The `ispiranteRanney` Package

Steven H. Ranney

Contents

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

Introduction

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 nad intercept values, parameter estimates, R^2 values) are often misinterpreted by those that posess none or little statistical training. In some cases, reliance on the output from stastical analyses may "cloud" a person's judgement; lack of training may let someone infer from a *significant* p-value that a relationship exists when, in reality, the results were a result of spurious data.

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. I will write the package—and the functions—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, 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 satisfcation score changes as a function of resolution or wait time, for example.

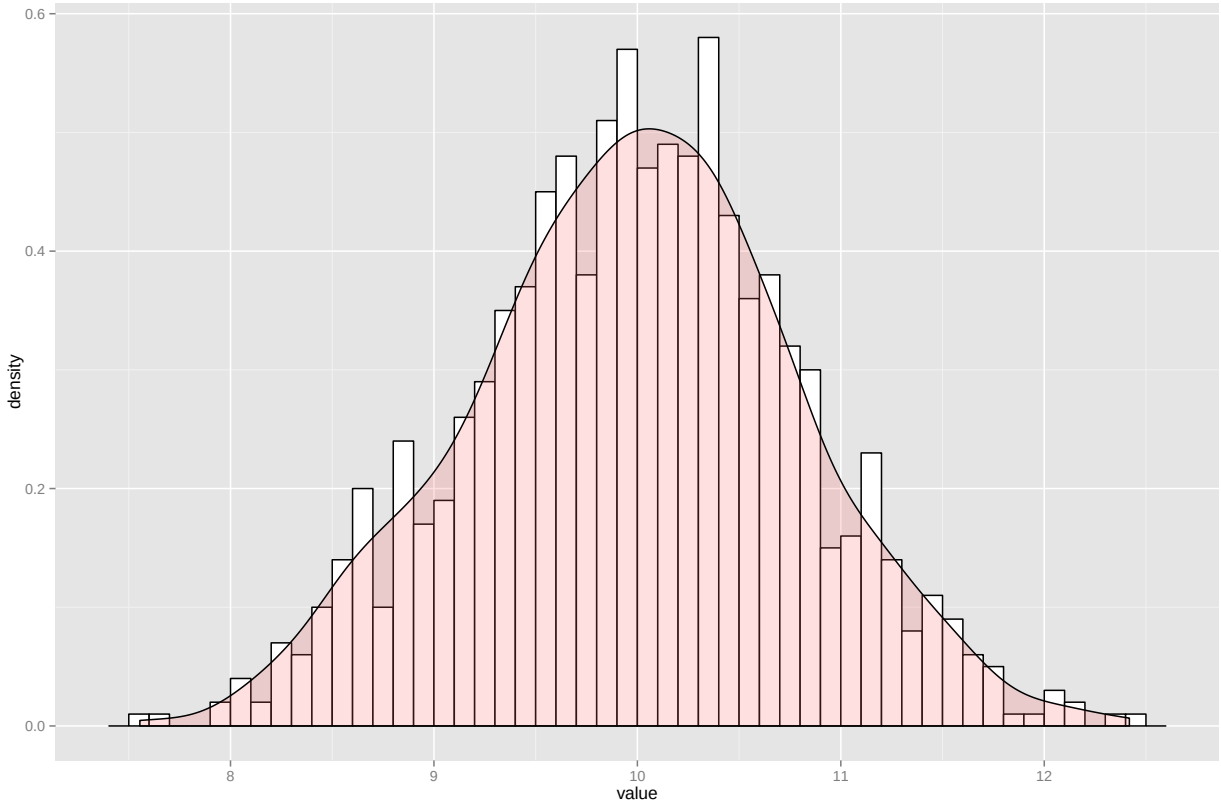
A note about R and plots

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

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

```
library(ggplot2)
# Histogram overlaid with kernel density curve
ggplot(tmp, aes(x=value)) +
    geom_histogram(aes(y=..density..), # Histogram with density instead of count on y-axis
                   binwidth=.1,
                   colour="black", fill="white") +
    geom_density(alpha=.2, fill="#FF6666")  # Overlay with transparent density plot
```

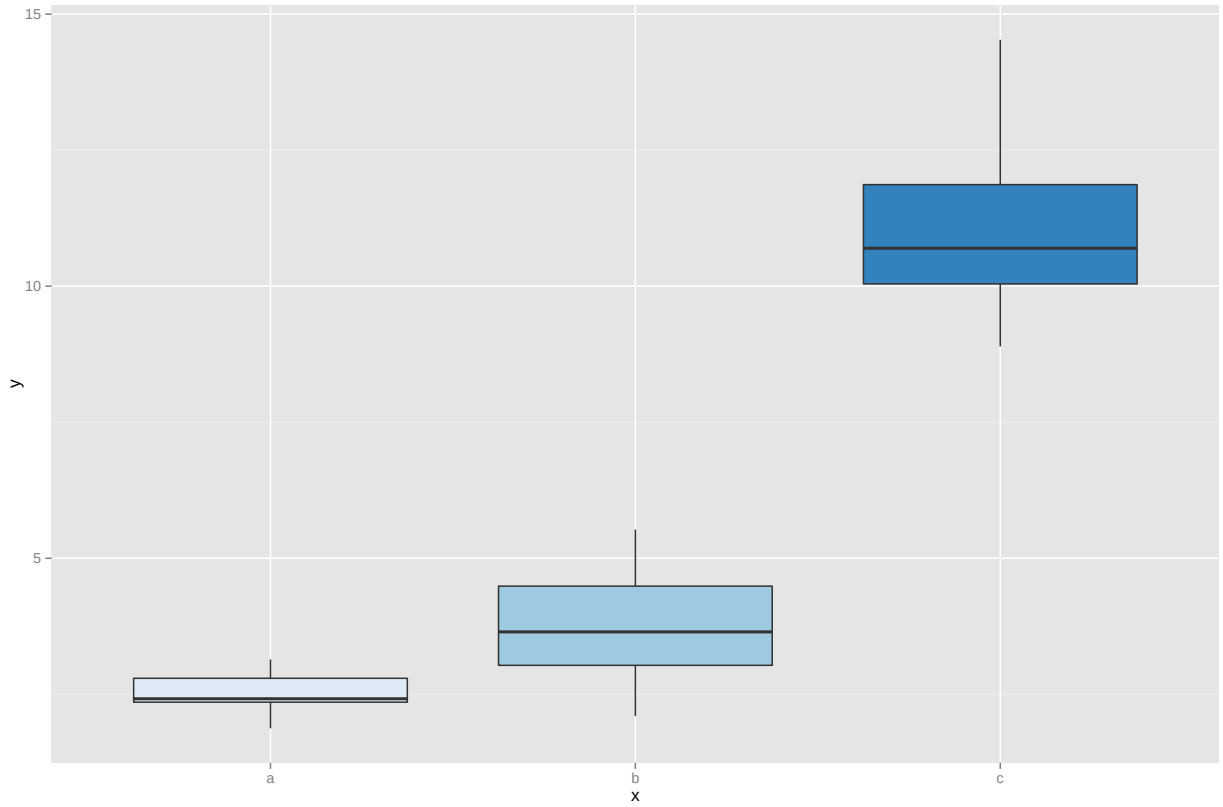


A note about boxplots

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

```
#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")
```



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` and both `a` and `b` appear to be quite different from `c`, but 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  303.0  151.48   86.01 6.14e-10 ***
## Residuals   18   31.7    1.76
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

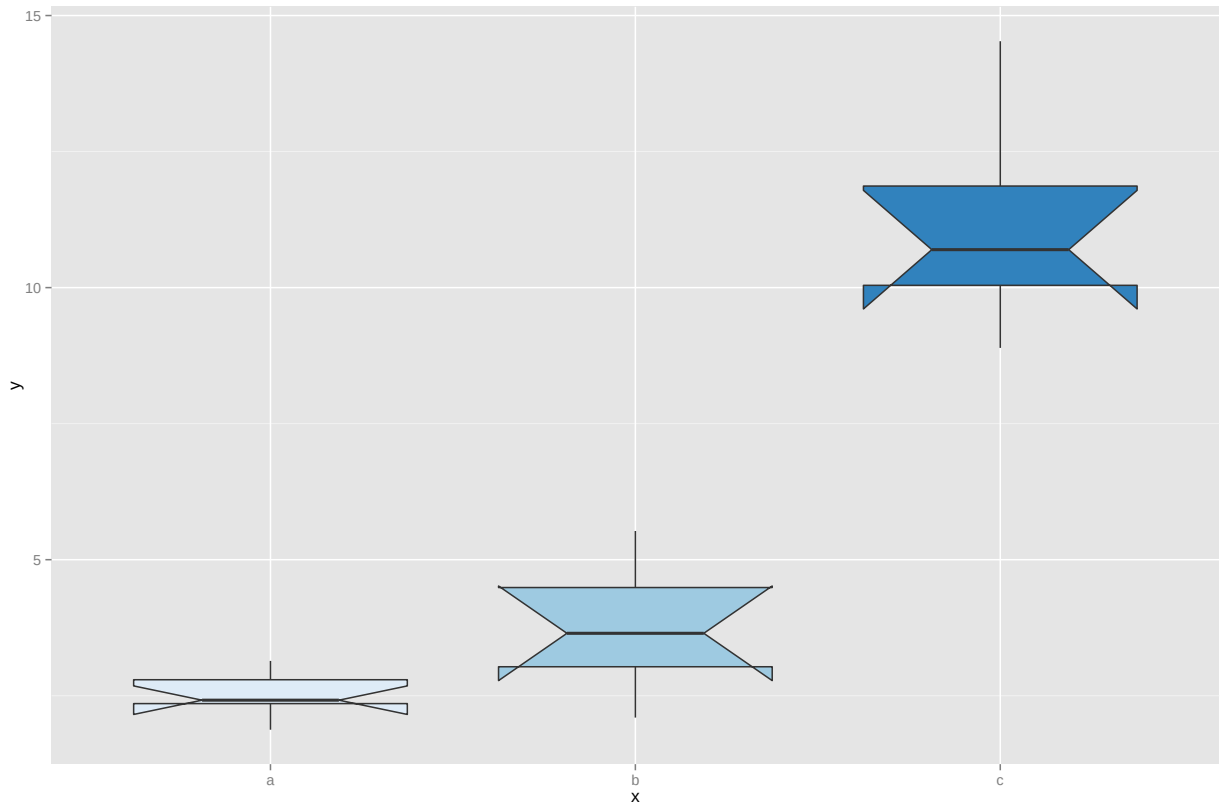
The results from the `summary(mod)` statement show us that the categorical `x` variable has a significant difference on the `y` variable. The three asterisks next to the `Pr(>F)` value in the right column tells us that the p-value is < 0.05. (A p-value < 0.05 is the 'rule of thumb' for a significant value.) Even though we know that `x` has a significant affect on `y`, we don't know where the 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.

```
#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 1.225945 -0.5845037  3.036395 0.2221883
## c-a 8.600140  6.7896903 10.410589 0.0000000
## c-b 7.374194  5.5637449  9.184643 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). Box and whisker plots in `R` allow a user to add a notch into the boxes. When `notch = TRUE`, `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 notch from `a` overlaps with the notch from `b` 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 box plot 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 be a quantification of 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, and 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 should't be based upon magic numbers alone. The human component must be preserved.

The `ispiranteRanney` Package

General Information

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

- ggplot2
- scales
- forecast

In most cases, only a few functions will be needed from 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 priority level will provide much insight. However, if customer satisfaction score is 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. 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 engineers–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 ticked as a function of the priority level for that ticket.

As disclosed, 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

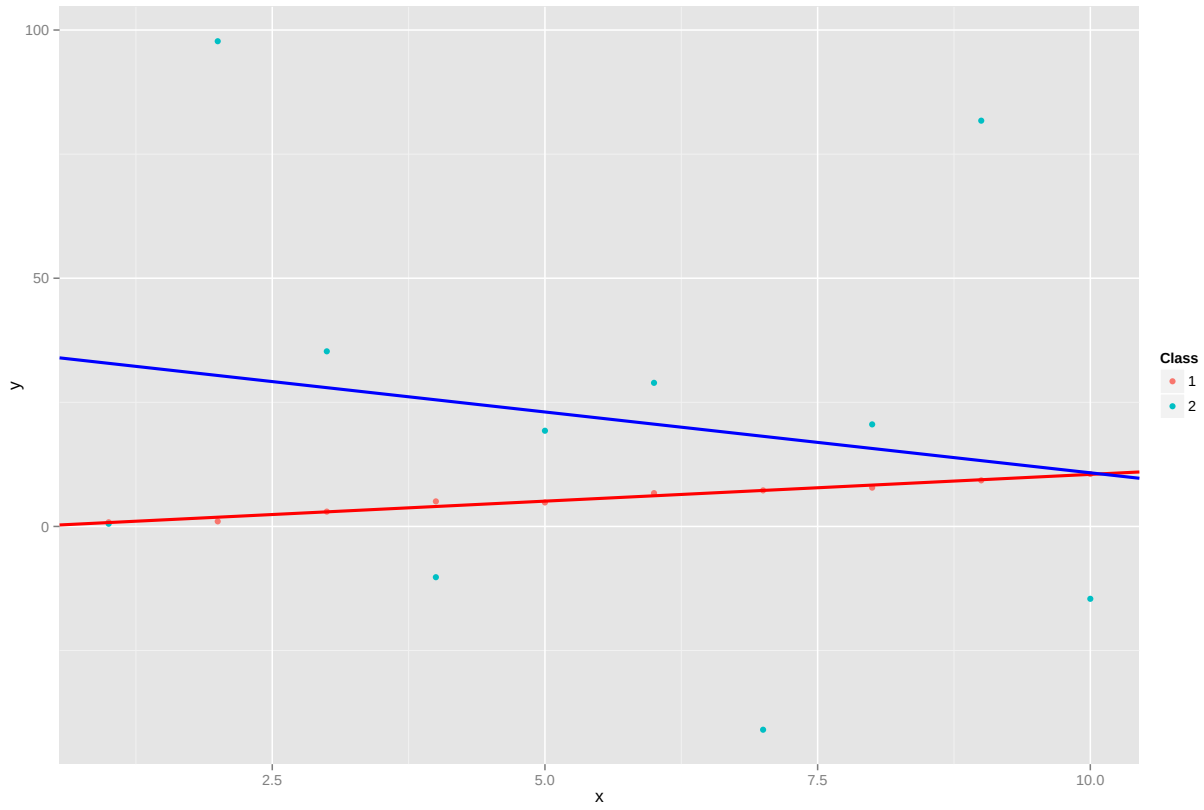
Regression is a stistical process for for estimating the relationships among variables. Like the boxplots discussed above, relationships between variables can be evaluated with certain statistical models. While boxplots evaluate the relationship between numeric and categorical data, regression models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

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

x <- 1:10
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", ])
```



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

```
summary(modRed)
```

```
##
## Call:
## lm(formula = y ~ x, data = sim[sim$Class == "1", ])
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.83729 -0.23079  0.02871  0.06642  1.02862
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -0.29629    0.37493   -0.79    0.452
## x            1.07834    0.06042   17.85 9.96e-08 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.5488 on 8 degrees of freedom
## Multiple R-squared:  0.9755, Adjusted R-squared:  0.9724
## F-statistic: 318.5 on 1 and 8 DF,  p-value: 9.956e-08
```

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

```
## [1] 9.956324e-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 here.

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`:

```
summary(modBlue)
```

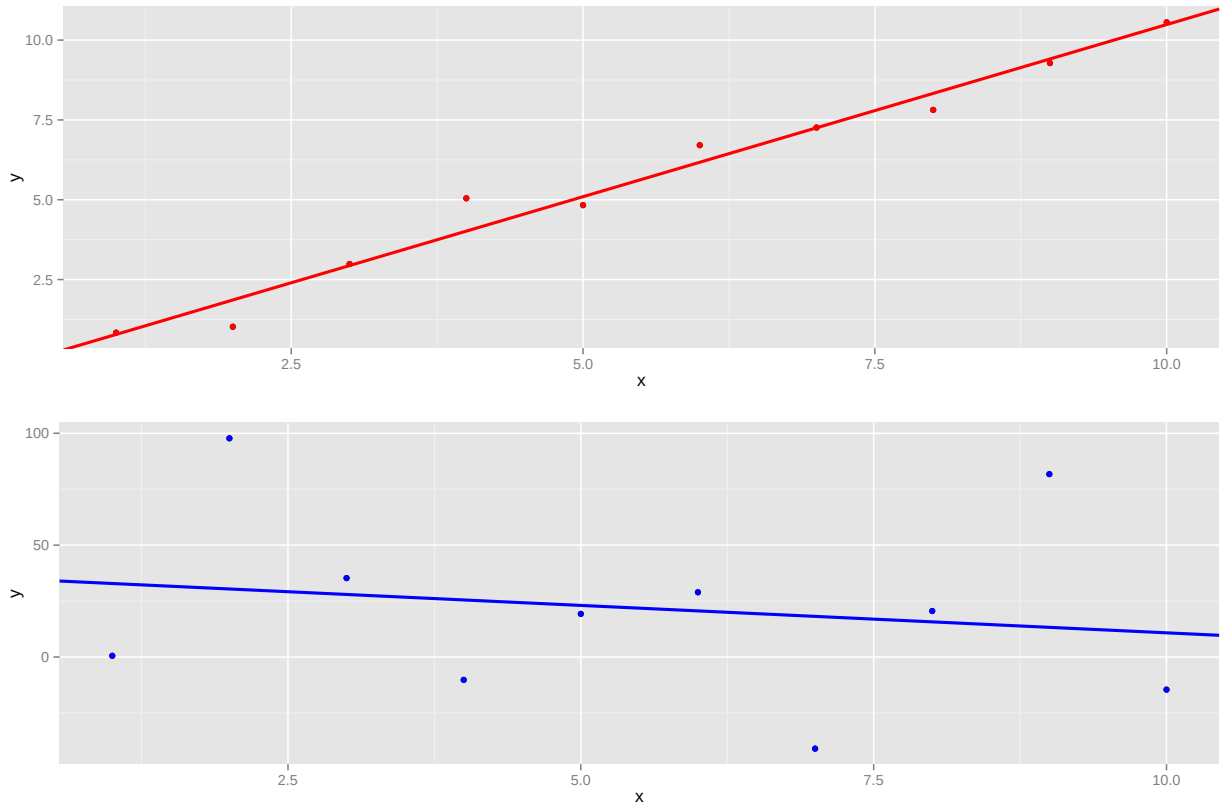
```
##
## Call:
## lm(formula = y ~ x, data = sim[sim$Class == "2", ])
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -59.109 -30.581   0.536   8.072  68.485
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)    35.31      30.40   1.161    0.279
## x              -2.45       4.90  -0.500    0.631
##
## Residual standard error: 44.5 on 8 degrees of freedom
## Multiple R-squared:  0.03031,    Adjusted R-squared:  -0.09091
## F-statistic:  0.25 on 1 and 8 DF,  p-value: 0.6305
```

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

```
## [1] 0.6305188
```

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 condenses 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 we chose, the strength of the relationship can also be visually examined, especially within the context of the randomness of the data points.

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)
- `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 provide the means for 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 is 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 were 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 the regression lines. 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.

PlotReopensByAgent()

`PlotReopensByAgent()` needs only the data frame argument to work. However, like many of the other functions previously discussed, the `"value"` argument can be provided with either `assigned` or `created`.

`PlotReopsnByAgent()` produces two plots in one plot window. The top plot is a plot of the total number of calls for each agent over the entirety of the dataset. The lower plot is a box plot of the number of reopens for each agent. Here, the red line is the mean number of reopens across the entirety of the data set nad the red points are outliers for each agent name (see `?PlotReopensByAgent()` for more information).

Here, it appears as though most agents have median `reopen` values that are 0, very close to 0, or at least less than the mean number of reopens for the entire dataset. This plot is insightful, however, because agents with a high number of reopens can be identified along with the number of calls they took over all of the data. For example, `Connor Hays`, even though he took close to 250 calls over during the time covered in the data set, still has a median number of reopens close to 0. This suggests then that `Connor Hays` is an efficient work who knows how to resolve customer issues. Meanwhile, `Christopher White` may still be in training as his range of `reopens` is quite high yet his call volume (top plot) is close to zero, implying that he may still be learning how to resolve customer issues.

PlotReopensByCallVolume()

`Reopens` and `call volume` are both numeric values. As a result, we can plot them against each other and evaluate their relationship with a linear model. Like other functions that deal with dates, the date argument `"value"` can be added here though the default is `assigned`.

As we would expect, as call volume increases, the red linear regression line suggests that the number of reopens increases, too. The reason for this is that at higher call volumes, there are no zero values for reopens. From this, it's clear that at higher call volumes, things get "missed" while resolving customer issues. As a result, it may be worth while to insist that help-desk agents take as much time with customer issues as they do during lower call-volume days.

PlotReopensByDate()

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. The `PlotReopensByDate()` requires only one argument, the data.

Like previous temporally-broad boxplots, this looks messy. Like other date functions discussed above however, we can limit the date range of these data. Lets look at just the week of June:

Clearly, there aren't many tickets reopened during the course of this week. From the looks of these data, there was only one ticket that was reopened on `June 2`, `June 4`, and `June 5`. The date of `June 1` likely had no calls thus resulting in, effectively, no data being shown for that date.

This 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.

PlotReopensByPriority()

Total Estimated Cost

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

Item	Hours	Cost
Data exploration and function writing	10	$850.00
Writing functions and documentation	35	$2975.00
Total	**45.00**	**$3825.00**

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.

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

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.