

## **EXPERIMENT 7**

**Aim:** Experiment to implement a regression model using Rapid Miner and Python.

### **To Do:**

1. Preprocess data. Split data into train and test set
2. Build Regression model using inbuilt library function on training data
3. Calculate metrics based on test data using inbuilt function
4. Build Regression model using a function defined by a student( model to be built using the same training data).
5. Calculate metrics based on test data using inbuilt function
6. Compare the results of all three ways of implementation.(Rapid Miner, Python Library)

### **Theory:**

Regression is a form of a supervised machine learning technique that tries to predict any continuous valued attribute. It analyzes the relationship between a target variable (dependent) and its predictor variable (independent). Regression is an important tool for data analysis that can be used for time series modeling, forecasting, and others. Regression data mining techniques are of varied types and help cover a broad spectrum of prediction and impact assumptions that are later useful for curating machine learning datasets.

Regression involves the process of fitting a curve or a straight line on various data points. It is done in such a way that the distances between the curve and the data points come out to be the minimum.

### **Types of Regression:**

Regression is divided into five different types

1. Linear Regression
2. Logistic Regression

3. Lasso Regression
4. Ridge Regression
5. Polynomial Regression

## **Linear Regression**

Linear regression is the type of regression that forms a relationship between the target variable and one or more independent variables utilizing a straight line. The given equation represents the equation of linear regression

$$Y = a + b * X + e.$$

Where,

a represents the intercept

b represents the slope of the regression line

e represents the error

X and Y represent the predictor and target variables, respectively.

If X is made up of more than one variable, termed as multiple linear equations.

In linear regression, the best fit line is achieved utilizing the least squares method, and it minimizes the total sum of the squares of the deviations from each data point to the line of regression. Here, the positive and negative deviations do not get canceled as all the deviations are squared.

## **Polynomial Regression**

If the power of the independent variable is more than 1 in the regression equation, it is termed a polynomial equation. With the help of the example given below, we will understand the concept of polynomial regression.

$$Y = a + b * x^2$$

In the particular regression, the best fit line is not considered a straight line like a linear equation; however, it represents a curve fitted to all the data points.

Applying linear regression techniques can lead to overfitting when you are tempted to minimize your errors by making the curve more complex. Therefore, always try to fit the curve by generalizing it to the issue.

## **Logistic Regression**

When the dependent variable is binary in nature, i.e., 0 and 1, true or false, success or failure, the logistic regression technique comes into existence. Here, the target value (Y) ranges from 0 to 1, and it is primarily used for classification-based problems. Unlike linear regression, it does not need any independent and dependent variables to have a linear relationship.

## **Ridge Regression**

Ridge regression refers to a process that is used to analyze various regression data that have the issue of multicollinearity. Multicollinearity is the existence of a linear correlation between two independent variables.

Ridge regression exists when the least square estimates are the least biased with high variance, so they are quite different from the real value. However, by adding a degree of bias to the estimated regression value, the errors are reduced by applying ridge regression.

## **Lasso Regression**

The term LASSO stands for Least Absolute Shrinkage and Selection Operator. Lasso regression is a linear type of regression that utilizes shrinkage. In Lasso regression, all the data points are shrunk towards a central point, also known as the mean. The lasso process is most fitted for simple and sparse models with fewer parameters than other regression. This type of regression is well fitted for models that suffer from multicollinearity.

## **Preprocessing Data**

Preprocessing data is an important step in preparing data for a regression analysis. Here are some common preprocessing steps for regression:

- Handling missing data
- Encoding categorical variables
- Feature scaling
- Removing outliers
- Feature selection
- Splitting data

## **Evaluating Regression Models**

There are three error metrics that are commonly used for evaluating and reporting the performance of a regression model; they are:

1. Mean Squared Error (MSE).
2. Root Mean Squared Error (RMSE).
3. Mean Absolute Error (MAE)

### **Mean Squared Error**

Mean Squared Error, or MSE for short, is a popular error metric for regression problems.

It is also an important loss function for algorithms fit or optimized using the least squares framing of a regression problem. Here “least squares” refers to minimizing the mean squared error between predictions and expected values.

The MSE is calculated as the mean or average of the squared differences between predicted and expected target values in a dataset.

$$\text{MSE} = 1 / N * \sum \text{for } i \text{ to } N (y_i - \hat{y}_i)^2$$

Where  $y_i$  is the  $i$ 'th expected value in the dataset and  $\hat{y}_i$  is the  $i$ 'th predicted value. The difference between these two values is squared, which has the effect of removing the sign, resulting in a positive error value.

## **Root Mean Squared Error**

The Root Mean Squared Error, or RMSE, is an extension of the mean squared error.

Importantly, the square root of the error is calculated, which means that the units of the RMSE are the same as the original units of the target value that is being predicted.

For example, if your target variable has the units “dollars,” then the RMSE error score will also have the unit “dollars” and not “squared dollars” like the MSE.

As such, it may be common to use MSE loss to train a regression predictive model, and to use RMSE to evaluate and report its performance.

The RMSE can be calculated as follows:

$$\text{RMSE} = \sqrt{1 / N * \sum \text{for } i \text{ to } N (y_i - \hat{y}_i)^2}$$

Where  $y_i$  is the  $i$ 'th expected value in the dataset,  $\hat{y}_i$  is the  $i$ 'th predicted value, and  $\sqrt{\phantom{x}}$  is the square root function.

## **Mean Absolute Error**

Mean Absolute Error, or MAE, is a popular metric because, like RMSE, the units of the error score match the units of the target value that is being predicted.

Unlike the RMSE, the changes in MAE are linear and therefore intuitive.

That is, MSE and RMSE punish larger errors more than smaller errors, inflating or magnifying the mean error score. This is due to the square of the error value. The MAE does not give more or less weight to different types of errors and instead the scores increase linearly with increases in error.

As its name suggests, the MAE score is calculated as the average of the absolute error values. Absolute or `abs()` is a mathematical function that simply makes a number positive. Therefore, the difference between an expected and predicted value may be positive or negative and is forced to be positive when calculating the MAE.

The MAE can be calculated as follows:

$$\text{MAE} = 1 / N * \sum \text{for } i \text{ to } N \text{ abs}(y_i - \hat{y}_i)$$

Where  $y_i$  is the  $i$ 'th expected value in the dataset,  $\hat{y}_i$  is the  $i$ 'th predicted value and `abs()` is the absolute function.

### **What is Rapidminer?**

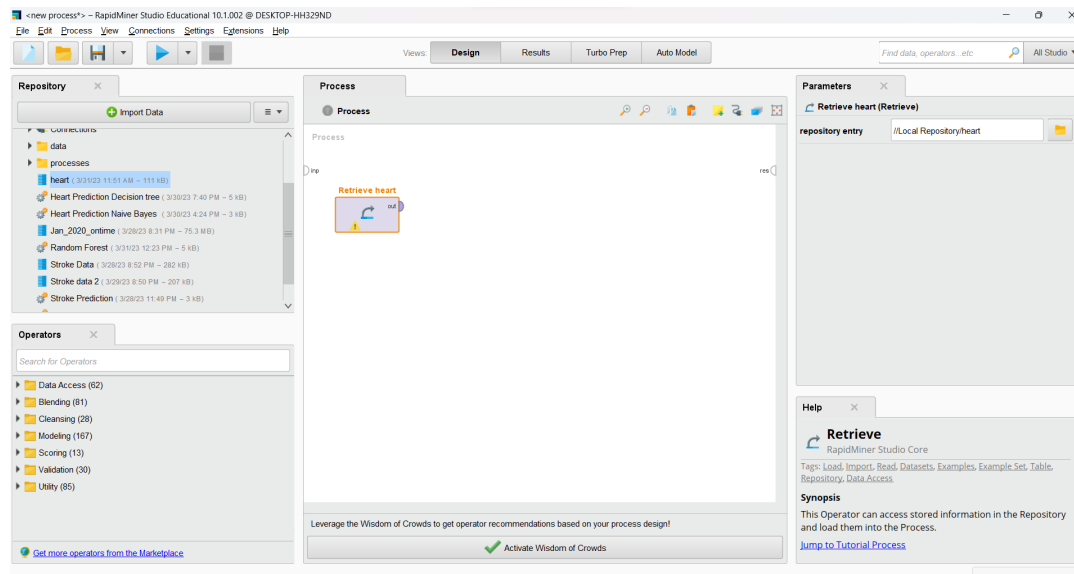
Rapidminer is a comprehensive data science platform with visual workflow design and full automation. It means that we don't have to do the coding for data mining tasks. Rapidminer is one of the most popular data science tools.

This is the graphical user interface of the blank process in rapidminer. It has the repository that holds our dataset. We can import our own datasets. It also offers many public datasets that we can try. We can also work with a database connection.

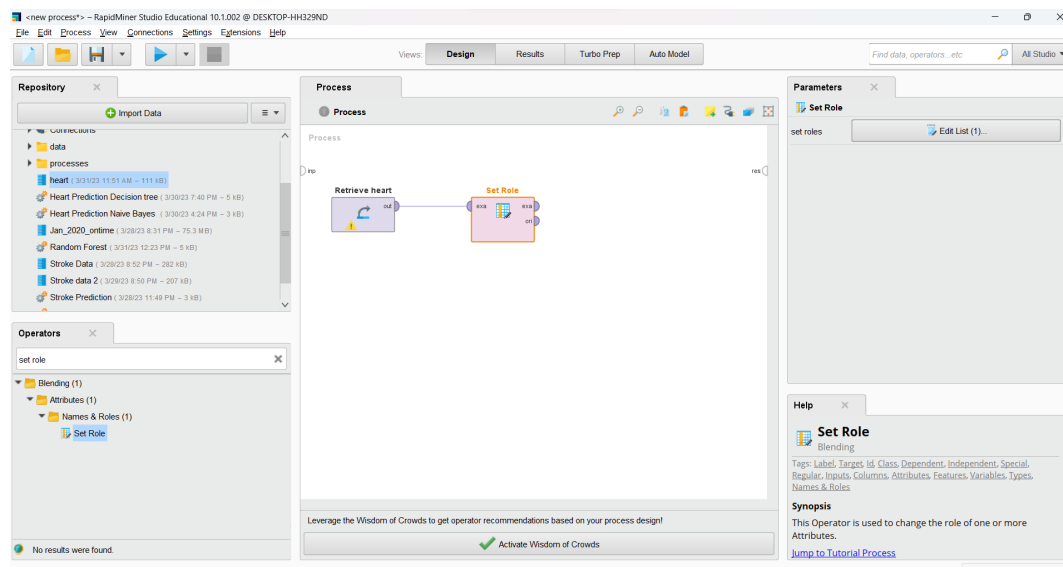
Below the repository window, it has an operator. The operators include everything we need to build a data mining process, such as data access, data cleansing, modeling, validation, and scoring.

# Implementation using RapidMiner:

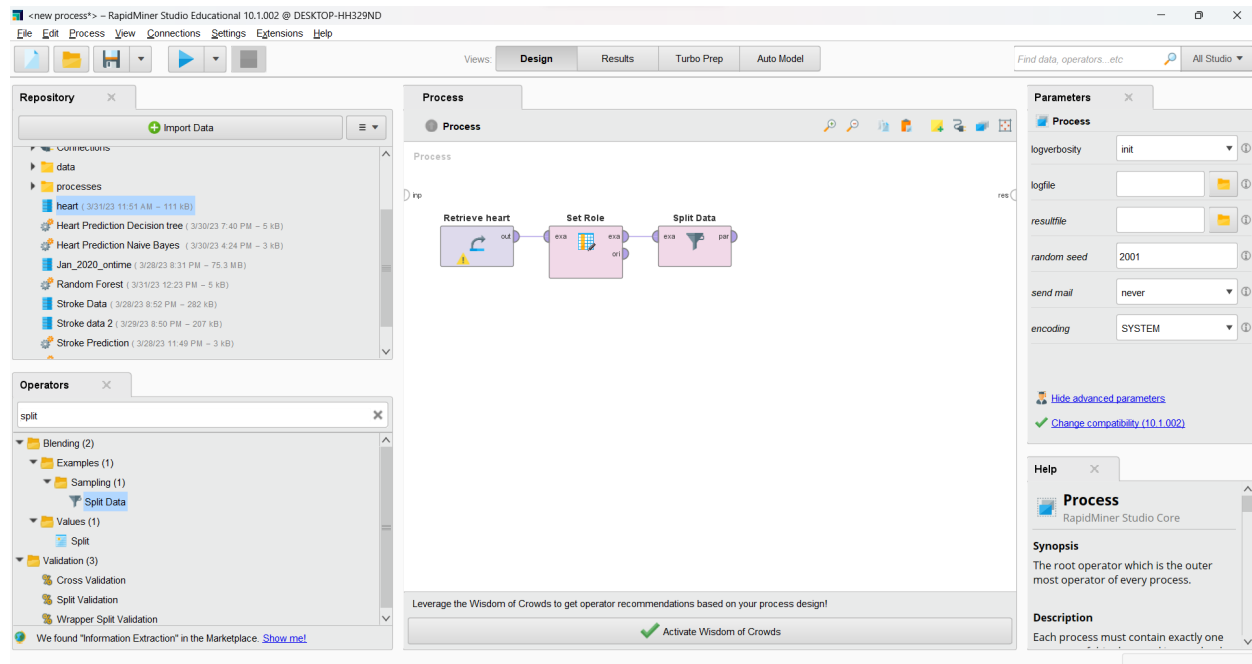
## Import the required dataset



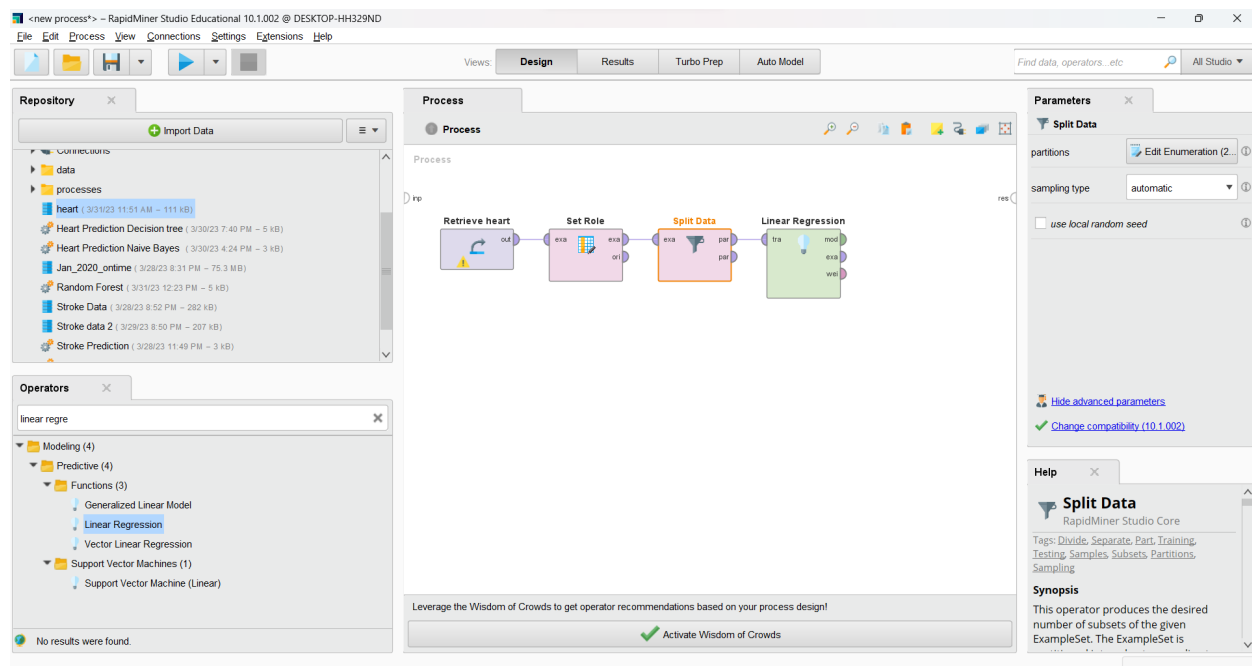
## Add set role operator by selecting the attribute as target



Then we will split the data into train and test data as 70% and 30%



Add Linear regression operator and perform the given connections to apply model performance as shown below.





**<new process>\* - RapidMiner Studio Educational 10.1.002 @ DESKTOP-IH4329ND**

File Edit Process View Connections Settings Extensions Help

Views: Design Results Turbo Prep Auto Model

Find data, operators...etc All Studio

**Repository**

- Connections
- data
- processes
  - heart (3/30/23 11:51 AM - 115 KB)
  - Heart Prediction Decision tree (3/30/23 7:40 PM - 5 KB)
  - Heart Prediction Naive Bayes (3/30/23 4:24 PM - 3 KB)
  - Jan\_2020\_online (3/29/23 8:31 PM - 75.3 MB)
  - Random Forest (3/30/23 12:23 PM - 5 KB)
  - Stroke Data (3/29/23 8:52 PM - 282 KB)
  - Stroke data 2 (3/29/23 8:58 PM - 207 KB)
  - Stroke Prediction (3/29/23 11:49 PM - 3 KB)

**Operators**

apply mo

- Modeling (1)
  - Time Series (1)
    - Forecasting (1)
      - Apply Forecast
  - Scoring (1)
    - Apply Model

We found "Shapelet" in the Marketplace. [Show me!](#)

**Process**

Process

Retrieve heart → Set Role → Split Data → Linear Regression → Apply Model

**Parameters**

**Split Data**

partitions Edit Enumeration (2) ①

sampling type automatic ①

☐ use local random seed ①

[Hide advanced parameters](#)

[Change compatibility \(10.1.002\)](#)

**Help**

**Split Data**

RapidMiner Studio Core

Tags: Divide, Separate, Part, Training, Testing, Samples, Subsets, Partitions, Sampling

**Synopsis**

This operator produces the desired number of subsets of the given ExampleSet. The ExampleSet is

Leverage the Wisdom of Crowds to get operator recommendations based on your process design!

✓ Activate Wisdom of Crowds

**//Local Repository/exp7dmbi\* - RapidMiner Studio Educational 10.1.001 @ Adiii**

File Edit Process View Connections Settings Extensions Help

Views: Design Results Turbo Prep Auto Model

Find data, operators...etc All Studio

**Repository**

- Connections
- data
- processes
  - diabetes (4/11/23 12:29 AM - 58 KB)
  - diabetes1 (4/11/23 1:22 AM - 59 KB)
  - exp7dmbi (4/11/23 2:21 AM - 5 KB)
  - insurance (4/1/23 4:27 PM - 6 KB)
  - naivebayes (3/30/23 5:20 PM - 7 KB)

**Operators**

performance

- Performance (r)
- Performance (Classification)
- Performance (Binominal Classi
- Performance (Regression)
- Performance (Costs)
- Performance (Ensemble)

We found "Model Management" in the Marketplace. [Show me!](#)

**Process**

Process

Set Role → Split Data → Linear Regression → Performance → Apply Model

**Parameters**

**Linear Regression**

feature selection M5 prime ①

☒ eliminate colinear features ①

min tolerance 0.05 ①

☒ use bias ①

ridge 1.0E-8 ①

[Hide advanced parameters](#)

**Help**

**Linear Regression**

RapidMiner Studio Core

Tags: Supervised, Classification, Regression, Model, Least squares, Ordinary, Ridge, OLS, Glim, Generalized, Functions

**Synopsis**

Leverage the Wisdom of Crowds to get operator recommendations based on your process design!

✓ Activate Wisdom of Crowds

Local Repository/exp7dmbi\* - RapidMiner Studio Educational 10.1.001 @ Adiii

File Edit Process View Connections Settings Extensions Help

Views: Design Results Turbo Prep Auto Model

Find data, operators...etc All Studio

ExampleSet (Apply Model) PerformanceVector (Performance)

Result History ExampleSet (Split Data)

Open in Turbo Prep Auto Model Filter (230 / 230 examples): all

Row No.	Outcome	prediction(O...	Pregnancies	Glucose	BloodPress...	SkinThickne...	Insulin	BA
1	0	0.019	1	85	66	29	0	26
2	0	-0.030	1	89	66	23	94	28
3	1	0.923	0	137	40	35	168	43
4	1	0.696	2	197	70	45	543	30
5	1	0.378	0	118	84	47	230	45
6	1	0.258	7	107	74	0	0	29
7	1	0.299	1	115	70	30	96	34
8	1	0.467	10	125	70	26	115	31
9	1	0.210	2	90	68	42	0	38
10	1	0.625	4	111	72	47	207	37
11	0	0.616	7	133	84	0	0	40
12	1	0.918	9	171	110	24	240	45

ExampleSet (230 examples, 2 special attributes, 8 regular attributes)

Repository

Import Data

- Training Resources (connected)
- Samples
- Community Samples (connected)
- Local Repository (Local)
  - Connections
  - data
  - processes
    - diabetes (4/11/23 12:29 AM - 58 kB)
    - diabetes1 (4/11/23 1:22 AM - 59 kB)
    - exp7dmbi (4/11/23 2:21 AM - 5 kB)
    - insurance (4/11/23 4:27 PM - 6 kB)
    - naivebayes (3/30/23 5:20 PM - 7 kB)
  - DB (Legacy)

Local Repository/exp7dmbi\* - RapidMiner Studio Educational 10.1.001 @ Adiii

File Edit Process View Connections Settings Extensions Help

Views: Design Results Turbo Prep Auto Model

ExampleSet (Apply Model) PerformanceVector (Performance)

Result History ExampleSet (Split Data)

Criterion

- root mean squared error
- absolute error
- relative error
- correlation**

**correlation**

correlation: 0.557

Local Repository/exp7dmbi\* - RapidMiner Studio Educational 10.1.001 @ Adiii

File Edit Process View Connections Settings Extensions Help

Views: Design Results Turbo Prep

ExampleSet (Apply Model) PerformanceVector (Performance)

Result History ExampleSet (Split Data)

Criterion

- root mean squared error
- absolute error**
- relative error
- correlation

**absolute\_error**

absolute\_error: 0.321 +/- 0.220

//Local Repository/exp7dmbi\* – RapidMiner Studio Educational 10.1.001 @ Adiii

File Edit Process View Connections Settings Extensions Help

Views: Design Results Turbo

ExampleSet (Apply Model) PerformanceVector (P

Result History ExampleSet (Split Da

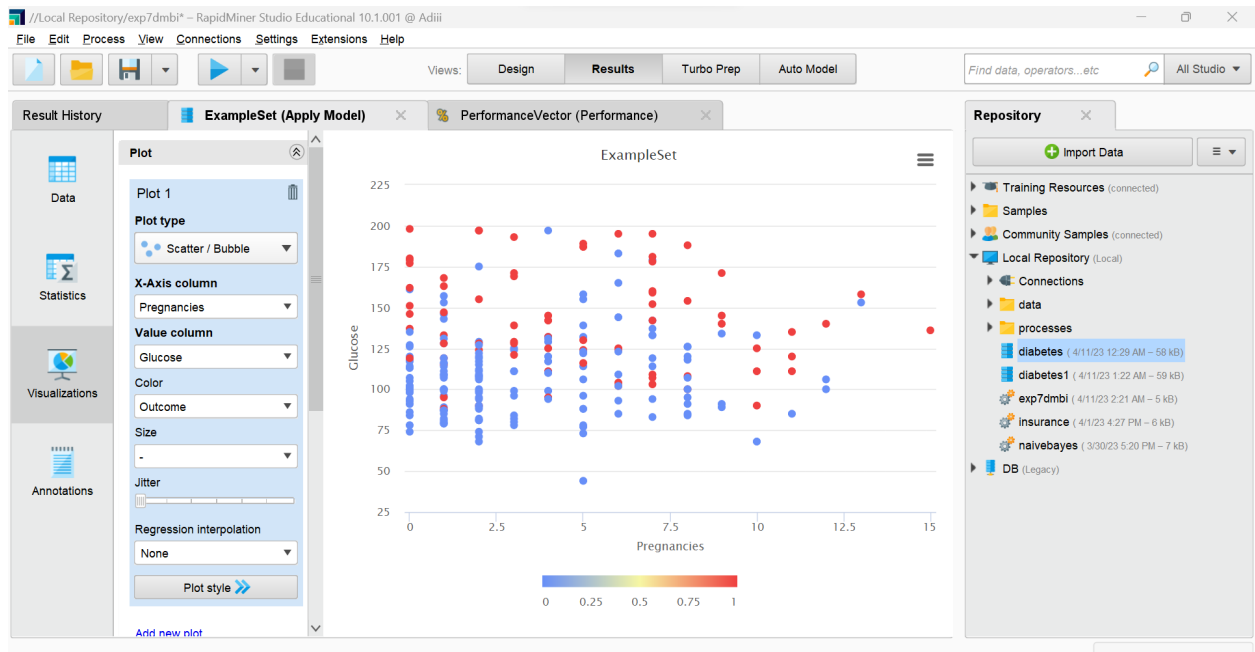
Performance

Criterion

- root mean squared error
- absolute error
- relative error
- correlation

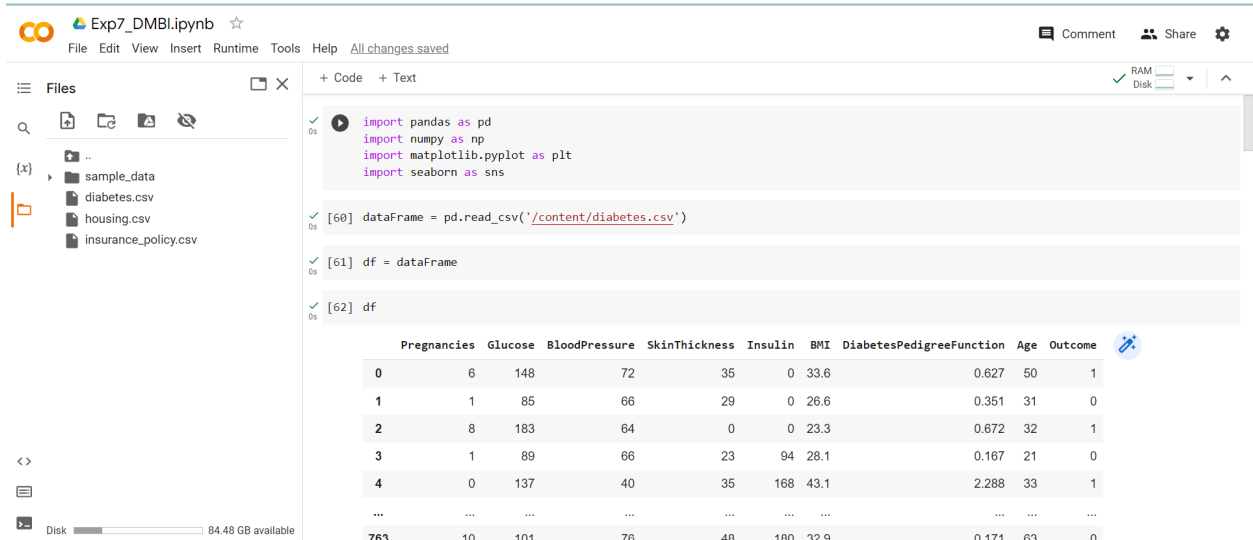
# root\_mean\_squared\_error

root\_mean\_squared\_error: 0.389 +/- 0.000



# Implementation using Python:

## Importing libraries and loading dataset



The Jupyter Notebook interface shows the following code and output:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

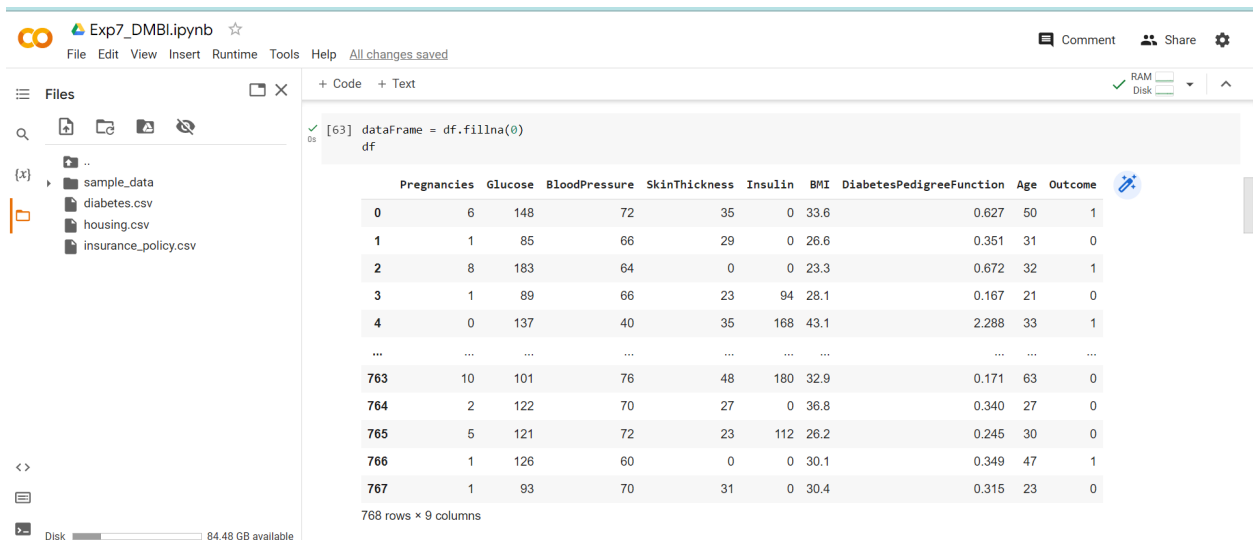
```
[60] dataframe = pd.read_csv('/content/diabetes.csv')
```

```
[61] df = dataframe
```

```
[62] df
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1
...	...	...	...	...	...	...	...	...	...
763	10	101	76	48	180	32.9	0.171	63	0

## Checking null values



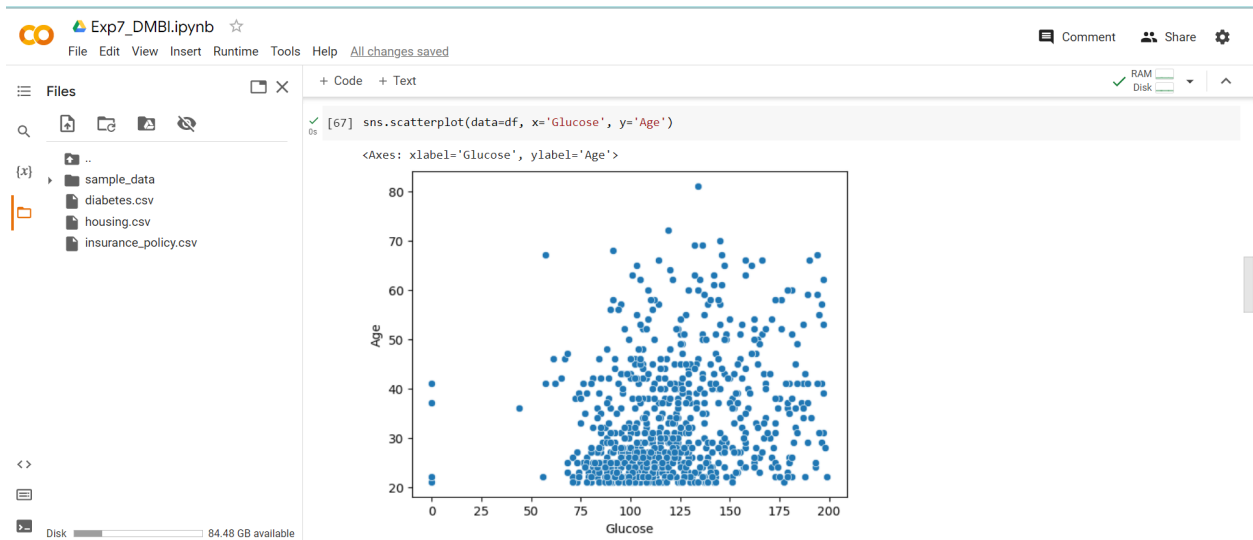
The Jupyter Notebook interface shows the following code and output:

```
[63] dataframe = df.fillna(0)
df
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1
...	...	...	...	...	...	...	...	...	...
763	10	101	76	48	180	32.9	0.171	63	0
764	2	122	70	27	0	36.8	0.340	27	0
765	5	121	72	23	112	26.2	0.245	30	0
766	1	126	60	0	0	30.1	0.349	47	1
767	1	93	70	31	0	30.4	0.315	23	0

768 rows x 9 columns

## Scatter plot for glucose and age



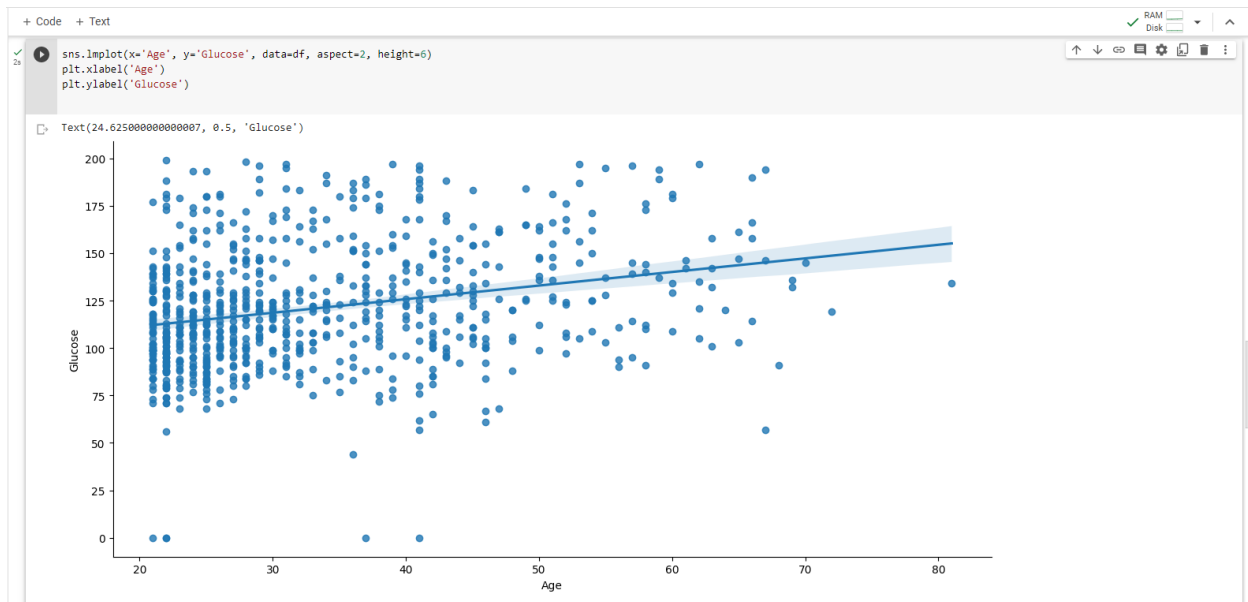
## Training the model

```
[68] import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn import tree

[69] from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Ridge
from sklearn.ensemble import RandomForestRegressor
models = {
    "Linear Regression": LinearRegression(),
    "Ridge": Ridge(),
    "Random Forest": RandomForestRegressor()
}

[71] X=df[['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI', 'DiabetesPedigreeFunction', 'Age']]
y=df['Outcome']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

## Linear regression graph for Glucose and Age:



## Mean square error for this model

```
[12] X=df[['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI', 'DiabetesPedigreeFunction', 'Age']]
      y=df['Outcome']
      X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

[13] clf1 = LinearRegression()
      clf1.fit(X_train, y_train)

[14] def modelAccuracy(models,X_train, X_test, y_train, y_test):
      np.random.seed(1)
      model_Score = {}
      for name, model in models.items():
          model.fit(X_train,y_train)
          model_Score[name] = model.score(X_test,y_test)*100
      return model_Score

[15] from sklearn.metrics import mean_squared_error
      y_pred1=clf1.predict(X_test)
      score=mean_squared_error(y_pred1,y_test)
      print(score)
```

0.17104527280850104

## Implementation using Self defined function:

```
✓ 0s [▶] from sklearn.base import BaseEstimator, RegressorMixin
      from sklearn.metrics import mean_squared_error

class LinearRegressor(BaseEstimator, RegressorMixin):
    def __init__(self, parameter1=1, parameter2=2):
        self.parameter1 = parameter1
        self.parameter2 = parameter2

    def fit(self, X, y):
        # Implement your own fitting method here
        # This example just calculates the mean of y
        self.y_mean = np.mean(y)
        return self

    def predict(self, X):
        # Implement your own prediction method here
        # This example just returns the mean of y for all instances
        y_pred = np.full((X.shape[0],), self.y_mean)
        return y_pred

    def score(self, X, y):
        # Implement your own scoring method here
        # This example just calculates the mean squared error
        y_pred = self.predict(X)
        mse = mean_squared_error(y, y_pred)
        return -mse # return negative MSE for use with GridSearchCV's default scoring

✓ 0s [▶] custom_Reg = LinearRegressor(parameter1=0.5,parameter2=0.1)
      custom_Reg.fit(X_train,y_train)

      y_pred = custom_Reg.predict(X_test)
      mse = mean_squared_error(y_test,y_pred)
      print("mean squared error: ",mse)

      mean squared error:  0.2296966394489976
```

## Comparison for linear regression :-

Rapid Miner	0.389
Python	0.1712
Self Defined	0.2296

**Conclusion:**

Hence, we compared the mean square error of each method. The performance of the Python method is the best followed by Python and self defined function since Python has the least mean squared error. We successfully implemented a Linear regression model using Rapid Miner and Python.