A Data Engineering Practicum Project Report

On the “Automated Rail Inclusion Application” Project

Submitted March of 2023 to the

Anderson College of Business and Computing

Regis University

in partial fulfillment of the requirements for a

Graduate Certificate in Data Science

by

Madeline Stelter

Practicum Project Advisor

Ksenia Polson, Ph.D.

Table of Contents

[Chapter 1-Introduction 1](#_Toc129116877)

[Chapter 2 – Technical Components 3](#_Toc129116878)

[Chapter 3 – Results 4](#_Toc129116879)

[Chapter 4 – Conclusions 13](#_Toc129116880)

# 

# Chapter 1-Introduction

**Overview**

The MSDE692 – Data Engineering (DE) Practicum course satisfies the Graduate Final Project requirement for graduation from Graduate Certificate of Data Science. A total of 12 credit hours of course work is required to graduate from RU with a Data Science Certificate. This course is taken as the last course within the program. The goal of the Data Engineering Practicum project is to facilitate a real-life hands-on learning experience via the development of data infrastructure for use by Data Scientists. Each student selects their own DE project to customize. The Database Practicum Project allows students to demonstrate their design and coding skills that they have acquired throughout their course work.

**Background:**

In day-to-day work as a process engineer for a Steel Meltshop, large amounts of data are generated and stored on a regular basis. This data is accessed often to monitor trending, look for correlation and causation, and to explore process capability. Many of the current access points for this data are cumbersome, and result in excel or pdf print outs that are limited in size to a few weeks’ worth of data in a single download. Collecting this data into a single source for analysis required an extensive amount of manual work, that was difficult to teach to new employees. I realized that through the correct application of Python code, it would be possible to connect directly to the data warehouses to find data of interest. From some exploration and research, I found that it might even be possible to automate the code in such a way that it could be run on any machine in the network, without the need for a base Python installation.

**Research Statement:**

The purpose of this research is to develop code capable of connecting to two (2) data warehouses and collecting all of the process data related to the Steel Casting Process and the data related to Rail Inclusions, a common defect. Once collected, the data will be cleaned, organized, and exported to a central location for use in data science visualizations and trending. Finally, the code used to collect the data needs to be accessible to any employee of the company, regardless of coding ability.

**Deliverables Statement:**

All casting data in the system from 2015 to present is to be collected for each batch of product made. All defect data for the same series of products is to be collected. The data is to be cleaned and organized so that all variables and defect data for a specific product are found in a single row. The code is to be packaged into an executable file that runs as a standalone, with no python installation required on the host computer. The final data export will be managed in a PowerBI dashboard to construct visualizations and variable comparison that will automatically update each time the data source is updated by the application.

# Chapter 2 – Technical Components

The data collection code was developed through Jupyter Notebook, managed by an Anaconda installation. The require libraries were numpy, pandas, PySimpleGUI, and PyInstaller. Code languages included Python and SQL. This code was written on and designed for a standard company laptop, which runs Windows 10 Pro on an Intel(R) Core(TM) i7-8560U CPU with 24.0 GB of RAM.

The code design developed SQL based queries to connect to the data warehouses and retrieve specified columns of data. The data is organized and cleaned using numpy and pandas. PySimpleGUI was used to develop a basic graphical interface for employee comfort. PyInstaller was used to package and condense the python files and libraries into a single-file executable.

# Chapter 3 – Results

For full copies of the code files used in the project, please visit <https://github.com/mstelter1/MSDS692-Final-Project>. This section will focus on the code and outputs that satisfied the deliverables of the project.

The first requirement was to connect to the data warehouses within the company. The first connection was developed by an onsite data engineer, as shown in Figure 1. It established a function that built a connection string that would communicate with the onsite servers. Based on the instructional code in Figure 1, I developed the second connection string for the other data warehouse in Figure 2. The functions are set up in an identical matter, but this required that I establish the correct names, passwords, and addresses to be used. Text

Description automatically generated

Figure

Text

Description automatically generated

Figure

The onsite data engineer also developed the first SQL query that would retrieve data from the warehouses, as seen in Figure 3. The original state of the project, which used graphic user interfaces to download excel files and pdf’s, was used to determine the correct table and column names that housed data of interest. I used this example query to develop connections to other tables to retrieve the full spectrum of data, as shown in Figure 4.

Timeline

Description automatically generated with medium confidence

Figure

Text

Description automatically generated with medium confidence

Figure

After the data was collected, some issues from the warehouses such as data type needed to be corrected, as shown in Figure 5. Also, the three queries used were read into three separate data frames, so the data was combined into a single frame for ease of analysis. The code of one of the merges and an example of the outputted data is found in Figure 6. This data included both numerical and text data, as well as some extreme outliers.

Text, letter

Description automatically generated

Figure

Graphical user interface, table

Description automatically generated

Figure

Outliers were identified using both knowledge of the process and physically impossible measurements. These outliers were replaced with NaN markers for the sake of this project. The expected visualizations included line graphs, bar charts, and scatter plots, which would not show negative results from NaN markers. If more machine learning or data analysis were to be performed, additional work would need to be done to replace the outliers with reasonable estimations. Figure 7 shows the use of a version of flooring and capping to replace the extreme outliers with NaN. At this stage of the project, data visualizations were originally being developed using Plotly and Dash, to generate an interactive dashboard each time the data was collected. It was determined by my IT group that the server host for the Dash library was not secure enough for use in the company. At this point, the project scope was adjusted to manage the visualization in PowerBI, a tool already vetted and approved for company use. Code was developed to export the final data frame into a local excel workbook to be used as a data source, as seen in Figure 8. An example of the excel output is found in Figure 9.

Text, letter

Description automatically generated

Figure

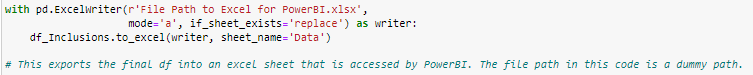


Figure 8

Graphical user interface, application, table, Excel

Description automatically generated

Figure 9

With the code file completed, an interface needed to be developed for users that was more familiar in appearance. The PySimpleGUI library was used to develop a simple window with buttons to launch the program. Figure 10 displays the code used to generate this window.

Graphical user interface, text, application, email

Description automatically generatedFigure 10

With both the data code and the GUI code complete, the code needed to be packaged into an application that any employee could use. The PyInstaller library was used for ease of execution. Figure 11 shows the line of code that was executed in the command prompt and a portion of the program executing the compilation. Figure 12 displays both the final application, its size, and the window that is displayed when the application is launched.

Graphical user interface, text

Description automatically generatedFigure 11

Graphical user interface, application

Description automatically generated

Figure 12

Finally, the data that was exported to excel was managed in PowerBI to develop graphs, charts, and trending typically requested of this data set. All slicers are interconnected, so a selection in one slicer or graph will dynamically shift the others so the same timeframe, product, sequence, etc, can be compared across all visualizations at the same time. Figure 13 shows an example of some of the graphs developed in PowerBI.

Chart, histogram

Description automatically generated

Figure 13

# Chapter 4 – Conclusions

Some vulnerabilities in this project were discovered in the course of the research. Given the condensed final package, either extensive coding knowledge or access to the original code would be required to implement updates. For example, if a password or database address was changed, the code to collect the data would fail. The final project output is mildly clunky; it requires running one python based application without much visual context that the program is complete, and then requires a separate launch of a PowerBI dashboard. This could be streamlined, possibly into a single user interfaces that allows access to both programs. The code to collect and clean the data is contained in a single file executable that was successfully run on company computers with no Python installations. Overall, the final product allows myself or other employees to monitor the data related to Rail Inclusions at a significantly faster pace, with minimal work involved on the front end. More of our time can be devoted to analysis and improvement of the defect rate.