Linear Regression Models

Neba Nfonsang

Linear Regression Models Outline

- Mathematical and statistical models
- Introduction to regression models
- Types of regression models
- Simple linear regression
- Gradient descent
- Regression in Python
- Error decomposition
- Sum of squares
- R-squared and adjusted R-squared
- Hypothesis testing
- ANOVA table
- Multicollinearity

Mathematical and Statistical Models

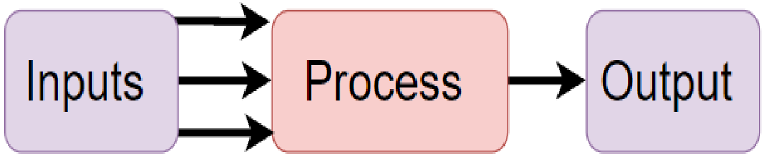
Regression Analysis

- Regression analysis is a statistical method for investigating the relationship between a dependent variable and independent variable(s).
- The earliest form of regression analysis was the least square regression method published by Legendre in 1805 and Gauss in 1809.

Mathematical Models

- The aim of regression analysis is to construct a mathematical model that describes or explains the relationship between variables (Seber & Lee, 2003)
- A mathematical model is a functional relationship between variables

- A mathematical model is a process or function that transforms input(s) to output



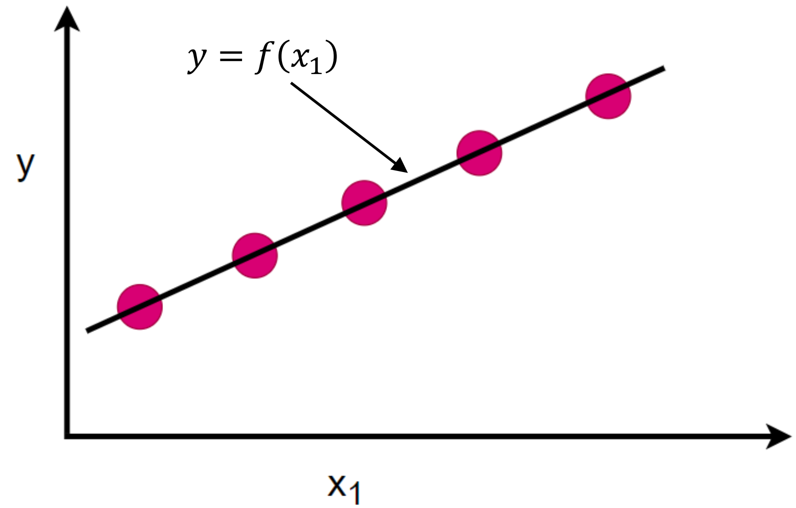
$$y = f(x_1, x_2, \cdots, x_k)$$

- y *is an output variable*
- $x_1, x_2, \cdots, x_k$ *represent input variables*
- $f(x_1, x_2, \cdots, x_k)$ is the functional relation that maps inputs and output

Mathematical Models (cont.)

- A mathematical model such as $y = f(x_1, x_2, \cdots, x_k)$ is a deterministic model because for any given input(s) the same output will always be predicted
- A deterministic model will always predict an exact output given some input(s)

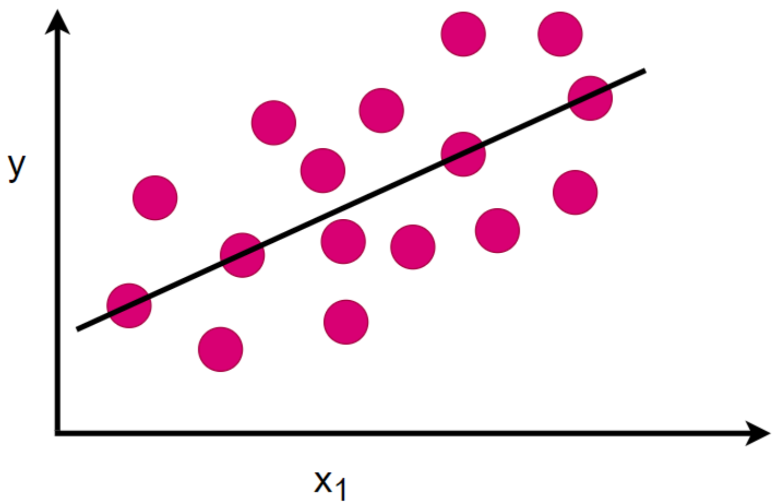
The graph below represents a mathematical model showing the relationship between x_1 and y



Statistical Models

- Real-world data is noisy. We may have the same input(s) for different cases, but the output may not always be the same.
- A mathematical linear function such as $y = f(x_1, x_2, \cdots, x_k)$ does not perfectly predict the output given some input(s) in real data.

The mathematical model only approximates the relationship between x_1 and y when the data is noisy.



Statistical Models (cont.)

- In real-world data, the output varies randomly even for cases with the same input.
- A statistical model is needed to capture the random variation in the output.
- A statistical model is obtained by adding random error term (ϵ) to the deterministic model.
  - $\boldsymbol{y = f(x_1, x_2, \cdots, x_k) + \epsilon}$
- A statistical model for a regression analysis has a mathematical (deterministic) part and statistical (stochastic or random) part.

Mathematical and Statistical Models

The End

Introduction to Regression Models

Regression Models

- Generally, a regression model is written as:
  - $\boldsymbol{y = f(x_1, x_2, \cdots, x_k) + \varepsilon}$
- This regression model represents the relationship between the output y and the inputs, $x_1, x_2, \cdots, and\ x_k$ in the population
- No statistical model is a "true" representation of reality; rather some are useful representations of reality
- This is because several models with different functional forms can reasonably represent reality

Regression Models (cont.)

- The regression model
  - $\boldsymbol{y = f(x_1, x_2, \cdots, x_k) + \varepsilon}$

Can also be written as:

  - $\boldsymbol{y = f(x) + \varepsilon}$
  - where x = $\{\boldsymbol{x_1, \cdots, x_k}\}$
- $f(x)$ is fixed but unknown function
- **For a linear regression**, $f(x)$ represents the population regression line or an underlying functional form of the model

Regression Models (cont.)

- For a linear regression,
  - $\boldsymbol{y} = \boldsymbol{f}(\boldsymbol{x}) + \boldsymbol{\varepsilon}$
- The random error ε is assumed to follow a normal distribution with mean (conditional expectation) zero and constant variance σ^2
- Since the random error is assumed to be normally distributed, y is also normally distributed with a mean E(y/x) = $f(x)$ and a variance of σ^2

Introduction to Regression Models

The End

Types of Regression Models

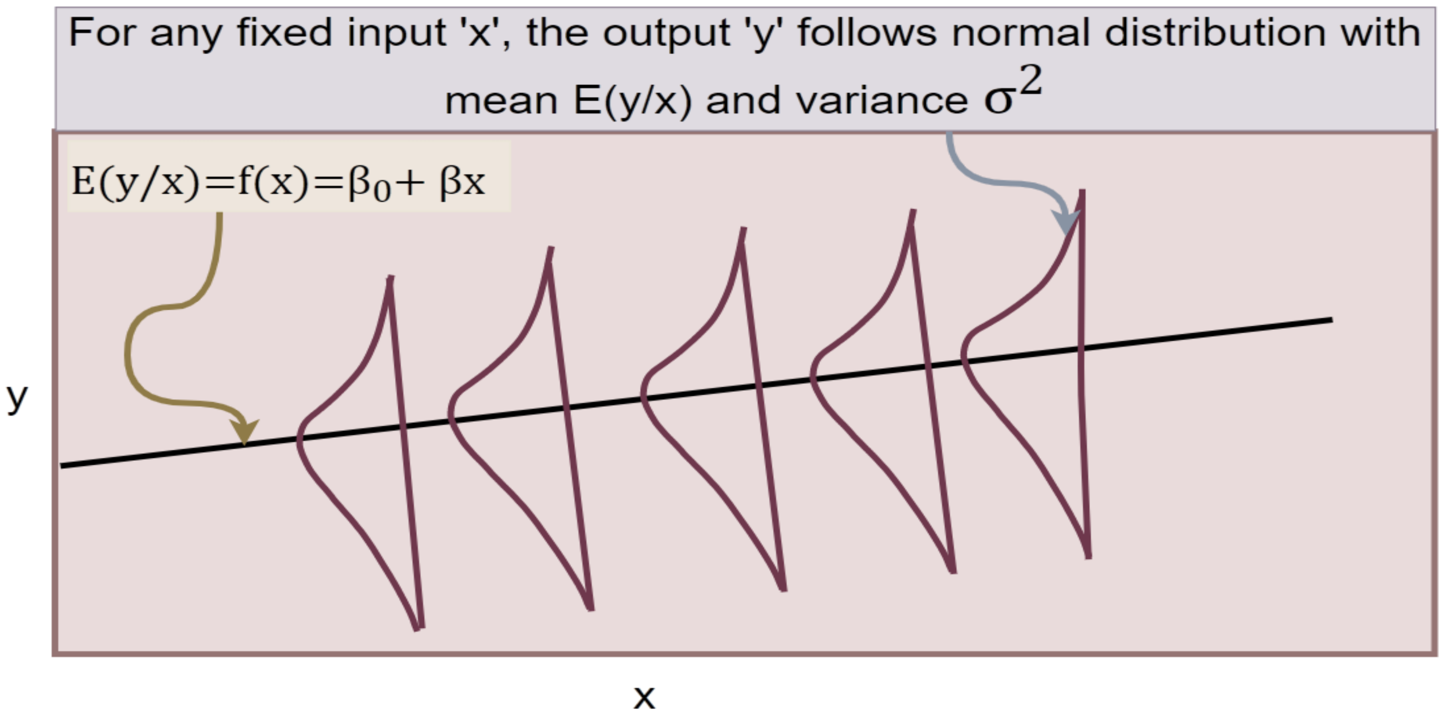
The Population Regression Function

- For a linear regression,
  - $\boldsymbol{y} = \boldsymbol{f}(\boldsymbol{x}) + \boldsymbol{\varepsilon}$
  - $f(x)$ is the population regression function
  - $f(x)$ is the hypothesized functional relationship between the y and x in the population, to be estimated
- The expectation of y at each x is written as:
  - $E(y/x) = f(x) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_k x_k$

The Population Regression Function (cont.)

- $E(y/x) = f(x) = \beta_0 + \ \beta_1 x_1 + \beta_2 x_2 + \cdots + \ \beta_k x_k$
- The population regression function $f(x)$ can be estimated using sample data and the estimated regression equation:
  - $\hat{y} = b_0 + \ b_1 x_1 + b_2 x_2 + \cdots + \ b_k x_k;\text{where}$
  - $\hat{y}$ is an unbiased estimate of $f(x)$
  - $b_0, \ \ b_1, \ b_2, \ldots, b_k$ are unbiased estimates of the population parameters $\beta_0, \ \beta_1, \ \beta_2, \ldots, \beta_k$

The Population Regression Function, Part III



$$y = \mathrm{E}(y/x) + e, \quad \text{where } e \sim N(0, \delta^2), hence, \quad y \sim N(E(y/x), \delta^2);$$

Types of Regression Models

The End

Simple Linear Regression

Linear Regression Model Assumptions

- **Linearity:** Each input variable(x) is linearly related to the output variable (y).
  - Use a scatter plot or scatter plot matrix to check this assumption.
- **Independence:** The random errors are independent.
  - This assumption is usually assumed met.
- **Normality:** The random errors are normally distributed with a mean of zero.
  - Use a histogram, Q-Q plot, or skewness to check this assumption.
- **Equal variance:** The random errors have equal variances at all values of x.
  - Check this assumption with residual plot.

Types of Regression Models

- Linear regression assumes that the model is linear in regression parameters
- There are three major types of regression analysis:
  - **Simple linear regression:** models the relationship between one input variable and one output variable
  - **Multiple linear regression:** models the relationship between several input variables and one output variable
  - **Nonlinear regression**: this is a regression model that is nonlinear in parameters and there is no possible transformation to make the parameters linear

A Simple Linear Regression Model

- **A simple linear regression model is written as:**

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

  - $y = the\ output\ variable$
  - $x_1 = the\ input\ variable$
  - $\beta_0 = y\ intercept\ (y\ when\ x = 0)$
  - $\beta_1 = coefficient\ of\ x\ (the\ effect\ of\ x\ on\ y)$
  - $\varepsilon = random\ error$

A Multiple Linear Regression Model

- A multiple regression model has one dependent variable (y) and several independent variables (x's)
- **The general form for a multiple regression model is:**
- $y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_k x_k + \varepsilon$

- $y = \mathit{the\ dependent\ variable}$
- $x's = \mathit{the\ independent\ variables}$
- $\beta_0, \beta_1, \ldots, \beta_k$ are the regression parameters (β_0 is the y-intercepts and $\beta_1, \ldots, \beta_k$ are the regression coeffients).
- k is the number of input variables.
- $\varepsilon = \mathit{random\ error, where}$ $E(\varepsilon) = 0; var(\varepsilon) = \sigma^2$

A Nonlinear Regression Model

- Nonlinear regression models capture nonlinear relationships, which are more complicated
- A nonlinear model assumes that the regression model is not linear in parameters
- Example of a nonlinear regression model is growth model
- **General form of a growth model:**

$$y = \frac{\alpha}{1 + e^{\beta t}} + \varepsilon$$

  - $y\ is\ growth\ rate, a\ function\ of\ time\ t$
  - $\alpha\ and\ \beta\ are\ model\ parameters$

Polynomial Regression Models

- Other forms of regression, such as polynomial regression, are still linear regression because the model is linear in parameters
- **The general form of a polynomial regression is:**

$$y = \beta_0 + \beta_1 \mathrm{x} + \beta_2 x^2 + \cdots + \beta_k x^k + \varepsilon$$

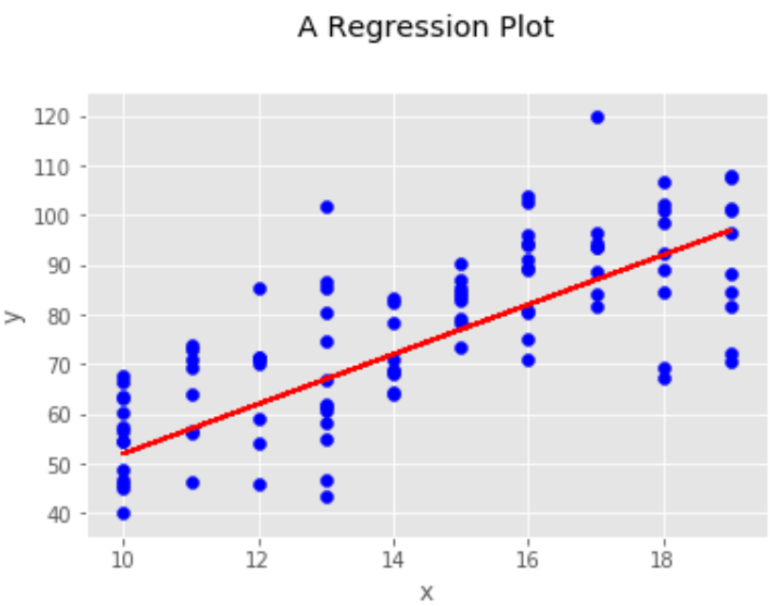
- k is the degree of polynomial
  - If k = 1; simple linear regression
  - If k = 2; quadratic polynomial
  - If k = 3; cubic polynomial
  - If k = 4; quartic polynomial, etc.
- y is the dependent variable, x is the independent variable
- $\beta_0, \beta_1, \ldots, \beta_k$ are the regression parameters

Goals of Regression Analysis

- Regression analysis is one of the most commonly used statistical methods in practice
- The purposes of regression analysis include:
  - To estimate the effect of an independent variable on a dependent variable
  - To estimate (predict) the value of the output variable y given a set of input values
  - To screen the x-variables to identify which ones are more important than others in explaining the variance in the y-variable

A Simple Linear Regression

- A simple linear regression model captures the average relationship between x and y.
- The red line is the regression line, which represents the average relationship between x and y.



A Simple Linear Regression (cont.)

- The “population” regression line or function is:
  - $f(x) = E(y/x) = \beta_0 + \beta_1 \mathrm{x}$
- So, the regression model is:
  - $y = \beta_0 + \beta_1 \mathrm{x} + \epsilon$

A Simple Linear Regression: Specification

- For a simple linear regression, there are n examples: $((x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n))$
- The model for the n examples can be written as:

$$y_i = \beta_0 + \beta_1 x_i + \epsilon_i$$

for $\mathrm{n} = 1, 2, \ldots, \mathrm{n}$.

- After the model is specified, the next task is to find the estimate for the parameter values.
- The model assumed that the x_i is measured "exactly" with no measurement error.

Simple Linear Regression: Estimation

- **Least square method** is the method used to find the estimates b_0 *and* b_1, by minimizing the sum of square errors (SSE).
- The prediction error or residual error for the ith example or case is given by: $\boldsymbol{e_i = y_i - \hat{y}_i}$

Where:

$y_i = \text{observed value of y}$

$\hat{y} = predicted\ value\ of\ y$

Sum of Square Error

$SSE = \sum_{i=1}^{n}(e_i^2)$

$SSE = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2$

$SSE = \sum_{i=1}^{n}[y_i - (b_0 + b_1\text{x})]^2$

$(b_0, b_1) = arg\ min_{(b_0,\ b_1)}\ SSE$

Simple Linear Regression: Estimation (cont.)

- To find the estimates, $b_0 \ and \ b_1$, differential calculus could be used to find the values of $b_0 \ and \ b_1$, where sum of square error $\sum_{i=1}^{n}[y_i \ -(b_0 + b_1\mathrm{x})]^2$ is minimum.

- So we need to solve:
  - $\partial \frac{SSE}{\partial b_0} = 0$ and $\partial \frac{SSE}{\partial b_1} = 0$
- Solving for $b_0 \ and \ b_1$in the above equations gives:

$$b_1 = \frac{cov(x,y)}{var(x)}$$

$$b_0 = \bar{y} - b_1\bar{x}$$

Simple Linear Regression: Estimation (cont.)

- Other methods can be used to estimate the parameters of a linear regression model including:
  - Gradient descent
  - Maximum likelihood

Simple Linear Regression

The End

Gradient Descent

Gradient Descent

- Gradient is an optimization algorithm that continually update the parameter values until the optimal parameter values are found that minimize the cost function.
- The gradient descent algorithm repeatedly updates each regression parameter until convergence, as follows:
- $\theta_j = \theta_j - \alpha \frac{\partial J(\theta)}{\partial \theta_j}$
- Where:
  - θ_j = the jth parameter (θ is the same as b_j in the regression equations).
  - α = learning rate
  - $J(\theta)$ = cost function, for example, MSE or mean square error.

Gradient Descent (cont.)

- Let’s represent the function that approximates y as:
- $h(x) = \theta_0 + \theta_1 x_1 + \cdots + \theta_k x_k$
- MSE would be:
- MSE = $\frac{1}{n}\sum_{i=1}^{n}(\mathrm{h}(x^{(i)}) - y^{(i)})^2$
- Where the pair ($x^{(i)}$, $y^{(i)}$) is the ith training example.
- The cost function, which is the MSE can be written as:
- $J(\theta) = \frac{1}{2n}\sum_{i=1}^{n}(\mathrm{h}(x^{(i)}) - y^{(i)})^2$
- (We used $\frac{1}{2}$ just for convenience to make the calculus look neat later).

Gradient Descent(cont.)

- Gradient descent starts with some initial parameter values, then updates parameters using:

$\theta_j = \theta_j - \alpha \frac{\partial J(\theta)}{\partial \theta_j}$ Where:

$J(\theta) = \frac{1}{2n} \sum_{i=1}^{n} (\mathrm{h}(x^{(i)}) - y^{(i)})^2$

- Lets find the partial derivative for a **single training example:**
- $\frac{\partial J(\theta)}{\partial \theta_j} = \frac{\partial}{\partial \theta_j} \frac{1}{2} (\mathrm{h}(x^{(i)}) - y^{(i)})^2$

$= 2 * \frac{1}{2} \mathrm{h}(x^{(i)}) - y^{(i)}) * \frac{\partial}{\partial \theta_j} (\mathrm{h}(x^{(i)}) - y^{(i)})$

but, $h(x) = \theta_0 + \theta_1 x_1 + \cdots + \theta_k x_k$

Gradient Descent (cont.)

$$\frac{\partial J(\theta)}{\partial \theta_j} = 2 * \frac{1}{2}\,(\mathrm{h}(x^{(i)}) - \ y^{(i)}) * \frac{\partial}{\partial \theta_j}\ (\mathrm{h}(x^{(i)}) - \ y^{(i)})$$

but, $h(x) \ = \theta_0 + \theta_1 x_1 + \cdots + \theta_k x_k$

$$\frac{\partial J(\theta)}{\partial \theta_j} = (\mathrm{h}(x^{(i)}) - \ y^{(i)}) * \frac{\partial}{\partial \theta_j}\ (\theta_0 + \theta_1 x_1^{(i)} + \cdots + \theta_k x_k^{(i)} - \ y^{(i)})$$

$\frac{\partial J(\theta)}{\partial \theta_j} = (\mathrm{h}(x^{(i)}) - \ y^{(i)}) * x_j^{(i)}$ for example $(x^{(i)}, y^{(i)})$.

So the update rule for a single example becomes

Gradient Descent (cont.)

- The general gradient descent update rule is :
- $\theta_j = \theta_j - \alpha \frac{\partial J(\theta)}{\partial \theta_j}$ but

$\frac{\partial J(\theta)}{\partial \theta_j} = \left(\mathrm{h}\left(x^{(i)}\right) - y^{(i)}\right) * x_j^{(i)}$ for a single example ($x^{(i)}$, $y^{(i)}$).

- So, the update rule for a single example becomes:
- $\theta_j = \theta_j - \alpha(\mathrm{h}(x^{(i)}) - y^{(i)})) * x_j^{(i)}$ or
- $\theta_j = \theta_j + \alpha\left(y^{(i)} - \mathrm{h}\left(x^{(i)}\right)\right) * x_j^{(i)}$ where $y^{(i)} - \mathrm{h}\left(x^{(i)}\right) = error$

Gradient Descent (cont.)

- There are two major types of gradient descent:
  - The batch gradient descent
  - The stochastic gradient descent
- The batch gradient descent algorithm uses all the training examples during each iteration to update the parameters.
- The stochastic gradient descent uses a single example to update the parameters during each iteration

Batch Gradient Descent

- $\theta_j = \theta_j + \alpha\left(y^{(i)} - \mathrm{h}\left(x^{(i)}\right)\right) * x_j^{(i)}$,the update is proportional to error.
- $\theta_j = \theta_j + \alpha(\text{error}) * x_j^{(i)}$. This rule is called the Least Mean Square or LMS rule. The rule is also called the Widrow-Hoff learning rule.
- Modify the rule to work for the entire training set as follows:

Repeat until converge {

$$\boldsymbol{\theta_j = \theta_j + \alpha * \frac{1}{n} * \sum_{i=1}^{n} [(y^{(i)} - \mathrm{h}(x^{(i)})) * x_j^{(i)}]}$$

} for every parameter j=0,1…k

Batch Gradient Descent Pseudo Code

```
b_0 = 0
b_1 = 0
b_2 = 0
α = 0.01
iterations = 1000
x_1 = np.array(x1_values)
x_2 = np.array(x2_values)
y = np.array(y_values)
For i in range(iterations):
        predicted_y = b_0 + b_1 * x_1 + b_2 * x_2 # a vector
        errors = y – predicted_y # a vector
        b_0 = b_0 + α * (1/n) * sum(errors)
        b_j = b_j + α * (1/n) * sum(error * x_j) # for all the j=1,…,k
print(b_0,…,b_k)# in this case, k=2
```

Stochastic Gradient Descent

- An alternative to the batch gradient descent is the **stochastic gradient descent.**
- The stochastic gradient descent is faster than the batch gradient descent because it uses only a single example in each iteration.
- The stochastic gradient descent algorithm is as follows:

For i in range(n):

$$\theta_j = \theta_j + \alpha(y^{(i)} - \mathrm{h}(x^{(i)}))*x_j^{(i)}$$

for every parameter j=0,1,…, k

Stochastic Gradient Descent Pseudo Code

```
b_0 = 0
b_1 = 0
b_2 = 0
α = 0.01
x_1 = np.array(x1_values)
x_2 = np.array(x2_values)
y = np.array(y_values)
For i in range(n): # n is number of examples
        predicted_y = b_0 + b_1 * x_1[i] + b_2 * x_2[i]
        error = y[i] - predicted_y
        b_0 = b_0 + α * (error)
        b_1 = b_1 + α * (error) * x_1[i]
        b_2 = b_2 + α * (error) * x_2[i] # for all the j=1,…,k
print(b_0, b_1,…, b_k)# in this case, k=2
```

Gradient Descent

The End

Regression in Python

Regression in Python

- Import packages and modules

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from sklearn import datasets
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import SGDRegressor
from sklearn import metrics
```

Regression in Python (cont.)

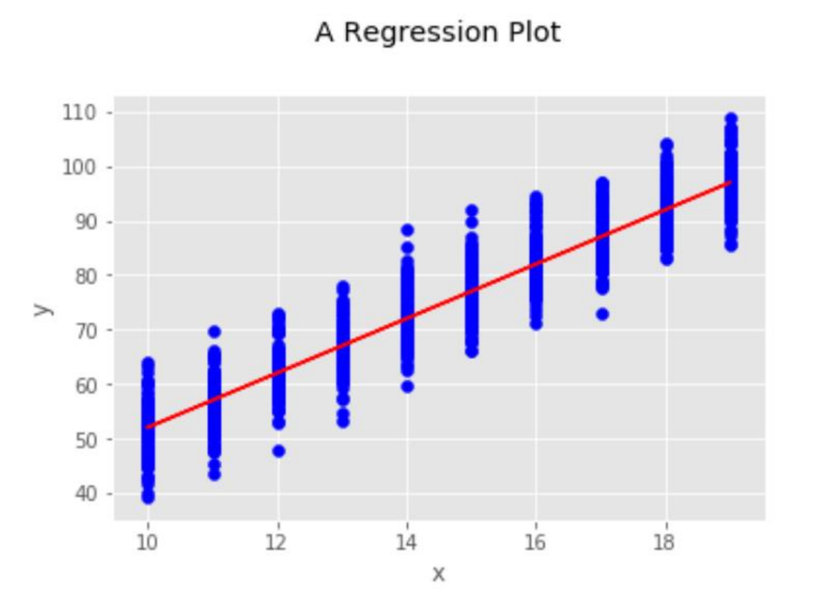
```
# let's create some data for a simple linear regression
np.random.seed(2020)
x = np.random.randint(10, 20, 1000)

# create some random noise
e = np.random.normal(loc=0, scale=5, size=1000)

# create the regression line
line = 2 + 5*x

# add noise to the regression line
y = line + e

plt.style.use("ggplot")
plt.plot(x, line, c="red")
plt.scatter(x, y, c="blue")
plt.xlabel("x")
plt.ylabel("y")
plt.title("A Regression Plot", y=1.1)
plt.show()
```



Regression in Python (cont.)

Batch Gradient Descent

```
np.random.seed(2020)
x = np.random.randint(10, 20, 1000)

# create some random noise
e = np.random.normal(loc=0, scale=5, size=1000)

# create the regression line
line = 2 + 5*x

# add noise to the regression line
y = line + e
```

```
b0 = 0
b1 = 0
learning_rate = 0.001
iterations = 1000
n = len(x)


for i in range(iterations):
    # compute the errors
    predicted_y = b0 + b1*x # a vector
    errors = y - predicted_y # a vector
    # update b0
    b0 = b0 + learning_rate*(1/n)*sum(errors)
    # update b1
    b1 = b1 + learning_rate*(1/n)*sum(errors*x)
    # update for for all bj; j=1,2,…,p
print(b0, b1)
```

```
0.359718543574295 5.121015201711109
```

Regression in Python (cont.)

Stochastic Gradient Descent

```
b0 = 0
b1 = 0
learning_rate = 0.001
n = len(x)

for i in range(n):
    # compute the errors
    predicted_y = b0 + b1*x[i] # a vector
    error = y[i] - predicted_y # a vector
    # update b0
    b0 = b0 + learning_rate*error
    # update b1
    b1 = b1 + learning_rate*error*x[i]
    # update for for all bj; j=1,2,…,p
print(b0, b1)
```

```
0.3703514954004253 5.158130352571659
```

Stochastic Gradient Descent in Scikit-learn

```
model= SGDRegressor(max_iter=1000, tol=1e-3)

model.fit(x.reshape(-1, 1), y)
model.intercept_, model.coef_
```

```
(array([0.46237163]), array([5.18613296]))
```

Regression in Python (cont.)

Linear Regression in Scikit-learn

```
ln = LinearRegression().fit(x.reshape(-1, 1), y)
ln.intercept_, ln.coef_
```

```
(0.843963216201l605, array([5.08884062]))
```

Linear Regression in statsmodels

```
from statsmodels.formula.api import ols
data = pd.DataFrame(zip(x, y), columns=["x", "y"])
model1 = ols("y ~ x", data = data).fit()
print(model1.summary2())
```

```
                  Results: Ordinary least squares
===================================================================
Model:              OLS              Adj. R-squared:     0.900
Dependent Variable: y                AIC:                6051.4873
Date:               2020-01-21 02:13 BIC:                6061.3028
No. Observations:   1000             Log-Likelihood:     -3023.7
Df Model:           1                F-statistic:        8951.
Df Residuals:       998              Prob (F-statistic): 0.00
R-squared:          0.900            Scale:              24.819
---------------------------------------------------------------------
             Coef.    Std.Err.     t      P>|t|     [0.025   0.975]
---------------------------------------------------------------------
Intercept    0.8440     0.7936   1.0635   0.2878   -0.7133   2.4012
x            5.0888     0.0538  94.6083   0.0000    4.9833   5.1944
---------------------------------------------------------------------
Omnibus:              2.785        Durbin-Watson:           1.917
Prob(Omnibus):        0.248        Jarque-Bera (JB):        2.764
Skew:                 0.096        Prob(JB):                0.251
Kurtosis:             2.828        Condition No.:           75
===================================================================
```

Regression in Python (cont.)

```
# make predictions in Scikit_learn
ln = LinearRegression().fit(x.reshape(-1, 1), y)
```

```
ln.predict(x.reshape(-1, 1))[0:50]
```

```
array([51.73236945, 92.44309444, 66.99889132, 82.2654132 , 66.99889132,
       66.99889132, 87.35425382, 92.44309444, 51.73236945, 51.73236945,
       92.44309444, 97.53193507, 66.99889132, 87.35425382, 61.9100507 ,
       66.99889132, 82.2654132 , 77.17657257, 51.73236945, 72.08773195,
       92.44309444, 82.2654132 , 72.08773195, 56.82121008, 56.82121008,
       77.17657257, 97.53193507, 77.17657257, 82.2654132 , 82.2654132 ,
       82.2654132 , 77.17657257, 72.08773195, 82.2654132 , 72.08773195,
       61.9100507 , 66.99889132, 72.08773195, 87.35425382, 56.82121008,
       72.08773195, 97.53193507, 66.99889132, 61.9100507 , 51.73236945,
       97.53193507, 56.82121008, 61.9100507 , 87.35425382, 56.82121008])
```

Regression in Python (cont.)

```
# model evaluation in Scikit-learn
y_pred = ln.predict(x.reshape(-1, 1))
print("MAE: ", metrics.mean_absolute_error(y, y_pred))
print("MSE: ", metrics.mean_squared_error(y, y_pred))
print("RMSE: ", np.sqrt(metrics.mean_squared_error(y, y_pred)))
print("R-Squared: ", ln.score(x.reshape(-1, 1), y))
```

```
MAE:  4.001156499778833
MSE:  24.7694302279653
RMSE:  4.97688961378455
R-Squared:  0.8996857009746718
```

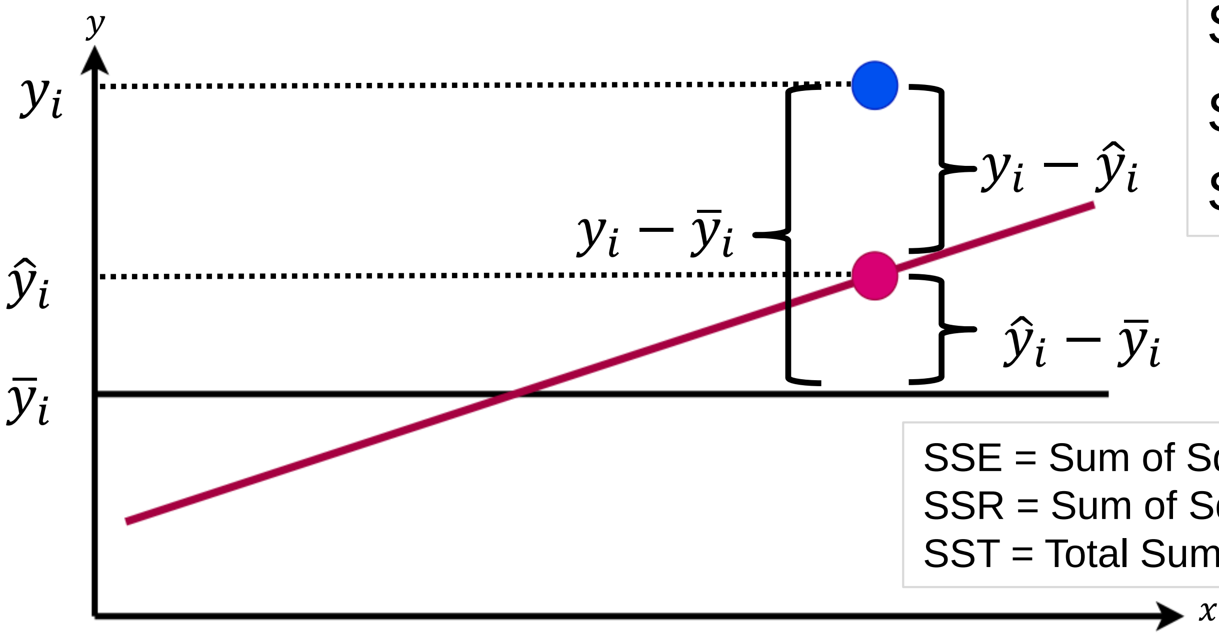
Regression in Python

The End

Error Decomposition, Sum of Squares, R-Squared, Adjusted R-Squared, Model Evaluation

Error Decomposition

$$SSE = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$
$$SSR = \sum_{i=1}^{n}(\hat{y}_i - \bar{y}_i)^2$$
$$SST = \sum_{1=1}^{n}(y_i - \bar{y}_i)^2$$
$$SST = SSR + SSE$$



SSE = Sum of Squares due to Error
SSR = Sum of Squares due to Regression
SST = Total Sum of Squares

Sum of Squares

- **Sum of Squares Error** (SSE):
  - a measure of the variation of the predicted output values from the actual output (prediction error)
- SSE = $\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$
- This is the unexplained variation in y.

- Sum of Square Due Regression
  - a measure of the variation of the predicted output values from the mean of the output data.
- SSR = $\sum_{i=1}^{n}(\hat{y}_i - \bar{y}_i)^2$
- This is the explained variation in y.

Sum of Squares (cont.)

- Total Sum of Squares (SST):
  - Measures the extend to which the actual output values vary from the mean of the output data.
- SST = $\sum_{1=1}^{n}(y_i - \bar{y}_i)^2$
- SST = SSR + SSE

- SST is a the total variation in y (both explained and unexplained).

Coefficient of Determination (R-Squared)

- The coefficient of determination (r-squared) is proportion of variance explained in the output variable by the input variable(s)

- $r - squared = \frac{explained\ variation\ in\ y}{total\ variation\ in\ y}$
- $r - squared = \frac{SSR}{SST}$
- $r - squared = \frac{SST - SSE}{SST}$
- $r - squared = 1 - \frac{SSE}{SST}$
- The value of r-squared is used to measure how well the estimated regression model fits the data.

Adjusted R-Squared

- When a variable is added to the model, r-squared becomes larger even if the added input variable is not statistically significant.
- Adjusted r-squared (R_a^2) is preferred for models with an added variable:
- $R_a^2 = 1 - (1 - R^2)\frac{n-1}{n-k-1}$
- Adjusted r-squared is used to adjust for the number of variables to avoid over estimating the amount of variance explained in the output by the input variables.
- Adjusted r-squared is useful for comparing nested models.

Model Evaluation

- The performance of the regression model can be evaluated using:
  - R-Squared = $1 - \frac{SSE}{SST}$
  - MSE = $\frac{SSE}{n-k-1}$
  - RMSE = $\sqrt{MSE}$
- MSE is mean square error: this is an unbiased estimate of the variance, σ^2.
- RMSE is root mean square error.

Error Decomposition, Sum of Squares, R-Squared, Adjusted R-Squared, Model Evaluation

The End

Hypothesis Test for Parameters

Hypothesis Test for Parameters: T-Test

- T-test for the population regression parameters can be used for the following:
  - Investigate whether there is a significant relationship between an input variable and the output variable.
  - Feature selection: non-significant features are removed.
- The hypotheses are:
  - H_0: $\beta_i = 0$
  - H_1: $\beta_i \neq 0$
  - Where β_i is the population parameter associated to x_i
- T-test is used to test for **individual significance.**

Hypothesis Test for Parameters: T-Test (cont.)

- An input variable having a non-significant relationship with y can be removed if:
  - There is still sufficient model performance or fit after removing the input variable.
  - There is no theory or experience indicating that the input variable has a significant relationship with the output variable.

$t_{STAT} = \frac{b_i}{s_{b_i}}$; where

- b_i is the point estimate for β_i
- s_{b_i} is an estimated standard deviation or standard error of b_i.

Hypothesis Test for Parameters: T-Test (cont.)

- An F-test can be used to determine whether there is a significant relationship between all the input variables and the output variable.
- The hypotheses for F-test are:
  - H_0: $\beta_1 = \beta_2 = \ldots = \beta_k = 0$
  - H_1: one or more parameters is not equal to zero (multiple regression).
- $F_{STAT} = \frac{MSR}{MSE}$; where
  - $MSR = \frac{SSR}{k}$
  - $MSE = \frac{SSE}{n-k-1}$
  - k is the number of input variables.
  - n is the number of cases or examples.

Hypothesis Test for Parameters

The End

ANOVA Table and Multicollinearity

The ANOVA Table for Linear Regression

Source of Variation	Sum of Squares	Degree of Freedom	Mean Square	F	P-value
Regression	SSR = $\sum_{i=1}^{n}(\hat{y}_i - \bar{y}_i)^2$	k	$MSR = \frac{SSR}{k}$	$F_{STAT} = \frac{MSR}{MSE}$	
Error	SSE = $\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$	$n - k - 1$	$MSE = \frac{SSE}{n-k-1}$		
Total	SST = $\sum_{1=1}^{n}(y_i - \bar{y}_i)^2$	$n - 1$			

F_{STAT} follows an F-distribution with k and n-k-1 degrees of freedom (in the numerator and denominator respectively).

Multicollinearity

- Multicollinearity is the correlation between input variables.
- Pairwise correlation or a correlation matrix can be used to check for multicollinearity.
- If multicollinearity between two input variables is too high, above 9.0, remove one of the variables.
- As such, checking for multicollinearity can be helpful for feature selection.

ANOVA Table and Multicollinearity

The End