Data pre-processing and its use for prediction

Dr. Anuradha Misra,
Amity University Uttar Pradesh, Lucknow Campus, amisra@lko.amity.edu,
Ms. Soumya Chandra,
Amity University Uttar Pradesh, Lucknow Campus, soumya1@student.amity.edu

Dr. Praveen Kumar Misra,
Department of Mathematics & Statistics, Dr. Shakuntala Misra National Rehabilitation University, Lucknow, praveenkumarmisra@gmail.com,

Dr. Amod Kumar Tiwari,
Associate Professor, Rajkiya Engineering College Sonbhadra, amod.tiwari@recsonbhadra.ac.in

Abstract: Information is now available in an overabundance, so much business and other popular industries has become very problematic with huge amount of data. In the past, the collection and storage of information was the primary issue. Currently, there are massive amounts of data both structured and unstructured, that need to be analyzed in an iterative, as well as in a time sensitive manner. In response to this need, data analytics tools and services have emerged as a means to solve this problem. Handling of data analytics with a modern development environment, makes easily accessible without coding.

The main aim of this research is to efficiently read the data and draw conclusion and interpret that would help in betterment. Here we use data preprocessing and simple linear regression tools to understand and study the trends in the data. Data preprocessing is an important step for initial preparation of data for further use. Simple linear regression model uses data preprocessing and the concept is to state relation between dependent variable and two or more independent variables using the best fit line. It is a linear estimation.

Keywords: linear regression, data preprocessing

INTRODUCTION TO DATA ANALYTICS

A data is basically factual information that can be presented in the form of records, characters, numbers, images and many other different ways of recording them. A data independently has no meaning use unless it its interpreted. On close examination of data we find patterns to assess the

information and then it can be used to enhance our knowledge. Data is growing exponentially and accumulating. Up until the dawn of time of 2005, we humans have created 130exabytes of data, by the end of 2010 it was about 1200 exabytes and by the end of 2020 it is estimated to be 40,900 exabytes of data and the pattern shows exponential growth of data over years according to the IDC's study.

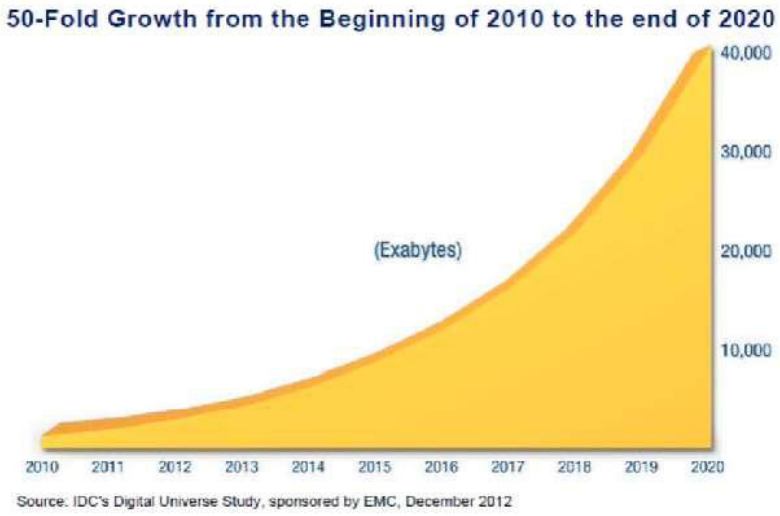

Figure 1: Growth Rate of Data from 2010 to 2020

Data analytics is the science of studying raw data with the aim to draw important conclusions from it. Data analytics uses statistical operations, research and management tools.[1]

USAGE OF DATA ANALYTICS IN DIFFERENT FIELDS

Data analysis is being broadly used in the many sectors like,

i. Banking and finance industries for prediction of market trends and risks and also detecting frauds and improve efficiency.

ii. With the combined usage of statistics with data analysis has always been helpful in finding trends in complex systems.

iii. Data analytics has extensive use in healthcare, in prediction of patient results, in betterment of diagnostic methods are just a few examples.

Why Python for data analytics?

Python can be used as one of the languages for data analytics as:

i. A good general-purpose programming language that is easy to use and learn.[1]
ii. Includes many libraries for scientific computing including matplotlib

DATA PREPROCESSING

Data Pre-processing is an important initial step. As data is present in incomplete and inconsistent state or contains errors, data pre-processing technique helps to reconstructs any raw data into meaningful and understandable form as well as solving those problems. The presence of redundant information or irrelevant information can increase the filtration process. Data pre-processing is mostly done on database-driven applications.[2]

It consists of certain tools:

1. Importing the libraries

Figure 2: Importing Libraries

Here, we see how to get the libraries ready for the future machinery model. We will import three libraries namely:

i. NumPy: It is an open source primary package used for scientific calculations in Python that allows us to work with multidimensional arrays and matrices. The key data structure of NumPy is NumPy array and all data are stored in arrays. They are faster than python lists. In the above code, we import the numpy library and for convenience denote it as np to call it faster.

ii. **Matplotlib:** This library allows us to plot some graphs, histogram, scatter-plots, charts and many more for data visualisation. Library is a collection of modules and here we are interested in particular module called 'pyplot' that helps to draw charts. So, in the above code we not only going to import matplotlib but particularly pyplot and to access it we add a dot in between matplotlib and pyplot as again we add a shortcut to it as 'plt'.

iii. **Pandas:** Pandas is among the most popular libraries for analysing data that provides with extremely optimized performance. It allows us to not only import dataset but also create matrix of features (independent variable) and dependent variable. It's useful in pre-processing dataset and adding a shortcut to it as pd (in the code above).

2. Importing dataset

The data table being used for data pre-processing is mentioned below with title 'Data.csv'.

	A	B	C	D
1	Country	Age	Salary	Purchased
2	France	44	72000	No
3	Spain	27	48000	Yes
4	Germany	30	54000	No
5	Spain	38	61000	No
6	Germany	40		Yes
7	France	35	58000	Yes
8	Spain		52000	No
9	France	48	79000	Yes
10	Germany	50	83000	No
11	France	37	67000	Yes

Table 1 : Dataset for preprocessing

To import the above dataset and integrate it,

1) Firstly, we will create a new variable named 'dataset' and this variable will be equal to output of certain function of pandas. It will read all the values from the dataset and create a data frame with same rows and columns. Since we are going to pandas library to read the

data using pandas function (read_csv ('Data.csv')) we first call the pandas itself and connect the function with dot.

2) Then, we create two new entities, one for matrix of features (named 'x') and other for independent variable (named 'y').
    i) X represents features that are columns that help to determine the dependent variable. In the dataset above Column Country, Age, Salary are features.
    ii) y represents dependent variable (here, Column Purchased) which needs to be predicted with the help of independent variable or feature.

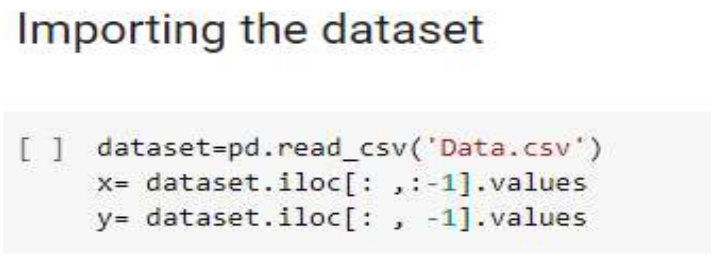

Figure 3: Importing Dataset

For x, we will take the indexes for the first three columns except the last one. We will use the variable dataset and one of the attributes of pandas data frame (iloc []) that will help extract indexes of rows and columns. As we want all the rows of first three column we add colon (:) which means all rows separating it with comma to mention columns required as colon -1 which means as we haven't mentioned any lower bound its 0 and the upper bound is -1 which is up to last column (python includes lower bound but excludes upper bound hence last column will be excluded). At last we add ('dot' values) that means to take all the values described.

For y, to take all the rows of last column we add colon separating it with a comma we will not mention any lower bound and upper bound but only the index of the last column i.e., -1 and last adding ('dot' values) to include the values.

The output for the matrix created as x and y after importing the data.csv file from the folder to Google Colaboratory is under

```
[ ] print(x)

[['France' 44.0 72000.0]
 ['Spain' 27.0 48000.0]
 ['Germany' 30.0 54000.0]
 ['Spain' 38.0 61000.0]
 ['Germany' 40.0 nan]
 ['France' 35.0 58000.0]
 ['Spain' nan 52000.0]
 ['France' 48.0 79000.0]
 ['Germany' 50.0 83000.0]
 ['France' 37.0 67000.0]]

[ ] print(y)

['No' 'Yes' 'No' 'No' 'Yes' 'Yes' 'No' 'Yes' 'No' 'Yes']
```

Figure 4: Output Matrix created as x and y

3. Taking care of missing data

Presence of any missing data can cause errors while training the machinery model so generally we don't want to have any of it therefore it needs to be handled. Few ways to handle them are:

i. By deleting the rows of missing data that works fine if there is a large dataset and loss of 1% data will not create much effect in the overall analysis.

ii. The classic method of handling the missing data is by replacing the missing data by the average of the data present in that column.[3]

To implement the classic method we introduce a library called scikit learn that includes many of the data pre-processing tools. To handle the missing data using scikit learn

i. Simple Imputer class needs to be imported first.

ii. Create instance or object of the Simple Imputer class that would allow us to replace the missing data with the average of the data.

iii. Next we'll get an updated matrix of feature (as it's only applied to matrix of feature only).

▾ Taking care of missing data

```
[ ]  from sklearn.impute import SimpleImputer
     imputer = SimpleImputer(missing_values=np.nan, strategy='mean')
     imputer.fit(x[:,1:3])
     x[:,1:3]= imputer.transform(x[:,1:3])
```

Figure 5: Dealing with missing values of dataset

In the code above, importing Simple Imputer class from the scikit learn toolkit using ‘sklearn’ from one of its module ‘impute’ belonging to scikit learn. NeXT, we create the instance of the Simple Imputer class so introducing a new variable called ‘imputer’ that will store the average salary (according to above datasheet) . Calling the Simple Imputer class and argument will specify which missing value do we have to replace (missing_values) that will be equal to numpy library np.nan and the second argument will tell that the missing values are to be replaced by ‘mean’ of the values i.e. strategy = ‘mean’. Lastly, connecting the object to the matrix of features using fit method (imputer.fit ()) that will connect the input to matrix of features i.e., will look for missing values in the salary column and compute them with average of salaries. And the next step is to replace them using the transform function (imputer.transform ()) which will apply the transformation by replacing them. Applying fit and transform function to the columns with numerical values only.

So the updated feature with replacement is below,

```
[ ] print(x)

[['France' 44.0 72000.0]
 ['Spain' 27.0 48000.0]
 ['Germany' 30.0 54000.0]
 ['Spain' 38.0 61000.0]
 ['Germany' 40.0 63777.77777777778]
 ['France' 35.0 58000.0]
 ['Spain' 38.77777777777778 52000.0]
 ['France' 48.0 79000.0]
 ['Germany' 50.0 83000.0]
 ['France' 37.0 67000.0]]
```

Figure 6: Updated feature x with replacement

4. Encoding categorical data

	A	B	C	D
1	Country	Age	Salary	Purchased
2	France	44	72000	No
3	Spain	27	48000	Yes
4	Germany	30	54000	No
5	Spain	38	61000	No
6	Germany	40		Yes
7	France	35	58000	Yes
8	Spain		52000	No
9	France	48	79000	Yes
10	Germany	50	83000	No
11	France	37	67000	Yes

If the dataset has some kind of categorical data (like France, Germany and Spain) , it will be difficult for machine learning models to find any correlations among them and if naming them as 0 1 and 2 there are possibility for future machinery model to find correlations among them and which is not the case, so, in order to avoid misinterpretations we use OneHotEncoding.

OneHotEncoding consists of turning the country column into three columns as there are three different categories in country column. It consists of creating binary vectors for each country so that there is no numerical order between these countries as its in 0 and 1.

Not only the country column but also the purchase column having yes and no will be converted into binary vectors

.

- **Encoding the independent variable**

  We'll have two classes first with column transform from compose module from scikit learn and second is OneHotEncoder class of pre-processing module of scikitlearn library.

```
[ ] from sklearn.compose import ColumnTransformer
    from sklearn.preprocessing import OneHotEncoder
    ct = ColumnTransformer(transformers=[('encoder', OneHotEncoder(), [0])], remainder='passthrough')
    x = np.array(ct.fit_transform(x))
```

Figure 7: Applying OneHotEncoder and Column Transform on ct

Next creating an object of Column Transformer class 'ct' equals to instance of class with two arguments, one Transformers that specifies the type of transformation we require which is 'encoding' then what the type of encoding that is OneHotEncoding and on which indexes of columns we want to encode i.e. the country column with index 0. Second is the remainder that specifies if we want to keep the columns that won't be applied hence it's equal to 'passthrough' which means to keep the columns where no transformations are applied. Lastly connecting the 'ct' to the dataset and applying transform using fit_transform function on 'x' but since the fit_transform does not return as numpy array so we'll force call the numpy array as (np.array).

```
[ ]  print(x)
```

```
[[1.0 0.0 0.0 44.0 72000.0]
 [0.0 0.0 1.0 27.0 48000.0]
 [0.0 1.0 0.0 30.0 54000.0]
 [0.0 0.0 1.0 38.0 61000.0]
 [0.0 1.0 0.0 40.0 63777.77777777778]
 [1.0 0.0 0.0 35.0 58000.0]
 [0.0 0.0 1.0 38.77777777777778 52000.0]
 [1.0 0.0 0.0 48.0 79000.0]
 [0.0 1.0 0.0 50.0 83000.0]
 [1.0 0.0 0.0 37.0 67000.0]]
```

Figure 8: x after fit_transform function

- **Encoding the dependant variable**

```
[ ]  from sklearn.preprocessing import LabelEncoder
     le = LabelEncoder()
     y= le.fit_transform(y)
```

Figure 9: Encoding Dependent Variable

Here we encode all the no's and yes's in zero's and one's. Here, we will be importing label encoder class from pre-processing module of scikit learn library and then creating its object named as le and call the label class encoder and at last applying the fit_transform module on 'y', converting text to numerical values.

On printing we get :

```
[ ]  print(y)
```

```
[0 1 0 0 1 1 0 1 0 1]
```

Figure 10: After encoding no with 0 and yes with 1

5. Splitting data into training and test set

In this method we split our dataset into two parts :

i) Training set : In this we train our machinery model on existing observations

ii) Test set : In this we evaluate the performance of machinery on new observations

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 1)
```

Figure 11: Splitting Dataset

Using the model_selection module from scikit learn library which consists of trained test split which helps to create four separate sets, pair of matrix of features and dependent variable for training set and a pair of matrix of features and dependent variable for test set. Following four sets are:

X strain: for matrix of features of the training set

X test: matrix of features for test sets

y train : dependent variable of training set

y test : dependent variable for test set

Creating these above four variables returned by the train_test_split function and since it's the function that returns the variables assigning it to train_test_split function that takes parameters X as the matrix of features and y as dependent variable. The next two arguments are first the split size as we want a lot of training sets and a few test for future machinery to understand and learn the correlations of dataset.

Above, we take 80% of dataset as training set and 20% as test set (test_size) .

```
print(X_train)
```

```
[[ 9.5]
 [ 2. ]
 [ 8.7]
 [ 7.9]
 [ 8.2]
 [ 2.2]
 [ 1.5]
 [ 9. ]
 [ 3. ]
 [ 5.9]
 [ 4.1]
 [ 3.2]
 [ 9.6]
 [ 1.3]
 [ 5.1]
 [ 1.1]
 [ 4.9]
 [10.5]
 [10.3]
 [ 3.7]
 [ 3.2]
 [ 4. ]
 [ 4. ]
 [ 2.9]]
```

Figure 12: Matrix of feature for training set

```
[13] print(y_train)
```

```
[116969.  43525. 109431. 101302. 113812.  39891.  37731. 105582.  60150.
  81363.  57081.  54445. 112635.  46205.  66029.  39343.  67938. 121872.
 122391.  57189.  64445.  56957.  55794.  56642.]
```

```
[14] print(y_test)
```

```
[83088. 98273. 63218. 93940. 61111. 91738.]
```

```
[15] print(X_test)
```

```
[[5.3]
 [7.1]
 [3.9]
 [6. ]
 [4.5]
 [6.8]]
```

Figure 13: Dependent variable of training and test set

SIMPLE LINEAR REGRESSION

The purpose of this model is to state relation between dependent variable (Y) and two or more independent variables using the best fit line. It is a linear estimation. The equation to represent is [2]

$$Y= a+ bX$$

where, a is intercept and b is slope of the line.

On the basis of predictor variables, the above equation can be used to predict the value of target variable.

Obtaining best fit line [3]

Least square method is the most common method to find a regression line. For observed data, this method calculates best fit by minimizing the sum of squares of vertical deviation from each data point to line. Mathematically, it can be written as

$$\text{Min } \{\text{Sum } (y - y')^2 \}$$

Here, y' is data point to line, y is the vertical deviation.

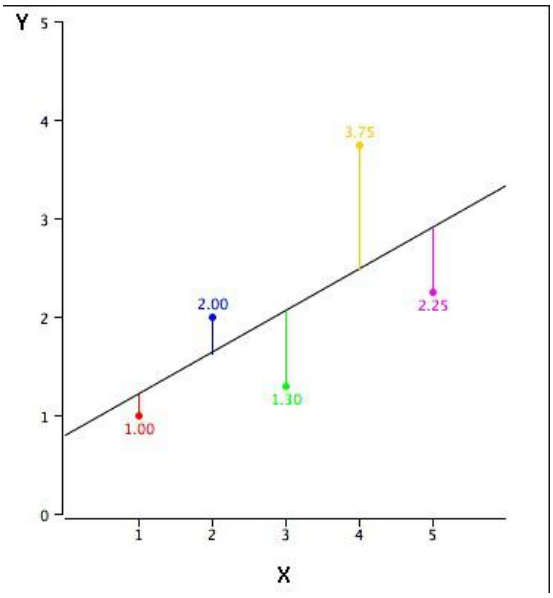

Figure 14: Graphical Representation of best fit line

The data used for this model is:

	A	B
1	YearsExperience	Salary
2	1.1	39343
3	1.3	46205
4	1.5	37731
5	2	43525
6	2.2	39891
7	2.9	56642
8	3	60150
9	3.2	54445
10	3.2	64445
11	3.7	57189
12	3.9	63218
13	4	55794
14	4	56957
15	4.1	57081
16	4.5	61111
17	4.9	67938
18	5.1	66029
19	5.3	83088
20	5.9	81363
21	6	93940
22	6.8	91738
23	7.1	98273
24	7.9	101302
25	8.2	113812
26	8.7	109431
27	9	105582
28	9.5	116969
29	9.6	112635
30	10.3	122391
31	10.5	121872

Figure 15: Dataset used for simple linear regression

The data model to be used consists of 31 observations and 2 columns and here Years Experience is a feature and Salary is dependent variable that is to be predicted.

1) Data Preprocessing

This step includes applying the data pre-processing tools

i) Importing the libraries

ii) Importing the dataset

iii) Splitting the dataset into Training set and Test set

Figure 16: Data Preprocessing

2) Training Simple Linear Regression model in training set

Using scikit learn library that allows to build simple linear regression model by accessing the module called linear model and from this model we call class called linear regression then we build an instance of this class.

```
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit (X_train,y_train)
```

Figure 17: Training model using Linear Regression

From the scikit learn library that contains a code name called sklearn and calling the linear model by adding a dot , we'll import Linear Regression and then creating a new variable 'regressor' as an instance of linear regression class and then call the Linear Regression class.[5]

Next is to connect to the training set and the function used for connecting is known as Fit function. To call the function firstly take the object 'regressor' and the method fit() with dot in between. Fit() method will train the regression model on the training set with arguments with matrix of features of X_train and second the dependent variable Y_train.

On running the cell, linear Regression model is created with default values.

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

Figure 18: Fit () method to train regression model on training set

3) Predicting the test set results

Earlier we have split the dataset into two parts training set and test set. The test set was choosen to be 20% of total observations (i.e. = last 6 observations) and now we need to predict the observations which means the salary for each of those employee.

```
y_pred = regressor.predict(X_test)
```

Figure 19: Result Prediction

Firstly, we call the object regressor and from that they predict method by adding a dot in between. The arguments for the predict function will be the features i.e. the number of years of experience that are contained in X_test. The next step is to visualize these test set, therefore, putting all these predictions of salaries in a variable naming as y_pred.

4) Visualisation of Training Set Result

Using the pyplot function from matplotlib library (with shortcut plt) to call the 'scatter' function that allows us to plot red point representing the real salaries and for coordinates of these points are given by X_train and y_train. X_train contains years of experience and y_train for salaries.

```
plt.scatter (X_train, y_train, color= 'red')
plt.plot (X_train, regressor.predict(X_train), color = 'blue')
plt.title ('salary vs experience (Training set')
plt.xlabel ('Years of expierence')
plt.ylabel ('salary')
plt.show()
```

Figure 20: Visualizing Training Set Result

Next is to plot regression lines (lines of predictions coming as close as possible to real results) and therefore points corresponding to predicted result will follow straight line and we'll use plot method to plot the curve of the function. The arguments for plot function will

- Contain coordinates of predicted salaries where x coordinate corresponds to X_train and y corresponds to vectors that will predict salaries of training set (regressor.predict (X_train), calling X_train means number of years' experience of employees will get exactly the predicted salaries of training set).
- Colour of the point.

Then, for building charts we use title function to give title to our graph and then specifying the training set further adding labels to x and y axis using xlabel and ylabel functions respectively and to finally show graphic we use plt. Show function. [6]

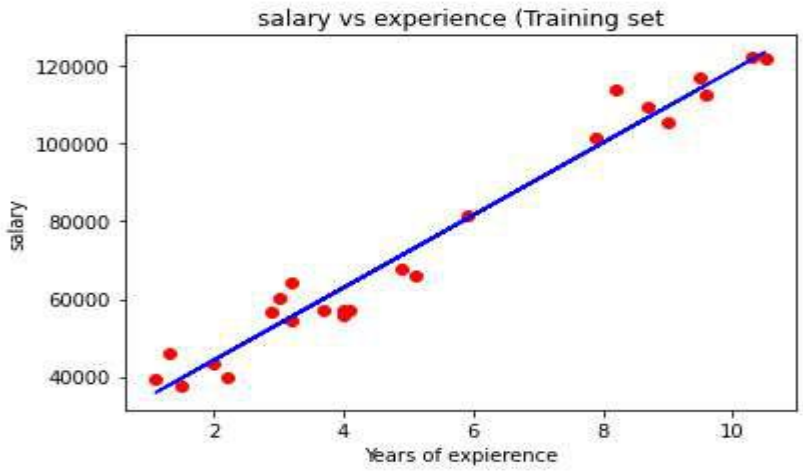

Figure 21: Graphical visualization of prediction

The above plot shows real salaries with red dots and blue regression line predicting salaries. Regression line has been calculated such that it comes as close as possible to the real salaries and for each year it has been predicted here by projecting years of experience to regression line. For eg predicted salary for eight year experience is about $100000 per year.

5) Visualisation of Test Set Result

As per visualisation of training set result, for test set result the few changes:

i) changing the coordinates of employee of test set hence replacing X_train by X_test and y_train by y_test for real observations.

ii) changing the title for plot from training to test set.

```
plt.scatter (X_test, y_test, color= 'red')
plt.plot (X_train, regressor.predict(X_train), color = 'blue')
plt.title ('salary vs experience (Test set)')
plt.xlabel ('Years of expierence')
plt.ylabel ('salary')
plt.show()
```

Figure 22: Visualization of test set result

The visualisation of test set

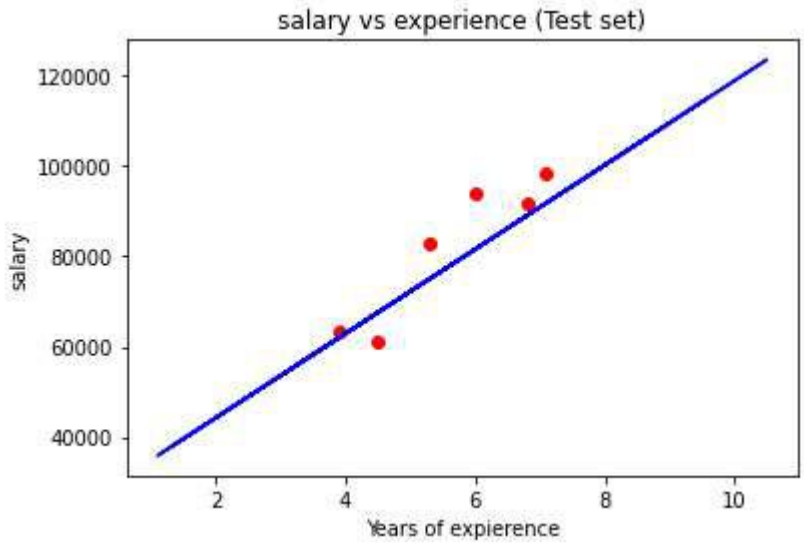

Figure 23: Graphical Visualization of test set

The predicted salaries for the test set are indeed close to the real salaries for which the linear regression model was able to efficiently predict the new observations.

CONCLUSION

The analysis of the data obtained in present study from data Preprocessing tools readies the data for further interpretations as its the basic and most initial step by dividing dataset into two categories of independent and dependent variables. In simple linear regression as from the datasheet used for analysis we can conclude that the predicted salaries for the test set are indeed close to the real salaries for which the linear regression model was able to efficiently predict the new observations.

FUTURE SCOPE OF DATA ANALYTICS

Augmented analytics is going to be the future of data analytics because it can scrub raw data for valuable parts for analysis, automating certain parts of the process and making the data preparation process easier. At the moment, data scientists spend around 80% of their time cleaning and preparing data for analysis.[6]

Data analytics initially "supported" the decision-making process, but is now enabling "better" decisions than we can make on our own. For example to combine sales, location and weather data to understand sales increase for certain stores and improve the replenishment process.

If it turns out in the future that a decision-making process based on data analytics will produce better results, the step to "automated" decision-making will be small.

REFERENCES

[1] Michael Berthold, David J. Hands (Eds.): "Intelligent Data Analysis".

[2] Satyanarayana Labani,Komal Wadhwa, Smita Asthana : "Basic Approach to Data Analysis and Writing of Results and Discussion Sections".

[3] Avinash Pandey, Garima Srivastava, Anuradha Sharma, 2018, "A Hybrid Approach To Data Mining Algorithms For Classification", International Journal of Emerging Technologies and Innovative Research(JETIR), vol.5

[4] Aditya Mishra, Christian L. Muller : "Computational Statistics & Data Analysis"

[5] Odd O. Aalen: "A linear regression model for the analysis of life times".

[6] R. R. Hocking: "Developments in Linear Regression Methodology: 1959-1982".

[7] John W. Tukey : "The Future of Data Analysis"