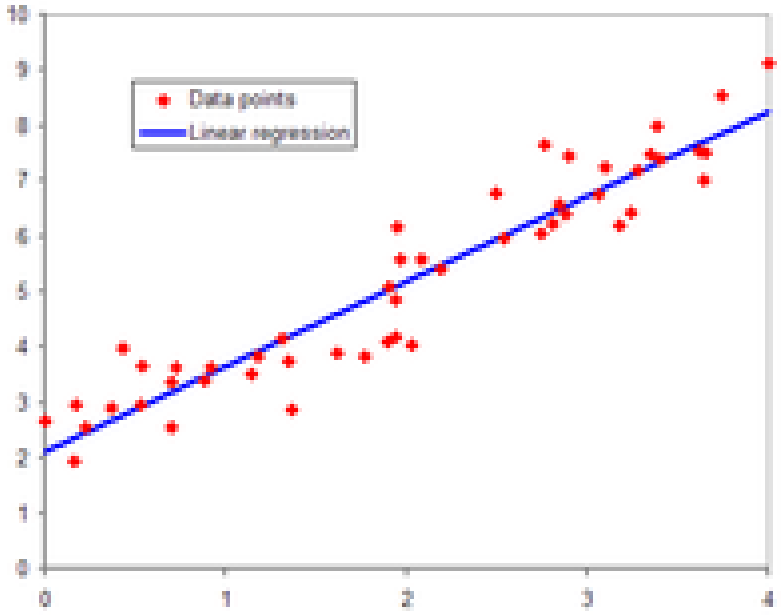
Regression

A regression is a statistical technique that relates a dependent variable to one or more independent (explanatory) variables. A regression model is able to show whether changes observed in the dependent variable are associated with changes in one or more of the explanatory variables.



Regression line for 50 random points in a Gaussian distribution around the line y=1.5x+2 (not shown)

Regression models (both linear and non-linear) are used for predicting a real value, like salary for example. If your independent variable is time, then you are forecasting future values, otherwise your model is predicting present but unknown values. Regression technique vary from Linear Regression to SVR and Random Forests Regression.

$$\hat{y} = b_0 + b_1 X_1$$

Dependent variable

y-intercept (constant)

Slope coefficient

Independent variable

In this part, you will understand and learn how to implement the following Machine Learning Regression models:

1. Simple Linear Regression
2. Multiple Linear Regression
3. Polynomial Regression
4. Support Vector for Regression (SVR)
5. Decision Tree Regression
6. Random Forest Regression

Simple Linear Regression

Simple linear regression is used to estimate the relationship between **two quantitative variables**. You can use simple linear regression when you want to know:

1. How strong the relationship is between two variables (e.g., the relationship between rainfall and soil erosion).
2. The value of the dependent variable at a certain value of the independent variable (e.g., the amount of soil erosion at a certain level of rainfall).

Dataset: Salary_Data.csv (csv file link)

	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

Simple linear regression:

Importing the libraries

```
importnumpyas np
importmatplotlib.pyplotasplt
import pandas as pd
```

Importing the dataset

```
dataset = pd.read_csv('Salary_Data.csv')
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values
```

Splitting the dataset into the Training set and Test set

```
fromsklearn.model_selectionimporttrain_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 1/3, random_state = 0)
```

Training the Simple Linear Regression model on the Training set

```
fromsklearn.linear_modelimportLinearRegression
regressor = LinearRegression()
regressor.fit(X_train, y_train)
```

Predicting the Test set results

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

Visualising the Training set results

```
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 Experience')
plt.ylabel('Salary')
plt.show()
```

Visualising the Test set results

```
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 Experience')
plt.ylabel('Salary')
plt.show()
```

Multiple Linear Regression

Multiple linear regression (MLR), also known simply as multiple regression, is a statistical technique that uses several explanatory variables to predict the outcome of a response variable. The goal of multiple linear regression is to model the linear relationship between the explanatory (independent) variables and response (dependent) variables. In essence, multiple regression is the extension of ordinary least-squares (OLS) regression because it involves more than one explanatory variable.

Formula and Calculation of Multiple Linear Regression

$$y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + ... + \beta_p x_{ip} + \epsilon$$

where, for $i = n$ observations:

y_i = dependent variable

x_i = explanatory variables

β_0 = y-intercept (constant term)

β_p = slope coefficients for each explanatory variable

ϵ = the model's error term (also known as the residuals)

What is a Dummy Variable Trap?

In linear regression models, to create a model that can infer relationship between features (having categorical data) and the outcome, we use the dummy variable technique.

Multicollinearity is a phenomenon in which two or more variables are highly correlated. In simple words, it means value of one variable can be predicted from the values of other variable(s).

						Dummy Variable State	
Profit	R&D Spend	Administration	Marketing Spe	State		New York	California
192261.83	165349.2	136897.8	471784.1	New York		1	0
191792.06	162597.7	151377.59	443898.53	California		0	1
191050.39	153441.51	101145.55	407934.54	California		0	1
182901.99	144372.41	118671.85	383199.62	New York		1	0
166187.94	142107.34	91391.77	366168.42	California		0	1
156991.12	131876.9	99814.71	362861.36	New York		1	0
156122.51	134615.46	147198.87	127716.82	California		0	1
155752.6	130298.13	145530.06	323876.68	New York		1	0

$y = b0 + b1*x1 + b2*x2 + b3*x3 + b4*D1 + \sout{b5*D2}$

P-Value:

In statistics, the p-value is the probability of obtaining results at least as extreme as the observed results of a statistical hypothesis test, assuming that the null hypothesis is correct. The p-value serves as an alternative to rejection points to provide the smallest level of significance at which the null hypothesis would be rejected. A smaller p-value means that there is stronger evidence in favor of the alternative hypothesis.

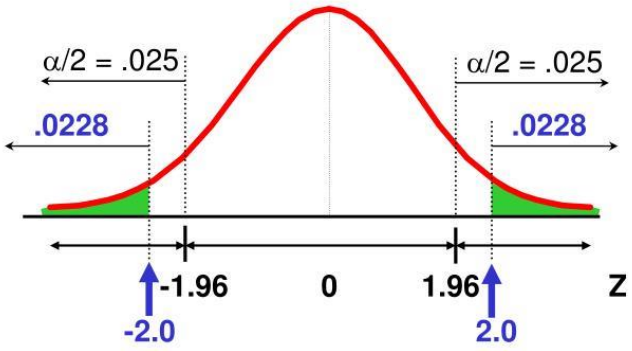
p-Value Example

(continued)

- Compare the p-value with α
  - If p-value $<$ α , reject H_0
  - If p-value $\geq$ α , do not reject H_0

Here: p-value = .0456
α = .05

Since .0456 < .05, we reject the null hypothesis



Building A Model:

5 methods of building models:

- All-in
- Backward Elimination
- Forward Selection
- Bidirectional Elimination
- Score Composition

Multiple Linear Regression

Importing the libraries

```
importnumpyas np
importmatplotlib.pyplotasplt
import pandas as pd
```

Importing the dataset

```
dataset = pd.read_csv('50_Startups.csv')
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values

print(X)
```

```
[[165349.2 136897.8 471784.1 'New York']
 [162597.7 151377.59 443898.53 'California']
 [153441.51 101145.55 407934.54 'Florida']
 [144372.41 118671.85 383199.62 'New York']
 [142107.34 91391.77 366168.42 'Florida']
 [131876.9 99814.71 362861.36 'New York']
…….
```

Encoding categorical data

```
fromsklearn.composeimportColumnTransformer
fromsklearn.preprocessingimportOneHotEncoder
ct = ColumnTransformer(transformers=[('encoder', OneHotEncoder(), [3])],
remainder='passthrough')
X = np.array(ct.fit_transform(X))
```

```
print(X)
```

```
[[0.0 0.0 1.0 165349.2 136897.8 471784.1]
 [1.0 0.0 0.0 162597.7 151377.59 443898.53]
 [0.0 1.0 0.0 153441.51 101145.55 407934.54]
 [0.0 0.0 1.0 144372.41 118671.85 383199.62]
 [0.0 1.0 0.0 142107.34 91391.77 366168.42]
 [0.0 0.0 1.0 131876.9 99814.71 362861.36]

……
```

Splitting the dataset into the Training set and Test set

```
fromsklearn.model_selectionimporttrain_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2,
random_state = 0)
```

Training the Multiple Linear Regression model on the Training set

```
fromsklearn.linear_modelimportLinearRegression
regressor = LinearRegression()
regressor.fit(X_train, y_train)
```

Predicting the Test set results

```
y_pred = regressor.predict(X_test)
np.set_printoptions(precision=2)
print(np.concatenate((y_pred.reshape(len(y_pred),1),
y_test.reshape(len(y_test),1)),1))
```

```
[[103015.2  103282.38]
 [132582.28 144259.4 ]
 [132447.74 146121.95]
 [ 71976.1   77798.83]
 [178537.48 191050.39]
 [116161.24 105008.31]
 [ 67851.69  81229.06]
 [ 98791.73  97483.56]
 [113969.44 110352.25]
 [167921.07 166187.94]]
```

**** we didn't use feature scaling because coefficient multiply each independent variable or each feature. So, that doesn't matter because some feature high value but coefficient compensate and make them same scale.**

So not required Feature scaling.

*****Do we have avoid to do something dummy variable?**

No. Automatically trained Multiple Linear Regression model and Dummy variable also trap. Advance implementation inbuild machine learning model just few code that will trap the dummy variable. Not required to do anything.

******do we have to work in feature for best one with technique introduce like backward elimination technique to get highest p-value or best feature significant?**

Answer is also No. because same reason as Dummy variable trap. Multiple Linear Regression class in build model which automatically identify high p-value or highly significant feature.

So, Scikit learn (sklearn) everything will take care. So, need not to worry.

R Studio:

```
# Multiple Linear Regression


# Importing the dataset
dataset = read.csv('50_Startups.csv')


# Encoding categorical data
dataset$State = factor(dataset$State,
                       levels = c('New York', 'California', 'Florida'),
                       labels = c(1, 2, 3))


# Splitting the dataset into the Training set and Test set
#install.packages('caTools')
library(caTools)
set.seed(123)
split = sample.split(dataset$Profit, SplitRatio = 0.8)
training_set = subset(dataset, split == TRUE)
test_set = subset(dataset, split == FALSE)
```

```
# Feature Scaling

# training_set = scale(training_set)

# test_set = scale(test_set)



# Fitting Multiple Linear Regression to the Training set

regressor = lm(formula = Profit ~ .,

          data = training_set)

summary(regressor)

# Predicting the Test set results

y_pred = predict(regressor, newdata = test_set)
```

summary(regressor)

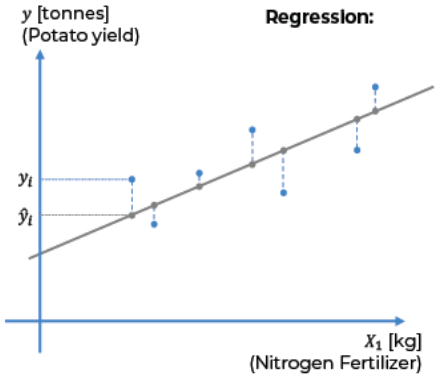
```
Call:
lm(formula = Profit ~ ., data = training_set)

Residuals:
   Min     1Q Median     3Q    Max
-33128  -4865      5   6098  18065

Coefficients:
                  Estimate Std. Error t value Pr(>|t|)
(Intercept)      4.965e+04  7.637e+03   6.501 1.94e-07 ***
R.D.Spend        7.986e-01  5.604e-02  14.251 6.70e-16 ***
Administration  -2.942e-02  5.828e-02  -0.505    0.617
Marketing.Spend  3.268e-02  2.127e-02   1.537    0.134
State2           1.213e+02  3.751e+03   0.032    0.974
State3           2.376e+02  4.127e+03   0.058    0.954
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 9908 on 34 degrees of freedom
Multiple R-squared:  0.9499,   Adjusted R-squared:  0.9425
F-statistic:   129 on 5 and 34 DF,  p-value: < 2.2e-16
```

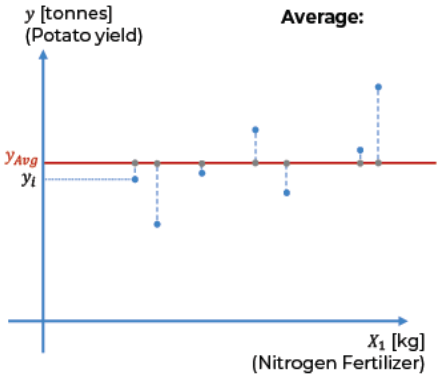
***** This information very interesting information that will build robust model, not only P-value that will help to select the optimal team of independent variable. R-squared and adjusted R-squared which will help to build more robust model. Last column don’t have name only star. P-value is less than 5% significant level then the more independent**

variable significant on dependent variable. P-value is more than 5% significant level then independent variable less significant on dependent variable.

$$Potatoes[t] = 8t + 3\frac{t}{kg} \times Fertilizer[kg] - 0.54\frac{t}{°C} \times AvgTemp[°C] + 0.04\frac{t}{mm} \times Rain[mm]$$

R-squared





$$SS_{res} = SUM(y_i - \hat{y}_i)^2$$

$$SS_{tot} = SUM(y_i - y_{avg})^2$$

$$R^2 = 1 - \frac{SS_{res}}{SS_{tot}}$$

Rule of thumb (for our tutorials)*:
1.0 = Perfect fit (suspicious)
~0.9 = Very good
<0.7 = Not great
<0.4 = Terrible
<0 = Model makes no sense for this data

**This is highly dependent on the context*

Adjusted R-squared:

$$R^2 = 1 - \frac{SS_{res}}{SS_{tot}}$$

R^2 – Goodness of fit
(greater is better)

Problem:

$\hat{y} = b_0 + b_1X_1 + b_2X_2 + b_3X_3$

$SS_{res} = SUM(y_i - \hat{y}_i)^2$

SS_{tot} doesn't change
SS_{res} will decrease or stay the same *(This is because of Ordinary Least Squares: SS_{res}-> Min)*

Solution:

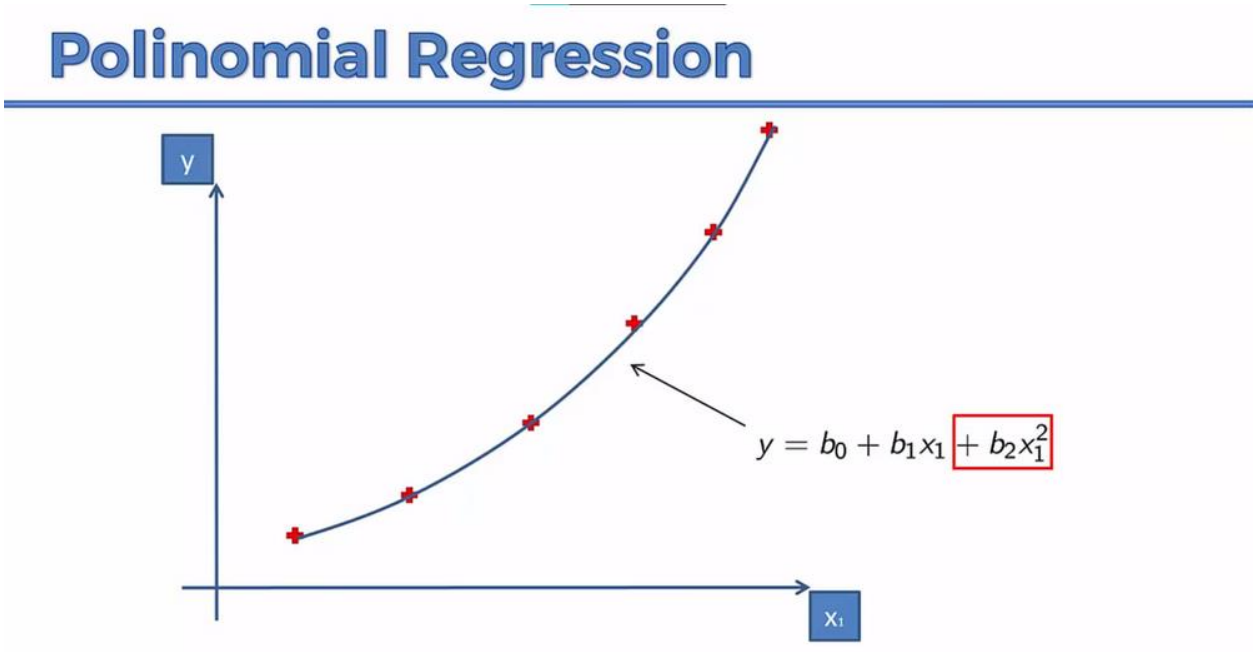
$$Adj\ R^2 = 1 - (1 - R^2) \times \frac{n-1}{n-k-1}$$

k – number of independent variables
n – sample size

Polynomial Regression

Polynomial regression is a form of regression analysis in which the relationship between the independent variable x and the dependent variable y is modeled as an n-th degree polynomial in x . It is used to model non-linear relationships between the variables . Polynomial regression fits a nonlinear relationship between the value of x and the corresponding conditional mean of y, denoted E(y|x) .
In polynomial regression, we describe the relationship between the independent variable x and the dependent variable y using an n-th degree polynomial in x. The polynomial regression equation is given by:

y = b0 + b1*x + b2*x^2 + b3*x^3 + ... + bn*x^n
where b0, b1, b2, ..., bn are coefficients and n is the degree of the polynomial



Position_Salaries_Court:

	A	B	C
1	Position	Level	Salary
2	Machine Operator	3	32750
3	Library Attendant	4	41500
4	Junior Judicial Asst	5	52875
5	Judicial Asst	7	65000
6	Senior Judicial Asst	8	78000
7	Administrative Officer	11	125000
8	Assistant Register	12	145000
9	Deputy Register	13	187700
10	Joint Register	14	247800
11	Register	15	310500

Importing the libraries

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

Importing the dataset

```
dataset = pd.read_csv('Position_Salaries_Court.csv')
X = dataset.iloc[:, 1:-1].values
y = dataset.iloc[:, -1].values
```

Training the Linear Regression model on the whole dataset

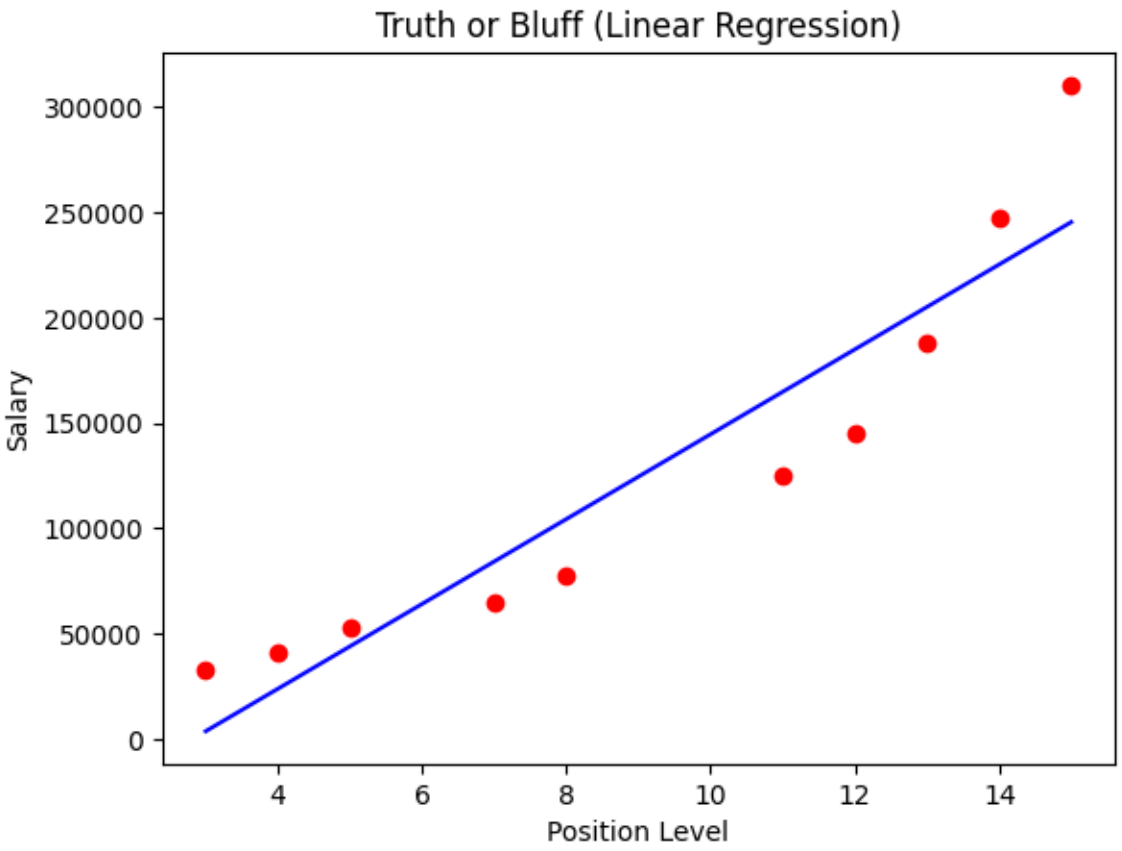
```
from sklearn.linear_model import LinearRegression
lin_reg = LinearRegression()
lin_reg.fit(X, y)
```

Training the Polynomial Regression model on the whole dataset

```
from sklearn.preprocessing import PolynomialFeatures
poly_reg = PolynomialFeatures(degree = 4)
X_poly = poly_reg.fit_transform(X)
lin_reg_2 = LinearRegression()
lin_reg_2.fit(X_poly, y)
```

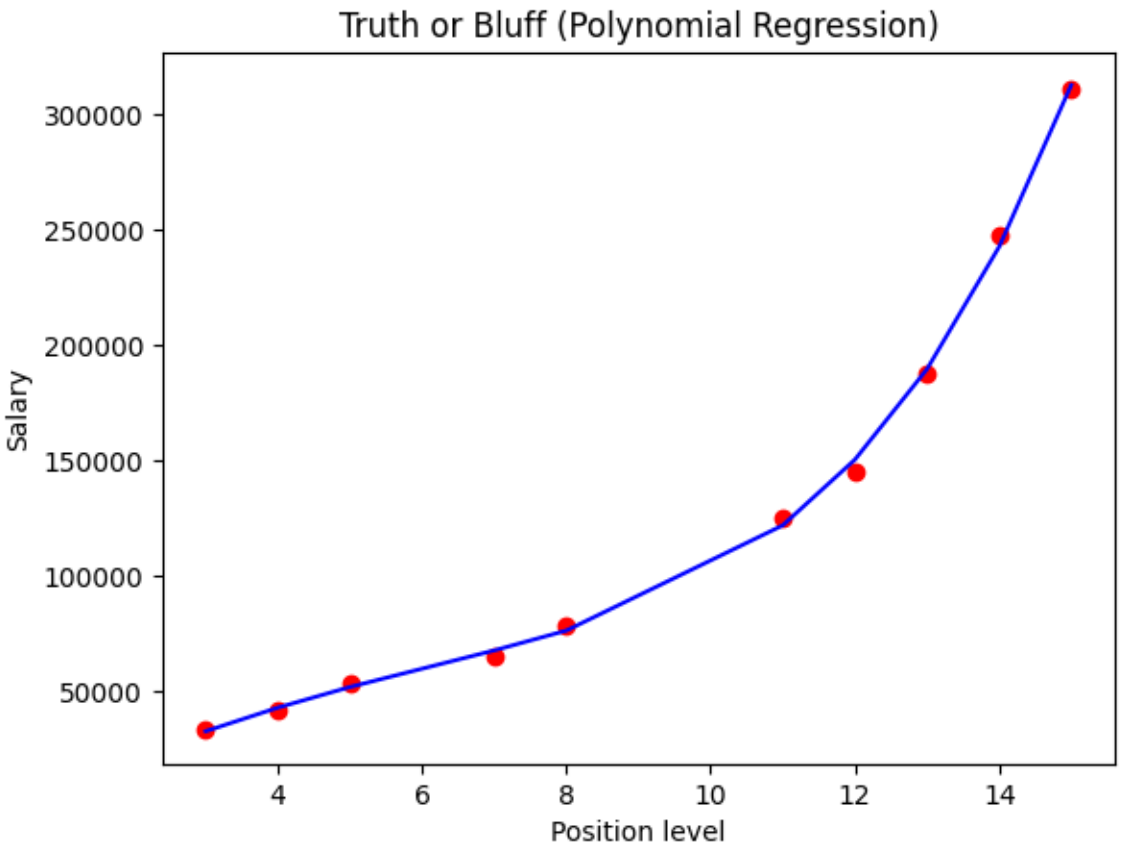
Visualising the Linear Regression results

```
plt.scatter(X, y, color = 'red')
plt.plot(X, lin_reg.predict(X), color = 'blue')
plt.title('Truth or Bluff (Linear Regression)')
plt.xlabel('Position Level')
plt.ylabel('Salary')
plt.show()
```



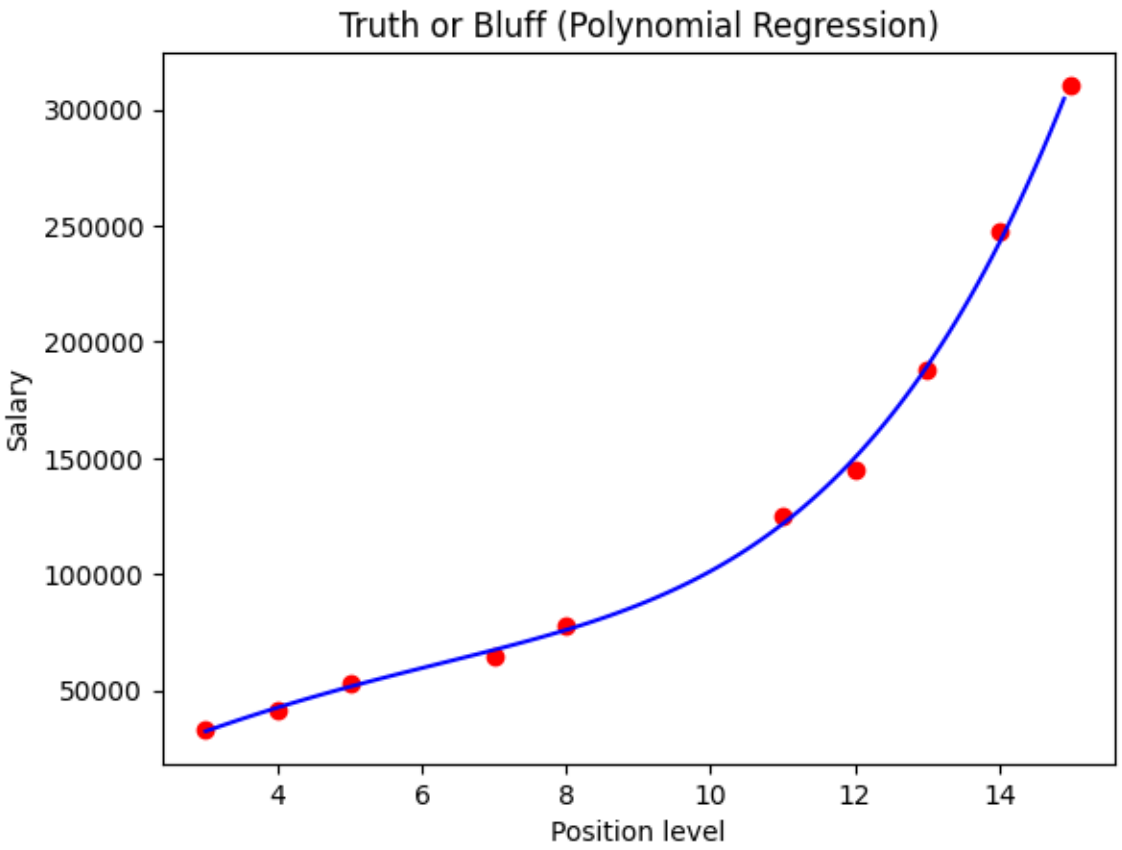
Visualising the Polynomial Regression results

```
plt.scatter(X, y, color = 'red')
plt.plot(X, lin_reg_2.predict(poly_reg.fit_transform(X)), color = 'blue')
plt.title('Truth or Bluff (Polynomial Regression)')
plt.xlabel('Position level')
plt.ylabel('Salary')
plt.show()
```



Visualising the Polynomial Regression results (for higher resolution and smoother curve)

```
X_grid = np.arange(min(X), max(X), 0.1)
X_grid = X_grid.reshape((len(X_grid), 1))
plt.scatter(X, y, color = 'red')
plt.plot(X_grid, lin_reg_2.predict(poly_reg.fit_transform(X_grid)), color =
'blue')
plt.title('Truth or Bluff (Polynomial Regression)')
plt.xlabel('Position level')
plt.ylabel('Salary')
plt.show()
```



Predicting a new result with Linear Regression

```
lin_reg.predict([[10.5]])
```

O/P: array([154802.46212121])

Predicting a new result with Polynomial Regression

```
lin_reg_2.predict(poly_reg.fit_transform([[10.5]]))
```

O/P: array([110617.935597])

Support Vector Regression (SVR)

Support Vector Regression (SVR) is a type of machine learning algorithm used for regression analysis. It is an extension of Support Vector Machines (SVMs) and works by finding a hyperplane that best fits the data points in a high-dimensional space. The goal of SVR is to find a function that approximates the relationship between the input variables and a continuous target variable, while minimizing the prediction error.

Support Vector Regression (SVR) using Linear and Non-Linear Kernels in Scikit Learn

Support vector regression (SVR) is a type of support vector machine (SVM) that is used for regression tasks. It tries to find a function that best predicts the continuous output value for a given input value.

SVR can use both linear and non-linear kernels. A linear kernel is a simple dot product between two input vectors, while a non-linear kernel is a more complex function that can capture more intricate patterns in the data. The choice of kernel depends on the data's characteristics and the task's complexity.

In scikit-learn package for Python, you can use the ***'SVR'*** class to perform SVR with a linear or non-linear ***'kernel'***. To specify the kernel, you can set the kernel parameter to ***'linear'*** or ***'RBF'*** (radial basis function).

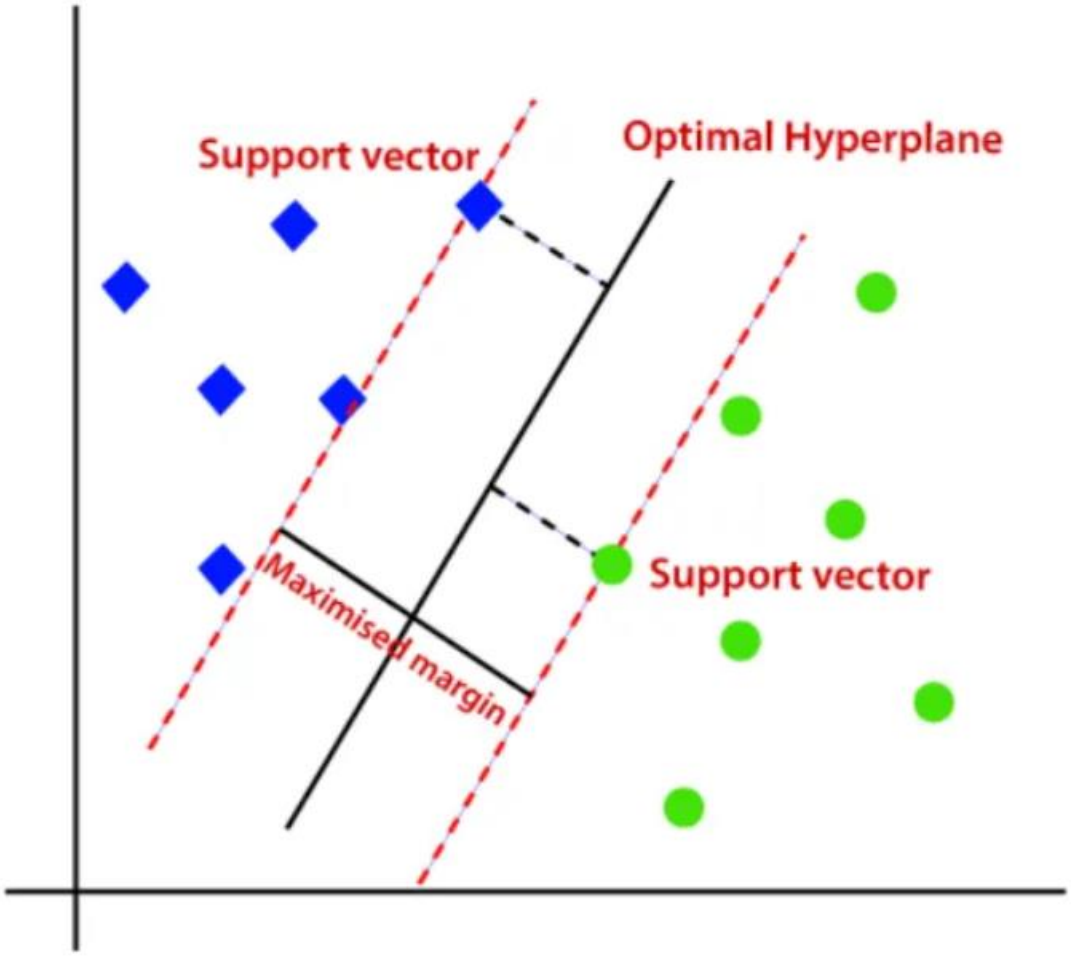
Concepts related to the Support vector regression (SVR):

There are several concepts related to support vector regression (SVR) that you may want to understand in order to use it effectively. Here are a few of the most important ones:

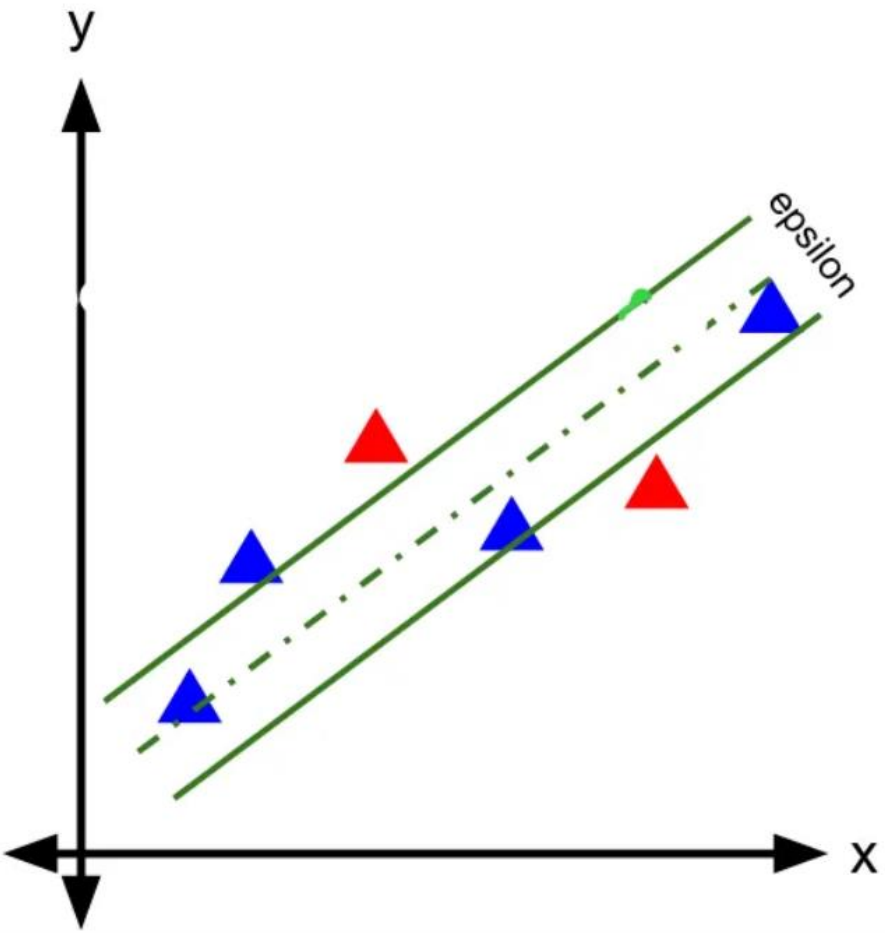
- **Support vector machines (SVMs):** SVR is a type of support vector machine (SVM), a supervised learning algorithm that can be used for classification or regression tasks. SVMs try to find the hyperplane in a high-dimensional space that maximally separates different classes or output values.
- **Kernels:** SVR can use different types of kernels, which are functions that determine the similarity between input vectors. A linear kernel is a simple dot product between two input vectors, while a non-linear kernel is a more complex function that can capture more intricate patterns in the data. The choice of kernel depends on the data's characteristics and the task's complexity.
- **Hyperparameters:** SVR has several hyperparameters that you can adjust to control the behavior of the model. For example, the **'C'** parameter controls the trade-off between the insensitive loss and the sensitive loss. A larger value of **'C'** means that the model will try to minimize the insensitive loss more, while a

smaller value of C means that the model will be more lenient in allowing larger errors.

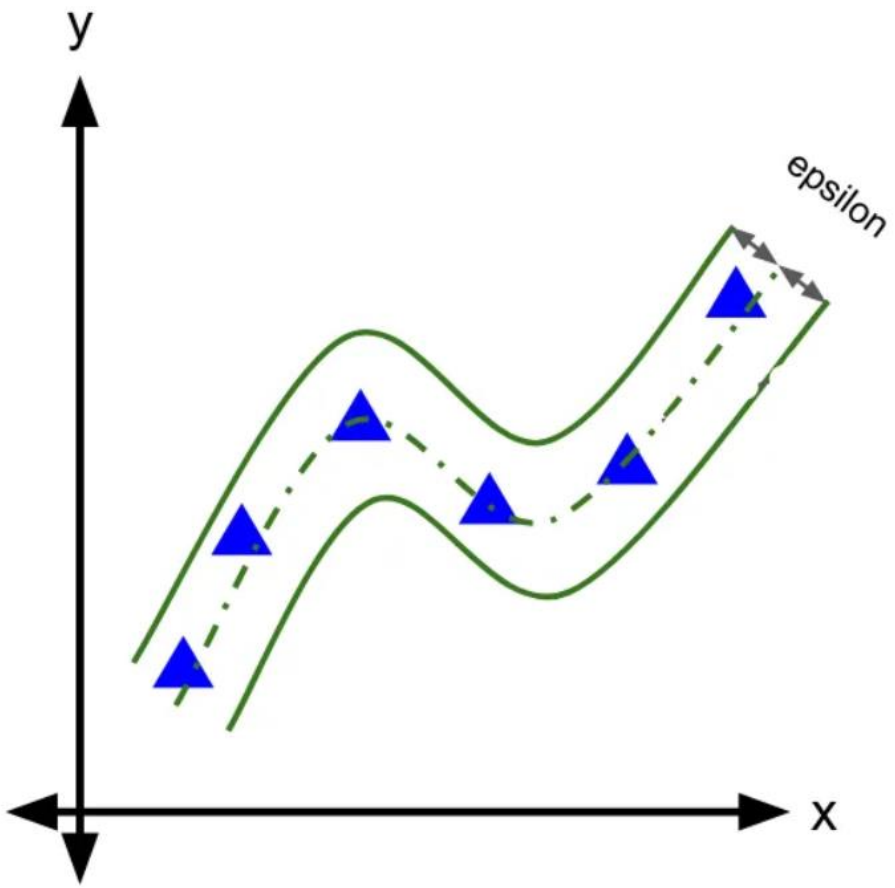
- **Model evaluation:** Like any machine learning model, it's important to evaluate the performance of an SVR model. One common way to do this is to split the data into a training set and a test set, and use the training set to fit the model and the test set to evaluate it. You can then use metrics like mean squared error (MSE) or mean absolute error (MAE) to measure the error between the predicted and true output values.



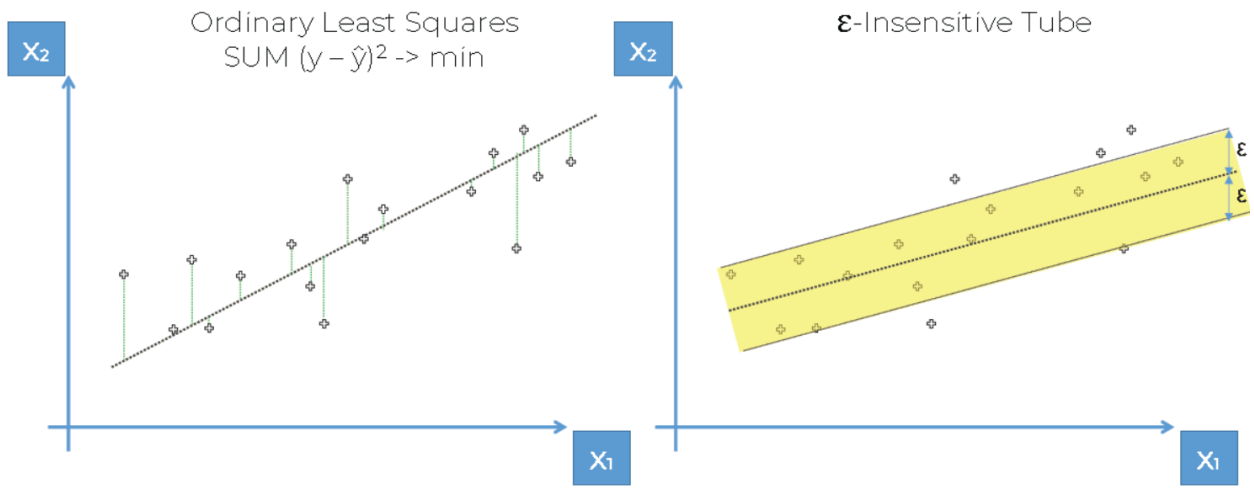
1. **Linear SVR:** This type of SVR uses a linear kernel function and aims to find a linear hyperplane that best fits the data.



2. **Nonlinear SVR:** Nonlinear SVR employs various kernel functions (e.g., polynomial, Gaussian radial basis function) to capture complex nonlinear relationships between variables.

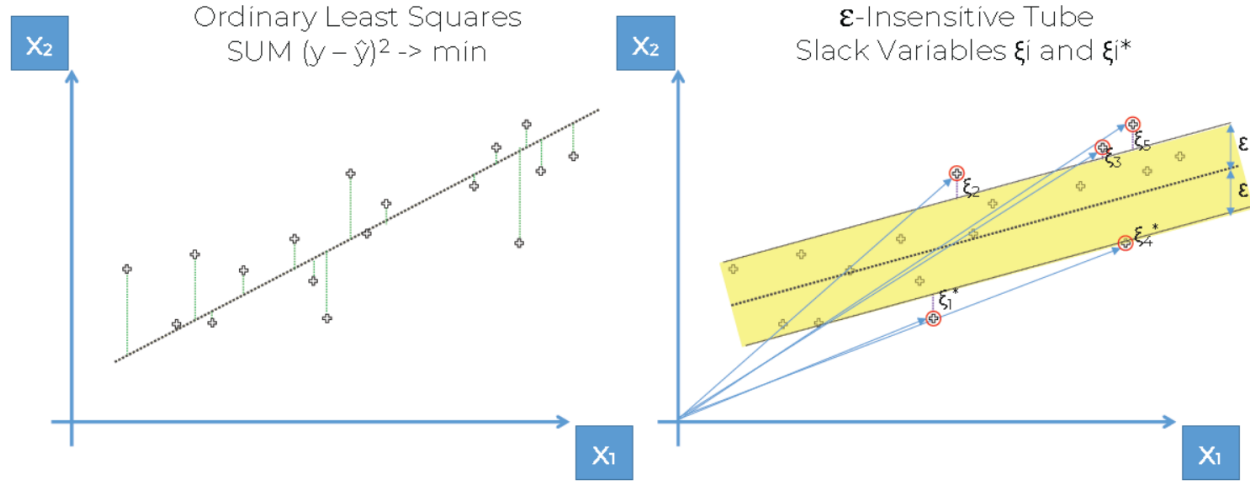


SVR Intuition



SVR Intuition

$$\frac{1}{2}\|w\|^2 + C\sum_{i=1}^{m}(\xi_i + \xi_i^*) \rightarrow min$$



Epsilon insensitive tube: This tube thinks about merging of error that we are allowing our model to have not care inside the tube mean any point discriminant or distort the line don't care because this tube is insensitive. Key behind the support vector model, it's given the little bit movement of little bit buffer to our model. Same time we have the points outside the insensible tube. For them we do care about the error of outside points. Measure the distance

form point to tube itself not trained line. There called Slack variables. Minimizing slack variable distances. Actually, this distance depends on what the tube looks like and how the tube positioned.

Why this method called support vector regression because all of the points outside the tube or any point on the path is a vector into two-dimensional space or multi two dimensional features. We highlighted here outside the tube, they are supporting vector because they are dictating how the tube structure actually they are support the machine of the tube. So, this is Support Vector Regression.

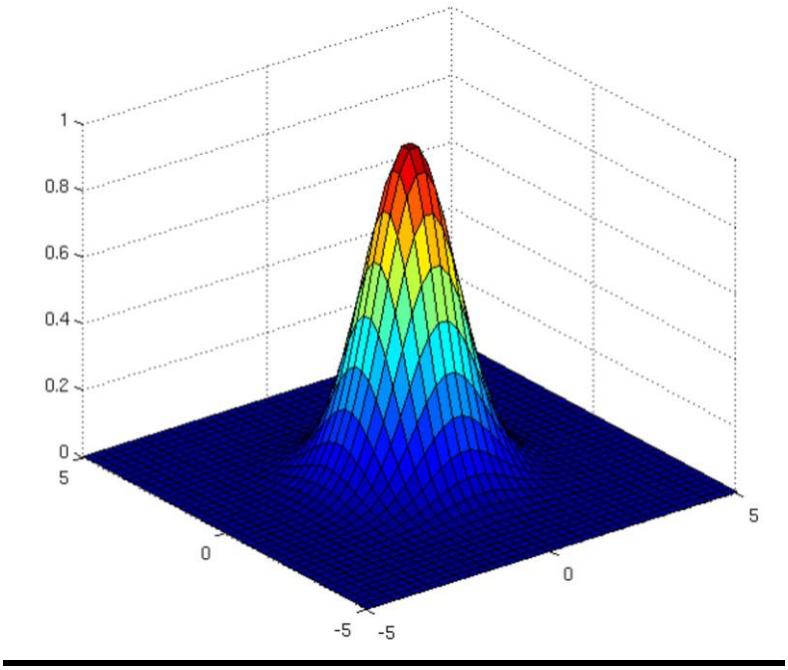
Non Linear SVR

SVR can use both linear and non-linear kernels. A linear kernel is a simple dot product between two input vectors, while a non-linear kernel is a more complex function that can capture more intricate patterns in the data. The choice of kernel depends on the data's characteristics and the task's complexity.

In scikit-learn package for Python, you can use the ***'SVR'*** class to perform SVR with a linear or non-linear ***'kernel'***. To specify the kernel, you can set the kernel parameter to ***'linear'*** or ***'RBF'*** (radial basis function).

Section on Kernel SVM:
• SVM Intuition

Kernel SVM Intuition
• Mapping to a higher dimension
• The Kernel Trick
• Types of Kernel Functions
• Non-linear Kernel SVR



Radial basis function kernel (aka squared-exponential kernel).

The RBF kernel is a stationary kernel. It is also known as the “squared exponential” kernel. It is parameterized by a length scale parameter l>0, which can either be a scalar (isotropic variant of the kernel) or a vector with the same number of dimensions as the inputs X (anisotropic variant of the kernel). The kernel is given by:

$$k(x_i, x_j) = \exp\left(-\frac{d(x_i, x_j)^2}{2l^2}\right)$$

where l is the length scale of the kernel and d(·,·) is the Euclidean distance.

This kernel is infinitely differentiable, which implies that GPs with this kernel as covariance function have mean square derivatives of all orders, and are thus very smooth.

RBF kernels are the most generalized form of kernelization and is one of the most widely used kernels due to its similarity to the Gaussian distribution. The RBF kernel function for two points X_1 and X_2 computes the similarity or how close they are to each other. This kernel can be mathematically represented as follows:

$$K(X_1, X_2) = exp(-\frac{||X_1 - X_2||^2}{2\sigma^2})$$

where,

1. ‘σ’ is the variance and our hyperparameter
2. $||X_1 - X_2||$ is the Euclidean (L_2-norm) Distance between two points X_1 and X_2

Let d_{12} be the distance between the two points X_1 and X_2, we can now represent d_{12} as follows:

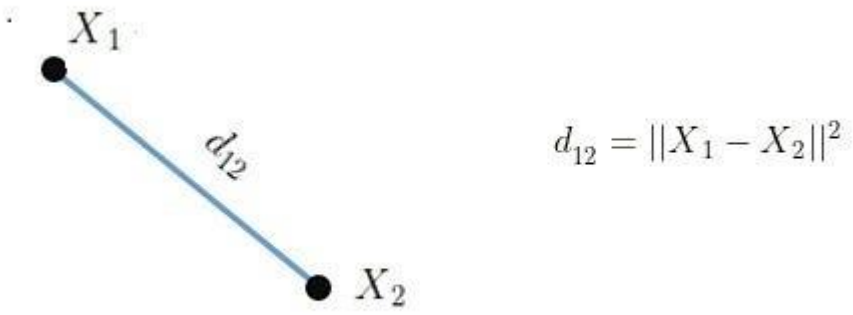


Fig 2: Distance between two points in space [Image by Author]

The kernel equation can be re-written as follows:

$$K(X_1, X_2) = exp(-\frac{d_{12}}{2\sigma^2})$$

The maximum value that the RBF kernel can be is 1 and occurs when d_{12} is 0 which is when the points are the same, i.e. $X_1 = X_2$.

1. When the points are the same, there is no distance between them and therefore they are extremely similar
2. When the points are separated by a large distance, then the kernel value is less than 1 and close to 0 which would mean that the points are dissimilar

Dataset: Position_Salaries_Court.csv

Position_Salaries_Court

Position	Level	Salary
Machine Operator	3	32750
Library Attendant	4	41500
Junior Judicial Asst	5	52875
Judicial Asst	7	65000
Senior Judicial Asst	8	78000
Administrative Officer	11	125000
Assistant Register	12	145000
Deputy Register	13	187700
Joint Register	14	247800
Register	15	310500

Python Code for SVR

Support Vector Regression (SVR)

Importing the libraries

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

Importing the dataset

```
dataset = pd.read_csv('Position_Salaries_Court.csv')
X = dataset.iloc[:, 1:-1].values
y = dataset.iloc[:, -1].values
```

```
print(X)
```

```
[[ 3]
 [ 4]
 [ 5]
 [ 7]
```

```
 [ 8]
 [11]
 [12]
 [13]
 [14]
 [15]]
```

```
print(y)
```
```
[ 32750  41500  52875  65000  78000 125000 145000 187700 247800 310500]
```

```
y = y.reshape(len(y),1)
print(y)
```
```
[[ 32750]
 [ 41500]
 [ 52875]
 [ 65000]
 [ 78000]
 [125000]
 [145000]
 [187700]
 [247800]
 [310500]]
```

[**Transformation salary in two-dimension array because standardization for feature scaling expect two dimension when we apply fit_transform method input twoD array]

Feature Scaling

```
from sklearn.preprocessing import StandardScaler
sc_X = StandardScaler()
sc_y = StandardScaler()
X = sc_X.fit_transform(X)
y = sc_y.fit_transform(y)
```

[**Here we are not using same standard scale object for Salary & Level because compute the mean and standard deviation different for Position Level & Salary that's why we are using two different objects]

```
print(X)
```
```
[[-1.4966941 ]
 [-1.25529183]
 [-1.01388955]
 [-0.531085  ]
 [-0.28968273]
 [ 0.43452409]
 [ 0.67592637]
 [ 0.91732864]
 [ 1.15873092]
```

```
 [ 1.40013319]]
```

```
print(y)
```

```
[[-1.07054353]
 [-0.97282799]
 [-0.84579779]
 [-0.71039197]
 [-0.5652146 ]
 [-0.04034256]
 [ 0.18300725]
 [ 0.65985908]
 [ 1.33102524]
 [ 2.03122688]]
```

Training the SVR model on the whole dataset

```
from sklearn.svm import SVR
regressor = SVR(kernel = 'rbf')
regressor.fit(X, y)
```

[** We didn't split training set and test set. Here sklearn→ library; svm→model; SVR→class; regressor object is the instance of the class SVR; kernel(parameter)='rbf'(value) → non-linear relationship RBF (A radial basis function (RBF) is a **real-valued function whose value depends only on the distance between the input and some fixed point**); `regressor.fit(X, y)→ train the whole data set use fit method`]

```
/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py:1143:
DataConversionWarning: A column-vector y was passed when a 1d array was
expected. Please change the shape of y to (n_samples, ), for example using
ravel().
  y = column_or_1d(y, warn=True)
```

Predicting a new result

```
sc_y.inverse_transform(regressor.predict(sc_X.transform([[6.5]])).reshape(-
1,1))
```

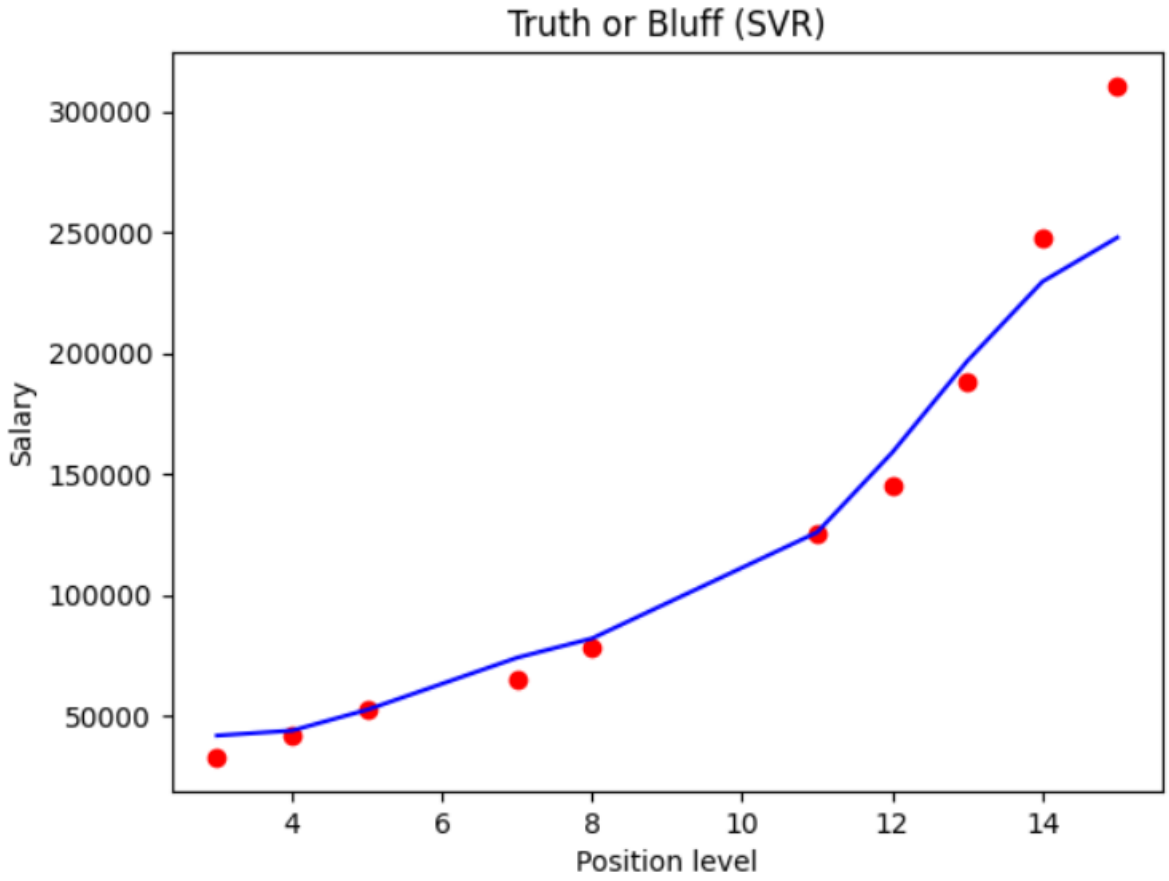
[** regressor.predict(sc_X.transform([[6.5]])→ predict method take input scale value of transform method twoD array value; sc_y.inverse_transform → Reverse scaling whole prediction that will give output as Salary. sc_y(object)→output; inverse_transform method reverse transform method; reshape(-1,1)→ reshape method to avoid format error]

```
array([[69084.67433045]])
```

Visualising the SVR results

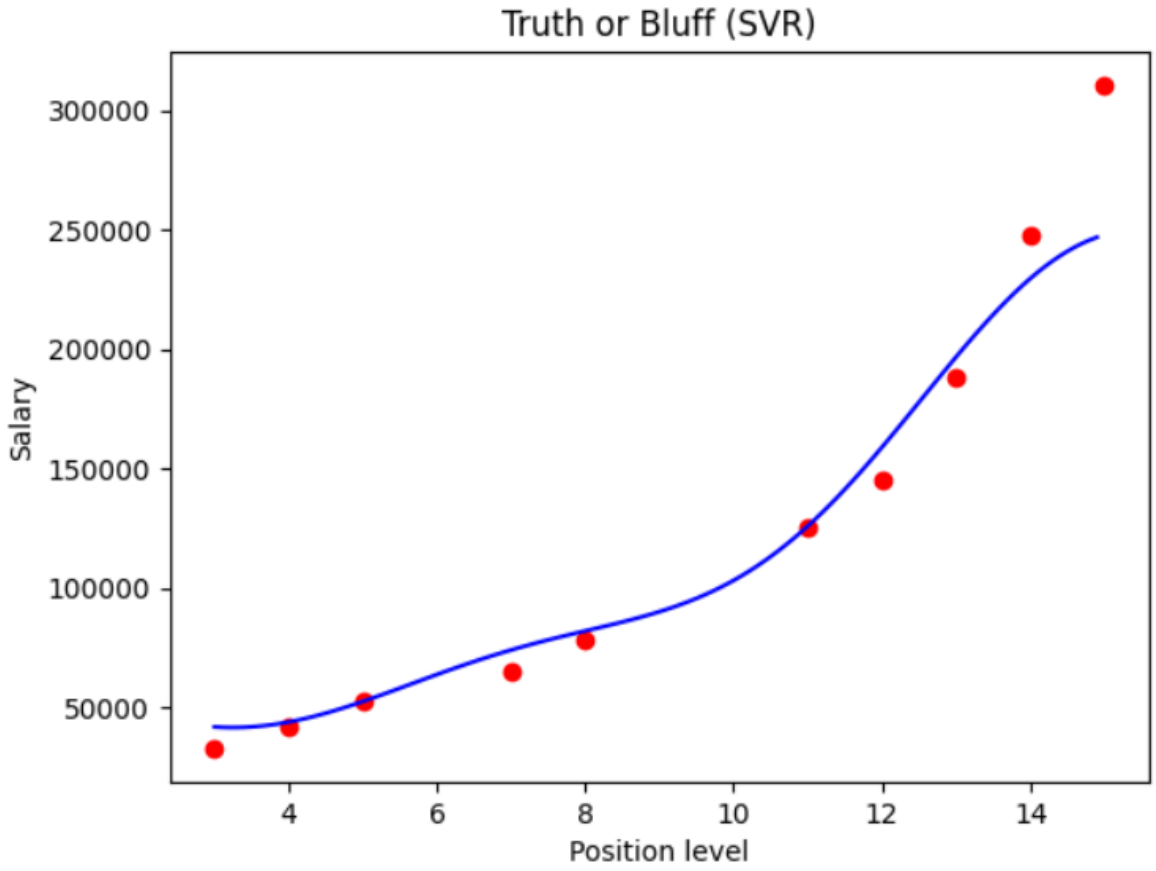
```
plt.scatter(sc_X.inverse_transform(X), sc_y.inverse_transform(y), color =
'red')
plt.plot(sc_X.inverse_transform(X),
sc_y.inverse_transform(regressor.predict(X).reshape(-1,1)), color = 'blue')
plt.title('Truth or Bluff (SVR)')
plt.xlabel('Position level')
plt.ylabel('Salary')
plt.show()
```

[**sc_X.inverse_transform(X), sc_y.inverse_transform(y)→ put them original scale that is reverse transform ;
sc_y.inverse_transform(regressor.predict(X).reshape(-1,1)) → same like prediction for all observation that's why X]



Visualising the SVR results (for higher resolution and smoother curve)

```
X_grid = np.arange(min(sc_X.inverse_transform(X)),
max(sc_X.inverse_transform(X)), 0.1)
X_grid = X_grid.reshape((len(X_grid), 1))
plt.scatter(sc_X.inverse_transform(X), sc_y.inverse_transform(y), color =
'red')
plt.plot(X_grid,
sc_y.inverse_transform(regressor.predict(sc_X.transform(X_grid)).reshape(-
1,1)), color = 'blue')
plt.title('Truth or Bluff (SVR)')
plt.xlabel('Position level')
plt.ylabel('Salary')
plt.show()
```

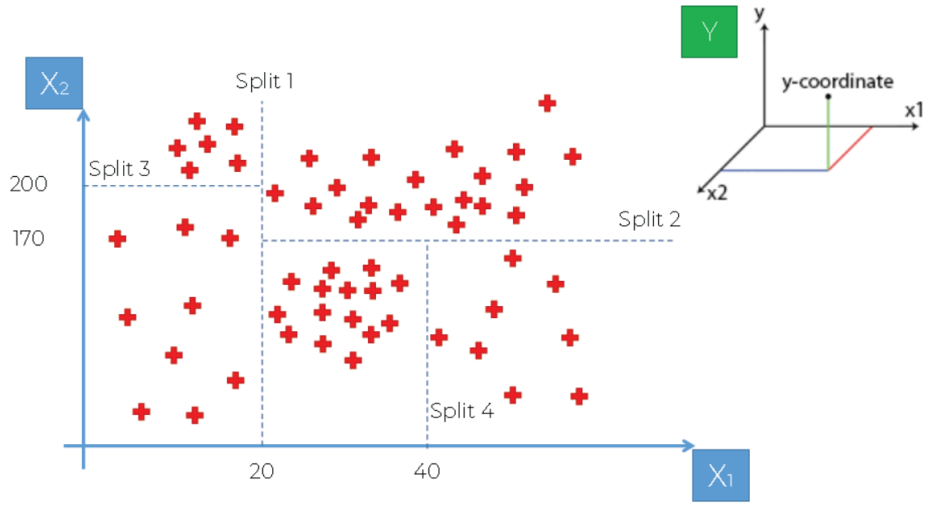


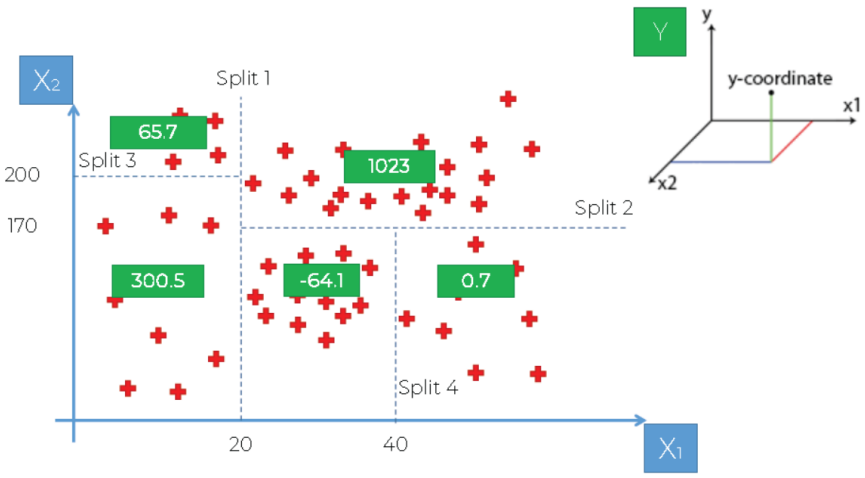
Decision Tree Regression

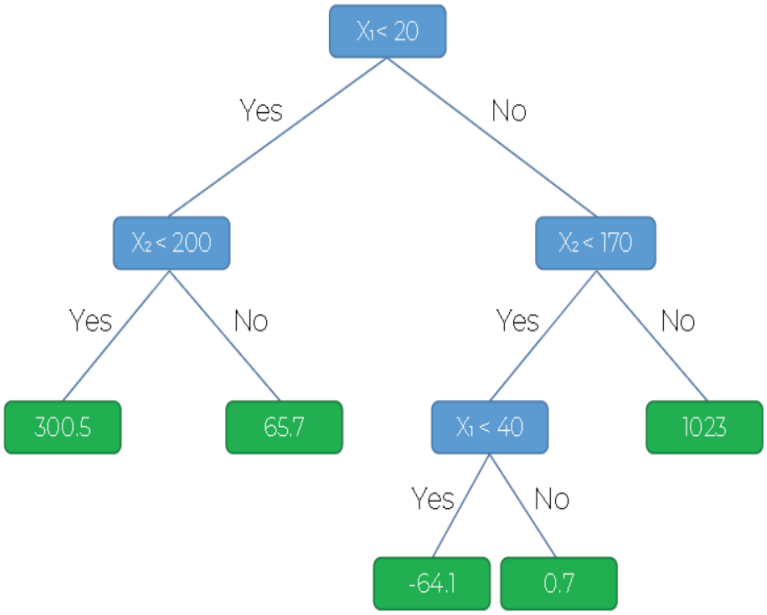
Decision tree regression is a **supervised learning** algorithm used for **predictive modeling**. It is a non-parametric method that can be used for both **classification** and **regression** tasks. The algorithm works by recursively partitioning the input space into smaller regions, where each region is associated with a **predictive model**. The model is constructed by fitting a simple function to the data in each region. The function is typically a **constant** value for **regression** tasks and a **majority class for classification** tasks. Decision tree regression is simple to understand and interpret, requires little data preparation, and can handle both numerical and categorical data.

CART(Classification And Regression Tree) is a variation of the decision tree algorithm. It can handle both classification and regression tasks. Scikit-Learn uses the Classification And Regression Tree (CART) algorithm to train Decision Trees (also called "growing" trees).

Decision Tree Intuition




X2
Split 1
65.7
Split 3
1023
200
170
Split 2
300.5
-64.1
0.7
Split 4
20
40
X1
Y
y
y-coordinate
x1
x2


X1 < 20
Yes
No
X2 < 200
X2 < 170
Yes
No
Yes
No
300.5
65.7
X1 < 40
1023
Yes
No
-64.1
0.7

Dataset: Position_Salaries_Court.csv

Position_Salaries_Court

Position	Level	Salary
Machine Operator	3	32750
Library Attendant	4	41500
Junior Judicial Asst	5	52875
Judicial Asst	7	65000
Senior Judicial Asst	8	78000
Administrative Officer	11	125000
Assistant Register	12	145000
Deputy Register	13	187700
Joint Register	14	247800
Register	15	310500

Python Code for Decision Regression

Importing the libraries

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

Importing the dataset

```
dataset = pd.read_csv('Position_Salaries_Court.csv')
X = dataset.iloc[:, 1:-1].values
y = dataset.iloc[:, -1].values
```

** Don’t have to apply feature scaling because prediction from decision tree regression are resulting of successive split of data you know different nodes of tree therefore not some equation like with previous model that why no feature scaling is needed to know split different values of feature to deferent categories lead to different prediction. So, no feature scaling.

Now, we will go for Training the Decision Tree model on the hole dataset. You can search from Google decision tree regression class of scikit learn. You can find the name of class probably 1st link. How to build and trained ML model scikit learn and you will learn Training the decision tree.

Training the Decision Tree Regression model on the whole dataset

```
from sklearn.tree import DecisionTreeRegressor
regressor = DecisionTreeRegressor(random_state = 0)
regressor.fit(X, y)
```

** tree → model; DecisionTreeRegressor →Class; regressor → object;
Here parameter random_state =0 because if you give parameter then tuning is required mean optimization required. Later we will learn parameter optimizing. fit(x,y) → method which take the input matrix of feature x, the whole matrix then dependent variable y. Actually trained the Decision tree regression corelation between Position level and Salary.

Predicting a new result

```
regressor.predict([[6.5]])
```

O/P: array([65000.])

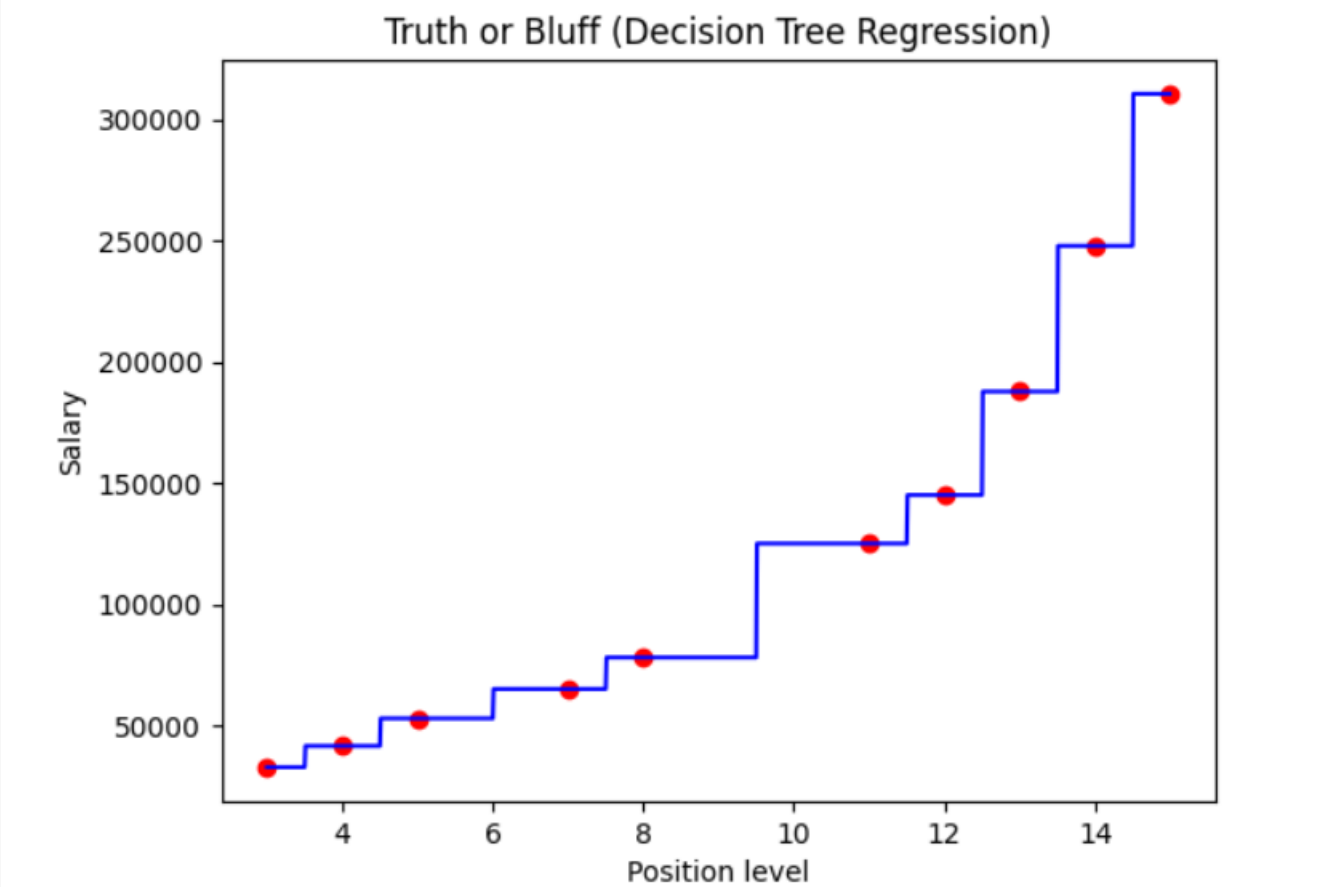
** predict method only observation 6.5 input as a twoD array that’s why double bracket.

Visualising the Decision Tree Regression results (higher resolution)

```
X_grid = np.arange(min(X), max(X), 0.01)
X_grid = X_grid.reshape((len(X_grid), 1))
plt.scatter(X, y, color = 'red')
plt.plot(X_grid, regressor.predict(X_grid), color = 'blue')
plt.title('Truth or Bluff (Decision Tree Regression)')
plt.xlabel('Position level')
```

```
plt.ylabel('Salary')
plt.show()
```

O/P:



** Here we are using same code like polynomial regressor but slightly change because no feature scaling so need not to require transform matrix. So, simple but one thing you can notice result is not so good because:

Decision tree regression model really is not a best adapted to two-dimension dataset on feature and dependent variable but implementation very easy. Decision tree splitting the data through successive nodes. For high dimensional dataset it is very useful.