Applied Statistical Methods

Simple Linear Regression Models

Xuemao Zhang
East Stroudsburg University

March 17, 2023

Outline

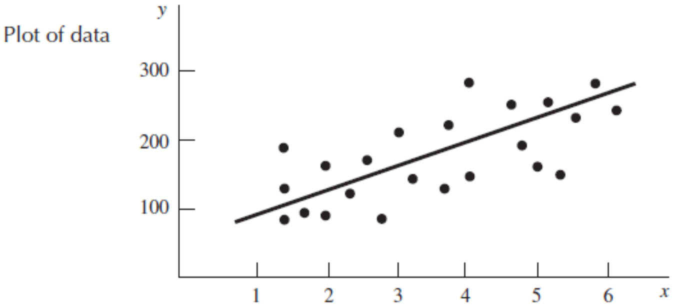
- Introduction to Simple Linear Regression Models
- Probabilistic Model
- Point Estimations
- Statistical Inferences
  - Hypothesis testing
  - Interval estimations
  - Overall accuracy of the model
  - ANOVA (Analysis of Variance)
  - An example
- Gradient Descent Algorithm

Linear Regression - Introduction

When two variables are measured (not always but usually on a single experimental unit), the resulting data are called bivariate data (or Paired data). When both of the variables (X, Y) are quantitative, call the variable X - the *independent variable*, and Y - the *dependent variable*. A random sample is of the form

$$(X_1, Y_1), (X_2, Y_2), \ldots, (X_n, Y_n).$$

Scatter plot can be used to check the relationship between X and Y. A typical scatter plot is like



Linear Regression - Introduction

Assume that visual examination of the scatter plot confirms that the points approximate a straight-line pattern

$$y = \beta_0 + \beta_1 x.$$

This model is called a **deterministic** mathematical model because it does not allow for any error in predicting y as a function of x.

However, the bivariate measurements that we observe do not generally fall exactly on a straight line, we choose to use a **probabilistic** model: for any fixed value of x,

$$E(Y|X = x) = \beta_0 + \beta_1 x.$$

or, equivalently,

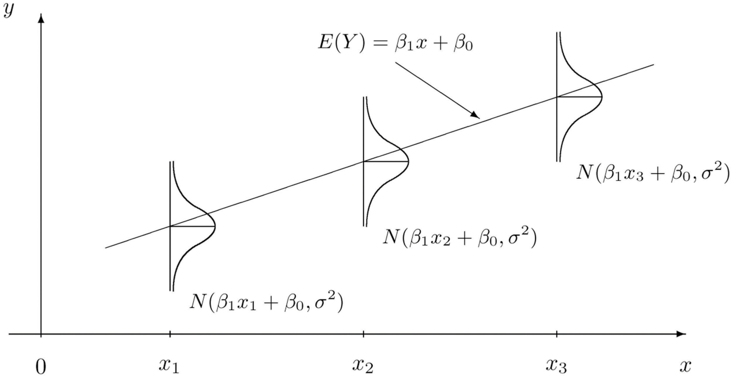
$$Y|_{X=x} = \beta_0 + \beta_1 x + \varepsilon,$$

where ε is a random variable possessing a specified probability distribution with mean 0.

For example, assume that ε's are independent normal random variables with mean 0 and common variance σ^2.

Linear Regression - Introduction

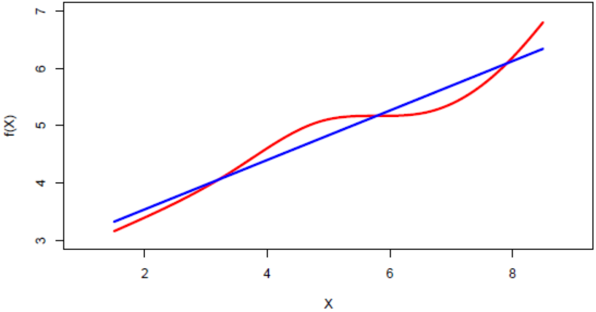
Simple Linear Regression Model Introductory Statistics (Shafer and Zhang), UC Davis Stat Wiki



We estimate the population parameters β_0 and β_1 using sample information.

Linear Regression - Introduction

- Linear regression is a simple approach to supervised learning. It assumes that the dependence of Y on the predictors $X_1, X_2, \ldots, X_p$ is linear.
- True regression functions are never linear!



- Although it may seem overly simplistic, linear regression is extremely useful both conceptually and practically.

Linear Regression - Introduction

- Consider the advertising data shown.
- Questions we might ask:
  - Is there a relationship between advertising budget and sales?
  - How strong is the relationship between advertising budget and sales?
  - Which media contribute to sales?
  - How accurately can we predict future sales?
  - Is the relationship linear?
  - Is there synergy among the advertising media?

SLR Probabilistic Model

Consider one dependent variable Y and 1 independent variables, X. The data will be in the form of

$$(x_1, y_1), \ldots, (x_n, y_n).$$

Or

Y	X
y_1	x_1
$\vdots$	$\vdots$
y_n	x_n

Our objective is to use the information provided by the X to predict the value of Y.

SLR Probabilistic Model

Definition. A simple linear statistical model relating a random response Y to an independent variables X is of the form

$$Y|_{X=x_i} = \beta_0 + \beta_1 x_i + \varepsilon_i,$$

where β_0 and β_1 are unknown parameters, ε is a random variable, and the variables $x_1, x_2, \ldots, x_n$ assume known values. We will assume that $E(\varepsilon_i) = 0, i = 1, \ldots, n$, and hence that

$$E(Y|_{X=x_i}) = \beta_0 + \beta_1 x_i, i = 1, \ldots, n.$$

SLR Point Estimations

- Method of Least Squares for simple linear regression models: The line of means

$$E(Y_i) = \beta_0 + \beta_1 x_i, \quad i = 1, 2, \ldots, n$$

describes average value of Y_i for any fixed value of $x_i, i = 1, 2, \ldots, n$.

Let $\widehat{\beta}_0$ and $\widehat{\beta}_1$ be the estimator of β_0 and β_1 respectively. Then

$$\widehat{Y} = \widehat{\beta}_0 + \widehat{\beta}_1 x$$

is clearly an estimator of $E(Y)$ when $X = x$. Let

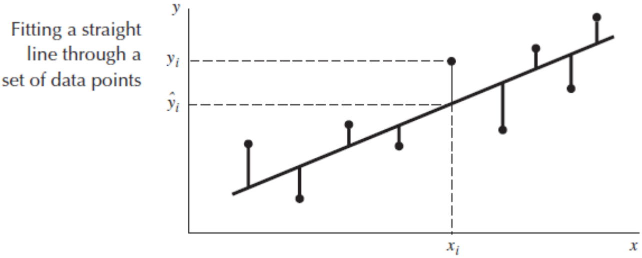
$$\widehat{y}_i = \widehat{\beta}_0 + \widehat{\beta}_1 x_i$$

be the predicted value of the ith y value (when $X = x_i$).

We choose our estimates $\widehat{\beta}_0$ and $\widehat{\beta}_1$ to estimate β_0 and β_1 so that the vertical distances of the points y_i from the line, are minimized. That is, $\widehat{\beta}_0$ and $\widehat{\beta}_1$ are chosen to minimize the sum of squares of deviations

$$SSE = \sum_{i=1}^{n} (y_i - \widehat{y}_i)^2 = \sum_{i=1}^{n} [y_i - (\widehat{\beta}_0 + \widehat{\beta}_1 x_i)]^2.$$

SLR Point Estimations



Least-Squares Estimators for the Simple Linear Regression Model.

1. $\widehat{\beta}_1 = \dfrac{S_{xy}}{S_{xx}}$ where $S_{xy} = \sum_{i=1}^{n}(x_i - \overline{x})(y_i - \overline{y})$ and $S_{xx} = \sum_{i=1}^{n}(x_i - \overline{x})^2$.
2. $\widehat{\beta}_0 = \overline{y} - \widehat{\beta}_1 \overline{x}$.

SLR Point Estimations

In the SLR model

$$Y_i|_{X=x_i} = \beta_0 + \beta_1 x_i + \varepsilon_i,$$

where $E(\varepsilon_i) = 0$. Furthermore, assume $V(\varepsilon_i) = \sigma^2$, $i = 1, 2, \ldots, n$.

- Properties of the Least-Squares Estimators:

Summary of properties:

1. $\widehat{\beta}_0$ and $\widehat{\beta}_1$ are unbiased. That is, $E(\widehat{\beta}_i) = \beta_i$, for $i = 0, 1$.
2. $V(\widehat{\beta}_0) = c_{00}\sigma^2$, where $c_{00} = \sum x_i^2/(nS_{xx})$.
3. $V(\widehat{\beta}_1) = c_{11}\sigma^2$, where $c_{11} = 1/S_{xx}$.
4. $Cov(\widehat{\beta}_0, \widehat{\beta}_1) = c_{01}\sigma^2$, where $c_{01} = -\overline{x}/S_{xx}$.
5. An unbiased estimator of σ^2 is $MSE = SSE/(n-2)$, and $SSE = S_{yy} - \widehat{\beta}_1 S_{xy}$.

SLR Point Estimations

If, in addition, the ε_i , for $i = 1, 2, \ldots, n$ are normal $N(0, \sigma^2)$,

6. Both $\widehat{\beta}_0$ and $\widehat{\beta}_1$ are normally distributed.
7. The random variable $\dfrac{(n-2)MSE}{\sigma^2}$ has a χ^2 distribution with $n-2$ df.
8. $Cov(\overline{Y}, \widehat{\beta}_1) = 0$.
9. The statistic S^2 is independent of both $\widehat{\beta}_0$ and $\widehat{\beta}_1$.

SLR Hypothesis testing

- Let $S = \sqrt{MSE}$. Then under $H_0 : \beta_i = \beta_{i0}, i = 0, 1$,

$$T = \frac{\widehat{\beta}_i - \beta_{i0}}{S\sqrt{c_{ii}}}, \quad i = 0, 1$$

possess a Student's t distribution with $n - 2$ df, where $c_{00} = \sum x_i^2/(nS_{xx})$ and $c_{11} = 1/S_{xx}$.

- Note that the testing of β_1 is actually to test

$$\begin{aligned} H_0 &: \text{There is no relationship between } X \text{ and } Y \text{ versus} \\ H_a &: \text{There is some relationship between } X \text{ and } Y \end{aligned}$$

SLR Hypothesis testing

- Test of Hypothesis for β_i:

$$H_0 : \beta_i = \beta_{i0}$$

$$H_a : \begin{cases} \beta_i > \beta_{i0}, & \text{(upper-tail alternative);} \\ \beta_i < \beta_{i0}, & \text{(lower-tail alternative);} \\ \beta_i \neq \beta_{i0}, & \text{(two-tailed alternative).} \end{cases}$$

$$\text{Test statistic:} T = \frac{\widehat{\beta}_i - \beta_{i0}}{S\sqrt{c_{ii}}}$$

$$\text{Rejection region:} \begin{cases} t > t_\alpha, & \text{(upper-tail rejection region);} \\ t < -t_\alpha, & \text{(lower-tail rejection region);} \\ |t| > t_{\alpha/2}, & \text{(two-tailed rejection region).} \end{cases}$$

where

$$c_{00} = \sum x_i^2/(nS_{xx}), c_{11} = 1/S_{xx}.$$

Notice that the t-distribution is based on (n-2) df.

SLR Interval Estimations

- A $100(1-\alpha)\%$ Confidence Interval for β_i

$$\boxed{\begin{array}{l} \qquad\qquad \widehat{\beta}_i \pm t_{\alpha/2,n-2} S\sqrt{c_{ii}} \\ \text{where} \\ \qquad c_{00} = \sum x_i^2/(nS_{xx}),\, c_{11} = 1/S_{xx}. \end{array}}$$

- One useful application of the hypothesis-testing and confidence interval techniques just presented is to the problem of estimating $E(Y)$, the mean value of Y , for a fixed value of the independent variable $x = x^*$.

$$\boxed{\begin{array}{c} \text{A } 100(1-\alpha)\% \text{ Confidence Interval for } E(Y) = \beta_0 + \beta_1 x^* : \\ \widehat{\beta}_0 + \widehat{\beta}_1 x^* \pm t_{\alpha/2,n-2} S\sqrt{\frac{1}{n} + \frac{(x^*-\overline{x})^2}{S_{xx}}} \end{array}}$$

SLR Interval Estimations

- Predicting a particular value of Y:

Let $x = x^*$ be a fixed value of the independent variable. Instead of estimating the mean Y value at $x = x^*$, we wish to predict the particular (individual) response Y that we will observe if the experiment is run at some time in the future (such as next Monday), denoted by Y^*. Then

$$Y^* = \beta_0 + \beta_1 x^* + \varepsilon.$$

It is natural to estimate Y^* by $\widehat{Y^*} = \widehat{\beta}_0 + \widehat{\beta}_1 x^*$.

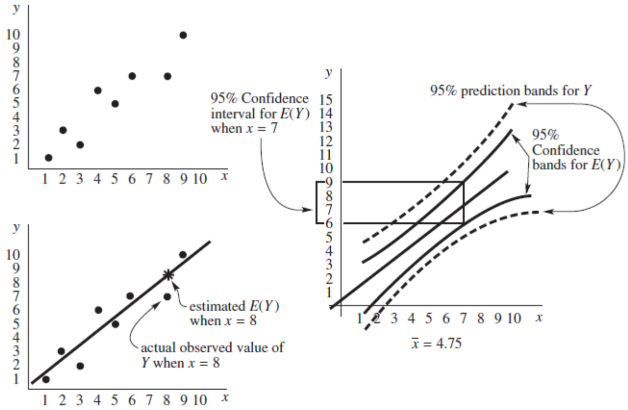
- A $100(1-\alpha)\%$ Prediction Confidence Interval for Y when $x = x^*$

$$\boxed{\widehat{\beta}_0 + \widehat{\beta}_1 x^* \pm t_{\alpha/2,n-2} S \sqrt{1 + \frac{1}{n} + \frac{(x^* - \overline{x})^2}{S_{xx}}}.}$$

SLR Interval Estimations

Remark. Prediction intervals for the actual value of Y are longer than confidence intervals for $E(Y)$ if both confidence levels are the same and both are determined for the same value of $x = x^*$.

Some hypothetical data and associated confidence and prediction bands



Overall Accuracy of the Model: Coefficient of Determination

The total variation is measured by the Total Sum of Squares (SS_{total}), a measure of the variation in the response values ignoring the regression model:

$$SS_{total} = \sum_{i=1}^{n} (y_i - \overline{y})^2.$$

Now

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

is a measure of the variation remaining in the response values after predicting them using the fitted regression equation and

$$SSR = \sum_{i=1}^{n} (\widehat{y}_i - \overline{y})^2$$

is a measure of the variation explained by using x in the SLR model.

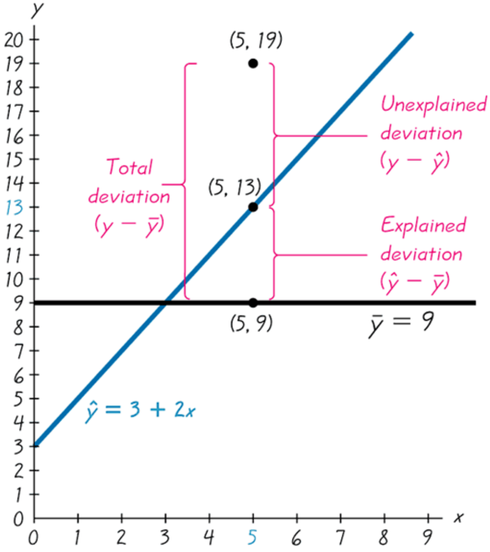
Overall Accuracy of the Model: Coefficient of Determination

The coefficient of determination, denoted by R^2, is the proportion of the variation in y that is explained by the regression line

$$R^2 = \frac{\text{explained variation}}{\text{total variation}}.$$

That is, it is a measure of: How much of the variation in the response is "explained" by the regression (the linear relationship between X and Y).

Overall Accuracy of the Model: Coefficient of Determination



Example:

- There is sufficient evidence of a linear correlation.
- The equation of the line is

  $\hat{y} = 3 + 2x$
- The sample mean of the y-values is 9.
- One of the pairs of sample data is $x = 5$ and $y = 19$.
- The point **(5,13)** is on the fitted regression line.

Overall Accuracy of the Model: Coefficient of Determination

It can be shown that

$$\sum_{i=1}^{n}(y_i - \overline{y})^2 = \sum_{i=1}^{n}(\widehat{y}_i - \overline{y})^2 + \sum_{i=1}^{n}(y_i - \widehat{y}_i)^2.$$

That is,

$$SS_{total} = SSR + SSE.$$

Therefore,

$$R^2 = \frac{SSR}{SS_{total}} = 1 - \frac{SSE}{SS_{total}}.$$

Analysis of Variance

The procedure of Analysis of Variance for SLR models can be summarized in the following table.

Source	df	SS	MS	F
Regression	1	SSR	$MSR = SSR/1$	MSR/MSE
Error	n-2	SSE	$MSE = SSE/(n-2)$	
Total	n-1	SS_{total}		

Note. The F-test for $H_0 : \beta_1 = 0$ versus $H_a : \beta_1 \neq 0$ is exactly equivalent to the t-test, with

$$t^2 = F.$$

And the F-test statistic has an F distribution under H_0 with $df_1 = 1, df_2 = n - 2$.

Simple Linear Regression Models

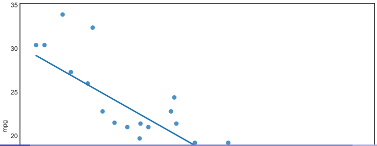
- Scatter plot with SLR model fit
  - https://seaborn.pydata.org/generated/seaborn.regplot.html

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import statsmodels.api as sm

plt.style.use('seaborn-white')
```

```
## <string>:1: MatplotlibDeprecationWarning: The seaborn styles shipped by
```

```
mtcars = sm.datasets.get_rdataset('mtcars',"datasets")
mtcars_data = pd.DataFrame(mtcars.data)
sns.regplot(data=mtcars_data,x='wt', y='mpg',ci=None)
plt.show()
```



Simple Linear Regression Models

- R code

```
library(ggplot2)
mtcars$cyl <- as.factor(mtcars$cyl)
ggplot(mtcars, aes(x=wt, y=mpg)) +
  geom_point(size=2);
# Add the regression line:
ggplot(mtcars, aes(x=wt, y=mpg)) +
  geom_point()+
  geom_smooth(method=lm, se=FALSE);
# Add confidence band:
ggplot(mtcars, aes(x=wt, y=mpg)) +
  geom_point()+
  geom_smooth(method=lm);
# Scatter plot with fitted regression line and equation:
library(ggpubr);
ggplot(mtcars, aes(x=wt, y=mpg)) +
  geom_point()+
  geom_smooth(method=lm)+
  stat_regline_equation(formula=y ~ x, label.x = 3,label.y = 32);
# label.x and label.y specifies the absolute positioning of the label
```

Simple Linear Regression Models

- There are two main ways to perform linear regression in Python — with `statsmodels` and `scikit-learn`.
  - `statsmodels` is "a Python module that provides classes and functions for the estimation of many different statistical models, as well as for conducting statistical tests, and statistical data exploration." (from the documentation)
  - See https://www.statsmodels.org/stable/generated/statsmodels.regression.linear_model.OLS.html

```
#import statsmodels.formula.api as smf
from statsmodels.formula.api import ols
model1=ols('mpg~wt', mtcars_data).fit()
print(model1.params)
```

```
## Intercept    37.285126
## wt           -5.344472
## dtype: float64
```

Simple Linear Regression Models

- Model fit summary

```
print(model1.summary())
```

```
##                             OLS Regression Results
## ==============================================================================
## Dep. Variable:                    mpg   R-squared:                       0.753
## Model:                            OLS   Adj. R-squared:                  0.745
## Method:                 Least Squares   F-statistic:                     91.38
## Date:                Thu, 16 Mar 2023   Prob (F-statistic):           1.29e-10
## Time:                        09:09:59   Log-Likelihood:                -80.015
## No. Observations:                  32   AIC:                             164.0
## Df Residuals:                      30   BIC:                             167.0
## Df Model:                           1
## Covariance Type:            nonrobust
## ==============================================================================
##                  coef    std err          t      P>|t|      [0.025      0.975]
## ------------------------------------------------------------------------------
## Intercept     37.2851      1.878     19.858      0.000      33.450      41.120
## wt            -5.3445      0.559     -9.559      0.000      -6.486      -4.203
## ==============================================================================
## Omnibus:                        2.988   Durbin-Watson:                   1.252
## Prob(Omnibus):                  0.225   Jarque-Bera (JB):                2.399
## Skew:                           0.668   Prob(JB):                        0.301
## Kurtosis:                       2.877   Cond. No.                         12.7
## ==============================================================================
##
## Notes:
## [1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

Simple Linear Regression Models

```
print(model1.summary().tables[1])
```

```
## ==================================================================================
##                   coef    std err          t      P>|t|      [0.025
## ----------------------------------------------------------------------------------
## Intercept      37.2851      1.878     19.858      0.000      33.450
## wt             -5.3445      0.559     -9.559      0.000      -6.486
## ==================================================================================
```

- Confidence interval of the parameters. For example, to get the 95% confidence intervals of the parameters,

```
model1.conf_int(alpha=0.05)
```

```
##                    0          1
## Intercept  33.450500  41.119753
## wt         -6.486308  -4.202635
```

Simple Linear Regression Models

- Fitted values

```
model1.predict()
```

```
## array([23.28261065, 21.9197704 , 24.88595212, 20.10265006, 18.90014396,
##        18.79325453, 18.20536265, 20.23626185, 20.45004071, 18.90014396,
##        18.90014396, 15.53312687, 17.3502472 , 17.08302362,  9.22665041,
##         8.29671236,  8.71892561, 25.52728871, 28.65380458, 27.47802083,
##        24.11100374, 18.47258623, 18.92686632, 16.76235533, 16.73563297,
##        26.94357367, 25.847957  , 29.19894068, 20.34315128, 22.48093991,
##        18.20536265, 22.4274952 ])
```

```
model1.fittedvalues
```

```
## Mazda RX4              23.282611
## Mazda RX4 Wag          21.919770
## Datsun 710             24.885952
## Hornet 4 Drive         20.102650
## Hornet Sportabout      18.900144
## Valiant                18.793255
## Duster 360             18.205363
## Merc 240D              20.236262
## Merc 230               20.450041
```

Simple Linear Regression Models

- Residuals

```
model1.resid
```

```
## Mazda RX4                -2.282611
## Mazda RX4 Wag            -0.919770
## Datsun 710               -2.085952
## Hornet 4 Drive            1.297350
## Hornet Sportabout        -0.200144
## Valiant                  -0.693255
## Duster 360               -3.905363
## Merc 240D                 4.163738
## Merc 230                  2.349959
## Merc 280                  0.299856
## Merc 280C                -1.100144
## Merc 450SE                0.866873
## Merc 450SL               -0.050247
## Merc 450SLC              -1.883024
## Cadillac Fleetwood        1.173350
## Lincoln Continental       2.103288
## Chrysler Imperial         5.981074
## Fiat 128                  6.872711
## Honda Civic               1.746195
```

Simple Linear Regression Models

- ANOVA table: https://www.statsmodels.org/stable/generated/statsmodels.stats.anova.anova_lm.html

```
anova = sm.stats.anova_lm(model1)
print(anova)
```

```
##               df      sum_sq     mean_sq          F        PR(>F)
## wt           1.0  847.725250  847.725250  91.375325  1.293959e-10
## Residual    30.0  278.321938    9.277398        NaN           NaN
```

Simple Linear Regression Models

- Model fit using package `sklearn`: https://scikit-learn.org/stable/supervised_learning.html#supervised-learning
  - Linear Regression: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html

```
import numpy as np
from sklearn.linear_model import LinearRegression
regr = LinearRegression()
y=mtcars_data['mpg']
X=np.array(mtcars_data['wt']).reshape(-1, 1)
#reshape as 1-column matrix since there is only one single feature
regr.fit(X,y)
```

```
## LinearRegression()
```

```
print(regr.intercept_)
```

```
## 37.28512616734204
```

```
print(regr.coef_)
```

```
## [-5.34447157]
```

Simple Linear Regression Models

- Fitted values

```
regr.predict(X)
```

```
## array([23.28261065, 21.9197704 , 24.88595212, 20.10265006, 18.90014396,
##        18.79325453, 18.20536265, 20.23626185, 20.45004071, 18.90014396,
##        18.90014396, 15.53312687, 17.3502472 , 17.08302362,  9.22665041,
##         8.29671236,  8.71892561, 25.52728871, 28.65380458, 27.47802083,
##        24.11100374, 18.47258623, 18.92686632, 16.76235533, 16.73563297,
##        26.94357367, 25.847957  , 29.19894068, 20.34315128, 22.48093991,
##        18.20536265, 22.4274952 ])
```

- R code

```
model1 = lm(mpg ~ wt, data=mtcars);
coef(model1);
model1summary=summary(model1);
model1summary;
model1summary$fstatistic;
confint(model1, level = 0.95);
anova(model1);
```

Simple Linear Regression Models

- Confidence intervals for future observations: https://www.statsmodels.org/stable/generated/statsmodels.regression.linear_model.OLSResults.get_prediction.html
- Suppose Suppose there are two future observation x_1=3.3 and x_2=3.5.
- Confidence intervals and Prediction confidence intervals

```
import pandas as pd
X1=pd.DataFrame({'wt': [3.3, 3.5]})
predictions=model1.get_prediction(X1)
print(round(predictions.summary_frame(alpha=0.05),3))
```

```
##      mean  mean_se  mean_ci_lower  mean_ci_upper  obs_ci_lower  obs_ci_u
## 0  19.648    0.540         18.545         20.752        13.331        25
## 1  18.579    0.561         17.433         19.726        12.254        24
```

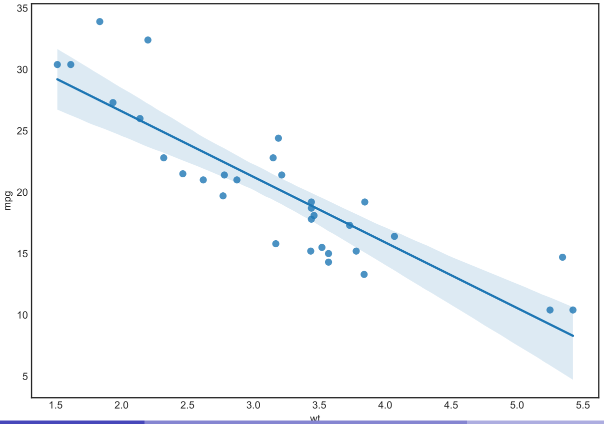
- R code

```
new =data.frame( wt=c(3.3, 3.5) )
predict(model1, newdata=new, interval="confidence", level=0.95)
predict(model1, newdata=new, interval="prediction",level=0.95)
```

Simple Linear Regression Models

- Plot both confidence band and the scatter plot

```
sns.regplot(data=mtcars_data,x='wt', y='mpg')
#sns.lmplot(data=mtcars_data,x='wt', y='mpg')
plt.show()
```

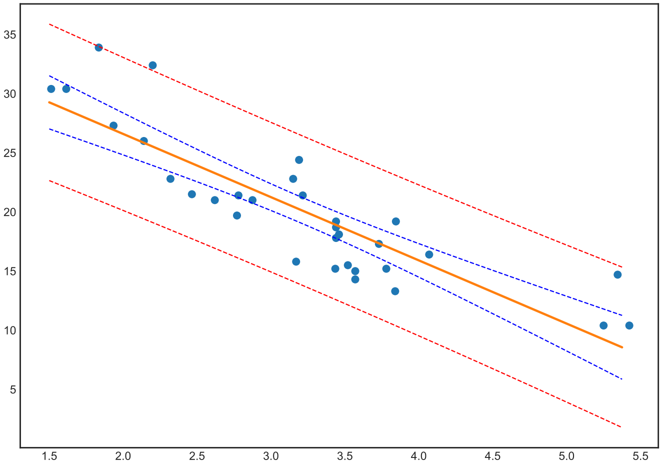


Simple Linear Regression Models

- Plot both confidence band and prediction confidence band to the scatter plot

```
import matplotlib.pyplot as plt
X=pd.DataFrame(np.arange(1.5, 5.5, 0.125),columns=['wt'])
#make sure the number of rows is 32
x=mtcars_data['wt']
predictions=model1.get_prediction(X)
pred_data=predictions.summary_frame(alpha=0.05)
fittedvalues = pred_data.iloc[:,0]
predict_mean_se  = pred_data.iloc[:, 1]
predict_mean_ci_low = pred_data.iloc[:, 2]
predict_mean_ci_upp = pred_data.iloc[:, 3]
predict_ci_low = pred_data.iloc[:, 4]
predict_ci_upp = pred_data.iloc[:, 5]
y=mtcars_data['mpg']
plt.plot(x, y, 'o')
plt.plot(X, fittedvalues, '-', lw=2)
plt.plot(X, predict_ci_low, 'r--', lw=1)
plt.plot(X, predict_ci_upp, 'r--', lw=1)
plt.plot(X, predict_mean_ci_low, 'b--', lw=1)
plt.plot(X, predict_mean_ci_upp, 'b--', lw=1)
plt.show()
```

Simple Linear Regression Models



Simple Linear Regression Models

- R code

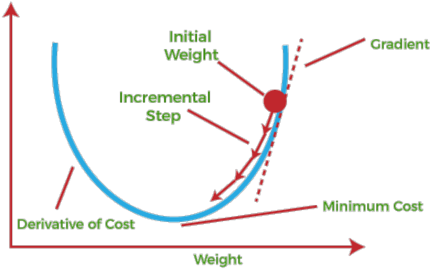
```
library(ggplot2)
temp_var=predict(model1, interval="prediction");
new_df = cbind(mtcars, temp_var);
ggplot(new_df, aes(wt, mpg))+
    geom_point() +
    geom_line(aes(y=lwr), color = "red", linetype = "dashed")+
    geom_line(aes(y=upr), color = "red", linetype = "dashed")+
    geom_smooth(method=lm, se=TRUE);
```

Gradient Descent Algorithm

- In mathematics, **gradient descent** (also often called **steepest descent**) is a first-order iterative optimization algorithm for finding a local minimum of an objective function, a differentiable function.
- It is a greedy technique that finds the optimal solution by taking a step in the direction of the maximum rate of decrease of the function.
- Gradient descent is by far the most popular optimization strategy used in deep learning.
- We introduce the algorithm for simple linear regression model fit.
  - Watch Lecture 2.6 — Linear Regression With One Variable | Gradient Descent Intuition and Lecture 2.7 — Linear Regression With One Variable | Gradient Descent For Linear Regression

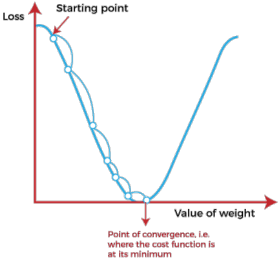
Gradient Descent Algorithm

- Idea: suppose the objective function $y = f(x)$ has one parameter x only and we try to find a local minimizer or **local minimum** of $f(x)$.
  - If we move towards a negative gradient or away from the gradient of the function at the current point, it will give the **local minimum** of that function.



Gradient Descent Algorithm

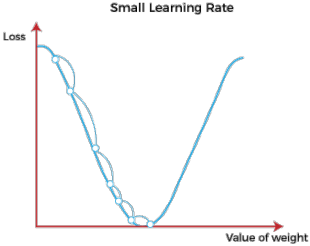
- The main objective of using a gradient descent algorithm is to minimize the **cost function** using iteration.

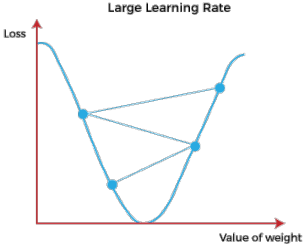


- To achieve this goal, it performs two steps iteratively:
  - (1) Calculates the first-order derivative of the function to compute the gradient or slope of that function.
  - (2) Move away from the direction of the gradient, which means slope increased from the current point by α times, where α is defined as **Learning Rate**.
  - It is a **tuning parameter** in the optimization process which helps to decide the length of the steps.

Gradient Descent Algorithm

- Learning Rate: It is defined as the step size taken to reach the minimum or lowest point. This is typically a small value that is evaluated and updated based on the behavior of the cost function.





Gradient Descent Algorithm

- The **cost/loss function** is defined as the measurement of difference or error between actual values and expected values at the current position and present in the form of a single real number.
- Linear Regression Cost/Loss Function:

$$J = J(\beta_0, \beta_1) = \sum_{i=1}^{n} [y_i - (\beta_0 + \beta_1 x_i)]^2.$$

- Gradient:

$$\nabla J = \left(\frac{\partial J}{\partial \beta_0}, \frac{\partial J}{\partial \beta_1} \right)$$

- Gradient Descent:

$$\Delta J = -\alpha \nabla J = -\alpha \left(\frac{\partial J}{\partial \beta_0}, \frac{\partial J}{\partial \beta_1} \right)$$

- Update $\boldsymbol{\beta} = (\beta_0, \beta_1)$ (so we need to choose an initial value) as

$$\boldsymbol{\beta} := \boldsymbol{\beta} + \Delta J.$$

Gradient Descent Algorithm

- If a data set is large, it may take very long time for GD to converge. The solution is SGD.
- SGD (Stochastic Gradient Descent) method uses a random sample of data to calculate the cost/loss function and its derivative in each iteration during the iterating procedure.

Gradient Descent Algorithm

- Let's use Gradient Descent Algorithm to estimate parameters in simple linear regression models

```
import numpy as np
def gradient_descent(X, y, theta0, theta1, learning_rate, tol, max_iter):
    error=1
    m = len(X)
    iter_count =0
    while error>tol and iter_count<max_iter:
        # Calculate the predicted values
        y_pred = theta0 + theta1*X

        cost = (1/(2*m))*sum((y-y_pred)**2) #cost function

        d_theta0 = -(1/m)*sum(y-y_pred)
        d_theta1 = -(1/m)*sum((y-y_pred)*X) # Calculate the gradients
        theta0_prev=theta0
        theta1_prev=theta1
        theta0=theta0-learning_rate*d_theta0
        theta1=theta1-learning_rate*d_theta1 #Update the parameters
        error=max(abs(theta0 - theta0_prev),abs(theta1 - theta1_prev))
        iter_count+=1

    return theta0, theta1, cost, iter_count
```

Gradient Descent Algorithm

- Apply the function above to mtcars data

```
y=np.array(mtcars_data.mpg)
X=np.array(mtcars_data.wt)

theta0 = 0
theta1 = 0 #initial values of beta0 and beta1
learning_rate = 0.1
tol=0.0001
max_iter=2000

theta0, theta1, cost, iter_count = gradient_descent(X,y,
theta0,theta1,learning_rate,tol,max_iter)

print("theta0 = ", theta0)
```

```
## theta0 =  37.27209076474596
```

```
print("theta1 = ", theta1)
```

```
## theta1 =  -5.340727813952249
```

```
print("Cost = ", cost)
```

```
## Cost =  4.348787371960817
```

```
print("Iterations = ", iter_count)
```

```
## Iterations =  1038
```

License