Regression

Own Algorithm

by

G Henry Jacob

Contents

1. Author's Note 1
2. Introduction 2
3. Percentage 3
4. Algorithm Note 4
5. Algorithm 5
    5.1 Fit 5
        5.1.1 Walk_data 5
        5.1.2 Space calculation 6
        5.1.3 Space note 6
        5.1.4 Example calculation of space 7
        5.1.5 X_space(linspace) 8
        5.1.6 To create dataframe 8
        5.1.7 No data missed 11
        5.1.8 Percentage calculation for first linspace 12
        5.1.9 Mean calculation 13
        5.1.10 No data 'nan' 13
        5.1.11 Overall percentage 14
        5.1.12 Replace 'nan' 16
        5.1.13 Training return 16
    5.2 Predict 17
        5.2.1 Inside linspace 17
        5.2.2 Outside linspace 18

5.3 Output 19
5.3.1 Numerically 19
5.3.2 Visually 20
5.4 Limitations 21
5.5 Pros 21
5.6 Cons 21
6. Glossary 22
6.1 Words and meanings 22
6.2 Symbols 25

Author's Note

The regression model which explain detailed in the pdf from my own knowledge and ideas. I built and performed this regression model with various dataset and almost got good root mean squared error as Linear regression. The main concept behind this model is percentage only. In this pdf I clearly explain the mathematics behind my model numerically and visually. Here I explain some concepts from scratch and I need your patience. I request everyone who read this to kindly go through each step and comment your opinion (positive or negative) on my linkedin post. It will be useful for me to update my knowledge.

-Henry.

Introduction:

As we all know regression is a set of statistical process for estimating the relationship between a dependent variable and one or more independent variable.

There is lot of regressions (linear regression, ridge regression, lasso regression…) in Supervised Machine learning algorithms.

Here, I created a regression algorithm which predict output based on percentage of independent (X) to dependent (y).

Percentage:

As I mentioned above, we can calculate the percentage of one(A) to other(B) is,

$$A * (x/100) = B.$$

Let assume a = 5 and b = 8,

The percentage will be calculated as follows:

$$5 * (x/100) = 8$$

$$x = (8*100)/5$$

$$x = 800/5$$

$$x = 160\%$$

Here b varies from a by 160%.

We can recheck using the same as follows,

$$b = 5 * (160/100)$$

$$b = 5 * (1.6)$$

$$b = 8$$

This is the basic concept used in this algorithm.

Let's get started with algorithm…

Algorithm Note:

1.You can select the independent and dependent feature, split them using train_test_split and pass in to the fit function.

Example

If ‘df’ is the dataframe contains independent feature ‘a’ and dependent feature ‘b’ then,

```
X = df[‘a’]
y = df[‘b’]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = .2, random_state = 0)

fit(X_train, y_train)
```

2.You must pass pandas series which drawn from pandas dataframe (inside fit function I created a dataframe using X (pandas series) and y (pandas series) which must have name), otherwise it will throw an error to you.

Algorithm

fit:

It is a measure of how well a machine learning model generalizes to similar data to that on which it was trained.

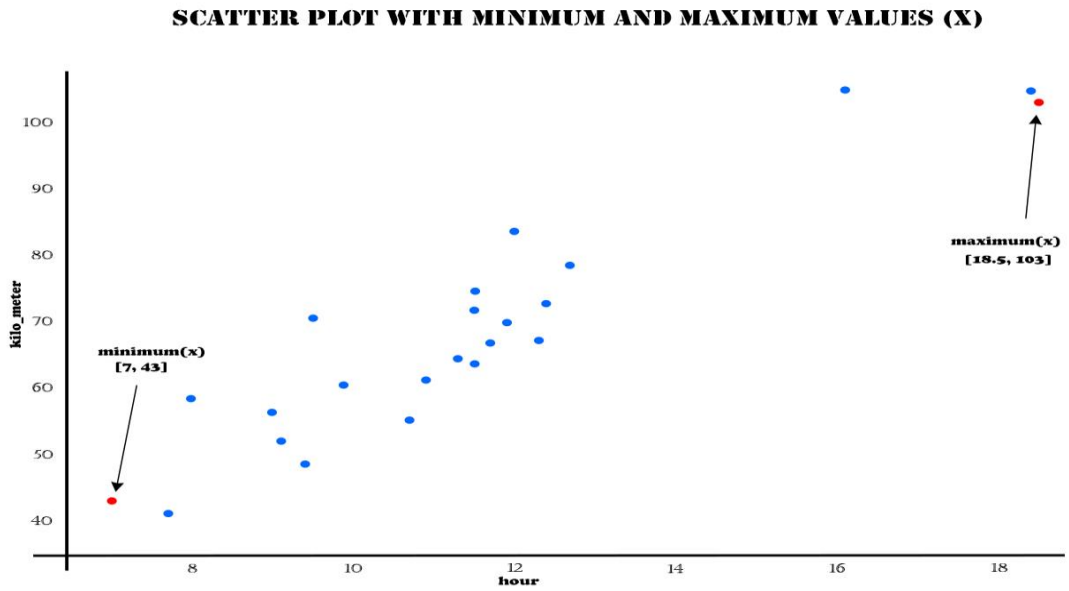
Walk_data:

Consider a data, kilo meters covered by human

(Generated data) – “walk_data” which will be used and shown for all upcoming images.

hour	16.1	11.7	10.9	9	9.1	18.5	9.4	7	10.7	12.4	8	11.5	7.7	11.9	12.7	9.875	12	12.3	11.5	11.5	11.3	18.4	9.5
kilo_meter	105	66.81	61.24	56.39	51.96	103	48.53	43	55.1	72.8	58.39	71.67	41.11	69.95	78.52	60.39	83.52	67.23	63.67	74.67	64.52	104.8	70.53

At first it creates a dataframe inside the function with the column name of pandas series name. Then it finds and store the minimum and maximum values of X.



Space calculation:

Then it calculates the number of spaces for the data based on total number of training data, using the formula.

Space = (len(x)//5) + 1

space – is used to create number of spaces.

Len(x) – total number of data present in X_train.

Space note:

I ran the program many times with range of values over various dataset and got good result with ‘5’,

That's why I used 5 as default parameter for floor division here.

Example calculation of space:

If total number of data in X_train is 47 then space is calculated as follows,

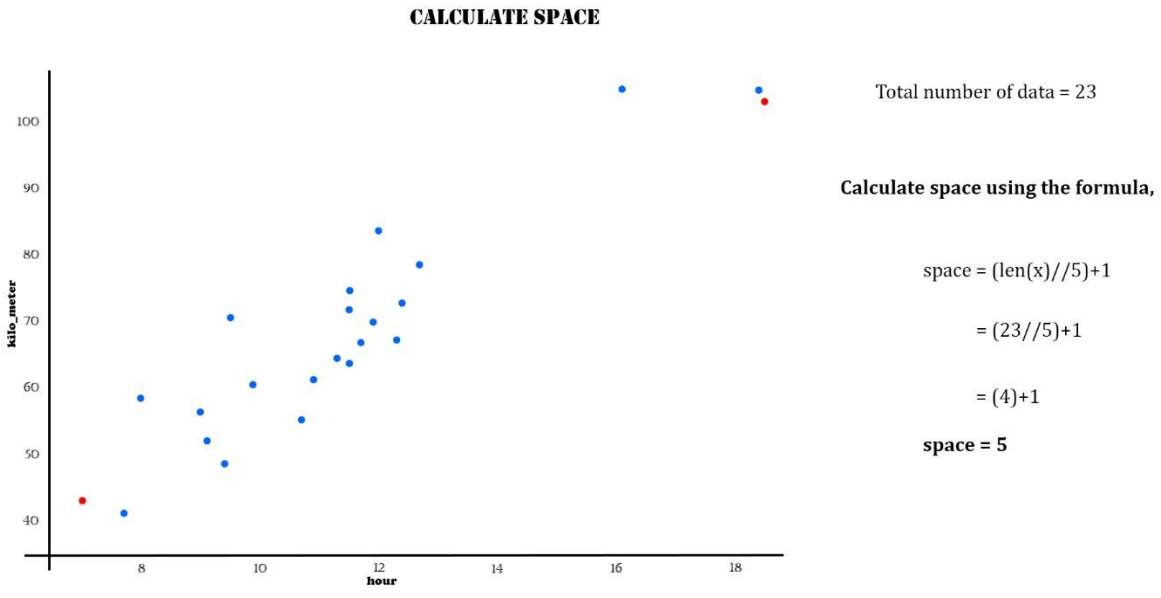
$$\text{space} = (\text{len}(x)//5) + 1$$

$$= (47//5) + 1$$

$$= 9 + 1$$

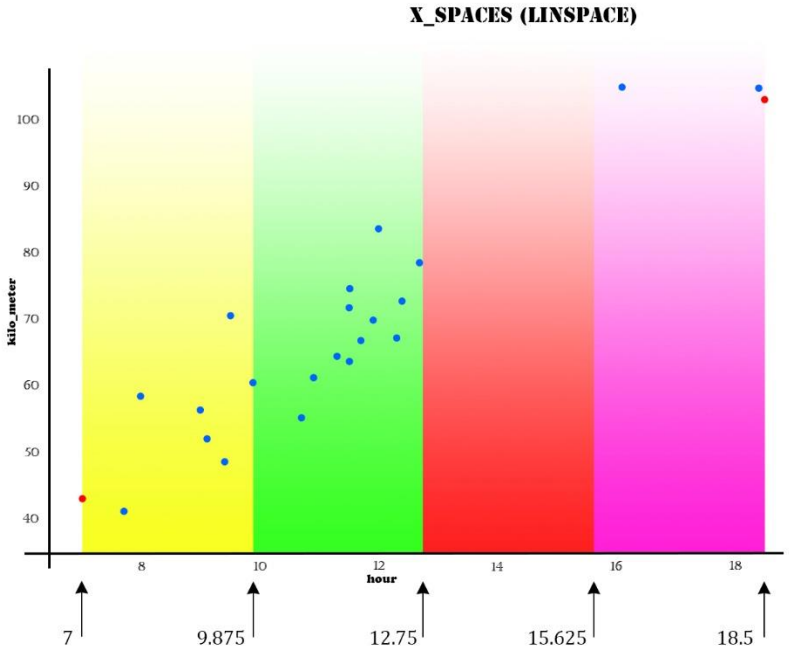
$$\text{space} = 10$$

Here is the calculation for walk_data:



X_spaces(linspace) :

Then it creates a list of values (x_spaces) from minimum and maximum with total space separated equally using package ‘Numpy’.



x_spaces

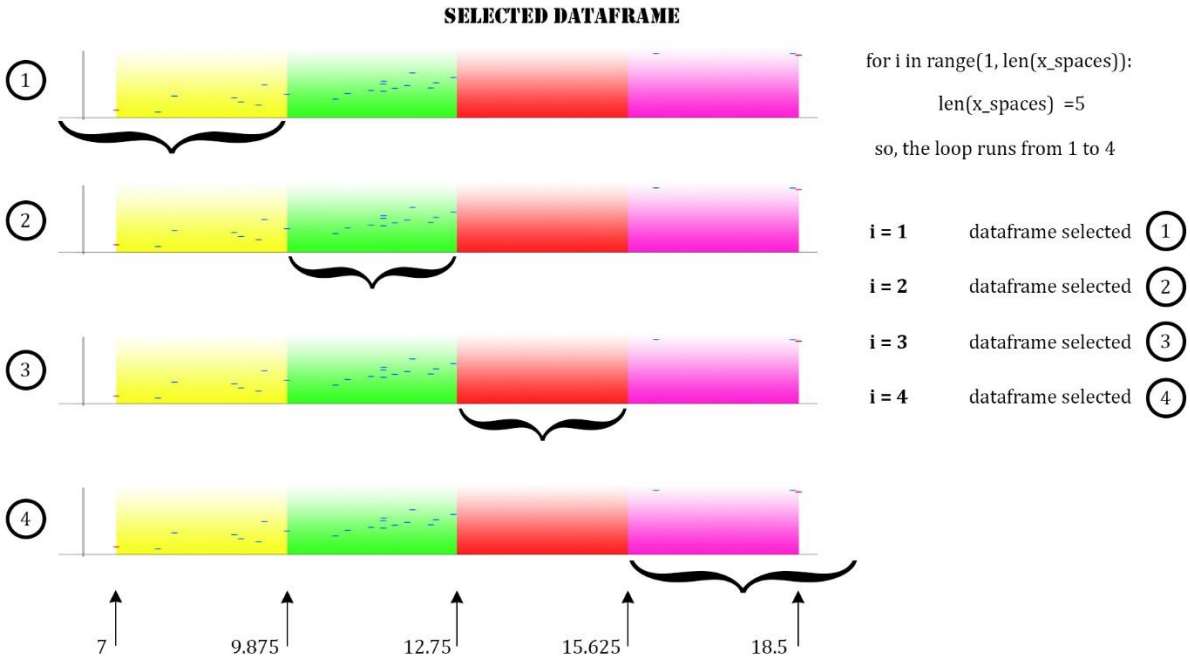
[7. , 9.875, 12.75 , 15.625, 18.5]

figure shows spaces and data point fall under which one

B2

To create dataframe:

It runs a for loop to get exact data which lie between the above x_spaces

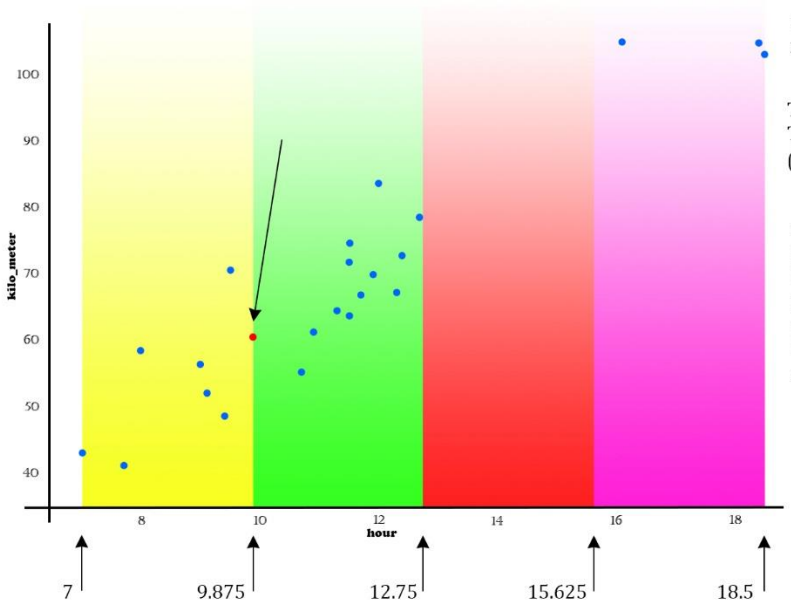


B3

.0000000001

If a value of X_train exactly equals to any x_space data, then there is possible that the data may exclude(missed) from taken dataframe. To avoid such thing, I am adding a least value (.0000000001).

.0000000001



Here the data point exactly lie on the x_spaces (linspace) x value = 9.875

This data must missed in selected dataframe To avoid this missing the least value added (.0000000001)

minimum value (x = 7) and maximum value (x = 18.5) is also lied in the linspace line, but they are included in the first dataframe (taking values less than second linspace) as well as last dataframe(taking value greater than second last linspace)

To get the data which exactly lied in linspace we are adding the least value.

0.0000000001

```
[93]: a = 1000000
      b = .0000000001
      print(a+b)

      1000000.0000000001
```

here python considered 0.0000000001 (upto ten values after decimal point) for adding another value (b) upto 1000000

```
[94]: a = 753
      b = .0000000001
      print(a+b)

      753.0000000001
```

this is another example shows the same

```
[95]: a = 10000000
      b = .00000000001
      print(a+b)

      10000000.0
```

it does not consider when i add one more digit(11th digit) after decimal.

```
[104]: a = 1000000
       b = .00000000001
       print(a+b)

       1000000.0
```

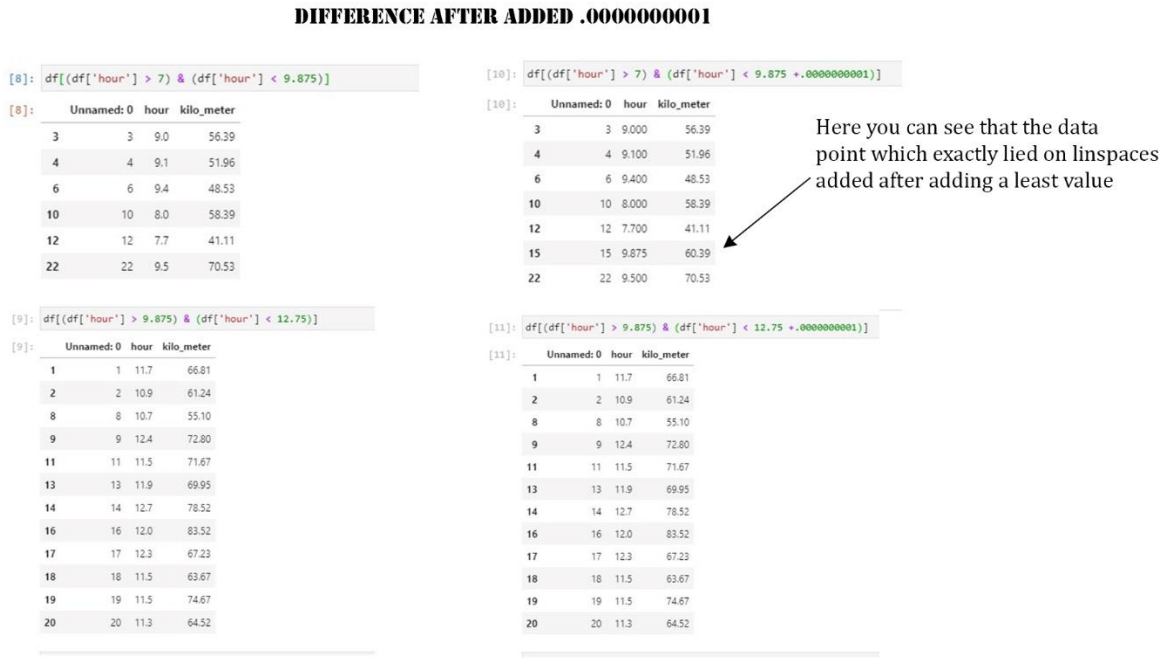
It does not consider even after i removed a digit from b. To make it considerable i have to remove one more digit from b

```
[105]: a = 100000
       b = .00000000001
       print(a+b)

       100000.00000000001
```

On my point of view here i take 0.0000000001 as least value where it can be considerable upto 1000000 while adding

No data missed:

Here you can see how the data missed and how we can get after added a least value



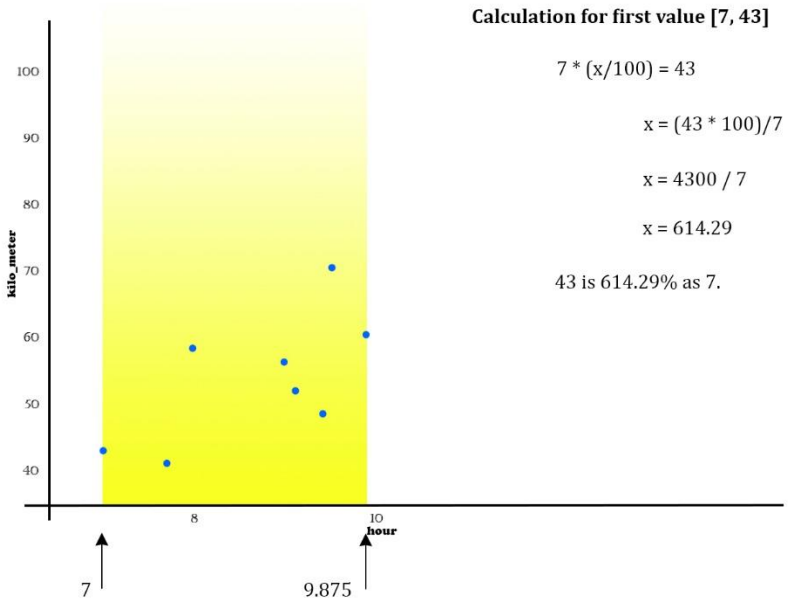
That's how the datapoint which exactly lied on the x_spaces (linspaces) added to the previous dataframe in the loop.

Once the particular dataframe taken it runs a for loop through each index and calculate how much percentage y varies from X.

B4

Percentage calculation for first linspace:

At first let us take our first linspace and the datapoints lied in it



PERCENTAGE CALCULATION

Calculation for first value [7, 43]

$7 * (x/100) = 43$

$x = (43 * 100)/7$

$x = 4300 / 7$

$x = 614.29$

43 is 614.29% as 7.

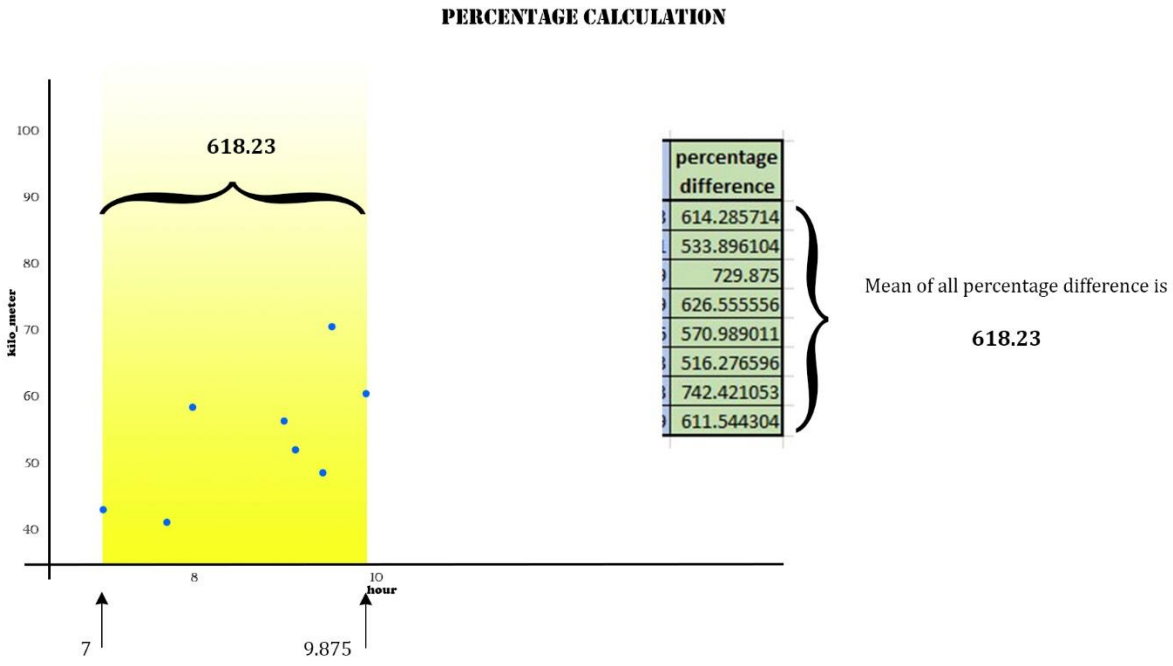
Rest of the calculation for first linspace

Data points falls under first linspace and percentage difference

hour	km	percentage difference
7	43	614.285714
7.7	41.11	533.896104
8	58.39	729.875
9	56.39	626.555556
9.1	51.96	570.989011
9.4	48.53	516.276596
9.5	70.53	742.421053
9.875	60.39	611.544304

Once the percentage calculated for datapoints lied on a x_spaces it calculates the mean for it.

Mean calculation:



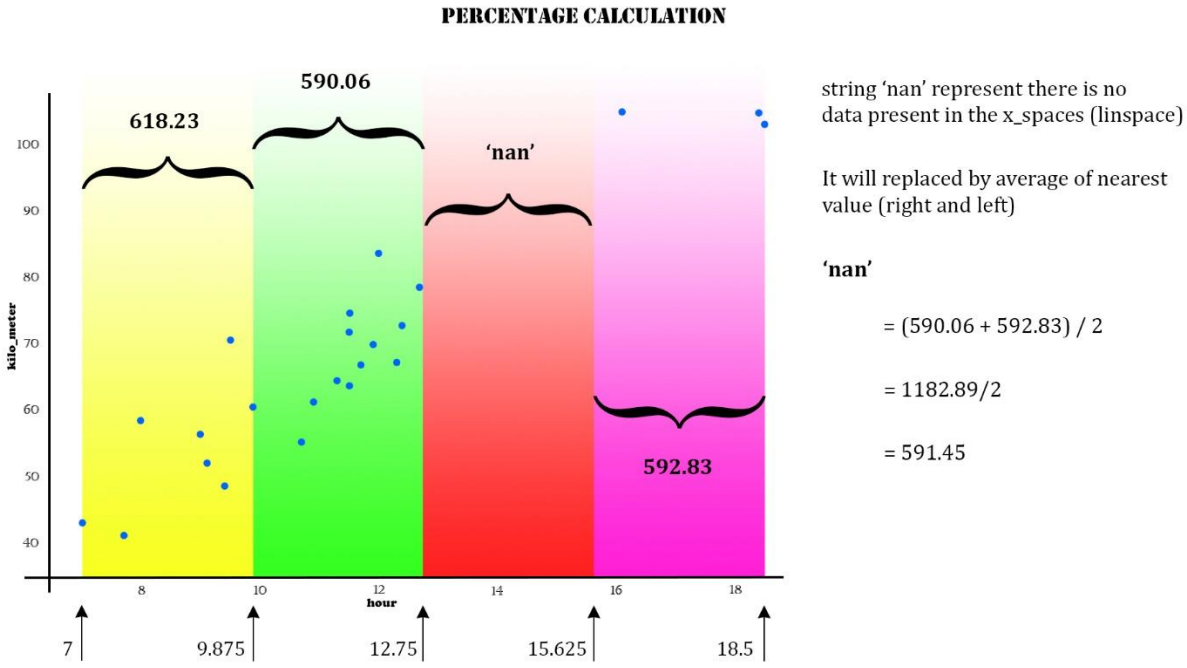
Once the loop completed, it checks and append the mean value of percentage change, mean value of X and mean value of y only if data present in the taken dataframe.

B5

No data – ‘nan’

If there is no data present in the taken dataframe then it appends a string ‘nan’.

That’s how the entire loop completed.

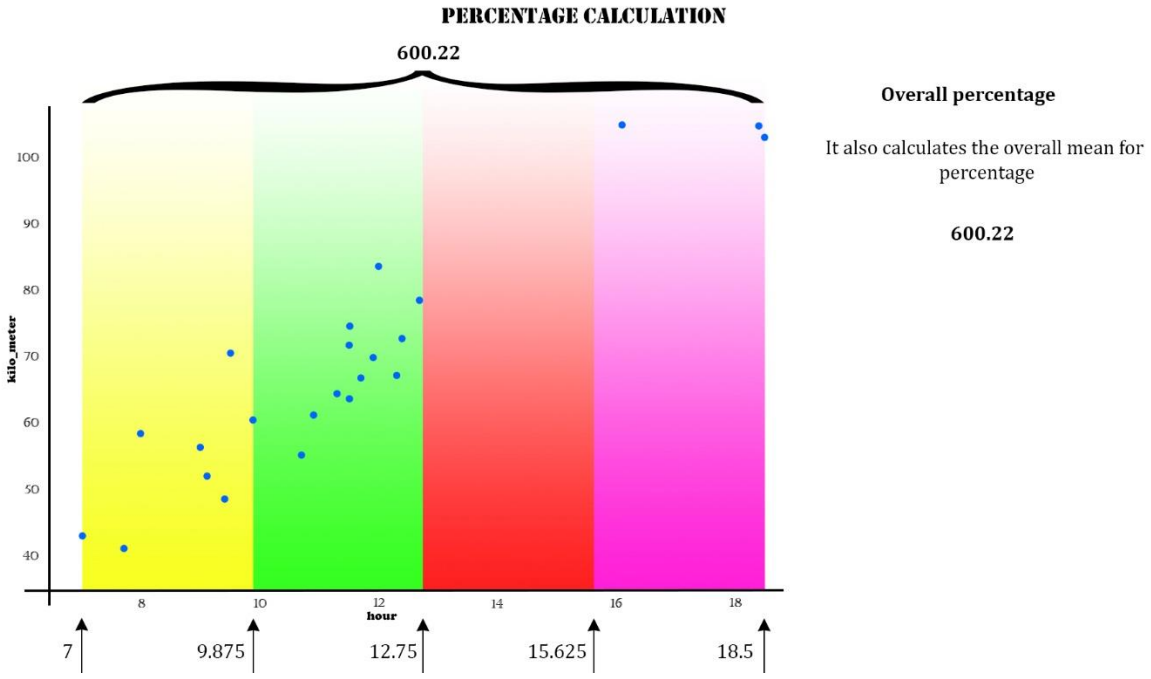


Here we can see 'nan' represent there is no data to calculate percentage for that particular x_spaces and the calculation to replace 'nan' with value.

 # B1

Overall percentage:

For each data in X it also appends how much percentage y varies from X to a new list (overall_percentage)



Once the loop completed and got all

1.mean value of percentage change for each linspace,

2.mean value of X for each linspace,

3.mean value of y for each linspace,

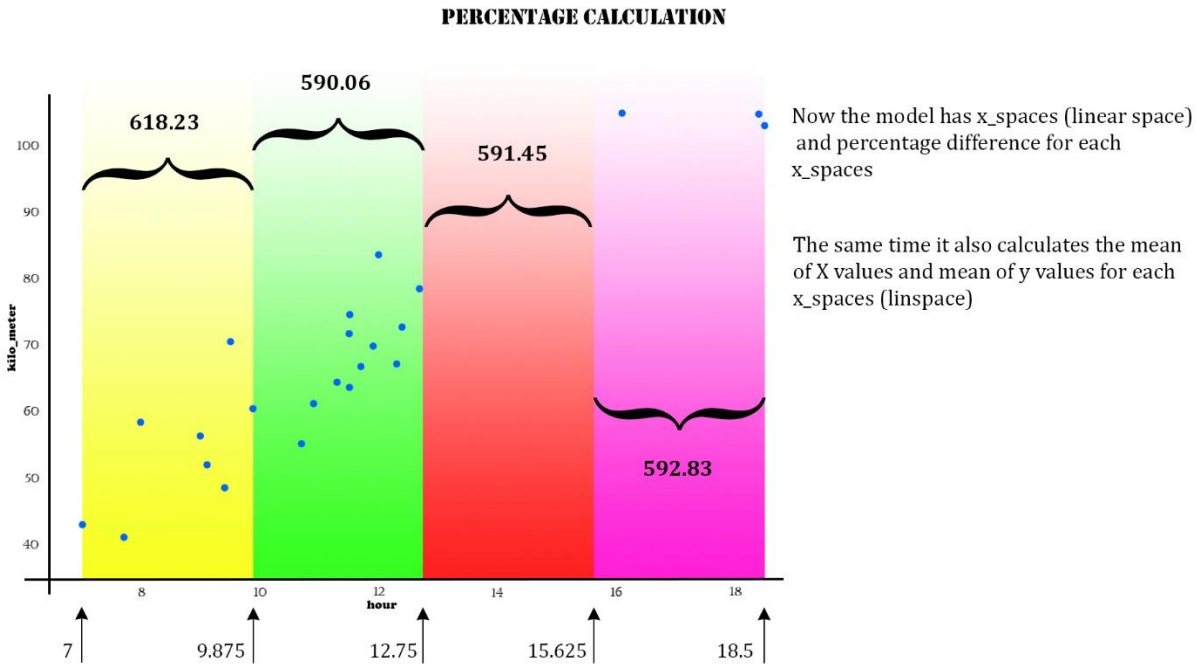
4.list of overall_percentage.

Then it runs a for loop to get the start = starting index of ‘nan’ and end = ending index of ‘nan’ +1.

B7

Replace ‘nan’:

It again runs a for loop and replace ‘nan’ with related value using start and end.



After replace ‘nan’ the value of x_spaces will be as above.

Training return:

Once the loop completed it return 5 values.

1.X mean values (list)

2.y mean values (list)

3.percentage values (list)

4.x_spaces (list)

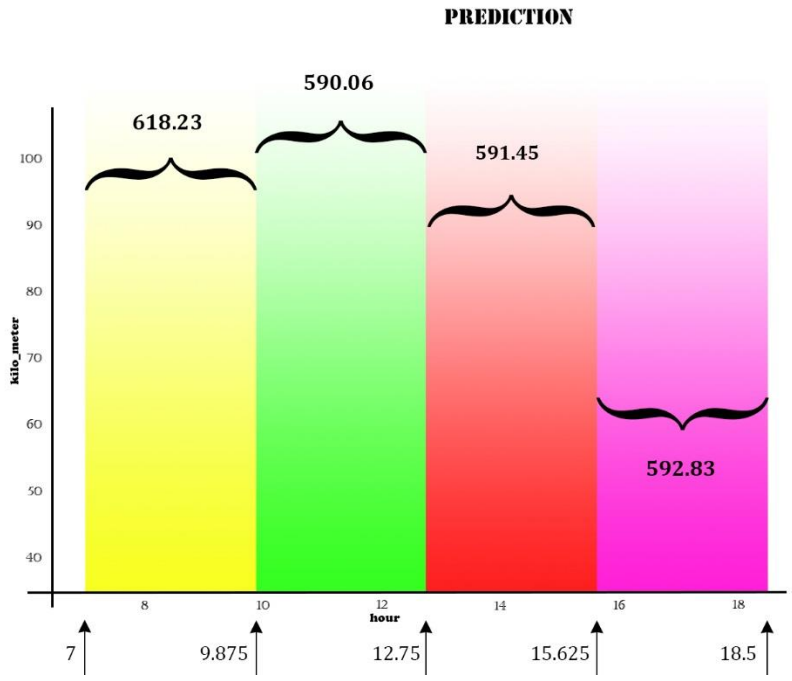
5.mean of overall percentage (float)

Predict:

 # B10

Inside linspace:

For each data of X_test, the model check what is the exact x_space the data lies and predict the y value with the exact percentage change.



If test data for the model is **8,**
It lies on first linspace
percentage differnce for first linspace is 618.23 so the calculation as follows,

= (8 * 618.23)/100

= (4945.84)/100

= 49.46

If test data for the model is **14,**
It lies on third linspace
percentage differnce for third linspace is 591.45 so the calculation as follows,

= (14 * 591.45)/100

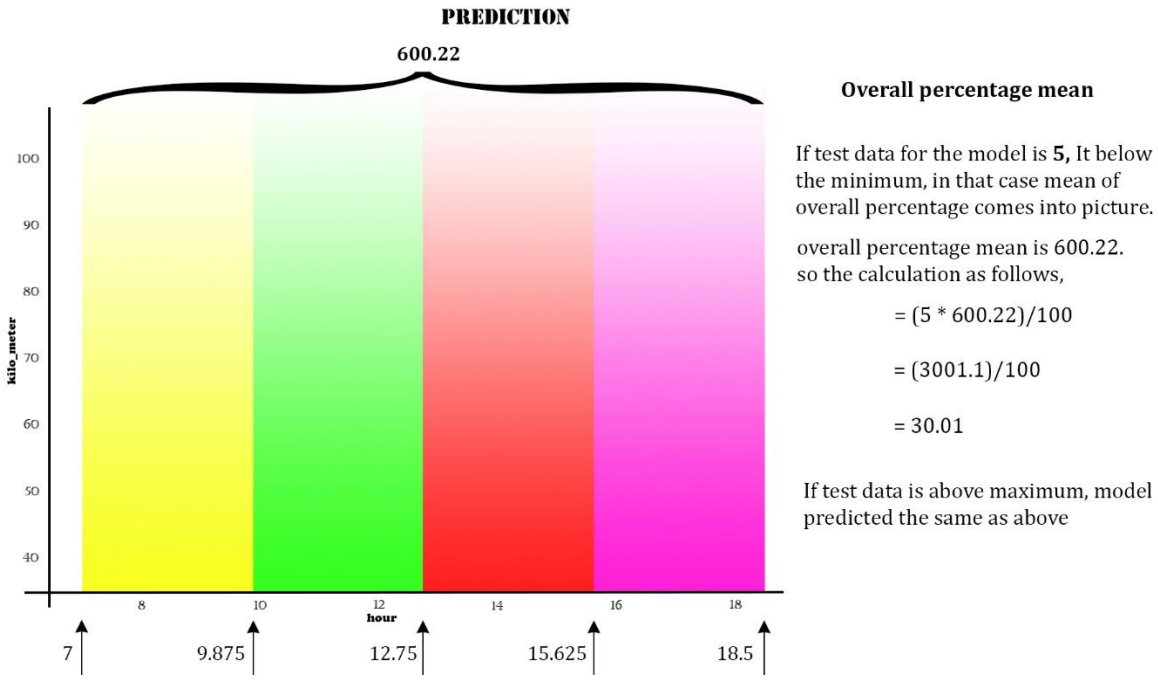
= (8280.3)/100

= 82.8

B8, B9

Outside linspace:

If the X_test does not lie in any linspace i.e. The value of X_test < minimum of X_train value or X_test > maximum of X_train value it takes overall percentage to predict y.



B11

Output:

Numerically:

Here is the difference between each data and it's predicted value.

PREDICTION OUTPUT

```
[15]: X_test

[15]: 11    11.5
      10     8.0
      21    18.4
      14    12.7
      20    11.3
      Name: hour, dtype: float64
```

```
[16]: y_test

[16]: 11     71.67
      10     58.39
      21    104.80
      14     78.52
      20     64.52
      Name: kilo_meter, dtype: float64
```

Linear regression

```
[40]: pred

[40]: array([ 68.33084143,  47.62316039, 109.15455548,  75.43061779,
              67.14754537])
```

```
[36]: np.sqrt(mean_squared_error(y_test, pred))

[36]: 5.700618311991743
```

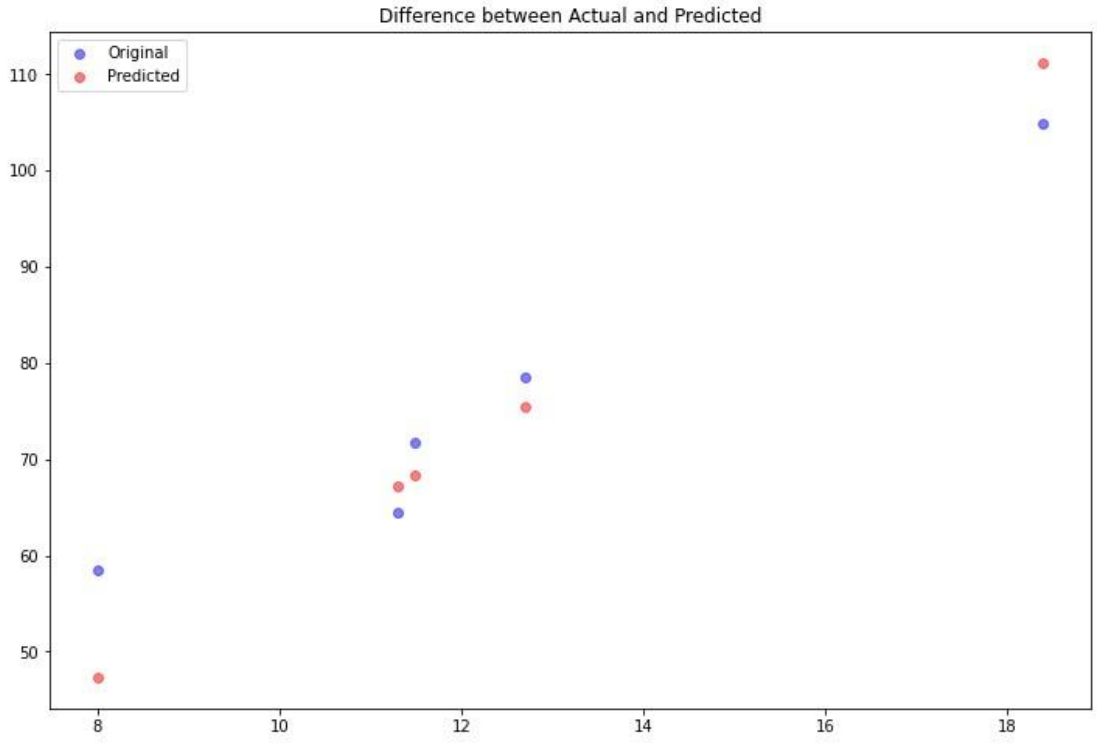
Own algorithm

```
[37]: predict_output

[37]: [68.32889293668597,
       47.309216079281505,
       111.22162162162162,
       75.45886437355755,
       67.14056436387405]
```

```
[10]: # Print statement to print root mean squared error.
      print(np.sqrt(mean_squared_error(y_test, predict_output)))

      6.187437302569467
```

Visually:

Here I show visually how far my model prediction differ from Actual (original).



Limitations:

1.There should be only one independent feature

Pros

Gives good result as linear regression.

Cons

It may predict wrong for linspace when few training data lied on that particular linspace which does not follow the correlation.

Glossary:

Words and meanings:

append – adds a single item to the existing list

dataframe – 2dimensional labeled data structure with columns of potentially different types.

Elif – short for else if

else – where “if” statement fails “else” statement executes

except – used to test code for an error which is written in the “try” statement.

Figure -contains all the plot elements.

Fit – method takes the training data as arguments

float – method converts a number stored in a string or integer into a floating point number, or a number with a decimal point

for – loop is used for iterating over a sequence

if – evaluates whether an expression is true or false

index – index position at which an item is found in a list or a string.

Legend – an area describing the elements of the graph

len – returns the length of a list, string, dictionary, or any other iterable data format

linspace – used to create sequences of evenly spaced numbers structured as a NumPy array

list – an ordered and mutable Python container

loc – used to specify the name of the rows and columns that we need to filter out

matplotlib – a multi-platform data visualization library

max – returns the largest item in an iterable

mean – the sum of data divided by the number of data points

mean_squared_error – average of the square of the difference between the observed and predicted values of a variable

min – returns the smallest item in an iterable

numpy – fundamental package for scientific computing in python.

or – True if either of the operands is true

pandas – Python library for data analysis

percentage – number or ratio that can be expressed as a fraction of 100.

Plot – used to draw points (markers) in a diagram

predict – the output of an algorithm after it has been trained on a historical dataset and applied to new data

range – returns a sequence of numbers starting from zero and increment by 1 by default and stops before the given number.

Return – used to end the execution of the function call and “returns” the result

scatter – used to observe relationship between variables and uses dots to represent the relationship between them

show – used to display all figures.

Sklearn – library for machine learning

sqrt – used to determine the positive square-root of an array

title – used to specify title of the visualization depicted

train_test_split – Split your data into training and testing

try – used to test code for an error which is written

x_test – independent test data

x_train – independent train data

y_test – dependent test data

y_train – dependent train data

Symbols:

‘&’ - and

‘*’ - multiplication

‘+’ - addition

‘-’ - subtraction

‘/’ - division

‘//’ - floor division

‘<’ - less than

‘=’ - equal to

‘= =’ - equality operator

‘>’ - greater than