The `ispiranteRanney` Package

Steven H. Ranney

Contents

Introduction

Essentially, all [statistical] models are wrong, but some are useful. —George E. P. Box

The best experiments require no statistics.
— Al Zale, USGS/Montana Cooperative Fishery Research Unit

Data analysis is normally undertaken by those with training in statistical analyses (Burnham and Anderson 2002, Anderson 2008). Statistical analyses, and their resulting metrics (e.g., p-values, slope andd intercept values, parameter estimates, R^2 values) are often misinterpreted by those that posess little or no statistical training. In some cases, reliance on the output from stastical analyses may "cloud" a person's judgement; lack of training may let someone feel confident in their inference from a *significant* result that a relationship exists when, in reality, the results were spurious.

I propose to deliver to ispirante the *ispiranteRanney* pckage, written in response to an onine advertisement and subsequent webinar with Stefano Spada. *ispiranteRanney* will have several functions to help ispirante customers visualize, understand, and gain insights into their data. The package functions will be written with minimal statistical analyses to allow for a deeper, more thorough customer understanding.

Most of the functions provided in *ispiranteRanney* will be simple plotting functions; for example wait time by day of week, resolution time by date, satisfaction score by `assignee_name`, etc. Data insights are often generated through simple visualization and help-desk ticket data is no exception. However, the real insight lies in comparing some values to others. In this respect, *ispiranteRanney* will provide the means to make those visual comparisons. Coupled with simple statistical analyses, users will be able explore how some variables change in the presence of others.

The *ispiranteRanney* package will be written in the most recent stable release of `R` version 3.1.2, "Pumpkin Helmet", and tested for stability on the development version of `R`, 3.2.0 (R Core Team 2014).

A note about `R` and plots

This document was produced with the `knitr` `R` package. There will be a combination of text, `R` code, and `R` output. `R` data will look like this:

```r
#Create 1000 random values from a normal distribution with a mean of 10 and a SD of 0.8
tmp <- as.data.frame(rnorm(1000, 10, .8))
names(tmp) <- "value"
```

Lines in the "chunks" of `R` code that begin with a `#` are comments from the author and are not run by `R`. The output from the `R` code is preceeded by `##`. Plain text will look like the prose on this page. Plots will appear in their own window. In some cases plots may not fit at the bottom of a page. In those instances, `knitr` will move the plot to the next page and the bottom of the precedding page may be blank, or nearly so.

A note about boxplots

Box and whisker plots (boxplots) provide an interesting way of examining differences between categorical variables. To test the difference between categorical variables, a statistician may use an Analysis of Variance (ANOVA) test. ANOVA is an extention of a linear model and can identify if there are significant differences (with the critical value, α, set to an arbitrary value, normally 0.05) among categories.

```r
  #Create a vector of numeric values and classify them
  y <- c(rnorm(7, 2.5, .5), rnorm(7, 3.5, 2), rnorm(7, 10, 1.5))
  x <- c(rep("a", 7), rep("b", 7), rep("c", 7))

  #Compile into a dataFrame
  tmp <- data.frame(x, y)
  names(tmp) <- c("x", "y")
```

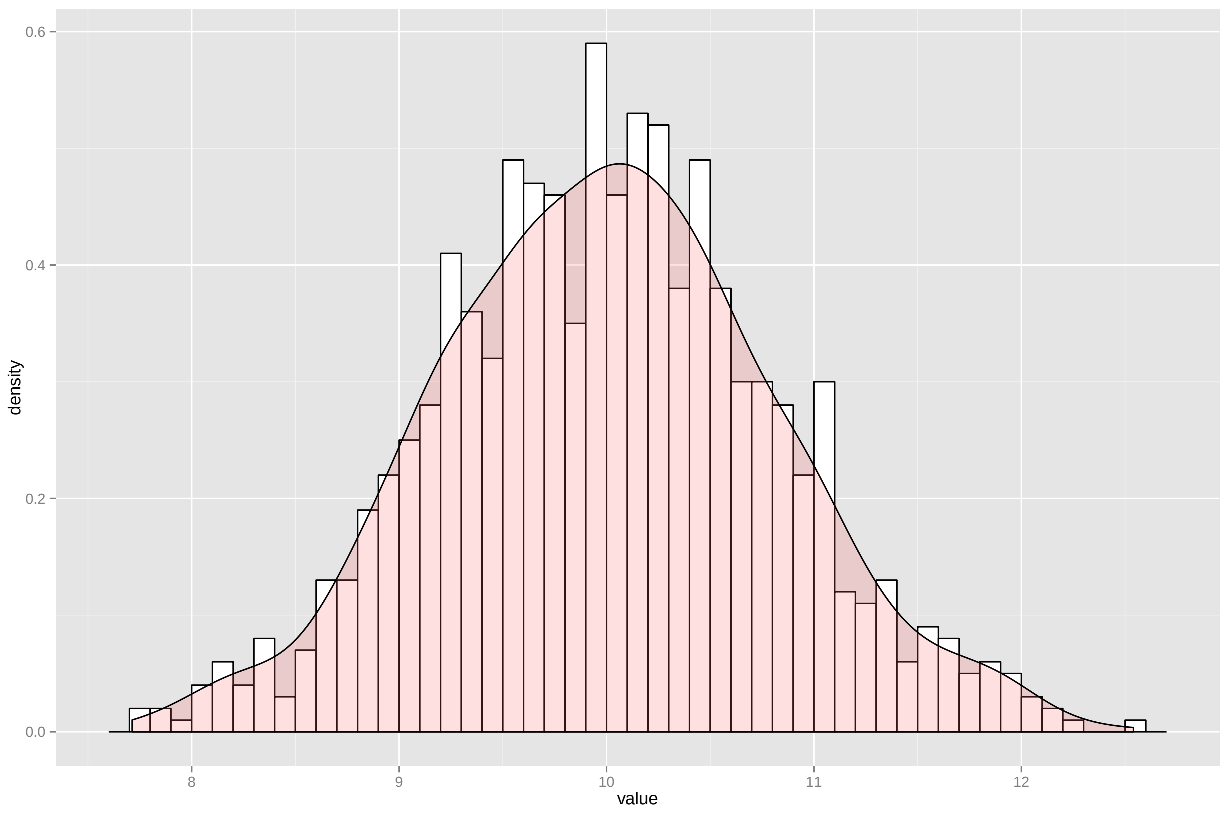
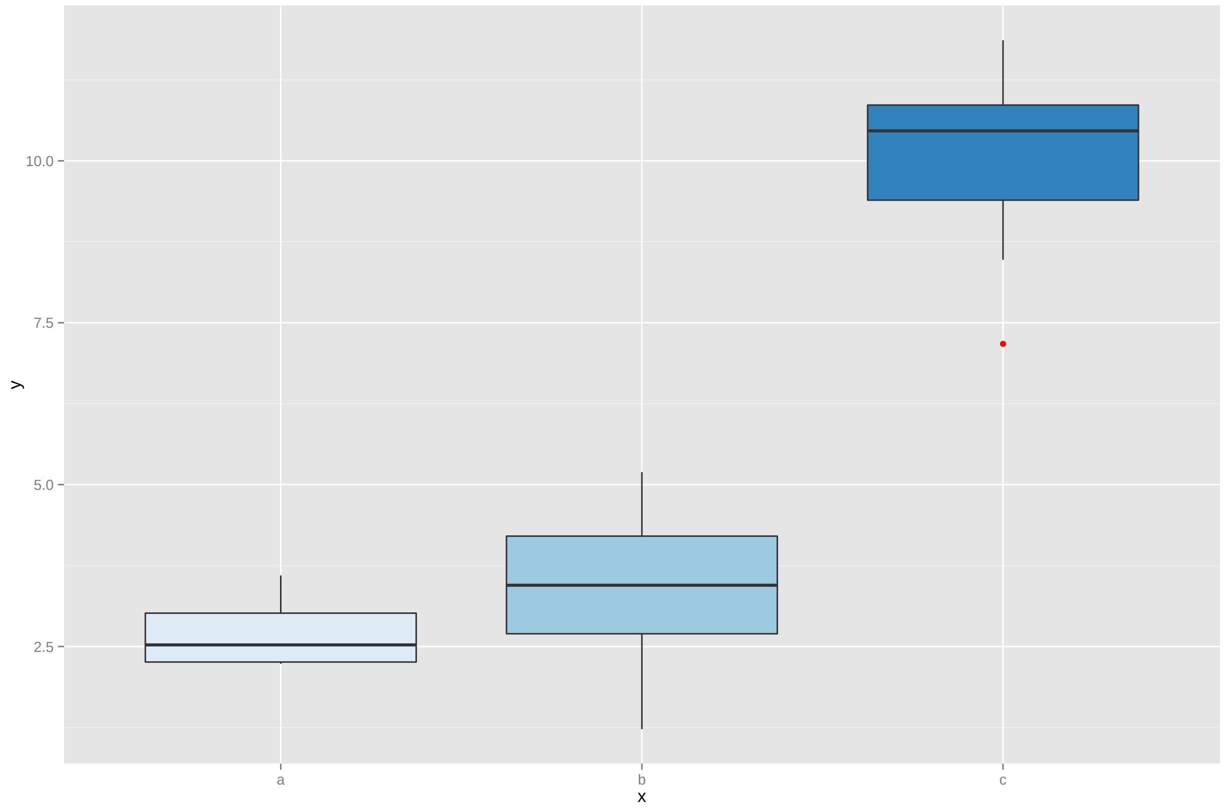


Figure 1: Histogram and kernel density of 1,000 random numbers with a mean = 10 and a standard deviation = 0.8



In looking at the plot, we can see that there are obvious differences among the groups. For example, group `a` appears similar to group `b` but both `a` and `b` appear to be quite different from `c`. Without a statistical test like an ANOVA, we can't know for certain.

```
  #Run an ANOVA on the values to look for significant differences
  mod <- aov(lm(y~x, data = tmp))
  summary(mod)
```

```
##             Df Sum Sq Mean Sq F value   Pr(>F)    
## x            2  227.7  113.86   73.73 2.13e-09 ***
## Residuals   18   27.8    1.54                     
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The results from the `summary(mod)` statement show us that the categorical `x` variable has a significant difference on the `y` variable. The asterisks (i.e., next to the `Pr(>F)` value in the right column tells us that the p-value is < 0.05. (A p-value < 0.05 is the 'rule of thumb' for a "significant" statistical" value.) Even though we know that `x` has a significant affect on `y`, we don't know specifically where the significant differences lie. A Tukey test will make pair-wise t-test comparisons between every `x` value and every other `x` value, automatically including the Bonferroni adjustment for multiple t-test comparisons (Day and Quinn 1989).

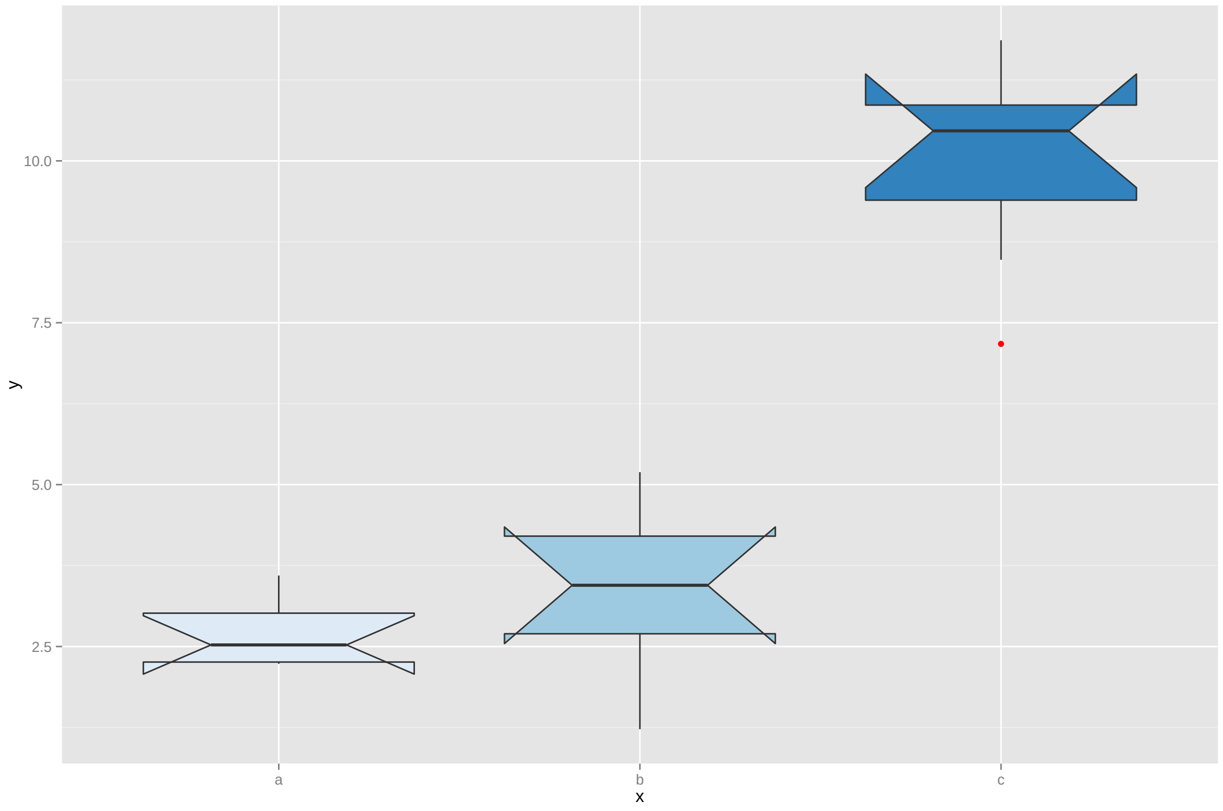
```
#Tukey's test will tell us where the significant (p < 0.05) differences lie
  TukeyHSD(mod)
```

```
##   Tukey multiple comparisons of means
##     95% family-wise confidence level
```

```
##
## Fit: aov(formula = lm(y ~ x, data = tmp))
##
## $x
##          diff       lwr      upr     p adj
## b-a 0.6798998 -1.015323 2.375123 0.5719913
## c-a 7.3006455  5.605422 8.995869 0.0000000
## c-b 6.6207457  4.925522 8.315969 0.0000000
```

From the results of the `TukeyHSD()` test, there is not a significant (p-value > 0.05) difference between group `a` and group `b` but there *is* a significant difference (p-value < 0.05) between `a` and `c` and `b` and `c`.

A Tukey test can be applied to any number of `x` variables produced by an ANOVA. However, making several comparisons among many different `x` variables would create a *messy* table and can be visually challenging to interpret (Burnham and Anderson 2002, Anderson (2008)). Box and whisker plots in `R` allow a user to add a notch into the boxes. When the argument `notch = TRUE` is added to the plotting statemtn, `R` will notch the box which provides a *rough* approximation of a 95% confidence interval around the median (the **heavy line** in the middle of the plot.)



In the above plot, the notches from `a` and `b` overlap each other but the neither the notches from `a` or `b` overlap with `c`. This is a good indicator that `a` and `b` are not significantly different from each other but that `a` and `b` are both significantly different from `c`.

While a statistical test is the only *valid* way an analyst can be sure if there are significant differences among categorical variables, adding the notches to a boxplot provide a very quick, easy, and tractable way to identify whether or not differences exist. As a result, rather than providing tables and tables of p-values in the *ispiranteRanney* package—which would quantify the differences between categorgical variables—boxplots will be notched and users will have an effective visual means of determining significant differences without having to rely on *magic* numbers like p-values. Always remember Mark Twain's quote about statistics: "*There are* *three kinds of lies: lies, damn lies, and* statistics." Although magic *p*-values are handy, decisions shouldn't be based upon magic numbers alone. The human component must be preserved.

The `ispiranteRanney` Package

General Information

The `ispiranteRanney` package will rely on three other packages for seamless operation. Those packages are:

1. ggplot2
2. scales
3. forecast

In most cases, only a few functions will be needed from each of the above packages. For example, `ggplot2` (Wickham 2009) and `scales` (Wickham 2014) will be used for all plots and only the `forecast()` (Hyndman 2014) function is used from the `forecast` package (Zeileis and Grothendieck 2005, Hyndman 2014, Team et al. 2014). Without these package dependencies, `ispiranteRanney` will not be able to operate successfully.

The `ispiranteRanney` package will include a number of functions to easily visualise how several metrics change as a function of day, date, priority level, agent, etc. In many cases, these insights may be negligible; for example, if all help-desk agents are of high quality and "know their stuff", it's unlikely that the number of reopens as a function of agent will provide much insight. However, if customer satisfaction scores are routinely low, then investigating how satisfaction score changes as a function of date or agent or a number of other variables may be important.

This section of the documentation will include several parts, one related to each of the major variables that will be used in the `ispiranteRanney` package: `callVolume`, `reopens`, `resolutionTime`, `satisfactionScore`, `waitTime`. In many cases, the `ispiranteRanney` package will plot these values against each other while in others, they will be plotted as a function of `priorityLevel`, `agent`, `dayOfWeek`, and `date`. In all cases, the visualization of the insight is completed by plotting one variable against another.

The functions used to gain insight into the data will be named consistently, normally as `Plot[y]By[x]()`. The naming consistency provides the means by which users—or software developers—will easily be able to identify what insights will be gained by any `ispiranteRanney` function. For example, a function named `PlotResolutionTimeByPriority.R` will do just that; plot the resolution time of a help-desk ticket as a function of the priority level.

As disclosed by Stefano Spada, a number of the columns in the sample data were `factors` rather than `numeric`. However, values that can be converted to `numeric`–like the `...resolution_time_in_minutes_within_business_hours` values can be redefined with `as.numeric()` and used in regression analyses. Similar to boxplots, the `aov()`, and `TukeyHSD()` test described above, numerical values can be compared against each other and the resulting relationship quantified.

A note on linear regressions

Regression is a stistical process for for estimating the relationships among variables. Like the boxplots discussed above, relationships between variables can be evaluated with certain statistical models. Boxplots evaluate the relationship between numeric and categorical data; regression models allow a statistician to evaluate the relationships among numeric data.

```r
#Siumulate data to use in a regression example
x <- 1:10
y <- x+rnorm(10, 0, .5)
```

```r
sim <- data.frame(x, y)
sim$Class <- "1"

x <- sample(1:10, 10, replace = F)
y <- x+rnorm(10, 0, 30)
sim1 <- data.frame(x, y)
sim1$Class <- "2"

#Bind the two data frames together into one
sim <- rbind(sim, sim1)

modRed <- lm(y~x, data = sim[sim$Class == "1", ])
modBlue <- lm(y~x, data = sim[sim$Class == "2", ])
```

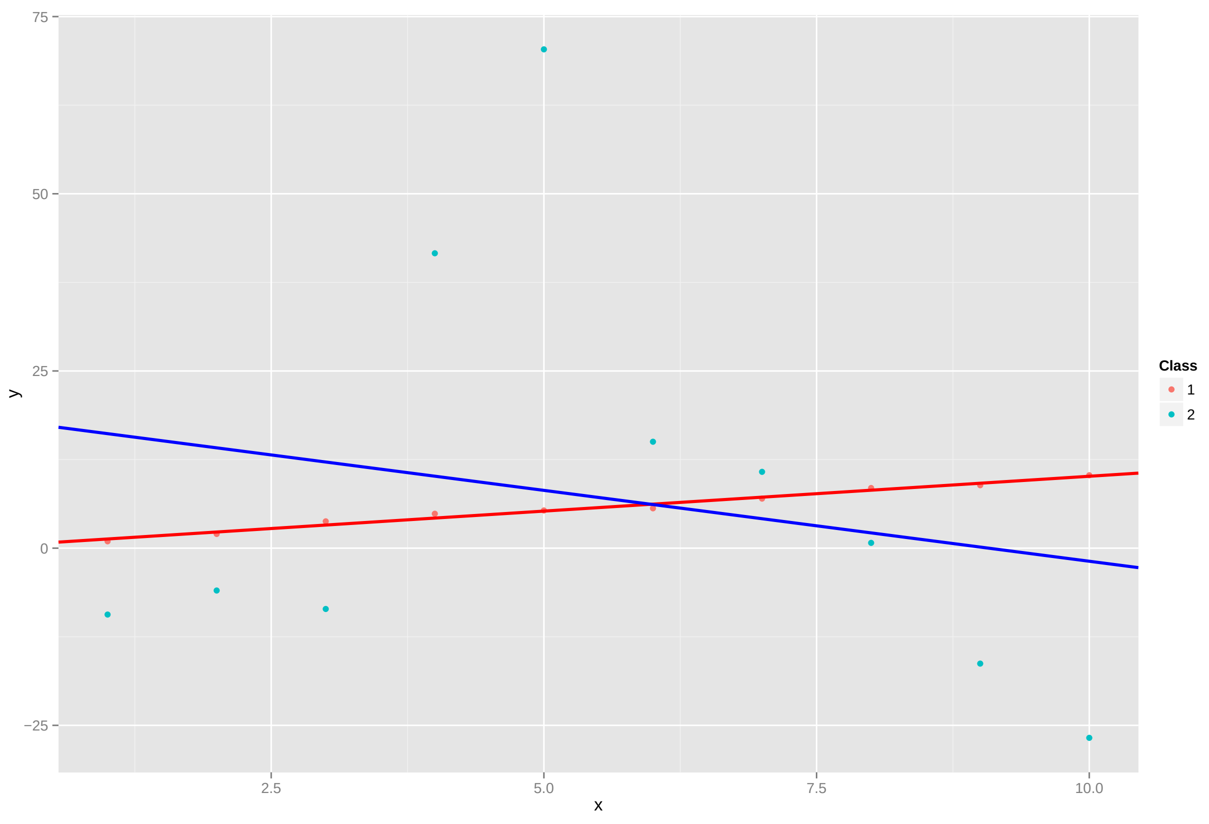


Figure 2: Linear regression of two data sets, one significant with p-value < 0.05 (red) and one with p-value $>$ 0.05 (blue)

The plot above was generated from simulated data. Note how closely the red line follows the red data points, especially compared to the blue line. That's a result of the relationship between the `x` and `y` variables:

```r
summary(modRed)
```

```
##
## Call:
## lm(formula = y ~ x, data = sim[sim$Class == "1", ])
```

```
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.59072 -0.28120 -0.05621  0.26738  0.61438
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.31028    0.28873   1.075    0.314
## x            0.98318    0.04653  21.129 2.65e-08 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.4227 on 8 degrees of freedom
## Multiple R-squared:  0.9824, Adjusted R-squared:  0.9802
## F-statistic: 446.4 on 1 and 8 DF,  p-value: 2.645e-08
```

```r
#P-value of the x variable
summary(modRed)$coefficients[2, 4]
```

```
## [1] 2.645383e-08
```

The *p*-value associated with the `x` variable the in the model is much less than 0.05, the "rule of thumb" for a significant value. CLearly, the red line follows the red points very closely; even without the *p*-value from the model, it's clear that there is a strong relationship between the red `x` and `y` variables.

The blue line and blue points appear to be much less closely related. We can infer from the 'scattershot' nature of the blue points that there is unlikely to be a strong impact of the `x` variable on `y`:

```r
summary(modBlue)
```

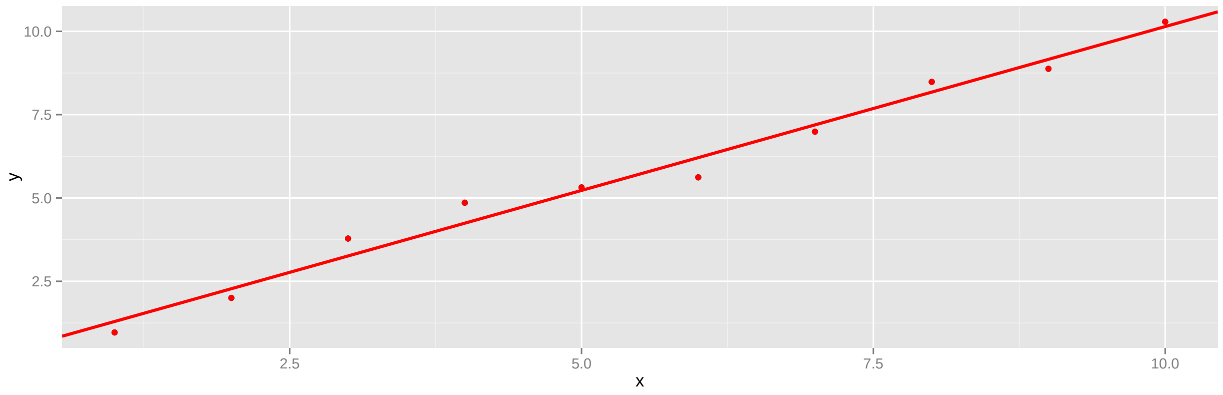
```
##
## Call:
## lm(formula = y ~ x, data = sim[sim$Class == "2", ])
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -25.521 -20.583  -8.925   8.300  62.225
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)    18.15      20.73   0.876    0.407
## x              -2.00       3.34  -0.599    0.566
##
## Residual standard error: 30.34 on 8 degrees of freedom
## Multiple R-squared:  0.0429, Adjusted R-squared:  -0.07674
## F-statistic: 0.3586 on 1 and 8 DF,  p-value: 0.5659
```

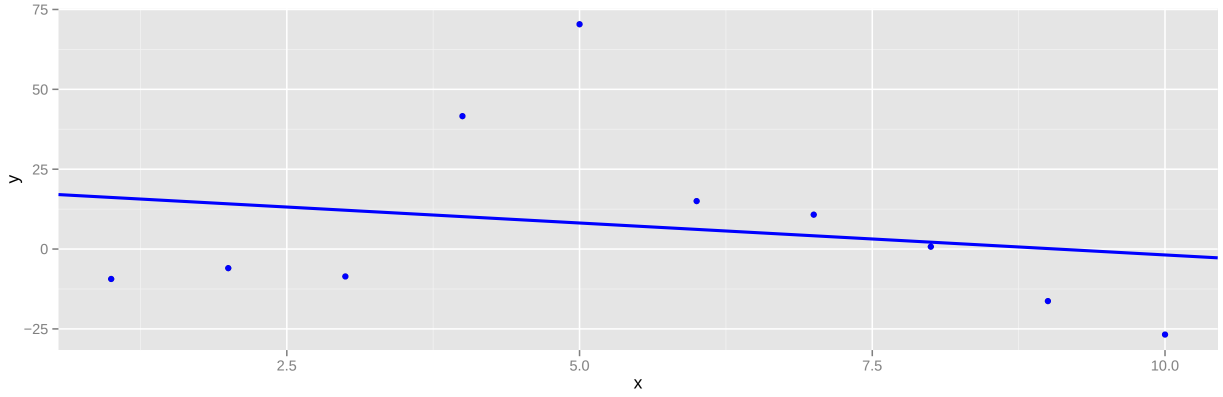
```r
#P-value of the x variable
summary(modBlue)$coefficients[2, 4]
```

```
## [1] 0.5658754
```

the p-value of the blue `x` on `y` is $>$ than 0.05, implying that there is not much of a relationship between the two.

Looking at the plots together makes it difficult to see the slope of the red line; the variability in `y` of the blue points expands the red scale, preventing us from seeing the slope. In looking at them separately:





it's much clearer that there is a strong relationship between the red `x` and `y` variables but not the blue `x` and `y`. While the *significance* can only be calculated by comparing the p-value to whatever α (the critical value, normally 0.05) we chose, the strength of the relationship can also be visually examined, especially within the context of the randomness of the data points.

Initial statistical considerations led me to think that the `ispiranteRanney` package will *not* include the actual statistical analyses from linear models presented in these analyses. Unless customers and users have a deep background in statistical analysis, the parameter estimates, standard errors, and p-values that result from a `summary()` call to a model can be mis-interpreted. As a result, similar to p-values that stem from an `ANOVA` (as identified in the *A Note About Box Plots*), only a red "trend" line will be provided when a numeric value is plotted as a function of another numeric value. Users can clearly see–without a p-value to cloud their judgement–the relationship between the `y` variable and the `x` variable. If the line trends up or down in a shallow manner, the relationship can be assumed to be relatively small. Conversely, the steeper the line becomes, the stronger the relationship.

If quantifiable relationships are required, p*-values, R^2 values, and other metrics can be provided.*

Resolution Time

Resolving help-desk ticket quickly is a key component in keeping customer satisfaction levels high. In the help-desk ticket sample data provided by Stefano Spada, there were a numer of values provided for resolution time. The `ispiranteRanney` package will focus on only two values:

- `first_resolution_time_in_minutes_within_business_hours`
- `full_resolution_time_in_minutes_within_business_hours`

The default resolution time value for all functions will be the `first...` value. However, users will be able to select the `full...` value for all visualizations.

There will be five functions that allow users to evaluate resolution time. Functions allow users to visualize resolution time by assignee name, date, day of the week, priority level, and the number of reopens of a ticket.

- `PlotResolutionTimeByAgent()`
- `PlotResolutionTimeByDate()`
- `PlotResolutionTimeByDay()`; day of week will be assigned by a separate, in-package function, `ConvertSatisfactionScoreToNumber()`
- `PlotResolutionTimeByPriority()`
- `PlotResolutionTimeByReopens()`

Resolution time is a key variable in identifying how long it takes a help desk to resolve support tickets. The functions listed here will allow customer service managers to understand how `"full"` or `"first"` resolution time within business hours fluctuates as a function of `agent`, `date`, `day`, `priority`, and the number of `reopens`.

Resolution time likely also plays a key role in customer `satisfaction_score`, another relationship explored in a different proposed function. Regardless of the relationship that resolution time has on any other help-desk ticket item, faster resolution times would increase the number of tickets that help-desk staff could resolve.

Satisfaction Score

Customer satisfaction score is a powerful measure of how satisfied a customer is with their help-desk experience. In many cases, `satisfaction_score` is not provided by the customer. In cases where satisifaction score *is* returned, it can be quite useful.

`ispiranteRanney` will have a number of ways to visualize how `satisfaction_score` changes with other variables. In the `ispiranteRanney` package, `satisfaction_score` will be converted to a numeric variable by the `ConvertSatisfactionScoreToNumber()`. This function is called "behind the scenes" so a user would never have to use it—or even know it is there. The function converts factor variables (e.g., "Good" and "Bad") into numeric variables:

Satisfaction Score	Number
Good	4
Bad	1
Not Offered	NA
Offered	NA

The sample datasets provided had these unique values for `satisfaction_score`; as a result, these were the only scores that could be converted to a numbers. If other help-desk ticket datasets generate additional score values, they could be incorporated into the functions below simply.

As in the `ResolutionTime` section above, `satisfaction_score` will be compared to other variables to understand how `satisfaction_score` fluctuates:

- `PlotScoreByCallsPerPerson()`
- `PlotSscoreByCallVolume()`

- `PlotScoreByResolutionTime()`
- `PlotScoreByWaitTime()`

These funtions will all be visualizations of `numeric` values. `CallsPerPerson`, `CallVolume`, `ResolutionTime`, and `WaitTime` will all be converted to `numeric` values. `satisfaction_score` will be regressed against the above variables with `lm()`; the resultant plot will be similar to the plots in `A note on linear regressions`.

`satisfaction_score` is an important metric by which to measure customer satisfaction. In the sample data, only one dataset included much `satisfaction_score` data; in other example data, `satisfaction_score` data was sparse. Help-desk managers should stress the importance of customers offering feedback to help-desk employees so that those `satisfaction_score` data can be used in evaluating the effectiveness of the help desk.

In all cases, our statistical hypotheses are that `satisfaction_score` would *decrease* as:

1. the number of calls per person increased;
2. call volume increased;
3. resolution time increased and;
4. wait time increased.

With the sample data provided, bits of these hypotheses play out in testing. In one case, the line was effectively parallel, suggesting no relationship while in another we had a spurious result likely generated from a very unhappy customer (very low wait time and low `satisfaction_score`).

Number of Reopens

The number of reopens of a ticket is the number of times that a "solved" ticket has been reopened. Reopens likely have a negative affect on a number of other numeric values (e.g., `satisfcation_score` and resolution time, discussed previously). However, simply understanding how the number of reopens changes with different categoriacal variables can help a manager understand how the number of reopens changes by `agent`, `call volume`, `date`, `day`, and `priority` level.

The number of reopens of a help-desk ticket will be visualised with functions:

- `PlotReopensByAgent()`
- `PlotReopensByCallVolume()`
- `PlotReopensByDate()`
- `PlotReopensByDay()`
- `PlotReopensByPriority()`

Plotting the reopens by `assignee_name` (i.e., `Agent`) will help customer-service managers understand how efficient each help-desk agent is. Boxplots of the reopens by agent will indicate which agents routinely resolve tickets the first time (i.e., `reopens = 0`) and which agents require more time (i.e,. `reopens > *0*`). Visulalizations will provide a horizontal line of the mean number of reopens for the entire dataset as well as the boxplot by agent. The horizontal line will be calculated from the entirety of the dataset and used for comparing individual reopens to the over all dataset.

by plotting the number of reopens as a function of call volume, `ispiranteRanney` will model the relationship with `lm()`. Managers will be able to see trends in the data. Questions like *if call volume goes up, what happens to the number of reopens?* can be answered. By plotting `reopens` as a function of call volume, managers can see for themselves if things get "missed" when help-desk agents are busy. From that, managers can determine staffing levels and, if necessary, have agents "on-call" to help out.

In the help-desk ticket data, `date` is effectively a categorical variable. As a result, while we can't create a linear model of the number of reopens as a function of date, we can plot boxplots and look for differences by date.

Reopens by date is an important relationship to evaluate because by looking at call volumes, agent effectiveness, and reopens by date, a customer service manager can start to put together a full picture of how effective and efficient an agent, a group of agents, or their entire help desk can be.

Wait Time

Wait time of customers calling into a help desk may have a negative affect on satisfaction score. To understand how wait time is affected by variables in the help-desk data, `waitTime` will be regressed as a function of agent, day of the week, priority level, call volume, and any other variables that assist in understanding how wait time fluctuates.

As a numeric variable plotted against categorical variables, users will be able to visualize and unerstand how the categorical variables affect wait time. For example, if wait time is consitently high on Fridays, customer-service managers can adjust staffing schedules to provide more help-desk agents on Fridays. Alternatively, if wait time is close to or at zero on other days of the week, there may be an over-abundance of help-desk agents available.

In conjunction with other available plots in the `ispiranteRanney` package, customer-service managers will be able to understand how univariate models affect variables important to customer satisfaction.

Call Volume

Call volume, unlike many of the previous ordinate variables, likely has little or nearly no affect on customer satisfaction score. However, if call volumes are high, help-desk agents may be less likely to resolve customer issues quickly. High call volumes likely:

1. increase resolution time;
2. lower customer satisfaction scores;
3. increase the number of reopens of a given ticket, and;
4. increase the wait time.

Call volume will be provided in visualizations by date and by day of week. Because call volume can fluctuate so drastically throughout the year, the sample data provided little ability to determine how call volume is impacted by other variables. Rather, call volume provides the means to generate insight in a number of other issues. For example, when evaluating `ResolutionTimeByAgent`, managers may find it helpful to visualize what the call volume of a given agent was on that day. Because that visualization would be helpful, call volume will be provided when other insights are investigated.

```
## NULL
```

In this example, a customer-service manager can see that some agents have very low resolution times while others are much higher. In an effort to identify if high resolution times are a function of call volume, it's easy to see that some agents have very high call volumes with average resolution times (i.e., `Assignee_name` == `Connor Hays`). This kind of insight is useful for managers to identify who are their top performing agents, who may benefit from additional training, and those that could help others.

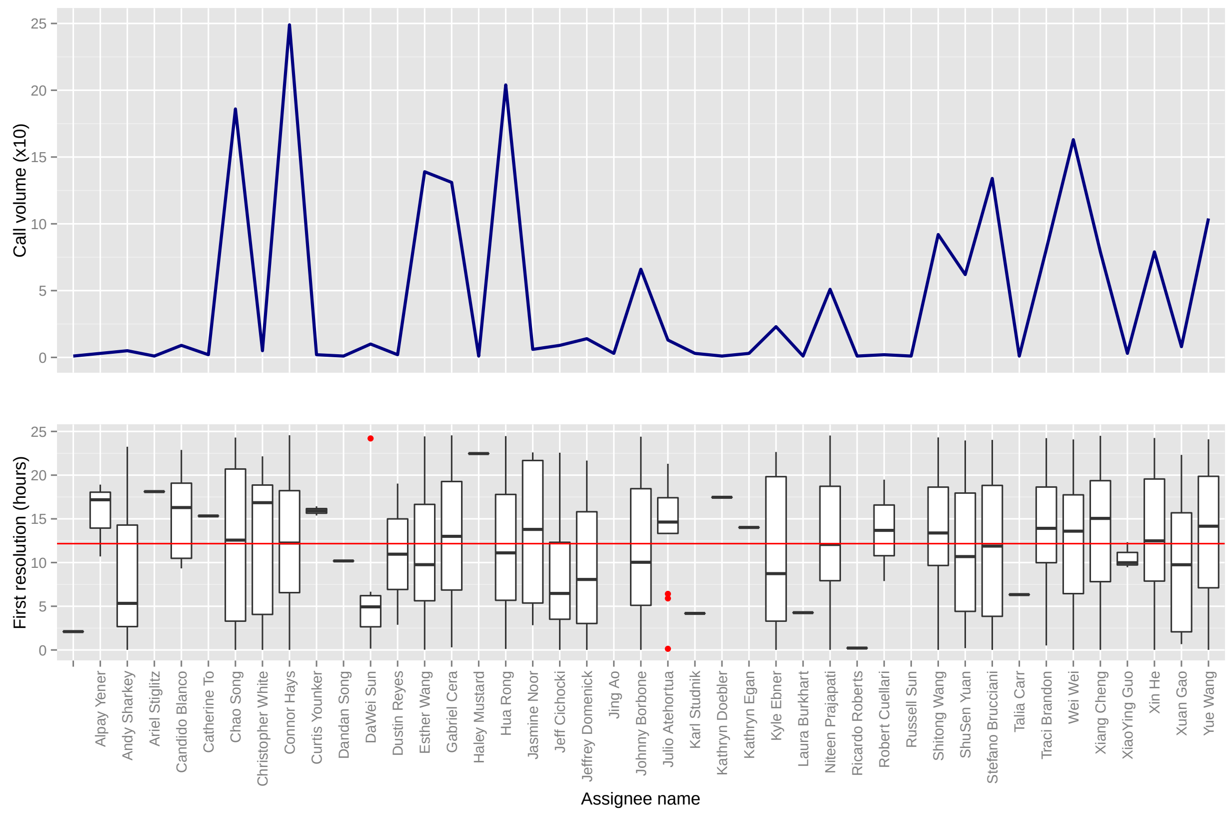


Figure 3: Upper plot of call volume by agent and lower plot of resolution time by agent. From these data, customer-service managers can determine if the resolution time of an `assignee` fluctuates as a result of call volume. For example, the assignee `Connor Hays` has an average resolution time value but his call volume was particularly high. This may be an indication that `Connor Hays` could be used in a training capacity for other agents

Forecasting Help-Desk Calls

Predicting the future is difficult. However, with timeseries and the `forecast` package (Hyndman 2014), valid statistical methodology can be applied to approximate how many calls a help-desk may receive x days in the future. For example, in some of the sample data provided, the data was yearly, ranging from `2014-01-01` through `2014-11-29`, for example. The `ispiranteRaney` package will use an `ARIMA` model (Autoregressive Inegrated Moving Average, (Hyndman 2014)) to predict the number of calls a help-desk will receive for as many days into the future as a manager would like.

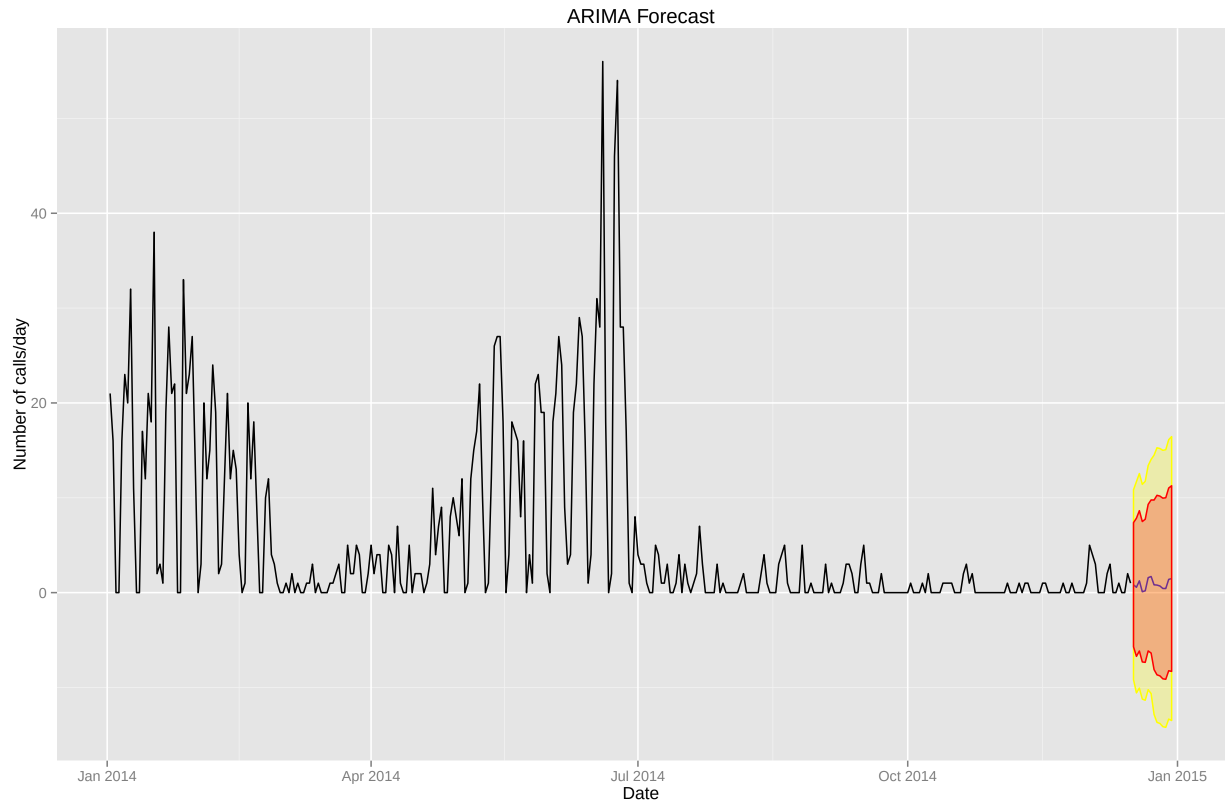


Figure 4: Forecast values for the number of help-desk calls received two weeks from the end of the dataset.

`ForecastHelpDeskCalls()` will use a 14-order model (e.g., a 14-day moving average) to predict the number of calls received. Dates that do not appear in the dataset will be assumed to have zero calls. With the function, users could easily predict out to whatever date they are interested, though, if data is sparse towards the end of the dataset, predictions will be of little value.

`ForecasthelpDeskCalls()` will be able to predict out as far or as short as the user would like. Indeed, narrow temporal windows would likely provide the most utility. However, monthly, quarterly, or even yearly predictions are possible. Regardless, like any statistical model, models are only as good as the data they are based on. Predicting a full quarter from the final date in the dataset may not provide much insight:

```
#Predicting for the next quarter, from whatever is the final date of the dataset
ForecastHelpDeskCalls(dF, 365/4)
```

Additional Proposed Functions

Additional *behind the scenes* functions for data manipulation include:

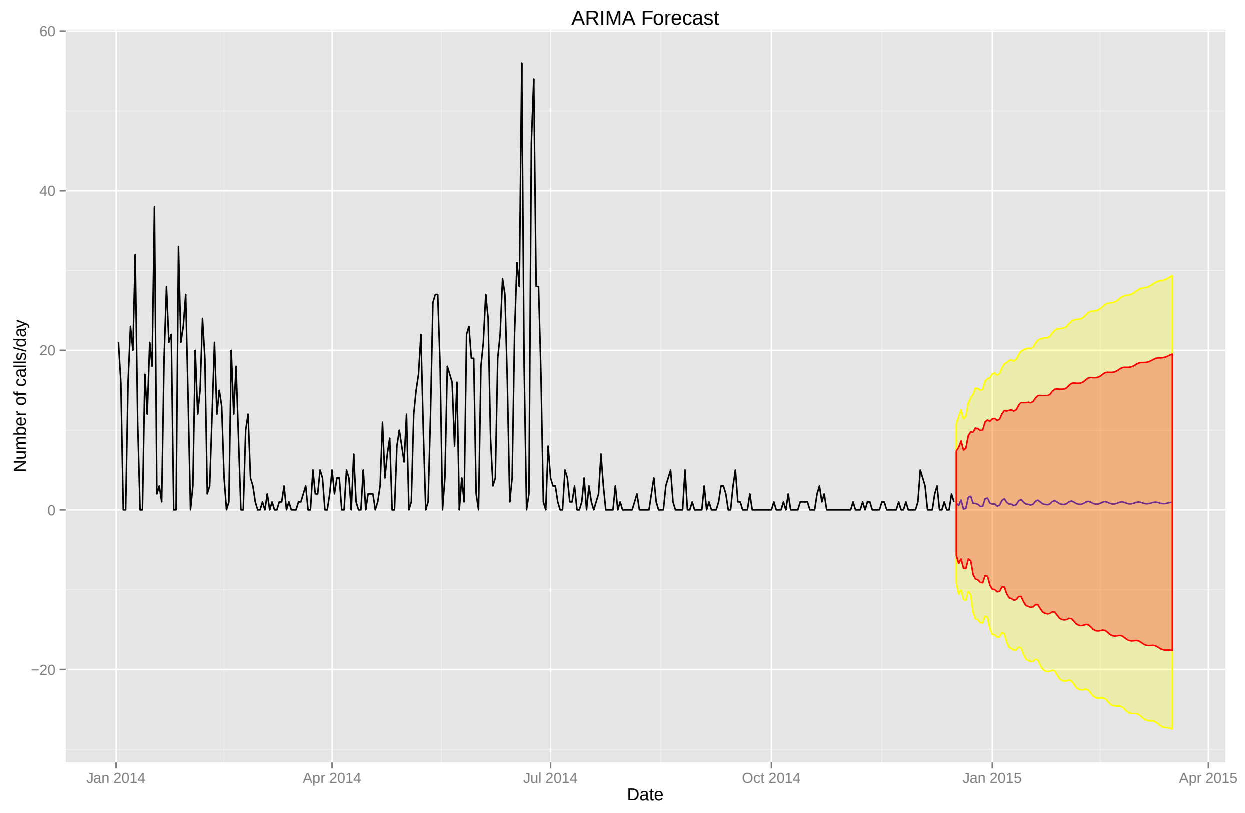


Figure 5: Forecast values for the number of help-desk calls received in the next quarter from the end of the dataset.

- `AssignDateAndDay()`, a function to convert either the `"assigned"` or "`created_at`" dates into the days of the week for use in higher-level functions that plot variables as a function of `day`;
- `ConvertSatisfactionScoreToNumber()`, discussed above, designed to convert character satisfaction values of `"Good"` or `"Bad"` to numeric values;
- `PlotMultipleObjects()`, a function to plot to `ggplot` objects in the same window. This function could easily be called to plot any two visualizations on top of each other, if this is a preferred feature.

These functions would never need to be accessed by the user. They are called from within the higher level functions on the "don't repeat yourself" principle.

Total Estimated Cost

The estimated cost for creating the `ispiranteRanney` functions, detailed help files, and final packaging into a `.zip` file is:

Item	Hours	Cost
Writing functions and documentation	50.00	$4250.00
Total	**50.00**	**$4250.00**

Estimated costs include data exploration, authoring and testing of the functions necessary to make analysis and visualization easy, and writing help files. Packaging of the final `R` file can be completed both through the standard `R CMD` procedure, installation through an online repository, or both.

References

Anderson, D. R. 2008. Model based inference in the life science: A primer on evidence. Springer, New York, NY.

Burnham, K. P., and D. R. Anderson. 2002. Model selection and multimodel inference: A practical information-theoretic approach, 2nd editions. Springer, New York, NY.

Day, R. W., and G. P. Quinn. 1989. Comparisons of treatments after an analysis of variance in ecology. Ecological Monographs 59(4):433–463.

Hyndman, R. J. 2014. Forecast: Forecasting functions for time series and linear models.

R Core Team. 2014. R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria.

Team, R. C., D. Wuertz, T. Setz, Y. Chalabi, M. Maechler, and J. W. Byers. 2014. TimeDate: Rmetrics - chronological and calendar objects.

Wickham, H. 2009. Ggplot2: Elegant graphics for data analysis. Springer New York.

Wickham, H. 2014. Scales: Scale functions for graphics.

Zeileis, A., and G. Grothendieck. 2005. Zoo: S3 infrastructure for regular and irregular time series. Journal of Statistical Software 14(6):1–27.