Predicting Incident Management Service Level Agreement (SLA) Failures

Capstone Written Report

Carolyn M. Hennings
Western Governors University
chenn15@wgu.edu
March 10, 2020

Abstract

Information Technology (IT) Service Management practices optimize the efficiency and effectiveness of IT services delivered to users. Incidents represent service disruptions. Service Level Agreements (SLA) establish thresholds for resolution of incidents within specified timeframes based on impact and urgency designations. Decreasing SLA breaches increases the availability of IT services and represents an important consideration for IT service providers.

This study explores indicators of incident SLA breaches with respect to data available during the early stages of an IT incident's lifecycle. The study built a Logistic Regression model using Python and a number of tools from the SciKit-Learn library. Some supplementary analysis leveraged the R language. This paper describes the data collection, the data extraction and preparation, and the analysis steps performed throughout the study followed by a summary of findings, implications, and recommendations.

Table of Contents

Table of Exhibits

1 Research Question

What factors predict Incident Management SLA compliance?

1.1 Justification for the Question

In the context of Information Technology (IT) organizations providing IT services to customers, Incident Management practices and processes serve as critical customer satisfaction enablers. Incident Management aims to minimize the duration of interruptions in normal service operations while also minimizing the impact of those interruptions (Hanna, 2011, p. 29). Service Level Agreements (SLA) describe services and establish service level targets as negotiated and agreed upon between the service provider and the customer (Hanna, 2011, p. 54). Common SLA elements identify target thresholds for the duration of incidents from initial identification (opened) to restoration of service at normal operational levels (resolved). The swift resolution of business-impacting incidents represents a primary focus for IT service support organizations. The ability to proactively identify incidents at risk of failure to meet an SLA threshold, "SLA-at-Risk", allows for decisions and actions that reduce the duration and severity of service disruptions.

Service Desk managers, those responsible for the Incident Management process within an IT organization, will benefit from the results of this study. With customization of the data feed to a specific environment, the model will assist in identifying characteristics of incidents at risk of failure to meet an SLA (Higgins, 2016). Within the IT Service Management (ITSM) lifecycle, the results of this study will also interest Problem Management and Continual Service Improvement practitioners.

1.2 Context

IT organizations use ITSM systems to capture information about the execution of Incident Management processes. These systems produce logs containing details about incidents, for example, the steps taken to resolve them, the individuals involved with the incidents, the elements within the IT environment impacted by the incident, and timestamps for actions taken throughout the lifecycle of an incident.

This project investigated an extract of Incident Management data from an ITSM system to determine indicators of failure to meet an SLA threshold and develop a model for predicting those incidents. Insight into SLA-at-Risk conditions notify management of IT environment components requiring attention, similar to a customer churn analysis identifying characteristics for marketing attention.

A literature scan identified related research in the field of process mining, a combination of data mining with process science (Van Der Aalst, 2018, p. 15). Amaral et al. investigated attribute selection methods to build completion time prediction models (Amaral et al., 2019). Hinkka et al. evaluated feature selection algorithms by comparing classification accuracy and response times. The study used the Gradient Boosting Machine classification method and an approximation of the mutual information score across feature selection methods on two different data sets. Sarnovsky and Surma used Random Forests and Gradient Boosting Machine classifiers to identify incident sources and predict impacts (Sarnovsky & Surma, 2018). Malley leveraged split-plot Analysis of Variance (ANOVA) techniques to assess the "extent to which IT staff use of organizational knowledge generated from data warehouse analytical measures reduces the number of IT incidents over a 30-day period". Buhler et al. used a multinomial logistic regression model to predict impact pattern categorizations (Buhler et al., n.d., p. 13). In summary, these studies focus on questions associated with the efficiency and effectiveness of incident management processes. In contrast, this study focuses on factors contributing to incidents causing harm to continual delivery of services.

1.3 Hypothesis Discussion

The hypotheses under study focus on identifying significant factors indicating the probability of an IT support organization's ability to close an incident within agreed service level thresholds. The study leveraged logistic regression techniques to test the following hypotheses:

H_0: Data contains no significant indicators about the final SLA status of an incident (the coefficients of a logistic regression model are zero, $\beta_i = 0$)
H_1: Data contains significant indicators about the final SLA status of an incident (the coefficients of a logistic regression model are not zero, $\beta_i \neq 0$)

The factors under consideration include data about incidents available in the early stages of an incident's lifecycle. If any one of the logistic regression model's coefficients significantly differs from zero, the study will accept the alternative hypothesis (H_1), otherwise the study will fail to reject the null hypothesis (H_0).

2 Data Collection

This section describes the data collected, the collection methodology, and associated challenges.

2.1 Collected Data

A study investigating factors contributing to incident management SLA risk requires an extract from an ITSM system used by an IT organization for tracking incidents over a specific period. This study leveraged an existing, publicly-available data set used in the *2014 Business Processing Intelligence Challenge (BPIC)* (*10th International Workshop on Business Process Intelligence 2014*, n.d.). While the challenge released four data sets, this project focused only on the Incident Records file (Van Dongen, 2014). The selected data set consists of 46,606 observations having 28 variables extracted from an ITSM system used by a bank located in the Netherlands (*Quick reference BPI Challenge 2014*, n.d.). The terms for use of the data set specify that "The user is allowed to remix, transform or build upon the data, but only for noncommercial purposes" (4TU.Centre for Research Data, 2016).

2.2 Methodology

Data collection followed a three-step approach.

2.2.1 Search

The first step in data collection for an analysis project involves identifying the location and availability of relevant data sets. Internet-based search tools used by this project to identify publicly available subject data sets included:

- Data.Gov (https://www.data.gov/)
- University of California, Irvine Machine Learning Repository (https://archive.ics.uci.edu/ml/index.php)
- Google Scholar (https://scholar.google.com/)

- Google Dataset Search (https://datasetsearch.research.google.com/)
- Kaggle Datasets (https://www.kaggle.com/datasets)

Search criteria identified a candidate pool of data sets with relevancy to the topic of Incident Management in an IT Service Management context.

2.2.2 Screen

The search identified two data sets for consideration, as listed below. Both data sets contain similar information.

- *2014 Business Processing Intelligence Challenge (BPIC)* Incident Records file (Van Dongen, 2014)
- UCI Machine Learning Repository Incident management process enriched event log Data Set (Amaral et al., 2019)

2.2.3 Select

This project used the first data set due to the availability of supporting documents, such as a detailed, accurate data dictionary and Incident Management process documentation.

2.3 Methodology Advantages and Disadvantages

A significant disadvantage of the data collection methodology was the restriction to publicly available data sets. While internet searches had the advantage of returning a broad range of results, a disadvantage was the need to carefully filter out topics such as cybersecurity and transportation incidents.

2.4 Challenges

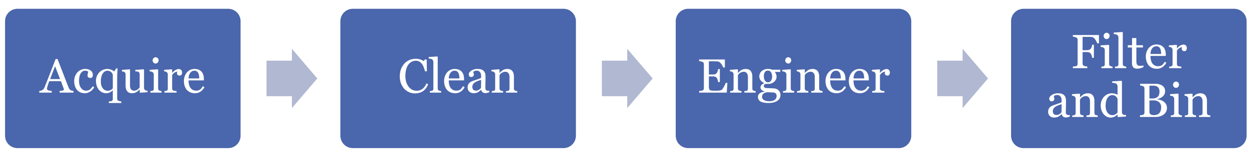
Internet searches for publicly available data sets return a large variety of results. Careful formulation and iterative refinement of search queries contributed to subsequently limiting the results to smaller, more relevant results. The represented additional time requirements. Without the ability to collect additional data, downstream challenges also arose.

3 Data Extraction and Preparation

This section describes the process for extracting and preparing the data for analysis followed by a discussion of the tools and techniques leveraged. See the appendices for a summary of the source data set as well as the executed notebooks showing performance of each data extraction and preparation step.

3.1 Approach

Data extraction and preparation involved four steps.



3.1.1 Acquire

The data set required downloading from the *2014 Business Processing Intelligence Challenge (BPIC)* website located at https://www.win.tue.nl/bpi/doku.php?id=2014:challenge. Initial exploratory data analysis (EDA) identified a relatively clean data set. Preliminary data cleaning steps included: conversion of strings representing dates to datetime data type, removal of non-incident records, and removal of records with a status other than `closed`.

3.1.2 Clean

Data profiling during the previous step revealed collinearity among some variables. This step addressed some collinearity through creating aggregated variables and noted other items for later consideration. The project addressed missing values by dropping records representing fewer than 4% of the total, setting values to zero, “Not Applicable”, and “Yes/No” where appropriate (Nisbet et al., 2009, pp. 50–75).

3.1.3 Engineer

The original data source lacked a binary indicator for the target variable. The project engineered the target variable, `SLAFail`, by setting the value to `1,` according to the business rules described in Exhibit 1. Stanford University IT provides an example of a similar business rule (*Measuring Response and Resolution Times in Remedy | University IT*, n.d.).

Exhibit 1. Business Rule for SLAFail Target Variable

Priority	Time Between Opened and Resolved
1 Very High	Greater than 240 minutes (4 hours)
2 High	Greater than 480 (8 hours)
3 Medium	Greater than 1440 (1 day)
4 Low	Greater than 2880 (2 days)
5 Very Low	Greater than 5760 (4 days)

The engineered target variable, `FailSLA`, resulted in 30% of the cases showing a failure to meet the defined SLA (closing the incident within the specified time based on `Priority`). The study will use proportional stratified random sampling to split the data set for training and testing purpose (Tufféry, 2011, p. 90)

3.1.4 Filter and Bin

This final data extraction and preparation step addressed the high dimensionality of datetime variables and restricted the data set based on timeframe.

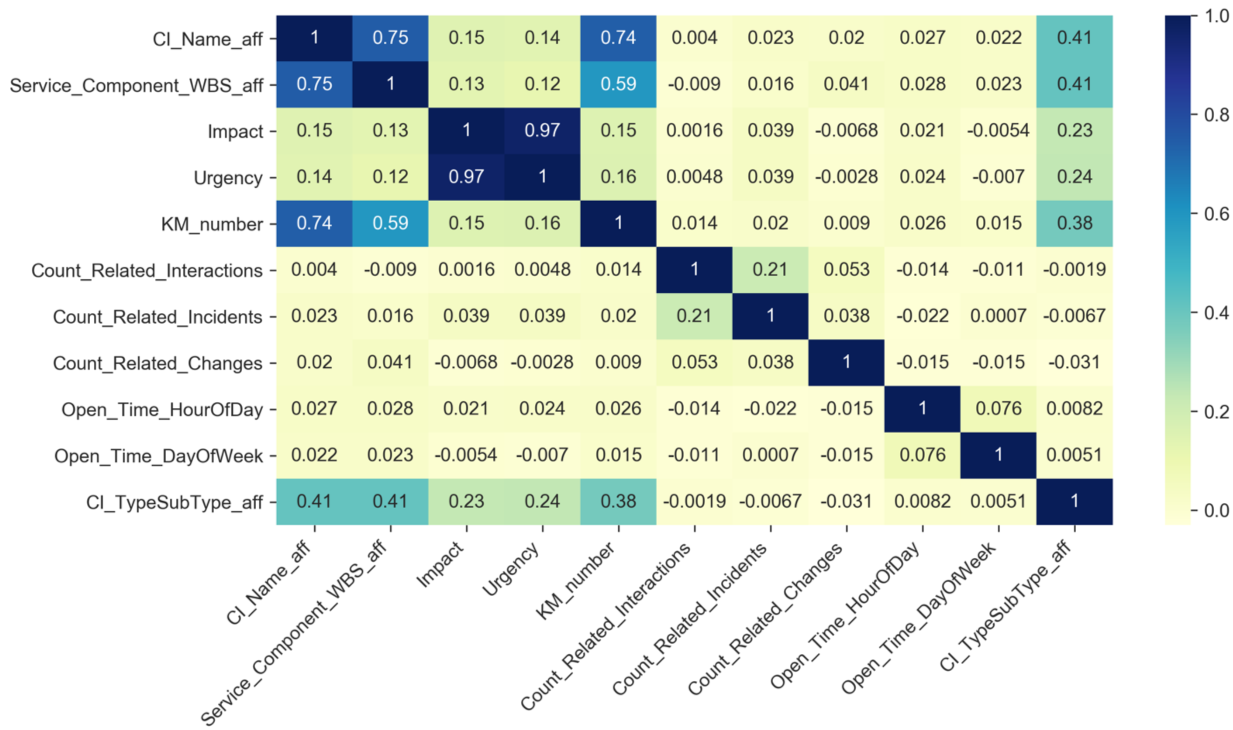
The datetime variables required binning (discretization) to reduce dimensionality (Tufféry, 2011, pp. 31–32). For each datetime variable (`Open_Time`, `Resolved_Time`, and `Closed_Time`), this step created two corresponding binned variables. One for the hour of the day and the other for the day of the week.

This step reduced the data set to include only those incidents within a six-month time window (1 October 2013 through 31 March 2014). This ensures that the records cover the entire lifecycle of the incident, i.e. both open and close dates exist within the window (Buffett et al., 2014, p. 4).

Since the study proposes to predict SLA-at-Risk (`SLAFail`), this filtering step limited the data set used for model development to only those variables available upon creation of an incident record. Given the business case of an Incident Manager needing to identify the SLA-at-Risk incidents shortly after identification, little value would result from including data only available at later stages of an incident's lifecycle.

Collinearity negatively affects logistic regression models (Tufféry, 2011, pp. 86–87). A heatmap based on Pearson's Correlation Coefficient showed significant correlation (> 0.70) among variables as shown in Exhibit 2.

Exhibit 2. Correlation Heatmap



The data set used for further analysis dropped the `Impact` and `CI_Name_aff` variables. Exhibit 3 lists the final data set of dependent variables.

Exhibit 3. Final Set of Dependent Variables

Variable	Description	Type
`KM_number`	Knowledge management article containing default attributes and questions for service desk analyst use	Categorical
`Urgency`	Indicates incident resolution urgency	Categorical
`Count_Related_Interactions`	Number of updates or changes to the incident record	Continuous
`Count_Related_Incidents`	Number of similar or related incidents (child records)	Continuous

Variable	Description	Type
`Count_Related_Changes`	Number of Change Management records associated with the incident	Continuous
`Open_Time_HourOfDay`	Date and time of incident creation	Categorical
`Open_Time_DayOfWeek`	Date and time of incident creation	Categorical
`CI_TypeSubType_aff`	Concatenation of the top-level and second-level categories for the affected CI	Categorical
`Service_Component_WBS_aff`	Service component identifier for the affected CI	Categorical

3.2 Techniques and Tools

Data extraction and preparation steps leveraged techniques described in Exhibit 4.

Exhibit 4. Data Extraction and Preparation Techniques – Advantages and Disadvantages

Techniques	Justification	Advantages	Disadvantages
Convert data types	Need to manipulate data stored as strings or numbers for recognition as type required during analysis	Increases data consistency and integrity	Additional time required
Aggregate variables	Need to consolidate multiple categorical variables into a single variable	Reduces collinearity	Additional time required
Engineer target variable	Need to create missing variable based on existing data	Establishes required data point	Introduces business rule assumptions Additional time required
Bin datetime variables	Need to consolidate multiple continuous values into discrete number of levels	Reduce variable dimensionality	Additional time required
Filter dataset by timeframe	Need to limit data set based on subject matter expertise	Reduce data	Introduces business case assumptions Additional time required

Tools leveraged during this stage of the study included Python, pandas, NumPy, and Pandas Profiling. The mature, stable, well-documented nature of Python, the extensive availability of examples and tutorials, and availability of the selected libraries contributed to selecting Python for this project. Additionally, this project used Jupyter Notebooks to integrate discussions and narrative with the data analytics code. "For data scientists, Jupyter has emerged as a de facto standard, says Lorena Barba, a mechanical and aeronautical engineer at George Washington University in Washington DC" (Perkel, 2018). The availability of this computational notebook capability also contributed to the selection of Python. Exhibit 5 describes the tools used for data extraction and preparation along with an advantage/disadvantage summary.

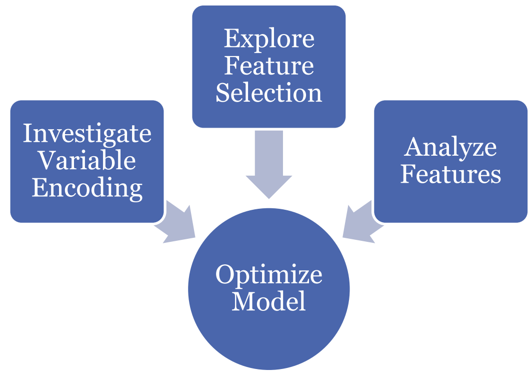
Exhibit 5. Data Extraction and Preparation Tools – Advantages and Disadvantages

Tools	Description	Advantages	Disadvantages
Python (*Welcome to Python.org*, n.d.)	Programming language	Free, open source reduced costs	Learning curve added time
pandas (*pandas—Python Data Analysis Library*, n.d.)	Data analysis and manipulation tool	Free, open source reduced costs	Learning curve added time
NumPy (*NumPy—NumPy*, n.d.)	Scientific computing package for Python	Free, open source reduced costs	Learning curve added time
Pandas Profiling (*Pandas_profiling API documentation*, n.d.)	Generates data set profiles identifying data types, unique values, missing values, quantile statistics, histograms, correlations	Free, open source reduced costs Ease of use reduced time	Version compatibility issues added time
Jupyter Notebooks (*Project Jupyter*, n.d.)	Web-based application that integrates code, equations, visualizations, and narrative text	Free, open source reduced costs Ease of use reduced time	Configuration issues added time

4 Analysis

With the goal of identifying factors that predict SLA-at-Risk incidents, the study focused on developing a logistic regression model supported by factor analysis techniques.

4.1 Process Overview



Analysis developed a logistic regression model that predicts the status of `SLAFail`. Given the binary nature of the dependent, target variable and mixed nature of the independent variables, Tufféry recommends applying logistic regression techniques as an appropriate predictive method (Tufféry, 2011, p. 170). Analysis investigated three feature-focused aspects: variable encoding, feature selection, and feature analysis. Optimization followed an iterative approach to refining the model. The standard procedure for each of the above steps included splitting the source data set into a training data set and a testing data set. The training data set provided the input for generating the model and the testing data set contributed to the generated evaluation metrics. Throughout the analysis process, decisions stemmed from review of classification accuracy rates and the Area Under the Curve (AUC) score obtained from the Receiver Operating Characteristic (ROC) curve diagnostic (Tufféry, 2011, pp. 454, 458).

4.2 Calculations and Results

4.2.1 Investigate Variable Encoding

The source data set contains both categorical and numeric data. This step investigated a variety of encoding techniques and tools. Exhibit 6 presents model evaluation metrics used to select an encoder for use in subsequent steps. The methodology executed the same steps while varying only the encoder used on categorical variables. This analyst chose to move forward with the Weight of Evidence (WOE) encoder given the greatest AUC. See the appendices for the WOE calculations performed and the results obtained.

Exhibit 6. Comparison of Model Metrics among Encoders

Name	Cross Validation Accuracy	F1 Score	AUC
MEstimate Encoder	0.73086	0.65857	0.77668
Target Encoder	0.73394	0.65680	0.77678
WOE Encoder	0.73776	0.66146	**0.78063**
Helmert Encoder	0.73678	0.66134	0.76650

4.2.2 Explore Feature Selection

The study evaluated the effectiveness of a variety of automated feature selection techniques available from SciKit-Learn (*1.13. Feature selection—Scikit-learn 0.22.1 documentation*, n.d.). Holding all other parameters equal, analysis used the feature selection methods listed in Exhibit 7 and elected to move forward with the KBest F-Classification (ANOVA F-value) method due to the highest AUC. See the appendices for the KBest F-Classification (ANOVA F-value) calculations performed and the results obtained.

Exhibit 7. Comparison of Model Metrics among Feature Selection Methods

Name	Cross Validation Accuracy	F1 Score	AUC
KBest F-Classification (ANOVA F-value)	0.737026	0.657922	**0.780698**
KBest Mutual Information	0.737107	0.654011	0.777989
Recursive Feature Extraction	0.735119	0.656594	0.779613

4.2.3 Analyze Features

Considering the identification of discriminating variables as the primary purpose of the study, the project leveraged a variety of feature analysis techniques. This section describes the techniques used and presents the findings. Techniques used included Principal Component Analysis (PCA) and Hierarchical Cluster Analysis.

4.2.3.1 Principal Component Analysis (PCA)

While investigating feature variability, PCA results showed the need for eight principle components to explain at least 95% of the variance (Exhibit 8). Further analysis of the variance within the first two principal components revealed the following (Exhibit 9):

- Three variables account for the greatest variance within `PC 01`:
  - `Service_Component_WBS_aff`
  - `KM_Number`
  - `CI_TypeSubType_aff`
- Two variables account for the greatest variance within `PC 02`:
  - `Count_Related_Incidents`
  - `Count_Related_Interactions`

Exhibit 8. PCA Explained and Cumulative Variance

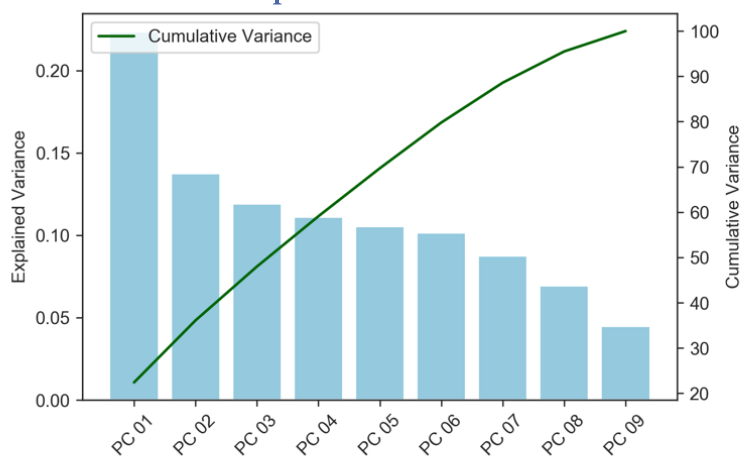
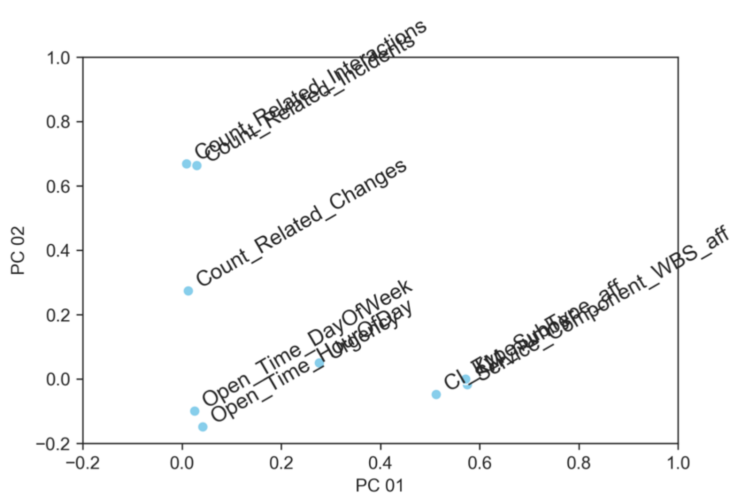


Exhibit 9. Contribution to Variance in First Two Principal Components



4.2.3.2 Hierarchical Cluster Analysis

Hierarchical cluster analysis contributed to the identification of variables having similar characteristics ((Tufféry, 2011, p. 236)). The first step created a dendrogram showing the sequence of partitions created by the agglomerative hierarchical clustering algorithm as shown in Exhibit 10. The second step evaluated the stability of the partitions created by the algorithm with the resulting chart shown in Exhibit 11.

Exhibit 10. Hierarchical Cluster Analysis Dendrogram

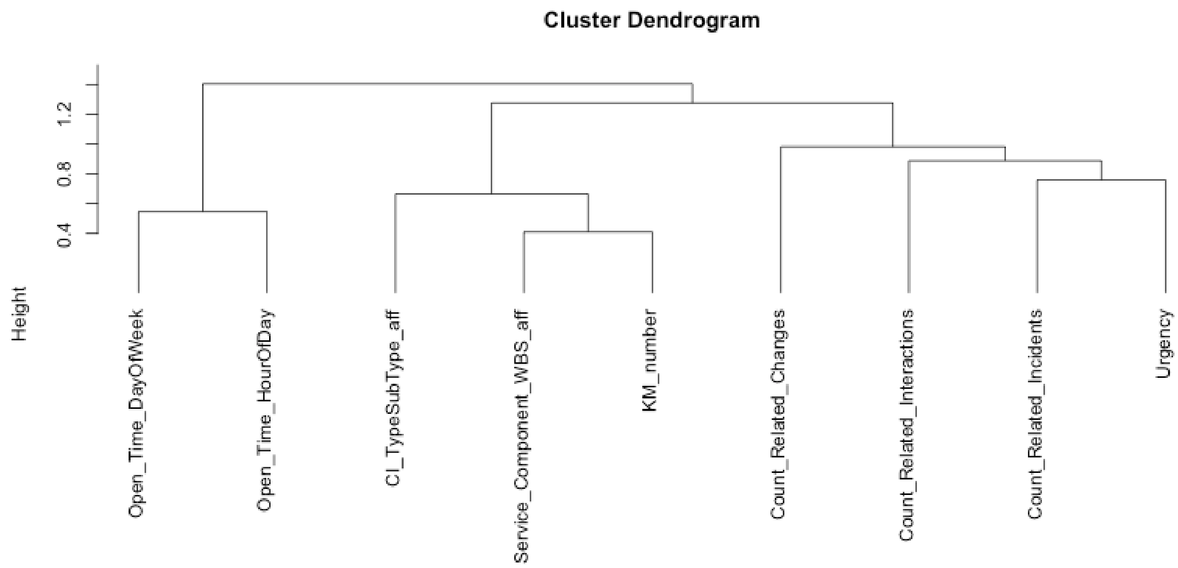
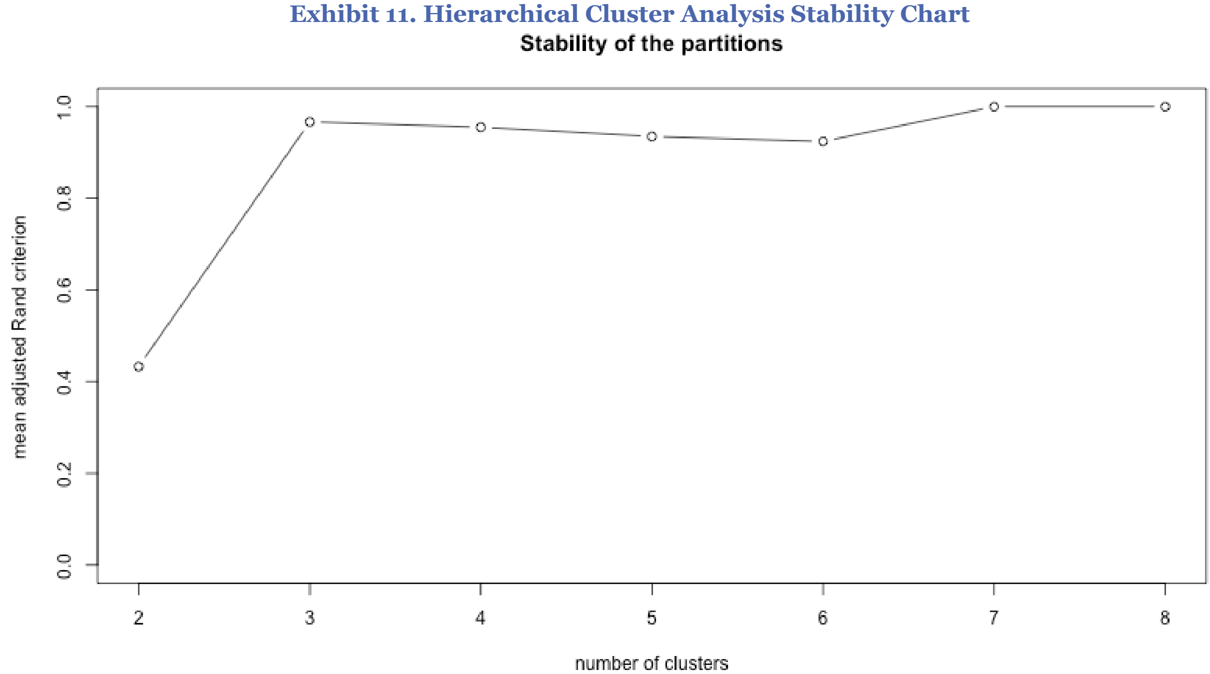


Exhibit 11. Hierarchical Cluster Analysis Stability Chart



At three clusters, the chart indicates the greatest gain in stability prior to leveling off. Exhibit 12 lists the clusters identified as a result of this analysis.

Exhibit 12. Identified Clusters

Cluster	Variables

1	`KM_number`
	`CI_TypeSubType_aff`
	`Service_Component_WBS_aff`

2	`Urgency`
	`Count_Related_Interactions`
	`Count_Related_Incidents`
	`Count_Related_Changes`

3	`Open_Time_HourOfDay` `Open_Time_DayOfWeek`

4.2.4 Optimize Model

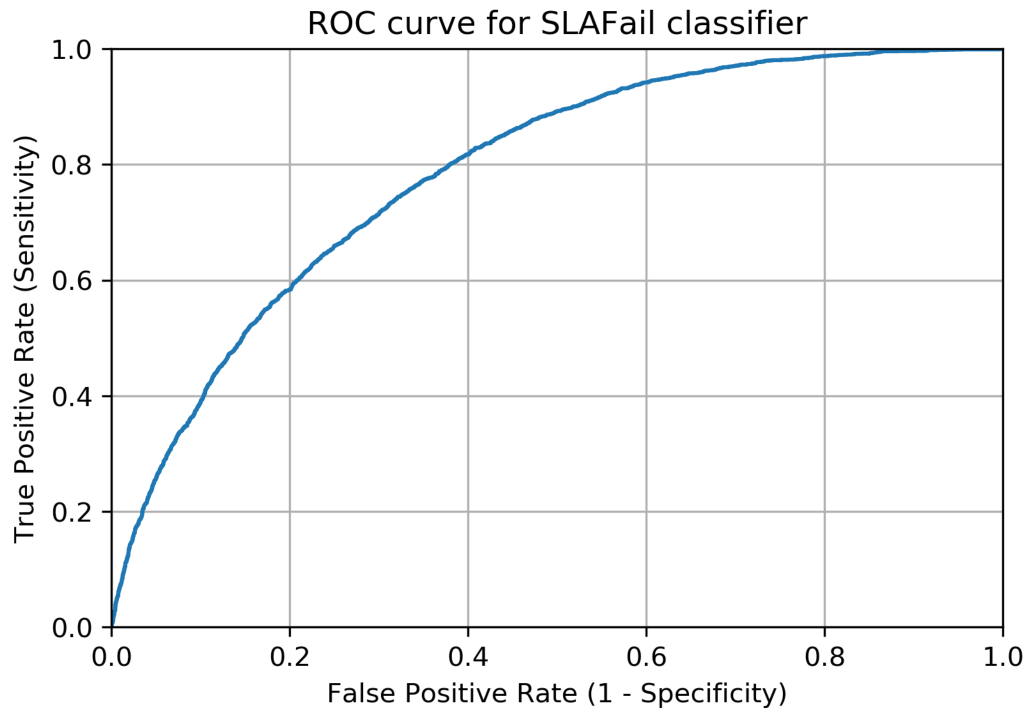
This step leverages the information and insight gained from the previous three steps to develop a logistic regression model that successfully predicts SLA compliance at the early stages of an incident's lifecycle. With a Null Accuracy of 70%, optimization efforts aim at achieving a greater Classification Accuracy score. Given the inability to obtain more training data or to add features, this analyst's optimization choices were limited to searching for less complicated/flexible models and looking at more complicate/flexible models (Vanderplas, n.d.). Exhibit 13 presents the results of each iteration performed during the Optimize Model step. See the appendices for an example executed notebook.

Exhibit 13. Optimization Results

Iteration		Classification Accuracy	AUC
1. Use automated feature selection of the k-best features based on F-score	Less complicated/ flexible model	0.746852	0.780629
2. Use automated feature selection by False Positive Rate with an acceptance threshold of 0.05 for alpha/p-value	Less complicated/ flexible model	0.746758	0.780663
3. Add interaction terms, use automated feature selection of the k-best features based on F-score	More complicated/ flexible model	**0.751775**	**0.786672**
4. Add interaction terms, use automated feature selection by False Positive Rate with an acceptance threshold of 0.05 for alpha/p-value	More complicated/ flexible model	0.748178	0.782095

The third iteration produced slightly higher accuracy and AUC scores than all other attempted methods. Exhibit 14 shows the ROC curve.

Exhibit 14. ROC Curve for Optimized Model



While the third iteration provided slightly better results, the introduction of interaction terms significantly increased the model's complexity. SciKit Learn's `PolynomialFeatures` created the interaction terms while a cross-validation grid search (`GridSearchCV`) selected the *k* best features based on f-scores. This resulted in the logistic regression model using 45 terms, versus the original nine.

Given the minimal performance improvement of the more complex model, this analyst recommends the second iteration step.

4.3 Techniques and Tools Used

Analysis leveraged a number of techniques supported by a variety of tools. Python and the SciKit-Learn machine learning library served as the primary tool set. The SciKit-Learn library's pipeline functionality automated repeatable steps. However, it introduced complexity in obtaining results from the sequence of steps, thus adding time and effort to reviewing and reporting on the output. This analyst encountered computing resource constraints when attempting to perform Hierarchical Cluster Analysis with Python and SciKit-Learn. As a result, R with the FactoMiner and CustOfVar libraries supplemented the tool set. As a free, open-source, well-supported, and well-documented tool, R provides advantages similar to Python (Tufféry, 2011, p. 126). The need for an alternative tool required iterative development in one tool and saving the results to a file for use in the other tool. While this could be considered a disadvantage, this analyst asserts that this modularized approach created greater flexibility in available options for downstream analysis. R's inherent graphical capabilities produce clear and usable plots with minimal configuration while the Python plotting libraries require additional knowledge and time to configure usable plots.

Exhibit 15. Techniques and Tools for Analysis – Advantages and Disadvantages

Techniques and Tools	Description	Advantages	Disadvantages
Weight of Evidence Coding `category_encoders.woe.WOEEncoder`	Converts categorical variables to numeric values as needed for logistic regression models using the log(odds) of the event (*SAS Training—Predictive Modeling Using Logistic Regression*, n.d.)	Returned highest AUC among tested encoders	Introduces an additional step

ANOVA F-value
`sklearn.feature_selection.SelectKBest`
`sklearn.feature_selection.f_classif`
`sklearn.feature_selection.SelectFpr`
	Measures the degree of linear dependency between random variables (*1.13. Feature selection—Scikit-learn 0.22.2 documentation*, n.d.)	Returned highest AUC among tested methods
		Valid for positive and negative values (chi-squared required only positive values)
			None identified

<table>
<tr><th>Techniques and Tools</th><th colspan="2">Description</th></tr>
<tr><td rowspan="3">Principal Component Analysis (PCA)
<code>sklearn.decomposition.PCA</code></td><td colspan="2">Groups variables based on their correlations (Tufféry, 2011, p. 175)</td></tr>
<tr><td>Advantages</td><td>Disadvantages</td></tr>
<tr><td>Industry accepted method for investigating variance and reducing dimensionality</td><td>Required encoding of categorical variables to numeric values</td></tr>
<tr><td rowspan="3">Hierarchical Cluster Analysis
<code>R</code>, <code>FactoMiner</code>, <code>ClustOfVar</code></td><td colspan="2">Groups data into disjoint clusters of observations (Hastie et al., n.d., p. 521)</td></tr>
<tr><td>Advantages</td><td>Disadvantages</td></tr>
<tr><td>Ability to identify similar variables
Highly interpretable visualization with dendrograms</td><td>Required use of R as an alternate tool</td></tr>
<tr><td rowspan="3">Logistic Regression
<code>sklearn.linear_model.LogisticRegression</code></td><td colspan="2">Statistical analysis technique for binary dependent variables (Tufféry, 2011, pp. 170–171)</td></tr>
<tr><td>Advantages</td><td>Disadvantages</td></tr>
<tr><td>Appropriate classification method given binary nature of the target variable</td><td>Requires careful interpretation and presentation of results</td></tr>
</table>

5 Data Summary and Implications

In response to the research question, this section summarizes analysis implications in the context of early identification of incidents likely to cause breaches in Service Level Agreement thresholds.

What factors predict Incident Management SLA compliance?

5.1 Discussion of Results

Given the study's research question and hypotheses, the project developed a logistic regression model predicting the final SLA status of an incident based on data available during the early stages of an incident's lifecycle. Recall:

H_0: Data contains no significant indicators about the final SLA status of an incident (the coefficients of a logistic regression model are zero, $\beta_i = 0$)
H_1: Data contains significant indicators about the final SLA status of an incident (the coefficients of a logistic regression model are not zero, $\beta_i \neq 0$)

If any one of the logistic regression model's coefficients significantly differs from zero, the study will accept the alternative hypothesis (H_1), otherwise the study will fail to reject the null hypothesis (H_0). Exhibit 16 lists the coefficients resulting from the selected optimization step of model development. With seven of the eight variables showing a significance level (P-value) less than alpha (0.05), the study accepts the alternative hypothesis that data does contain significant indicators of an incident's final SLA status. Note that automated feature selection removed `CI_TypeSubType_aff` as insignificant prior to model generation.

Exhibit 16. Logistic Regression Model Coefficients

Variable	Coefficient	F score	P-value
`Service_Component_WBS_aff`	0.1040	141.5840	0.0000
`Urgency`	0.1881	6104.4400	0.0000
`KM_number`	1.1217	177.7560	0.0000
`Count_Related_Interactions`	1.1726	679.7630	0.0000
`Count_Related_Incidents`	1.2459	902.1090	0.0000
`Count_Related_Changes`	0.0388	62.0758	0.0000
`Open_Time_HourOfDay`	0.1824	5.6072	0.0179
`Open_Time_DayOfWeek`	-0.0358	0.3519	0.5530

While the study provided statistical evidence for the acceptance of the alternative hypothesis, multiple limitations restrict the practical value of the resulting model. The best model identified by the study achieved a classification accuracy score of 78%, while the null accuracy (always selecting the majority event) resided at 70%. Given implementation costs, this analyst questions the value gained with only an 8% accuracy increase. Also, given the nature of the source data set containing anonymous information, the study was unable to investigate additional factors that could provide further discrimination and benefit to the model. Categories within Knowledge Management articles represented by `KM_number`, as well as data representing configuration item types (`Service_Component_WBS_aff`), could potentially improve the model.

5.2 Recommendations

Based on these results, this analyst recommends a course of action focused on further investigation of the specific Configuration Items and Knowledge Management articles causing incidents that exceed SLA thresholds. Analysis also showed that a subset of variables provide Incident Management professionals with direction for swift identification of incidents that may breach an SLA threshold. Principal component analysis and the hierarchical cluster analysis both grouped the following variables together:

- `Service_Component_WBS_aff`
- `KM_Number`
- `CI_TypeSubType_aff`

With this understanding, coupled with the identification of `Service_Component_WBS_aff` and `KM_Number` as significant indicators of SLA-at-Risk, Incident Management professionals can prioritize attention on identifying specific items in the IT environment causing incidents that exceed SLA thresholds.

Recommendations for future study include seeking out additional data for use in model development and consideration of alternative classification models. As previously mentioned, additional information about categories of Knowledge Articles and Configuration Items could improve model accuracy. Further investigation of data describing the users impacted by incidents could provide additional discrimination unavailable from the source data set and the necessity of removing one of the two highly correlated variables (`Impact` and `Urgency`). Without additional data, two alternatives for further study exist. First, investigating the efficacy of other classification techniques may produce superior results, for example, decision trees, support vector machines (SVM), naive Bayesian classifiers, or *k* nearest neighbors. Second, in-depth factor analysis of individuals could identify the specific Knowledge Articles and Configuration Items with the highest contribution towards SLA breaches.

6 References

1.13. Feature selection—Scikit-learn 0.22.1 documentation. (n.d.). Retrieved March 4, 2020, from https://scikit-learn.org/stable/modules/feature_selection.html

1.13. Feature selection—Scikit-learn 0.22.2 documentation. (n.d.). Retrieved March 8, 2020, from https://scikit-learn.org/stable/modules/feature_selection.html#univariate-feature-selection

4TU.Centre for Research Data. (2016, August). *General terms of use for 4TU.Centre for Research Data*. https://data.4tu.nl/repository/resource:terms_of_use/PDF

10th International Workshop on Business Process Intelligence 2014. (n.d.). https://www.win.tue.nl/bpi/doku.php?id=2014:challenge

Amaral, C., Fantinato, M., Reijers, H., & Peres, S. (2019). *Enhancing Completion Time Prediction Through Attribute Selection*. https://doi.org/10.1007/978-3-030-15154-6_1

Buffett, S., Emond, B., & Goutte, C. (2014). *Using Sequence Classification to Label Behavior from Sequential Event Logs*. 27.

Buhler, P., Callaghan, R. O., Aubry, S., DeJoy, D., Kuo, E., Shoup, N., Khosla, I., Ginsburg, M., Hartman, N., & McBride, N. S. (n.d.). *Service Desk and Incident Impact Patterns Following ITIL Change Implementation*. 36.

Hanna, A. (2011). *ITIL(r) glossary and abbreviations*. AXELOS Limited. https://www.axelos.com/corporate/media/files/glossaries/itil_2011_glossary_gb-v1-0.pdf

Hastie, T., Tibshirani, R., & Friedman, J. (n.d.). *The Elements of Statistical Learning*. 764.

Higgins, S. (2016, April 26). *How predictive analytics have turned Incident Management on its head* -. http://www.theitsmreview.com/2016/04/predictive-analytics-turned-incident-management-head/

Measuring Response and Resolution Times in Remedy | University IT. (n.d.). Stanford University IT. Retrieved February 19, 2020, from https://uit.stanford.edu/service/helpdesk/support/sla

Nisbet, R., Elder, J. F., & Miner, G. (2009). *Handbook of statistical analysis and data mining applications*. Academic Press/Elsevier.

NumPy—NumPy. (n.d.). Retrieved February 24, 2020, from https://numpy.org/

Pandas_profiling API documentation. (n.d.). Retrieved February 24, 2020, from https://pandas-profiling.github.io/pandas-profiling/docs/

pandas—Python Data Analysis Library. (n.d.). Retrieved February 24, 2020, from https://pandas.pydata.org/

Perkel, J. M. (2018). Why Jupyter is data scientists' computational notebook of choice. *Nature*, *563*(7729), 145–146. https://doi.org/10.1038/d41586-018-07196-1

Project Jupyter. (n.d.). Retrieved February 24, 2020, from https://www.jupyter.org

Quick reference BPI Challenge 2014. (n.d.). Retrieved February 17, 2020, from https://www.win.tue.nl/bpi/lib/exe/fetch.php?media=2014:quick_reference_bpi_challenge_2014.pdf

Sarnovsky, M., & Surma, J. (2018). PREDICTIVE MODELS FOR SUPPORT OF INCIDENT MANAGEMENT PROCESS IN IT SERVICE MANAGEMENT. *Acta Electrotechnica et Informatica*, *18*(1), 57–62. https://doi.org/10.15546/aeei-2018-0009

SAS Training—Predictive Modeling Using Logistic Regression. (n.d.). SAS Virtual Learning Environment. Retrieved February 19, 2020, from https://support.sas.com/edu/schedules.html?id=13086&ctry=US

Tufféry, S. (2011). *Data Mining and Statistics for Decision Making: Tufféry/Data Mining and Statistics for Decision Making*. John Wiley & Sons, Ltd. https://doi.org/10.1002/9780470979174

Van Der Aalst, W. M. P. (2018). *Process Mining: Data Science in Action*. Springer Publishing Company, Incorporated.

Van Dongen, B. F. (Boudewijn). (2014). *BPI Challenge 2014: Incident details*. Rabobank Nederland. https://doi.org/10.4121/UUID:3CFA2260-F5C5-44BE-AFE1-B70D35288D6D

Vanderplas, J. (n.d.). *Hyperparameters and Model Validation*. Python Data Science Handbook. Retrieved March 7, 2020, from https://jakevdp.github.io/PythonDataScienceHandbook/05.03-hyperparameters-and-model-validation.html

Welcome to Python.org. (n.d.). Python.Org. Retrieved February 24, 2020, from https://www.python.org/

7 Appendix Describing the Source Data Set

Variable	Type	Description
`Alert_Status`	Categorical, CONSTANT value "closed"	Status for monitoring service levels
`Category`	Categorical, four levels	Grouping of types of incidents
`CI_Name_aff`	Categorical, HIGH CARDINALITY	Configuration Item (CI) where a disruption is noticed (Affected CI)
`CI_Name_CBy`	Categorical, HIGH CARDINALITY	Configuration Item (CI) which caused the disruption (Caused-By CI)
`CI_Subtype_aff`	Categorical, HIGH CARDINALITY	Second-level category of Affected CI
`CI_Subtype_CBy`	Categorical, HIGH CARDINALITY	Second-level category of Caused-By CI
`CI_Type_aff`	Categorical, 13 levels	
`CI_Type_CBy`	Categorical, 14 levels	Top-level category of Caused-By CI
`Close_Time`	Date Minimum: 2013-10-01 06:45:43 Maximum: 2014-03-31 22:47:32	Date and time of incident closure
`Closure_Code`	Categorical, 15 levels 1.0% missing	Classification of disruption type
`Count_Reassignments`	Continuous, ZEROS (58.9%)	Number of times responsibility for the incident changed
`Count_Related_Changes`	Continuous, MISSING (98.8%)	Number of Change Management records associated with the incident
`Count_Related_Incidents`	Continuous, MISSING (97.4%)	Number of similar or related incidents (child records)
`Count_Related_Interactions`	Continuous, SKEWED, MISSING (0.03%)	Number of updates or changes to the incident record
`Handle_Time_Hours`	Continuous	Time required to actively resolve the incident
`Impact`	Categorical, five levels	Impact of the disruption to the customer
`Incident_ID`	Categorical, UNIQUE, HIGH CARDINALITY	Unique identifier for each incident

Variable	Type	Description
KM_number	Categorical, HIGH CARDINALITY	Knowledge management article containing default attributes and questions for service desk analyst use
Open_Time	Date Minimum: 2012-02-05 13:32:57 Maximum: 2014-03-31 17:24:49	Date and time of incident creation
Priority	Categorical, five levels	Priority derived from the Impact and Urgency values
Related_Change	Categorical, MISSING (98.8%), HIGH CARDINALITY	Change record identifier (if only one change is related to the incident)
Related_Interaction	Categorical, HIGH CARDINALITY	Interaction record identifier (if only one interact is related to the incident)
Reopen_Time	Date Minimum: 2013-04-10 09:15:55 Maximum: 2014-03-31 16:21:15 MISSING (95.9%),	If the incident is re-opened shortly after closure based on customer feedback, the date and time the incident was re-opened
Resolved_Time	Date Minimum: 2013-10-01 06:45:36 Maximum: 2014-03-31 22:47:29 3.8% Missing	Date and time of incident resolution
Service_Component_WBS_aff	Categorical, HIGH CARDINALITY	Service component identifier for the Affected CI
ServiceComp_WBS_CBy	Categorical, HIGH CARDINALITY	Service component identifier for the Caused-By CI
Status	Categorical, two levels	Status of the incident
Urgency	Categorical, five levels	Indicates incident resolution urgency

8 Appendix of Executed Python Notebooks and Scripts

The following pages show the content and results generated by a few of the Jupyter Notebooks used throughout this project.

Report Section	Notebook Title	File Name	Page
3.1.1	01. Exploratory Data Analysis and Preliminary Cleaning	01. EDA_Detail_Incident.ipynb	8.1-1
3.1.2	02. Cleaning the Source Data Set	02. Cleaning_Detail_Incident.ipynb	8.2-1
3.1.3	03. Creating the Target Variable (SLAFail)	03. Create_SLAFail.ipynb	8.3-1
3.1.4	04. Final Data Preparation	04. Final Data Prep.ipynb	8.4-1
4.2.1	05.01.c Bare Bones Analysis Using a Weight of Evidence Encoder	05.01.c Bare Bones Analysis WOE Encoder.ipynb	8.5-1
4.2.2	05.02 Feature Selection KBest with ANOVA F-value Score Function	05.02.a Feature Selection KBest with f_classif.ipynb	8.6-1
4.2.4	06.01.b Optimize the Logistic Regression Model	06.01.b Optimize 2 Select FPR with f_classif and pval.ipynb	8.7-1

8.1 Notebook: 01. Exploratory Data Analysis and Preliminary Cleaning

Output from executed notebook begins on the next page.

01. Exploratory Data Analysis and Preliminary Cleaning

[1]
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib
from matplotlib import pyplot as plt
from pandas_profiling import ProfileReport
from pathlib import Path
```

[2]
```
df = pd.read_csv("data/Detail_Incident.csv", sep=";",
decimal=',')
```

[3]
```
df.columns
```

```
Index(['CI Name (aff)', 'CI Type (aff)', 'CI Subtype (aff)',
       'Service Component WBS (aff)', 'Incident ID', 'Status', 'Impact',
       'Urgency', 'Priority', 'Category', 'KM number', 'Alert Status',
       '# Reassignments', 'Open Time', 'Reopen Time', 'Resolved Time',
       'Close Time', 'Handle Time (Hours)', 'Closure Code',
       '# Related Interactions', 'Related Interaction', '# Related
Incidents',
       '# Related Changes', 'Related Change', 'CI Name (CBy)', 'CI Type
(CBy)',
       'CI Subtype (CBy)', 'ServiceComp WBS (CBy)', 'Unnamed: 28',
       'Unnamed: 29', 'Unnamed: 30', 'Unnamed: 31', 'Unnamed: 32',
       'Unnamed: 33', 'Unnamed: 34', 'Unnamed: 35', 'Unnamed: 36',
       'Unnamed: 37', 'Unnamed: 38', 'Unnamed: 39', 'Unnamed: 40',
       'Unnamed: 41', 'Unnamed: 42', 'Unnamed: 43', 'Unnamed: 44',
       'Unnamed: 45', 'Unnamed: 46', 'Unnamed: 47', 'Unnamed: 48',
       'Unnamed: 49', 'Unnamed: 50', 'Unnamed: 51', 'Unnamed: 52',
       'Unnamed: 53', 'Unnamed: 54', 'Unnamed: 55', 'Unnamed: 56',
       'Unnamed: 57', 'Unnamed: 58', 'Unnamed: 59', 'Unnamed: 60',
       'Unnamed: 61', 'Unnamed: 62', 'Unnamed: 63', 'Unnamed: 64',
       'Unnamed: 65', 'Unnamed: 66', 'Unnamed: 67', 'Unnamed: 68',
       'Unnamed: 69', 'Unnamed: 70', 'Unnamed: 71', 'Unnamed: 72',
       'Unnamed: 73', 'Unnamed: 74', 'Unnamed: 75', 'Unnamed: 76',
       'Unnamed: 77'],
      dtype='object')
```

Remove empty rows and columns

```
[4]    df.dropna(axis='columns', how='all', inplace=True)
```

```
[5]    df.dropna(axis='rows', how='all', inplace=True)
```

```
[6]    df.shape
```

```
(46606, 28)
```

Adjust column names for easier reference

```
[7]    df.columns = df.columns.str.replace(' ', '_')
       df.columns = df.columns.str.replace('(', '')
       df.columns = df.columns.str.replace(')', '')
       df.columns = df.columns.str.replace('#', 'Count')
```

```
[8]    df.columns
```

```
Index(['CI_Name_aff', 'CI_Type_aff', 'CI_Subtype_aff',
       'Service_Component_WBS_aff', 'Incident_ID', 'Status', 'Impact',
       'Urgency', 'Priority', 'Category', 'KM_number', 'Alert_Status',
       'Count_Reassignments', 'Open_Time', 'Reopen_Time',
'Resolved_Time',
       'Close_Time', 'Handle_Time_Hours', 'Closure_Code',
       'Count_Related_Interactions', 'Related_Interaction',
       'Count_Related_Incidents', 'Count_Related_Changes',
'Related_Change',
       'CI_Name_CBy', 'CI_Type_CBy', 'CI_Subtype_CBy',
'ServiceComp_WBS_CBy'],
      dtype='object')
```

Convert date columns to datetime

```
[9]    colsDatetime = ['Open_Time', 'Reopen_Time', 'Resolved_Time',
       'Close_Time']
```

```
[10] for i in colsDatetime:
         df[i] = pd.to_datetime(df[i], format='%d/%m/%Y %H:%M:%S',
     errors='coerce' )
```

```
[11] df.dtypes
```

```
CI_Name_aff                              object
CI_Type_aff                              object
CI_Subtype_aff                           object
Service_Component_WBS_aff                object
Incident_ID                              object
Status                                   object
Impact                                  float64
Urgency                                  object
Priority                                float64
Category                                 object
KM_number                                object
Alert_Status                             object
Count_Reassignments                     float64
Open_Time                        datetime64[ns]
Reopen_Time                      datetime64[ns]
Resolved_Time                    datetime64[ns]
Close_Time                       datetime64[ns]
Handle_Time_Hours                       float64
Closure_Code                             object
Count_Related_Interactions              float64
Related_Interaction                      object
Count_Related_Incidents                 float64
Count_Related_Changes                   float64
Related_Change                           object
CI_Name_CBy                              object
CI_Type_CBy                              object
CI_Subtype_CBy                           object
ServiceComp_WBS_CBy                      object
dtype: object
```

Investigate `Urgency` as an object

```
[12] df.Urgency.value_counts()
```

```
4                18349
5                14094
3                 5362
4                 4239
5                 2685
3                 1174
2                  607
2                   89
1                    4
```

```
1                   2
5 - Very Low        1
Name: Urgency, dtype: int64
```

Fix `Urgency`, convert it along with `Impact` and `Priority` to string

```
[13]  df.Impact = df.Impact.astype(str).str[:1]
      df.Priority = df.Priority.astype(str).str[:1]
      df.Urgency = df.Urgency.astype(str).str[:1]
```

```
[14]  df.Urgency.value_counts()
```

```
4    22588
5    16780
3     6536
2      696
1        6
Name: Urgency, dtype: int64
```

```
[15]  df.dtypes
```

```
CI_Name_aff                          object
CI_Type_aff                          object
CI_Subtype_aff                       object
Service_Component_WBS_aff            object
Incident_ID                          object
Status                               object
Impact                               object
Urgency                              object
Priority                             object
Category                             object
KM_number                            object
Alert_Status                         object
Count_Reassignments                 float64
Open_Time                    datetime64[ns]
Reopen_Time                  datetime64[ns]
Resolved_Time                datetime64[ns]
Close_Time                   datetime64[ns]
Handle_Time_Hours                   float64
Closure_Code                         object
Count_Related_Interactions          float64
Related_Interaction                  object
Count_Related_Incidents             float64
Count_Related_Changes               float64
Related_Change                       object
CI_Name_CBy                          object
```

```
CI_Type_CBy                                object
CI_Subtype_CBy                             object
ServiceComp_WBS_CBy                        object
dtype: object
```

Output file and create profile report

```
[16]  with open("data/01.a.Detail_Incident.csv",'w') as f:
          df.to_csv(f, index=False)
```

```
[17]  profile = ProfileReport(df, title="Profile of BPIC 2014
      Detail_Incident Data after Initial Cleaning", html={'style':
      {'full_width': True}})
```

```
[18]  profile.to_file(Path(str("reports/01.b.Detail_Incident_Profile.ht
      ml")))
```

```
[ ]
```

8.2 Notebook: 02. Cleaning the Source Data Set

Output from executed notebook begins on the next page.

02. Cleaning the Source Data Set

```
[28]  import pandas as pd
      import numpy as np
      import seaborn as sns
      import matplotlib
      from matplotlib import pyplot as plt
      from pandas_profiling import ProfileReport
      from pathlib import Path
```

```
[29]  df = pd.read_csv("data/01.a.Detail_Incident.csv", parse_dates=
      ['Open_Time', 'Reopen_Time', 'Resolved_Time','Close_Time', ])
```

```
[30]  df.dtypes
```

```
CI_Name_aff                          object
CI_Type_aff                          object
CI_Subtype_aff                       object
Service_Component_WBS_aff            object
Incident_ID                          object
Status                               object
Impact                                int64
Urgency                               int64
Priority                              int64
Category                             object
KM_number                            object
Alert_Status                         object
Count_Reassignments                 float64
Open_Time                    datetime64[ns]
Reopen_Time                  datetime64[ns]
Resolved_Time                datetime64[ns]
Close_Time                   datetime64[ns]
Handle_Time_Hours                   float64
Closure_Code                         object
Count_Related_Interactions          float64
Related_Interaction                  object
Count_Related_Incidents             float64
Count_Related_Changes               float64
Related_Change                       object
CI_Name_CBy                          object
CI_Type_CBy                          object
CI_Subtype_CBy                       object
ServiceComp_WBS_CBy                  object
dtype: object
```

Drop Records where Resolved_Time is Missing

```
[31]  df.iloc[:,13:17].isnull().sum()
```

```
Open_Time            0
Reopen_Time      44322
Resolved_Time     1780
Close_Time           0
dtype: int64
```

```
[32]  df = df.dropna(subset=['Resolved_Time'])
```

```
[33]  df.iloc[:,13:17].isnull().sum()
```

```
Open_Time            0
Reopen_Time      42607
Resolved_Time        0
Close_Time           0
dtype: int64
```

Limit timeframe of all records

greater than 1 october 2013

less than 31 march 2014

```
[34]  df = df[df['Open_Time'] >= pd.to_datetime('10-01-2013')]
```

```
[35]  df.iloc[:,13:17].describe()
```

	Open_Time	Reopen_Time	Resolved_Time	Close_Time
count	43709	2038	43709	43709

	Open_Time	Reopen_Time	Resolved_Time	Close_Time
unique	43455	2036	43496	43500
top	2014-01-22 15:46:06	2013-11-12 10:36:33	2013-11-22 16:34:33	2014-02-27 15:04:32
freq	3	2	3	3
first	2013-10-01 07:33:21	2013-10-01 11:43:47	2013-10-01 08:18:27	2013-10-01 08:18:30
last	2014-03-31 17:24:49	2014-03-31 16:21:15	2014-03-31 22:47:29	2014-03-31 22:47:32

Deal with Status of 'work in progress'

[36]
```
df.Status.value_counts()
```

```
Closed              43700
Work in progress        9
Name: Status, dtype: int64
```

[37]
```
df = df[ df['Status'] == 'Closed' ]
```

[38]
```
df.Status.value_counts()
```

```
Closed    43700
Name: Status, dtype: int64
```

Remove non-incident records

[39]
```
print(df.Category.value_counts())
```

```
incident                  35208
request for information    8482
complaint                     9
request for change            1
Name: Category, dtype: int64
```

```
[40] df = df[ df['Category'] == 'incident' ]
```

```
[41] print(df.Category.value_counts())
     print(df.Status.value_counts())
     print(df.Alert_Status.value_counts())
```

```
incident    35208
Name: Category, dtype: int64
Closed    35208
Name: Status, dtype: int64
closed    35208
Name: Alert_Status, dtype: int64
```

Deal with Reopen_Time Missing Values

```
[42] df.Reopen_Time.isnull().sum()
```

```
33782
```

```
[43] df['ReopenedFlag'] = ~ df.Reopen_Time.isnull()
```

```
[44] df['ReopenedFlag'] = df['ReopenedFlag'].astype(int)
```

```
[45] df['ReopenedFlag'].value_counts()
```

```
0    33782
1     1426
Name: ReopenedFlag, dtype: int64
```

Set Missing to Zero for `Count_Related_Changes`, `Count_Related_Incidents`, and `Count_Related_Interactions`

```
[46] print(df['Count_Related_Changes'].isnull().sum())
     print(df['Count_Related_Incidents'].isnull().sum())
     print(df['Count_Related_Interactions'].isnull().sum())
```

```
34732
34164
111
```

```
[47] df['Count_Related_Changes'] =
     df['Count_Related_Changes'].fillna(0)
     df['Count_Related_Incidents'] =
     df['Count_Related_Incidents'].fillna(0)
     df['Count_Related_Interactions'] =
     df['Count_Related_Interactions'].fillna(0)
```

```
[48] print(df['Count_Related_Changes'].isnull().sum())
     print(df['Count_Related_Incidents'].isnull().sum())
     print(df['Count_Related_Interactions'].isnull().sum())
```

```
0
0
0
```

Set Missing to "Not Applicable" for `Related_Change`

```
[49] df['Related_Change'].value_counts().sum()
```

```
476
```

```
[50] df['Related_Change'] = df['Related_Change'].fillna("Not
     Applicable")
```

```
[51] df['Related_Change'].value_counts()
```

```
Not Applicable    34732
C00003013           110
C00014762            78
#MULTIVALUE          18
```

C00001012	10
C00012714	10
C00000713	9
C00009165	7
C00009722	7
C00017302	5
C00008750	5
C00014221	5
C00006833	4
C00004344	3
C00015613	3
C00009821	3
C00000829	3
C00001807	3
C00001026	3
C00006448	2
C00012545	2
C00011501	2
C00013454	2
C00012116	2
C00002389	2
C00014458	2
C00003404	2
C00002268	2
C00016781	2
C00000527	2
C00007098	2
C00001250	2
C00016192	2
C00001507	2
C00001549	2
C00005866	2
C00004739	2
C00008442	2
C00013072	2
C00008726	2
C00008222	2
C00004294	2
C00007015	2
C00005261	2
C00011591	1
C00001137	1
C00016571	1
C00012062	1
C00013379	1
C00015705	1
C00007202	1
C00010941	1
C00004044	1
C00006401	1
C00006599	1
C00001730	1
C00004090	1
C00000360	1
C00015923	1
C00004994	1
C00007161	1
C00006745	1

C00001831	1
C00009025	1
C00010379	1
C00008467	1
C00007055	1
C00004385	1
C00017230	1
C00001062	1
C00006823	1
C00013606	1
C00006824	1
C00008356	1
C00015758	1
C00002378	1
C00014707	1
C00008486	1
C00005050	1
C00016689	1
C00010182	1
C00000385	1
C00015776	1
C00004490	1
C00015609	1
C00008700	1
C00009448	1
C00009947	1
C00014475	1
C00009567	1
C00011182	1
C00013064	1
C00014075	1
C00014624	1
C00000589	1
C00000600	1
C00007747	1
C00003040	1
C00009563	1
C00005456	1
C00007132	1
C00014360	1
C00010785	1
C00013595	1
C00016295	1
C00014661	1
C00018294	1
C00014375	1
C00014122	1
C00004950	1
C00014622	1
C00018435	1
C00004493	1
C00016153	1
C00011170	1
C00012038	1
C00004854	1
C00008054	1
C00000122	1
C00018267	1

C00015544	1
C00015025	1
C00010344	1
C00018403	1
C00011406	1
C00015140	1
C00011858	1
C00014296	1
C00001455	1
C00002178	1
C00017553	1
C00013740	1
C00009966	1
C00001667	1
C00014876	1
C00014981	1
C00007983	1
C00005369	1
C00004384	1
C00017136	1
C00018421	1
C00017031	1
C00017321	1
C00008787	1
C00006302	1
C00004614	1
C00015047	1
C00010749	1
C00010740	1
C00010259	1
C00013104	1
C00013982	1
C00009069	1
C00016233	1
C00011366	1
C00004679	1
C00007092	1
C00000596	1
C00013273	1
C00013125	1
C00005110	1
C00004549	1
C00007263	1
C00001215	1
C00017594	1
C00000633	1
C00005847	1
C00012923	1
C00005815	1
C00013867	1
C00003624	1
C00002337	1
C00018549	1
C00010314	1
C00017161	1
C00005858	1
C00007572	1
C00002375	1

```
C00007099             1
C00000050             1
C00003468             1
C00002007             1
C00006422             1
C00015040             1
Name: Related_Change, dtype: int64
```

Drop columns

- with constant values,
- longer needed (`Reopen_Time`)

```
[52]  df = df.drop(['Category', 'Status', 'Alert_Status',
      'Reopen_Time'], axis='columns')
```

```
[53]  df.columns
```

```
Index(['CI_Name_aff', 'CI_Type_aff', 'CI_Subtype_aff',
       'Service_Component_WBS_aff', 'Incident_ID', 'Impact', 'Urgency',
       'Priority', 'KM_number', 'Count_Reassignments', 'Open_Time',
       'Resolved_Time', 'Close_Time', 'Handle_Time_Hours',
'Closure_Code',
       'Count_Related_Interactions', 'Related_Interaction',
       'Count_Related_Incidents', 'Count_Related_Changes',
'Related_Change',
       'CI_Name_CBy', 'CI_Type_CBy', 'CI_Subtype_CBy',
'ServiceComp_WBS_CBy',
       'ReopenedFlag'],
      dtype='object')
```

END and OUTPUT

```
[54]  with open("data/02.a.Detail_Incident.csv",'w') as f:
          df.to_csv(f, index=False)
```

```
[55]  df.reset_index(drop=True, inplace=True)
      profile = ProfileReport(df, title="Profile of BPIC 2014
      Detail_Incident Data after Secondary Cleaning", html={'style':
```

```
{'full_width': True}})
```

```
[56]  profile.to_file(Path(str("reports/02.b.Detail_Incident_Profile.ht
      ml")))
```

```
[ ]
```

8.3 Notebook: 03. Creating the Target Variable (SLAFail)

Output from executed notebook begins on the next page.

03. Creating the Target Variable (`SLAFail`)

```
[1]  import pandas as pd
     import numpy as np
     import seaborn as sns
     import matplotlib
     from matplotlib import pyplot as plt
     from pandas_profiling import ProfileReport
     from pathlib import Path
```

```
[2]  df = pd.read_csv("data/02.a.Detail_Incident.csv", parse_dates=
     ['Open_Time', 'Resolved_Time','Close_Time'])
```

```
[3]  df.dtypes
```

```
CI_Name_aff                          object
CI_Type_aff                          object
CI_Subtype_aff                       object
Service_Component_WBS_aff            object
Incident_ID                          object
Impact                                int64
Urgency                               int64
Priority                              int64
KM_number                            object
Count_Reassignments                 float64
Open_Time                    datetime64[ns]
Resolved_Time                datetime64[ns]
Close_Time                   datetime64[ns]
Handle_Time_Hours                   float64
Closure_Code                         object
Count_Related_Interactions          float64
Related_Interaction                  object
Count_Related_Incidents             float64
Count_Related_Changes               float64
Related_Change                       object
CI_Name_CBy                          object
CI_Type_CBy                          object
CI_Subtype_CBy                       object
ServiceComp_WBS_CBy                  object
ReopenedFlag                          int64
dtype: object
```

```
[4] df.head()
```

	CI_Name_aff	CI_Type_aff	CI_Subtype_aff	Service_Component_WBS
0	APP000005	application	Citrix	WBS000292
1	DSK000457	computer	Desktop	WBS000187
2	SBA000263	application	Server Based Application	WBS000072
3	SBA000154	application	Server Based Application	WBS000027
4	LAP000019	computer	Laptop	WBS000091

5 rows × 25 columns

```
[5] df['TimeToResolve'] = df.Resolved_Time - df.Open_Time
```

```
[6] df.TimeToResolve.describe()
```

```
count                          35208
mean       3 days 16:21:45.273148
std       10 days 08:24:08.475153
min              0 days 00:00:17
25%        0 days 01:12:33.250000
50%        0 days 16:20:28.500000
75%        3 days 02:57:33.500000
max            175 days 06:40:30
Name: TimeToResolve, dtype: object
```

```
[7] df.TimeToResolve.mode()
```

```
0   00:08:22
dtype: timedelta64[ns]
```

```
[8] df.head()
```

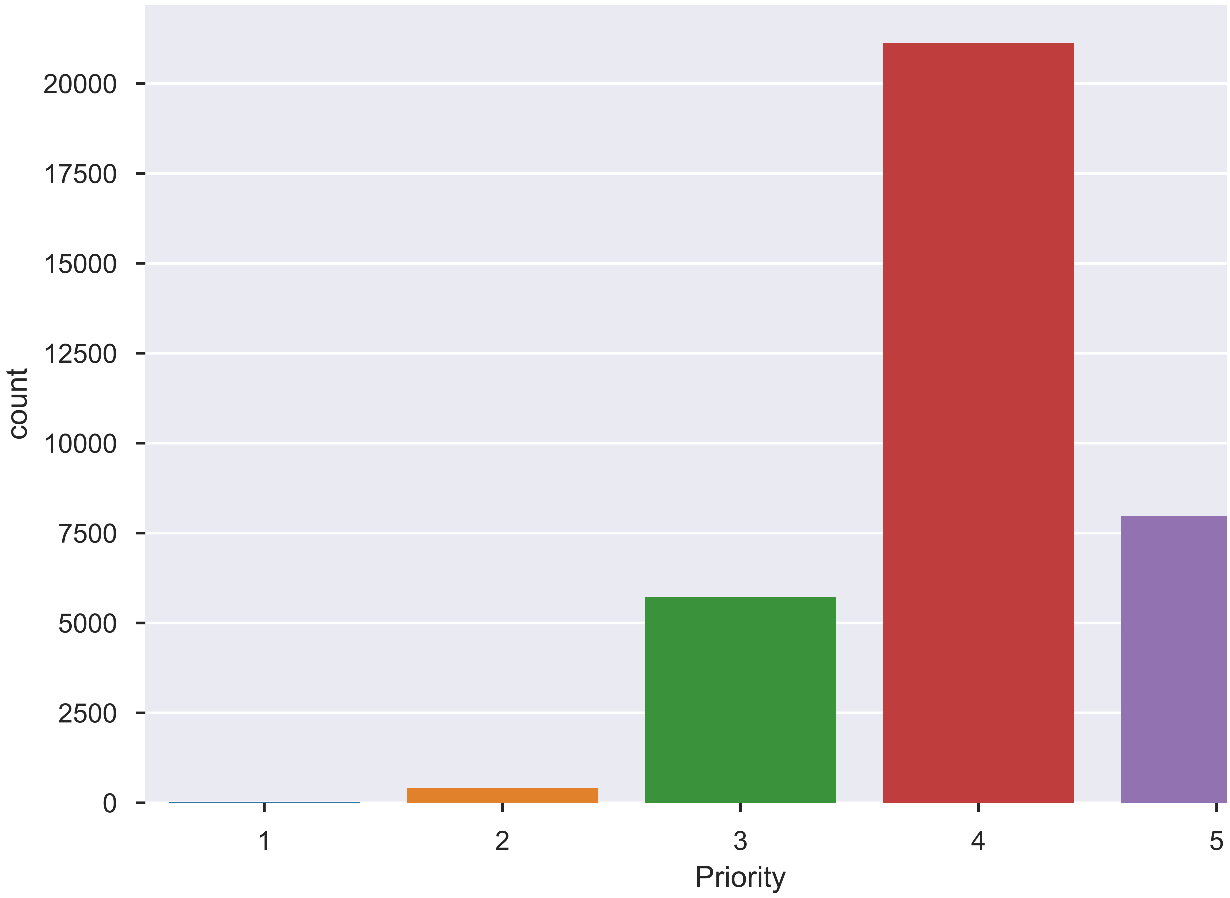
	CI_Name_aff	CI_Type_aff	CI_Subtype_aff	Service_Component_WBS
0	APP000005	application	Citrix	WBS000292
1	DSK000457	computer	Desktop	WBS000187

	CI_Name_aff	CI_Type_aff	CI_Subtype_aff	Service_Component_WBS
2	SBA000263	application	Server Based Application	WBS000072
3	SBA000154	application	Server Based Application	WBS000027
4	LAP000019	computer	Laptop	WBS000091

5 rows × 26 columns

```
[9]  sns.countplot(x='Priority', data=df)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a2c3d5d90>
```




```
[10]  df['TimeToResolve_Minutes'] = df.TimeToResolve.dt.total_seconds()
      / 60
```

```
[11]  df.head()
```

	CI_Name_aff	CI_Type_aff	CI_Subtype_aff	Service_Component_WBS
0	APP000005	application	Citrix	WBS000292
1	DSK000457	computer	Desktop	WBS000187
2	SBA000263	application	Server Based Application	WBS000072
3	SBA000154	application	Server Based Application	WBS000027
4	LAP000019	computer	Laptop	WBS000091

5 rows × 27 columns

SLA Business Rule

Priority	SLA in Minutes	SLA in Hours	SLA in Days
1 Very High	240	4	0.16
2 High	480	8	0.3
3 Medium	1440	24	1
4 Low	2880	48	2
5 Very Low	5760	96	4

SLAFail = (Priority == 1 & TimeToResolve_Minutes > 240) | (Priority == 2 & TimeToResolve_Minutes > 480) | (Priority == 3 & TimeToResolve_Minutes > 1440) | (Priority == 4 & TimeToResolve_Minutes > 2880) | (Priority == 5 & TimeToResolve_Minutes > 5760)

[12]
```
df['SLAFail'] = ( (df['Priority'] == 1) &
(df['TimeToResolve_Minutes'] > 240) ) | ( (df['Priority'] == 2) &
(df['TimeToResolve_Minutes'] > 480) ) | ( (df['Priority'] == 3) &
(df['TimeToResolve_Minutes'] > 1440) ) | ( (df['Priority'] == 4)
& (df['TimeToResolve_Minutes'] > 2880) ) | ( (df['Priority'] ==
5) & (df['TimeToResolve_Minutes'] > 5760) )
```

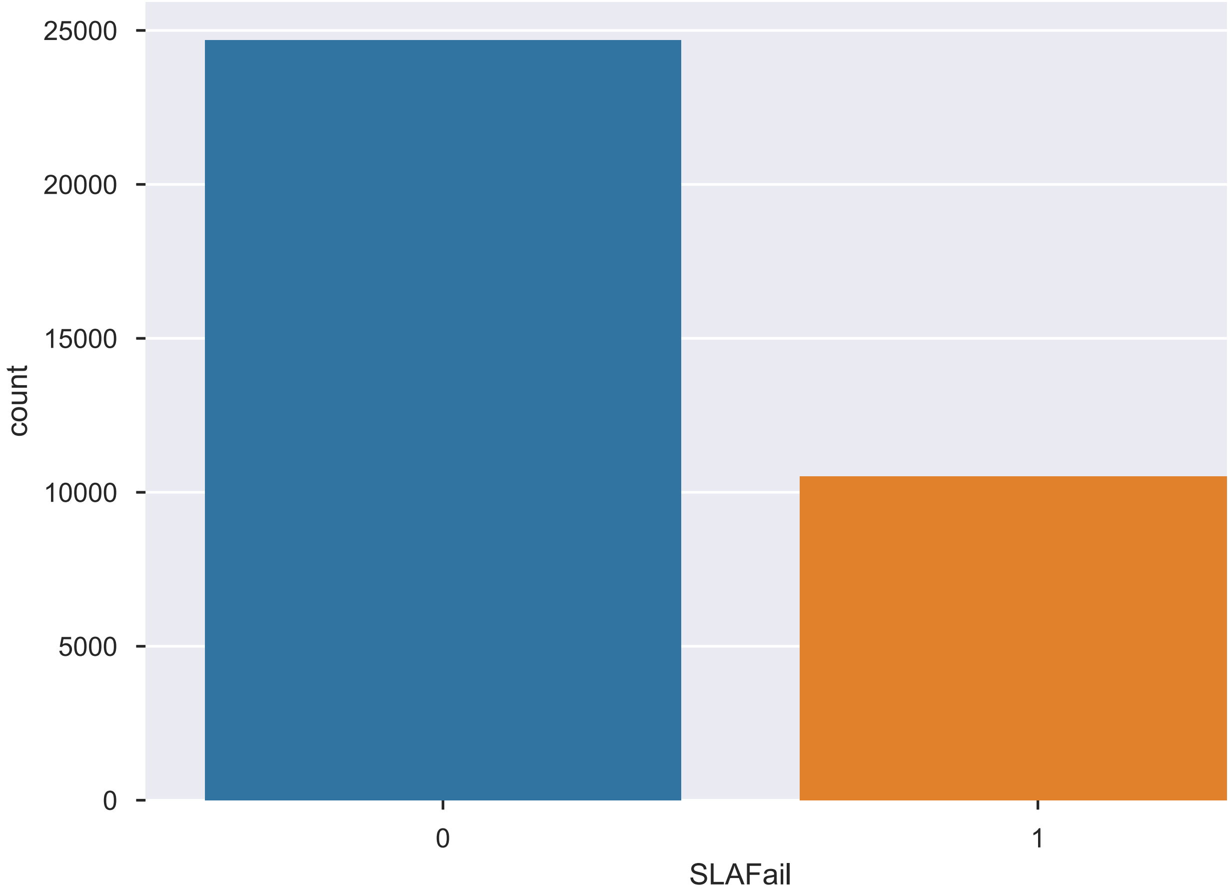
[13]
```
df.SLAFail = df.SLAFail.astype(int)
```

[14]
```
df.SLAFail.value_counts(normalize=True)
```

```
0    0.701261
1    0.298739
Name: SLAFail, dtype: float64
```

```
[15]  sns.countplot(x='SLAFail', data=df)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a273d6450>
```




```
[16]  #
      df = df.drop(['TimeToResolve'], axis='columns')
```

END and OUTPUT

```
[17]  with open("data/03.a.Detail_Incident.csv",'w') as f:
          df.to_csv(f, index=False)
```

```
[18]  df.reset_index(drop=True, inplace=True)
```

```
profile = ProfileReport(df, title="Profile of BPIC 2014
Detail_Incident Data after Adding SLAFail", html={'style':
{'full_width': True}})
```

[19]
```
profile.to_file(Path(str("reports/03.b.Detail_Incident_Profile.ht
ml")))
```

[ ]

8.4 Notebook: 04. Final Data Preparation

Output from executed notebook begins on the next page.

04. Final Data Preparation

```
[1]  import pandas as pd
     import numpy as np
     import seaborn as sns
     import matplotlib
     from matplotlib import pyplot as plt
     from pandas_profiling import ProfileReport
     from pathlib import Path
```

```
[2]  df = pd.read_csv("data/03.a.Detail_Incident.csv", parse_dates=
     ['Open_Time',  'Resolved_Time','Close_Time'])
```

```
[3]  df.dtypes
```

```
CI_Name_aff                              object
CI_Type_aff                              object
CI_Subtype_aff                           object
Service_Component_WBS_aff                object
Incident_ID                              object
Impact                                    int64
Urgency                                   int64
Priority                                  int64
KM_number                                object
Count_Reassignments                     float64
Open_Time                        datetime64[ns]
Resolved_Time                    datetime64[ns]
Close_Time                       datetime64[ns]
Handle_Time_Hours                       float64
Closure_Code                             object
Count_Related_Interactions              float64
Related_Interaction                      object
Count_Related_Incidents                 float64
Count_Related_Changes                   float64
Related_Change                           object
CI_Name_CBy                              object
CI_Type_CBy                              object
CI_Subtype_CBy                           object
ServiceComp_WBS_CBy                      object
ReopenedFlag                              int64
TimeToResolve_Minutes                   float64
SLAFail                                   int64
dtype: object
```

```
[4] df.head()
```

	CI_Name_aff	CI_Type_aff	CI_Subtype_aff	Service_Component_WBS
0	APP000005	application	Citrix	WBS000292
1	DSK000457	computer	Desktop	WBS000187
2	SBA000263	application	Server Based Application	WBS000072
3	SBA000154	application	Server Based Application	WBS000027
4	LAP000019	computer	Laptop	WBS000091

5 rows × 27 columns

```
[5] df.Priority.value_counts()
```

```
4    21120
5     7962
3     5721
2      402
1        3
Name: Priority, dtype: int64
```

```
[6] df["Priority"] = pd.cut(x=df['Priority'], bins=[-1,1,2,3,4,5],
    labels=["1 Very High", "2 High", "3 Medium", "4 Low", "5 Very
    Low"])
    df["Impact"] = pd.cut(x=df['Impact'], bins=[-1,1,2,3,4,5],
    labels=["1 Very High", "2 High", "3 Medium", "4 Low", "5 Very
    Low"])
    df["Urgency"] = pd.cut(x=df['Urgency'], bins=[-1,1,2,3,4,5],
    labels=["1 Very High", "2 High", "3 Medium", "4 Low", "5 Very
    Low"])
```

```
[7] df.Priority.value_counts()
```

```
4 Low          21120
5 Very Low      7962
3 Medium        5721
2 High           402
```

```
1 Very High        3
Name: Priority, dtype: int64
```

[8]
```
df['Open_Time_HourOfDay'] = df.Open_Time.dt.hour
df['Resolved_Time_HourOfDay'] = df.Resolved_Time.dt.hour
df['Close_Time_HourOfDay'] = df.Close_Time.dt.hour
```

[9]
```
df.head()
```

	CI_Name_aff	CI_Type_aff	CI_Subtype_aff	Service_Component_WBS
0	APP000005	application	Citrix	WBS000292
1	DSK000457	computer	Desktop	WBS000187
2	SBA000263	application	Server Based Application	WBS000072
3	SBA000154	application	Server Based Application	WBS000027
4	LAP000019	computer	Laptop	WBS000091

5 rows × 30 columns

[10]
```
df['Open_Time_DayOfWeek'] = df.Open_Time.dt.day_name()
df['Resolved_Time_DayOfWeek'] = df.Resolved_Time.dt.day_name()
df['Close_Time_DayOfWeek'] = df.Close_Time.dt.day_name()
```

[11]
```
df.head()
```

	CI_Name_aff	CI_Type_aff	CI_Subtype_aff	Service_Component_WBS
0	APP000005	application	Citrix	WBS000292
1	DSK000457	computer	Desktop	WBS000187
2	SBA000263	application	Server Based Application	WBS000072
3	SBA000154	application	Server Based Application	WBS000027

	CI_Name_aff	CI_Type_aff	CI_Subtype_aff	Service_Component_WBS
4	LAP000019	computer	Laptop	WBS000091

5 rows × 33 columns

```
[12] df['OpenShift'] = pd.cut(x=df['Open_Time_HourOfDay'], bins=[-1,
     8, 16, 25], labels=['Night','Day','Evening'])
     df['ResolvedShift'] = pd.cut(x=df['Resolved_Time_HourOfDay'],
     bins=[-1, 8, 16, 25], labels=['Night','Day','Evening'])
     df['CloseShift'] = pd.cut(x=df['Close_Time_HourOfDay'], bins=[-1,
     8, 16, 25], labels=['Night','Day','Evening'])
```

```
[13] df.head()
```

	CI_Name_aff	CI_Type_aff	CI_Subtype_aff	Service_Component_WBS
0	APP000005	application	Citrix	WBS000292
1	DSK000457	computer	Desktop	WBS000187
2	SBA000263	application	Server Based Application	WBS000072
3	SBA000154	application	Server Based Application	WBS000027
4	LAP000019	computer	Laptop	WBS000091

5 rows × 36 columns

```
[14] df.columns
```

```
Index(['CI_Name_aff', 'CI_Type_aff', 'CI_Subtype_aff',
       'Service_Component_WBS_aff', 'Incident_ID', 'Impact', 'Urgency',
       'Priority', 'KM_number', 'Count_Reassignments', 'Open_Time',
       'Resolved_Time', 'Close_Time', 'Handle_Time_Hours',
'Closure_Code',
       'Count_Related_Interactions', 'Related_Interaction',
       'Count_Related_Incidents', 'Count_Related_Changes',
'Related_Change',
       'CI_Name_CBy', 'CI_Type_CBy', 'CI_Subtype_CBy',
'ServiceComp_WBS_CBy',
       'ReopenedFlag', 'TimeToResolve_Minutes', 'SLAFail',
```

```
       'Open_Time_HourOfDay', 'Resolved_Time_HourOfDay',
       'Close_Time_HourOfDay', 'Open_Time_DayOfWeek',
       'Resolved_Time_DayOfWeek', 'Close_Time_DayOfWeek', 'OpenShift',
       'ResolvedShift', 'CloseShift'],
      dtype='object')
```

```
[15] df['CI_TypeSubType_aff'] = df.CI_Type_aff + "-" +
     df.CI_Subtype_aff
     df['CI_TypeSubType_CBy'] = df.CI_Type_CBy + "-" +
     df.CI_Subtype_CBy
```

```
[16] df.head()
```

	CI_Name_aff	CI_Type_aff	CI_Subtype_aff	Service_Component_WBS
0	APP000005	application	Citrix	WBS000292
1	DSK000457	computer	Desktop	WBS000187
2	SBA000263	application	Server Based Application	WBS000072
3	SBA000154	application	Server Based Application	WBS000027
4	LAP000019	computer	Laptop	WBS000091

5 rows × 38 columns

```
[17] df = df.drop([ "CI_Type_aff", "CI_Subtype_aff", "CI_Type_CBy",
     "CI_Subtype_CBy"], axis='columns')
```

```
[18] df = df.drop(['Incident_ID', "Related_Interaction",
      "Related_Change"], axis='columns')
```

```
[19] df.columns
```

```
Index(['CI_Name_aff', 'Service_Component_WBS_aff', 'Impact', 'Urgency',
       'Priority', 'KM_number', 'Count_Reassignments', 'Open_Time',
       'Resolved_Time', 'Close_Time', 'Handle_Time_Hours',
'Closure_Code',
       'Count_Related_Interactions', 'Count_Related_Incidents',
```

```
       'Count_Related_Changes', 'CI_Name_CBy', 'ServiceComp_WBS_CBy',
       'ReopenedFlag', 'TimeToResolve_Minutes', 'SLAFail',
       'Open_Time_HourOfDay', 'Resolved_Time_HourOfDay',
       'Close_Time_HourOfDay', 'Open_Time_DayOfWeek',
       'Resolved_Time_DayOfWeek', 'Close_Time_DayOfWeek', 'OpenShift',
       'ResolvedShift', 'CloseShift', 'CI_TypeSubType_aff',
       'CI_TypeSubType_CBy'],
      dtype='object')
```

```
[20] dfAtOpen = df[['CI_Name_aff', 'Service_Component_WBS_aff',
'Impact', 'Urgency',
        'KM_number', 'Count_Related_Interactions',
'Count_Related_Incidents',
       'Count_Related_Changes', 'SLAFail',
       'Open_Time_HourOfDay',  'Open_Time_DayOfWeek',
       'CI_TypeSubType_aff']]
```

```
[21] dfAtOpen.columns
```

```
Index(['CI_Name_aff', 'Service_Component_WBS_aff', 'Impact', 'Urgency',
       'KM_number', 'Count_Related_Interactions',
'Count_Related_Incidents',
       'Count_Related_Changes', 'SLAFail', 'Open_Time_HourOfDay',
       'Open_Time_DayOfWeek', 'CI_TypeSubType_aff'],
      dtype='object')
```

```
[22] dfAtOpen.shape
```

```
(35208, 12)
```

END and OUTPUT

```
[23] with open("data/04.a.Detail_Incident.csv",'w') as f:
         df.to_csv(f, index=False)
```

```
[24] df.reset_index(drop=True, inplace=True)
     profile = ProfileReport(df, title="Profile of Final BPIC 2014
Detail Incident Data", html={'style': {'full_width': True}})
```

```
[25] profile.to_file(Path(str("reports/04.b.Detail_Incident_Profile.ht
     ml")))
```

```
[26] with open("data/04.a.Detail_Incident_AtOpen.csv",'w') as f:
         dfAtOpen.to_csv(f, index=False)
```

```
[27] dfAtOpen.reset_index(drop=True, inplace=True)
     profile = ProfileReport(dfAtOpen, title="Profile of Final BPIC
     2014 Detail Incident At Open Data", html={'style': {'full_width':
     True}})
```

```
[28] profile.to_file(Path(str("reports/04.b.Detail_Incident_AtOpen_Pro
     file.html")))
```

```
[ ]
```

This notebook captures a review of correlations among variables remaining in our preparede data set and results in the data set used in subsequent model development steps.

[1]
```
# Load libraries
import pandas as pd
import numpy as np

import statsmodels.api as sm
import category_encoders as ce

import matplotlib.pyplot as plt
import seaborn as sns

sns.set_style("ticks")
sns.set_palette("Blues")
```

Read Prepared Data

[2]
```
df = pd.read_csv("data/04.a.Detail_Incident_AtOpen.csv")
print("df.shape: " + str(df.shape))
print("df.columns: " + str(df.columns))
print("df.dtypes: \n" + str(df.dtypes))
```

```
df.shape: (35208, 12)
df.columns: Index(['CI_Name_aff', 'Service_Component_WBS_aff', 'Impact',
'Urgency',
       'KM_number', 'Count_Related_Interactions',
'Count_Related_Incidents',
       'Count_Related_Changes', 'SLAFail', 'Open_Time_HourOfDay',
       'Open_Time_DayOfWeek', 'CI_TypeSubType_aff'],
      dtype='object')
df.dtypes:
CI_Name_aff                   object
Service_Component_WBS_aff     object
Impact                        object
Urgency                       object
KM_number                     object
Count_Related_Interactions   float64
```

```
Count_Related_Incidents       float64
Count_Related_Changes         float64
SLAFail                         int64
Open_Time_HourOfDay             int64
Open_Time_DayOfWeek            object
CI_TypeSubType_aff             object
dtype: object
```

Set X and y

```
[3]  y = df.SLAFail
     y.shape
```

```
(35208,)
```

```
[4]  X = df.drop(['SLAFail'], axis='columns')
     X.shape
```

```
(35208, 11)
```

```
[5]  X.Open_Time_HourOfDay = X.Open_Time_HourOfDay.astype('object')
```

```
[6]  categorical_features = X.select_dtypes(include=
     ['object']).columns
     categorical_features
```

```
Index(['CI_Name_aff', 'Service_Component_WBS_aff', 'Impact', 'Urgency',
       'KM_number', 'Open_Time_HourOfDay', 'Open_Time_DayOfWeek',
       'CI_TypeSubType_aff'],
      dtype='object')
```

```
[7]  encoder = ce.WOEEncoder()
```

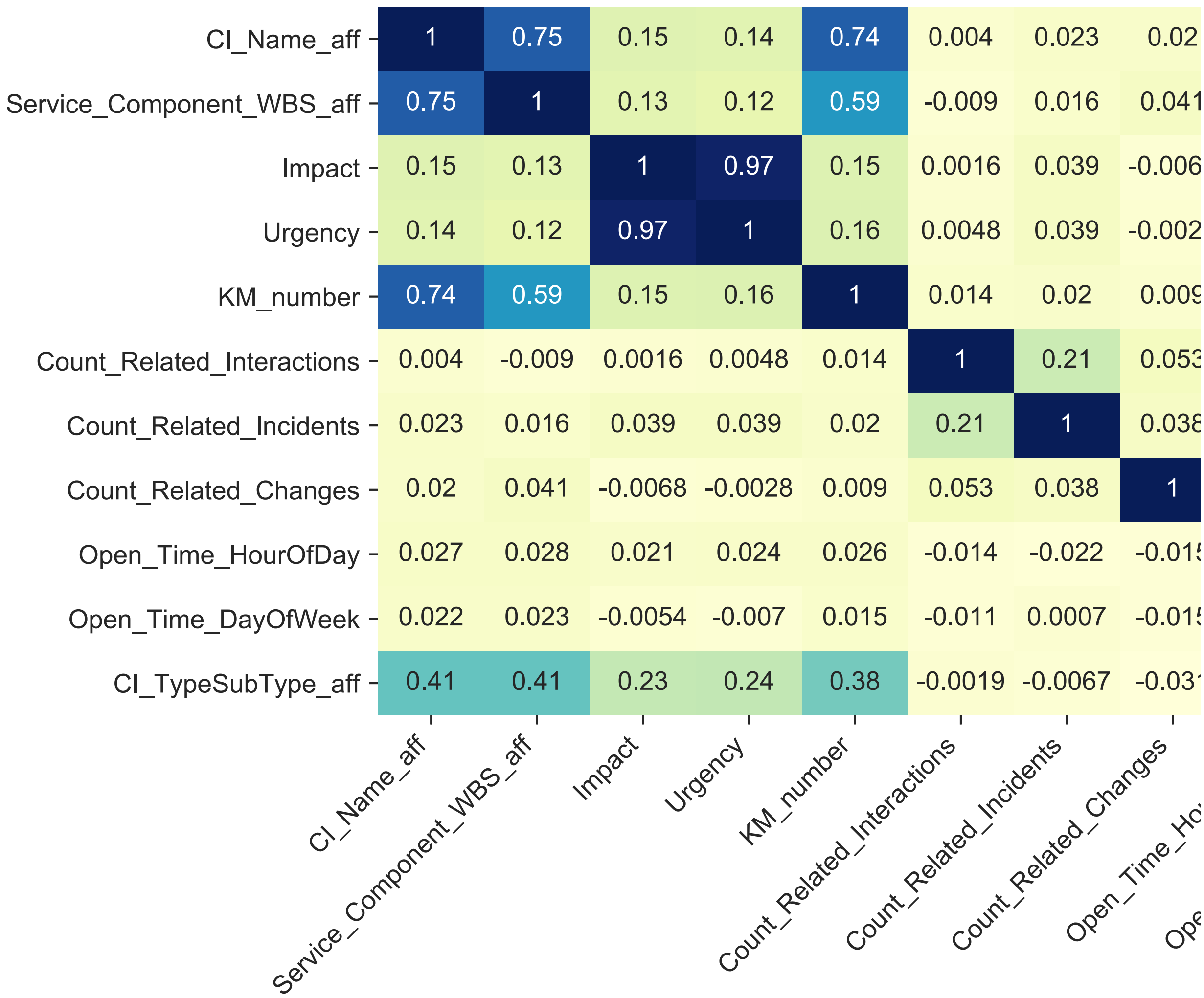
```
[8]  X = encoder.fit_transform(X,y)
```

```
[9] corrMatrix = X.corr()
```

```
[10] corrMatrix
```

	CI_Name_aff	Service_Component_WBS_aff
CI_Name_aff	1.000000	0.749470
Service_Component_WBS_aff	0.749470	1.000000
Impact	0.146698	0.130958
Urgency	0.140842	0.115506
KM_number	0.742310	0.589654
Count_Related_Interactions	0.004046	-0.008976
Count_Related_Incidents	0.022772	0.016204
Count_Related_Changes	0.019978	0.040969
Open_Time_HourOfDay	0.027004	0.027506
Open_Time_DayOfWeek	0.022354	0.023286
CI_TypeSubType_aff	0.412196	0.411194

```
[11] #fig, ax = plt.subplots()
     plt.figure(figsize=(10,5))
     chart = sns.heatmap(corrMatrix, cmap="YlGnBu", annot=True)
     chart.set_xticklabels(chart.get_xticklabels(), rotation=45,
     horizontalalignment='right')
     plt.savefig("reports/04.05.a Correlation Heatmap before.png",
     dpi=300, bbox_inches='tight')
```



Observation: `Impact` and `Urgency` represent a highly correlated pair.

- **Action:** Drop `Impact`

Observation: `CI_Name_aff`, `Service_Component_WBS_aff`, and `KM_number` represent a highly correlated trio.

- **Action:** Drop `CI_Name_aff`

[12]
```
XnoCIName = X.drop(['CI_Name_aff', 'Impact'], axis='columns')
```

[13]
```
corrMatrixNoCIName = XnoCIName.corr()
```

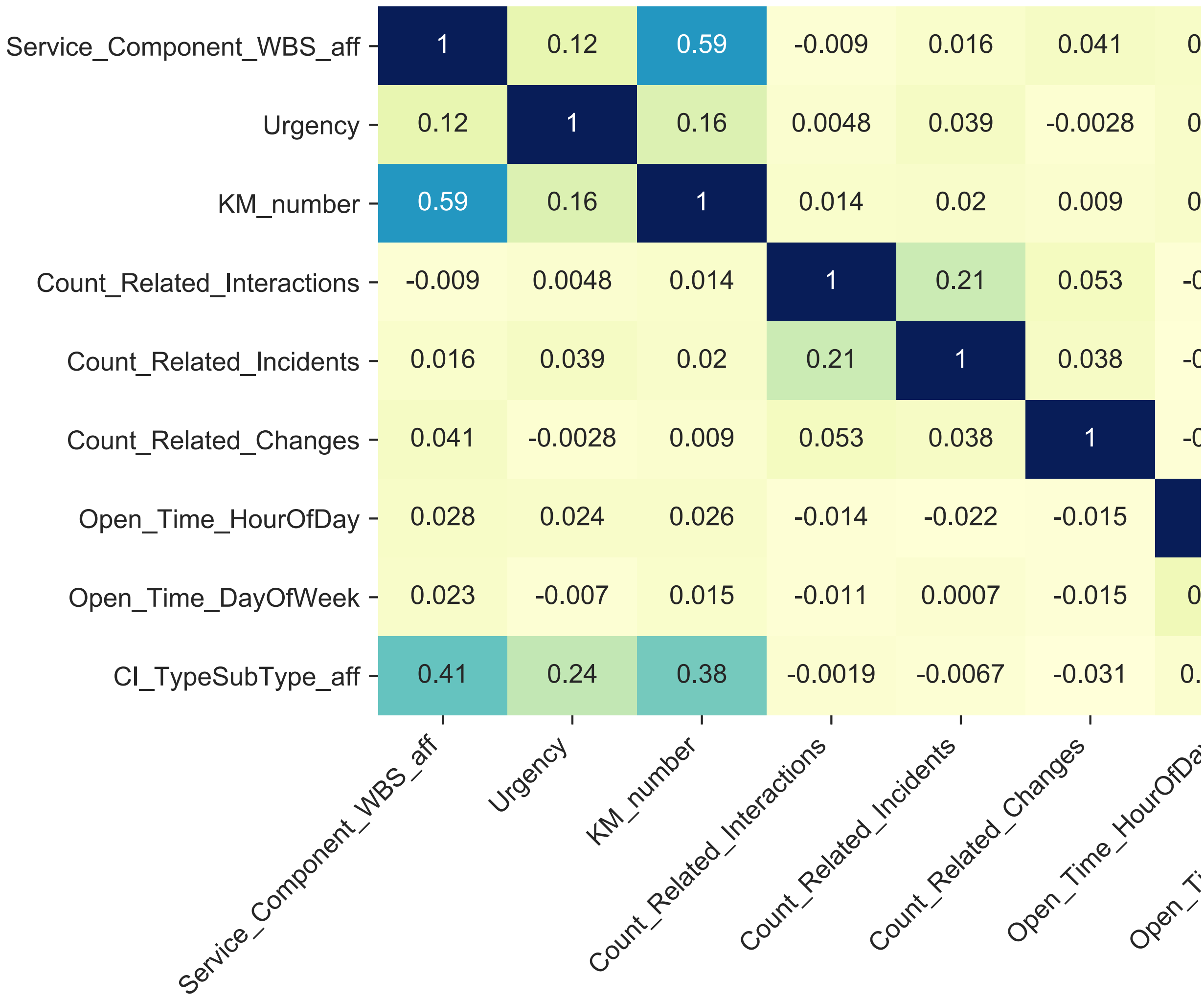
[14]
```
corrMatrixNoCIName
```

	Service_Component_WBS_aff	Urgency	KM

	Service_Component_WBS_aff	Urgency	KM
Service_Component_WBS_aff	1.000000	0.115506	0.5
Urgency	0.115506	1.000000	0.1
KM_number	0.589654	0.156142	1.0
Count_Related_Interactions	-0.008976	0.004823	0.0
Count_Related_Incidents	0.016204	0.039041	0.0
Count_Related_Changes	0.040969	-0.002827	0.0
Open_Time_HourOfDay	0.027506	0.024177	0.0
Open_Time_DayOfWeek	0.023286	-0.007022	0.0
CI_TypeSubType_aff	0.411194	0.238587	0.3

[19]

```
plt.figure(figsize=(10,5))
chart = sns.heatmap(corrMatrixNoCIName, cmap="YlGnBu",
annot=True, annot_kws={'size':10})
chart.set_xticklabels(chart.get_xticklabels(), rotation=45,
horizontalalignment='right')
plt.savefig("reports/04.05.b Correlation Heatmap after.png",
dpi=300, bbox_inches='tight')
```




```
[16]  df_out = df.drop(['CI_Name_aff', 'Impact'], axis='columns')
```

```
[17]  df_out.dtypes
```

```
Service_Component_WBS_aff       object
Urgency                         object
KM_number                       object
Count_Related_Interactions     float64
Count_Related_Incidents        float64
Count_Related_Changes          float64
SLAFail                          int64
Open_Time_HourOfDay              int64
Open_Time_DayOfWeek             object
CI_TypeSubType_aff              object
dtype: object
```

```
[18]  with open("data/05.00 Incident Data.csv",'w') as fo:
          df_out.to_csv(fo, index=False)
```

```
[ ]
```

8.5 Notebook: 05.01.c Bare Bones Analysis Using a Weight of Evidence Encoder

Output from executed notebook begins on the next page.

05.01.c Bare Bones Analysis Using a Weight of Evidence Encoder

Goal: identify the factors that most contribute to SLAFail

Tuning Adjustments: Focus on finding the most predictive set of predictor variables

Read Prepared Data -> Split Data -> Develop Pipeline -> Evaluate

Split Data using sklearn.model_selection.train_test_split

Pipeline includes:

- Preprocessing variables
  - sklearn.compose.make_column_transformer
    - Scale numeric variables: sklearn.preprocessing.StandardScaler
    - Encode categorical variables: category_encoders.WOEEncoder
- Selecting features
  - None
- Instantiate model
  - sklearn.linear_model.LogisticRegression
- Fit the model using training data
- Cross-validate the model with training data
  - sklearn.model_selection.cross_val_score
- Output performance measures

Evaluate involves running the pipeline with the testing data and capturing metrics

[1]
```
# Load libraries
import pandas as pd
import numpy as np
import pickle

# allow plots to appear in the notebook
%matplotlib inline
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split

from sklearn.compose import make_column_transformer
import category_encoders as ce
from sklearn.preprocessing import StandardScaler

from sklearn.linear_model import LogisticRegression

from sklearn.pipeline import make_pipeline
```

```python
from sklearn.model_selection import cross_val_score

from sklearn import metrics
```

Read Prepared Data

```python
[2]  df = pd.read_csv("data/05.00 Incident Data.csv")
     print("df.shape: " + str(df.shape))
     print("df.columns: " + str(df.columns))
     print("df.dtypes: \n" + str(df.dtypes))
```

```
df.shape: (35208, 10)
df.columns: Index(['Service_Component_WBS_aff', 'Urgency', 'KM_number',
       'Count_Related_Interactions', 'Count_Related_Incidents',
       'Count_Related_Changes', 'SLAFail', 'Open_Time_HourOfDay',
       'Open_Time_DayOfWeek', 'CI_TypeSubType_aff'],
      dtype='object')
df.dtypes:
Service_Component_WBS_aff      object
Urgency                        object
KM_number                      object
Count_Related_Interactions    float64
Count_Related_Incidents       float64
Count_Related_Changes         float64
SLAFail                         int64
Open_Time_HourOfDay             int64
Open_Time_DayOfWeek            object
CI_TypeSubType_aff             object
dtype: object
```

Set X and y

```python
[3]  y = df.SLAFail
     y.shape
```

```
(35208,)
```

```python
[4]  X = df.drop(['SLAFail'], axis='columns')
     X.shape
```

```
(35208, 9)
```

Set `Open_Time_HourOfDay` for recognition as a Categorical variable

```
[5]  X.Open_Time_HourOfDay = X.Open_Time_HourOfDay.astype('object')
     X.dtypes
```

```
Service_Component_WBS_aff      object
Urgency                        object
KM_number                      object
Count_Related_Interactions    float64
Count_Related_Incidents       float64
Count_Related_Changes         float64
Open_Time_HourOfDay            object
Open_Time_DayOfWeek            object
CI_TypeSubType_aff             object
dtype: object
```

Create a list of numeric variable column names

```
[6]  numericVars = X.select_dtypes(include=['float64']).columns
     numericVars
```

```
Index(['Count_Related_Interactions', 'Count_Related_Incidents',
       'Count_Related_Changes'],
      dtype='object')
```

Create a list of categorical variables

```
[7]  categoricalVars = X.select_dtypes(include=['object']).columns
     categoricalVars
```

```
Index(['Service_Component_WBS_aff', 'Urgency', 'KM_number',
       'Open_Time_HourOfDay', 'Open_Time_DayOfWeek',
'CI_TypeSubType_aff'],
      dtype='object')
```

Split Data

Create Training and Testing Data Sets

[8]
```
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=2020)
```

[9]
```
print(X_train.shape)
print(X_train.columns)
```

```
(24645, 9)
Index(['Service_Component_WBS_aff', 'Urgency', 'KM_number',
       'Count_Related_Interactions', 'Count_Related_Incidents',
       'Count_Related_Changes', 'Open_Time_HourOfDay',
'Open_Time_DayOfWeek',
       'CI_TypeSubType_aff'],
      dtype='object')
```

Develop Pipeline

[10]
```
# create dictionary to store information about the pipeline and
results for later reporting and review
PipeLineMetadata = { "Name" : "Bare Bones with WOE Encoder" }
```

Encode Variables

Numeric and categorical variables require different treatment

Set up column transformer for scaling numeric variables and encoding categorical variables

[11]
```
column_trans = make_column_transformer(
    (ce.WOEEncoder(), categoricalVars),
    (StandardScaler(), numericVars),
    remainder='passthrough')
```

Take a peek at the column transformer results

```
[12]  pd.DataFrame(column_trans.fit_transform(X_train, y_train),
      columns=X_train.columns).describe()
```

	Service_Component_WBS_aff	Urgency	KM_number	Coun
count	24645.000000	24645.000000	24645.000000	24645
mean	-0.072791	-0.004750	-0.244768	-0.003
std	0.692654	0.159450	1.268953	0.176
min	-1.862162	-0.171046	-3.152313	-1.552
25%	-0.380439	-0.069880	-1.025914	-0.137
50%	-0.380439	-0.069880	-0.020923	-0.062
75%	0.520466	-0.069880	0.729024	0.057
max	3.148473	1.539035	3.330795	1.739

```
[13]  column_trans
```

```
ColumnTransformer(n_jobs=None, remainder='passthrough',
sparse_threshold=0.3,
                  transformer_weights=None,
                  transformers=[('woeencoder',
                                 WOEEncoder(cols=None,
drop_invariant=False,
                                            handle_missing='value',
                                            handle_unknown='value',
                                            random_state=None,
randomized=False,
                                            regularization=1.0,
return_df=True,
                                            sigma=0.05, verbose=0),
                                 Index(['Service_Component_WBS_aff',
'Urgency', 'KM_number',
       'Open_Time_HourOfDay', 'Open_Time_DayOfWeek',
'CI_TypeSubType_aff'],
      dtype='object')),
                                ('standardscaler',
                                 StandardScaler(copy=True,
with_mean=True,
                                                with_std=True),
                                 Index(['Count_Related_Interactions',
'Count_Related_Incidents',
       'Count_Related_Changes'],
      dtype='object'))],
                  verbose=False)
```

Feature Selection

```
[14]  ## placeholder: none for Bare Bones
```

Specify Classifier (Logistic Regression)

```
[15]  classifier = LogisticRegression(solver="lbfgs")
```

Compose Pipeline

```
[16]  pipe = make_pipeline(column_trans,
                           classifier)
```

Fit the Model Using the Pipeline

```
[17]  pipe.fit(X_train,y_train)
```

```
Pipeline(memory=None,
         steps=[('columntransformer',
                 ColumnTransformer(n_jobs=None, remainder='passthrough',
                                   sparse_threshold=0.3,
                                   transformer_weights=None,
                                   transformers=[('woeencoder',
                                                  WOEEncoder(cols=None,

drop_invariant=False,

handle_missing='value',

handle_unknown='value',

random_state=None,
```

```
randomized=False,

regularization=1.0,

return_df=True,
                                                                    sigma=0.05,
                                                                    verbo...

Index(['Count_Related_Interactions', 'Count_Related_Incidents',
       'Count_Related_Changes'],
      dtype='object'))],
                                   verbose=False)),
                ('logisticregression',
                 LogisticRegression(C=1.0, class_weight=None,
dual=False,
                                    fit_intercept=True,
intercept_scaling=1,
                                    l1_ratio=None, max_iter=100,
                                    multi_class='auto', n_jobs=None,
                                    penalty='l2', random_state=None,
                                    solver='lbfgs', tol=0.0001,
verbose=0,
                                    warm_start=False))],
         verbose=False)
```

Capture model information

```
[18]  PipeLineMetadata['Column Transforms'] =
      list(pipe.named_steps.columntransformer.named_transformers_.keys(
      ))
      PipeLineMetadata['Classifier'] =
      pipe.named_steps.logisticregression
      PipeLineMetadata
```

```
{'Name': 'Bare Bones with WOE Encoder',
 'Column Transforms': ['woeencoder', 'standardscaler'],
 'Classifier': LogisticRegression(C=1.0, class_weight=None, dual=False,
fit_intercept=True,
                    intercept_scaling=1, l1_ratio=None, max_iter=100,
                    multi_class='auto', n_jobs=None, penalty='l2',
                    random_state=None, solver='lbfgs', tol=0.0001,
verbose=0,
                    warm_start=False)}
```

```
[19]  PipeLineMetadata['Classifier - Intercept'] =
      pipe.named_steps.logisticregression.intercept_[0]
```

```
PipeLineMetadata['Classifier - Coefficients'] =
pd.DataFrame(pipe.named_steps.logisticregression.coef_,
columns=X_train.columns).transpose()
PipeLineMetadata
```

```
{'Name': 'Bare Bones with WOE Encoder',
 'Column Transforms': ['woeencoder', 'standardscaler'],
 'Classifier': LogisticRegression(C=1.0, class_weight=None, dual=False,
fit_intercept=True,
                    intercept_scaling=1, l1_ratio=None, max_iter=100,
                    multi_class='auto', n_jobs=None, penalty='l2',
                    random_state=None, solver='lbfgs', tol=0.0001,
verbose=0,
                    warm_start=False),
 'Classifier - Intercept': -0.9376431030653062,
 'Classifier - Coefficients':                             0
 Service_Component_WBS_aff   0.103594
 Urgency                     0.187539
 KM_number                   1.121845
 Count_Related_Interactions  1.173003
 Count_Related_Incidents     1.246162
 Count_Related_Changes       0.039549
 Open_Time_HourOfDay         0.182241
 Open_Time_DayOfWeek        -0.035957
 CI_TypeSubType_aff          0.004179}
```

Cross-validate the Model with Training Data

[20]

```
PipeLineMetadata['Metrics - Cross Validation Accuracy'] =
cross_val_score(pipe, X_train, y_train, cv=5,
scoring="accuracy").mean()
PipeLineMetadata['Metrics - Cross Validation Accuracy']
```

```
0.7377561371474944
```

Evaluate with Test Data

Get predicted classification based on the model

```
[21]  y_pred_class = pipe.predict(X_test)
      y_pred_prob = pipe.predict_proba(X_test)[:,1]
```

```
[22]  PipeLineMetadata['Metrics - F1 score'] = metrics.f1_score(y_test,
      y_pred_class, average='macro')
      PipeLineMetadata['Metrics - F1 score']
```

```
0.6614631809806331
```

Look at the resulting confusion matrix

Save True Positive (TP), True Negative (TN), False Positive(FP), and False Negative (FN) values

```
[23]  confusion = metrics.confusion_matrix(y_test, y_pred_class)
      TP = confusion[1, 1]
      TN = confusion[0, 0]
      FP = confusion[0, 1]
      FN = confusion[1, 0]
      print(confusion)
      print("TN: %d \t FP: %d \nFN: %d \t TP: %d " % (TN, FP, FN, TP))
```

```
[[6597  848]
 [1826 1292]]
TN: 6597          FP: 848
FN: 1826          TP: 1292
```

Capture a few classification metrics:

- Classification Accuracy: Overall, how often is the classifier correct?
- Classification Error: Overall, how often is the classifier incorrect?
- True Positive Rate (Recall, Sensitivity): When the actual value is positive, how often is the prediction correct?
- True Negative Rate (Specificity): When the actual value is negative, how often is the prediction correct?
- False Positive Rate: When the actual value is negative, how often is the prediction incorrect?
- Precision: When a positive value is predicted, how often is the prediction correct?

```
[24]  PipeLineMetadata['Metrics - Confusion Matrix Classification
      Accuracy'] = metrics.accuracy_score(y_test,y_pred_class)
      PipeLineMetadata['Metrics - Confusion Matrix Classification
      Error'] = 1- metrics.accuracy_score(y_test,y_pred_class)
```

```
PipeLineMetadata['Metrics - Confusion Matrix True Positive Rate']
= metrics.recall_score(y_test, y_pred_class)
PipeLineMetadata['Metrics - Confusion Matrix True Negative Rate']
=  TN / float(TN + FP)
PipeLineMetadata['Metrics - Confusion Matrix False Positive
Rate'] =  FP / float(TN + FP)
PipeLineMetadata['Metrics - Confusion Matrix Precision'] =
metrics.precision_score(y_test,y_pred_class)
```

```
[25]  PipeLineMetadata
```

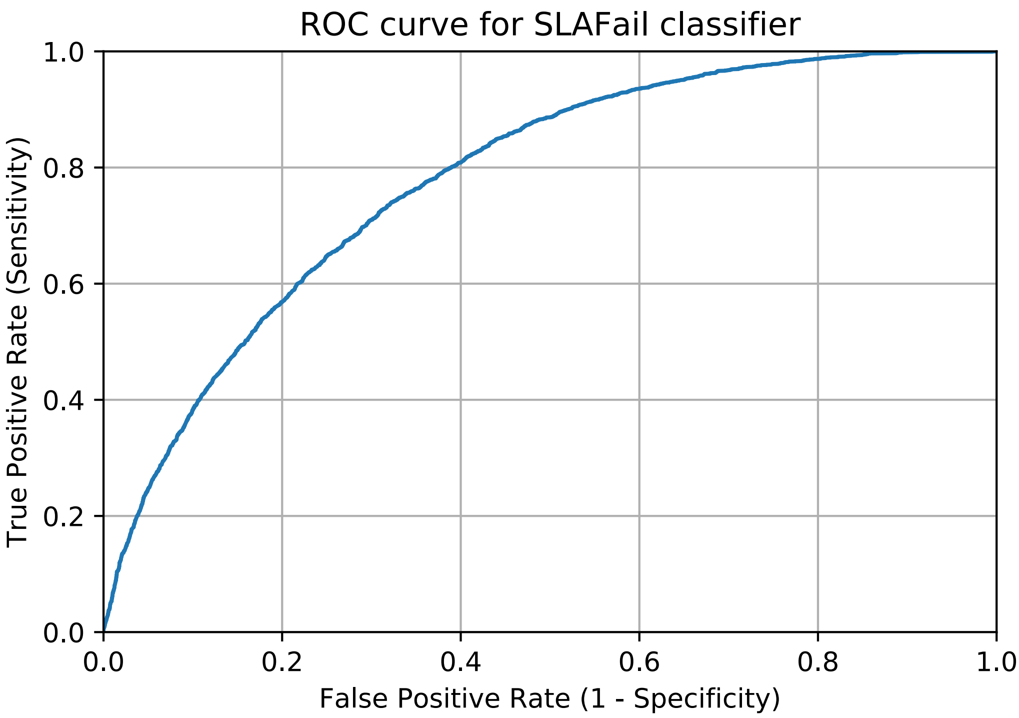
```
{'Name': 'Bare Bones with WOE Encoder',
 'Column Transforms': ['woeencoder', 'standardscaler'],
 'Classifier': LogisticRegression(C=1.0, class_weight=None, dual=False,
fit_intercept=True,
                    intercept_scaling=1, l1_ratio=None, max_iter=100,
                    multi_class='auto', n_jobs=None, penalty='l2',
                    random_state=None, solver='lbfgs', tol=0.0001,
verbose=0,
                    warm_start=False),
 'Classifier - Intercept': -0.9376431030653062,
 'Classifier - Coefficients':                                    0
 Service_Component_WBS_aff   0.103594
 Urgency                     0.187539
 KM_number                   1.121845
 Count_Related_Interactions  1.173003
 Count_Related_Incidents     1.246162
 Count_Related_Changes       0.039549
 Open_Time_HourOfDay         0.182241
 Open_Time_DayOfWeek        -0.035957
 CI_TypeSubType_aff          0.004179,
 'Metrics - Cross Validation Accuracy': 0.7377561371474944,
 'Metrics - F1 score': 0.6614631809806331,
 'Metrics - Confusion Matrix Classification Accuracy':
0.7468522200132538,
 'Metrics - Confusion Matrix Classification Error': 0.2531477799867462,
 'Metrics - Confusion Matrix True Positive Rate': 0.4143681847338037,
 'Metrics - Confusion Matrix True Negative Rate': 0.8860980523841504,
 'Metrics - Confusion Matrix False Positive Rate': 0.11390194761584957,
 'Metrics - Confusion Matrix Precision': 0.6037383177570094}
```

Add some ROC curve information and AUC result

```
[26]  fpr, tpr, thresholds = metrics.roc_curve(y_test, y_pred_prob)
```

```
[27]  PipeLineMetadata['Metrics - ROC Curve fpr array'] = fpr
      PipeLineMetadata['Metrics - ROC Curve tpr array'] = tpr
```

```
[28]  plt.plot(fpr, tpr)
      plt.xlim([0.0, 1.0])
      plt.ylim([0.0, 1.0])
      plt.title('ROC curve for SLAFail classifier')
      plt.xlabel('False Positive Rate (1 - Specificity)')
      plt.ylabel('True Positive Rate (Sensitivity)')
      plt.grid(True)
```



Capture resulting AUC

```
[29]  PipeLineMetadata['Metrics - AUC'] = metrics.roc_auc_score(y_test,
      y_pred_prob)
      print("Metrics = AUC: %f " % PipeLineMetadata['Metrics - AUC'])
```

```
Metrics = AUC: 0.780629
```

Save Details and Performance Measures for Comparison to other Models

```
[30]  PipeLineMetadata
```

```
{'Name': 'Bare Bones with WOE Encoder',
 'Column Transforms': ['woeencoder', 'standardscaler'],
 'Classifier': LogisticRegression(C=1.0, class_weight=None, dual=False,
fit_intercept=True,
                    intercept_scaling=1, l1_ratio=None, max_iter=100,
                    multi_class='auto', n_jobs=None, penalty='l2',
                    random_state=None, solver='lbfgs', tol=0.0001,
verbose=0,
                    warm_start=False),
 'Classifier - Intercept': -0.9376431030653062,
 'Classifier - Coefficients':                                    0
 Service_Component_WBS_aff   0.103594
 Urgency                     0.187539
 KM_number                   1.121845
 Count_Related_Interactions  1.173003
 Count_Related_Incidents     1.246162
 Count_Related_Changes       0.039549
 Open_Time_HourOfDay         0.182241
 Open_Time_DayOfWeek        -0.035957
 CI_TypeSubType_aff          0.004179,
 'Metrics - Cross Validation Accuracy': 0.7377561371474944,
 'Metrics - F1 score': 0.6614631809806331,
 'Metrics - Confusion Matrix Classification Accuracy':
0.7468522200132538,
 'Metrics - Confusion Matrix Classification Error': 0.2531477799867462,
 'Metrics - Confusion Matrix True Positive Rate': 0.4143681847338037,
 'Metrics - Confusion Matrix True Negative Rate': 0.8860980523841504,
 'Metrics - Confusion Matrix False Positive Rate': 0.11390194761584957,
 'Metrics - Confusion Matrix Precision': 0.6037383177570094,
 'Metrics - ROC Curve fpr array': array([0.        , 0.        , 0.
, ..., 0.99946273, 0.99973136,
        1.        ]),
 'Metrics - ROC Curve tpr array': array([0.00000000e+00, 3.20718409e-04,
2.88646568e-03, ...,
        1.00000000e+00, 1.00000000e+00, 1.00000000e+00]),
 'Metrics - AUC': 0.7806294265709925}
```

[31]
```
with open("data/05.01.c BareBones WOE.pkl",'wb') as fo:
    pickle.dump(PipeLineMetadata, fo)
```

[32]
```
# with open("data/05.01.BareBones.pkl", 'rb') as fi:
#     BareBonesMetadata = pickle.load(fi)
```

[ ]

8.6 Notebook: 05.02 Feature Selection KBest with ANOVA F-value Score Function

Output from executed notebook begins on the next page.

05.02 Feature Selection KBest with ANOVA F-value Score Function

Goal: identify the factors that most contribute to SLAFail

Tuning Adjustments: Focus on finding the most predictive set of predictor variables

Read Prepared Data -> Split Data -> Develop Pipeline -> Evaluate

Split Data using sklearn.model_selection.train_test_split

Pipeline includes:

- Preprocessing variables
  - sklearn.compose.make_column_transformer
  - Scale numeric variables: sklearn.preprocessing.StandardScaler
  - Encode categorical variables: category_encoders.MEstimateEncoder
- Selecting features
  - sklearn.feature_selection.SelectKBest
  - sklearn.feature_selection.f_classif
- Instantiate model
  - sklearn.linear_model.LogisticRegression
- Fit the model using training data
- Cross-validate the model with training data
  - sklearn.model_selection.cross_val_score
- Output performance measures

Evaluate involves running the pipeline with the testing data and capturing metrics

[1]

```
# Load libraries
import pandas as pd
import numpy as np
import pickle

# allow plots to appear in the notebook
%matplotlib inline
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split

from sklearn.compose import make_column_transformer
import category_encoders as ce
from sklearn.preprocessing import StandardScaler

from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import f_classif
```

```
from sklearn.linear_model import LogisticRegression

from sklearn.pipeline import make_pipeline

from sklearn.model_selection import cross_val_score

from sklearn import metrics
```

Read Prepared Data

```
[2]  df = pd.read_csv("data/04.a.Detail_Incident_AtOpen.csv")
     print("df.shape: " + str(df.shape))
     print("df.columns: " + str(df.columns))
     print("df.dtypes: \n" + str(df.dtypes))
```

```
df.shape: (35208, 12)
df.columns: Index(['CI_Name_aff', 'Service_Component_WBS_aff', 'Impact',
'Urgency',
       'KM_number', 'Count_Related_Interactions',
'Count_Related_Incidents',
       'Count_Related_Changes', 'SLAFail', 'Open_Time_HourOfDay',
       'Open_Time_DayOfWeek', 'CI_TypeSubType_aff'],
      dtype='object')
df.dtypes:
CI_Name_aff                    object
Service_Component_WBS_aff      object
Impact                         object
Urgency                        object
KM_number                      object
Count_Related_Interactions    float64
Count_Related_Incidents       float64
Count_Related_Changes         float64
SLAFail                         int64
Open_Time_HourOfDay             int64
Open_Time_DayOfWeek            object
CI_TypeSubType_aff             object
dtype: object
```

Set X and y

```
[3]  y = df.SLAFail
     y.shape
```

```
(35208,)
```

```
[4]  X = df.drop(['SLAFail'], axis='columns')
     X.shape
```

```
(35208, 11)
```

Set `Open_Time_HourOfDay` for recognition as a Categorical variable

```
[5]  X.Open_Time_HourOfDay = X.Open_Time_HourOfDay.astype('object')
     X.dtypes
```

```
CI_Name_aff                    object
Service_Component_WBS_aff      object
Impact                         object
Urgency                        object
KM_number                      object
Count_Related_Interactions    float64
Count_Related_Incidents       float64
Count_Related_Changes         float64
Open_Time_HourOfDay            object
Open_Time_DayOfWeek            object
CI_TypeSubType_aff             object
dtype: object
```

Create a list of numeric variable column names

```
[6]  numericVars = X.select_dtypes(include=['float64']).columns
     numericVars
```

```
Index(['Count_Related_Interactions', 'Count_Related_Incidents',
       'Count_Related_Changes'],
      dtype='object')
```

Create a list of categorical variables

```
[7]  categoricalVars = X.select_dtypes(include=['object']).columns
```

```
categoricalVars
```

```
Index(['CI_Name_aff', 'Service_Component_WBS_aff', 'Impact', 'Urgency',
       'KM_number', 'Open_Time_HourOfDay', 'Open_Time_DayOfWeek',
       'CI_TypeSubType_aff'],
      dtype='object')
```

Split Data

Create Training and Testing Data Sets

```
[8]
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=2020)
```

```
[9] print(X_train.shape)
    print(X_train.columns)
```

```
(24645, 11)
Index(['CI_Name_aff', 'Service_Component_WBS_aff', 'Impact', 'Urgency',
       'KM_number', 'Count_Related_Interactions',
'Count_Related_Incidents',
       'Count_Related_Changes', 'Open_Time_HourOfDay',
'Open_Time_DayOfWeek',
       'CI_TypeSubType_aff'],
      dtype='object')
```

Develop Pipeline

```
[10] # create dictionary to store information about the pipeline and
     results for later reporting and review
     PipeLineMetadata = { "Name" : "Feature Selection KBest F-Classif"
     }
```

Encode Variables

Numeric and categorical variables require different treatment

Set up column transformer for scaling numeric variables and encoding categorical variables

```
[11]  column_trans = make_column_transformer(
          (ce.WOEEncoder(), categoricalVars),
          (StandardScaler(), numericVars),
          remainder='passthrough')
```

```
[12]  column_trans
```

```
ColumnTransformer(n_jobs=None, remainder='passthrough',
sparse_threshold=0.3,
                  transformer_weights=None,
                  transformers=[('woeencoder',
                                 WOEEncoder(cols=None,
drop_invariant=False,
                                            handle_missing='value',
                                            handle_unknown='value',
                                            random_state=None,
randomized=False,
                                            regularization=1.0,
return_df=True,
                                            sigma=0.05, verbose=0),
                                 Index(['CI_Name_aff',
'Service_Component_WBS_aff', 'Impact', 'Urgency',
       'KM_number', 'Open_Time_HourOfDay', 'Open_Time_DayOfWeek',
       'CI_TypeSubType_aff'],
      dtype='object')),
                                ('standardscaler',
                                 StandardScaler(copy=True,
with_mean=True,
                                                with_std=True),
                                 Index(['Count_Related_Interactions',
'Count_Related_Incidents',
       'Count_Related_Changes'],
      dtype='object'))],
                  verbose=False)
```

Feature Selection

```
[13]  selector = SelectKBest(score_func=f_classif, k=10)
```

Specify Classifier (Logistic Regression)

```
[14]  classifier = LogisticRegression(solver="lbfgs")
```

Compose Pipeline

```
[15]  pipe = make_pipeline(column_trans,
                           selector,
                           classifier)
```

Fit the Model Using the Pipeline

```
[16]  pipe.fit(X_train,y_train)
```

```
Pipeline(memory=None,
         steps=[('columntransformer',
                 ColumnTransformer(n_jobs=None, remainder='passthrough',
                                   sparse_threshold=0.3,
                                   transformer_weights=None,
                                   transformers=[('woeencoder',
                                                  WOEEncoder(cols=None,

drop_invariant=False,

handle_missing='value',

handle_unknown='value',

random_state=None,

randomized=False,

regularization=1.0,

return_df=True,
                                                             sigma=0.05,
                                                             verbo...
                ('selectkbest',
                 SelectKBest(k=10,
                             score_func=<function f_classif at
0x1a204b5830>)),
                ('logisticregression',
                 LogisticRegression(C=1.0, class_weight=None,
dual=False,
                                    fit_intercept=True,
```

```
intercept_scaling=1,
                                                    l1_ratio=None, max_iter=100,
                                                    multi_class='auto', n_jobs=None,
                                                    penalty='l2', random_state=None,
                                                    solver='lbfgs', tol=0.0001,
verbose=0,
                                                    warm_start=False))],
             verbose=False)
```

Capture model information

```
[17]  PipeLineMetadata['Column Transforms'] =
      list(pipe.named_steps.columntransformer.named_transformers_.keys(
      ))
      PipeLineMetadata['Selector'] =
      pipe.named_steps.selectkbest.get_params
      PipeLineMetadata['Classifier'] =
      pipe.named_steps.logisticregression
      PipeLineMetadata
```

```
{'Name': 'Feature Selection KBest F-Classif',
 'Column Transforms': ['woeencoder', 'standardscaler'],
 'Selector': <bound method BaseEstimator.get_params of SelectKBest(k=10,
score_func=<function f_classif at 0x1a204b5830>)>,
 'Classifier': LogisticRegression(C=1.0, class_weight=None, dual=False,
fit_intercept=True,
                    intercept_scaling=1, l1_ratio=None, max_iter=100,
                    multi_class='auto', n_jobs=None, penalty='l2',
                    random_state=None, solver='lbfgs', tol=0.0001,
verbose=0,
                    warm_start=False)}
```

Identify the features retained

```
[18]  pipe.named_steps.selectkbest.scores_
```

```
array([4.04035978e+03, 2.62605572e+03, 1.40220303e+02, 1.41583607e+02,
       6.10443839e+03, 1.77755544e+02, 6.79762696e+02, 9.02108720e+02,
       6.20757993e+01, 5.60722172e+00, 3.51935177e-01])
```

```
[19]  pipe.named_steps.selectkbest.pvalues_
```

```
array([0.00000000e+000, 0.00000000e+000, 2.91451608e-032, 1.47282598e-
032,
       0.00000000e+000, 2.06821476e-040, 7.61382496e-148, 1.10035990e-
194,
       3.44070391e-015, 1.78942746e-002, 5.53025157e-001])
```

```
[20]  PipeLineMetadata['Selector - Scores'] =
      pd.DataFrame([pipe.named_steps.selectkbest.scores_,
      pipe.named_steps.selectkbest.pvalues_], columns=X_train.columns,
      index=['scores','p-value']).transpose().sort_values(by=['p-
      value'])
      PipeLineMetadata
```

```
{'Name': 'Feature Selection KBest F-Classif',
 'Column Transforms': ['woeencoder', 'standardscaler'],
 'Selector': <bound method BaseEstimator.get_params of SelectKBest(k=10,
score_func=<function f_classif at 0x1a204b5830>)>,
 'Classifier': LogisticRegression(C=1.0, class_weight=None, dual=False,
fit_intercept=True,
                    intercept_scaling=1, l1_ratio=None, max_iter=100,
                    multi_class='auto', n_jobs=None, penalty='l2',
                    random_state=None, solver='lbfgs', tol=0.0001,
verbose=0,
                    warm_start=False),
 'Selector - Scores':                              scores        p-
value
 CI_Name_aff                 4040.359780   0.000000e+00
 Service_Component_WBS_aff   2626.055718   0.000000e+00
 KM_number                   6104.438394   0.000000e+00
 Count_Related_Changes        902.108720  1.100360e-194
 Count_Related_Incidents      679.762696  7.613825e-148
 Count_Related_Interactions   177.755544   2.068215e-40
 Urgency                      141.583607   1.472826e-32
 Impact                       140.220303   2.914516e-32
 Open_Time_HourOfDay           62.075799   3.440704e-15
 Open_Time_DayOfWeek            5.607222   1.789427e-02
 CI_TypeSubType_aff             0.351935   5.530252e-01}
```

```
[21]  # returns a mask of features retained
      pipe.named_steps.selectkbest.get_support().tolist()
```

```
[True, True, True, True, True, True, True, True, True, True, False]
```

```
[22]  # apply mask to X_train column names
      selectedFeatures = np.array(X_train.columns.tolist())
      [pipe.named_steps.selectkbest.get_support().tolist()]
      selectedFeatures
```

```
array(['CI_Name_aff', 'Service_Component_WBS_aff', 'Impact', 'Urgency',
       'KM_number', 'Count_Related_Interactions',
       'Count_Related_Incidents', 'Count_Related_Changes',
       'Open_Time_HourOfDay', 'Open_Time_DayOfWeek'], dtype='<U26')
```

[23]

```
PipeLineMetadata['Classifier - Intercept'] =
pipe.named_steps.logisticregression.intercept_[0]
PipeLineMetadata['Classifier - Coefficients'] =
pd.DataFrame(pipe.named_steps.logisticregression.coef_,
columns=selectedFeatures).transpose()
PipeLineMetadata
```

```
{'Name': 'Feature Selection KBest F-Classif',
 'Column Transforms': ['woeencoder', 'standardscaler'],
 'Selector': <bound method BaseEstimator.get_params of SelectKBest(k=10,
score_func=<function f_classif at 0x1a204b5830>)>,
 'Classifier': LogisticRegression(C=1.0, class_weight=None, dual=False,
fit_intercept=True,
                    intercept_scaling=1, l1_ratio=None, max_iter=100,
                    multi_class='auto', n_jobs=None, penalty='l2',
                    random_state=None, solver='lbfgs', tol=0.0001,
verbose=0,
                    warm_start=False),
 'Selector - Scores':                                  scores        p-
value
 CI_Name_aff                  4040.359780   0.000000e+00
 Service_Component_WBS_aff    2626.055718   0.000000e+00
 KM_number                    6104.438394   0.000000e+00
 Count_Related_Changes         902.108720  1.100360e-194
 Count_Related_Incidents       679.762696  7.613825e-148
 Count_Related_Interactions    177.755544   2.068215e-40
 Urgency                       141.583607   1.472826e-32
 Impact                        140.220303   2.914516e-32
 Open_Time_HourOfDay            62.075799   3.440704e-15
 Open_Time_DayOfWeek             5.607222   1.789427e-02
 CI_TypeSubType_aff              0.351935   5.530252e-01,
 'Classifier - Intercept': -0.9295114210180985,
 'Classifier - Coefficients':                                     0
 CI_Name_aff                  0.149591
 Service_Component_WBS_aff    0.020888
 Impact                       0.169542
 Urgency                      0.039663
 KM_number                    1.068318
 Count_Related_Interactions   1.176742
 Count_Related_Incidents      1.246995
 Count_Related_Changes        0.027675
 Open_Time_HourOfDay          0.183534
 Open_Time_DayOfWeek         -0.037030}
```

Cross-validate the Model with Training Data

```
[24]  PipeLineMetadata['Metrics - Cross Validation Accuracy'] =
      cross_val_score(pipe, X_train, y_train, cv=5,
      scoring="accuracy").mean()
      PipeLineMetadata
```

```
{'Name': 'Feature Selection KBest F-Classif',
 'Column Transforms': ['woeencoder', 'standardscaler'],
 'Selector': <bound method BaseEstimator.get_params of SelectKBest(k=10,
score_func=<function f_classif at 0x1a204b5830>)>,
 'Classifier': LogisticRegression(C=1.0, class_weight=None, dual=False,
fit_intercept=True,
                    intercept_scaling=1, l1_ratio=None, max_iter=100,
                    multi_class='auto', n_jobs=None, penalty='l2',
                    random_state=None, solver='lbfgs', tol=0.0001,
verbose=0,
                    warm_start=False),
 'Selector - Scores':                              scores        p-
value
 CI_Name_aff                  4040.359780   0.000000e+00
 Service_Component_WBS_aff    2626.055718   0.000000e+00
 KM_number                    6104.438394   0.000000e+00
 Count_Related_Changes         902.108720  1.100360e-194
 Count_Related_Incidents       679.762696  7.613825e-148
 Count_Related_Interactions    177.755544   2.068215e-40
 Urgency                       141.583607   1.472826e-32
 Impact                        140.220303   2.914516e-32
 Open_Time_HourOfDay            62.075799   3.440704e-15
 Open_Time_DayOfWeek             5.607222   1.789427e-02
 CI_TypeSubType_aff              0.351935   5.530252e-01,
 'Classifier - Intercept': -0.9295114210180985,
 'Classifier - Coefficients':                                    0
 CI_Name_aff                  0.149591
 Service_Component_WBS_aff    0.020888
 Impact                       0.169542
 Urgency                      0.039663
 KM_number                    1.068318
 Count_Related_Interactions   1.176742
 Count_Related_Incidents      1.246995
 Count_Related_Changes        0.027675
 Open_Time_HourOfDay          0.183534
 Open_Time_DayOfWeek         -0.037030,
 'Metrics - Cross Validation Accuracy': 0.737025765875431}
```

Evaluate with Test Data

Get predicted classification and predicted probabilities based on the model

```
[25] y_pred_class = pipe.predict(X_test)
     y_pred_prob = pipe.predict_proba(X_test)[:,1]
```

```
[26] PipeLineMetadata['Metrics - F1 score'] = metrics.f1_score(y_test,
     y_pred_class, average='macro')
     PipeLineMetadata['Metrics - F1 score']
```

```
0.6579221552692892
```

Look at the resulting confusion matrix

Save True Positive (TP), True Negative (TN), False Positive(FP), and False Negative (FN) values

```
[27] confusion = metrics.confusion_matrix(y_test, y_pred_class)
     TP = confusion[1, 1]
     TN = confusion[0, 0]
     FP = confusion[0, 1]
     FN = confusion[1, 0]
     print(confusion)
     print("TN: %d \t FP: %d \nFN: %d \t TP: %d " % (TN, FP, FN, TP))
```

```
[[6581  864]
 [1839 1279]]
TN: 6581         FP: 864
FN: 1839         TP: 1279
```

Capture a few classification metrics:

- Classification Accuracy: Overall, how often is the classifier correct?
- Classification Error: Overall, how often is the classifier incorrect?
- True Positive Rate (Recall, Sensitivity): When the actual value is positive, how often is the prediction correct?
- True Negative Rate (Specificity): When the actual value is negative, how often is the prediction correct?
- False Positive Rate: When the actual value is negative, how often is the prediction incorrect?
- Precision: When a positive value is predicted, how often is the prediction correct?

```
[28] PipeLineMetadata['Metrics - Confusion Matrix Classification
     Accuracy'] = metrics.accuracy_score(y_test,y_pred_class)
```

```
PipeLineMetadata['Metrics - Confusion Matrix Classification
Error'] = 1- metrics.accuracy_score(y_test,y_pred_class)
PipeLineMetadata['Metrics - Confusion Matrix True Positive Rate']
= metrics.recall_score(y_test, y_pred_class)
PipeLineMetadata['Metrics - Confusion Matrix True Negative Rate']
=  TN / float(TN + FP)
PipeLineMetadata['Metrics - Confusion Matrix False Positive
Rate'] =  FP / float(TN + FP)
PipeLineMetadata['Metrics - Confusion Matrix Precision'] =
metrics.precision_score(y_test,y_pred_class)
```

[29]

```
PipeLineMetadata
```

```
{'Name': 'Feature Selection KBest F-Classif',
 'Column Transforms': ['woeencoder', 'standardscaler'],
 'Selector': <bound method BaseEstimator.get_params of SelectKBest(k=10,
score_func=<function f_classif at 0x1a204b5830>)>,
 'Classifier': LogisticRegression(C=1.0, class_weight=None, dual=False,
fit_intercept=True,
                    intercept_scaling=1, l1_ratio=None, max_iter=100,
                    multi_class='auto', n_jobs=None, penalty='l2',
                    random_state=None, solver='lbfgs', tol=0.0001,
verbose=0,
                    warm_start=False),
 'Selector - Scores':                                   scores        p-
value
 CI_Name_aff                  4040.359780   0.000000e+00
 Service_Component_WBS_aff    2626.055718   0.000000e+00
 KM_number                    6104.438394   0.000000e+00
 Count_Related_Changes         902.108720  1.100360e-194
 Count_Related_Incidents       679.762696  7.613825e-148
 Count_Related_Interactions    177.755544   2.068215e-40
 Urgency                       141.583607   1.472826e-32
 Impact                        140.220303   2.914516e-32
 Open_Time_HourOfDay            62.075799   3.440704e-15
 Open_Time_DayOfWeek             5.607222   1.789427e-02
 CI_TypeSubType_aff              0.351935   5.530252e-01,
 'Classifier - Intercept': -0.9295114210180985,
 'Classifier - Coefficients':                                   0
 CI_Name_aff                  0.149591
 Service_Component_WBS_aff    0.020888
 Impact                       0.169542
 Urgency                      0.039663
 KM_number                    1.068318
 Count_Related_Interactions   1.176742
 Count_Related_Incidents      1.246995
 Count_Related_Changes        0.027675
 Open_Time_HourOfDay          0.183534
 Open_Time_DayOfWeek         -0.037030,
 'Metrics - Cross Validation Accuracy': 0.737025765875431,
 'Metrics - F1 score': 0.6579221552692892,
 'Metrics - Confusion Matrix Classification Accuracy':
0.7441067878443623,
```

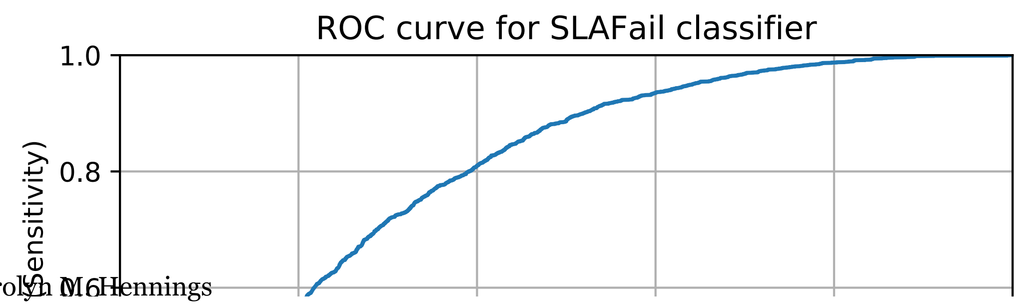
```
'Metrics - Confusion Matrix Classification Error': 0.25589321215563765,
'Metrics - Confusion Matrix True Positive Rate': 0.41019884541372675,
'Metrics - Confusion Matrix True Negative Rate': 0.883948959032908,
'Metrics - Confusion Matrix False Positive Rate': 0.11605104096709201,
'Metrics - Confusion Matrix Precision': 0.5968268782081194}
```

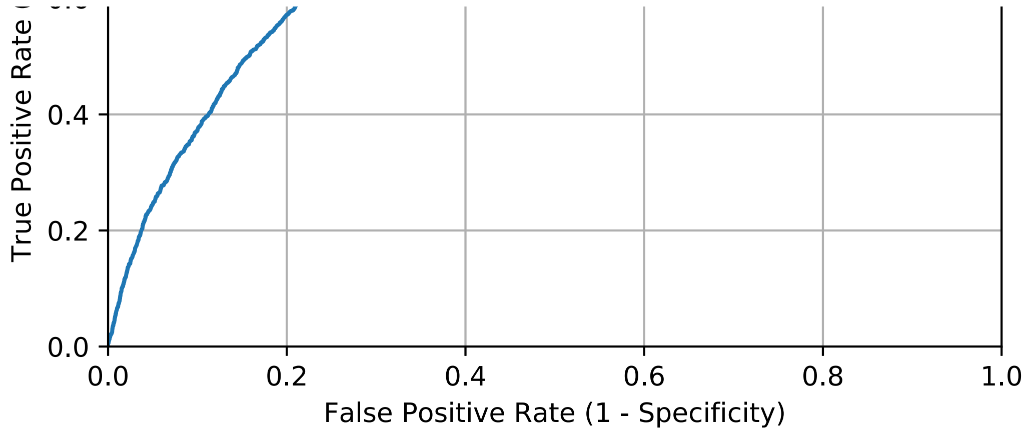
Add some ROC curve information and AUC result

[30]
```
fpr, tpr, thresholds = metrics.roc_curve(y_test, y_pred_prob)
```

[31]
```
PipeLineMetadata['Metrics - ROC Curve fpr array'] = fpr
PipeLineMetadata['Metrics - ROC Curve tpr array'] = tpr
```

[32]
```
plt.plot(fpr, tpr)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.0])
plt.title('ROC curve for SLAFail classifier')
plt.xlabel('False Positive Rate (1 - Specificity)')
plt.ylabel('True Positive Rate (Sensitivity)')
plt.grid(True)
```





Capture resulting AUC

```
[33]  PipeLineMetadata['Metrics - AUC'] = metrics.roc_auc_score(y_test,
      y_pred_prob)
      print("Metrics = AUC: %f " % PipeLineMetadata['Metrics - AUC'])
```

```
Metrics = AUC: 0.780698
```

Save Details and Performance Measures for Comparison to other Models

```
[34]  PipeLineMetadata
```

```
{'Name': 'Feature Selection KBest F-Classif',
 'Column Transforms': ['woeencoder', 'standardscaler'],
 'Selector': <bound method BaseEstimator.get_params of SelectKBest(k=10,
score_func=<function f_classif at 0x1a204b5830>)>,
 'Classifier': LogisticRegression(C=1.0, class_weight=None, dual=False,
fit_intercept=True,
                    intercept_scaling=1, l1_ratio=None, max_iter=100,
                    multi_class='auto', n_jobs=None, penalty='l2',
                    random_state=None, solver='lbfgs', tol=0.0001,
verbose=0,
                    warm_start=False),
 'Selector - Scores':                              scores        p-
value
 CI_Name_aff                   4040.359780   0.000000e+00
 Service_Component_WBS_aff     2626.055718   0.000000e+00
 KM_number                     6104.438394   0.000000e+00
 Count_Related_Changes          902.108720  1.100360e-194
 Count_Related_Incidents        679.762696  7.613825e-148
 Count_Related_Interactions     177.755544   2.068215e-40
```

```
Urgency                      141.583607   1.472826e-32
Impact                       140.220303   2.914516e-32
Open_Time_HourOfDay           62.075799   3.440704e-15
Open_Time_DayOfWeek            5.607222   1.789427e-02
CI_TypeSubType_aff             0.351935   5.530252e-01,
'Classifier - Intercept': -0.9295114210180985,
'Classifier - Coefficients':                                    0
CI_Name_aff                 0.149591
Service_Component_WBS_aff   0.020888
Impact                      0.169542
Urgency                     0.039663
KM_number                   1.068318
Count_Related_Interactions  1.176742
Count_Related_Incidents     1.246995
Count_Related_Changes       0.027675
Open_Time_HourOfDay         0.183534
Open_Time_DayOfWeek        -0.037030,
'Metrics - Cross Validation Accuracy': 0.737025765875431,
'Metrics - F1 score': 0.6579221552692892,
'Metrics - Confusion Matrix Classification Accuracy':
0.7441067878443623,
'Metrics - Confusion Matrix Classification Error': 0.25589321215563765,
'Metrics - Confusion Matrix True Positive Rate': 0.41019884541372675,
'Metrics - Confusion Matrix True Negative Rate': 0.883948959032908,
'Metrics - Confusion Matrix False Positive Rate': 0.11605104096709201,
'Metrics - Confusion Matrix Precision': 0.5968268782081194,
'Metrics - ROC Curve fpr array': array([0.        , 0.        , 0.
, ..., 0.99919409, 0.99946273,
       1.        ]),
'Metrics - ROC Curve tpr array': array([0.00000000e+00, 3.20718409e-04,
1.28287364e-03, ...,
       1.00000000e+00, 1.00000000e+00, 1.00000000e+00]),
'Metrics - AUC': 0.7806984811861712}
```

```
[35]  with open("data/05.02.a Feature Select KBest f_classif.pkl",'wb')
      as fo:
          pickle.dump(PipeLineMetadata, fo)
```

```
[36]  # with open("FILENAME", 'rb') as fi:
      #     BareBonesMetadata = pickle.load(fi)
```

```
[ ]
```

8.7 Notebook: 06.01.b Optimize the Logistic Regression Model

Output from executed notebook begins on the next page.

06.01.b Optimize the Logistic Regression Model

Goal: identify the factors that most contribute to SLAFail

Tuning Adjustments: Focus on finding the most predictive set of predictor variables

Read Prepared Data -> Split Data -> Develop Pipeline -> Evaluate

Split Data using sklearn.model_selection.train_test_split

Pipeline includes:

- Preprocessing variables
  - sklearn.compose.make_column_transformer
  - Scale numeric variables: sklearn.preprocessing.StandardScaler
  - Encode categorical variables: category_encoders.MEstimateEncoder
- Selecting features
  - sklearn.feature_selection.SelectFpr
  - sklearn.feature_selection.f_classif
- Instantiate model
  - sklearn.linear_model.LogisticRegression
- Fit the model using training data
- Cross-validate the model with training data
  - sklearn.model_selection.cross_val_score
- Output performance measures

Evaluate involves running the pipeline with the testing data and capturing metrics

https://github.com/justmarkham/scikit-learn-videos/blob/master/08_grid_search.ipynb

[5]

```
# Load libraries
import pandas as pd
import numpy as np
import pickle

# allow plots to appear in the notebook
%matplotlib inline
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split

from sklearn.compose import make_column_transformer
import category_encoders as ce
from sklearn.preprocessing import StandardScaler
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV

from sklearn.feature_selection import SelectFpr

from sklearn import metrics
```

[6]
```
# create dictionary to store information about the pipeline and
results for later reporting and review
PipeLineMetadata = { "Name" : "Optimize Round 2 Selector
SelectFpr with f_classif and p-value <= 0.05" }
```

[7]
```
outFileName = "data/06.00.b Optimize 2 Select FPR.pkl"
```

[8]
```
df = pd.read_csv("data/05.00 Incident Data.csv")
print("df.shape: " + str(df.shape))
print("df.columns: " + str(df.columns))
print("df.dtypes: \n" + str(df.dtypes))
```

```
df.shape: (35208, 10)
df.columns: Index(['Service_Component_WBS_aff', 'Urgency', 'KM_number',
       'Count_Related_Interactions', 'Count_Related_Incidents',
       'Count_Related_Changes', 'SLAFail', 'Open_Time_HourOfDay',
       'Open_Time_DayOfWeek', 'CI_TypeSubType_aff'],
      dtype='object')
df.dtypes:
Service_Component_WBS_aff      object
Urgency                        object
KM_number                      object
Count_Related_Interactions    float64
Count_Related_Incidents       float64
Count_Related_Changes         float64
SLAFail                         int64
Open_Time_HourOfDay             int64
Open_Time_DayOfWeek            object
CI_TypeSubType_aff             object
dtype: object
```

Read Prepared Data

Set X and y

```
[9]   y = df.SLAFail
      y.shape
```

```
(35208,)
```

```
[10]  X = df.drop(['SLAFail'], axis='columns')
      X.shape
```

```
(35208, 9)
```

Set `Open_Time_HourOfDay` for recognition as a Categorical variable

```
[11]  X.Open_Time_HourOfDay = X.Open_Time_HourOfDay.astype('object')
      X.dtypes
```

```
Service_Component_WBS_aff       object
Urgency                         object
KM_number                       object
Count_Related_Interactions     float64
Count_Related_Incidents        float64
Count_Related_Changes          float64
Open_Time_HourOfDay             object
Open_Time_DayOfWeek             object
CI_TypeSubType_aff              object
dtype: object
```

Create a list of numeric variable column names

```
[12]  numericVars = X.select_dtypes(include=['float64']).columns
      numericVars
```

```
Index(['Count_Related_Interactions', 'Count_Related_Incidents',
       'Count_Related_Changes'],
      dtype='object')
```

Create a list of categorical variables

```
[13]  categoricalVars = X.select_dtypes(include=['object']).columns
      categoricalVars
```

```
Index(['Service_Component_WBS_aff', 'Urgency', 'KM_number',
       'Open_Time_HourOfDay', 'Open_Time_DayOfWeek',
'CI_TypeSubType_aff'],
      dtype='object')
```

Split Data

Create Training and Testing Data Sets

```
[14]
      X_train, X_test, y_train, y_test = train_test_split(X, y,
      test_size=0.3, random_state=2020)
```

```
[15]  print(X_train.shape)
      print(X_train.columns)
```

```
(24645, 9)
Index(['Service_Component_WBS_aff', 'Urgency', 'KM_number',
       'Count_Related_Interactions', 'Count_Related_Incidents',
       'Count_Related_Changes', 'Open_Time_HourOfDay',
'Open_Time_DayOfWeek',
       'CI_TypeSubType_aff'],
      dtype='object')
```

Calculate Null Accuracy

Null accuracy: accuracy that could be achieved by always predicting the most frequent class

This means that a dumb model that always predicts 0 would be right 68% of the time

This shows how classification accuracy is not that good as it's close to a dumb model It's a good way to know the minimum we should achieve with our models

```
[16]  # examine the class distribution of the testing set (using a
      Pandas Series method)
      y_test.value_counts()
```

```
0    7445
1    3118
Name: SLAFail, dtype: int64
```

```
[17]  # calculate the percentage of ones
      print("Percentage of Ones: %f " % y_test.mean())
      print("Percentage of Zeros: %f " % (1 - y_test.mean()))
      print("Percentage of Zeros: %f " % ( 1 - y_test.mean() ) )
      null_accuracy = max(y_test.mean(), 1 - y_test.mean())
      PipeLineMetadata = { "Null Accuracy" : null_accuracy }
      print("Null Accuracy: %f " % null_accuracy )
```

```
Percentage of Ones: 0.295181
Percentage of Zeros: 0.704819
Percentage of Zeros: 0.704819
Null Accuracy: 0.704819
```

This means that a 'dumb' model that always predicts 0 would be right 70% of the time.

The developed model must exceed a 70% accuracy rate to be considered better than the 'dumb' model.

Develop Pipeline

Encode Variables

Numeric and categorical variables require different treatment

Set up column transformer for scaling numeric variables and encoding categorical variables

```
[18]  column_trans = make_column_transformer(
          (ce.WOEEncoder(), categoricalVars),
          (StandardScaler(), numericVars),
          remainder='passthrough')
```

Feature Selection

```
[19]  # default score function is f_classif and p-value 0.05
      selector = SelectFpr()
```

Specify Classifier (Logistic Regression)

```
[20]  classifier = LogisticRegression(solver="lbfgs")
```

Compose Pipeline

```
[21]  pipe = Pipeline(steps=[('column_trans', column_trans),
                            ('selector',  selector),
                            ('classifier', classifier)
                            ])
```

Train the model

```
[22]  pipe.fit(X_train, y_train)
```

```
Pipeline(memory=None,
         steps=[('column_trans',
                 ColumnTransformer(n_jobs=None, remainder='passthrough',
                                   sparse_threshold=0.3,
                                   transformer_weights=None,
                                   transformers=[('woeencoder',
                                                  WOEEncoder(cols=None,

drop_invariant=False,

handle_missing='value',

handle_unknown='value',
```

```
random_state=None,

randomized=False,

regularization=1.0,

return_df=True,
                                                                                    sigma=0.05,

verbose=0)...
                                       verbose=False)),
                  ('selector',
                   SelectFpr(alpha=0.05,
                              score_func=<function f_classif at
0x1a2489ae60>)),
                  ('classifier',
                   LogisticRegression(C=1.0, class_weight=None,
dual=False,
                                        fit_intercept=True,
intercept_scaling=1,
                                        l1_ratio=None, max_iter=100,
                                        multi_class='auto', n_jobs=None,
                                        penalty='l2', random_state=None,
                                        solver='lbfgs', tol=0.0001,
verbose=0,
                                        warm_start=False))],
          verbose=False)
```

[23]
```
##### Save model information
PipeLineMetadata['Column Transforms'] =
list(pipe.named_steps.column_trans.named_transformers_.keys())
PipeLineMetadata['Selector'] = pipe.named_steps.selector
PipeLineMetadata['Classifier'] = pipe.named_steps.classifier
PipeLineMetadata
```

```
{'Null Accuracy': 0.7048187068067784,
 'Column Transforms': ['woeencoder', 'standardscaler'],
 'Selector': SelectFpr(alpha=0.05, score_func=<function f_classif at
0x1a2489ae60>),
 'Classifier': LogisticRegression(C=1.0, class_weight=None, dual=False,
fit_intercept=True,
                      intercept_scaling=1, l1_ratio=None, max_iter=100,
                      multi_class='auto', n_jobs=None, penalty='l2',
                      random_state=None, solver='lbfgs', tol=0.0001,
verbose=0,
                      warm_start=False)}
```

[24]
```
# returns a mask of features retained
pipe.named_steps.selector.get_support().tolist()
```

```
[True, True, True, True, True, True, True, True, False]
```

```
[25]  # apply mask to X_train column names
      selectedFeatures = np.array(X_train.columns.tolist())
      [pipe.named_steps.selector.get_support().tolist()]
      selectedFeatures
```

```
array(['Service_Component_WBS_aff', 'Urgency', 'KM_number',
       'Count_Related_Interactions', 'Count_Related_Incidents',
       'Count_Related_Changes', 'Open_Time_HourOfDay',
       'Open_Time_DayOfWeek'], dtype='<U26')
```

```
[26]  PipeLineMetadata['Classifier - Intercept'] =
      pipe.named_steps.classifier.intercept_[0]
      PipeLineMetadata['Classifier - Coefficients'] =
      pd.DataFrame(pipe.named_steps.classifier.coef_,
      columns=selectedFeatures).transpose()
      PipeLineMetadata
```

```
{'Null Accuracy': 0.7048187068067784,
 'Column Transforms': ['woeencoder', 'standardscaler'],
 'Selector': SelectFpr(alpha=0.05, score_func=<function f_classif at
0x1a2489ae60>),
 'Classifier': LogisticRegression(C=1.0, class_weight=None, dual=False,
fit_intercept=True,
                    intercept_scaling=1, l1_ratio=None, max_iter=100,
                    multi_class='auto', n_jobs=None, penalty='l2',
                    random_state=None, solver='lbfgs', tol=0.0001,
verbose=0,
                    warm_start=False),
 'Classifier - Intercept': -0.9376094581875354,
 'Classifier - Coefficients':                                     0
 Service_Component_WBS_aff    0.104001
 Urgency                      0.188064
 KM_number                    1.121735
 Count_Related_Interactions   1.172629
 Count_Related_Incidents      1.245934
 Count_Related_Changes        0.038820
 Open_Time_HourOfDay          0.182436
 Open_Time_DayOfWeek         -0.035750}
```

Test the Model

Get predicted classification and predicted probabilities based on the model

```
[27]  y_pred_class = pipe.predict(X_test)
```

```
[28]  y_pred_prob = pipe.predict_proba(X_test)[:,1]
```

```
[29]  print(metrics.classification_report(y_test, y_pred_class))
```

```
precision    recall  f1-score   support

           0       0.78      0.89      0.83      7445
           1       0.60      0.41      0.49      3118

    accuracy                           0.75     10563
   macro avg       0.69      0.65      0.66     10563
weighted avg       0.73      0.75      0.73     10563
```

Look at the resulting confusion matrix

Save True Positive (TP), True Negative (TN), False Positive(FP), and False Negative (FN) values

```
[30]  confusion = metrics.confusion_matrix(y_test, y_pred_class)
      TP = confusion[1, 1]
      TN = confusion[0, 0]
      FP = confusion[0, 1]
      FN = confusion[1, 0]
      print(confusion)
      print("TN: %d \t FP: %d \nFN: %d \t TP: %d " % (TN, FP, FN, TP))
```

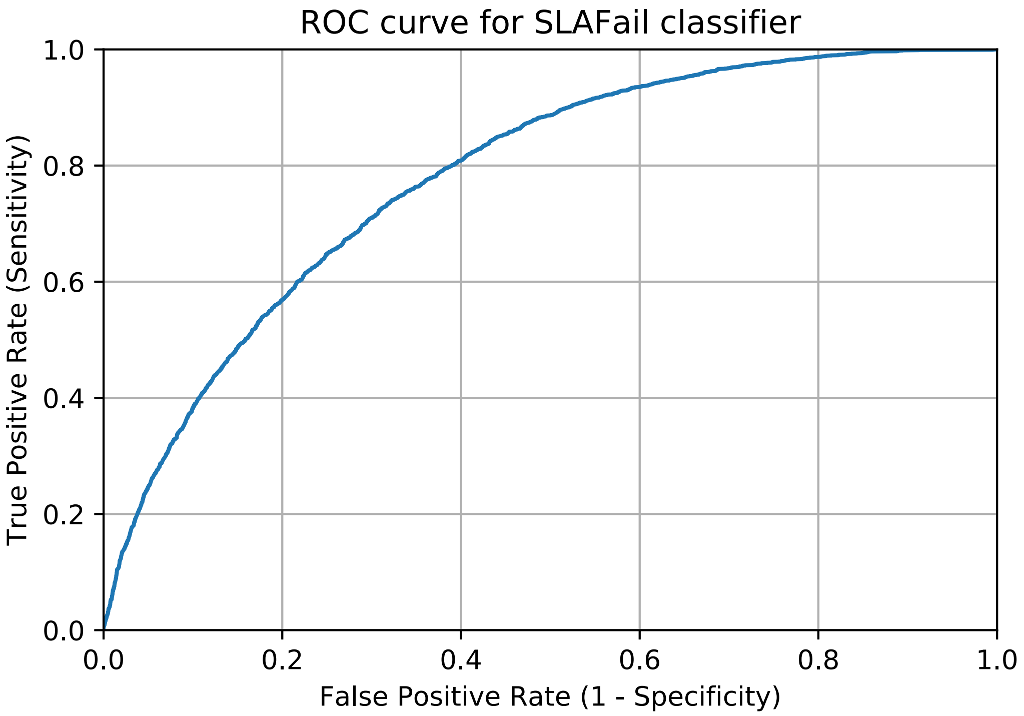
```
[[6597  848]
 [1827 1291]]
TN: 6597         FP: 848
FN: 1827         TP: 1291
```

Add some ROC curve information and AUC result

```
[31] fpr, tpr, thresholds = metrics.roc_curve(y_test, y_pred_prob)
     metrics.roc_auc_score(y_test, y_pred_prob)
```

```
0.7806632430855996
```

```
[32] plt.plot(fpr, tpr)
     plt.xlim([0.0, 1.0])
     plt.ylim([0.0, 1.0])
     plt.title('ROC curve for SLAFail classifier')
     plt.xlabel('False Positive Rate (1 - Specificity)')
     plt.ylabel('True Positive Rate (Sensitivity)')
     plt.grid(True)
```



Capture resulting AUC

Save Details and Performance Measures for Comparison to other Models

Capture a few classification metrics:

- Classification Accuracy: Overall, how often is the classifier correct?
- Classification Error: Overall, how often is the classifier incorrect?

- True Positive Rate (Recall, Sensitivity): When the actual value is positive, how often is the prediction correct?
- True Negative Rate (Specificity): When the actual value is negative, how often is the prediction correct?
- False Positive Rate: When the actual value is negative, how often is the prediction incorrect?
- Precision: When a positive value is predicted, how often is the prediction correct?

```
[38]  pipe.named_steps.selector.scores_
```

```
array([2.62605572e+03, 1.41583607e+02, 6.10443839e+03, 1.77755544e+02,
       6.79762696e+02, 9.02108720e+02, 6.20757993e+01, 5.60722172e+00,
       3.51935177e-01])
```

```
[39]  pipe.named_steps.selector.pvalues_
```

```
array([0.00000000e+000, 1.47282598e-032, 0.00000000e+000, 2.06821476e-
040,
       7.61382496e-148, 1.10035990e-194, 3.44070391e-015, 1.78942746e-
002,
       5.53025157e-001])
```

```
[36]  # returns a mask of features retained
      pipe.named_steps.selector.get_support()
```

```
array([ True,  True,  True,  True,  True,  True,  True,  True, False])
```

```
[37]  # apply mask to X_train column names
      selectedFeatures = np.array(X_train.columns.tolist())
      [pipe.named_steps.selector.get_support().tolist()]
      selectedFeatures
```

```
array(['Service_Component_WBS_aff', 'Urgency', 'KM_number',
       'Count_Related_Interactions', 'Count_Related_Incidents',
       'Count_Related_Changes', 'Open_Time_HourOfDay',
       'Open_Time_DayOfWeek'], dtype='<U26')
```

```
[41]  PipeLineMetadata['Selector - Scores'] = pd.DataFrame(
          [ selectedFeatures, pipe.named_steps.selector.scores_[1:],
      pipe.named_steps.selector.pvalues_[1:] ],
```

```
    index=['feature names', 'scores','p-value']
    ).transpose()
PipeLineMetadata['Selector - Scores']
```

	feature names	scores	p-value
0	Service_Component_WBS_aff	141.584	1.47283e-32
1	Urgency	6104.44	0
2	KM_number	177.756	2.06821e-40
3	Count_Related_Interactions	679.763	7.61382e-148
4	Count_Related_Incidents	902.109	1.10036e-194
5	Count_Related_Changes	62.0758	3.4407e-15
6	Open_Time_HourOfDay	5.60722	0.0178943
7	Open_Time_DayOfWeek	0.351935	0.553025

[42]
```
PipeLineMetadata['Metrics - Classification Report'] =
metrics.classification_report(y_test, y_pred_class)
PipeLineMetadata['Metrics - Confusion Matrix']
=metrics.confusion_matrix(y_test, y_pred_class)
PipeLineMetadata['Metrics - Confusion Matrix Classification
Accuracy'] = metrics.accuracy_score(y_test,y_pred_class)
PipeLineMetadata['Metrics - Confusion Matrix Classification
Error'] = 1- metrics.accuracy_score(y_test,y_pred_class)
PipeLineMetadata['Metrics - Confusion Matrix True Positive Rate']
= metrics.recall_score(y_test, y_pred_class)
PipeLineMetadata['Metrics - Confusion Matrix True Negative Rate']
=  TN / float(TN + FP)
PipeLineMetadata['Metrics - Confusion Matrix False Positive
Rate'] =  FP / float(TN + FP)
PipeLineMetadata['Metrics - Confusion Matrix Precision'] =
metrics.precision_score(y_test,y_pred_class)
```

[43]
```
PipeLineMetadata['Metrics - AUC'] = metrics.roc_auc_score(y_test,
y_pred_prob)
```

[44]
```
PipeLineMetadata['Metrics - ROC Curve fpr array'] = fpr
PipeLineMetadata['Metrics - ROC Curve tpr array'] = tpr
```

[46]
```
PipeLineMetadata
```

```
{'Null Accuracy': 0.7048187068067784,
 'Column Transforms': ['woeencoder', 'standardscaler'],
 'Selector': SelectFpr(alpha=0.05, score_func=<function f_classif at
0x1a2489ae60>),
 'Classifier': LogisticRegression(C=1.0, class_weight=None, dual=False,
fit_intercept=True,
                    intercept_scaling=1, l1_ratio=None, max_iter=100,
                    multi_class='auto', n_jobs=None, penalty='l2',
                    random_state=None, solver='lbfgs', tol=0.0001,
verbose=0,
                    warm_start=False),
 'Classifier - Intercept': -0.9376094581875354,
 'Classifier - Coefficients':                                    0
 Service_Component_WBS_aff   0.104001
 Urgency                     0.188064
 KM_number                   1.121735
 Count_Related_Interactions  1.172629
 Count_Related_Incidents     1.245934
 Count_Related_Changes       0.038820
 Open_Time_HourOfDay         0.182436
 Open_Time_DayOfWeek        -0.035750,
 'Selector - Scores':                 feature names    scores       p-
value
 0   Service_Component_WBS_aff   141.584   1.47283e-32
 1                     Urgency   6104.44             0
 2                   KM_number   177.756   2.06821e-40
 3  Count_Related_Interactions   679.763  7.61382e-148
 4     Count_Related_Incidents   902.109  1.10036e-194
 5       Count_Related_Changes   62.0758    3.4407e-15
 6         Open_Time_HourOfDay   5.60722     0.0178943
 7         Open_Time_DayOfWeek  0.351935      0.553025,
 'Metrics - Classification Report': '              precision    recall
f1-score   support\n\n           0       0.78      0.89      0.83
7445\n           1       0.60      0.41      0.49      3118\n\n
accuracy                           0.75     10563\n   macro avg
0.69      0.65      0.66     10563\nweighted avg       0.73      0.75
0.73     10563\n',
 'Metrics - Confusion Matrix': array([[6597,  848],
        [1827, 1291]]),
 'Metrics - Confusion Matrix Classification Accuracy':
0.7467575499384644,
 'Metrics - Confusion Matrix Classification Error': 0.25324245006153556,
 'Metrics - Confusion Matrix True Positive Rate': 0.414047466324567,
 'Metrics - Confusion Matrix True Negative Rate': 0.8860980523841504,
 'Metrics - Confusion Matrix False Positive Rate': 0.11390194761584957,
 'Metrics - Confusion Matrix Precision': 0.6035530621785882,
 'Metrics - AUC': 0.7806632430855996,
 'Metrics - ROC Curve fpr array': array([0.        , 0.        , 0.
, ..., 0.99946273, 0.99973136,
        1.        ]),
 'Metrics - ROC Curve tpr array': array([0.00000000e+00, 3.20718409e-04,
2.88646568e-03, ...,
        1.00000000e+00, 1.00000000e+00, 1.00000000e+00])}
```

[47]
```
with open(outFileName,'wb') as fo:
    pickle.dump(PipeLineMetadata, fo)
```

[34]
```
# with open("FILENAME", 'rb') as fi:
#     BareBonesMetadata = pickle.load(fi)
```

[ ]