Learning Haskell Data Analysis

Analyze, manipulate, and process datasets of varying sizes efficiently using Haskell

James Church
In this package, you will find:

- The author biography
- A preview chapter from the book, Chapter 6 'Correlation and Regression Analysis'
- A synopsis of the book’s content
- More information on Learning Haskell Data Analysis

About the Author

James Church is an assistant professor of computer science at the University of West Georgia. James completed his PhD in computer science from the University of Mississippi under the advisement of Dr. Yixin Chen, with a research focus on computational geometry.

While at the University of Mississippi, he learned the skills necessary for data analysis in his side job, where he worked as a database administrator and analyst for the Marijuana Potency Monitoring Program (MPMP) led by Dr. Mahmoud ElSohly. The software written by James is used by the laboratory to store and track the chemical composition of marijuana samples. This data is provided to the United States National Institute on Drug Abuse to report marijuana potency.

The knowledge gained through his experience as an analyst for the MPMP (as well as other companies) was turned into a data analysis course for undergraduates at the University of Mississippi. This course was taught using the languages of Python and R.

James enjoys spending time with his wife, Michelle Allen, teaching, and playing board games with his friends.
Learning Haskell Data Analysis

This book serves as an introduction to data analysis methods and practices from a computational and mathematical standpoint. Data is the collection of information within a particular domain of knowledge. The language of data analysis is mathematics. For the purposes of computation, we will use Haskell, the free, general-purpose language. The objective of each chapter is to solve a problem related to a common task in the craft of data analysis. The goals for this book are two-fold. The first goal is to help the reader gain confidence in working with large datasets. The second goal is to help the reader understand the mathematical nature of data. We don't just recommend libraries and functions in this book. Sometimes, we ignore popular libraries and write functions from scratch in order to demonstrate their underlying processes. By the end of this book, you should be able to solve seven common problems related to data analysis (one problem per chapter after the first chapter). You will also be equipped with a mental flowchart of the craft, from understanding and cleaning your dataset to asking testable questions about your dataset. We will stick to real-world problems and solutions. This book is your guide to your data.

What This Book Covers

Chapter 1, Tools of the Trade, discusses the software and the essential libraries used in the book. We will also solve two simple problems—how to find the median of a list of numbers and how to locate the vowels in a word. These problems serve as an introduction to working with small datasets. We also suggest two nonessential tools to assist you with the projects in this text—Git and Tmux.

Chapter 2, Getting Our Feet Wet, introduces you to csv files and SQLite3. CSV files are human-and machine-readable and are found throughout the Internet as a common format to share data. Unfortunately, they are difficult to work with in Haskell. We will introduce a module to convert csv files into SQLite3 databases, which are comparatively much easier to work with. We will obtain a small csv file from the US Geological Survey, convert this dataset to an SQLite3 database, and perform some analysis on the earthquake data.

Chapter 3, Cleaning Our Datasets, discusses the oh-so-boring, yet oh-so-necessary topic of data cleaning. We shouldn't take clean, polished datasets for granted. Time and energy must be spent on creating a metadata document for a dataset. An equal amount of time must also be spent cleaning this document. This involves looking for blank entries or entries that do not fit the standard that we defined in our metadata document. Most of the work in this area is performed with the help of regular expressions. Regular expressions are a powerful tool by which we can search and manipulate data.
Chapter 4, Plotting, looks at the plotting of data. It's often easier to comprehend a dataset visually than through raw numbers. Here, we will download the history of the publicly traded companies on the New York Stock Exchange and discuss the investment strategy of growth investing. To do this, we will visually compare the yearly growth rate of Google, Microsoft, and Apple. These three companies belong to a similar industry (technology) but have different growth rates. We will discuss the normalization function, which allows us to compare companies with different share prices on the same graph.

Chapter 5, Hypothesis Testing, trains us to be skeptical of our own claims so that we don't fall for the trap of fooling ourselves. We will give ourselves the challenge of detecting an unfair coin. Successive coin flips follow a particular pattern called the binomial distribution. We will discuss the mathematics behind detecting whether a particular coin is following this distribution or not. We will follow this up with a question about baseball—"Is there a benefit if one has home field advantage?" To answer this question, we will download baseball data and put this hypothesis to the test.

Chapter 6, Correlation and Regression Analysis, discusses regression analysis. Regression analysis is a tool by which we can interpolate data where there is none. In keeping with the baseball theme, we will try to measure how much benefit there is to scoring baseball runs and winning baseball games. We will compute the runs-per-game and the win percentage of every team in Major League Baseball for the 2014 season and evaluate who is overperforming and underperforming on the field. This technique is simple enough to be used on other sports teams for similar analysis.

Chapter 7, Naive Bayes Classification of Twitter Data, analyzes the tweets from the popular social networking site, Twitter. Twitter has broad international appeal and people from around the world use the site. Twitter's API allows us to look at the language of each tweet. Using the individual words and the identified language, we will build a Naive Bayes classifier to detect the language of the sentences based on a database of downloaded tweets.
Chapter 8, Building a Recommendation Engine, continues with the analysis of the Twitter data and helps us create our own recommendation engine. This recommendation will help users find other users with similar interests based on the frequency of the words used in their tweets. There is a lot of data in word frequencies and we don't need all of it. So, we will discuss a technique to reduce the dimensionality of our data called Principal Component Analysis (PCA). PCA engines are used to recommend similar products for you to purchase or watch movies on commercial websites. We will cover the math and the implementation of a recommendation engine from scratch.

In each chapter we will introduce new functions. These functions will be added to a module file titled LearningDataAnalysis0X (where X is the current chapter number). We will frequently use functions from the earlier chapters to solve the problem from the chapter at hand. It will help you follow the chapters of this book in order so that you know when special functions mentioned in this book have been introduced.

Appendix, Regular Expressions in Haskell, focuses on the use of regular expressions in Haskell. If you aren't familiar with regular expressions, this will be a short reference guide to their usage.
In the previous chapter, we examined the classical procedure to test claims using the normal distribution curve. We also discussed the fundamental concept of variance and presented the function that computes the standard deviation of a dataset.

This chapter will examine the relationships between the input and output data. This is a truism to most sports fans—there exists a relationship between scoring and winning. It should seem obvious that sports teams that score higher points tend to win more games. As you might expect, the teams that don't often score high points tend to not win that often. The craft of measuring the relationship between the input (the number of points scored) and output (whether the team won or not) data is known as correlation analysis. Regression analysis allows us to estimate the result of an unknown output based on the input by creating an equation that minimizes errors between the independent and dependent variables that are believed to be linked. After creating a regression equation, we estimate the output for each of the known outputs based on the known inputs. This understanding is not without its drawbacks. A discussion of this approach should also come with an understanding of the potential errors that result from using it.

In this chapter, we will cover the following:

- The terminology of correlation and regression
- Study – is there a relationship between scoring and winning in baseball?
- Regression analysis
- The pitfalls of regression analysis
Correlation and Regression Analysis

The terminology of correlation and regression
Before we explore any data, we will discuss some terminology. When using regression analysis, we need a set of input and output variables. Analysis where a single column of data is used as an input variable is known as **univariate analysis**. Analysis that takes multiple sets of input variables is known as **multivariate analysis**. Regression analysis allows us to estimate unknown values in a single column of output. The input data is known as an **independent variable**. There are no assumptions being made as to how independent variables behave. Unlike the input, the output is known as a **dependent variable**. The assumption in this case is that the input variables impact the output variable, for example, we will return to our assumption that scoring high points leads to a team winning more often than an average number of times. An average team in this context is a team that wins and loses an equal number of times. A team can work hard, practice, and exercise to their full potential in order to account for the offensive effort of the team, which will account for the team scoring more points than the average number of points. We generally believe that these actions translate to a team that wins more often than an average team (while we ignore some equally important factors like the defensive abilities of an opposing team). Likewise, a team puts forth a minimal amount of effort into practice in order to score fewer points. We also believe that these actions translate to a team that wins less often than an average number of times. In both cases, the team represents an independent variable. The amount of hard work put towards an offensive effort is purely under their control. Unlike offensive effort, winning is not completely under their control. An opposing team might have a better offensive or defensive plan, or more capable players on their side. These qualities represent an unknown variable when performing regression. Finally, there is luck. Ignoring luck, our assumption is that winning is dependent on scoring, and scoring is dependent on offensive effort.

The expectation of a variable
An independent variable is denoted as \( X \). If we have multiple independent variables, then you will see each independent variable denoted as \( X_1, X_2, X_3, \ldots, X_m \). A dependent variable is denoted as \( Y \). The \( X \) and \( Y \) variables represent a dataset of \( n \) values (or observations). The mean of \( X \) is known as \( X \)-bar. If we wish to take every value of \( X \) and subtract \( X \)-bar from it, we will write it in the following way:

\[
X - \bar{X}
\]
The term *average* (which is the result of a sum of a listing of values divided by the number of values) also goes by the term *mean* as well as *expectation*. Sometimes, we will see the preceding formula written as follows:

\[ X - E[X] \]

You can think of \( E \) as a function that computes the average of a list of values.

The result of this operation will be a new dataset with the average being 0. If a value in this dataset is positive, we know that the result is above average. Likewise, negative values will give the result as below average.

**The variance of a variable**

We would like to know the spread of this variable. To do this, we will take each value in the dataset, subtract the mean, and then square the result. This produces a dataset that consists of positive values. After this step, add up all of the values and divide the sum by the number of observations to get the average squared distance from the average. This, as we discussed in *Chapter 5, Hypothesis Testing*, is the population variance. To find out the sample variance, rather than dividing by \( n \), we divide it by \((n-1)\):

\[
Var[X] = \frac{1}{n} \sum_{i=1}^{n} (X_i - \bar{X})^2
\]

We can write the same formula with the help of a cleaner notation by using the \( E \) function:

\[
Var[X] = E \left[ (X - E[X])^2 \right]
\]

To find out the population standard deviation, we will take the square root of the population variance. The standard deviation is signified by the Greek letter, sigma (denoted as \( \sigma \)):

\[
\sigma = \sqrt{E \left[ (X - E[X])^2 \right]}
\]
Correlation and Regression Analysis

Normalizing a variable

Now that we know the average distance from the average of the values in the dataset, we can use this to normalize the $X$ variable. To do this, we will divide each value in $X - X$-bar by the standard deviation:

$$\frac{X - \bar{X}}{\sigma}$$

This normalized version of our dataset still contains positive and negative values, but it is also a measure of how extreme in distance the normalized variable is from the mean. A score between -1 and 1 means that a value is closer to the mean than the typical data value. Scores ranging from -2 to -1 and from 1 to 2 mean that the value is between one and two times distant from the mean than the typical value. Most of your data should fall between a score of -2 and 2. Scores from -3 to -2 and from 2 to 3 indicate that the value is a little more distant. A value greater than 3 or less than -3 means that this value is more than 3 times the distance from the average of the values in the dataset than the typical value. Values with a score in this range are considered rare and indicate special circumstances that merit investigation. Values that deviate significantly from the majority of a dataset are called outliers. When an outlier value has been identified, it possibly represents special circumstances and should be investigated for unique qualities. It can also mean something less special—a noisy data point was not properly identified in the cleaning phase of data analysis.

The covariance of two variables

When working with two variables (in our case, an input and an output variable), we may want to study how the variables move in conjunction with each other. Like variance, the tool known as covariance helps us to measure how variables relate to each other. Instead of one $X$ variable, we now have two—$X$ and $Y$.

We will begin by subtracting the mean of $X$ from each value of $X$, whose answer is then multiplied by the result of the mean of $Y$ subtracted from each value of $Y$:

$$(X - \bar{X})(Y - \bar{Y})$$
Again, we add each of these distance measures and divide the sum by the number of observations to find out the population covariance coefficient. To find the sample covariance coefficient, we change $n$ to $n-1$:

$$ Cov[X, Y] = \frac{1}{n} \sum (X - \bar{X})(Y - \bar{Y}) $$

Again, we can write the same formula by using the $E$ notation:

$$ Cov[X, Y] = E\left[ \left( X - E[X] \right) \left( Y - E[Y] \right) \right] $$

The covariance coefficient is a measurement of how the variables relate to each other. If the $X$ variable increases as the $Y$ variable increases, then this coefficient will have a positive value. Likewise, if $X$ or $Y$ increases and the other variable decreases, then this coefficient will have a negative value.

**Finding the Pearson r correlation coefficient**

We can normalize this value as we did before. This normalized value will always have a value from -1 to 1. We need the standard deviations of $X$ and $Y$ ($\sigma_x$ and $\sigma_y$). This normalized version of the covariance value is known as the **Pearson r correlation coefficient**. The formula for this can be denoted as follows:

$$ r = \frac{Cov[X, Y]}{\sigma_x \sigma_y} $$

As with the correlation coefficient, a positive $r$ value informs us that the variables are linearly correlated (as the value of one variable increases, the value of the other increases) and a negative $r$ value informs us that the variables are inversely correlated (as the value of a variable increases, the other decreases). The closer an $r$ value is to the extremes of -1 or 1, the more the strength of the correlation it indicates. An $r$ value that is close to 0 tells us that the connection between the two variables is weak.
Finding the Pearson $r^2$ correlation coefficient

The $r$ value is tweaked further to $r^2$ (we simply multiply $r$ by itself). Because $r$ ranges from -1 to 1, the value of $r^2$ will always be a value from 0 to 1. A higher $r^2$ implies that there is stronger evidence of a correlation. A lower $r^2$ implies little or no correlation. An $r^2$ that is greater than 0.9 is considered to be an excellent correlation, while an $r^2$ that is less than 0.5 is considered to be weak. Interpreting an $r^2$ is a form of art—it is what you make of it. The $r^2$ values go by the name, coefficient of determination.

We will not continue this discussion without talking about a key pitfall of $r^2$—the discovery of an input and output variable that produces a high $r^2$ value does not automatically imply that the input has an impact on the output. This is known by the saying, correlation does not imply causation. When data mining for correlations, you are bound to find pairs of variables that produce high $r^2$ values. Some of these input variables might have a causal effect on the output variable, and at times, you may find a similar pattern between two unrelated variables. This frequently happens when two output variables are compared. This measurement only tells you what correlates, not whether the discovered correlations are sensible. You might discover a link between the sale of food and the sale of beverages at a restaurant, but there is nothing interesting about this correlation because these variables are dependent on the number of customers who frequent the restaurant.

Translating what we've learned to Haskell

We can express these formulas to Haskell in the following way:

```haskell
module LearningDataAnalysis06 where

import Data.List
import Graphics.EasyPlot
import LearningDataAnalysis02
import LearningDataAnalysis04
import LearningDataAnalysis05

{- Covariance -}
covariance :: [Double] -> [Double] -> Double

\[ \text{covariance } x \ y = \text{average } (x - x_{avg}) \cdot (y - y_{avg}) \]

\[ \text{where} \]

\[ x_{avg} = \text{average } x \]

\[ y_{avg} = \text{average } y \]
```
Study – is there a connection between scoring and winning?

We will continue using the baseball dataset that we introduced in the last chapter. We can check whether there is a correlation between scoring and winning. To do this, we will compute the runs scored per game for each team as well as the win percentage for each team. We must compute the total number of runs scored, the total number of wins, and the total number of games played by each team.

A consideration before we dive in – do any games end in a tie?

Before we can proceed with our analysis, we need to make sure that we handle data that does not quite fit into our model. In this case, we are talking about tie games. There are many games played each year and, often, games have an extra inning. However, does any game ever end up as a tie? A few select queries will let us know whether the home score equals the away score. The first query will tell us the number of games in our database and the second will report the number of games that ended in a tie:

```
> queryDatabase "winloss.sql" "SELECT COUNT(*) FROM winloss"
[[SqlByteString "2429"]]
> queryDatabase "winloss.sql" "SELECT COUNT(*) FROM winloss WHERE awayScore==homeScore;"
[[SqlByteString "0"]]
```

Games that ended in a tie are not a part of this dataset. All the 2,429 games of the 2014 season had clear winners.
Compiling the essential data

You may recall that the data is organized according to the individual games. For each record in the database, the away team is listed before the home team. We are going to split the information gathering phase into two steps—a step to gather the home team stats and a step for the away team stats. The information that we are going to gather is the 3-letter code for the team name, the number of home wins, the total number of runs scored at home games, and the number of home games. First, we will gather the stats related to the home team. The GROUP BY statement will make sure that all the teams stay in the alphabetical order, according to the team code:

```plaintext
> homeRecord <- queryDatabase "winloss.sql" "SELECT homeTeam, SUM(homescore > awayscore), SUM(homescore), COUNT(*) FROM winloss GROUP BY homeTeam;"
```

Second, we will collect the stats pertaining to the away team. With the following query, we will switch from looking at the information related to the home field to the data related to the away field:

```plaintext
> awayRecord <- queryDatabase "winloss.sql" "SELECT awayTeam, SUM(awayscore > homescore), SUM(awayscore), COUNT(*) FROM winloss GROUP BY awayTeam;"
```

Now, we must combine these datasets. First, we will gather the total number of wins. In the two matrices, this information appears in column index 1:

```plaintext
> let totalWins = zipWith (+) (readDoubleColumn homeRecord 1) (readDoubleColumn awayRecord 1)
> totalWins
[98.0, 64.0, 79.0, 96.0, 71.0, 73.0, 76.0, 71.0, 73.0, 85.0, 66.0, 90.0, 70.0, 93.0, 77.0, 82.0, 70.0, 84.0, 79.0, 88.0, 73.0, 88.0, 77.0, 87.0, 88.0, 90.0, 77.0, 67.0, 83.0, 96.0]
```

Next, we will gather the total number of runs scored. This information appears in column index 2:

```plaintext
> let totalRuns = zipWith (+) (readDoubleColumn homeRecord 2) (readDoubleColumn awayRecord 2)
> totalRuns
[773.0, 614.0, 573.0, 705.0, 634.0, 660.0, 614.0, 595.0, 669.0, 755.0, 757.0, 629.0, 651.0, 715.0, 645.0, 650.0, 715.0, 633.0, 629.0, 729.0, 619.0, 682.0, 535.0, 634.0, 65.0, 619.0, 612.0, 637.0, 723.0, 686.0]
```
Finally, we will gather the total number of games played by each team. There are 162 games in a typical baseball season, but we should not depend on this tidbit of knowledge and should instead calculate it ourselves. This information is in column index 3:

```haskell
> let totalGames = zipWith (+) (readDoubleColumn homeRecord 3) (readDoubleColumn awayRecord 3)
> totalGames
[162.0,161.0,162.0,162.0,162.0,162.0,162.0,162.0,162.0,162.0,162.0,162.0,162.0,162.0,161.0,162.0,162.0,162.0,162.0,162.0,162.0,162.0,162.0,162.0,162.0,162.0,162.0,162.0,162.0,162.0,162.0,162.0]
```

As you can see, almost every team played 162 games (save two teams that played 161).

We are still not done with the compiling of information. We now need the win percentage and the number of runs per game, both of which can be obtained by dividing totalWins and totalRuns by totalGames, as follows:

```haskell
> let winPercentage = zipWith (/) totalWins totalGames
> let runsPerGame = zipWith (/) totalRuns totalGames
```

### Searching for outliers

We mentioned previously that a simple definition of an outlier is a data point that is more than 3 standard deviation units from the mean of the dataset. Let's explore these two datasets (runsPerGame and winPercentage) for outliers. First, we will look at the runsPerGame function. Here, we will take the absolute value of the normalized dataset and check to see whether any of these values exceed 3 units:

```haskell
> any (\xi -> abs( (xi - average runsPerGame) / standardDeviation runsPerGame) > 3) runsPerGame
False
```

None of the values in the runsPerGame function exceed 3 units. Let's do this again for the winPercentage dataset:

```haskell
> any (\xi -> abs((xi - average winPercentage) / standardDeviation winPercentage) > 3) winPercentage
False
```
Again, none of the values in either the runsPerGame dataset or the winPercentage dataset seem to indicate that they contain an outlier. If we were to encounter an outlier, it might indicate that either something is wrong with the dataset, or there is something unique about that particular data point. Either way, outliers will require further investigation on the part of the data analyst. If you believe that an outlier in your data exists based on certain unique circumstances, you will be forgiven if you exclude this observation from your overall analysis and make a note of the exclusion.

**Plot – runs per game versus the win percentage of each team**

Let’s plot this information. Plotting will prove useful during the interpretation of the results. The x-axis and y-axis of this chart will represent the number of runs per game and the win percentage of each team respectively:

```haskell
> import Graphics.EasyPlot
> plot (PNG "runs_and_wins.png") $ Data2D [Title "Runs Per Game VS Win % in 2014"] [] $ zip runsPerGame winPercentage
True
```

The preceding statement would give the following chart as a result:
What immediately pops out when we look at this image is how cloudy the data looks. We can see that the data does trend upward, and this is evidence of a positive correlation. Yet, it is hard to make out a line in the data. You might notice that at the bottom right corner of the image is a data point representing the team with the third highest number of runs per game in 2014, and yet they have a dismal win percentage of less than 0.45. This (again) is the Colorado Rockies, who tend to have high scoring games at their one-mile-above-sea-level stadium, but this does not translate into a majority of wins for the team. At the opposite corner of this graph, we can see that there are three teams that scored less than 4 runs per game in more than half of their games in 2014. These are the teams with winning seasons in spite of having a less-than-stellar offense. This contradicts the idea that higher scores lead to the winning of more games.

**Performing correlation analysis**

Let's measure the correlation between these two variables:

```
> pearsonR runsPerGame winPercentage
0.40792278544575666
> pearsonRsqrd runsPerGame winPercentage
0.1664009988582482
```

A Pearson r score of 0.41 is positive, which indicates that there is a positive correlation between the runs per game and the win percentage. When you square this value to compute the $r^2$ score, you will see that it is 0.17. As we mentioned earlier, an $r^2$ score of less than 0.5 is considered a weak correlation of the two variables. We see in this analysis that scoring is a part of winning, but the relationship between these two variables should be considered weak. In other words, the idea that a team will improve their win percentage only by improving the runs scored per game is considered to be weak.

**Regression analysis**

Should we tell our coaches that scoring is not important? Of course not. A team needs to score at least one run to have a chance of winning a game. We should communicate to our coaches the importance of scoring more runs per game, even when we know that there is a weak correlation between scoring and winning. We communicate this importance by using regression analysis. With regression analysis, we create an equation that will allow us to estimate the win percentage of a team based on their runs per game value.
The approach that we will follow is known as simple linear regression. Linear regression is the simplest type of regression. The working assumption is that our data forms a straight line. While we admit that it is difficult to make out a line in our data, we shall make the assumption that a line exists. This line indicates the increase in the win percentage of a team as the team scores more runs per game. When one factor goes up, the other goes up linearly.

**The regression equation line**

A linear equation is as follows:

\[ y = A + Bx \]

In this equation, \( x \) represents the number of runs per game, \( B \) represents the slope (or the gradient) of the line this equation gives, \( A \) represents the y-intercept, and \( y \) represents the estimated winning percentage of a team that is able to score \( x \) runs per game. The equation line will represent a best fit that will closely follow down the middle of the data, minimizing the difference in the distance between the points above and below the line.

**Estimating the regression equation**

The regression equation is a best fit line. The goal in crafting this equation is to produce the smallest overall error between the real data and what the equation will estimate the data to be. We can minimize the error term by computing the covariance of the \( X \) and \( Y \) variables and dividing it by the variance of \( X \):

\[
B = \frac{\text{Cov}[X,Y]}{\sigma_x^2} = \frac{\text{Cov}[X,Y]}{\text{Var}[X]}
\]

We can compute the value of \( A \) (our y-intercept) by computing the average of \( X \) and \( Y \) and substituting these values into our linear equation:

\[
A = \bar{Y} - \bar{X}B
\]
Translate the formulas to Haskell

In Haskell, the preceding formula will look like the following code:

```haskell
{- Perform simple linear regression to find a best-fit line. 
  Returns a tuple of (gradient, intercept) -}
linearRegression :: [Double] -> [Double] -> (Double, Double)
linearRegression x y = (gradient, intercept)
  where
    xavg = average x
    yavg = average y
    xstdev = standardDeviation x
    gradient = covariance x y / (xstdev * xstdev)
    intercept = yavg - gradient * xavg
```

Here, I have renamed $B$ to gradient and $A$ to intercept.

Returning to the baseball analysis

When we execute the preceding code with our baseball data, we get the following output:

```haskell
> let (gradient, intercept) = linearRegression runsPerGame winPercentage
> gradient
6.702070689192714e-2
> intercept
0.22742671114723823
```

The value of the slope is 0.07 and the value of the intercept is 0.23. A slope of value 0.07 indicates that if a team increases their runs per game by 1 run per game, it should increase their seasonal win percentage by an estimated seven percent. This percentage grows with a linear progression, so an increase of 2 runs per game will increase their win percentage by an estimated 14 percent (and so forth). Here, we will estimate the win percentage for a fictional team that was able to score 3, 4, and 5 runs per game for an entire season:

```haskell
> 3*gradient+intercept
0.42848883182301967
> 4*gradient+intercept
0.4955095387149468
> 5*gradient+intercept
0.5625302456068739
```
Correlation and Regression Analysis

A team that scores 3 runs per game should win about 43 percent of their games. At 4 runs per game, the estimated win percentage is 50 percent. At 5 runs per game, this increases to 56 percent. In our dataset, the team with the highest win percentage won over 60 percent of their games while not quite hitting 5 runs per game.

Plotting the baseball analysis with the regression line

In the previous chart that displays the runs per game and the win percentage for each team, the chart ranges between 3.2 and 4.8 runs per game. I created a new dataset of line estimates based on the values of runs per year that range from 3.3 to 4.7. This way, we have a line that can fit nicely within the existing chart:

```haskell
> let winEstimate = map (\x -> x*gradient + intercept) [3.3, 3.4 .. 4.7]
> let regressionLine = zip [3.3, 3.4 .. 4.7] winEstimate
> plot (PNG "runs_and_wins_with_regression.png") [Data2D [Title "Runs Per Game VS Win % in 2014"] [] (zip runsPerGame winPercentage), Data2D [Title "Regression Line", Style Lines, Color Blue] [] regressionLine]
```

The preceding statements would give the following chart as a result:
What makes the regression line so helpful in our analysis is that it allows us to quickly identify the teams that are above average and below average. Above the line exists the teams that did better than our estimate suggests. Below the line exists the teams that did worse than our estimate.

The pitfalls of regression analysis

There are several pitfalls of regression analysis. We will go over some of the limitations of this analysis:

- **Keep your estimates close to the original input variable range:**
  The regression equation is in the form of a straight line. A line in a two-dimensional plane has a slope and a y-intercept and extends infinitely in two directions. Because of this, we are capable of estimating values beyond the range of our input variables, for example, the y-intercept of our equation is 0.22. This means that a baseball team that scores an average of 0 runs per game should win an estimated 22% of their games (which is laughable). Just because we can estimate values outside of the range of our input, it doesn't mean that we should.

- **There is more to regression than simple linear regression:** This chapter only looks at simple linear regression, but there are several other types of regression, including log regression (where we first compute the natural log of the output variable) and log-log regression (where we first compute the natural log of both the input and the output variables). The fundamental approach is the same, but having an understanding of the curvature of the data (and not automatically assuming that the data forms a line) sometimes yields better results.

- **Any analysis that uses the mean of a dataset is easily skewed:** Since the regression line is so dependent on the average of both the input and output variables, using simple linear regression can create a line that is distorted based on a single heavily skewed data point.

The next example illustrates this clearly. The following table represents a contrived dataset known as an **Anscombe's quartet**. This dataset was manually developed so that each dataset has a nearly identical mean of the \( x \) column, mean of the \( y \) column, correlation coefficient, and linear regression line. It demonstrates the problem of simple linear regression. The procedure is not robust with respect to outlier data values or data that isn't in a linear order. In each of these four datasets, a simple linear regression analysis reports that all four datasets follow the same linear path when it is clear that three of them do not.
The data for the following table was taken from the Wikipedia entry for an Anscombe's quartet:

<table>
<thead>
<tr>
<th></th>
<th>I</th>
<th>II</th>
<th>III</th>
<th>IV</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>x</td>
<td>10.0</td>
<td>8.04</td>
<td>10.0</td>
<td>7.46</td>
</tr>
<tr>
<td>y</td>
<td>8.04</td>
<td>10.0</td>
<td>6.58</td>
<td>8.0</td>
</tr>
<tr>
<td>II</td>
<td>8.0</td>
<td>6.95</td>
<td>8.0</td>
<td>6.77</td>
</tr>
<tr>
<td>x</td>
<td>8.0</td>
<td>6.95</td>
<td>8.0</td>
<td>6.77</td>
</tr>
<tr>
<td>y</td>
<td>6.95</td>
<td>8.0</td>
<td>8.0</td>
<td>6.77</td>
</tr>
<tr>
<td>III</td>
<td>13.0</td>
<td>7.58</td>
<td>13.0</td>
<td>12.74</td>
</tr>
<tr>
<td>x</td>
<td>13.0</td>
<td>7.58</td>
<td>13.0</td>
<td>12.74</td>
</tr>
<tr>
<td>y</td>
<td>7.58</td>
<td>13.0</td>
<td>13.0</td>
<td>12.74</td>
</tr>
<tr>
<td>IV</td>
<td>9.0</td>
<td>8.81</td>
<td>9.0</td>
<td>7.11</td>
</tr>
<tr>
<td>x</td>
<td>9.0</td>
<td>8.81</td>
<td>9.0</td>
<td>7.11</td>
</tr>
<tr>
<td>y</td>
<td>8.81</td>
<td>9.0</td>
<td>8.81</td>
<td>9.0</td>
</tr>
<tr>
<td></td>
<td>11.0</td>
<td>8.33</td>
<td>11.0</td>
<td>8.47</td>
</tr>
<tr>
<td>I</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>x</td>
<td>14.0</td>
<td>9.96</td>
<td>14.0</td>
<td>8.84</td>
</tr>
<tr>
<td>y</td>
<td>9.96</td>
<td>14.0</td>
<td>8.84</td>
<td>14.0</td>
</tr>
<tr>
<td>II</td>
<td>6.0</td>
<td>7.24</td>
<td>6.0</td>
<td>6.08</td>
</tr>
<tr>
<td>x</td>
<td>6.0</td>
<td>7.24</td>
<td>6.0</td>
<td>6.08</td>
</tr>
<tr>
<td>y</td>
<td>7.24</td>
<td>6.0</td>
<td>7.24</td>
<td>6.08</td>
</tr>
<tr>
<td>III</td>
<td>4.0</td>
<td>4.26</td>
<td>4.0</td>
<td>5.39</td>
</tr>
<tr>
<td>x</td>
<td>4.0</td>
<td>4.26</td>
<td>4.0</td>
<td>5.39</td>
</tr>
<tr>
<td>y</td>
<td>4.26</td>
<td>4.0</td>
<td>4.26</td>
<td>4.0</td>
</tr>
<tr>
<td>IV</td>
<td>12.0</td>
<td>10.84</td>
<td>12.0</td>
<td>19.0</td>
</tr>
<tr>
<td>x</td>
<td>12.0</td>
<td>10.84</td>
<td>12.0</td>
<td>19.0</td>
</tr>
<tr>
<td>y</td>
<td>10.84</td>
<td>12.0</td>
<td>10.84</td>
<td>12.0</td>
</tr>
<tr>
<td></td>
<td>7.0</td>
<td>4.82</td>
<td>7.0</td>
<td>6.42</td>
</tr>
<tr>
<td>I</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>x</td>
<td>7.0</td>
<td>4.82</td>
<td>7.0</td>
<td>6.42</td>
</tr>
<tr>
<td>y</td>
<td>4.82</td>
<td>7.0</td>
<td>4.82</td>
<td>7.0</td>
</tr>
<tr>
<td>II</td>
<td>5.0</td>
<td>5.68</td>
<td>5.0</td>
<td>5.73</td>
</tr>
<tr>
<td>x</td>
<td>5.0</td>
<td>5.68</td>
<td>5.0</td>
<td>5.73</td>
</tr>
<tr>
<td>y</td>
<td>5.68</td>
<td>5.0</td>
<td>5.68</td>
<td>5.0</td>
</tr>
</tbody>
</table>

Plotting the data of this table would give us the following charts:
These graphs were created using the Haskell functions, the EasyPlot library that was defined in this chapter, and the same procedure that was used to analyze the baseball data (except that all the outlier observations were retained for the purpose of this demonstration).

**Summary**

In this chapter, we took a crash course in univariate analysis. We took a closer look at the ways used to compute covariance, the Pearson r score, and the Pearson r² score, as well as the methods deployed to interpret this data. We then took a look at the baseball dataset again and explored the question, *Is there a relationship between scoring and winning?* It may be a surprise to the reader, but we found out that the relationship between the two variables is weak. We also looked at regression analysis, which allows us to estimate unknown output variables based on the covariance of the existing data. We also spent time knowing about the pitfalls of blindly using simple linear regression.

The next chapter looks more at prediction, but this time from the perspective of Bayesian analysis. Bayesian probability is a form of conditional probability where we estimate the likelihood of events happening based on the evidence of past probabilities.
Where to buy this book

You can buy Learning Haskell Data Analysis from the Packt Publishing website.

Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.

Click here for ordering and shipping details.