More Regression

Types of Regression

- Simple Linear Regression
- Multiple Linear Regression
- Polynomial Regression
- Support Vector Regression
- Decision Tree Regression
- Random Forest Regression

Simple Linear Regression

$$y = \beta_0 + \beta_1 x_1 + \epsilon$$

where:

- y = dependent variable
- x = explanatory variable
- β_0 = y intercept
- β_n = slope coefficients
- ϵ = the model's error term

Multiple Linear Regression

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 \ldots + \beta_n x_n + \epsilon$$

Polynomial Regression

$$y = \beta_0 + \beta_1 x + \beta_2 x^2 \ldots + \beta_n x^n + \epsilon$$

- Polynomial regression adds powers of x (like x^2, x^3) to capture curved relationships.
- Use polynomial regression when the relationship between x and y is curved (e.g., U-shape), not well described by a straight line.
- When we talk about linear, we're talking about the coefficients, not x
- The degree is the highest power of x in the polynomial (degree 1 = linear, degree 2 = quadratic, etc.)

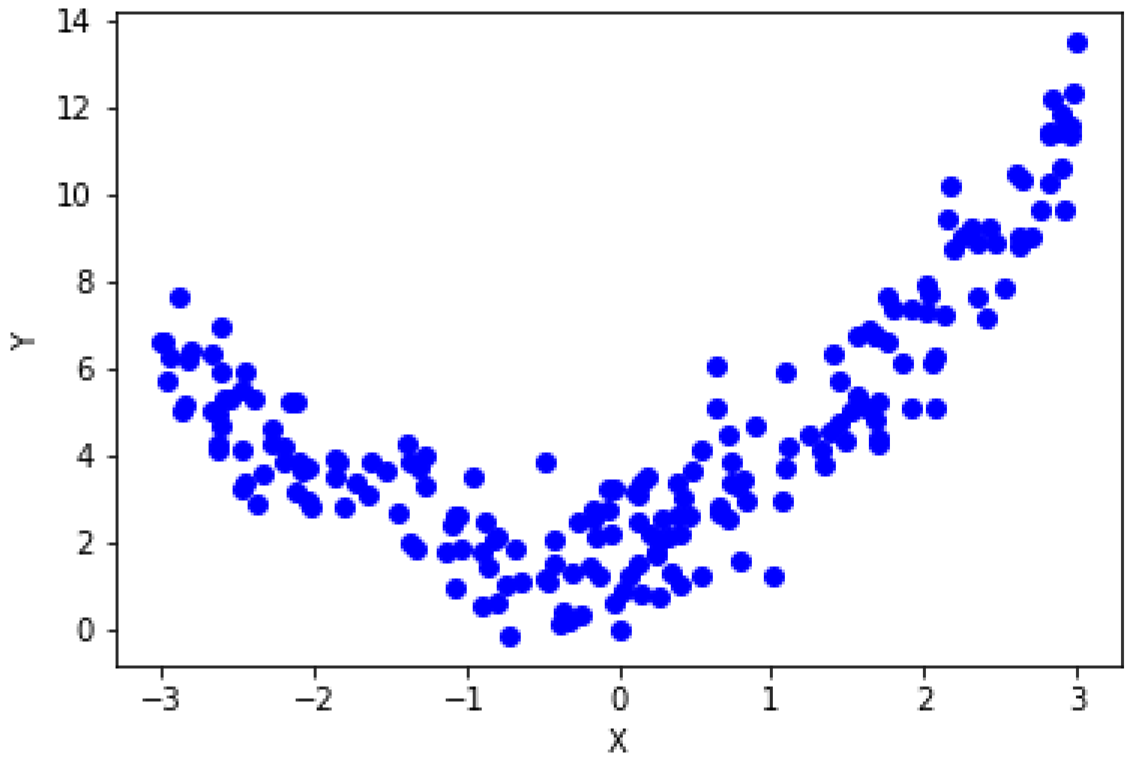
Check this out:

```
Degree |    Name      | Example
  0    | Constant     | 5
  1    |  Linear      | x + 3
  2    | Quadratic    | x^2 - x + 1
etc.
```

https://www.mathsisfun.com/algebra/degree-expression.html

```
    import numpy as np
    import matplotlib.pyplot as plt
    from sklearn.model_selection import train_test_split
    from sklearn.linear_model import LinearRegression
```

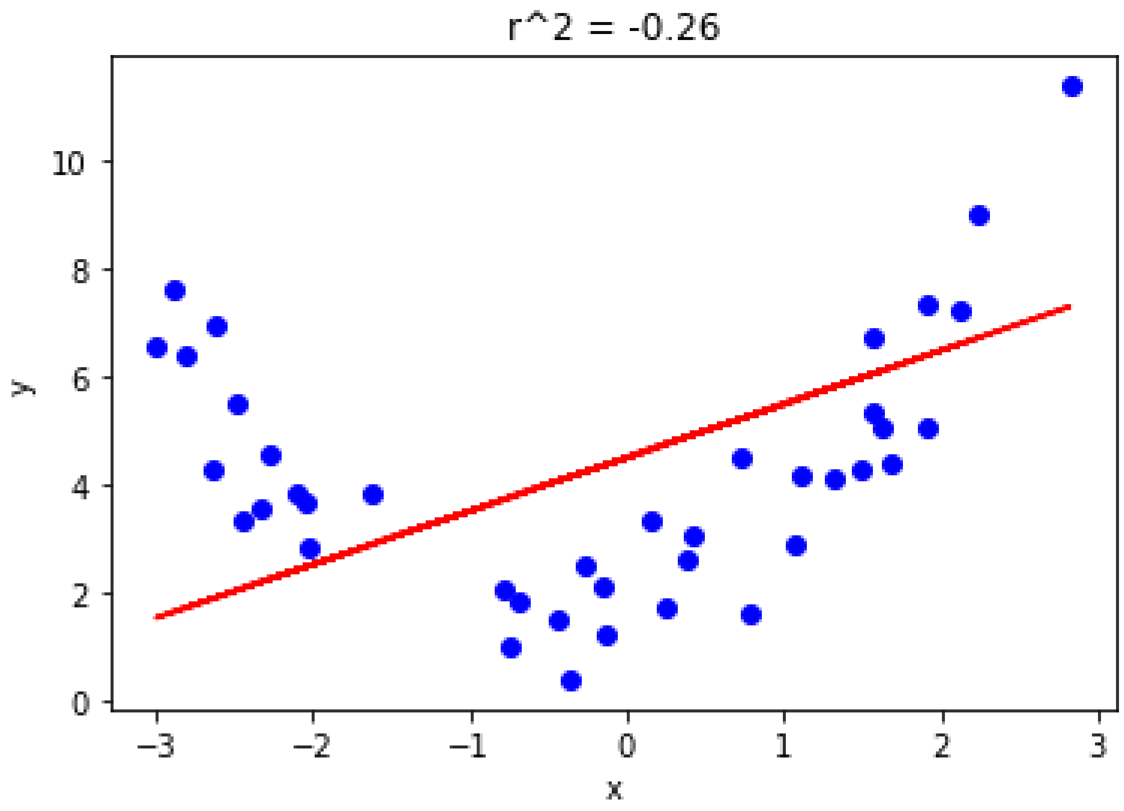
```
from sklearn.preprocessing import PolynomialFeatures
from sklearn.metrics import r2_score

X = 6 * np.random.rand(200, 1) - 3
noise = np. random. normal(0, 1, X.shape)
y = 0.8*X**2 + 0.9*X + 2 + noise
# y = 0.7 * X**3 + 0.8 * X**2 + 0.9*X + 2 + noise

plt.scatter(X, y, color='blue')
plt.xlabel("X")
plt.ylabel("Y")
plt.show()
```



```
# train test split
X_train, X_test, y_train, y_test = train_test_split(X
```

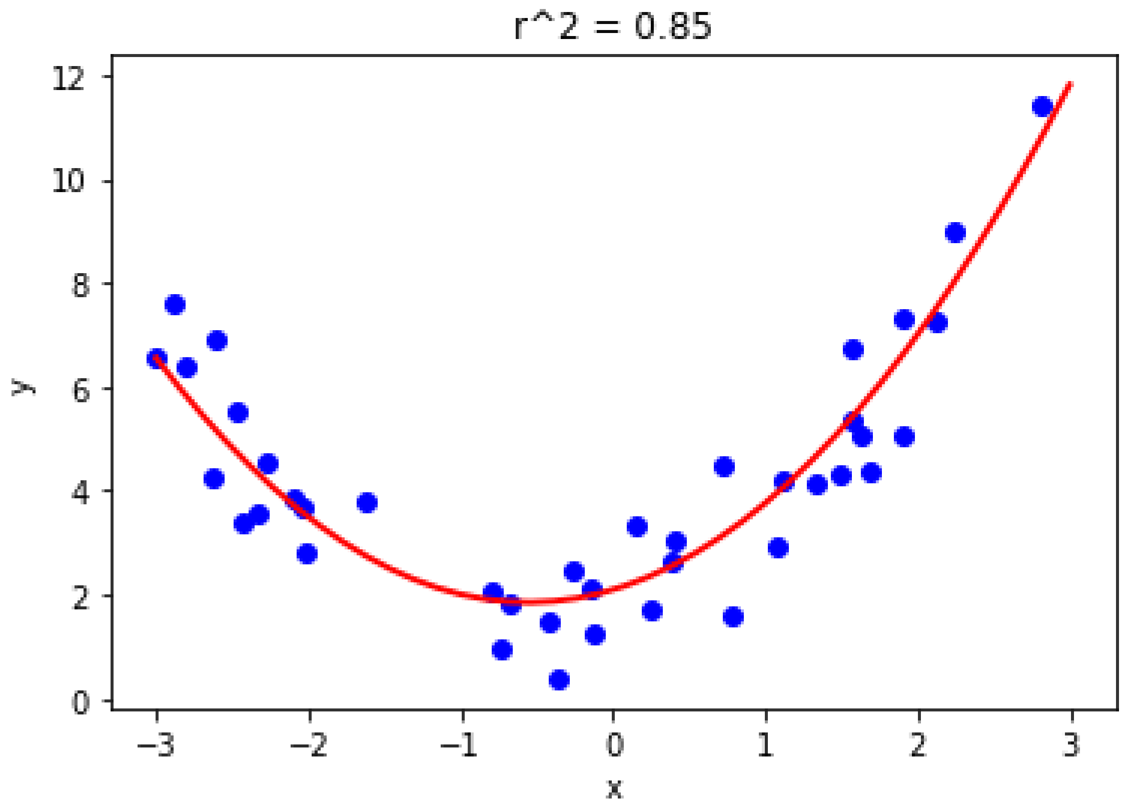
```
model = LinearRegression()
model.fit(X_train, y_train)
predictions = model.predict(X_test)
```

```
plt.plot(X_test, predictions, color='red')
plt.scatter(X_test, y_test, color='blue')
plt.xlabel('x')
plt.ylabel('y')
plt.title(f'r^2 = {r2_score(y_test, predictions):.2}'
plt.show()
```



```
# add quadratic polynomial features
poly = PolynomialFeatures(degree=2)
poly_train = poly.fit_transform(X_train)
poly_test = poly.transform(X_test)

model = LinearRegression()
model.fit(poly_train, y_train)
y_pred = model.predict(poly_test)
r2score = r2_score(y_test, y_pred)

xs = np.linspace(-3, 3, len(X_test)).reshape(-1, 1)
Xpoly = poly.transform(xs)
predictions = model.predict(Xpoly)
```

```
    plt.plot(xs, predictions, color='red', linewidth=2)
    plt.scatter(X_test, y_test, color='blue')
    plt.xlabel("x")
    plt.ylabel("y")
    plt.title(f'r^2 = {r2score:.2}')
    plt.show()
```



Underfitting and Overfitting

In statistics, overfitting is "the production of an analysis that corresponds too closely or exactly to a particular set of data, and may therefore fail to fit additional data or predict future observations reliably". An overfitted model is a statistical model that contains more parameters than can be justified by the data.

Try removing the lesser performing features. The model is just memorizing the training data.

Underfitting occurs when a statistical model cannot adequately capture the underlying structure of the data. An under-fitted model is a model where some parameters or terms that would appear in a correctly specified model are missing. Under-fitting would occur, for example, when fitting a linear model to non-linear data. Such a model will tend to have poor predictive performance.

Try adding more data or more features.

https://en.wikipedia.org/wiki/Overfitting

Be familiar with the following picture:

https://scikit-learn.org/stable/_images/sphx_glr_plot_underfitting_overfitting_001.png

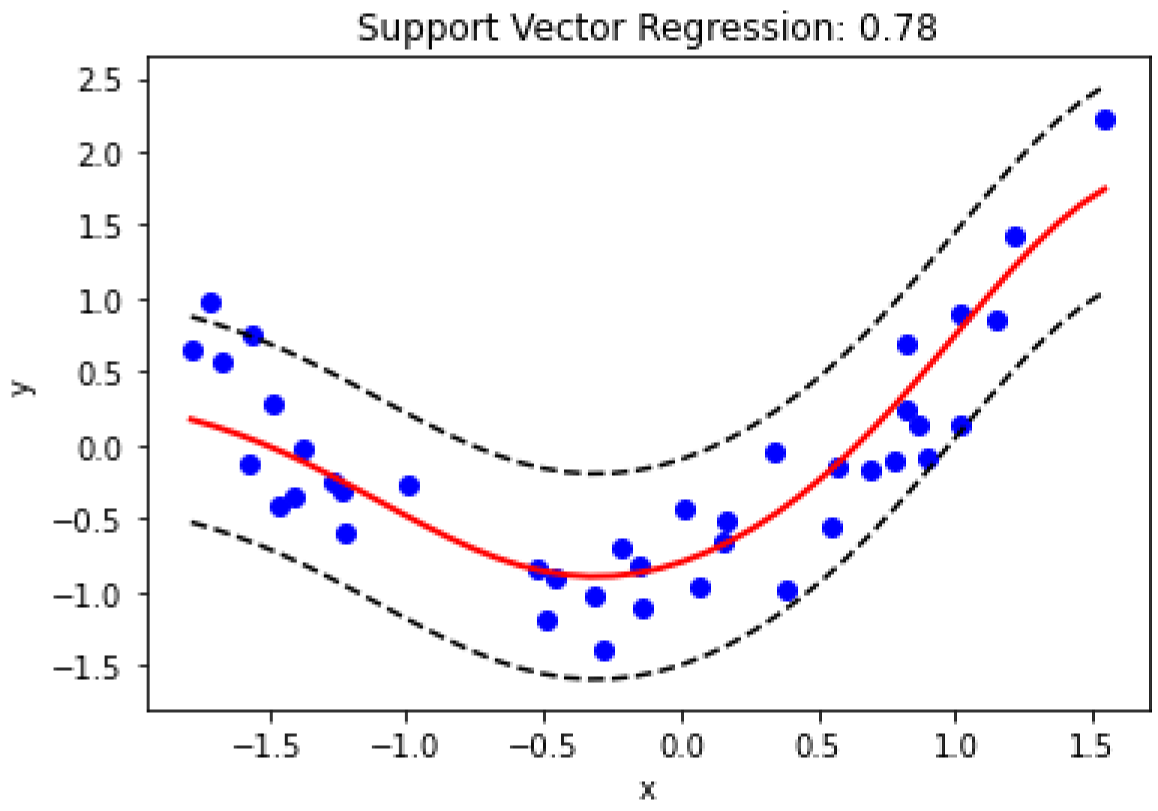
Support Vector Regression

- Similar to linear regression but $y = \beta_0 + \beta_1 x_1$ is refered to as a hyperplane
- SVR uses a margin that tries to capture as many of the plots as possible
- Margins are measured by epsilon which defines the epsilon insensitive tube
- Points in the tube are not considered errors

- Support vectors are points along the edges of the tube or decision surface
- More on this with Support Vector Classifiers

```
# scale data
from sklearn.preprocessing import StandardScaler

sc_X = StandardScaler()
sc_y = StandardScaler()
X_train = sc_X.fit_transform(X_train)
X_test = sc_X.transform(X_test)
y_train = sc_y.fit_transform(y_train)
y_test = sc_y.transform(y_test)
```

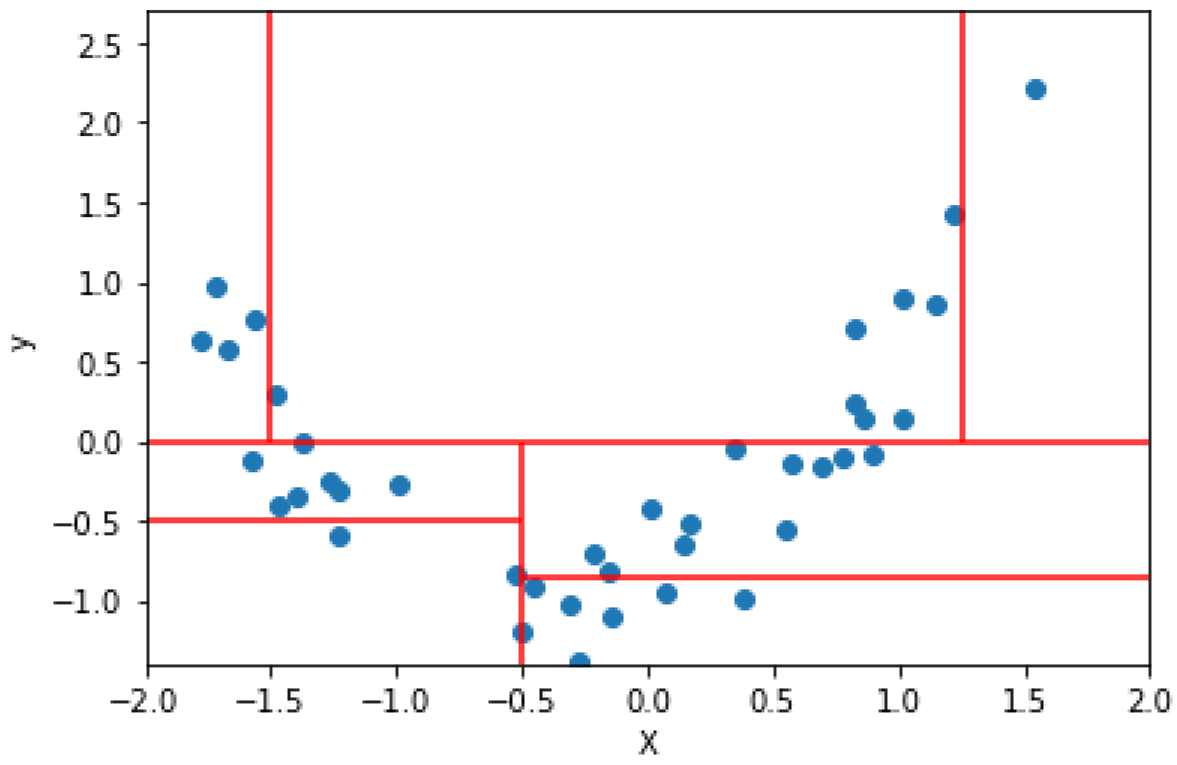
```
# SVR model
from sklearn.svm import SVR

model = SVR(kernel='rbf', epsilon=0.7)
model.fit(X_train, y_train.ravel())
predictions = model.predict(X_test)
r2score = r2_score(y_test, predictions)

xs = np.linspace(X_test.min(), X_test.max(), len(X_te
reg_line = model.predict(xs)

plt.scatter(X_test, y_test, color = 'blue')
plt.plot(xs, reg_line, 'r-', linewidth=2)
plt.plot(xs, reg_line + model.epsilon, 'k--')
plt.plot(xs, reg_line - model.epsilon, 'k--')
plt.title(f'Support Vector Regression: {r2score:.2}')
plt.xlabel('x')
plt.ylabel('y')
plt.show()
```



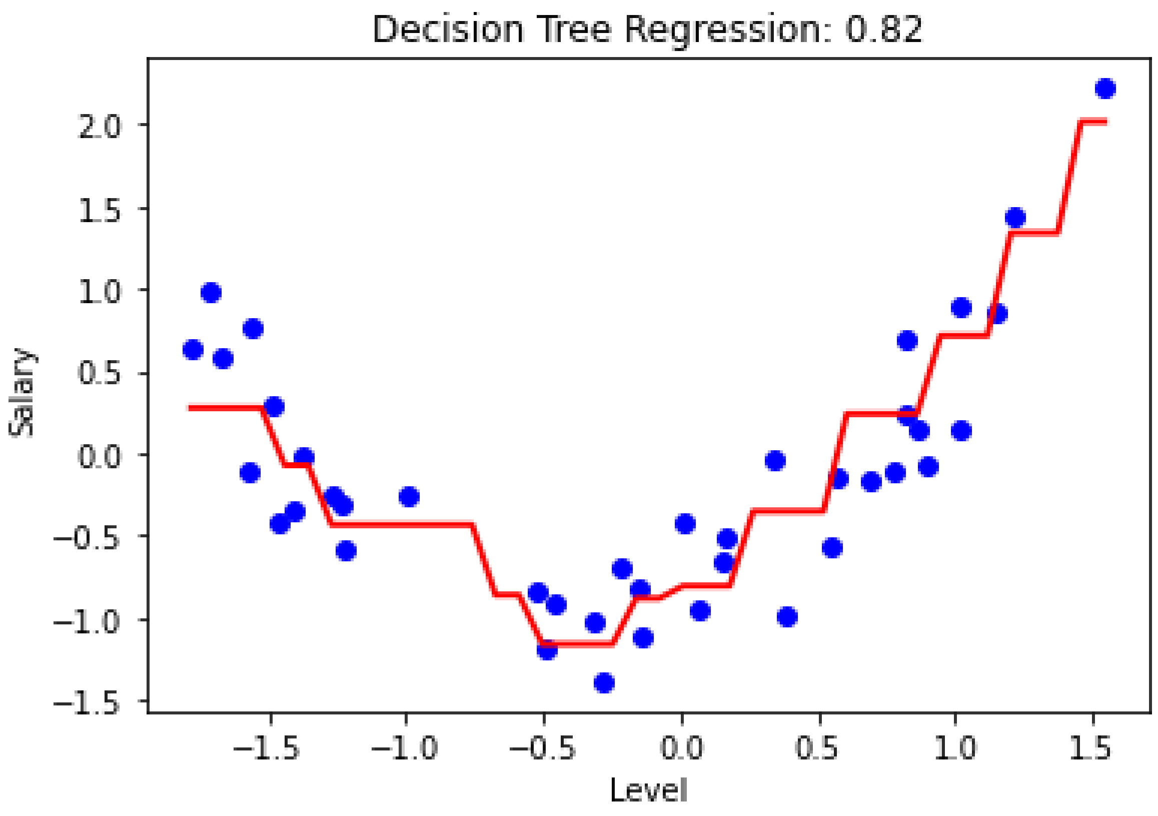
Decision Tree Regression

- CART: Classification and Regression Trees
- Decision trees use splits to partition data points
- When drawing our regression line, we first find out where the split is and then return the mean of that split. The most common labeled value is returned in classification
- The regression line is fit on the training data and is used to predict test data

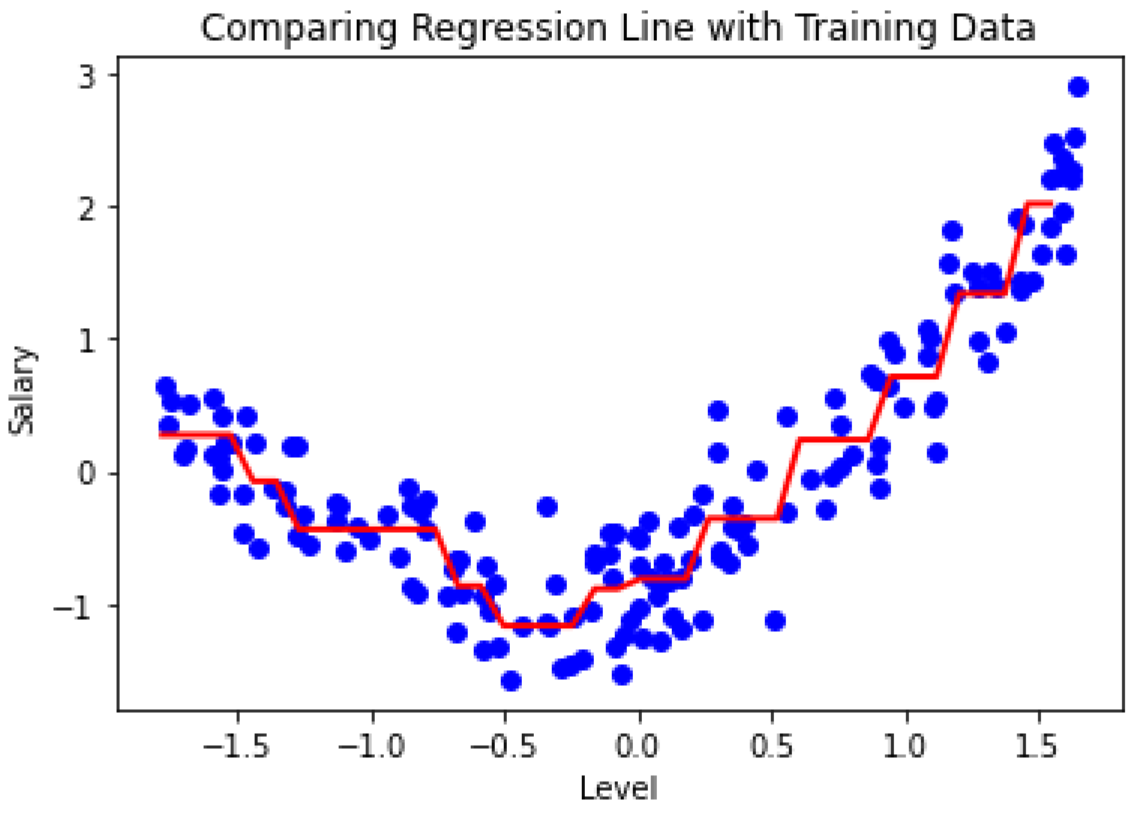
```
plt.scatter(X_test ,y_test)
plt.axhline(y=0, color='red')
plt.hlines(y=-0.5, xmin=-2.0, xmax=-0.5, color='red')
plt.hlines(y=-0.85, xmin=-0.5, xmax=2, color='red')
plt.vlines(x=-1.5, ymin=0, ymax=2.7, color='red')
plt.vlines(x=-0.5, ymin=-1.4, ymax=0, color='red')
plt.vlines(x=1.25, ymin=0, ymax=2.7, color='red')
plt.xlim(-2, 2)
plt.ylim(-1.4, 2.7)
plt.xlabel('x')
plt.ylabel('y')
plt.show()
```



```
from sklearn.tree import DecisionTreeRegressor

model = DecisionTreeRegressor(min_samples_leaf=10)
model.fit(X_train.reshape(-1, 1), y_train)
predictions = model.predict(X_test)
r2score = r2_score(y_test, predictions)

xs = np.linspace(X_test.min(), X_test.max(), len(X_te
reg_line = model.predict(xs)

plt.scatter(X_test, y_test, color = 'blue')
plt.plot(xs, reg_line, 'r-', linewidth=2)
plt.title(f'Decision Tree Regression: {r2score:.2}')
plt.xlabel('x')
plt.ylabel('y')
plt.show()
```



```
# Comparing Regression Line with Training Data
plt.scatter(X_train, y_train, color = 'blue')
plt.plot(xs, reg_line, 'r-', linewidth=2)
plt.title(f'Comparing Regression Line with Training D
plt.xlabel('x')
plt.ylabel('y')
plt.show()
```

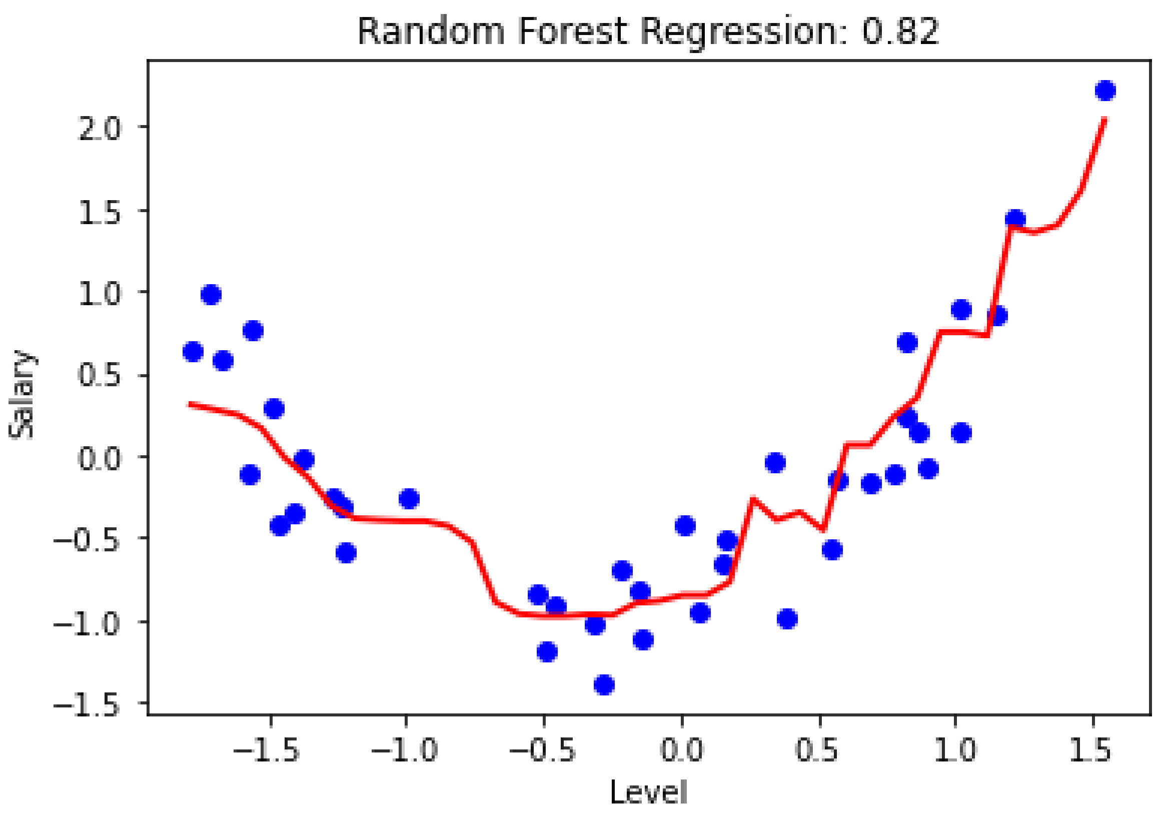


Random Forest Regression

- Random forest is a version of ensemble learning
- Ensemble learning combines multiple algorithms or multiple attempts of one algorithm to provide better predictive performance
- Random forests pick random data points from training set
- Builds a decision tree that best fits the random data points
- Repeat this multiple times (ex. 1000 times)
- Return average

```
from sklearn.ensemble import RandomForestRegressor
```

```
    model = RandomForestRegressor(n_estimators = 50, max
    model.fit(X_train.reshape(-1, 1), y_train.ravel())
    predictions = model.predict(X_test)
    r2score = r2_score(y_test, predictions)

    xs = np.linspace(X_test.min(), X_test.max(), len(X_t
    reg_line = model.predict(xs)

    plt.scatter(X_test, y_test, color = 'blue')
    plt.plot(xs, reg_line, 'r-', linewidth=2)
    plt.title(f'Random Forest Regression: {r2score:.2}')
    plt.xlabel('x')
    plt.ylabel('y')
    plt.show()
```


Random Forest Regression: 0.82
Salary
Level