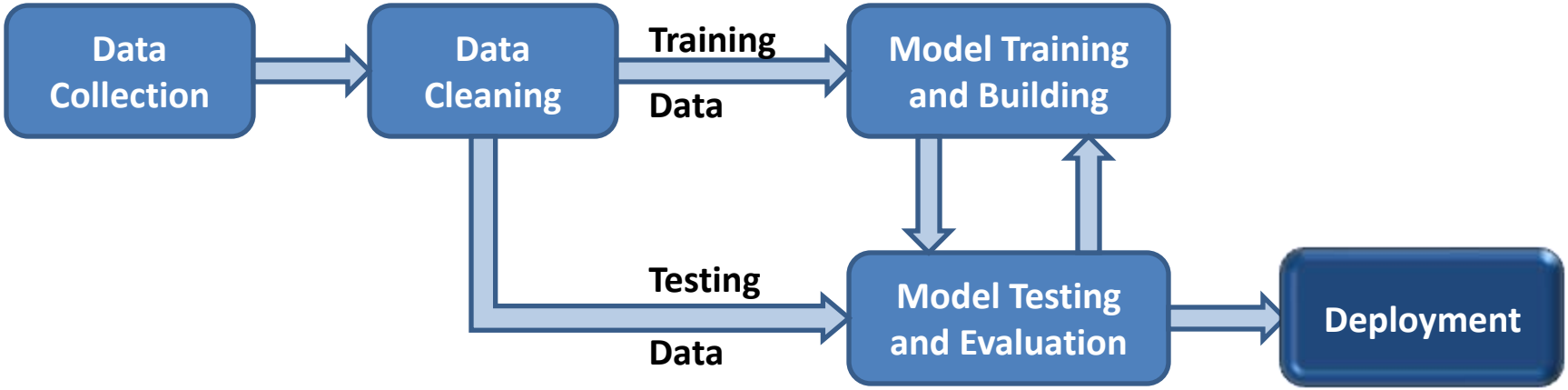
- Or more formally it is often represented using the character *beta* for the coefficients, as given below or it can be written in Sigma Σ notation also.

  $y = \beta 0 + \beta 1x1 + \beta 2x2 + \beta 3x3 + . . . + \beta nxn$

- Using Spark MLlib APIs we can create a `LinearRegression` object and pass the input data in the format required on which it trains and builds the model.
- Basically we will put the features and the corresponding labels in a dataframe; we will then call `fit()` method of the object passing the dataframe as the input parameter.
- It returns the model as the output after the learning process i.e. deriving the beta coefficients $\beta 0, \beta 1, . . ., \beta n$ from the training data passed.

- It is not uncommon to have some *noise* in the data i.e. some data points that do not represent the true properties of the dataset but are present due to random chance more as aberrations.
- If the learning process tries and accommodates these points also into the model then it will lead to overfitting. That is the model becomes highly tailored to the training dataset and so, for the new inputs the predicted output will be inaccurate.
- To avoid overfitting another quantity is added to the function factoring in the coefficients with goal of constraining them from becoming too large. This is called *Regularization.*
- Spark MLlib API for Linear Regression provides for Regularization Parameter to be specified while initializing the model.

- To train and build the model, the MLlib APIs require the historical data to be formatted into a dataframe with 2 columns – Features and Label.
- Though this is an additional task it is necessary and worthwhile because Spark MLlib can handle large scale dataframes which are distributed across the cluster nodes and processed & analysed in parallel.
- Also MLlib provides a number of APIs for all the operations necessary in the process of machine learning model building and deployment.
- Let us take a look at the steps of the process.



- **Data collection:** In a project if the data comes from different sources or different tables from the same source database then the data needs to be gathered and collated. If this needs sizeable effort then it can be done as a separate exercise.

- **Data cleaning:** Most often before formatting the data into the dataframe as required by the MLlib APIs we need to make sure that the data is clean. This involves the right data types are used for the columns. Converting some of categorical variables say income group from string type of *''low, 'medium'* and *'high'* to numeric type like 0, 1 and 2 and similar tasks. Spatk MLlib provides several APIs as part of Pipelines, Extracting, Transforming features for these tasks.

- Once the data is cleaned and transformed into dataframe it is split into two parts as training data and test data usually in 70:30 or 80:20 ratio.

- **Model Training and Building:** The training data is used for the learning, model training and model building. Here as well MLlib makes these operations easy by providing components like Estimator which work on large scale dataframes in distributed fashion under the hood. For example `LinearRegression` is the Estimator in this case. It abstracts the concept of the learning algorithm that fits or trains on data and generates the Model. Code-wise, the Estimator `LinearRegression` implements the method `fit()`, which accepts the DataFrame and produces the Model.

- **Model Testing and Evaluation:** MLlib provides an abstraction Transformer which includes not only feature transformer that formats dataframes as per requirements but also the models. The model created in the previous step takes an input dataframe, reads the feature vector column, applies the model to perform prediction for each feature vector and outputs a new dataframe which contains the additional column of predictions.
- For evaluation in the case of Linear Regression models generally metrics like Root Mean Square Error (RMSE), Mean Square Error (MSE), Mean Absolute Error (MEA) etc are used. We can use model.summary to get the summary metrics. Moreover MLlib provides the abstraction Evaluator for the purpose of getting the relevant metrics for different algorithms. For example in this case we can use RegressionEvaluator which provides all the metrics to evaluate the efficiency of our model.