Team DAY Project

Dustin Bracy, Adam Ruthford, Yang Zhang

1/23/2020

##### name

# Introduction

Gilsbar offers a comprehensive reporting package to both its internal employees and external clients. As our user base has grown over the last several years, we have seen an increased demand for reports and consequent load on report servers. Report run times vary significantly, sometimes reports are returned in minutes, and in the worst case: hours. In order to mitigate server load and improve report response time, we would like to better understand factors affecting delivery time, as well as to be able to provide our users an estimated delivery time of reports. We would like to answer the following questions about our data:

1. What is the expected average delivery time of a new report?
2. What are the best & worst times of day, week, or month to run a report?
3. Is there a difference in performance between report servers?
4. What are the biggest impacts on performance of reports?

## Data Description

Gilsbar has provided report data spanning a 12 month period from January 2019 through January 2020. The available parameter list is very large, having over 500 parameters used in the past year. In order to narrow the scope of the project, we focused on completed reports only, and determined the parameters show below would be most useful in predicting report run time. A description of other, non-utilized fields is available below in the appendix. Also listed there is a crosswalk provided which groups like-type reports into 11 categories.

## Approach Methodology

We’d like to tackle these questions using two separate models for interpretation. For our first objective, we use Multiple Linear Regression techniques to predict run-time using several of the report metrics and features. As our second objective, we will utilize two-way ANOVA techniques to compare ReportDeliveryTime vs Server, Report Category, and Report Format.

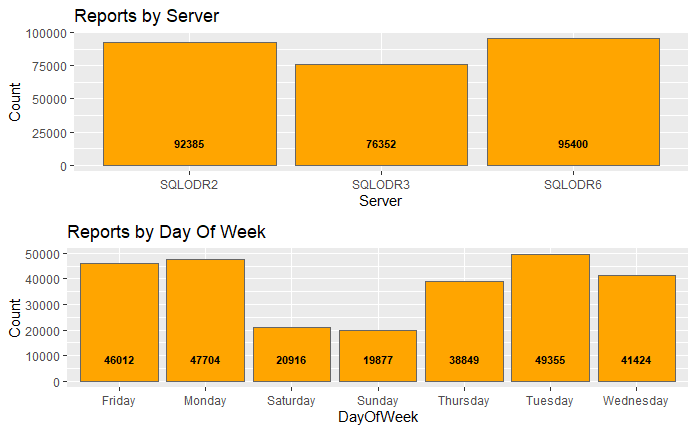
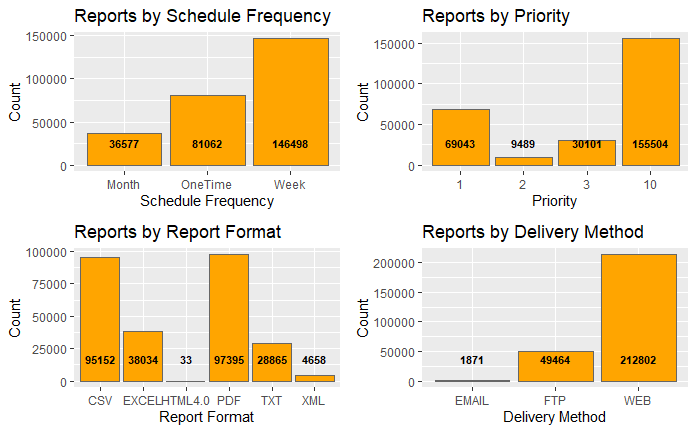
# Exploratory Data Analysis

Our EDA analysis revealed that there are several categorical variables that don’t show an apparent linear relationship with ReportDeliveryTime. Also, we found that our response variable, ReportDeliveryTime, is heavily right skewed and likely needs a log transformation in order to continue analysis.

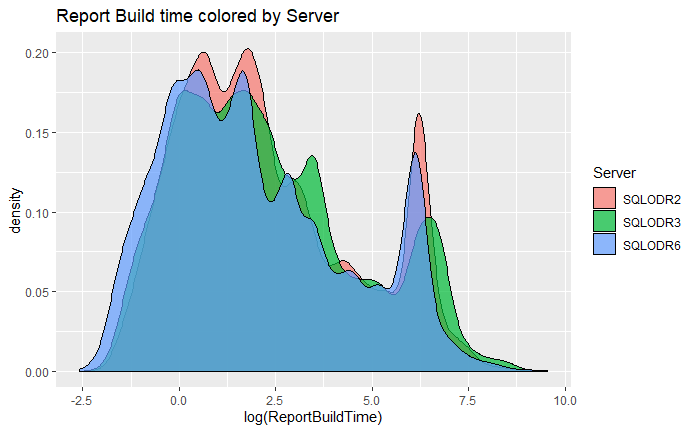
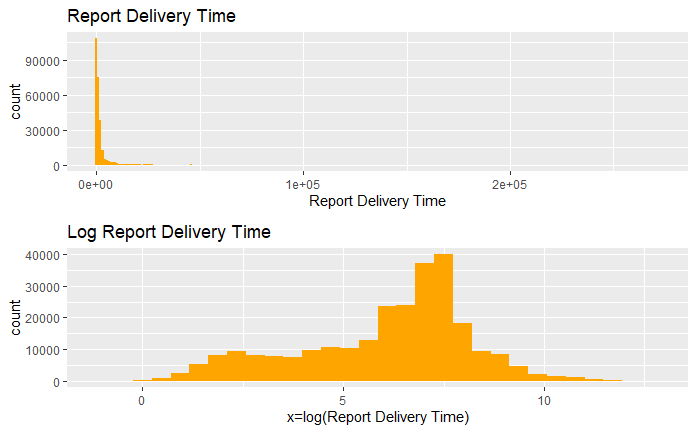
Given some additional knowledge of the data-set, we learned that only three production servers are useful for prediction (SQLODR2, SQLODR3 and SQLODR6) and that the ReportID (Specific Report) may be related to ReportBuildTime. To explore this, we converted the ReportID to factors and conducted simple linear regression, giving us an R^2 of 0.79 which is very good, however the ReportID has over 1500 factor levels which make the explanation difficult when introducing additional features, so we continued to explore other features and temporarily ignore ReportID.

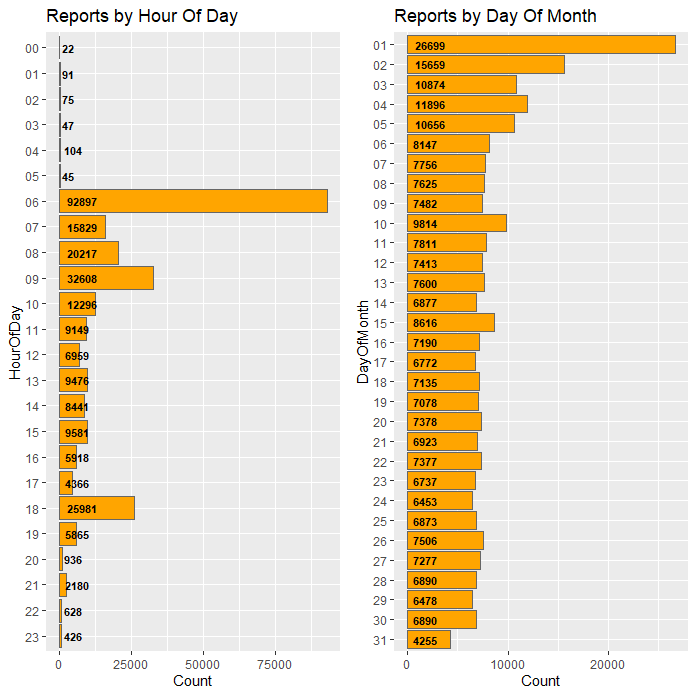
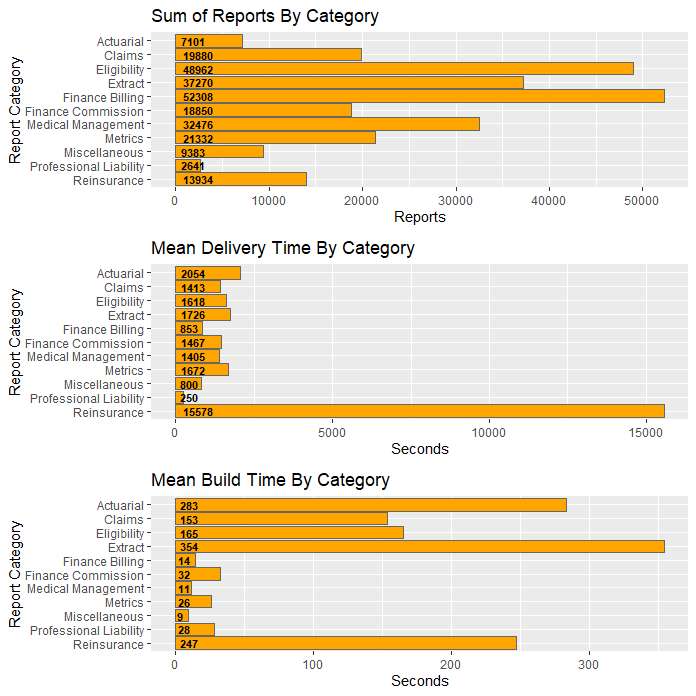
Our EDA resulted in the following additional observations:

* Submission times are most often 6am, with 9am and 6pm as secondary peak times.
* The first day of Month is peak report day
* Scheduled frequencies occur more often by Weekly, One-time, Monthly respectively.
* Reinsurance reports grossly exceed delivery times of all other reports, PL reports deliver fastest.
* Extract reports take the longest for servers to build, with miscellaneous building quickest.
* Finance/Billing and Eligibility reports outnumber others, and PL reports are run the least.(Note: PL Category is newest, so does not contain a full year worth of data).
* Most reports utilize Delivery Method: Web.
* Production reports far outnumber test reports.
* 3 Servers processed most reports are SQLODR2, SQLODR3, SQLODR6.
* SQLODR2, SQLODR3, and SQLODR6 are all marked as production, while the others are marked as test.
* Popular Report formats are CSV/PDF followed by TXT/EXCEL.
* Reports are generated more on WeekDays with most occurring on Monday/Tuesday.
* Priorities only have 0.1, 2, 3, and 10, with 10 as the majority.
* SQLODR3 processes less reports than SQLODR2 or SQLODR6, and also appears to have a longer mean build time (i.e. worse performance) than the other two production servers.



## `stat\_bin()` using `bins = 30`. Pick better value with `binwidth`.





# Objective 1

**Using known parameters of a desired report, can we predict how long it will take to run a particular type of report?**

We have found it best to use all parameters made available to us within the transformed ‘reports’ data-set to fit the response variable “ReportDeliveryTime.” Within the report, intuition tells us a few key parameters contribute strongly to the delivery time. For example: ReportBuildTime, QueueTime, LagTime and ReportBytes, cannot be used because they are all unknown before report completion.

## Simple Linear Regression Model

Looking at ReportBuildTime vs ReportId gives us the best at approximately .79 and a 2.83s RMSE. ReportDeliveryTime vs ReportId gives us an acceptable of about .49 and a 4.5s RMSE. This performance is acceptable, but becomes very difficult to explain when adding additional features due to the large number of reports (over 1,000). Thus, we will attempt to answer a broader question: given a report group how long should I expect to wait?

## Multiple Linear Regression subset

Because the amount of data is very large and we didn’t find a dependency of ReportID with ReportDeliveryTime, we chose a single type of report “ReportID=93”, which is roughly 1/10 of the overall data-set as a representative subset and used it for the remaining analysis.

With this subset, we filtered out a few more parameters and left 10 parameters: Server, ReportCategory, SchedFreq, HourOfDay, DayOfMonth, ReportCategory,Server, AgentCount, GroupCount,DelivMethod.

First we did a fit with all parameters and get a resulted of 0.54, which is not too bad, but is quite difficult to explain.

Ultimately after using several feature selection methodologies, we discovered that using all 30 features nets an of 0.54, while using just 5 of the most statistically significant features results in an of 0.52.

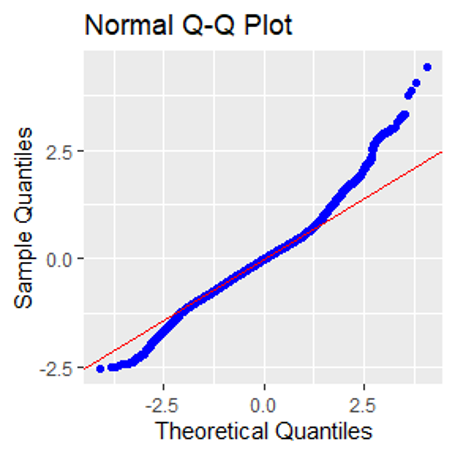
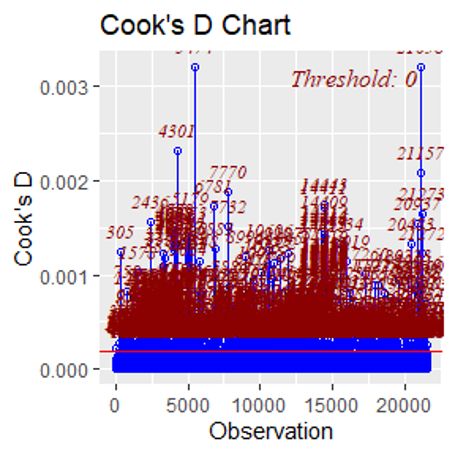
### Checking Assumptions

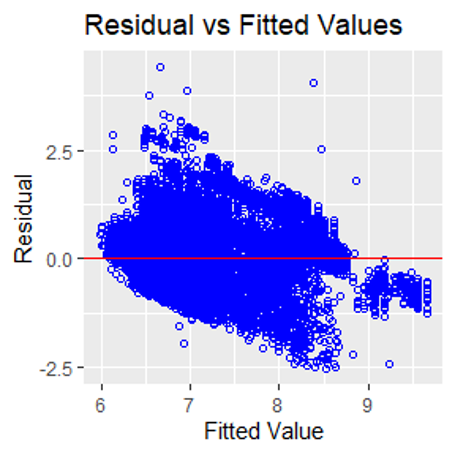
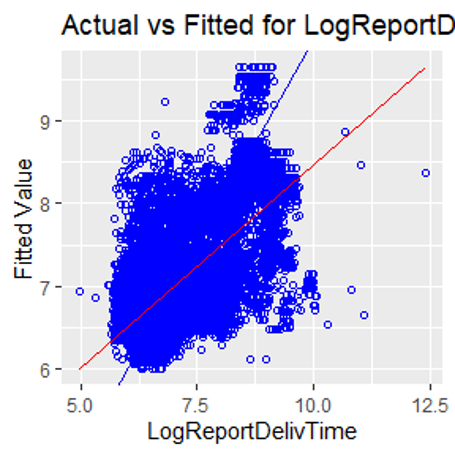
Linear Regression analysis has basic assumptions of residual normality, linearity of features, constant variance (equal spread), and independence of observations.

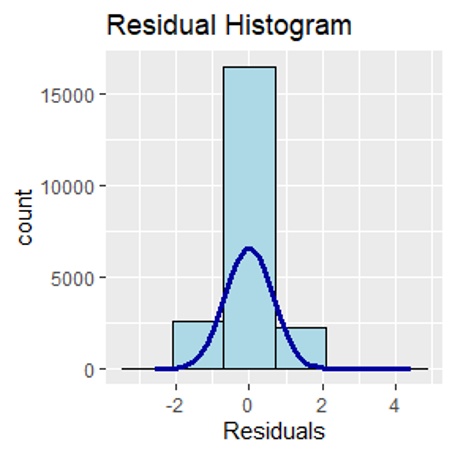
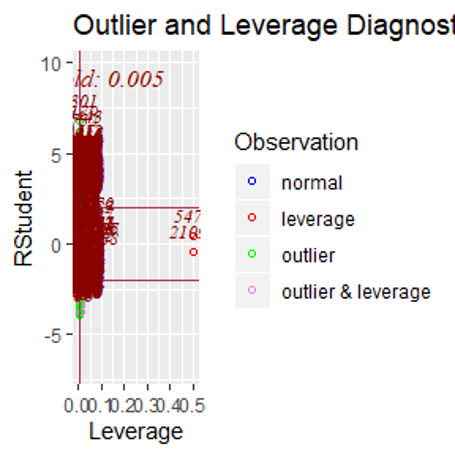
The q-q plot below (and previous histogram) shows that the assumption of normality has much improved after log transforming our response variable. We confirmed this to be correct by fitting the final model after log transformation.

After transformation we observed the q-q plot to exhibit qualities indicating the data is more normal: ideally we want a mostly straight line following closely to the residual 0 line.

* QQ Plot shows a slight departure from normality
* Cook’s D shows no major leverage points (.003)
* There are a few outliers with leverage in the sample selection, but not as much in the full data
* Histogram shows normality with a bit of right skew
* Ideally we’d like to see a totally random cloud in the fitted vs residuals
* The outliers are impacting predictions fairly significantly as can be seen by the leverage chart

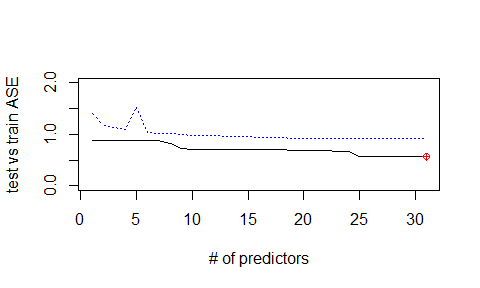
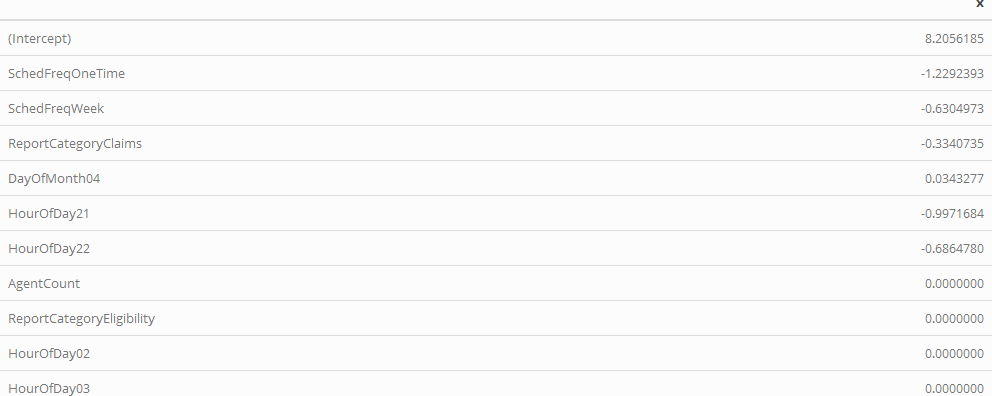
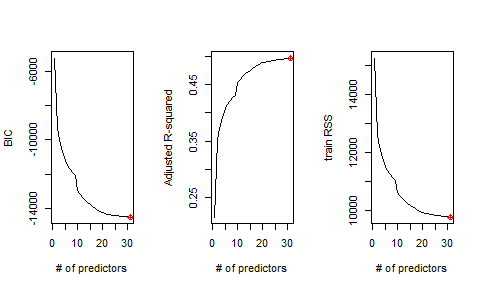
 

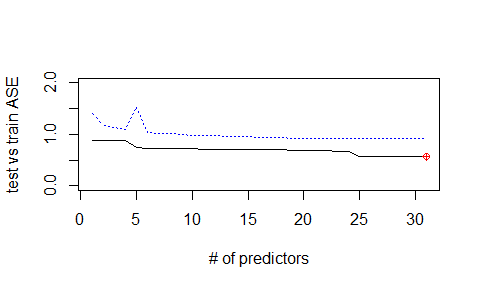
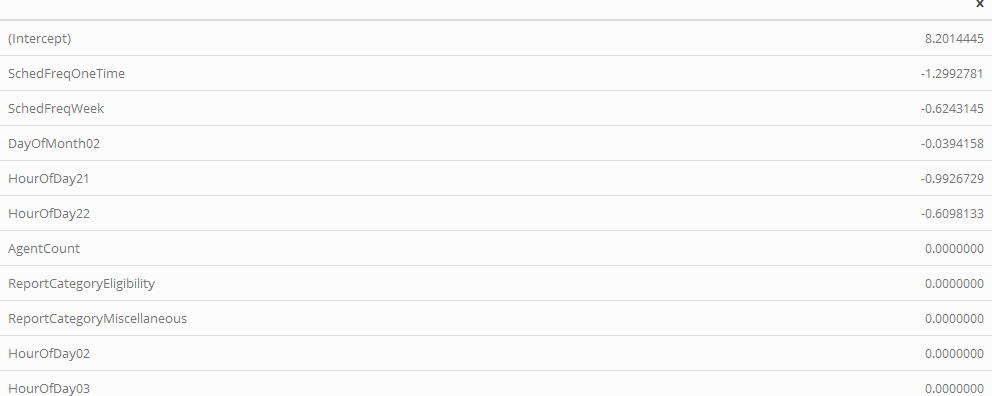
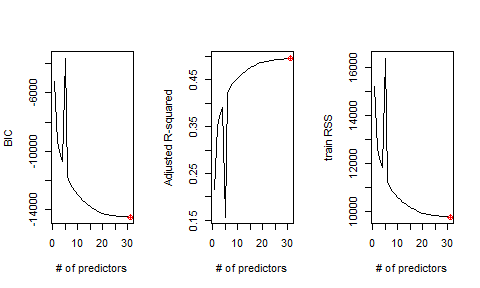
 

### Model Selection

#### Feature Selection: Forward



#### Feature Selection: Stepwise



#### Final Model Selection:

After the log transformation and re-fitting we get an of 0.54! This is good but because we have several features with numerous factor levels, we decided to do a round of feature selection using forward, backward and other criteria.

Using forward model selection and nvmax=30, we recognized that the BIC, adjusted and RSS shown in plot all show a “plateau” after roughly 25-30 parameters, indicating that we can get best fit and avoid further overfitting.

We also tried a stepwise selection process and found that the BIC, adjusted and RSS showed a similar trend to forward selection. There are 4 variables that appear to be significant: SchedFreq, HourOfDay, DayOfMonth and ReportCategory. The lowest BIC/RSS is similar to forward selection.

By checking the coefficients we identified SchedFreqOneTime and SchedFreqWeek to be statistically significant, so we included SchedFreq in the final model. Some variables related to HourOfDay are siginificant, to include them all we include HourOfDay as a variable in the final model. The final selection composes of 5 variables and the model is below:

When we initially build our intuitive model we did not split our dataset into train and test portions. We further conduct LASSO, Ridge and Elastic net for model fitting and there’s no fair way to compare the model of all above methods without a train/test experiment. So we perform a roughly 70-30 split on our original dataset to create a train and a test set, then use our train set to fit our optimized linear model and create a prediction vector from our test dataset. Then we can calculate our MSE of this model and compare it to those of forward, stepwise, LASSO, Ridge and Elastic net, we get the result below.

**A comparison of Feature Selection methods and their RMSE (lower is better):**

|  |  |
| --- | --- |
| Method | RMSE |
| Lasso | 1967.324 |
| Ridge | 1908.051 |
| Elastic Net | 1905.987 |
| Forward | 2099.483 |
| Stepwise | 2099.574 |
| LM (5 variables) | 1963.45 |

From the table above we can see that Lasso, Ridge and Elastic Net all give us a RMSE very close to each other, and slightly smaller than the multi-linear model. And the forward and stepwise model give us a higher-than-other RMSE. Because the Lasso models used all the variables we are at a risk of overfitting. Below shows a test versus train ASE plot and we can see that the test and train error are very close with a suggestion of including ~30 variables (really 5 category variables).

Below is a graphical representation of the results of the forward model:

##### name

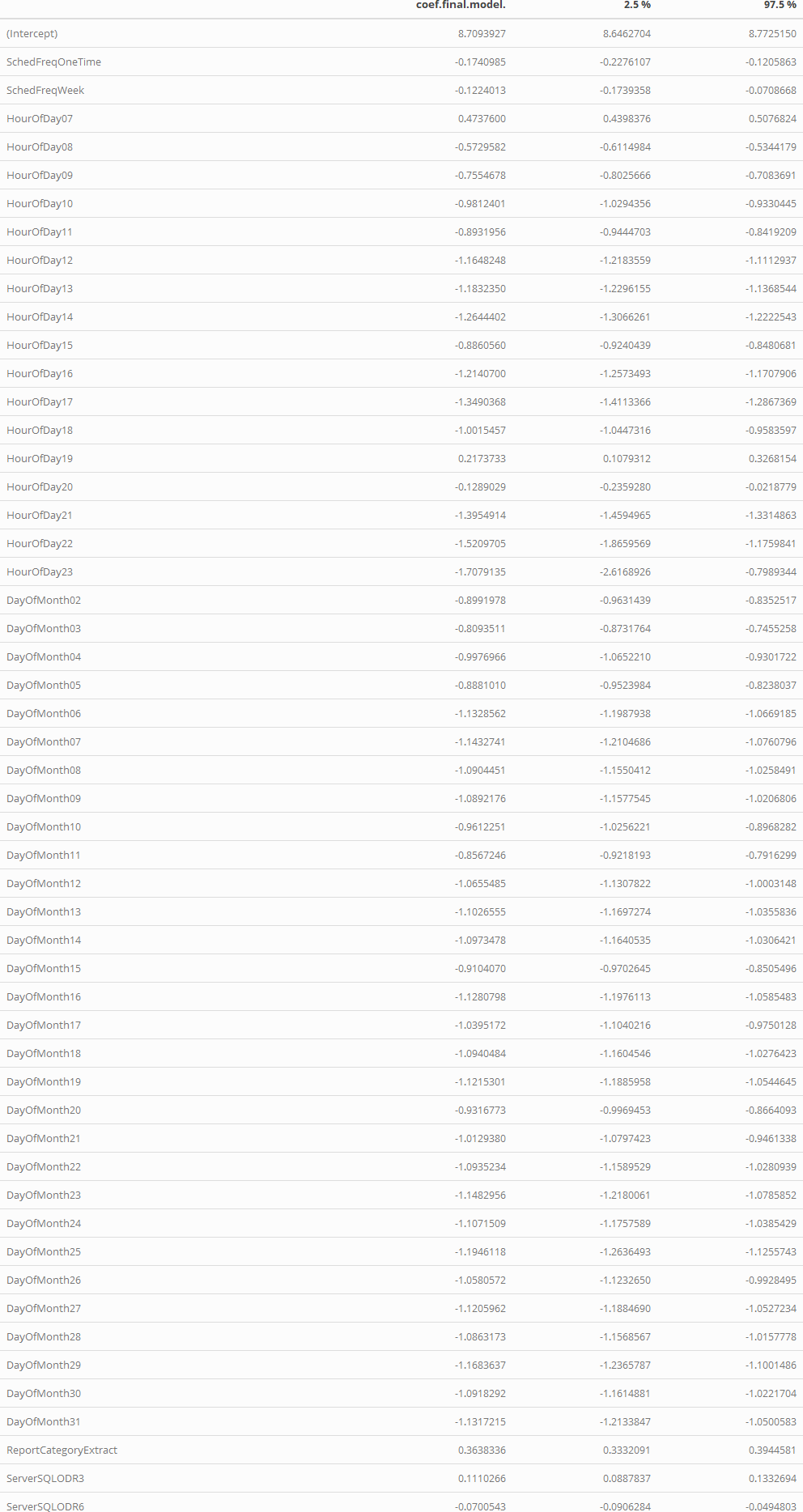
### Interpretation

Interpretting the outcome of the model requires us to check the coefficients in the table below and provide an explanation. In each categorical variable we can see one level has been chosen to use as the reference. We can check and find all the variables which listed below are with statistically significant p-value.

* Intercept is the mean estimate for the report delivery time keeping other variables constant.
* SchedFreqOneTime coefficient can be interpreted as the mean delivery time change from the reference of SchedFreqMonth.
* HourOfDay07 can be interpreted as the mean delivery time change from the reference of HourOfDay06.
* DayofMonth02 can be interpreted as the mean delivery time change from the reference of DayofMonth01.
* ReportCategoryExtract can be interpreted as the mean delivery time change from the reference of ReportCategoryClaims.
* ServerSQLODR3 can be interpreted as the mean delivery time change from the reference of ServerSQLODR2.  
  The logged variables need to be transformed back from their log in order to be properly interpreted in the context of other variables. Because this is a liner-log model, when interpreting the coefficient of ReportDelivTime, a one unit change in the ReportDelivTime would result in an exp(Beta) change in the mean of y.

The confidence intervals for the parameter estimates are in the table below. The confidence intervals show that all the listed variables don’t contain zero at the 95% level. This indicates that we can reject the null hypothesis and say that these estimates are significantly different from zero.

## Both width and height were defined. Use width only by default.



## Conclusion

In conclusion, the mulit-linear model with 5 predictors leveraging a logged response gave the best MSE with avoidance of over-fitting. We used a train and test split to validate the MSE across many methods and reached this conclusion. We know that the day of the month, the hour of the day, and the report category cause significant changes to estimated report delivery, with several unknown factors possibly related the number of queued reports contributing to large outliers in delivery time.

We also suspect that multiple linear regression isn’t the best model for predicting run time given this extremely variable data-set. Several high leverage outliers keep the MLR model from tightening predictions to desirable tolerances. Report delivery time varies from milliseconds to several days, which may be the result of a failed report which was restarted, or a large number of queued reports (such as those beginning at 6am on the first.) Having indicators for restarted reports, or the count of reports in the queue at the time the report starts would be useful in predicting run-time.

# Objective 2

**Our question of interest from Gilsbar is which server, and what time of day yields the best performance to schedule new reports?**

We ran a two-way ANOVA on the data to satisfy the second objective. The business requirements drove the selection of the explanatory variables. The business wants to know what time of day and on which server new reports should be added based upon current report run times as expressed by the ReportDeliveryTime response variable.

### Explanatory and Response variable creation

The two explanatory variables require some explanation. The “server” variable required us to remove the data associated with testing and development servers. The testing and development report run times have very different characteristics than production data. Thus, all data associated with servers “SQLODR1” and “SQLODR5” were removed.

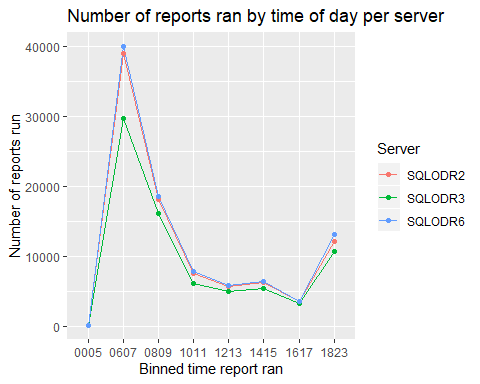
“HourBinned” variable indicates the time of day the report ran. Several things should be noted about it. A maintenance window exists every night between the hours of midnight and six AM. Only a few reports run during this maintenance window. Likewise, relatively few reports run during the late evening hours between eight PM and midnight. We binned these two time periods separately. The remaining time during the days was binned in two-hour blocks. The summary tables below give counts and means of the various bins.

The table above shows that server “SQLODR3” runs fewer reports than the other two servers and has differences in mean report run time and standard deviation.

### Server Performance Analysis

## hourstats...1.1. N Mean SD SE  
## 1 0005 384 69285.472 13610.750 35.44466176  
## 2 0607 108726 2021.303 4943.980 0.04547192  
## 3 0809 52825 1589.290 4707.032 0.08910615  
## 4 1011 21445 3045.266 7553.906 0.35224555  
## 5 1213 16435 2525.486 9787.416 0.59552270  
## 6 1415 18022 1483.042 5885.803 0.32658992  
## 7 1617 10284 3169.675 8681.578 0.84418305  
## 8 1823 36016 1902.417 7372.851 0.20471044

Looking at the data above for the BinnedHour variable the late evening hours and the overnight maintenance window bins display very different standard deviations than the rest of the bins. Also, of note the sample count size differs for those two bins. Finally reports should not be added to the overnight bin that includes the maintenance window.

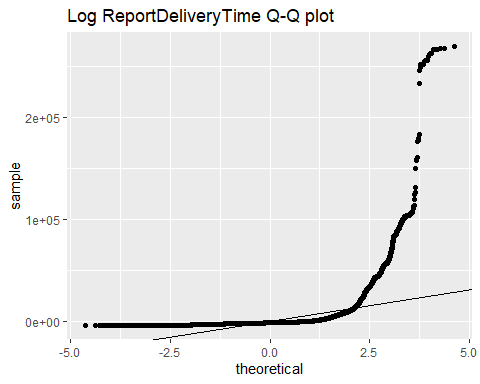
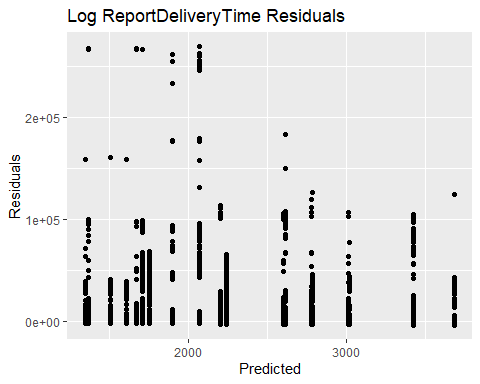


The graph shows that each server tracks the others in terms of reports run at a particular hour. A simple interpretation of the results benefits the business directly. The first analysis of the data included an unlogged response variable.

The bins for overnight and late evening (“0005”,”2023”) have been excluded from the remainder of analysis.

## Pr(>F)   
## (Intercept) < 2.2e-16 \*\*\*  
## Server 0.0002485 \*\*\*  
## HourBinned < 2.2e-16 \*\*\*  
## Server:HourBinned < 2.2e-16 \*\*\*  
## Residuals   
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1

The type 3 ANOVA table indicates that both terms as well as the interaction term are statistically significant with very low p-values.



The residual plot appears to demonstrate a cone shape with residuals beneath 2000 being noticeably more spread out than those above 3000. The Q-Q plot displays dramatic differences from normality. A log transform of the response variable is necessary.

## Pr(>F)   
## (Intercept) < 2.2e-16 \*\*\*  
## Server 8.985e-06 \*\*\*  
## HourBinned < 2.2e-16 \*\*\*  
## Server:HourBinned < 2.2e-16 \*\*\*  
## Residuals   
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1

The type III ANOVA table again shows that both the interaction term as well as the individual terms are significant.

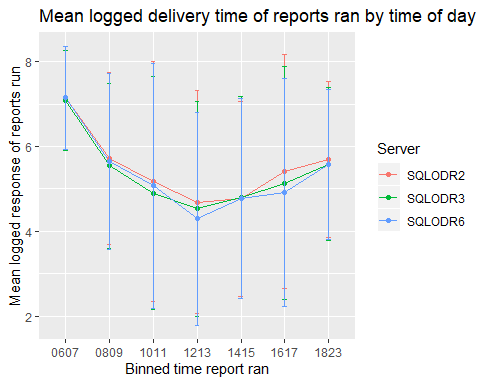
## servestats.fltr...1.1. N Mean SD SE  
## 1 SQLODR2 92239 6.139216 2.109653 2.287159e-05  
## 2 SQLODR3 76262 5.963995 2.108081 2.764261e-05  
## 3 SQLODR6 95252 6.057637 2.146274 2.253259e-05

The mean and standard deviation of the logged ReportDeliveryTime track extremely close now with respect to the server variable.

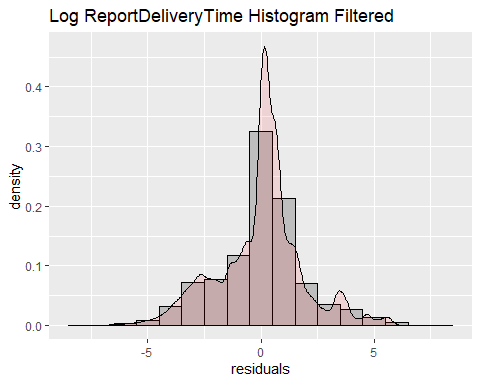
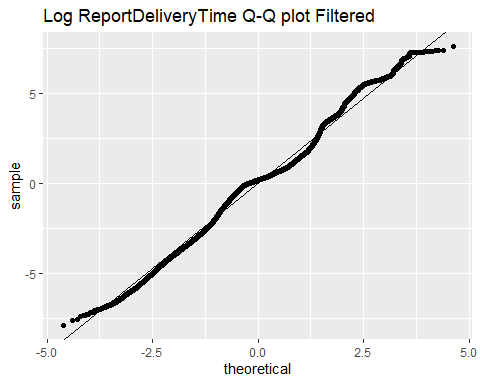
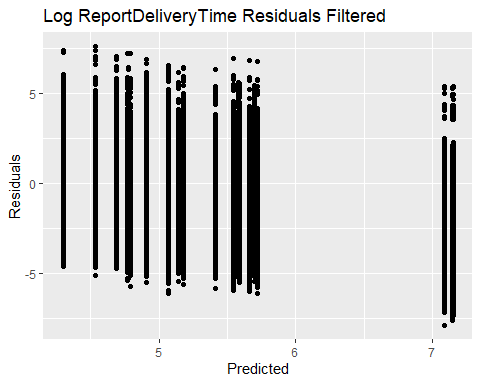
## hourstats.fltr...1.1. N Mean SD SE  
## 1 0607 108726 7.134787 1.200840 1.104465e-05  
## 2 0809 52825 5.644595 2.022867 3.829374e-05  
## 3 1011 21445 5.060337 2.841504 1.325019e-04  
## 4 1213 16435 4.503144 2.569200 1.563249e-04  
## 5 1415 18022 4.776321 2.349070 1.303446e-04  
## 6 1617 10284 5.155736 2.742066 2.666342e-04  
## 7 1823 36016 5.624229 1.804220 5.009497e-05

The HourBinned variable does display some differences in standard deviation. This issue is examined below. Please note that the over night and late evening hours were dropped from the model. An explanation for a penalty (4x multiplier on P-Value) we have applied to account for the variance in standard deviation may be found in the appendix section titled: 2-way ANOVA violation in SD.

The following graph shows that the mean log of the delivery time varies by the time of day and has a different standard deviation during times of high loads.



The following plots were created to further check for normality



Residuals, Q-Q plot, and the Histogram all support the assumption of normality. Finally, a pairwise comparison based upon server and HourBinned was run. Remember to adjust p-value by a factor of four due to the unequal variance and sample sizes.

## $Server  
## diff lwr upr p adj  
## SQLODR3-SQLODR2 -0.17522067 -0.19695630 -0.15348505 0.000000e+00  
## SQLODR6-SQLODR2 -0.08157831 -0.10209362 -0.06106299 3.030909e-14  
## SQLODR6-SQLODR3 0.09364237 0.07206289 0.11522184 2.764455e-14

## $HourBinned  
## diff lwr upr p adj  
## 0809-0607 -1.48603874 -1.51566816 -1.45640932 0.0000000000  
## 1011-0607 -2.07261693 -2.11435978 -2.03087407 0.0000000000  
## 1213-0607 -2.62750501 -2.67426105 -2.58074897 0.0000000000  
## 1415-0607 -2.35476266 -2.39969481 -2.30983052 0.0000000000  
## 1617-0607 -1.97484263 -2.03247929 -1.91720598 0.0000000000  
## 1823-0607 -1.50637952 -1.54034499 -1.47241405 0.0000000000  
## 1011-0809 -0.58657818 -0.63181361 -0.54134275 0.0000000000  
## 1213-0809 -1.14146627 -1.19136522 -1.09156731 0.0000000000  
## 1415-0809 -0.86872392 -0.91691807 -0.82052977 0.0000000000  
## 1617-0809 -0.48880389 -0.54901819 -0.42858959 0.0000000000  
## 1823-0809 -0.02034078 -0.05851709 0.01783554 0.7010836081  
## 1213-1011 -0.55488808 -0.61280586 -0.49697031 0.0000000000  
## 1415-1011 -0.28214574 -0.33860138 -0.22569010 0.0000000000  
## 1617-1011 0.09777429 0.03076440 0.16478418 0.0003399546  
## 1823-1011 0.56623740 0.51805032 0.61442449 0.0000000000  
## 1415-1213 0.27274235 0.21248539 0.33299930 0.0000000000  
## 1617-1213 0.65266238 0.58242003 0.72290472 0.0000000000  
## 1823-1213 1.12112549 1.06853598 1.17371500 0.0000000000  
## 1617-1415 0.37992003 0.31087832 0.44896174 0.0000000000  
## 1823-1415 0.84838314 0.79740838 0.89935790 0.0000000000  
## 1823-1617 0.46846311 0.40600102 0.53092520 0.0000000000

Conclusion and discussion

The business asked that we determine on what server and at what time of day additional reports should run.

Due to the residual and Q-Q plot of the untransformed data it was found to be necessary to perform a log transform of the response data. The data responded well to the transformation displaying all the characteristics of normalized data. Using the table “Mean logged response of reports ran by time of day per server” above we can make the following recommendations. • The number of reports run on server “SQLODR03” is about 20 % less than the other two servers. However, at most it has only 18% faster mean report delivery time(95% CI[5.3% - 32%]) this is between server “SQLODR03” and “SQLODR6” for the 10 AM through noon time period • The time period 6 AM to 8 AM has the highest mean report delivery time, followed by 8 AM to 10 AM and 6 PM to 8 pm. Every effort should be made to not schedule new reports during these time frames. • The time period noon to 2 PM has the lowest mean report delivery time, thus if possible new reports should be added there. The relative loading of the servers is as follows. Server “SQLODR02” has a 48% greater mean report delivery time (95% CI[31%, 68%]) than server”SQLODR06” for that time frame. While server “SQLODR3” has a 27% greater mean report delivery time (95% CI[11% , 44%]) than server ”SQLODR06”. Server ”SQLODR06” is the lightest loaded server as measured by mean report delivery time during the noon to 2PM time frame. • The very late evening hours 8PM to midnight remain an option for scheduling reports. Care must be taken that any reports scheduled during that time do not run past midnight into the maintenance window.

# Appendix: Features

## Report Parameters

* Agents - Individual AgentIDs
* CurrencyCode - Collection of different lines of business
* GroupCustom - Collection of different client types
* GroupFI - Fully Insured Client
* GroupSF - Self Funded Client
* ReportID - Uniquely identifies a report
* QueueID - Uniquely identifies a queue of one or more reports
* FormID - Uniquely identifies collection of report parameters

## Report Metrics

* Priority - Controlled report parameter
* QueuedDateTime - When report entered queue
* RenderStartDateTime - When server began work on queued report(s) in queueID batch
* ReportStartDateTime - When server began work on individual report
* ReportEndDateTime - When server completed work on individual report
* RenderEndDateTime - When server completed work on queued report(s) in queueID batch
* ReportBytes - Size in bytes of completed report
* RptFormat - Format of the report delivered
* SchedFreq - Scheduled Report Frequency
* Server - Server on which the report was built

## Engineered Features

* AgentCount - Number of agents selected
* DayOfMonth - Day of the month report ran
* DayOfWeek - Day of the week report ran
* GroupCount - Number of clients selected
* HourOfDay - Hour of the day report ran
* LagTime - Time report waited for batch to finish (RenderEndDateTime less ReportEndDateTime)
* QueueTime - Time report entered queue (RenderStartDateTime less QueuedDateTime)
* ReportBuildTime - Time spent to build the report (ReportEndDateTime less ReportStartDateTime)
* ReportDeliveryTime - Time user spent waiting for report (RenderEndDateTime less QueuedDateTime)

# Appendix: Report Categories

## Report Category crosswalk:

|  |  |
| --- | --- |
| Category | Groups |
| Eligibility | “Eligibility”,“Accumulator”,“EventHistory”,“COBRA”,“RedCard”,“ComPsych” |
| Miscellaneous | “Miscellaneous” |
| Reinsurance | “Reinsurance”,“Aggregate Report” |
| Professional Liability | “PL” |
| Actuarial | “Actuarial”,“ISBReports”,“Renewal”, *except when rptCategoryDesc = ‘Renewal’ and reportGroup = ‘CommissionsTFB’ use ‘Commissions’* |
| Finance Commission | “Commissions”,“CRMOnlineFinance”,“CommCnt”,“CommCompare” |
| Claims | “ClaimsAudit”,“CMSRepting”,“Repository”,“GilsbarPPO”,“Claims” |
| Medical Management | “Wellness Repository”,“MedCom”,“TVC”,“MedInsight” |
| Metrics | “iTrac”,“TMS”,“Customer Service”,“ODR Admin”,“PortalStats”,“AutoAdjudication”,“Genelco”,“Supervisor” |
| Extract | “DataExtracts”,“Large Claims Reports” |
| Finance Billing | “Billing”, “Coverage”,“Refunds”,“Credit Card Process”,“Deposit”,“CheckRecon”,“Check Register”,“FinancePremium”, “Premium” |

# Appendix: Data Preparation

## ETL of original data

tic('total run-time')  
  
# Note I have relabeled this originalReports  
load("./data/originalReports.RData")  
load("./data/customGroups.RData")  
load("./data/currencyCode.RData")  
  
# Remove imported reports & dev servers:  
reports <- reports %>% filter(Server %in% c('SQLODR2','SQLODR3','SQLODR6'))  
  
# Drop report with bad data:  
reports <- reports %>% filter(!is.na(RunTimeSEC))  
  
tic('clean XML')  
  
# Clean up the XML:  
reports$XMLResponseStringClean <- str\_replace(reports$XMLResponseString, '>>', '>')  
reports$XMLResponseStringClean <- str\_replace(reports$XMLResponseStringClean, '&', '-and-')  
reports$XMLResponseStringClean <- str\_replace(reports$XMLResponseStringClean, 'Transaction Entry Date Thru', 'TransactionEntryDateThru')  
reports$XMLResponseStringClean <- str\_replace(reports$XMLResponseStringClean,'(<AgentCode>.\*</AgentCode>)<ODRParameters>','<NewDataSet><ODRParameters>\\1')  
toc()  
  
tic('XML to Cols')  
# Parse XML values into Columns: (this might take a minute or two each)  
reports$GroupCustom <- sapply(reports$XMLResponseStringClean, function(x)   
 ifelse(x %like% '<GroupCustom>', xml\_find\_all(read\_xml(x), './/GroupCustom') %>% xml\_text(), NA))  
reports$CurrencyCode <- sapply(reports$XMLResponseStringClean, function(x)   
 ifelse(x %like% '<CurrencyCode>', xml\_find\_all(read\_xml(x), './/CurrencyCode') %>% xml\_text() %>% toupper(), NA))  
reports$groupFI <- sapply(reports$XMLResponseStringClean, function(x)   
 ifelse(x %like% '<GroupFI>', xml\_find\_all(read\_xml(x), './/GroupFI') %>% xml\_text(), NA))  
reports$groupSF <- sapply(reports$XMLResponseStringClean, function(x)   
 ifelse(x %like% '<GroupSF>', xml\_find\_all(read\_xml(x), './/GroupSF') %>% xml\_text(), NA))  
reports$Agents <- sapply(reports$XMLResponseStringClean, function(x)   
 ifelse(x %like% '<AgentCode>', xml\_find\_all(read\_xml(x), './/AgentCode') %>% xml\_text(), NA))  
toc()  
  
# Capture number of days:  
tic("Dates")  
reports$DateFrom <- sapply(reports$XMLResponseStringClean, function(x)   
 ifelse(x %like% 'DateFrom>\\d+'| x %like% 'DateRangeFrom', str\_replace(x, '.\*Date(RangeFrom|From)>(\\d+)</.\*', '\\2'), NA))  
reports$DateThru <- sapply(reports$XMLResponseStringClean, function(x)   
 ifelse(x %like% 'DateThru>\\d+'| x %like% 'DateRangeThru', str\_replace(x, '.\*Date(RangeThru|Thru)>(\\d+)</.\*', '\\2'), NA))  
  
reports$queued <- as\_date(reports$QueuedDateTime)  
reports$queued <- str\_replace\_all(reports$queued, "-", "")  
  
reports$DateThru <- ifelse(is.na(reports$DateThru),NA,ifelse((as\_date(reports$DateThru) > as\_date('20251231') | reports$DateThru == "") & as\_date(reports$DateFrom) != as\_date(reports$DateThru), reports$queued, reports$DateThru))  
  
reports$NumDays <- ifelse(is.na(reports$DateFrom) | is.na(reports$DateThru) , NA , difftime(as\_date(reports$DateThru), as\_date(reports$DateFrom) , units = "days"))  
reports$NumDays <- ifelse(as\_date(reports$DateFrom) == as\_date(reports$DateThru), 1, reports$NumDays)  
reports$NumDays <- abs(reports$NumDays)  
  
reports$NumDays <- ifelse(reports$XMLResponseStringClean %like% '>(CURRENT|PRIOR)YEAR<', 365, reports$NumDays)  
reports$NumDays <- ifelse(reports$XMLResponseStringClean %like% '>(CURRENT|PRIOR)QUARTER<', 91, reports$NumDays)  
reports$NumDays <- ifelse(reports$XMLResponseStringClean %like% '>(CURRENT|PRIOR)MONTH<', 30, reports$NumDays)  
reports$NumDays <- ifelse(reports$XMLResponseStringClean %like% '>(CURRENT|PRIOR)WEEK<', 7, reports$NumDays)  
reports$NumDays <- ifelse(reports$XMLResponseStringClean %like% '>(CURRENT|PRIOR)DAY<', 1, reports$NumDays)  
reports$NumDays <- ifelse(reports$XMLResponseStringClean %like% '>ROLLING3MONTHS<', 91, reports$NumDays)  
reports$NumDays <- ifelse(reports$XMLResponseStringClean %like% '>ROLLING6MONTHS<', 182, reports$NumDays)  
reports$NumDays <- ifelse(reports$XMLResponseStringClean %like% '>ROLLING12MONTHS<', 365, reports$NumDays)  
reports$NumDays <- ifelse(reports$XMLResponseStringClean %like% '>ROLLING15MONTHS<', 456, reports$NumDays)  
reports$NumDays <- ifelse(reports$XMLResponseStringClean %like% '>ROLLING24MONTHS<', 730, reports$NumDays)  
toc()  
  
# Count commas to determine number of Agents  
reports$AgentCount <- ifelse(is.na(reports$Agents), 0, str\_count(reports$Agents, ',')+1)  
  
  
# Join GroupCount to GroupCustom:  
reports <- left\_join(reports, CustomGroups, by=c('GroupCustom' = 'CustomGroupCode'))  
  
  
# Capture original GroupCount  
reports$CustomGroupCount <- reports$GroupCount  
  
  
# Build CurrencyCode counts: (this will take a few minutes)  
tic("CC Function")  
reports$CurrencyCodeGroups <- sapply(reports$CurrencyCode, function(x) ifelse(is.na(x), 0, getCCgroups(x)))  
toc()  
  
# Merge GroupCounts:  
reports$GroupCount <- ifelse(!is.na(reports$groupFI), 1, reports$GroupCount)  
reports$GroupCount <- ifelse(!is.na(reports$groupSF), 1, reports$GroupCount)  
reports$GroupCount <- ifelse(reports$XMLResponseStringClean %like% '<ExtGroupNumber>', 1, reports$GroupCount)  
reports$GroupCount <- ifelse(reports$XMLResponseStringClean %like% '>ALLGROUPS<', 8353, reports$GroupCount)  
reports$GroupCount <- ifelse(is.na(reports$GroupCount), reports$CurrencyCodeGroups, reports$GroupCount)  
reports$GroupCount <- ifelse(reports$CurrencyCodeGroups < reports$GroupCount & reports$CurrencyCodeGroups > 0, reports$CurrencyCodeGroups, reports$GroupCount)  
reports$GroupCount <- ifelse(is.na(reports$GroupCount),8353,reports$GroupCount)  
reports$GroupCount <- ifelse((reports$GroupCount == 0),8353,reports$GroupCount)  
  
# Update report categories: Adam  
  
  
reports$ReportCategory <- case\_when(reports$RptCategoryDesc %in%   
 c("Eligibility","Accumulator","EventHistory","COBRA",  
 "RedCard","ComPsych") ~ "Eligibility",  
 reports$RptCategoryDesc %in%   
 c("Miscellaneous") ~ "Miscellaneous",  
 reports$RptCategoryDesc %in%   
 c("Reinsurance","Aggregate Report")~ "Reinsurance",  
 reports$RptCategoryDesc %in%   
 c("PL")~ "Professional Liability",  
 reports$RptCategoryDesc %in% c("Renewal") &   
 reports$ReportGroupDescription == "Commissions TFB" ~ "Finance Commission",  
 reports$RptCategoryDesc %in%   
 c("Actuarial","ISBReports","Renewal")~ "Actuarial",  
 reports$RptCategoryDesc %in%   
 c("Commissions","CRMOnlineFinance","CommCnt",  
 "CommCompare")~ "Finance Commission",  
 reports$RptCategoryDesc %in%   
 c("ClaimsAudit","CMSRepting","Repository","GilsbarPPO","Claims")~ "Claims",  
 reports$RptCategoryDesc %in%   
 c("Wellness Repository","MedCom","TVC","MedInsight")~ "Medical Management",  
 reports$RptCategoryDesc %in%   
 c("iTrac","TMS","Customer Service","ODR Admin","PortalStats",  
 "AutoAdjudication","Genelco","Supervisor")~ "Metrics",  
 reports$RptCategoryDesc %in%   
 c("DataExtracts","Large Claims Reports")~ "Extract",  
 reports$RptCategoryDesc %in%   
 c("Billing", "Coverage","Refunds","Credit Card Process","Deposit"  
 ,"CheckRecon","Check Register","FinancePremium", "Premium")~ "Finance Billing",  
 TRUE~"OTHER")  
  
# Engineered Features - Adam  
  
reports$ReportBuildTime <- as.numeric(difftime(reports$ReportEndDateTime, reports$ReportStartDateTime, units = "secs"))  
reports$ReportDeliveryTime <- as.numeric(difftime(reports$RenderEndDateTime, reports$QueuedDateTime, units = "secs"))  
reports$LagTime <- as.numeric(difftime(reports$RenderEndDateTime, reports$ReportEndDateTime, units = "secs"))  
reports$QueueTime <- as.numeric(difftime(reports$RenderStartDateTime, reports$QueuedDateTime, units = "secs"))  
reports$DayOfMonth <- format(as\_datetime(reports$ReportStartDateTime),"%d")  
reports$DayOfWeek <- format(as\_datetime(reports$ReportStartDateTime),"%A")  
reports$HourOfDay <- format(as\_datetime(reports$ReportStartDateTime),"%H")  
  
  
# NAs at a glance:  
MissingValues <- sapply(reports, function(x) sum(is.na(x)))  
MissingValues %>% kable("html") %>% kable\_styling()  
  
# Drop extreme outliers, reports which are highly likely to be errors:  
reports <- reports %>% filter(ReportDeliveryTime < 300000)  
  
# Remove delivery time < 0  
reports <- reports %>% filter(ReportDeliveryTime > 0)  
  
# Convert to factors:  
reports$SchedFreq <- as.factor(reports$SchedFreq)  
reports$TestProdIndicator <- as.factor(reports$TestProdIndicator)  
reports$DelivMthd <- as.factor(reports$DelivMthd)  
reports$DayOfMonth <- as.factor(reports$DayOfMonth)  
reports$DayOfWeek <- as.factor(reports$DayOfWeek)  
reports$ReportCategory <- as.factor(reports$ReportCategory)  
reports$RptFrmt <- as.factor(reports$RptFrmt)  
reports$Priority <- as.factor(reports$Priority)  
reports$HourOfDay <- as.factor(reports$HourOfDay)  
reports$Server <- as.factor(reports$Server)  
  
# Add some time binning variables:  
reports$HourBinned <- case\_when(reports$HourOfDay %in% c("06","07") ~ "0607",  
 reports$HourOfDay %in% c("08","09") ~ "0809",  
 reports$HourOfDay %in% c("10","11")~ "1011",  
 reports$HourOfDay %in% c("12","13")~ "1213",  
 reports$HourOfDay %in% c("14","15")~ "1415",  
 reports$HourOfDay %in% c("16","17")~ "1617",  
 reports$HourOfDay %in% c("18","19","20","21","22","23")~ "1823",  
 reports$HourOfDay %in% c("00","01","02","03","04","05")~ "0005",  
 TRUE~"OTHER")  
  
reports$TimeBlock3h <- case\_when(reports$HourOfDay %in%   
 c("00","01","02","03","04","05") ~ "Maintenance",  
 reports$HourOfDay %in%   
 c("06","07","08") ~ "6am-8am",  
 reports$HourOfDay %in%   
 c("09","10","11")~ "9am-11am",  
 reports$HourOfDay %in%   
 c("12","13","14")~ "12pm-2pm",  
 reports$HourOfDay %in%   
 c("15","16","17")~ "3pm-5pm",  
 reports$HourOfDay %in%   
 c("18","19","20","21","22","23")~ "Overnight",  
 TRUE~"OTHER")  
  
reports$TimeBlock4h <- case\_when(reports$HourOfDay %in%   
 c("00","01","02","03","04","05") ~ "Maintenance",  
 reports$HourOfDay %in%   
 c("06","07","08","09") ~ "6am-9am",  
 reports$HourOfDay %in%   
 c("10","11","12","13")~ "10am-1pm",  
 reports$HourOfDay %in%   
 c("14","15","16","17")~ "2pm-5pm",  
 reports$HourOfDay %in%   
 c("18","19","20","21","22","23")~ "Overnight",  
 TRUE~"OTHER")  
  
  
# Stash the original data before we drop columns  
reportsOrg <- reports  
  
# Drop columns we don't need:  
dropColumns <- c('XMLResponseString','ReportGroupDescription','ReportPath','StatusDesc','NotifiedDateTime','DeletedDateTime','FormDeleted','ProcessId','FormId','QueueId','RunTimeInMinsSecs','RptCategoryId','CurrencyCodeGroups','CustomGroupCount','RptCategoryDesc', 'RuntimeSec', 'NearestStartHour','XMLResponseStringClean','RunTimeSEC','ReportStartDay')  
  
reports <- reports[ , !(names(reports) %in% dropColumns)]  
  
# Save data for analysis:  
save(reports, file="./data/reports.RData")  
save(reportsOrg, file="./data/reportsFull.RData")  
  
toc()

# Appendix: Obj1 Performance Testing

# Appendix: 2-way ANOVA violation in SD

The difference in standard deviation between the various groups is concerning. On-line research yielded the following article https://link.springer.com/article/10.3758/s13428-017-0918-2?shared-article-renderer#Tab6 . The article indicates that a useful measure of unequal variance is the Variance ratio. This ratio is simply the largest group variance divided by the smallest group variance. Looking at the summary statistics table for hours binned we find that:

Variance Ratio= (σ\_max )^2/(σ\_min )^2 = 〖2.84〗2/〖1.20〗2 =5.6 The correlation between the sample size and variance is -0.81. The coefficient of sample size variation (Δn) was calculated to be (SD(Group Sample Size))/(mean(Group Sample Size))=0.932 According to the article’s conclusions with the parameters as calculated highlighted below (Table 10 of the article) the Type I error ratio will be liberal. However, most p-values reported by the two-way ANOVA are very small. The effects of unequal sample size and unequal variance within the sample groups should only affect relatively few term pairings. Assume p-values can be as much as four times larger than the given value. Variance ratio Pairing Coefficient of sample size variation Type I error rate 1.6, 1.7, 1.8 + or −1 0.50 Liberal 2 −1 0.33; 0.50 Liberal 3 1 0.50 Conservative −.50 0.50 Liberal −1 0.33; 0.50 Liberal 5 & 9 1 0.50 Conservative −.50 0.33; 0.50 Liberal −1 0.16; 0.33; 0.50 Liberal