Predicting Gross Revenue with CMS Cost Reports

By

Marissa Munoz-Ruiz

A Capstone Submitted to:

The M.S. Health Data Science Program

Department of Health and Clinical Outcomes Research

School of Medicine

Saint Louis University

Saint Louis, Missouri

December 19, 2022

TABLE OF CONTENTS

Page

LIST OF FIGURES iii

1 ABSTRACT 1

2 INTRODUCTION 2

3 METHODS 3
- 3.1 Data Science Platforms 3
- 3.2 Data Preprocessing 3
- 3.3 Machine Learning 4

4 RESULTS 5
- 4.1 Data Cleaning 5
- 4.2 Model Performance 5

5 DISCUSSION 7
- 5.1 Data Science Platforms 7
- 5.2 Model Comparison 8
- 5.3 Limitations 9
- 5.4 Future work 9

6 REFERENCES 10

REFERENCES 11

7 APPENDIX 13

LIST OF FIGURES

Figure Page

4.1 Model performance when predictors that had a null percentage greater than 25% were removed in preprocessing. 6

4.2 Model performance when predictors that had a null percentage greater than 50% were removed in preprocessing. 6

1. ABSTRACT

CMS Hospital Cost reports provide substantial facility and financial information about hospitals across the United States. Regression machine learning algorithms were used to predict gross revenue from CMS Hospital Cost Reports. With a 80% training set and 20% testing set with 5-Fold Cross Validation training, the Extreme Gradient Boosted Decision Tree outperformed the Multiple Linear Regression algorithm. The XGBoost model had a training RMSE of 40 million and a testing RMSE of 20 million. Although, the computation time of the MLR model was 5 times faster than the XGBoost model, the XGBoost model captured more information from the data and had a better variance-bias trade-off.

2. INTRODUCTION

Medicare institutional providers such as hospitals, skilled nursing facilities, home health agencies, renal facilities, hospices facilities, and organ procurement organizations are required to submit an annual Medicare cost report to the Centers for Medicare and Medicaid Services (CMS) (The Centers for Medicare & Medicaid, 2020). Each type of provider has a unique Medicare cost report form that provides information about facility characteristics, utilization, cost and charges, settlement data, and financial statements. The cost reports are used to determine if CMS over or unpaid the provider, to set prospective payment rates, and to investigate the cost and financial health of a facility (The Centers for Medicare & Medicaid, 2013). A subset of the Hospital Provider Cost report data is available to the public and provided directly by CMS (The Centers for Medicare & Medicaid, 2022b).

Research studies have used Hospital Provider Cost reports to calculate costs for post-acute care claims (Coomer et al., 2017), to investigate the relationship between hospital financial performance and self-reported quality of care (Akinleye et al., 2019) and to investigate the relationship between hospital financial performance and marketing expenditures (Harrison et al., 2022). Machine learning algorithms have yet to be used in any research associated with cost reports. Therefore this capstone project will use supervised machine learning techniques to determine if hospital revenue can be predicted by facility characteristics and financial information provided by Hospital Provider Cost reports.

3. METHODS

3.1 Data Science Platforms

Two data science platforms, Anaconda and Databricks, were used for code development on a windows 10 Pro Dell Inspiron 5565 laptop (Dell Inc., 2017). The Anaconda python version used was 3.7.3 (build version py37_1) and the open source windows 64-Bit Graphical Installer was downloaded locally (Anaconda Inc., 2022). The integrated development environment (IDE) used from Anaconda was Jupyter version 4.4.0. The Databricks Community edition, which is the free version of the cloud-based big data science platform Databricks was used online (Databricks Inc., 2022a). The Databricks Python version was 3.9.5 and the cluster runtime version was standard 11.0 (Scala 2.12, Spark 3.3.0) with 0 workers and 1 Driver (15.3 GB Memory, 2 cores, 1 DBU). Both Pyspark and Python programming languages were used.

3.2 Data Preprocessing

Jupyter was used to web scrape UUIDs from the CMS website (see Appendix: SLU_Capstone_Selenium Notebook). The UUIDs are a unique identifier for each year the CMS Hospital Cost Report is available to the public. The data for each cost report can be accessed via a url that contains the UUID. With a Jupyter notebook, the UUIDs were retrieved for each dataset year from the Hospital Provider Cost Report API documentation (The Centers for Medicare & Medicaid, 2022a) and exported as a csv file locally. The Python web scraping module used to retrieve the UUIDs was selenium (Scrapfly, 2022). Selenium was used in order to bypass JavaScript access issues specific to the CMS website. To use Selenium a web driver has to be downloaded locally in the same location as the Jupyter notebook. The chrome WebDriver version 107 was used (Chromium, 2017).

The csv file containing the UUIDs was directly uploaded into Databrick's virtual directory. The UUIDs were used to import each annual Hospital Cost Report data set via a url (see Appendix: SLU_Capstone_Processing Notebook). The data within the url was a JSON format and imported as a Pyspark Dataframe (Stack Overflow, 2019). The same columns occurring across all annual data sets were kept and combined into a single data set. All categorical columns were removed from the data set. Columns that summed to a total value where the total value column was included in the data set were dropped (The Centers for Medicare & Medicaid., 2020). Individual predictors that contained more than 25% null values were removed. Rows that did not have a value for the dependent variable were also removed. Feature selection was performed with the Scikit-learn machine learning module via the SelectKBest function (Brownlee, 2022; Veigas, 2021). The predictors with an importance score greater than the average importance score across all predictors were used for modeling.

3.3 Machine Learning

Multiple machine learning models were developed to predict gross revenue from the Hospital Cost Report data set (see Appendix: SLU_Capstone_ML Notebook). The Patsy and Scikit-learn modules were used to split the data set into a 80% training set and 20% testing set with 5-Fold Cross Validation training. The LinearRegression function from Scikit-learn was used to build a Multiple Linear Regression (MLR). The xgboost module was used to build an Extreme Gradient Boosted Decision Tree (XGBoost). The XGBoost model had a maximum tree depth of 1 and a learning rate of 0.1. TensorFlow and Keras modules were used to build a Neural Network (NN). The NN had two layers with 64 nodes and two relu activation functions. The performance metric used to compare all models was root mean square error (RMSE).

4. RESULTS

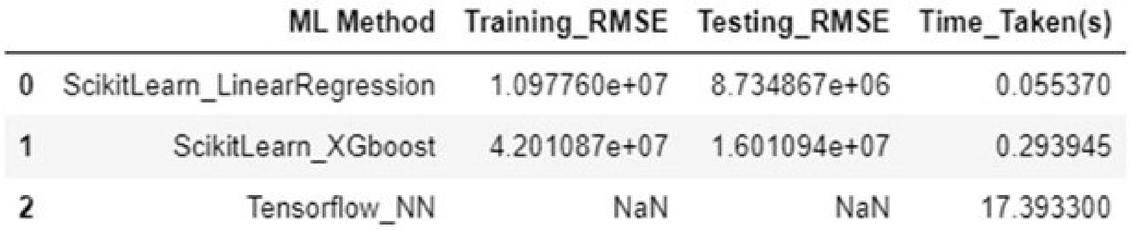
4.1 Data Cleaning

The CMS Hospital Cost reports from 2011 to 2019 were combined across the same columns. The combined data set initially had 55,789 rows and 116 columns. There were 5 categorical predictors that were removed. There were 51 quantitative predictors that summed to a total value column where the total value column was included. There were 28 predictors that had a null percentage greater than 25%. After feature selection, the data set used in the machine learning algorithms had 53,508 rows and 32 predictors.

4.2 Model Performance

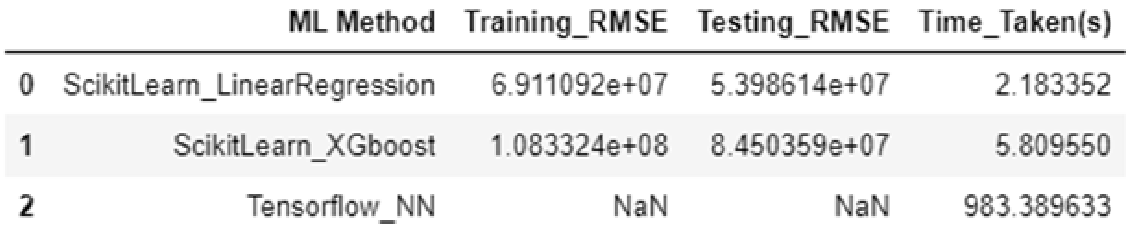
The training RMSE, the testing RMSE, and the length of the computation for each algorithm was computed (Fig. 4.1). The MLR model had a testing RMSE of approximately 9 million and the XGBoost model had a testing RMSE of 20 million. The smallest RMSE was the testing set for the MLR algorithm. The computation time of the MLR model was 5 times faster than the XGBoost model. Neural Networks can't handle null values and therefore did not have any results. The mean gross revenue was approximately 500 million ± 1 billion (see Appendix: Capstone_Preprocessing notebook for dependent variable statistics).

To see how the null values affected model performance, the algorithms were rerun with different predictors. Instead of removing 28 predictors that had a null percentage greater than 25%, 21 predictors that had a null percentage greater than 50% were removed (Fig. 4.2). After feature selection, the data set used in the machine learning algorithms had 53,508 rows and 39 predictors. The MLR model had a testing RMSE of approximately 60 million and the XGBoost model had a testing RMSE of 80 million. The smallest RMSE was the testing set for the MLR algorithm. The computation time of the MLR model was 2.5 times faster than the XGBoost model.



	ML Method	Training_RMSE	Testing_RMSE	Time_Taken(s)
0	ScikitLearn_LinearRegression	1.097760e+07	8.734867e+06	0.055370
1	ScikitLearn_XGboost	4.201087e+07	1.601094e+07	0.293945
2	Tensorflow_NN	NaN	NaN	17.393300

Fig. 4.1: Model performance when predictors that had a null percentage greater than 25% were removed in preprocessing.



	ML Method	Training_RMSE	Testing_RMSE	Time_Taken(s)
0	ScikitLearn_LinearRegression	6.911092e+07	5.398614e+07	2.183352
1	ScikitLearn_XGboost	1.083324e+08	8.450359e+07	5.809550
2	Tensorflow_NN	NaN	NaN	983.389633

Fig. 4.2: Model performance when predictors that had a null percentage greater than 50% were removed in preprocessing.

5. DISCUSSION

5.1 Data Science Platforms

Both Jupyter and Databricks handled the computational needs of the data set and the machine learning algorithms. Databricks has built in parallel processing and distributed computing within notebooks that automatically makes data processing faster and more efficient. Although Databricks has its own unique machine learning environment and modules, it is still being actively developed. Therefore, all machine learning was built with Python modules within a Databricks notebook. The use of Jupyter was necessary because Selenium has a lot of known issues with Databricks (Databricks Inc., 2022b; Databricks Inc., 2022c; Stack Overflow, 2020). With a local setup, Jupyter easily accessed the chromedriver.exe file. Although the chromedriver.exe file was directly uploaded into the Databricks virtual directory (DBFS), Selenium was unable to access it.

With Jupyter, all available data sets will be exported as a csv file, which ensures the latest data set will be analyzed. Although the size of the data set was relatively small, all the preprocessing in Databricks was performed with Pyspark dataframes. Pyspark dataframes can handle billions of rows of data within a few seconds. The computational power of Pyspark within databricks ensures that as the CMS cost report grow in size and volume over time this code will still work efficiently. The use of multiple data science platforms and programming languages can be beneficial during data ingestion, data preparation, and data modeling. As big data continues to gain more popularity in healthcare, data scientists will need creative solutions to handle complex data sets.

5.2 Model Comparison

Two predictors had substantial importance scores during feature selection (see Appendix: Capstone_Preprocessing notebook). The outpatient_total_charges predictor is the total outpatient gross patient charges including charity care for that cost center and the less_contractual_allowance_and_discounts_on_patients' accounts is the total patient revenues that were not received (The Centers for Medicare & Medicaid., 2020). The outpatient total charges predictor had 7.26% null values and the less_contractual_allowance_and_discounts_on_patients' accounts had 1.29% null values. Predictors with higher percentages of null values tended to have a lower importance score. It seems that both MLR and XGBoost machine learning algorithms can handle a certain amount of null values but there is a threshold.

A lower RMSE generally reflects better model performance. However, in machine learning models the RMSE for both the training and testing subsets have to be taken into account. In the first iteration of the machine learning algorithms, predictors that had a null percentage greater than 25% were removed (Fig. 4.1). In this situation, the machine learning model with the best performance is XGBoost because the training RMSE (40 million) and the test RMSE (16 million) are within a reasonable range of each other i.e. the RMSE value have like terms (the same exponent). The training and testing subsets have a similar performance which indicates that there is reasonable bias-variance tradeoff.

In the second iteration of the machine learning models, the predictors that had a null percentage greater than 50% were removed (Fig. 4.2). In this situation, there are more predictors in the models as the feature selection is more flexible. The RMSE for both models increased which indicates that including more predictors results in worse performance. The extra predictors are not adding any additional information. Having the top two predictors from feature selection as the only predictors in the models would most likely result in the best performance with the smallest RMSE value.

5.3 Limitations

There are several limitations regarding the data set. It is unclear how the hospitals collected their data for each annual CMS Hospital Cost Report. The validity of the data set comes into question as there are no requirements that the public is aware of with regards to hospital financial information. The financial nature of the data set itself makes it difficult to parse out predictors because so many predictors were related. Although related predictors were removed, there may have been important information in the removed predictors that outweigh the predictor that was included in the data set. The amount of null values may impact the ability of machine learning models to capture the most relevant information. Of the 32 predictors used in the model there were only three predictors that did not have null values.

5.4 Future work

The context and understanding of predictors should be confirmed with an expert in the field. Due to the lack of context, there may be predictors that were not captured by feature selection. The MLR and XGBoost models should also be rerun: first with all predictors and then secondly with the top predictors from feature selection to better understand how null values affect the model performance. The value of predicting gross revenue also needs to be addressed with hospitals and CMS to ensure this information can be implemented to improve resource allocation and patient outcomes.

6. REFERENCES

REFERENCES

REFERENCES

Akinleye, D. D., McNutt, L.-A., Lazariu, V., and McLaughlin, C. C. (2019). Correlation between hospital finances and quality and safety of patient care. *PLOS ONE*, 14(8):1–19.

Anaconda Inc. (December 15 2022). Jupyter Notebooks. `https://www.anaconda.com/products/distribution`.

Brownlee, J. (August 18 2022). How to Perform Feature Selection for Regression Data. `https://machinelearningmastery.com/feature-selection-for-regression-data/`.

Chromium (2017). ChromeDriver. `https://chromedriver.chromium.org/downloads`.

Coomer, N. M., Ingber, M., Coots, L. A., and Morley, M. (2017). Using medicare cost reports to calculate costs for post-acute care claims. *RTI Press*, OP-0036-1701.

Databricks Inc. (December 15 2022a). Databricks Community Edition. `https://community.cloud.databricks.com/login.html`.

Databricks Inc. (July 29 2022c). How Selenium Webdriver works on Azure Databricks? I am unable to run a simple code. `https://community.databricks.com/s/question/0D58Y000090N1IpSAK/how-selenium-webdriver-works-on-azure-databricks-i-am-unable-to-run-a-simple-code`.

Databricks Inc. (November 1 2022b). Errors Using Selenium/Chromedriver in DataBricks. `https://community.databricks.com/s/question/0D58Y00009PlBaaSAF/errors-using-seleniumchromedriver-in-databricks`.

Dell Inc. (2017). Inspiron 5565, Version 10.0.19044 Build 19044. `https://www.dell.com/en-us`.

Harrison, B. M. D. P., Cockrell, S. R., Ferrell, B. P., and Edison, W. (2022). The influence of financial performance on marketing expenditures among u.s. hospitals participating in the medicare program. *Health Marketing Quarterly*, 39(2):150–158. PMID: 34736357.

Scrapfly (January 10 2022). Web Scraping with Selenium and Python Tutorial + Example Project. `https://scrapfly.io/blog/web-scraping-with-selenium-and-python/`.

Stack Overflow (June 18 2020). How to use Selenium in Databricks and accessing and moving downloaded files to mounted storage and keep Chrome and ChromeDriver versions in sync? `https://stackoverflow.com/questions/67830079/how-to-use-selenium-in-databricks-and-accessing-and-moving-downloaded-files-to-m`.

Stack Overflow (September 15 2019). Python urllib2: Receive JSON response from url. https://stackoverflow.com/questions/13921910/python-urllib2-receive-json-response-from-url.

The Centers for Medicare & Medicaid. (April 21 2020). Hospital Provider Cost Report Data Dictionary. https://data.cms.gov/resources/hospital-provider-cost-report-data-dictionary.

The Centers for Medicare & Medicaid (April 4 2013). Introduction to Medicare Cost Reports. http://resdac.umn.edu/sites/resdac.umn.edu/files/Introduction%20to%20Medicare%20Cost%20Reports%20(Slides).pdf.

The Centers for Medicare & Medicaid (August 4 2020). Cost Reports. https://www.cms.gov/research-statistics-data-and-systems/downloadable-public-use-files/cost-reports.

The Centers for Medicare & Medicaid (June 15 2022b). Hospital Provider Cost Report. [Data file]. https://data.cms.gov/provider-compliance/cost-report/hospital-provider-cost-report#details/.

The Centers for Medicare & Medicaid (September 30 2022a). Access API for Hospital Provider Cost Report. https://data.cms.gov/provider-compliance/cost-report/hospital-provider-cost-report/api-docs.

Veigas, R. (August 1 2021). Feature Selection using Filter Methods. https://www.linkedin.com/pulse/feature-selection-using-filter-methods-runa-veigas.

7. APPENDIX

databricks SLU_Capstone_Selenium

(https://databricks.com)

Author: Marissa Munoz-Ruiz

SLU Capstone: HDS 5960

Jupyter Notebook: SLU_Capstone_Selenium

Goal of Script: Web Scrape UUIDs from CMS website

- The module Selenium was used to web scrape https://data.cms.gov/provider-compliance/cost-report/hospital-provider-cost-report/api-docs/ (https://data.cms.gov/provider-compliance/cost-report/hospital-provider-cost-report/api-docs/)
- To use Selenium, a chromedrive.exe file has to be downloaded and saved to same location as jupyter notebook
    - The stable version ChromeDriver 107.0.5304.62 was used from the website https://chromedriver.chromium.org/home (https://chromedriver.chromium.org/home)

```
#pip install selenium
```

```
#pip show selenium
```

```
import os
import pandas as pd
from selenium import webdriver
from selenium.webdriver.chrome.service import Service
from selenium.webdriver.chrome.options import Options
from selenium.webdriver.common.by import By

cms_url = "https://data.cms.gov/provider-compliance/cost-report/hospital-
provider-cost-report/api-docs/"

# configure webdriver
options = Options()
options.headless = True  # hide GUI
options.add_argument("--window-size=1920,1080")  # set window size to native
GUI size
options.add_argument("start-maximized")  # ensure window is full-screen

# get path of chromedrive
path = os.getcwd() + '\\chromedriver.exe'
print("chromedriver location: {}".format(path))

# access data.cms.gov website
s = Service(path)
driver = webdriver.Chrome(service=s, options=options)
driver.get(cms_url)
print("driver information: {}".format(driver))
```

```
chromedriver location: C:\Users\mmuno\Desktop\GitHub\Capstone\chromedriver.exe
driver information: <selenium.webdriver.chrome.webdriver.WebDriver (session="4
35cd35e2be0b17c76f056b54ca6491f")>
```

```
# get row count of table
rows = len(driver.find_elements(by=By.XPATH,
value="/html/body/div/div/div/div/div/div/div[2]/div[2]/div/div/table/tbody/tr"
))
cols = len(driver.find_elements(by=By.XPATH,
value="/html/body/div/div/div/div/div/div/div[2]/div[2]/div/div/table/thead/tr/
th"))
```

```
# Webscrape cms website
version_ls = []

for r in range(1,rows+1):
    values = []
    for c in range(1,cols+1):
        xpath =
"/html/body/div/div/div/div/div/div/div[2]/div[2]/div/div/table/tbody/tr["+str(
r)+"]/td["+str(c)+"]"
        driver_value = driver.find_elements(by=By.XPATH, value=xpath)
        col_value = driver_value[0].text
        values.append(col_value)

    version_ls.append(values)

# create dataframe with webscraped information from cms website
version_df = pd.DataFrame(version_ls, columns =['year', 'version'])
version_df
```

	year	version
0	2019	6ebd03b1-ff48-4994-94ed-0a54e90c1bd6
1	2018	90869abf-c649-4d65-84b3-4d6a1b568b69
2	2017	b2a1e8c3-62c3-4c47-94b2-5fa16b122a4d
3	2016	2981f550-653f-46a6-a5a5-06a3408eb245
4	2015	73e66edc-0b70-4e88-b1af-7d2b98a243f5
5	2014	9855c8b2-1514-47f8-a3ed-c34c2d41eed3
6	2013	7d1090ac-9d79-47d6-b42d-533a1f3edd7a
7	2012	9f3eee40-cdbc-4082-af3f-30e1807399b9
8	2011	db36eabb-2344-4053-a579-6fa48602bb29

```
version_df.to_csv("URL_Versions.csv", index=False) #export data as excel file
```

databricks SLU_Capstone_Preprocessing

(https://databricks.com)

Author: Marissa Munoz-Ruiz
SLU Capstone: HDS 5960
Databricks Notebook: SLU_Capstone_Preprocessing

Goal of Script: Import & Preprocess CMS Cost Report Data for ML Models

- Use UUIDs csv file retrieved from CMS website (see SLU_Capstone_Selenium) to import data
- Perform data cleaning and feature selection to produce dataframe for Machine Learning

Import webscraped URLs from csv

```
#See https://data.cms.gov/provider-compliance/cost-report/hospital-provider-cost-report#details and
#"https://data.cms.gov/provider-compliance/cost-report/hospital-provider-cost-report/api-docs for information related to
the data

#https://scrapfly.io/blog/web-scraping-with-selenium-and-python/
from pyspark.sql.functions import concat, col, lit

version_dir = "/FileStore/tables/URL_Versions.csv"
version_df = spark.read.format("csv").options(header='True',delimiter=',').load(version_dir)

api_url = 'https://data.cms.gov/data-api/v1/dataset/'
uuid_df = (version_df
           .withColumnRenamed('version','uuid')
           .withColumn('url',concat(lit(api_url),col('uuid'),lit('/data.json')))
          )

uuid_df.display()
```

Table

	year	uuid	url
1	2019	6ebd03b1-ff48-4994-94ed-0a54e90c1bd6	https://data.cms.gov/data-api/v1/dataset/6ebd03b1-ff48-4994-94ed-0a54e90c1bd6/da
2	2018	90869abf-c649-4d65-84b3-4d6a1b568b69	https://data.cms.gov/data-api/v1/dataset/90869abf-c649-4d65-84b3-4d6a1b568b69/d
3	2017	b2a1e8c3-62c3-4c47-94b2-5fa16b122a4d	https://data.cms.gov/data-api/v1/dataset/b2a1e8c3-62c3-4c47-94b2-5fa16b122a4d/da
4	2016	2981f550-653f-46a6-a5a5-06a3408eb245	https://data.cms.gov/data-api/v1/dataset/2981f550-653f-46a6-a5a5-06a3408eb245/da
5	2015	73e66edc-0b70-4e88-b1af-7d2b98a243f5	https://data.cms.gov/data-api/v1/dataset/73e66edc-0b70-4e88-b1af-7d2b98a243f5/da
6	2014	9855c8b2-1514-47f8-a3ed-c34c2d41eed3	https://data.cms.gov/data-api/v1/dataset/9855c8b2-1514-47f8-a3ed-c34c2d41eed3/da
7	2013	7d1090ac-9d79-47d6-b42d-533a1f3edd7a	https://data.cms.gov/data-api/v1/dataset/7d1090ac-9d79-47d6-b42d-533a1f3edd7a/d

Showing all 9 rows.

Read in data from CMS website

```
from pyspark.sql import SparkSession, DataFrame
from urllib.request import urlopen
from functools import reduce
import json

# Use pyspark to read json from url
# https://stackoverflow.com/questions/13921910/python-urllib2-receive-json-response-from-url
def json_to_DF(url):
    spark = SparkSession.builder.getOrCreate()
    response = urlopen(url)
    jsonData = json.load(response)
    rdd = spark.sparkContext.parallelize(jsonData)
    df = spark.read.json(rdd)
    return df
```

Get unique columns across all dataset years

```
from collections import Counter

# Create list of url & year from UUID dataframe
uuid_ls = uuid_df.rdd.map(lambda x: x['url']).collect()
year_ls = uuid_df.rdd.map(lambda x: x['year']).collect()

cols_ls = []
cols_len = []

# Import data & get column names
for url in uuid_ls:
    col_names = json_to_DF(url).columns
    cols_ls.append(col_names)
    cols_len.append(len(col_names))

# Get occurrence of each column name
cols_flat = [element for innerList in cols_ls for element in innerList]
cols_occurrence = Counter(cols_flat)

# Get columns occurring across all dataset years
common_cols = [k for k,v in cols_occurrence.items() if v == len(cols_len)]
```

Combine data with the same columns

```
df_ls = []

for ind, uuid in enumerate(uuid_ls):
    df = json_to_DF(uuid)
    df = df.select(*common_cols).withColumn("data_year", lit(year_ls[ind]))
    df_ls.append(df)

data_df = reduce(DataFrame.unionAll, df_ls)
data_df = data_df.toDF(*[c.lower() for c in data_df.columns]) #make column names lowercase

data_rows = data_df.count()
data_cols = len(data_df.columns)

print("Number of rows: {}".format(data_df.count()))
print("Number of cols: {}".format(len(data_df.columns)))
```

```
Number of rows: 55789
Number of cols: 116
```

Preprocessing - Remove irrelevant columns

```
from pyspark.sql.functions import when
from pyspark.sql.types import StringType

# convert empty rows to null
data_new = data_df.select([when(col(c)=="",None).otherwise(col(c)).alias(c) for c in data_df.columns])

# list of irrevalent numerical columns
num_cols = [
             'hospital name',
             'street address',
             'city',
             'state code',
             'zip code',
             'county',
             'medicare cbsa number',
             'fiscal year begin date',
             'fiscal year end date',
             'total days title v',
             'total days title xviii',
             'total days title xix',
             'number of beds',
             'total discharges title v',
             'total discharges title xviii',
             'total discharges title xix',
             'total days title v + total for all subproviders',
             'total days title xviii + total for all subproviders',
             'total days title xix + total for all subproviders',
             'number of beds + total for all subproviders',
             'total bed days available + total for all subproviders',
             'total discharges title v + total for all subproviders',
             'total discharges title xviii + total for all subproviders',
             'total discharges title xix + total for all subproviders',
             'total discharges (v + xviii + xix + unknown)',
             'total days (v + xviii + xix + unknown)',
             'hospital total days title v for adults & peds',
             'hospital total days title xviii for adults & peds',
             'hospital total days title xix for adults & peds',
             'hospital number of beds for adults & peds',
             'hospital total discharges title v for adults & peds',
             'hospital total discharges title xviii For adults & peds',
             'hospital total discharges title xix for adults & peds',
             'cost of charity care',
             'inpatient Total Charges',
             'combined Outpatient + Inpatient Total Charges',
             'other current assets',
             'total current assets',
             'total fixed assets',
             'other assests',
             'total other assets',
             'other current liabilities',
             'total current liabilities',
             'Other long term liabilities',
             'total long term liabilities',
             'general fund assets',
             'general fund liabilities',
             'total liabilities and fund balances',
             'allowable dsh percentage',
             'drg amounts other than outlier payments',
             'drg amounts before October 1',
             'drg amounts after October 1',
             'outlier payments for discharges',
             'outpatient revenue',
             'inpatient revenue',
             'total costs',
             'total charges',
             'net patient revenue',
             'less total operating expenses',
             'total other income',
             'total income',
             'total other expenses',
             'rpt_rec_num',
             'data_year'
            ]

#list of factor/categorical columns
factor_cols = ['ccn facility type',
               'provider ccn',
               'rural versus urban',
               'provider type',
               'type of control'
              ]


# drop irrelevant columns and rows
data_clean = (data_new
              .drop(*num_cols, *factor_cols)
              .filter(col('gross revenue').isNotNull())
             )

clean_rows = data_clean.count()
clean_cols = len(data_clean.columns)

print("Number of rows: {}".format(clean_rows))
print("Number of cols: {}".format(clean_cols))
```

```
Number of rows: 53508
Number of cols: 60
```

Percentage of null values in each column

```
from pyspark.sql.functions import isnull, count, round
import pandas as pd

#check how many non-null values there are in each column
null_perc = data_clean.select([count(when(isnull(c), c)).alias(c) for c in data_clean.columns])

#convert pyspark to pdf and transform columns
null_perc_pdf = null_perc.toPandas()
null_perc_pdf = pd.melt(null_perc_pdf)
null_perc_pdf.columns = ["column","count"]

#get perc of null values
null_perc_melt = spark.createDataFrame(null_perc_pdf)
total_rows = data_clean.count()

null_perc_df = (null_perc_melt
                .withColumn('perc',round((col('count')/total_rows)*100,2))
                .orderBy(col('perc').desc())
               )

null_perc_df.display()
```

Table

	column	count	perc
1	unsecured loans	51093	95.49
2	notes receivable	50577	94.52
3	wage-related costs (rhc/fqhc)	50000	93.44
4	health information technology designated assets	48531	90.7
5	wage related costs for interns and residents	47023	87.88
6	deferred income	46511	86.92
7	mortgage payable	45548	85.12
8	net revenue from stand-alone schip	44518	83.2
9	stand-alone schip charges	44405	82.99
10	minor equipment depreciable	44298	82.79

11	wage related costs for part - a teaching physicians	44282	82.76
12	total ime payment	43667	81.61
13	number of interns and residents (fte)	42386	79.21
14	temporary investments	41316	77.21
15	managed care simulated payments	40479	75.65
16	payroll taxes payable	32969	61.62
17	investments	30683	57.34
18	notes payable	30218	56.47
19	leasehold improvements	29286	54.73
20	disproportionate share adjustment	28460	53.19
21	notes and loans payable (short term)	27004	50.47
22	contract labor	24643	46.05
23	fixed equipment	22786	42.58
24	wage-related costs (core)	20239	37.82
25	land improvements	18415	34.42
26	total salaries (adjusted)	17522	32.75
27	less: allowances for uncollectible notes and accounts receivable	15440	28.86

Showing all 60 rows.

Remove columns w/ excessive null values

```
#find columns that have 50% or more null values within the column
null_cols = null_perc_df.filter(col('perc') > 25)
null_cols_ls=null_cols.rdd.map(lambda x: x['column']).collect()

data_clean_nulls = data_clean.drop(*null_cols_ls)

data_clean_nulls_rows = data_clean_nulls.count()
data_clean_nulls_cols = len(data_clean_nulls.columns)

print("Number of rows: {}".format(data_clean_nulls_rows))
print("Number of cols: {}".format(data_clean_nulls_cols))
```

```
Number of rows: 53508
Number of cols: 32
```

Remove spaces & symbols from column names

```
#remove spaces from column names
model_data = data_clean_nulls.select([col(c).alias(c.replace('+', '')
                                                    .replace(':', '')
                                                    .replace("'", '')
                                                    .replace(',', '')
                                                    .replace('-', '')
                                                    .replace('/', '')
                                                    .replace('(','')
                                                    .replace(')','')
                                                    .replace('(','')
                                                    .replace(')','')
                                                    .replace(' ','_')
                                                    .replace('__','_')) for c in data_clean_nulls.columns])

#convert pyspark dataframe into pandas dataframe
model_data_pdf = model_data.toPandas()
```

Determine influential features

```
#https://machinelearningmastery.com/feature-selection-for-regression-data/
#https://www.linkedin.com/pulse/feature-selection-using-filter-methods-runa-veigas
```

```
import numpy as np
from patsy import dmatrices
from sklearn.model_selection import train_test_split
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import f_regression
from matplotlib import pyplot

#patsy handles one-hot encoding
## Create formula for all variables in model
vars_remove = ['gross_revenue']
vars_left = set(model_data_pdf.columns) - set(vars_remove)
formula = "gross_revenue ~ " + " + ".join(vars_left)

## Use Patsy to create model matrices
Y,X = dmatrices(formula,model_data_pdf,return_type='dataframe')

## Split Data into training and sample
X_train, X_test, Y_train, Y_test = train_test_split(X,
                                                    np.ravel(Y), # prevents dimensionality error later!
                                                    test_size=0.20,
                                                    random_state=30)

print('Train', X_train.shape, Y_train.shape)
print('Test', X_test.shape, Y_test.shape)
```

```
Train (18248, 32) (18248,)
Test (4563, 32) (4563,)
```

```
formula
```

```
Out[11]: 'gross_revenue ~ total_bad_debt_expense + medicaid_charges + cost_to_charge_ratio + buildings + major_movable_eq
uipment + salaries_wages_and_fees_payable + total_unreimbursed_and_uncompensated_care + net_income + cash_on_hand_and_in_
banks + total_assets + cost_of_uncompensated_care + depreciation_cost + inventory + other_assets + total_fund_balances +
total_salaries_from_worksheet_a + less_total_operating_expense + total_days_v_xviii_xix_unknown_total_for_all_subprovider
s + accounts_payable + total_bed_days_available + less_contractual_allowance_and_discounts_on_patients_accounts + net_rev
enue_from_medicaid + net_income_from_service_to_patients + outpatient_total_charges + overhead_nonsalary_costs + prepaid_
expenses + total_discharges_v_xviii_xix_unknown_total_for_all_subproviders + fte_employees_on_payroll + accounts_receivab
le + general_fund_balance + total_liabilities'
```

```
# feature selection function
def select_features(X_train, Y_train, X_test):
        # configure to select all features
        fs = SelectKBest(score_func=f_regression, k='all')
        # learn relationship from training data
        fs.fit(X_train, Y_train)
        # transform train input data
        X_train_fs = fs.transform(X_train)
        # transform test input data
        X_test_fs = fs.transform(X_test)
        return X_train_fs, X_test_fs, fs

# apply feature selection function
X_train_fs, X_test_fs, fs = select_features(X_train, Y_train, X_test)

# what are scores for the features
feature_ls =[]

for i in range(len(fs.scores_)):
    feature_ls.append([X.columns[i],fs.scores_[i]])

feature_pdf = pd.DataFrame(feature_ls, columns = ['feature','score'])
feature_desc_pdf = feature_pdf.sort_values(by=['score'],ascending=False).reset_index().drop('index',axis=1)
feature_desc_pdf
```

```
/databricks/python/lib/python3.9/site-packages/sklearn/feature_selection/_univariate_selection.py:302: RuntimeWarning: di
vide by zero encountered in true_divide
  corr /= X_norms
/databricks/python/lib/python3.9/site-packages/sklearn/feature_selection/_univariate_selection.py:307: RuntimeWarning: in
valid value encountered in true_divide
  F = corr ** 2 / (1 - corr ** 2) * degrees_of_freedom
```

	feature	score
0	less_contractual_allowance_and_discounts_on_pa...	872841.844026
1	outpatient_total_charges	294390.117277
2	overhead_nonsalary_costs	73356.805913
3	less_total_operating_expense	71238.323771
4	total_salaries_from_worksheet_a	50450.571723
5	medicaid_charges	45115.986847
6	total_bed_days_available	44753.895495
7	total_discharges_v_xviii_xix_unknown_total_for...	44139.083608
8	depreciation_cost	34046.954648
9	inventory	30812.506232
10	major_movable_equipment	27167.902688
11	total_assets	26602.248139
12	buildings	24492.922340
13	total_days_v_xviii_xix_unknown_total_for_all_s...	22418.859575
14	total_fund_balances	19477.181784
15	fte_employees_on_payroll	19003.433378
16	general_fund_balance	17771.201089
17	accounts_receivable	16122.339385
18	total_unreimbursed_and_uncompensated_care	14271.447754
19	cost_of_uncompensated_care	9099.911176
20	total_bad_debt_expense	7647.701690
21	salaries_wages_and_fees_payable	7635.040459
22	net_income	6950.048888
23	total_liabilities	5473.058593
24	prepaid_expenses	4112.458937
25	accounts_payable	3716.239359
26	other_assets	3551.822933
27	cost_to_charge_ratio	2739.188012
28	cash_on_hand_and_in_banks	2004.027584
29	net_revenue_from_medicaid	237.550037
30	net_income_from_service_to_patients	111.347165
31	Intercept	NaN

```
avg_score = feature_desc_pdf['score'].mean()
top_features = feature_desc_pdf[feature_desc_pdf['score'] > avg_score]
#top_features
#top_features.plot(kind = 'bar', title='Top Features')


top_feature_ls = top_features['feature'].values.tolist()
final_vars = top_feature_ls + ['gross_revenue']
MLdata_pdf = model_data_pdf[final_vars]
#MLdata_pdf
```

	gross_revenue
count	5.350800e+04
mean	5.409652e+08
std	1.082218e+09
min	-1.770319e+08
25%	3.811593e+07
50%	1.271728e+08
75%	5.918225e+08
max	2.939014e+10

databricksSLU_Capstone_ML

(https://databricks.com)

Author: Marissa Munoz-Ruiz
SLU Capstone: HDS 5960
Databricks Notebook: SLU_Capstone_ML

Goal of Script: Process and Compare Regression ML Models

- Multiple Linear Regression, Extreme Gradient Boosting, and Neural Networks were used
  - Note: NN models can't handle null values
- RMSE was used as the performance metric

Install Modules for ML

```
%pip install xgboost tensorflow
```

```
Python interpreter will be restarted.
Requirement already satisfied: xgboost in /local_disk0/.ephemeral_nfs/envs/pythonEnv-1143e325-3842-427c-a637-9db4632dccc
c/lib/python3.9/site-packages (1.7.1)
Requirement already satisfied: tensorflow in /local_disk0/.ephemeral_nfs/envs/pythonEnv-1143e325-3842-427c-a637-9db4632dc
ccc/lib/python3.9/site-packages (2.11.0)
Requirement already satisfied: numpy in /databricks/python3/lib/python3.9/site-packages (from xgboost) (1.20.3)
Requirement already satisfied: scipy in /databricks/python3/lib/python3.9/site-packages (from xgboost) (1.7.1)
Requirement already satisfied: flatbuffers>=2.0 in /local_disk0/.ephemeral_nfs/envs/pythonEnv-1143e325-3842-427c-a637-9db
4632dcccc/lib/python3.9/site-packages (from tensorflow) (22.11.23)
Requirement already satisfied: tensorflow-estimator<2.12,>=2.11.0 in /local_disk0/.ephemeral_nfs/envs/pythonEnv-1143e325-
3842-427c-a637-9db4632dcccc/lib/python3.9/site-packages (from tensorflow) (2.11.0)
Requirement already satisfied: termcolor>=1.1.0 in /local_disk0/.ephemeral_nfs/envs/pythonEnv-1143e325-3842-427c-a637-9db
4632dcccc/lib/python3.9/site-packages (from tensorflow) (2.1.1)
Requirement already satisfied: setuptools in /usr/local/lib/python3.9/dist-packages (from tensorflow) (58.0.4)
Requirement already satisfied: packaging in /databricks/python3/lib/python3.9/site-packages (from tensorflow) (21.0)
Requirement already satisfied: tensorboard<2.12,>=2.11 in /local_disk0/.ephemeral_nfs/envs/pythonEnv-1143e325-3842-427c-a
637-9db4632dcccc/lib/python3.9/site-packages (from tensorflow) (2.11.0)
Requirement already satisfied: astunparse>=1.6.0 in /local_disk0/.ephemeral_nfs/envs/pythonEnv-1143e325-3842-427c-a637-9d
b4632dcccc/lib/python3.9/site-packages (from tensorflow) (1.6.3)
Requirement already satisfied: grpcio<2.0,>=1.24.3 in /local_disk0/.ephemeral_nfs/envs/pythonEnv-1143e325-3842-427c-a637-
9db4632dcccc/lib/python3.9/site-packages (from tensorflow) (1.51.1)
```

```
%run ./SLU_Capstone_Preprocessing
```

```
Number of rows: 55789
Number of cols: 116

Number of rows: 53508
Number of cols: 60
```

Table

	column	count	perc
1	unsecured loans	51093	95.49
2	notes receivable	50577	94.52
3	wage-related costs (rhc/fqhc)	50000	93.44
4	health information technology designated assets	48531	90.7
5	wage related costs for interns and residents	47023	87.88
6	deferred income	46511	86.92
7	mortgage payable	45548	85.12
8	net revenue from stand-alone schip	44518	83.2
9	stand-alone schip charges	44405	82.99
10	minor equipment depreciable	44298	82.79
11	wage related costs for part - a teaching physicians	44282	82.76
12	total ime payment	43667	81.61

	total fte payment	[illegible]	[illegible]
13	number of interns and residents (fte)	42386	79.21
14	temporary investments	41316	77.21
15	managed care simulated payments	40479	75.65
16	payroll taxes payable	32969	61.62
17	investments	30683	57.34
18	notes payable	30218	56.47
19	leasehold improvements	29286	54.73
20	disproportionate share adjustment	28460	53.19
21	notes and loans payable (short term)	27004	50.47
22	contract labor	24643	46.05
23	fixed equipment	22786	42.58
24	wage-related costs (core)	20239	37.82
25	land improvements	18415	34.42
26	total salaries (adjusted)	17522	32.75
27	less: allowances for uncollectible notes and accounts receivable	15440	28.86

Showing all 60 rows. | 1.05 minutes runtime

```
Number of rows: 53508
Number of cols: 32
```

```
Out[11]: 'gross_revenue ~ total_bad_debt_expense + medicaid_charges + cost_to_charge_ratio + buildings + major_movable_eq
uipment + salaries_wages_and_fees_payable + total_unreimbursed_and_uncompensated_care + net_income + cash_on_hand_and_in_
banks + total_assets + cost_of_uncompensated_care + depreciation_cost + inventory + other_assets + total_fund_balances +
total_salaries_from_worksheet_a + less_total_operating_expense + total_days_v_xviii_xix_unknown_total_for_all_subprovider
s + accounts_payable + total_bed_days_available + less_contractual_allowance_and_discounts_on_patients_accounts + net_rev
enue_from_medicaid + net_income_from_service_to_patients + outpatient_total_charges + overhead_nonsalary_costs + prepaid_
expenses + total_discharges_v_xviii_xix_unknown_total_for_all_subproviders + fte_employees_on_payroll + accounts_receivab
le + general_fund_balance + total_liabilities'
```

```
/databricks/python/lib/python3.9/site-packages/sklearn/feature_selection/_univariate_selection.py:302: RuntimeWarning: di
vide by zero encountered in true_divide
  corr /= X_norms
/databricks/python/lib/python3.9/site-packages/sklearn/feature_selection/_univariate_selection.py:307: RuntimeWarning: in
valid value encountered in true_divide
  F = corr ** 2 / (1 - corr ** 2) * degrees_of_freedom
```

Import Libraries

```
## Import Modules
import os
import sys
import time
import random
#import numpy as np
#import pandas as pd
import seaborn as sns
from patsy import dmatrices

from sklearn.linear_model import LinearRegression
from sklearn.linear_model import SGDRegressor
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.pipeline import Pipeline
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.metrics import mean_squared_error as MSE

from xgboost import XGBRegressor
from xgboost import cv
from xgboost import DMatrix

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Dropout
from tensorflow.keras.regularizers import l1
```

Develop Functions for Machine Learning Models

```
def LinearRegression_predict(data_df,y_var):

    t0 = time.time()

    ## Create formula for all variables in model
    vars_remove = [y_var]
    vars_left = set(data_df.columns) - set(vars_remove)
    formula = y_var + " ~ " + " + ".join(vars_left)

    ## Use Patsy to create model matrices
    Y,X = dmatrices(formula,data_df,return_type='dataframe')

    ## Split Data into training and sample
    X_train, X_test, Y_train, Y_test = train_test_split(X,
                                                    np.ravel(Y), # prevents dimensionality error later!
                                                    test_size=0.20,
                                                    random_state=30)

    ## Fit Linear Regression model
    model = LinearRegression(fit_intercept=True)
    model.fit(X_train,Y_train)

    ## Get 5-CV train results
    cv = KFold(n_splits=5,shuffle=True,random_state=None)
    train_results = cross_val_score(model,X_train,Y_train,scoring='neg_mean_squared_error',cv=cv,n_jobs=-1)
    train_rmse = np.sqrt(np.absolute(train_results).mean())

    ## Predict Linear model
    pred = model.predict(X_test)
    test_rmse = np.sqrt(MSE(Y_test, pred))

    t1 = time.time()

    model_str = "ScikitLearn_LinearRegression"

    return [model_str,train_rmse,test_rmse,(t1-t0)]
```

```
def xgb_predict(data_df,y_var):

    t0 = time.time()
    X,Y = data_df.loc[:,data_df.columns != y_var], data_df.loc[:,data_df.columns == y_var]
    X_train, X_test, Y_train, Y_test = train_test_split(X,
                                                        Y, # prevents dimensionality error later!
                                                        test_size=0.20,
                                                        random_state=30)

    ## Use Dmatrix object optimized for XGBoost
    train_dmatrix = DMatrix(data = X_train, label = Y_train)
    test_dmatrix = DMatrix(data = X_test, label = Y_test)

    ## Create CV df w/ hyperparameters
    params = {"objective":"reg:linear",
              'colsample_bytree': 1,
              'learning_rate': 0.1,
              'gamma': 0,
              'max_depth': 1,
              'min_child_weight':1,
              'subsample':1,
              'nthread':3}

#     params = {"objective":"reg:linear",
#                   'colsample_bytree': 1,
#                   'learning_rate': 0.1,
#                   'gamma': 0,
#                   'max_depth': 5,
#                   'min_child_weight':1,
#                   'subsample':1,
#                   'nthread':3}

    ## Fit XGB model
    model = XGBRegressor(**params, verbosity=0)
    model.fit(X_train,Y_train)

    cv = KFold(n_splits=5,shuffle=True,random_state=None)

    ## Get 5-CV train results
    train_results = cross_val_score(model,X_train,Y_train,scoring='neg_mean_squared_error',cv=cv,n_jobs=-1)
    train_rmse = np.sqrt(np.absolute(train_results).mean())

    ## Predict XGB model
    pred = model.predict(X_test)
    test_rmse = np.sqrt(MSE(Y_test, pred))

    t1 = time.time()

    model_str = "ScikitLearn_XGboost"

    return [model_str,train_rmse,test_rmse,(t1-t0)]
```

```
def NN_model(data_df,y_var):

    t0 = time.time()

    # create training & testing data sets
    X,Y = data_df.loc[:,data_df.columns != y_var], data_df.loc[:,data_df.columns == y_var]
    X_train, X_test, Y_train, Y_test = train_test_split(X,Y,train_size=0.8,random_state=30)

    ## Create NN model structure
    model = Sequential()
    model.add(Dense(64,
                    activation='relu',
                    input_dim=X_train.shape[1]))
    model.add(Dense(64,
                    activation='relu'))
    model.add(Dense(1,
                    activation='linear'))
    model.compile(optimizer='rmsprop',
                  loss='mse',
                  metrics=['mse'])

    # fit NN model/architecture to data
    model.fit(X_train,
              Y_train,
              epochs=500,
              validation_split=0.2,
              verbose=0)

    # performance metrics
    epoch_mse = np.sqrt(np.min(model.history.history['val_mse']))
    epoch_pos = np.argmin(model.history.history['val_mse'])
    train_rmse = np.sqrt(model.history.history['mse'][epoch_pos])

    test_mse = model.evaluate(x=X_test,y=Y_test,verbose=0)
    test_rmse = np.sqrt(test_mse[0])

    t1 = time.time()

    model_str = "Tensorflow_NN"

    return [model_str,train_rmse,test_rmse,(t1-t0)]
```

ML Model Comparison

```
#data_test = MLdata_pdf[0:100]
data_test = MLdata_pdf
y_var = 'gross_revenue'

results = []
results.append(LinearRegression_predict(data_test,y_var))
results.append(xgb_predict(data_test,y_var))
results.append(NN_model(data_test,y_var))
```

	ML Method	Training_RMSE	Testing_RMSE	Time_Taken(s)
0	ScikitLearn_LinearRegression	6.911259e+07	5.398614e+07	2.181160
1	ScikitLearn_XGboost	1.085732e+08	8.450359e+07	6.005373
2	Tensorflow_NN	NaN	NaN	983.385372