

CLOUD COMPUTING PROJECTS

FA18

Gregor von Laszewski
Geoffrey C. Fox

laszewski@gmail.com

CLOUD COMPUTING PROJECTS

Gregor von Laszewski

(c) Gregor von Laszewski, 2018

CLOUD COMPUTING PROJECTS

1 Managing AWS Lambda Using REST API

- 1.1 Abstract
- 1.2 Introduction
- 1.3 Architecture
- 1.4 Implementation
- 1.5 Testing and Results
- 1.6 Recreating Project Environment
- 1.7 Technologies Used
- 1.8 Conclusion
- 1.9 Acknowledgement

2 Explore OpenFaaS Development and Deployment Aspects

- 2.1 Abstract
- 2.2 Introduction
 - 2.2.1 Micro-Services
 - 2.2.2 API Gateways
 - 2.2.3 Serverless
 - 2.2.4 OpenFaaS
- 2.3 Requirements
- 2.4 Design
 - 2.4.1 Neural Network
 - 2.4.2 Deep Neural Network
 - 2.4.3 Convolutional Neural Network (CNN)
- 2.5 Architecture
- 2.6 Dataset
- 2.7 Implementation
 - 2.7.1 Install Docker Swarm (Single-Node Cluster), Docker and OpenFaaS
 - 2.7.2 Trouble Shooting
 - 2.7.3 Build and deploy a serverless OpenFaaS function
 - 2.7.4 Deploying to AWS
 - 2.7.5 Deploying to Raspberry PI Clusters
 - 2.7.6 Project Files
- 2.8 Conclusion
- 2.9 Team Members and Work Breakdown

[2.10 Acknowledgement](#)

[3 HySDS on Kubernetes: ARIA InSAR Processing on XSEDE Jetstream](#)

[3.1 Abstract](#)

[3.2 Background](#)

[3.2.1 Architecture](#)

[3.2.2 Processing Algorithm Library](#)

[3.2.3 Pedigree](#)

[3.3 Requirements](#)

[3.4 Design](#)

[3.5 Architecture](#)

[3.5.1 Mozart](#)

[3.5.2 Metrics](#)

[3.5.3 GRQ](#)

[3.5.4 hysds-k8s](#)

[3.6 Dataset](#)

[3.6.1 Sentinel-1 SLCs](#)

[3.6.2 Sentinel-1 Orbits and Calibrations](#)

[3.6.3 DEMs](#)

[3.7 Implementation](#)

[3.7.1 Create a Kubernetes Cluster on IU Jetstream](#)

[3.7.2 Configuring OpenStack Cloud Provider](#)

[3.7.3 Create HySDS Buckets \(Swift Containers\)](#)

[3.7.4 Create the HySDS cluster](#)

[3.7.5 Register the lightweight-jobs and ariamh repositories in Jenkins](#)

[3.7.6 Submit Sentinel-1 Interferogram Jobs](#)

[3.8 Benchmark](#)

[3.9 Conclusion](#)

[3.10 Acknowledgement](#)

[4 Scalable Data Processing for Retail](#)

[4.1 Introduction](#)

[4.2 Dataset](#)

[4.2.1 Dataset.](#)

[4.3 Implementation](#)

[4.3.1 Hadoop](#)

[4.3.2 HDFS](#)

[4.3.3 Hive](#)

[4.3.4 Data Storage](#)

[4.3.5 API](#)

[4.4 Benchmark](#)

[4.4.1 Query and Results](#)

[4.5 Conclusion](#)

[5 Manage Files Across Cloud Providers](#)

[5.1 Abstract](#)

[5.2 Introduction](#)

[5.3 Requirements](#)

[5.4 Architecture](#)

[5.5 Design](#)

[5.6 Implementation](#)

[5.6.1 AWS access from Python:](#)

[5.6.2 Google Cloud Platform:](#)

[5.6.3 MongoEngine GridFS](#)

[5.7 Dataset](#)

[5.8 Conclusion](#)

[5.9 Acknowledgement](#)

[5.10 References](#)

[6 Cloudmesh GraphQL App](#)

[6.1 Abstract](#)

[6.2 Introduction](#)

[6.3 Requirements](#)

[6.4 Architecture](#)

[6.5 Dataset](#)

[6.6 Implementation](#)

[6.7 Summary](#)

[6.7.1 Sorting support](#)

[6.7.2 Indexing](#)

[6.8 Conclusion and future work](#)

[6.9 Acknowledgement](#)

[6.10 Workbreakdown](#)

[7 Open API with AWS EMR and Jupyter](#)

[7.1 Abstract](#)

[7.2 Introduction](#)

[7.2.1 Open API](#)

[7.2.2 AWS EMR](#)

[7.2.3 AWS EC2](#)

[7.2.4 AWS S3](#)

[7.2.5 JupyterHub](#)

[7.3 Implemenation](#)

[7.3.1 Setting up AWS CLI](#)

[7.3.2 Setting up AWS Admin Access](#)

[7.3.3 Creating and Configuring EC2 Instance to Host API](#)

[7.3.4 Create S3 Storage](#)

[7.3.5 Codegen Set Up](#)

[7.3.6 Building the EMR Rest Service](#)

[7.3.7 Deploy S3 Rest Service](#)

[7.3.8 Deploy Notebook Rest Service](#)

[7.3.9 Running all APIs](#)

[7.4 Conclusion](#)

[8 Morphological Image-based Profiling of Skin Lesions for Scientific Community](#)

[8.1 Abstract](#)

[8.2 Introduction](#)

[8.3 Requirements](#)

[8.4 Design](#)

[8.5 Architecture](#)

[8.6 Installation](#)

[8.7 Dataset](#)

[8.8 Implementation](#)

[8.9 Benchmark](#)

[8.10 Conclusion](#)

[8.11 Acknowledgement](#)

[9 Orchestrating Microservices for a Credit Scoring Application in Kafka](#)

[9.1 Abstract](#)

[9.2 Introduction](#)

[9.3 Apache Kafka](#)

[9.4 Requirements](#)

[9.5 Architecture](#)

[9.6 Design](#)

[9.7 Dataset](#)

[9.8 Implementation](#)

[9.9 Conclusion](#)

[9.10 Acknowledgement](#)

[10 Apache NiFi](#)

[10.1 Apache NiFi Introduction](#)

[10.2 NiFi History](#)

[10.3 NiFi Features](#)

[10.4 NiFi Architecture](#)

[10.4.1 Web Server](#)

[10.4.2 Flow Controller](#)

[10.4.3 FlowFile Repository](#)

[10.4.4 Content Repository](#)

[10.4.5 Provenance Repository](#)

[10.4.6 Processors](#)

[10.4.7 NiFi Clusters](#)

[10.5 Install NiFi](#)

[10.5.1 Apache NiFi - Windows](#)

[10.6 Building a NiFi Flow](#)

[10.7 Linking NiFi Flow to Apache Kafka](#)

[10.8 Conclusion](#)

[11 Historical Storm Data Analysis with Cosmos DB](#)

[11.1 Abstract](#)

[11.2 Introduction](#)

[11.3 Implementation](#)

[11.3.1 Data set](#)

[11.3.2 Related Work](#)

[11.4 Technologies Used](#)

[11.5 Data Visualization](#)

[11.6 Machine Learning](#)

[11.6.1 Feature Engineering](#)

[11.6.2 Cross Validation](#)

[11.6.3 Multinomial Naive Bayes](#)

[11.6.4 Logistic Regression](#)

[11.6.5 Random Forest](#)

[11.6.6 Support Vector Machines \(SVM\)](#)

[11.6.7 SVM with Stochastic Gradient Descent \(SGD\)](#)

[11.6.8 XGBoost](#)

[11.7 Summary](#)

[11.8 Future Work](#)

[11.9 Acknowledgement](#)

[11.10 Work Breakdown](#)

[12 Kickstarter Projects Analysis](#)

[12.1 Abstract](#)

[12.2 Introduction](#)

[12.3 Literature review](#)

[12.4 Dataset Description](#)

[12.4.1 Kaggle API](#)

[12.4.2 Kickstarter Projects Dataset](#)

[12.5 Design and Methods](#)

[12.6 Technologies](#)

[12.6.1 Technologies and Tools Used](#)

[12.7 Code Organization](#)

[12.7.1 bin](#)

[12.8 Architecture](#)

[12.9 Observations and Visualizations](#)

[12.9.1 Exploratory Analysis](#)

[12.9.2 Time Series Analysis](#)

[12.9.3 Logistic Regression](#)

[12.9.4 MongoDB Queries](#)

[12.10 Conclusion](#)

[12.11 Acknowledgement](#)

[12.12 Workbreakdown](#)

[12.12.1 Nishad Tupe](#)

[12.12.2 Vishal Bhoyar](#)

[12.12.3 Izolda Fetko](#)

[12.13 Nishad Tupe, Vishal Bhoyar, Izolda Fetko](#)

[13 Twitter Text Mining with Python and MongoDB](#)

[13.1 Abstract](#)

[13.2 Introduction](#)

[13.3 Twitter API and Python Implementation](#)

[13.4 Data](#)

[13.5 Twitter Cloud Storage](#)

[13.6 Data Science Algorithms for Twitter Data](#)

[13.7 Running a Twitter Script in Python](#)

[13.7.1 Python Libraries](#)

[13.7.2 Visualizations](#)

[13.7.3 Machine Learning Algorithms](#)

[13.8 Conclusion](#)

[13.9 Future Research](#)

[13.10 Acknowledgements](#)

[14 Predict EPL results using Tweets](#)

[14.1 Abstract](#)

[14.2 Introduction](#)

[14.3 Related Work](#)

[14.4 Implementation](#)

[14.4.1 Data](#)

[14.4.2 MongoDB](#)

[14.4.3 Tweet Extraction](#)

[14.4.4 Natural Language Processing](#)

[14.4.5 Machine Learning Approaches](#)

[14.5 Results](#)

[14.6 Conclusion and future work](#)

[14.7 Acknowledgement](#)

[14.8 Work Breakdown](#)

[15 Data analysis Of Yelp reviews](#)

[15.1 Abstract](#)

[15.2 Introduction](#)

[15.3 Literature review](#)

[15.4 Dataset](#)

[15.5 Data processing](#)

[15.6 Analysis methods](#)

[15.7 Results](#)

[15.8 Discussion](#)

[15.9 Conclusions](#)

[15.10 Project members and Work Breakdown](#)

[16 Do I buy or sell? - Using Big Data to predict Stock Performance](#)

[16.1 Abstract](#)

[16.2 Introduction](#)

[16.3 Implementation](#)

[16.3.1 Data](#)

[16.3.2 Design](#)

[16.4 Results](#)

[16.5 Conclusion](#)

[16.6 Acknowledgements](#)

[17 Loan Defaulter Analysis](#)

[17.1 Abstract](#)

[17.2 Introduction](#)

[17.3 Related work](#)

[17.4 Data](#)

[17.5 Technologies used](#)

[17.6 Exploratory Data Analysis](#)

[17.7 Feature Preprocessing](#)

[17.8 Feature Engineering](#)

[17.9 Modeling Techniques](#)

[17.9.1 Random Forest](#)

[17.9.2 K nn](#)

[17.9.3 XGBOOST](#)

[17.9.4 Model 1](#)

[17.9.5 Model 2](#)

[17.9.6 Model 3](#)

[17.9.7 Model 4](#)

[17.9.8 Model 5](#)

[17.9.9 Model 6](#)

[17.10 Conclusion](#)

[17.11 Acknowledgment](#)

[18 Utilizing Machine Learning to Predict Breast Cancer Tumor Diagnosis](#)

[18.1 Abstract](#)

[18.2 Keywords](#)

[18.3 Introduction](#)

[18.3.1 Dataset](#)

[18.3.2 Xcyt](#)

[18.4 Implementation](#)

[18.4.1 Requirements](#)

[18.4.2 MongoDB Atlas Database](#)

[18.5 Results](#)

[18.5.1 Visualizaton](#)

[18.5.2 K-Means Clustering](#)

[18.5.3 Clustering Assessment](#)

[18.5.4 Benchmarks](#)

[18.6 Limitations](#)

[18.7 Conclusion](#)

19 Credit Scoring Algorithm and its Implementation in Production Environment

19.1 Abstract

19.2 Keywords

19.3 Introduction

19.4 Design

19.4.1 Dataset

19.4.2 Data Visualization

19.4.3 Data Preparation

19.4.4 Model Training

19.4.5 Result Comparison

19.5 Implementation

19.5.1 Technologies Used

19.5.2 Prerequisites

19.5.3 Project Code Structure and Components

19.5.4 Code Running Instruction

19.6 Results

19.6.1 Deployment Benchmarks

19.6.2 Application Benchmarks

19.7 Limitations

19.8 Conclusion

19.9 Acknowledgements

19.10 References

20 OCR Extraction Implementation with Tesseract

20.1 Abstract

20.2 Introduction

20.3 Overview of Optical Character Recognition

20.4 Context Based Extraction Engine

20.4.1 Image Thresholding

20.4.2 OCR Process

20.4.3 Transform HOGR Data

20.4.4 Define Candidates

20.4.5 Set Context

20.4.6 Group Context

20.4.7 Score Context

20.4.8 Output Result

20.5 Example

- [20.5.1 Tkinter GUI](#)
- [20.5.2 Sample Images](#)
- [20.5.3 Sample Configuration - Invoice Number](#)
- [20.5.4 Sample Configuration - Total Amount](#)
- [20.6 Tools and Technology](#)
 - [20.6.1 Installation](#)
 - [20.6.2 Terresact](#)
 - [20.6.3 Beautiful Soup](#)
 - [20.6.4 FuzzyWuzzy](#)
 - [20.6.5 Python](#)
 - [20.6.6 Numpy](#)
 - [20.6.7 OpenCV](#)
 - [20.6.8 Python Imaging Library](#)
 - [20.6.9 Tkinter](#)
- [20.7 Conclusion](#)
- [20.8 Acknowledgement](#)
- [21 Automation on Drug Interactions Profiling](#)
 - [21.1 Introduction](#)
 - [21.2 Dataset](#)
 - [21.3 Implementation](#)
 - [21.3.1 Code Structure](#)
 - [21.3.2 Technologies Used](#)
 - [21.3.3 Prerequisites](#)
 - [21.3.4 Architecture](#)
 - [21.4 Results](#)
 - [21.4.1 Data Visualization](#)
 - [21.4.2 Discussion](#)
 - [21.4.3 Clinically Relevant Findings](#)
 - [21.5 Conclusion](#)
 - [21.6 Acknowledgments](#)
 - [21.7 Work Breakdown](#)
 - [21.8 Appendix](#)
 - [21.8.1 Chameleon Cloud](#)
 - [21.8.2 Amazon Web Services \(AWS\) EC2 Usage](#)
 - [21.8.3 Microsoft Azure Cloud Shell Usage](#)
- [22 Scalable Microservices to Label yelp images using Kuberenets](#)
 - [22.1 Abstract](#)

[22.2 Introduction](#)

[22.3 Implementation](#)

[22.3.1 Yelp Dataset](#)

[22.3.2 Cloud Vision API](#)

[22.3.3 Docker](#)

[22.3.4 Kubernetes](#)

[22.3.5 Apache Kafka](#)

[22.4 Design](#)

[22.4.1 Initial Setup](#)

[22.4.2 Frontend Microservices](#)

[22.4.3 Kafka Message Broker](#)

[22.4.4 Backend Microservice](#)

[22.5 Results](#)

[22.6 Benchmark](#)

[22.7 Conclusion](#)

[22.8 Acknowledgement](#)

[23 New Approaches to Managing Metadata at Scale in Research Libraries](#)

[23.1 Abstract](#)

[23.2 Introduction](#)

[23.3 New Approaches to Metadata Management](#)

[23.4 Blockchains for Research Libraries](#)

[23.5 Design Requirements](#)

[23.6 Scope](#)

[23.7 BigchainDB](#)

[23.7.1 Evolution](#)

[23.7.2 Benchmark](#)

[23.7.3 Architecture](#)

[23.8 Dataset](#)

[23.9 Implementation](#)

[23.9.1 Use Case](#)

[23.9.2 Installation](#)

[23.9.3 Data Management](#)

[23.9.4 Role-Based Access Control in BigchainDB](#)

[23.10 Conclusion](#)

[23.11 Artifacts Developed by Author](#)

[23.12 Acknowledgment](#)

[24 SciKit Learn Algorithms with Rest API](#)

- [24.1 Abstract](#)
 - [24.2 Introduction](#)
 - [24.3 Scope of work](#)
 - [24.4 Reason](#)
 - [24.5 Technology Stack](#)
 - [24.6 Dataset Details](#)
 - [24.7 Data Visualization](#)
 - [24.8 Data Exploration](#)
 - [24.9 Data Preprocessing](#)
 - [24.10 Azure ML Studio](#)
 - [24.11 Train Model with Azure](#)
 - [24.12 Predictive Model](#)
 - [24.13 Web Service Deployment](#)
 - [24.14 Azure Cloud deployment](#)
 - [24.15 Drawback of using Azure ML Studio](#)
 - [24.16 Azure with Notebook Instance](#)
 - [24.16.1 Code](#)
 - [24.17 AWS with Notebook Instance](#)
 - [24.17.1 Code](#)
 - [24.18 LocalMachine Instance](#)
 - [24.18.1 Local Execution - Prediction Time](#)
 - [24.19 Docker](#)
 - [24.20 Code Reproducing steps](#)
 - [24.21 Performance Comparison](#)
 - [24.22 Conclusion](#)
 - [24.23 Appendix](#)
 - [24.24 Acknowledgement](#)
 - [24.25 Workbreakdown](#)
- [25 Visualizing Global Commodity Statistics using Azure, PySpark and Python](#)
-

- [25.1 Abstract](#)
- [25.2 Introduction](#)
- [25.3 Implementation](#)
 - [25.3.1 Data](#)
 - [25.3.2 Related work](#)
- [25.4 Technologies used](#)
- [25.5 Visualizations](#)

[25.6 Summary](#)

[25.7 Future Work](#)

[25.8 Acknowledgement](#)

[25.9 Work breakdown](#)

[26 Big Data in Education](#)

[26.1 Abstract](#)

[26.2 Forms of Big Data in Education institution](#)

[26.2.1 Administrative Data](#)

[26.3 The Value of Systematic, Real-Time Data](#)

[26.4 Learning Analytics](#)

[26.5 Requirements](#)

[26.5.1 Prediction](#)

[26.5.2 Clustering](#)

[26.5.3 Relationship Mining](#)

[26.6 Architecture](#)

[26.6.1 Data Management](#)

[26.7 Implementation](#)

[26.7.1 Factors to Consider Before Implementing the Big Data](#)

[26.7.2 Steps of Successful implementation of the Big Data](#)

[26.8 Benchmarking](#)

[26.8.1 Planning](#)

[26.8.2 Generating Data](#)

[26.8.3 Generating Tests](#)

[26.8.4 Execution](#)

[26.9 Conclusion](#)

[27 Analyzing Big Data Sorting Algorithms](#)

[27.1 Abstract](#)

[27.2 Introduction](#)

[27.3 Requirements](#)

[27.4 Design](#)

[27.4.1 Bubble Sort](#)

[27.4.2 Merge Sort](#)

[27.4.3 Insertion Sort](#)

[27.4.4 Shell Sort](#)

[27.4.5 Selection Sort](#)

[27.4.6 Strand Sort](#)

[27.4.7 Python Sort \(Timsort\)](#)

[27.4.8 Heap Sort](#)

[27.5 Architecture](#)

[27.6 Dataset](#)

[27.7 Implementation](#)

[27.8 Limitations](#)

[27.9 Benchmark](#)

[27.10 Conclusion](#)

[27.11 Acknowledgements](#)

1 MANAGING AWS LAMBDA USING REST API

Varun Joshi

vajoshi@iu.edu

Indiana University

hid: fa18-516-08

github: [blue](#)

Learning Objectives

- Understand RESTful APIs and explore basic CRUD operations
- Learn about OpenAPI documentation using Swagger 2.0
- Apply REST understanding to Amazon's FaaS offering - AWS Lambda
- Learn AWS SDK for Python - boto3
- Use boto3 and Python Flask framework for building RESTful API with AWS Lambda as the resource

Keywords: FaaS, AWS Lambda,serverless,REST,OpenAPI example,Swagger

1.1 ABSTRACT

Amazon's Function as Service offering, AWS Lambda, provides serverless computing and eliminates the overhead of provisioning and managing servers. AWS Lambda can also integrate with other AWS services like S3 and Dynamo DB, extending it's capabilities for building highly scalable applications. AWS Lambda can be triggered by other AWS resource events, HTTP endpoints, mobile applications etc. giving the flexibility for serverless computing. This project explores managing AWS Lambda by providing a solution for basic CRUD operations through REST API build using OpenAPI (Swagger 2.0) specification.

1.2 INTRODUCTION

The goal of this project is to build a solution utilizing REST APIs to manage AWS Lambda service. The solution focuses on providing basic REST CRUD operations with AWS Lambda as resource. Python and Python flask framework is used for constructing REST service and AWS SDK for Python (boto3) is used for managing AWS Lambda.

1.3 ARCHITECTURE

The project is build of three components (see Figure 1).

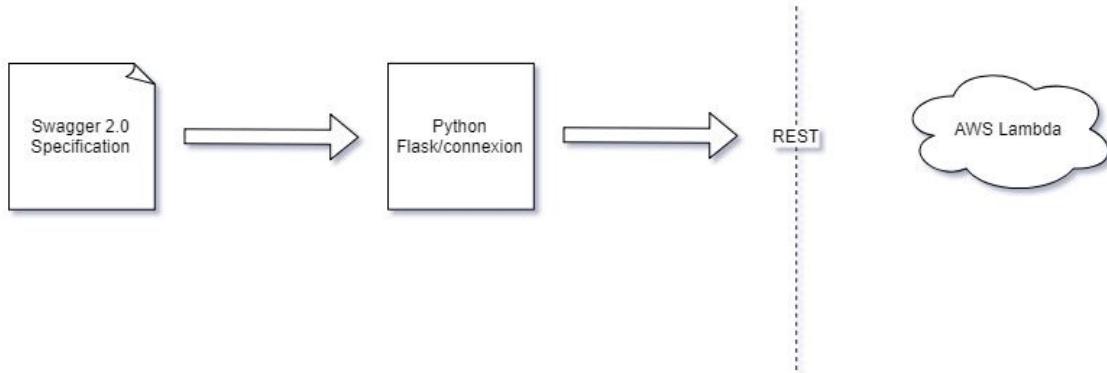


Figure 1: Project Architecture

- Swagger 2.0 is used for writing API specification. The specification describes endpoints for AWS Lambda CRUD operations and defines operation for each endpoint.
- Python flask framework consumes the OpenAPI specification and directs the endpoints to Python functions by building a RESTful app.
- AWS SDK for Python (boto3) is used to define Python functions which operate on endpoints and expose the resource, which is AWS Lambda , over REST API.

1.4 IMPLEMENTATION

Refer to the architecture (see Figure 1). I have enhanced the figure to include implementation details (see Figure 2).

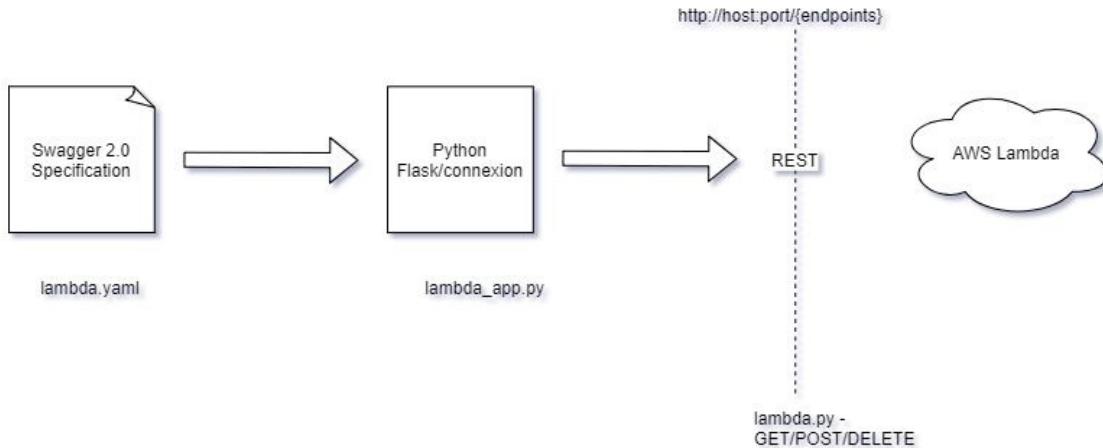


Figure 2: Project Architecture With Implementation

- OpenAPI specification (Swagger 2.0) :
 - /project-code/lambda.yaml : Defines the endpoints for CRUD operations on AWS lambda
- Python flask framework :
 - /project-code/lambda_app.py : Creates a connexion app which uses Flask under the hood. Consumes OpenAPI specification (Swagger 2.0) file lambda.yaml and routes the endpoints defined in the specification to the respective Python functions. The app is exposed as REST API running on local host and port 8080.
- Python program for building endpoints :
 - /project-code/lambda.py : Defines multiple Python functions utilizing boto3 which is AWS SDK for Python. Each function builds the endpoints to support REST operations.
- Other files :
 - /project-code/create_lambda_basic_exec_role.py : This Python program generates a basic AWS Lambda execution role which is attached to the newly created AWS Lambda function. To create an AWS Lambda function with an enhanced role such as accessing, the policy attribute in the program can be changed to the desired policy.
 - /project-code/config.yaml : This is a Python configuration file in YAML format. It holds reusable variables used across the Python programs in this project.
 - /project-code/requirements.txt : File with Python package and

- libraries dependency to be installed using pip command.
- /project-code/test/hello.zip : Sample AWS Lambda deployment package zip file which holds a Python program for building simple AWS Lambda Function as a Service.
 - /project-code/test/fun.json : Body parameters for REST POST operation for creating AWS Lambda. The body parameter consumes json content. The format is: { “cfile”: “hello.py”, “handler”: “lambda_handler”, “pkg”: “hello.zip”, “renv”: “python3.7” } The key values are required and are explained as below:
 - “cfile” : The name of the code file for the application which is used by AWS Lambda for processing the functionality of the application. This project uses simple Hello World Python program as the code file.
 - “handler” : Name of the handler or function inside the code file which is invoked by AWS Lambda. In this project lambda_handler is the name of the Python function inside the code file hello.py.
 - “pkg” : The name of the zip file which has code file as well as all dependencies required for the code file. In this project this zip file is physically present in the directory /project-code/test.
 - “renv” : Runtime environment in AWS Lambda for the code file. This project uses Python 3.7 runtime environment provided by AWS Lambda.
 - /project-code/shell:
 - setup.sh : sets the project environment by cloning this GIT repository
 - runAPI.sh : Runs the /project-code/lambda_app.py to start REST service
 - testAPI.sh : basic GET/POST/DELETE curl commands to test REST service for managing AWS Lambda
 - clean.sh : Deletes the project environment

1.5 TESTING AND RESULTS

In the /project-code directory, run the Python program lambda_app.py: python lambda_app.py

REST service will start on port 8080 (see Figure 3)

```
* Serving Flask app "lambda_app" (lazy loading)
* Environment: production
  WARNING: Do not use the development server in a production environment.
  Use a production WSGI server instead.
* Debug mode: on
* Running on http://0.0.0.0:8080/ (Press CTRL+C to quit)
* Restarting with stat
* Debugger is active!
* Debugger PIN: 334-271-300
```

Figure 3: Start REST Service

Once the REST service has started , bring up the Swagger UI (see +????: SwaggerUI).curl can also be used in the command line to test the REST service)

Type http://0.0.0.0:8080/lambda/ui/ on a web browser to open Swagger UI:

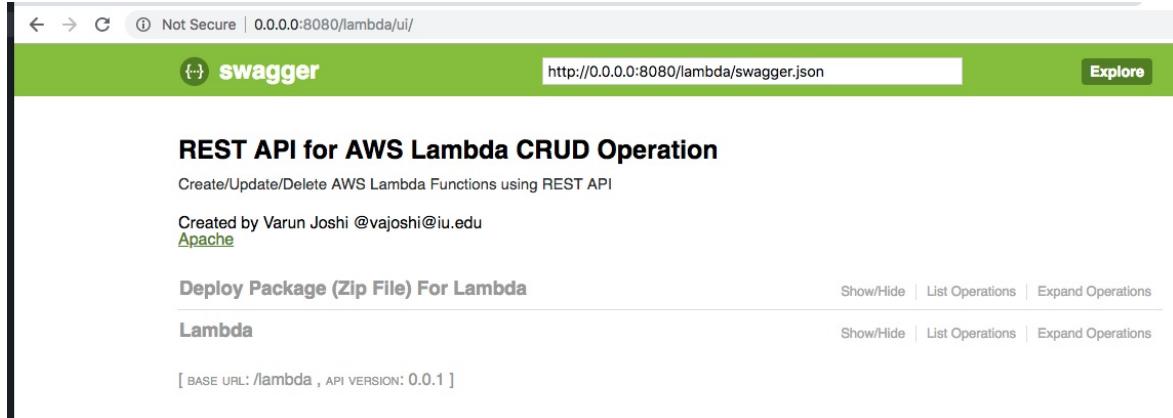


Figure 4: Open Swagger UI

For the LAMBDA tag in the Swagger UI, click “List Operations”. All available REST operations for AWS Lambda resource will be displayed (see Figure 5).

REST API for AWS Lambda CRUD Operation

Create/Update/Delete AWS Lambda Functions using REST API

Created by Varun Joshi @vajoshi@iu.edu
Apache

Deploy Package (Zip File) For Lambda		Show/Hide List Operations Expand Operations
Lambda		Show/Hide List Operations Expand Operations
DELETE	/function/{fname}	Delete AWS Lambda Function by name
GET	/function/{fname}	Get AWS Lambda function configuration by name
POST	/function/{fname}	Create AWS Lambda Function
PUT	/function/{fname}	Update AWS Lambda Function
GET	/functions	Get name of all AWS Lambda Functions

[BASE URL: /lambda , API VERSION: 0.0.1]

Figure 5: Expand operations for TAG Lambda

Test each operation: * GET all AWS Lambda functions: Click on GET /function in the Swagger UI and then click “Try it out!” (see Figure 6).

The screenshot shows the Swagger UI interface for the AWS Lambda API. The top navigation bar has 'GET' selected for the method and '/functions' for the path. To the right, the description reads 'Get name of all AWS Lambda Functions'. Below this, there's a section for 'Implementation Notes' stating 'lists all available Lambda Functions (limit 50)'. Under 'Response Class (Status 200)', it says 'Lambda Functions'. There are tabs for 'Model' and 'Example Value'. The 'Example Value' tab is active, displaying a JSON object: { "result": "string" }. At the bottom, the 'Response Content Type' is set to 'application/json' with a dropdown arrow, and a 'Try it out!' button is visible.

Figure 6: GET All Lambda Functions

Result for GET functions (see +????).

Curl

```
curl -X GET --header 'Accept: application/json' 'http://0.0.0.0:8080/lambda/functions'
```

Request URL

```
http://0.0.0.0:8080/lambda/functions
```

Response Body

```
[
  {
    "CodeSha256": "0MZhXAbLbLVXxeSu3CCZcNqUtH7f+Fc1i27WFEz22Ho=",
    "CodeSize": 300,
    "Description": "",
    "FunctionArn": "arn:aws:lambda:us-east-2:249033861349:function:testDeploy1",
    "FunctionName": "testDeploy1",
    "Handler": "hello.lambda_handler",
    "LastModified": "2018-11-26T08:20:06.463+0000",
    "MemorySize": 128,
    "RevisionId": "a107d513-c312-4092-999f-193dd82440bb",
    "Role": "arn:aws:iam::249033861349:role/Lambda_Basic_Exec2",
    "Runtime": "python3.7",
    "Timeout": 3,
    "TracingConfig": {
      "Mode": "PassThrough"
    },
    "Version": "$LATEST"
  }
]
```

{#fig: GETFunctionsResult}

- GET function by name: Click on GET /function/{fname} in the Swagger UI and then type the function name to get and click “Try it out!” (see +???: GETFunctionByName).

The screenshot shows the AWS Lambda Swagger API documentation for the GET /function/{fname} endpoint. The top navigation bar includes 'GET /function/{fname}' and 'Get AWS Lambda function configuration by name'. Below this, the 'Response Class (Status 200)' section is shown with a 'Function Search' input field. A JSON example is provided:

```
{
  "result": "string"
}
```

The 'Response Content Type' is set to 'application/json'. The 'Parameters' table has one row for 'fname' with a value of 'SearchText'. A 'Try it out!' button is located at the bottom left.

Parameter	Value	Description	Parameter Type	Data Type
fname	SearchText		path	string

Figure 7: GET Function by name

Result for GET function by name (see Figure 8).

The screenshot shows the AWS Lambda console interface. At the top, there's a 'Curl' button followed by a command line snippet:

```
curl -X GET --header 'Accept: application/json' 'http://0.0.0.0:8080/lambda/function/SearchText'
```

Below that is a 'Request URL' field containing:

```
http://0.0.0.0:8080/lambda/function/SearchText
```

Then there's a 'Response Body' field containing a JSON object:

```
{  
    "CodeSha256": "08xi3XvJvep9iiDnfwccrjVycbzRpus5uu0edXVVz4Y=",  
    "CodeSize": 680,  
    "Description": "",  
    "FunctionArn": "arn:aws:lambda:us-east-2:249033861349:function:SearchText",  
    "FunctionName": "SearchText",  
    "Handler": "lambda_function.search_text",  
    "LastModified": "2018-11-22T23:12:06.324+0000",  
    "MemorySize": 128,  
    "RevisionId": "dbfe2b20-f6cf-45e3-b7bf-13c7e0f2e684",  
    "Role": "arn:aws:iam::249033861349:role/lambda_read_s3",  
    "Runtime": "python3.7",  
    "Timeout": 3,  
    "TracingConfig": {  
        "Mode": "PassThrough"  
    },  
    "Version": "$LATEST",  
    "VpcConfig": {  
        "SecurityGroupIds": [],  
        "SubnetIds": []  
    }  
}
```

Figure 8: GET function by name result

- POST a new function: Expand POST /function/{fname} , type in the new AWS Lambda function name to create and in the body parameter type in the json format for the required values or click the json under “Example Value” to auto pouplate the json in the body parameter. Click “Try it out!” (see Figure 9).

POST /function/{fname}

Create AWS Lambda Function

Response Class (Status 200)

Function Created

Model | Example Value

```
{
  "result": "string"
}
```

Response Content Type application/json

Parameters

Parameter	Value	Description	Parameter Type	Data Type
fname	NewFunc	Name for the AWS Lambda path Function which will be created		string
fprop	<pre>{ "cfile": "hello.py", "handler": "lambda_handler", "pkg": "hello.zip", "renv": "python3.7" }</pre>	Required parameters to create AWS Lambda Function -Name of the file containing Lambda code (cfile),Runtime env (renv), AWS Lambda handler name (handler), zip file name of the AWS Lambda Deployment package(pkg)	body	Model Example Value <pre>{ "cfile": "hello.py", "handler": "lambda_handler", "pkg": "hello.zip", "renv": "python3.7" }</pre>

Parameter content type: application/json

Try it out!

Figure 9: POST Function

Result for POST function (see Figure 10).

Curl

```
curl -X POST --header 'Content-Type: application/json' --header 'Accept: application/json' -d '{ \
  "cfile": "hello.py", \
  "handler": "lambda_handler", \
  "pkg": "hello.zip", \
  "renv": "python3.7" \
}' 'http://0.0.0.0:8080/lambda/function/NewFunc'
```

Request URL

http://0.0.0.0:8080/lambda/function/NewFunc

Response Body

```
{
  "CodeSha256": "FjB80CwWt0cLe5LEjeKBPk8xqTs4iTHUIKcw/ASg56I=",
  "CodeSize": 300,
  "Description": "",
  "FunctionArn": "arn:aws:lambda:us-east-2:249033861349:function:NewFunc",
  "FunctionName": "NewFunc",
  "Handler": "hello.lambda_handler",
  "LastModified": "2018-11-28T03:21:09.708+0000",
  "MemorySize": 128,
  "RevisionId": "ab9e6773-580c-44b0-8fed-70db52ce6138",
  "Role": "arn:aws:iam::249033861349:role/Lambda_Basic_Exec5",
  "Runtime": "python3.7",
  "Timeout": 3,
  "TracingConfig": {
    "Mode": "PassThrough"
  },
  "Version": "$LATEST"
}
```

Figure 10: POST function result

- Delete a function: Expand DELETE /function/{fname} in the Swagger UI. Type in the function name to delete and then click “Try it out!” (see Figure 11).

DELETE /function/{fname} Delete AWS Lambda Function by name

Response Class (Status 200)
Function Deleted

Model Example Value

```
{
  "result": "string"
}
```

Response Content Type application/json

Parameters

Parameter	Value	Description	Parameter Type	Data Type
fname	NewFunc		path	string

Try it out!

Figure 11: DELETE Function

Result for DELETE function (see Figure 12).

Parameter Value Description Parameter Type Data Type

fname	NewFunc		path	string
-------	---------	--	------	--------

Curl

```
curl -X DELETE --header 'Accept: application/json' 'http://0.0.0.0:8080/lambda/function/NewFunc'
```

Request URL

```
http://0.0.0.0:8080/lambda/function/NewFunc
```

Response Body

```
{"Message : Requested Function Deleted"}
```

Response Code

```
200
```

Response Headers

```
{
  "date": "Wed, 28 Nov 2018 03:27:03 GMT",
  "server": " Werkzeug/0.14.1 Python/3.7.0",
  "content-length": "41",
  "content-type": "application/json"
}
```

Figure 12: DELETE Function Result

1.6 RECREATING PROJECT ENVIRONMENT

To recreate the project environment in any Ubuntu machine, use the /project-code/shell/setup.sh

The setup.sh has following commands:

- git clone https://github.com/cloudmesh-community/fa18-516-08
- cd fa18-516-08/project-code
- sudo pip install -r requirements.txt

Then, open config.yaml to update your AWS access key id and AWS secret access key for the user id which has permission to create AWS services and resources.

To start REST service use /project-code/shell/runAPI.sh

- sudo python lambda_app.py

Commands to test REST are provided in /project-code/shell/testAPI.sh

- Testing using an AWS Lambda deployment package other than “hello.zip”:
 - If you would like to test the API with your own AWS Lambda deployment package, then do the following:
 - use the following curl command to upload the deployment package to the API server test directory which is part of the project environment setup. curl -X POST -F ‘pkg=??? path to your deployment package’ http://0.0.0.0:8080/lambda/deploy-pkg
 - To create new AWS Lambda based on your deployment package, use the appropriate body parameters for the POST operation in the curl command which match the required parameters as per your deployment package. Refer to project-code/test/fun.json
 - If you are deploying the API in a distributed client server model, the above method will be extremely useful for providing the deployment package to the API server to create a new AWS Lambda function. In the distributed setup, the curl

will look like this:

- curl -X POST -F 'pkg=??? path to your deployment package' http://APIHost:APIPort/lambda/deploy-pkg

1.7 TECHNOLOGIES USED

- OS : ubuntu 18.04
 - For project runtime environment and to host REST service.
- Programming Language : Python 3.7
 - For building REST service using Flask framework with connexion
 - Building functions for REST endpoints
- Cloud Provider : Amazon Web services
 - For exposing AWS Lambda, Amazon's FaaS offering , as REST endpoint for CRUD
 - boto3 - AWS SDK for Python
- API Specification : OpenAPI
 - Swagger 2.0 specification for building REST API
 - Swagger UI editor for testing REST operations

1.8 CONCLUSION

The goal of this project was to learn about OpenAPI documentation for writing REST API using Swagger 2.0 and applying this learning to explore REST CRUD operations for AWS Lambda Function as a Service. The end result of the project produced a nice OpenAPI (Swagger 2.0) documentation with AWS Lambda as endpoints. The endpoint operations were written using Python 3.7. The API documentation generated as part of this project can be reused to operate on Lambda endpoints utilizing any supported programming language and not just Python 3.7. The reusability of this project and demonstration of AWS Lambda exposed as REST endpoints is the biggest gain of this project learning exercise. The OpenAPI documentation can also be reused to expose more endpoints over REST which are similar to AWS Lambda such as AWS API Gateway, AWS S3 or AWS Dynamo DB.

1.9 ACKNOWLEDGEMENT

I would like to thank professor Gregor von Laszewski and associate instructors for the course ENGR 516 for providing their guidance and help in achieving the goal of this project learning.

2 EXPLORE OPENFAAS DEVELOPMENT AND DEPLOYMENT ASPECTS

Murali Cheruvu, Anand Sriramulu
mcheruvu@iu.edu, asriram@iu.edu

Indiana University
hid: fa18-516-11 fa18-516-23

github: [cloudnative](#)
code: [cloudnative](#)

Keywords: OpenFaaS, Serverless, Micro-Services, Function-as-a-service (FaaS)

2.1 ABSTRACT

Explore creating micro-services using OpenFaaS that makes serverless functions simple for containers like Docker and Kubernetes. Integrate OpenFaaS serverless functions into public cloud offerings such as, AWS and Azure, to make them even better.

2.2 INTRODUCTION

Goals of this project is to learn how to create cloud-native micro-services using server-less concepts for better scalability and cheaper to maintain. Function-as-a-service (FaaS) methodology allows to decouple each functionality from the rest of the application, for better support, isolated deployment and scalability at each function level. We will use OpenFaaS, an open source alternative framework to develop and deploy micro-services as FaaS in cloud-agnostic way. OpenFaaS has an opiniated way of developing FaaS and deploying them to achieve the required scalability without much depending on the infrastructure of the public cloud offerings, using container concepts wrapped around our FaaS functions. In the context of developing loosely couple components, we can interchangeably use micro-service for function-as-a-service (FaaS).

Let us introduce some of the key concepts that are related to the function-as-a-service.

2.2.1 Micro-Services

Micro-Services is a new paradigm in the software architecture to break down complex monolithic applications into more manageable and decoupled components that can be created and supported in silos. Loosely coupled components offer scalability and also we can use programming-of-choice based on the nature and complexity of the component, anywhere from advanced Object-Oriented Programming (OOP) languages like Java and C#.net, to modern functional-programming (FP) languages like Scala or Python. Deploying micro-services can be as flexible as deploying each individual functionality as a separate micro-service to grouping of related micro-services into one deployment package [1]. Micro-Services, targeting for web-based interfaces, can be implemented using simple REST-based API methodology.

2.2.2 API Gateways

Micro-Services offer flexibility and scalability but they bring complexity into the deployment and support. Too many micro-services can create confusion in discovering them and also, client applications may have to make multiple micro-service calls, hence create more network traffic even to populate a single webpage. As an example, Amazon uses lots of micro-services to display a typical product search result webpage. To reduce the network round-trips, it is advised, to create a gateway - API gateway, so that clients make unified calls to the gateway and all the related micro-services to fulfill a request will be made within the server and the results of these micro-services are bundled into one result, hence reduce the number of calls to make [2]. Cloud Offerings such as Microsoft Azure and AWS offer API Gateways with automatic API discovery and quota-based usage along with lots of DevOp tools for continuous monitoring the scalability, performance and health-check of the micro-services. OpenFaaS has built-in API Gateway and integrates well with continuous monitoring tools such as Grafana and Prometheus [3].

2.2.3 Serverless

Serverless is the new methodology that cloud providers such as AWS, Azure or Google Cloud brought to simplify the deployment, execution and support of micro-services. Cloud providers take care the responsibility of the deployment, execution of the code and supporting - monitoring, tracking and notifying errors, auto-scaling (up or down) and performance metrics. Application providers are responsible to develop in a cloud-native fashion and leave the rest of the responsibilities to the cloud providers. AWS Lambda, Azure Functions, Google Functions are serverless offerings that are available today with high scalability turned on and cheaper to host such serverless functions [4].

2.2.4 OpenFaaS

OpenFaaS (Functions as a Service) is a framework for creating micro-services that can be hosted in containers like Docker or Kubernetes and make these services ready to be served in a serverless fashion [5]. OpenFaaS can easily deployed into all the popular public cloud providers such as AWS, Azure and Google Cloud.

![6](images/open-faas.png){#fig:OpenFaaS}

2.3 REQUIREMENTS

This project has two goals - (1) How to deploy machine learning algorithm to production and make it to work as serverless function to get best scalability and (2) Explore OpenFaaS and related tools to develop, test and deploy onto public cloud providers such as Azure, AWS and Google Cloud.

High level requirements include: setting up OpenFaaS locally within the development environment in Windows and also create deployable aspects: containerized OpenFaaS to be able to deploy to public cloud offerings including AWS, Azure and Google Cloud. Create python based project exploring high level concepts of serverless/micro-services for web. Explore container features in the process.

- Development Environment: Windows 10 Enterprise
- Install Docker Community Edition and Docker Swarm with single-node cluster

- Setup developer account with Docker Hub for publishing Docker Images on the internet
- Install OpenFaaS CLI - command line interface for OpenFaaS
- Deploy OpenFaaS into the development environment to make it ready to use
- Install Python and related libraries to make it ready for the actual project for machine learning algorithms to run and also write OpenFaaS functions in python.
- Write Python based code for object (image) detection using Convolved Deep Neural Network (CNN) algorithms
- Train the model using around 20,000 images of dogs and cats provided by one of Kaggle competition [7].
- create OpenFaaS function to predict the uploaded image - whether it is a dog or cat
- create another OpenFaaS function to detect the uploaded image as what animal using pre-trained model by Keras library.

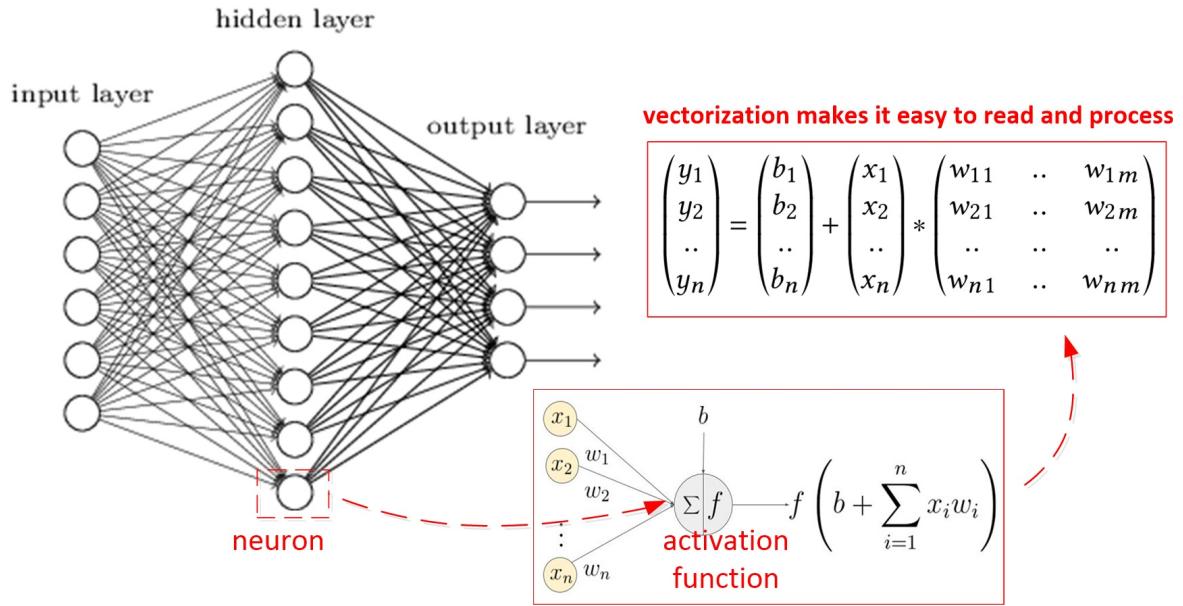
2.4 DESIGN

Create image object detection and classification model using Convolutional Neural Network by exploring very small number of training examples, from the dataset of 20,000 images of dogs and cats, using Python libraries such as Keras and Tensorflow. We will train the small neural network as a baseline and apply fine-tuning of an existing pre-trained network provided by popular VGGNet image network [8]. We will provide some background on the concepts of neural network to understand the project domain.

2.4.1 Neural Network

Neural Network is modeled after the human brain, specifically the way it solves complex problems. Perceptron, the first generation neural network, created a simple mathematical model or a function, mimicking neuron - the basic unit of the brain, by taking several binary inputs and produced single binary output. Sigmoid Neuron improved learning by giving some weightage to the input based on importance of the corresponding input to the output so that tiny changes in the output due to the minor adjustments in the input weights (or biases) can be measured effectively. Neural Network is, a directed graph, organized by layers

and layers are created by number of interconnected neurons (or nodes). Every neuron in a layer is connected with all the neurons from the previous layer; there will be no interaction of neurons within a layer. As shown in Figure 13, a typical Neural Network contains three layers: input (left), hidden (middle) and output (right) [9]. The middle layer is called hidden only because the neurons of this layer are neither the input nor the output. However, the actual processing happens in the hidden layer as the data passes through layer by layer, each neuron acts as an activation function to process the input. The performance of a Neural Network is measured using cost or error function and the dependent input weight variables. Forward-propagation and backpropagation are two techniques, neural network uses repeatedly until all the input variables are adjusted or calibrated to predict accurate output. During, forward-propagation, information moves in forward direction and passes through all the layers by applying certain weights to the input parameters. Back-propagation method minimizes the error in the weights by applying an algorithm called gradient descent at each iteration step.



An example of a neuron showing the input ($x_1 - x_n$), their corresponding weights ($w_1 - w_n$), a bias (b) and the activation function f applied to the weighted sum of the inputs.

Figure 13: Neural Network [9]

2.4.2 Deep Neural Network

Deep Learning is an advanced neural network, with multiple hidden layers (thousands or even more deep), that can work well with supervised (labeled) and unsupervised (unlabeled) datasets. Applications, such as speech, image and behavior patterns, having complex relationships in large-set of attributes, are best suited for Deep Learning Neural Networks. Deep Learning vectorizes the input and converts it into output vector space by decomposing complex geometric and polynomial equations into a series of simple transformations. These transformations go through neuron activation functions at each layer parameterized by input weights. For it to be effective, the cost function of the neural network must guarantee two mathematical properties: continuity and differentiability.

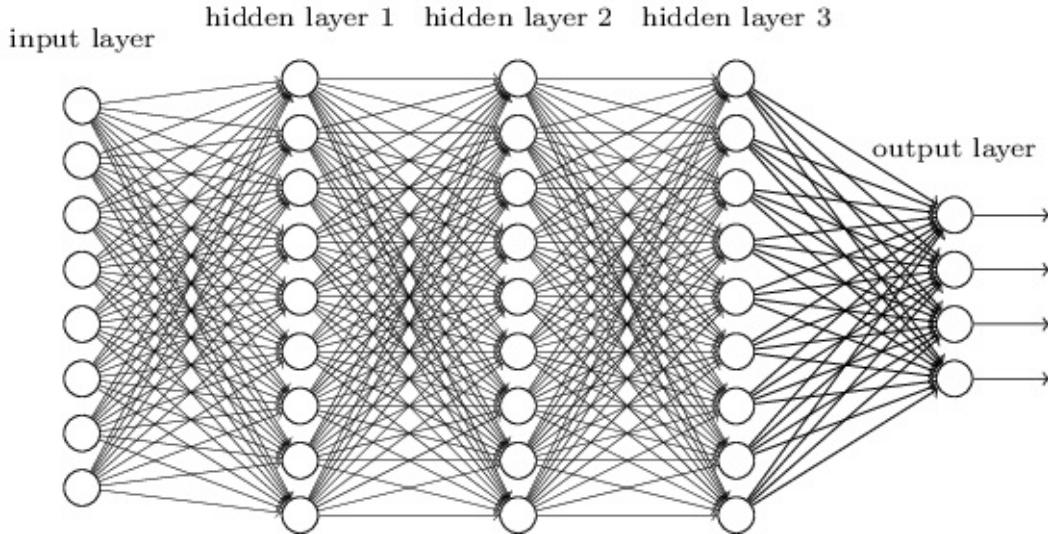


Figure 14: Deep Neural Network [9]

2.4.3 Convolutional Neural Network (CNN)

Convolutional Neural Network (CNN), also called multilayer perceptron (MLP), is a deep feedforward network, consists of (1) convolutional layers - to identify the features using weights and biases, followed by (2) fully connected layers - where each neuron is connected from all the neurons of previous layers - to provide nonlinearity, sub-sampling or max-pooling, performance and control data overfitting [10]. CNN is used in image and voice recognition applications by effectively using multiples copies of same neuron and reusing group of neurons in several places to make them modular. CNNs are constrained by fixed-size vectorized inputs and outputs.

Convolution Neural Network has two key components: (a) feature extraction - in this component, the network performs a series of convolutions (mathematical operation) and pooling operations to create the feature-maps, the list of features from the images. (b) classification - fully connected layers will serve as a classifier on top of these extracted features. They will assign probability for the object on the image being what the algorithm predicts it is.

- Softmax Layer: Classifier

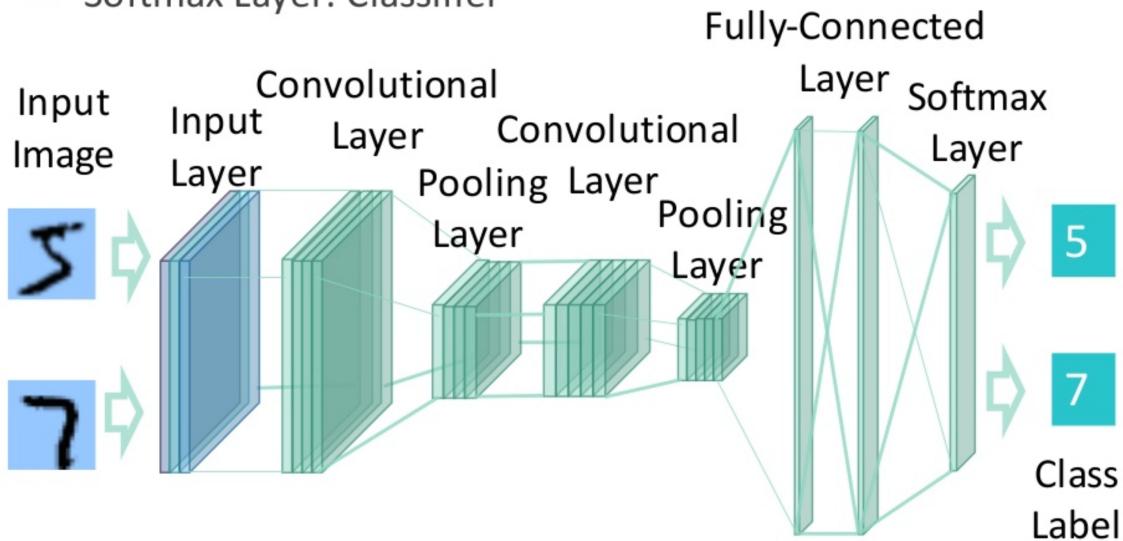


Figure 15: Convolutional Neural Network (CNN) [11]

2.5 ARCHITECTURE

Computer vision is an interdisciplinary field that deals with how computers can be made for gaining high-level understanding from digital images or videos. Images are treated as a matrix of pixel values. By applying convolutional, mathematical operation, features such as edges, brightness or blur can be extracted as feature maps from the images. This process goes through series of convolutions followed by pooling to reduce the size of the images for further processing. In the end, various features of the image are flattened into vector and create deep neural network to classify the image, in our case, whether it is a dog or cat. VGGNet is very popular image network that reduced the errors in the image classification and improved the processing performance from the predecessor image network models. We will create our based neural network model using a small sample dataset of 2000 images from the given 20,000 images and apply this on top of the pre-trained model that is optimized based on VGGNet algorithm. So that our effort is incremental and minimal.

2.6 DATASET

About 20,000 images of dogs and cats are provided part of the Kaggle competition. Dataset can be downloaded from [HERE](#).

2.7 IMPLEMENTATION

2.7.1 Install Docker Swarm (Single-Node Cluster), Docker and OpenFaaS

- Prerequisites: Windows 10 Professional or Enterprise Edition, open the command prompt in Administrator mode
- Step 1: Install Docker [Community Edition](#)
- Step 2: Install Git Bash for pulling the latest OpenFaaS artifacts and all the other software from GitHub
- Step 3: Run docker swarm init to set up the single-node docker swarm cluster
- Step 4: Create an account with Docker Hub, if the created docker images to be shared with others through internet
- Step 5: Run docker login to make sure docker is linked to your account
- Step 6: Download latest faas-cli.exe from [HERE](#)
- Step 7: Copy the faas-cli.exe to *C:folder to make it available for the command prompt. Or you will need to add the path of the faas-cli.exe into the system environment variables
- Step 8: Test the faas-cli using the command - faas-cli version
- Step 9: Clone the OpenFaaS artifacts from GitHub using : git clone <https://github.com/openfaas/faas>
- Step 10: Go into the faas folder that to checkout the git master repository - cd faas and git checkout master
- Step 11: Run deploy_stack.sh –no-auth to deploy the latest OpenFaaS into our environment
- Step 12: Run docker service ls to verify whether openfaas has been deployed to our environment

2.7.2 Trouble Shooting

If there are any issues with the docker and/or OpenFaaS functions, we can reset the environment using the following commands

- restart docker - to restart the docker
- docker stack rm func - to remove all the functions
- docker swarm leave –force - to shutdown the docker cluster

- docker swarm init - to initialize docker swarm cluster
- {open_faas.github.folder}/deploy_stack.sh - to pull the latest code from openfaas GitHub

2.7.3 Build and deploy a serverless OpenFaaS function

2.7.3.1 Get FaaS-CLI

```
curl -sSL https://cli.openfaas.com | sudo sh
```

2.7.3.2 Build, deploy and push to Docker Hub

```
$ cd fa18-516-11/project-code
$ docker build -t faas-ressnet .
$ faas-cli deploy --image faas-ressnet --name faas-ressnet
$ docker tag faas-ressnet $anandid/faas-ressnet
$ docker push $anandid/faas-ressnet
```

2.7.3.3 Testing OpenFaaS function

2.7.3.3.1 Test Request : 1



faas - OpenFaaS - tiger

```
Input:
curl -X POST -H \
--data-binary @data/tiger.png \
"http://127.0.0.1:8080/function/faas-resnet"

Output:
Predicted: [('n02129604', 'tiger', 0.92411584), ('n02123159', 'tiger_cat', 0.04635064),
('n02391049', 'zebra', 0.017654872)]
```

2.7.3.3.2 Test Request : 2



faas - OpenFaas - cow

Input:

```
curl -X POST -H \
--data-binary @data/cow.jpg \
"http://127.0.0.1:8080/function/faas-resnet"
```

Output:

```
Predicted: [('n02403003', 'ox', 0.55445725), ('n03868242', 'oxcart', 0.36393312),
('n02109047', 'Great_Dane', 0.035532992)]
```

2.7.4 Deploying to AWS

2.7.4.1 Setup AWS Instance

1. Purchased Spot Instance for Ubuntu 16.04, and with instance type we'll use m4.xlarge
2. Enabled Security group allowing ports 22, 31112, and 6443 for ingress
3. Created a key-pair file, so that we can SSH in to the instance
4. Test the Instance

```
ssh -i "faas.pem" ubuntu@ec2-18-191-176-209.us-east-2.compute.amazonaws.com
```

2.7.4.2 Setting up Kubernetes on AWS

1. Prep the machine by installing some necessary components. Run the following commands to enter superuser mode, install some necessary components from this gist, then exit back into the ubuntu user.

```
$ sudo su
$ curl -sSL
https://gist.githubusercontent.com/ericstoeckl/1d4372e9398d9cec7ec028629b2c36e2/raw/6f03cf3c
| sh
exit
```

2. Deploy Kubernetes

```
$ sudo kubeadm init --kubernetes-version stable-1.8
```

3. Networking layer for the cluster, to allow inter-pod communication

```
$ kubectl apply -f "https://cloud.weave.works/k8s/net?k8s-version=$(kubectl version | base64 | tr -d '\n')"
```

4. To Allow container placement on the master node and confirm the cluster is running

```
$ kubectl taint nodes --all node-role.kubernetes.io/master-
$ kubectl get all -n kube-system
```

2.7.4.3 Deploying OpenFaas on Kuberetes using faas-netes

1. Clone the node, Deploy the Whole Stack and deploy OpenFaas

```
$ git clone https://github.com/openfaas/faas-netes
$ kubectl apply -f https://raw.githubusercontent.com/openfaas/faas-
netes/master/namespaces.yml
$ cd faas-netes && \
kubectl apply -f ./yaml
```

2. Install the CLI, deploy samples

```
$ curl -sL https://cli.openfaas.com | sudo sh
$ git clone https://github.com/openfaas/faas-cli
```

3. Pull docker image (OpenFaas functions)

```
docker pull anandid:faas-resnet
```

4. Deploy OpenFaas Functions

```
faas-cli deploy --image anandid/faas-resnet --name faas-resnet --gateway
http://18.191.176.209:31112
```

5. Test OpenFaas function

```
curl http://18.191.176.209:31112/function/faas-resnet --data-binary @data/tiger.jpg
```

2.7.5 Deploying to Raspberry PI Clusters



PI Cluster Case [12]

2.7.5.1 Burn 3 Raspberry PI clusters thru cm-burn

```
$ git clone https://github.com/cloudmesh-community/cm-burn
$ cd cm-burn
$ python setup.py install
$ cmburn create --group g1 --names red[001-003] --key c:/users/anand/.ssh/id_rsa.pub --
image 2018-06-27-raspbian-stretch.img --bootdrive I --rootdrive G --domain 192.168.1.254 -
-ip 192.168.1.[111-113]``
```

2.7.5.2 Steps to setup OpenFass in Rasberry PI

1. Install Docker using the following utility script

```
$ curl -sSL https://get.docker.com | sh
```

Note: the above step can take between 2 to 5 minutes

2. Run the following command to use Docker as non-root user

```
$ sudo usermod pi -aG docker
```

3. Change default Password

```
$ sudo passwd pi
```

4. Logout and log back to take this effect for the above 2 steps

5. Setup docker swarm cluster

```
$ docker swarm init
```

Copy the output from the above command as like following, and need to be used in other PI nodes to join the cluster docker swarm join –token SWMTKN-1-25qnthaepgkcx9qhfov7yx0ht23od2shf44bw8tfphibsod8-b1c8o5ljetm0vjkamo3kk9k 192.168.1.111:2377

6. Setup OpenFaas

```
$ git clone https://github.com/alexellis/faas/
$ cd faas
```

7. Deploy sample functions

```
$ ./deploy_stack.armhf.sh
```

Other RPis will now be instructed by Docker Swarm to start pulling the Docker images from the internet and extracting them to the SD card. The work will be spread across all the RPis so that none of them are overworked.

8. After few minutes, the following command will provide the status of the functions

```
$ watch 'docker service ls'
```

9. Testing a function to see the scheduled RPI for this function

```
$ docker service ps func_markdown
```

10. The Openfaas functions can be access via

```
http://192.168.1.111:8080
```

11. Pull docker image (OpenFaas functions)

```
docker pull anandid/faas-resnet
```

12. Deploy the image to the OpenFaas

```
$ faas-cli deploy --image anandid/faas-resnet --name faasresnet --gateway  
http://192.168.1.111:8080
```

13. Test OpenFaas function

```
curl http://192.168.1.111:8080/function/faasresnet --data-binary @data/tiger.jpg
```

2.7.5.3 Install Python Libraries

Following are the steps used to install Python libraries:

- Install latest Anaconda for Windows 64-bit for Python 3
- Once Anaconda installed, use the Anaconda command prompt, to install Tensorflow and Keras
- We can add Python exe folder into windows system environment path variable, so that we can use python from the regular command prompt
- We can install Jupyter Notebook as well, given we are using Jupyter notebook to write the CNN algorithm for the image classification
- Detailed steps are provided in the installation/deployment instructions

2.7.6 Project Files

File	Description
readme.md	Instructions on how to deploy and use OpenFaaS Serverless Functions.
Dockerfile	Docker Image file with our OpenFaaS function and all the python dependencies that can be deployed onto typical public cloud providers: Azure, AWS, Google Cloud.
resnet_pretrained_classify	OpenFaaS serverless function folder with related files to classify the uploaded animal image using ResNet image network pre-trained model using Keras and Tensorflow libraries. The classification happens very quick,

	hence qualifies to be a serverless function.
index.py	Entry python file to call in the docker image.
image_classifier_dogsandcats.ipynb	Jupyter notebook with detailed analysis and exploration of Convolutional Neural Network to train the model using about 20,000 images of dogs and cats. The saved model will be used in the OpenFaaS function to test the classification of the uploaded image. Python uses Keras and Tensorflow Neural Network to perform the modeling.
classify_pre_trained_model.ipynb	Jupyter notebook to classify image of an animal using ResNet50 pre-trained model through Keras and Tensorflow.

2.8 CONCLUSION

OpenFaaS facilitates clean design, development, deployment and support of the function-as-a-service (micro-services) implementations. OpenFaaS creates competitive (co-operation and competition) environment with public cloud providers. With all the needed built-in methodologies - API Gateway, FaaS, etc. and tools - security, logging, integrations with DevOp tools, etc., OpenFaaS is already a very good open source alternative for building micro-services and maturity of this framework is drastically increasing with growing usage community and adoption. OpenFaaS can be helpful to host variety of FaaS functions all the way from http-based functions to complex functions such as machine learning based predictions and classifications.

2.9 TEAM MEMBERS AND WORK BREAKDOWN

- Murali Cheruvu worked on the CNN, OpenFaaS function, Docker Image and Deploying Azure (Single-Node Cluster)
- Anand Sriramulu worked on OpenFaaS function, Docker Image and

Deploying to Raspberry Pi (Multi-Node Cluster)

2.10 ACKNOWLEDGEMENT

The author would like to thank Dr. Gregor von Laszewski and the Teaching Assistants for their support and valuable suggestions. Author would also like to thank authors listed in the bibliography along with OpenFaas and the community of OpenFaaS for great collaboration and providing invaluable documentation and sample projects.

3 HySDS ON KUBERNETES: ARIA InSAR PROCESSING ON XSEDE JETSTREAM

Gerald Manipon, Gregor von Laszewski, Hook Hua

gmanipon@iu.edu, laszewski@gmail.com, hook.hua@jpl.nasa.gov

Indiana University, Indiana University, NASA Jet Propulsion Laboratory

hid: fa18-516-14

github: [cloud](#)

code: [cloud](#)

Keywords: E516, NASA, JPL, HySDS, Hybrid Cloud Science Data System, InSAR, Interferogram, Radar Interferometry, Sentinel-1, ISCE, InSAR Scientific Computing Environment, Kubernetes, OpenStack, Jetstream, Python, Terraform

3.1 ABSTRACT

Developed at NASA's Jet Propulsion Laboratory, the core of HySDS (Hybrid Cloud Science Data System) [13] has been running on AWS (Amazon Web Services) and OpenStack since 2011 generating science data products for projects such as ARIA (Advanced Rapid Imaging and Analysis) [14], GRFN (Getting Ready for NISAR) [15], and WVCC (Water Vapor Cloud Climatology) [16]. The upcoming NASA missions, SWOT (Surface Water and Ocean Topography) [17] and NISAR (NASA-ISRO SAR Mission) [18], are slated to launch in September 2021 and January 2022, respectively, and will use HySDS in AWS as the baseline science data system. To mitigate the risk of vendor lock-in and increase the capabilities of the system, HySDS now needs to fully function on the IaaS (Infrastructure as a Service) services provided by other public and private cloud vendors such as Google Cloud Platform (GCP), Microsoft Azure, FutureSystems, Jetstream, ChameleonCloud, High Performance Computing/High End Computing and other XSEDE compute resources. Multiple funded efforts are currently in progress at JPL and EOS (Earth Observatory of Singapore) to add support for Microsoft Azure, Google Cloud Platform and the Pleiades Supercomputer however there is currently no plan to add support for NSF-funded research-based cloud resources such as

those provided by XSEDE: Jetstream, ChameleonCloud and FutureSystems. In addition, HySDS was originally developed to operate at the IaaS layer however by adapting HySDS to run on Kubernetes, a container orchestration framework open-sourced by Google, portability to other cloud vendors can be vastly improved and simplified. HySDS can leverage Kubernetes as a PaaS (Platform as a Service) service that provides an abstraction layer to the vendor-specific IaaS services. In this project for the Fall 2018 E516 course at Indiana University at Bloomington, we pathfind a potential avenue for utilizing the NSF-funded XSEDE cloud resources by prototyping a real-life science use case: ARIA InSAR processing of Sentinel-1 SLC data running on HySDS on a Kubernetes cluster on IU’s Jetstream Cloud. We found that by generalizing the way HySDS handles product generation executables (PGEs) and their inputs, HySDS can successfully leverage the IaaS abstraction provided by Kubernetes and can thus be provisioned onto any private or public cloud vendor capable of running Kubernetes. Further work however is needed to assess the additional overhead and complexity that Kubernetes adds to the end-to-end science data system as well as how well such a system performs and scales in a production environment.

3.2 BACKGROUND

HySDS was originally developed as the framework used to create the science data system (SDS) that “*scalably powers the ingestion, metadata extraction, cataloging, high-volume data processing, and publication of the geodetic data products for the Advanced Rapid Imaging & Analysis (ARIA), Getting Ready for NISAR (GRFN), and Water Vapor Cloud Climatology (WVCC) projects at JPL*” [13]. The role of the SDS is to process the raw level0 (L0) data downloaded by the ground data system (GDS) from the satellites to higher-order level1 (L1), level2 (L2) and level3 (L3) products which are more usable to end-users. The SDS then delivers these products to a distributed active archive center (DAAC) for the purpose of providing public distribution, access, and discovery of these products. Figure [16](#) depicts the role of the SDS in the end-to-end system for the upcoming NISAR mission.

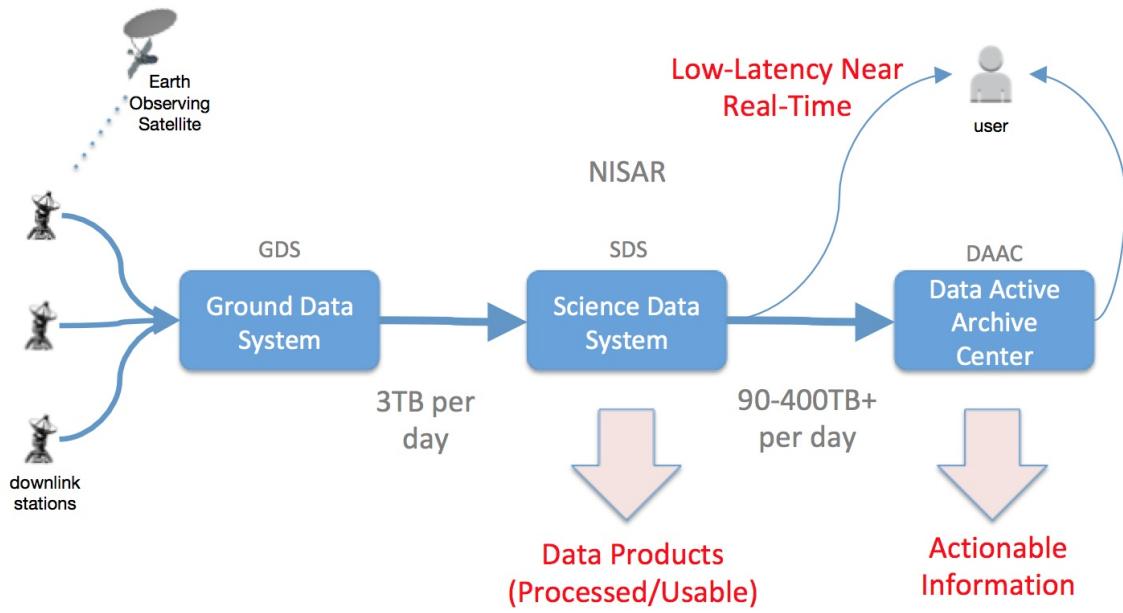


Figure 16: Large Data Flow

3.2.1 Architecture

The SDS itself is comprised of various components to orchestrate and facilitate the processing of the raw L0 satellite data. Figure 17 depicts the functional architecture of the SDS in the context of running in AWS.

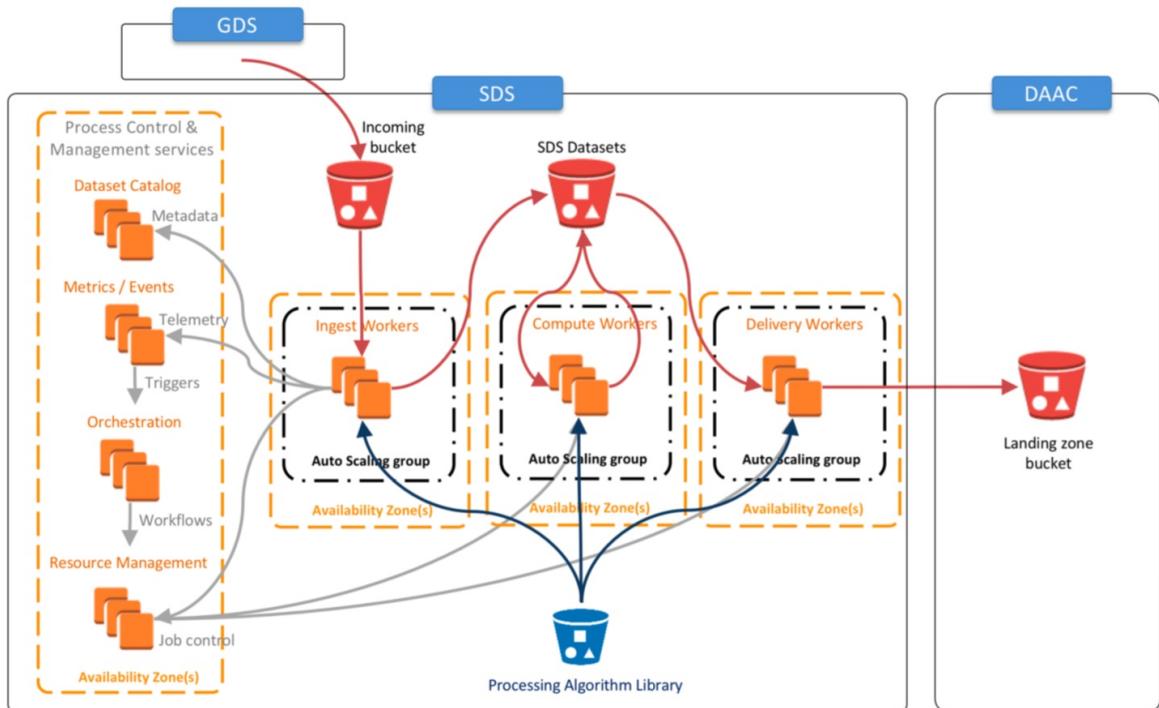


Figure 17: Functional Architecture

The GDS system will stage L0 data to the incoming bucket of the HySDS cluster. When an L0 dataset file is deposited, an event trigger submits an ingestion job to the resource manager which is then pulled by an autoscaling group of workers specifically provisioned for the purpose of dataset/product ingestion. During the ingestion job, the L0 data file is verified, metadata is extracted and browse images are generated. They are then copied to the SDS dataset bucket and the metadata/location of the L0 dataset is indexed into the dataset catalog. Upon indexing of the dataset into the catalog, trigger rules are evaluated and if their conditions are met, further downstream processing jobs are submitted to the resource manager. For example, the use-case we will be prototyping is the ARIA InSAR processing of Sentinel-1 SLCs into L2 interferograms. In the ARIA SDS, ingestion of an SLC triggers the Sentinel-1 interferogram workflow. This workflow, which runs on a different autoscaling group, determines if all the conditions have been met (e.g. pertinent ancillary orbit files, calibration files, digital elevation models exist) for processing an interferogram job. If so, it proceeds to generate the interferogram dataset which in turn gets ingested into the dataset bucket and indexed into the dataset catalog. Finally, the ingestion of these higher-order products trigger an evaluation of a delivery rule which runs on yet a different autoscaling group to deliver the interferogram to the DAAC's bucket or to notify them that it can be picked up from the SDS's dataset bucket.

3.2.2 Processing Algorithm Library

Note also in Figure 17 that HySDS provides a processing algorithm library which serves as the catalog of all PGEs (product generation executors) that can be used by the worker fleet. Processing algorithms are usually developed by subject matter experts (e.g. radar scientists) and integrated into HySDS by installing them in container images (Docker, Singularity, rkt) along with interface documents needed by the rest of the system to submit and run jobs. The Processing Algorithm Library serves as the “App Store” for all scientific data processing that would be run on a particular HySDS cluster and provides a mechanism to import or export these PGEs for exchange.

3.2.3 Pedigree

The SDS architecture detailed in Figure 17 is *NOT* novel and is very much aligned with the architectural design patterns of most other cloud data processing systems out there (e.g. AWS Batch, GCP Dataflow, Azure Batch). In fact, other data processing systems take a more cloud-native, all-in approach to their implementation. The goal of HySDS from day-one was to be as ***vendor-neutral*** and ***cross-cloud interoperable*** as possible all the while taking advantage of any performance gains by utilizing vendor-specific cloud-native features. The value-added that HySDS provides is the experience, pedigree and lessons-learned from operating science data systems at high scale since 2011. One such example is the suite of HySDS watchdogs built into the system. These watchdog subcomponents of the resource manager are dispatched to monitor and mitigate any potential issues. For example, take an SDS that has 100 jobs submitted to the resource manager. There are 5 compute workers running and executing jobs. If one of those workers had an issue, for example the root disk volume was mounted as read-only on boot-up, the job immediately fails. The compute worker then pulls the next job but that too immediately fails. Within seconds, the rogue compute worker has drained the job queue resulting in what is known as “job drain”. A highly-resilient data processing system would employ a job drain watchdog to detect this situation, determine that the rogue compute worker was not healthy, and shut down that worker immediately before more damage can be done.

Many of these lessons-learned have come from employing HySDS on forward processing and bulk reprocessing campaigns for various NASA missions and projects. For example, during a Pleiades maintenance downtime in 2015, the OCO-2 mission [19] needed to run some major bulk reprocessing but did not have enough resources on-premise to provide the results in time. They utilized HySDS on AWS to process 6 million soundings (1.6 TB output) in under a day by scaling out to 1000 compute workers. Figure 18 depicts the number EC2 instances running OCO-2 sounding PGEs over time.



Figure 18: OCO-2 Autoscaling

3.3 REQUIREMENTS

In this project, we propose to extend the capabilities of the open-source HySDS core components to support running on a Kubernetes cluster provisioned on Indiana University’s Jetstream Cloud, a compute resource provided through the XSEDE (Extreme Science and Engineering Discovery Environment) program. This requires that the HySDS system must be adapted to run process and control management nodes (PCM) and compute workers on a Kubernetes cluster. We will also adapt the ingestion, processing, and cataloguing of products generated through a scientific data pipeline running on the HySDS cluster to utilize cloud-specific but highly scalable features of the XSEDE resources. Once the HySDS Core system is completely functional on Kubernetes, research and flight mission adaptations can then be integrated and applied to provide real-world production use-cases. One such adaption is the ARIA SAR interferogram generation pipeline. The adapted HySDS/Kubernetes/Jetstream system would utilize JPL’s ingested synthetic-aperture radar data, orbits, calibrations, and DEMs in order to run this pipeline to generate L2 interferogram products. Use cases for running this pipeline on XSEDE resources such as Jetstream include forward processing of level 2 SAR interferometry products as well as generating real-time urgent response products to aid in disaster response to earthquakes, hurricanes, volcanic eruptions and flooding. A sample forward-processing scenario will be conducted to assess the stability, efficiency, and capabilities of the HySDS/Kubernetes/Jetstream system. We will also compare and benchmark the execution of the ARIA interferogram pipeline on this system to the baseline performance of the ARIA HySDS system on AWS. A large number of jobs will

be submitted to the system for a given amount of time. Afterwards, the results and observations will be documented to understand the system's performance.

3.4 DESIGN

Figure 19 provides a high level overview of the various infrastructure fabric that can be used by HySDS.

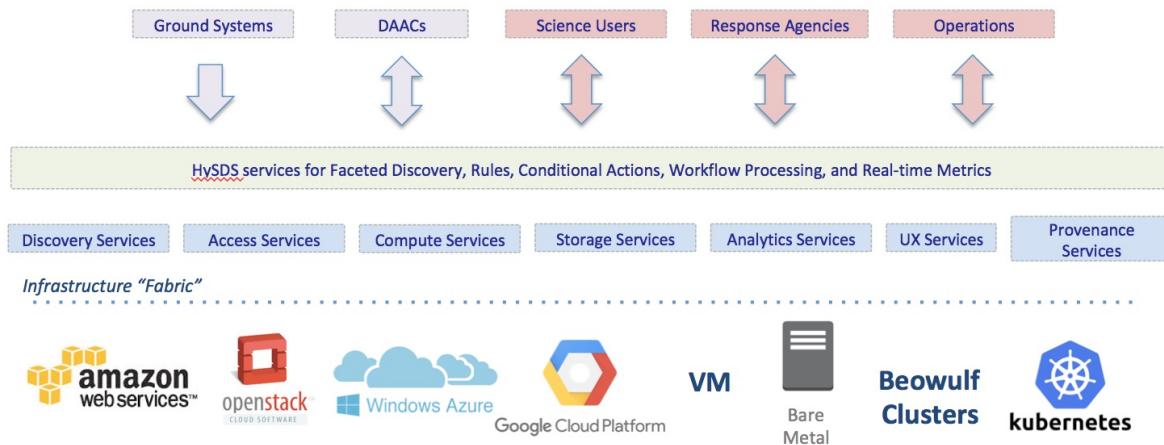


Figure 19: HySDS Infrastructure Fabric

In this project, we are adding support for HySDS to run on Kubernetes. To do this, we need to re-architect the HySDS components to conform to the way Kubernetes does things. More specifically, we need to decompose mozart (resource management), metrics (job and worker metrics/analysis), and grq (dataset catalog) components into Kubernetes pods and services.

3.5 ARCHITECTURE

3.5.1 Mozart

Figure 20 shows the architecture of the resource management component of HySDS, mozart.

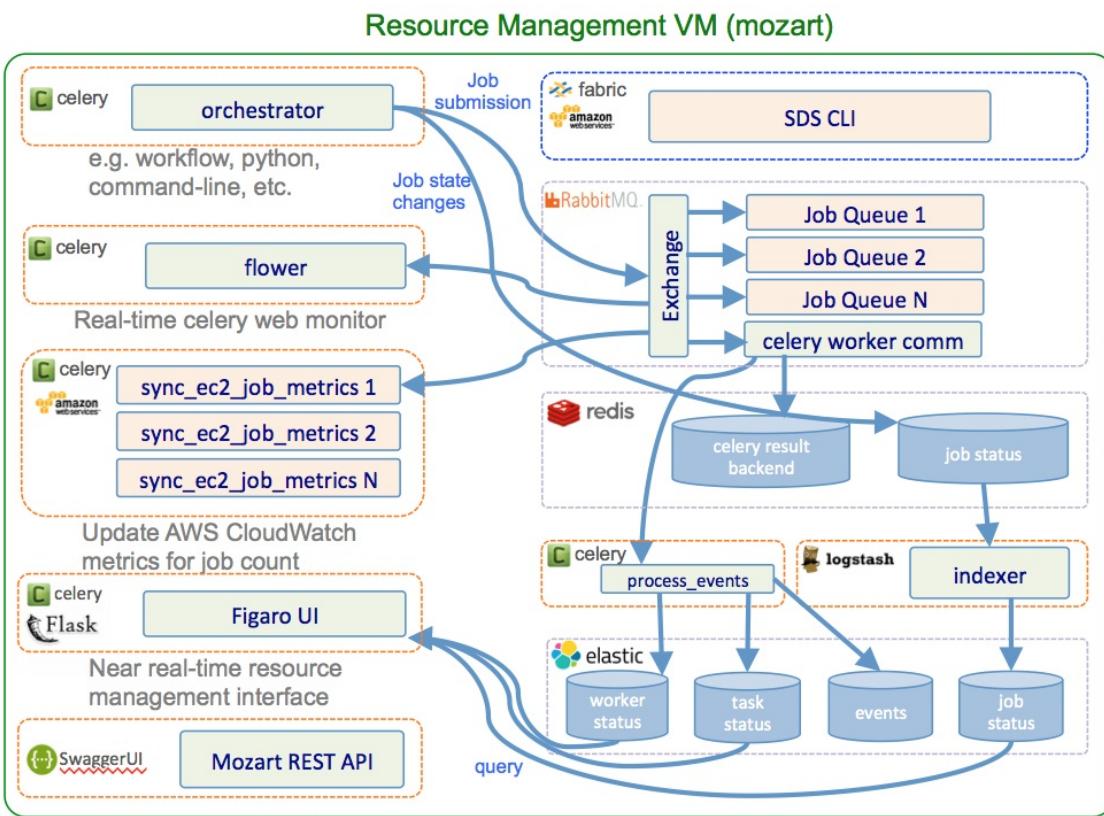


Figure 20: Mozart VM

To conform to Kubernetes best practices regarding microservices adaptation, we decided to decompose the components to the following Kubernetes pods:

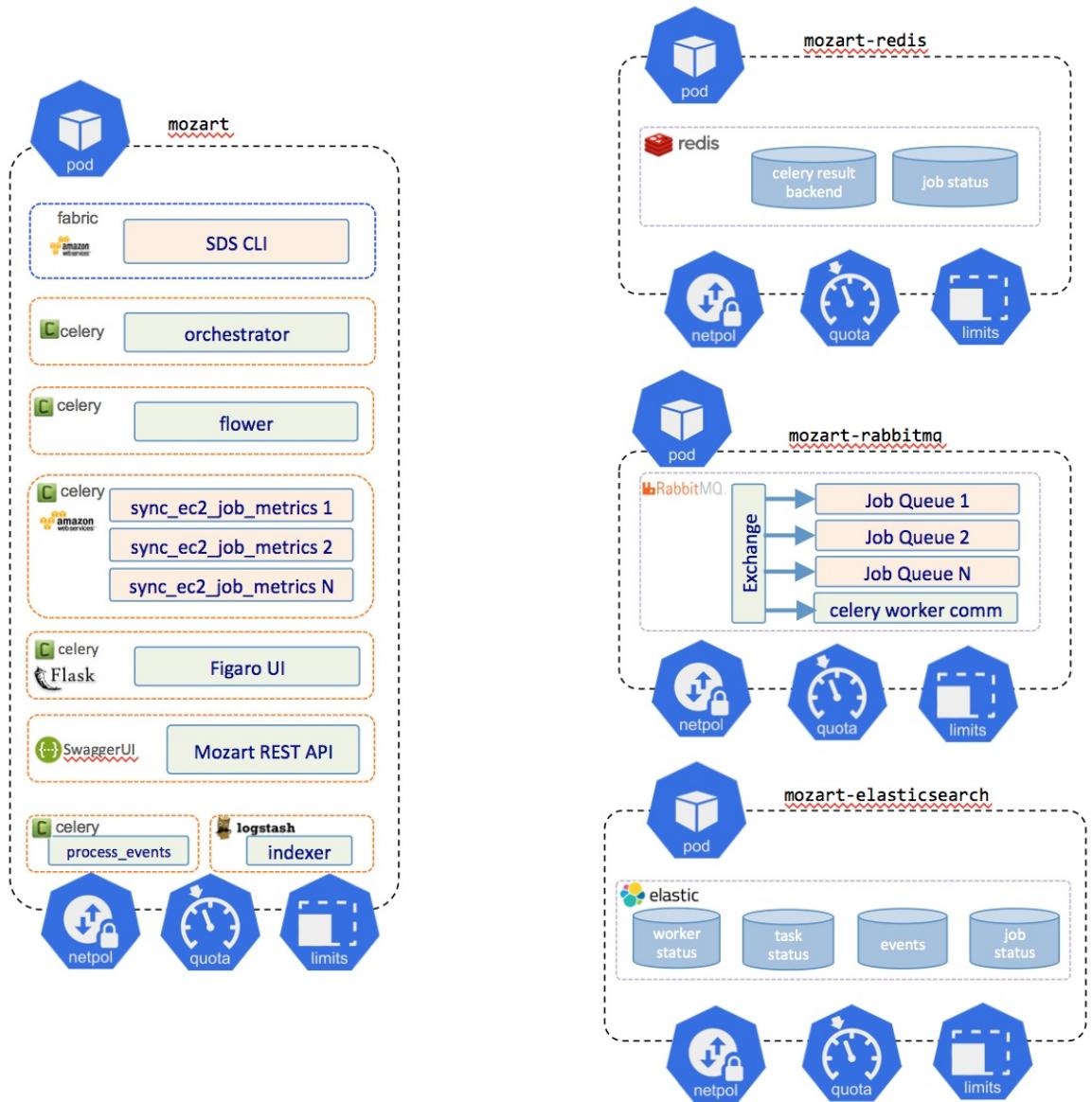


Figure 21: Mozart Pods/Services

3.5.2 Metrics

Similarly, Figure 22 shows the architecture of the metrics component of HySDS and its subsequent decomposition to Kubernetes pods:

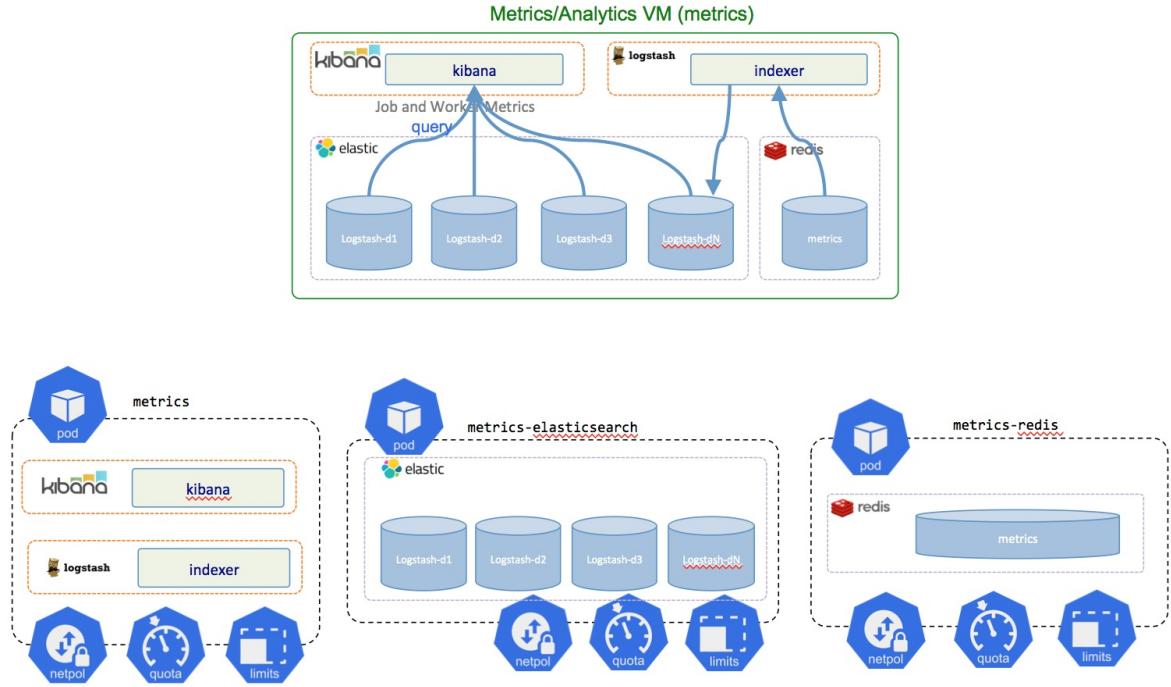


Figure 22: Metrics VM

3.5.3 GRQ

Finally, Figure 23 shows the architecture of the dataset catalog component of HySDS and its subsequent decomposition to Kubernetes pods:

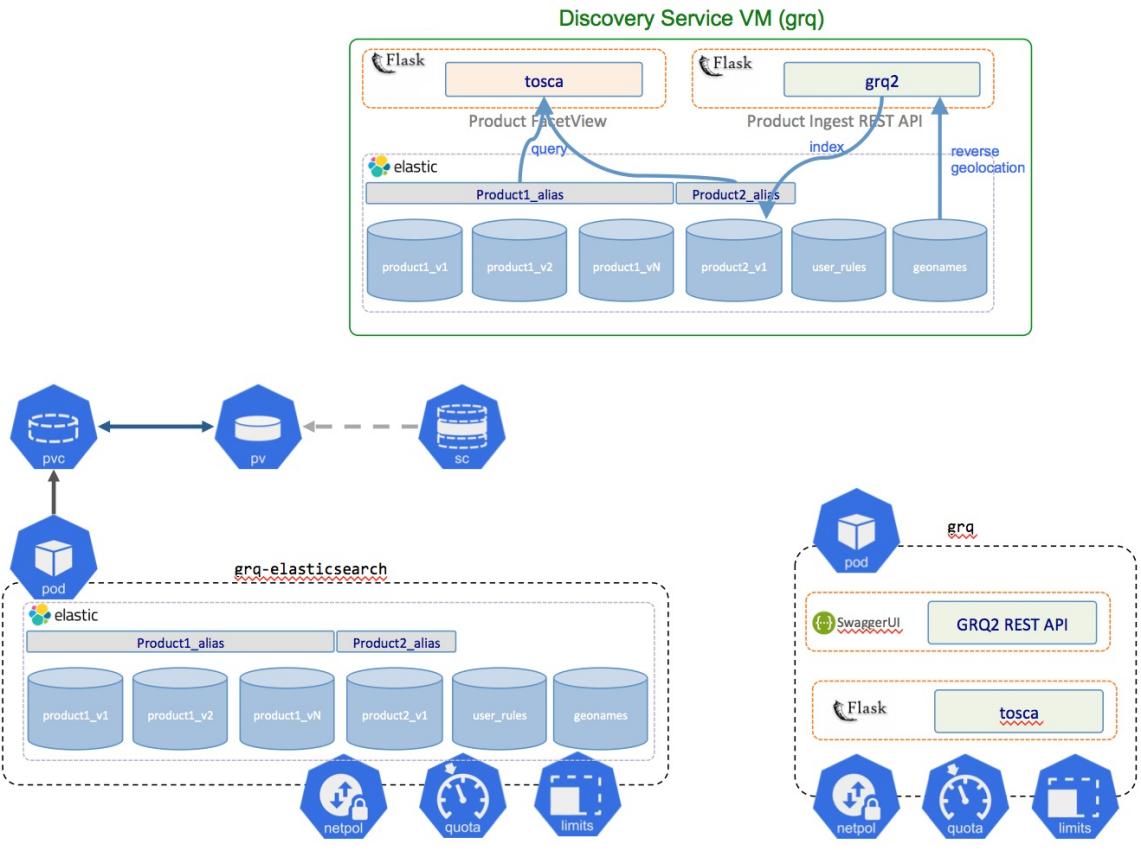


Figure 23: GRQ VM

3.5.4 hysds-k8s

The decomposition of the HySDS components to microservice pods was implemented in a new github repository located at <https://github.com/pymonger/hysds-k8s> on the “grfn-jetstream-iu” branch: <https://github.com/pymonger/hysds-k8s/tree/grfn-jetstream-iu>. To verify and validate the operation of the HySDS cluster running on Kubernetes, we exercised the “Hello World” and “Hello Dataset” tutorials located at <https://github.com/hysds/hysds-framework/wiki>Hello-World> and <https://github.com/hysds/hysds-framework/wiki>Hello-Dataset>, respectively.

3.6 DATASET

As stated before, once we’ve adapted HySDS to run on Kubernetes, we will run a real-world use case on the HySDS cluster: the ARIA Sentinel-1 interferogram pipeline. Figure 24 gives an overview of the various pipelines that ingest input

datasets needed by the Sentinel-1 interferogram pipeline.

S1-IFG Overview

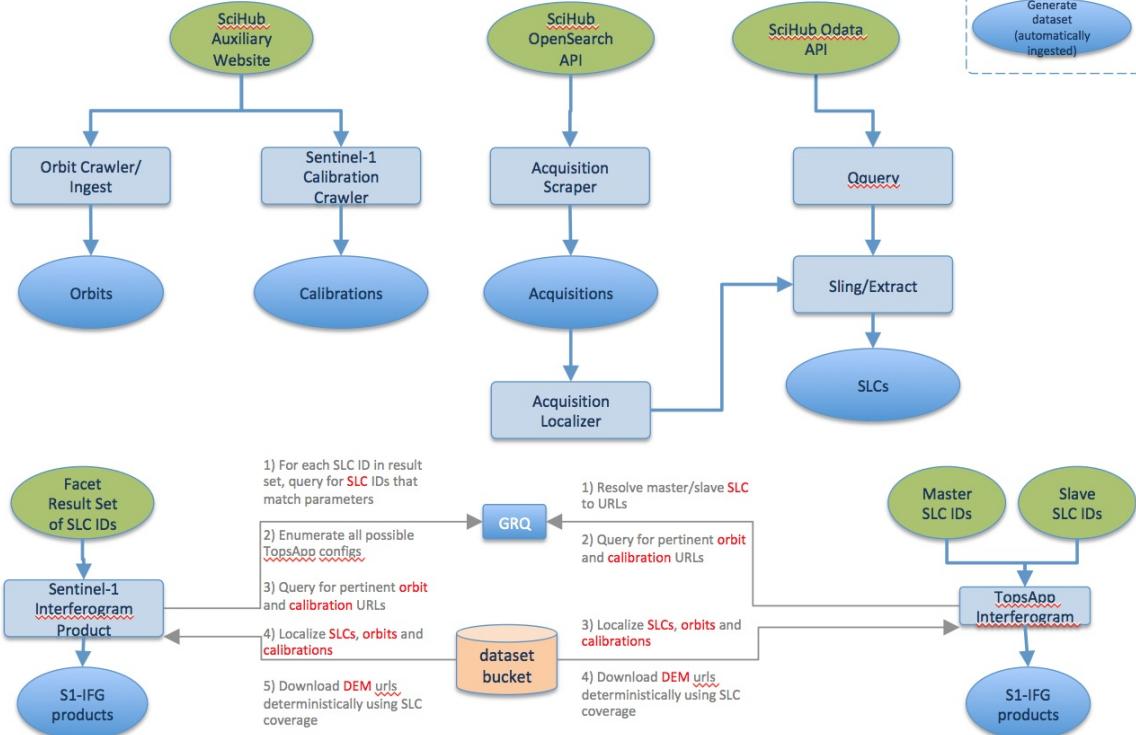


Figure 24: Sentinel-1 Interferogram

The Sentinel-1 interferogram pipeline wraps the actual PGE (product generation executor): ISCE (InSAR Scientific Computing Environment). ISCE is open source and available for download at <https://winsar.unavco.org/software/isce>. To process an interferogram, the ISCE PGE needs the following inputs:

- Sentinel-1 SLCs (single look complex)
- Sentinel-1 precise orbits
- Sentinel-1 calibrations
- USGS DEMs (digital elevation models)

3.6.1 Sentinel-1 SLCs

The ESA data product description describes the SLC dataset as follows:

“Level-1 Single Look Complex (SLC) products consist of focused SAR data geo-referenced using orbit and attitude data from the satellite and provided in zero-Doppler slant-range geometry. The products include a single look in each

dimension using the full transmit signal bandwidth and consist of complex samples preserving the phase information.” [20]

The main source for this dataset is the SciHub portal at <https://scihub.copernicus.eu/>. ESA provides a rest API to query and download the SLCs as described at <https://scihub.copernicus.eu/twiki/do/view/SciHubWebPortal/APIHubDescription>

3.6.2 Sentinel-1 Orbits and Calibrations

In order to process the Sentinel-1 SLC dataset into an interferogram, the ISCE PGE needs multiple ancillary files pertaining to the satellite acquisition date of each SLC. It needs the precise orbit files (https://qc.sentinel1.eo.esa.int/aux_poeorb/) and it needs the auxiliary calibration files (https://qc.sentinel1.eo.esa.int/aux_cal/).

3.6.3 DEMs

Finally, the ISCE PGE requires the DEM (digital elevation model) for the area covered by the SLC scene to process the interferogram. According to the USGS:

“Digital Elevation Models (DEMs) consist of a raster grid of regularly spaced elevation values that have been primarily derived from the USGS topographic map series.” [21]

The DEMs are made available at the following website: <https://e4ftl01.cr.usgs.gov/MEASURES/SRTMGL1.003/2000.02.11/>.

3.7 IMPLEMENTATION

3.7.1 Create a Kubernetes Cluster on IU Jetstream

The first order of business after we have access to the IU Jetstream OpenStack cluster is to stand up a Kubernetes cluster.

1. Provision Base CentOS7 VMs for a master and 5 worker nodes according to

<https://iujetstream.atlassian.net/wiki/spaces/JWT/pages/44826638/Setup>

The screenshot shows the OpenStack Instances page. At the top, there are navigation links: Project, API Access, Compute, and Overview. The Compute link is selected and expanded, showing sub-categories: Instances, Images, Key Pairs, Shares, Volumes, Container Infra, Network, Orchestration, Data Processing, Object Store, Identity, and Workflow. The Instances tab is selected. Below the navigation, there is a search bar with fields for Instance ID, Filter, Launch Instance, Delete Instances, and More Actions. The main table displays 6 items, each representing an instance with columns for Instance Name, Image Name, IP Address, Flavor, Key Pair, Status, Availability Zone, Task, Power State, Time since created, and Actions (Create Snapshot). The instances listed are:

Instance Name	Image Name	IP Address	Flavor	Key Pair	Status	Availability Zone	Task	Power State	Time since created	Actions
hsds-kube-node-6.novalocal	hsds-kub-e-node-v1.0	10.0.0.8 149.165.169.221	Floating IPs: m1.medium	hsds	Active	zone-r3	None	Running	10 hours, 57 minutes	Create Snapshot
hsds-kube-node-5.novalocal	hsds-kub-e-node-v1.0	10.0.0.16 149.165.157.245	Floating IPs: m1.xlarge	hsds	Active	zone-r1	None	Running	13 hours, 32 minutes	Create Snapshot
hsds-kube-node-4.novalocal	hsds-kub-e-node-v1.0	10.0.0.7 149.165.169.136	Floating IPs: m1.medium	hsds	Active	zone-r1	None	Running	1 week, 1 day	Create Snapshot
hsds-kube-node-3.novalocal	hsds-kub-e-node-v1.0	10.0.0.4 149.165.170.108	Floating IPs: m1.medium	hsds	Active	zone-r1	None	Running	1 week, 2 days	Create Snapshot
hsds-kube-node-2.novalocal	hsds-kub-e-node-v1.0	10.0.0.12 149.165.170.183	Floating IPs: m1.medium	hsds	Active	zone-r6	None	Running	1 week, 2 days	Create Snapshot
hsds-kube-node-1.novalocal	hsds-kub-e-node-v1.0	10.0.0.5 149.165.168.35	Floating IPs: m1.medium	hsds	Active	zone-r2	None	Running	1 week, 2 days	Create Snapshot

Jetstream Instances

2. Log into kube-master via ssh and sudo to root.
3. Elasticsearch requires the following kernel tuning parameter in `/etc/sysctl.conf`:

```
vm.max_map_count=262144
```

Ensure this is configured on each node you bring up (master + worker nodes). To set this on a live system without having to reboot:

```
sudo sysctl -w vm.max_map_count=262144
```

4. Configure kubernetes yum repo:

```
cat <<EOF > /etc/yum.repos.d/kubernetes.repo
[kubernetes]
name=Kubernetes
baseurl=https://packages.cloud.google.com/yum/repos/kubernetes-el7-x86_64
enabled=1
gpgcheck=1
repo_gpgcheck=1
EOF
```

```
gpgkey=https://packages.cloud.google.com/yum/doc/yum-key.gpg  
https://packages.cloud.google.com/yum/doc/rpm-package-key.gpg  
exclude=kube*  
EOF
```

5. Disable SELinux:

```
setenforce 0
```

6. Update and install kubernetes:

```
yum update -y  
yum install -y kubelet kubeadm kubectl --disableexcludes=kubernetes
```

7. Start up kubelet service via systemd:

```
systemctl enable kubelet && systemctl start kubelet
```

8. Initialize your master:

```
kubeadm init
```

```
kubeadm init  
[init] using Kubernetes version: v1.12.1  
[preflight] running pre-flight checks  
[preflight/images] Pulling images required for setting up a Kubernetes cluster  
[preflight/images] This might take a minute or two, depending on the speed of your  
internet connection  
[preflight/images] You can also perform this action in beforehand using 'kubeadm config  
images pull'  
[kubelet] Writing kubelet environment file with flags to file "/var/lib/kubelet/kubeadm-  
flags.env"  
[kubelet] Writing kubelet configuration to file "/var/lib/kubelet/config.yaml"  
[preflight] Activating the kubelet service  
[certificates] Generated ca certificate and key.  
[certificates] Generated apiserver certificate and key.  
[certificates] apiserver serving cert is signed for DNS names [js-156-171.jetstream-  
cloud.org kubernetes.kubernetes.default.kubernetes.default.svc  
kubernetes.default.svc.cluster.local] and IPs [10.96.0.1 172.28.26.7]  
[certificates] Generated apiserver-kubelet-client certificate and key.  
[certificates] Generated front-proxy-ca certificate and key.  
[certificates] Generated front-proxy-client certificate and key.  
[certificates] Generated etcd/ca certificate and key.  
[certificates] Generated etcd/server certificate and key.  
[certificates] etcd/server serving cert is signed for DNS names [js-156-171.jetstream-  
cloud.org localhost] and IPs [127.0.0.1 ::1]  
[certificates] Generated etcd/peer certificate and key.  
[certificates] etcd/peer serving cert is signed for DNS names [js-156-171.jetstream-  
cloud.org localhost] and IPs [172.28.26.7 127.0.0.1 ::1]  
[certificates] Generated etcd/healthcheck-client certificate and key.  
[certificates] Generated apiserver-etcd-client certificate and key.  
[certificates] valid certificates and keys now exist in "/etc/kubernetes/pki"  
[certificates] Generated sa key and public key.  
[kubeconfig] Wrote KubeConfig file to disk: "/etc/kubernetes/admin.conf"  
[kubeconfig] Wrote KubeConfig file to disk: "/etc/kubernetes/kubelet.conf"  
[kubeconfig] Wrote KubeConfig file to disk: "/etc/kubernetes/controller-manager.conf"  
[kubeconfig] Wrote KubeConfig file to disk: "/etc/kubernetes/scheduler.conf"  
[controlplane] wrote Static Pod manifest for component kube-apiserver to
```

```
"/etc/kubernetes/manifests/kube-apiserver.yaml"
[controlplane] wrote Static Pod manifest for component kube-controller-manager to
"/etc/kubernetes/manifests/kube-controller-manager.yaml"
[controlplane] wrote Static Pod manifest for component kube-scheduler to
"/etc/kubernetes/manifests/kube-scheduler.yaml"
[etcd] Wrote Static Pod manifest for a local etcd instance to
"/etc/kubernetes/manifests/etcd.yaml"
[init] waiting for the kubelet to boot up the control plane as Static Pods from directory
"/etc/kubernetes/manifests"
[init] this might take a minute or longer if the control plane images have to be pulled
[apiclient] All control plane components are healthy after 31.502384 seconds
[uploadconfig] storing the configuration used in ConfigMap "kubeadm-config" in the "kube-
system" Namespace
[kubelet] Creating a ConfigMap "kubelet-config-1.12" in namespace kube-system with the
configuration for the kubelets in the cluster
[markmaster] Marking the node js-156-171.jetstream-cloud.org as master by adding the label
"node-role.kubernetes.io/master=''"
[markmaster] Marking the node js-156-171.jetstream-cloud.org as master by adding the
taints [node-role.kubernetes.io/master:NoSchedule]
[patchnode] Uploading the CRI Socket information "/var/run/dockershim.sock" to the Node
API object "js-156-171.jetstream-cloud.org" as an annotation
[bootstraptoken] using token: r7zbx5.c5p8wc2yrwx0iikj
[bootstraptoken] configured RBAC rules to allow Node Bootstrap tokens to post CSRs in
order for nodes to get long term certificate credentials
[bootstraptoken] configured RBAC rules to allow the csrapprover controller automatically
approve CSRs from a Node Bootstrap Token
[bootstraptoken] configured RBAC rules to allow certificate rotation for all node client
certificates in the cluster
[bootstraptoken] creating the "cluster-info" ConfigMap in the "kube-public" namespace
[addons] Applied essential addon: CoreDNS
[addons] Applied essential addon: kube-proxy
```

Your Kubernetes master has initialized successfully!

To start using your cluster, you need to run the following as a regular user:

```
mkdir -p $HOME/.kube
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

You should now deploy a pod network to the cluster.

Run "kubectl apply -f [podnetwork].yaml" with one of the options listed at:
<https://kubernetes.io/docs/concepts/cluster-administration/addons/>

You can now join any number of machines by running the following on each node
as root:

```
kubeadm join 172.28.26.7:6443 --token r7zbx5.c5p8wc2yrwx0iikj --discovery-token-ca-cert-
hash sha256:e887df59ebb48a3483e55a101fe9e7ed0475182420608734725e46941817f35c
```

Make a record of the kubeadm join command that kubeadm init outputs. You
need this command to join nodes to your cluster.

If you ever lose the join command, run this on master:

```
kubeadm token generate
kubeadm token create <generated-token> --print-join-command --ttl=0
```

8. Exit out of root back to a normal user and configure for administration:

```

exit
mkdir -p $HOME/.kube
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
sudo chown $(id -u):$(id -g) $HOME/.kube/config

```

9. Deploy a pod network (Weave Net):

```

sudo sysctl net.bridge.bridge-nf-call-iptables=1
kubectl apply -f "https://cloud.weave.works/k8s/net?k8s-version=$(kubectl version | base64 | tr -d '\n')"

```

Confirm that it is working by checking that the CoreDNS pod is running in the output of `kubectl get pods --all-namespaces`. Once the CoreDNS pod is up and running, you can continue by joining your nodes:

NAMESPACE	NAME	READY	STATUS
RESTARTS	AGE		
kube-system	coredns-576cbf47c7-dd68p	1/1	Running 0
9m35s			
kube-system	coredns-576cbf47c7-m7r6t	1/1	Running 0
9m35s			
kube-system	etcd-js-156-171.jetstream-cloud.org	1/1	Running 0
8m52s			
kube-system	kube-apiserver-js-156-171.jetstream-cloud.org	1/1	Running 0
8m33s			
kube-system	kube-controller-manager-js-156-171.jetstream-cloud.org	1/1	Running 0
8m52s			
kube-system	kube-proxy-vt9cn	1/1	Running 0
9m35s			
kube-system	kube-scheduler-js-156-171.jetstream-cloud.org	1/1	Running 0
8m36s			
kube-system	weave-net-wrtn6	2/2	Running 0
2m3s			

9. Repeat step 2-6 above for each worker node you want to have join your cluster.
10. For each worker, ssh into it, sudo to root and run the join command output by `kubeadm init`:

```

kubeadm join --token <token> <master-ip>:<master-port> --discovery-token-ca-cert-hash
sha256:<hash>

```

For example:

```

kubeadm join 172.28.26.7:6443 --token r7zbx5.c5p8wc2yrwx0iikj --discovery-token-ca-cert-
hash sha256:e887df59ebb48a3483e55a101fe9e7ed0475182420608734725e46941817f35c
[preflight] running pre-flight checks
    [WARNING RequiredIPVSKernelModulesAvailable]: the IPVS proxier will not be used,
because the following required kernel modules are not loaded: [ip_vs_wrr ip_vs_sh ip_vs
ip_vs_rr] or no builtin kernel ipvs support: map[nf_conntrack_ipv4:{} ip_vs:{} ip_vs_rr:{}]
ip_vs_wrr:{} ip_vs_sh:{}]
you can solve this problem with following methods:
  1. Run 'modprobe -' to load missing kernel modules;

```

2. Provide the missing builtin kernel ipvs support

```
[discovery] Trying to connect to API Server "172.28.26.7:6443"
[discovery] Created cluster-info discovery client, requesting info from
"https://172.28.26.7:6443"
[discovery] Requesting info from "https://172.28.26.7:6443" again to validate TLS against
the pinned public key
[discovery] Cluster info signature and contents are valid and TLS certificate validates
against pinned roots, will use API Server "172.28.26.7:6443"
[discovery] Successfully established connection with API Server "172.28.26.7:6443"
[kubelet] Downloading configuration for the kubelet from the "kubelet-config-1.12"
ConfigMap in the kube-system namespace
[kubelet] Writing kubelet configuration to file "/var/lib/kubelet/config.yaml"
[kubelet] Writing kubelet environment file with flags to file "/var/lib/kubelet/kubeadm-
flags.env"
[preflight] Activating the kubelet service
[tlsbootstrap] Waiting for the kubelet to perform the TLS Bootstrap...
[patchnode] Uploading the CRI Socket information "/var/run/dockershim.sock" to the Node
API object "js-156-120.jetstream-cloud.org" as an annotation
```

This node has joined the cluster:

```
* Certificate signing request was sent to apiserver and a response was received.
* The Kubelet was informed of the new secure connection details.
```

Run 'kubectl get nodes' on the master to see this node join the cluster.

Verify that the worker node was registered by running `kubectl get nodes` on the master node:

```
kubectl get nodes
NAME                  STATUS   ROLES     AGE      VERSION
js-156-120.jetstream-cloud.org   Ready    <none>   41s      v1.12.1
js-156-171.jetstream-cloud.org   Ready    master    18m      v1.12.1
```

3.7.2 Configuring OpenStack Cloud Provider

In order to enable the OpenStack Cloud Provider for Kubernetes, the following steps must be performed:

1. Log onto the master node and run `hostname -f` to determine it's FQDN
2. Use the OpenStack dashboard or CLI to ensure that the instance name matches the FQDN

```
(env) centos@hysds-kube-node-1:~$ source ~/TG-CDA180009-openrc-v3.sh
Please enter your OpenStack Password for project TG-CDA180009 as user gmanipon:
(env) centos@hysds-kube-node-1:~$ openstack server list
+-----+-----+-----+-----+
| ID              | Name            | Status | Networks
| Image           | Flavor          |        |
+-----+-----+-----+-----+
| 53455d81-0644-4a8a-949d-e1cb1fa09622 | hysds-kube-node-3 | ACTIVE | hysds_net=10.0.0.4,
149.165.170.108 | hysds-kube-node-v1.0 | m1.medium |
```

```

| 56c94413-f1b6-4416-9ce3-688e765cb159 | hysds-kube-node-2 | ACTIVE | hysds_net=10.0.0.12,
149.165.170.183 | hysds-kube-node-v1.0 | m1.medium |
| 5f2e73e3-26fa-418d-b3d4-169f90e2d15a | hysds-kube-node-1 | ACTIVE | hysds_net=10.0.0.5,
149.165.168.35 | hysds-kube-node-v1.0 | m1.medium |
+-----+-----+-----+
-----+-----+-----+
(env) centos@hysds-kube-node-1:~$ hostname -f
hysds-kube-node-1.novalocal

```

3. In this example, the master node's FQDN is `hysds-kube-node-1.novalocal` while the instance name in openstack is `hysds-kube-node-1`.
4. They need to match for the in-tree OpenStack cloud provider to work so use the dashboard or the CLI to change the instance name to match the FQDN.
5. Do this for the master and all kube nodes. #### On the master node
6. Create the cloud config file which contains the pertinent OpenStack configuration as `/etc/kubernetes/cloud.conf`. This info can be pulled from the OpenStack rc file. Note that if the `availability zone` for Nova instances and Cinder volumes are different, set the `ignore-volume-az` parameter to `true`. You can determine this by creating an openstack instance and cinder volume and comparing the `availability zone` they exist in [Cloud Providers - Kubernetes](#):

```

[Global]
auth-url=https://iu.jetstream-cloud.org:5000/v3
username=gmanipon
password=*****
tenant-id=b16ff2e9abbb41f0b3dcb6b5ad0bd423
domain-id=decf397762654fa2945ae7d4cc49d8c2
tenant-name=TG-CDA180009
domain-name=tacc
region = RegionOne

[BlockStorage]
ignore-volume-az=true

```

7. Under `/etc/kubernetes/manifests` modify the `kube-controller-manager.yaml` and `kube-apiserver.yaml` to add the `cloud-config` and `cloud-provider` options as well as volume mounts to the `cloud.conf` you just created:

```

[...]
spec:
  containers:
    - command:
        - kube-controller-manager (and kube-apiserver)
        - --cloud-provider=openstack
        - --cloud-config=/etc/kubernetes/cloud.conf

[...]
  volumeMounts:
    - mountPath: /etc/kubernetes/cloud.conf

```

```

name: cloud-config
readOnly: true

[...]
volumes:
- hostPath:
  path: /etc/kubernetes/cloud.conf
  type: FileOrCreate
  name: cloud-config
[...]

```

8. Once edited, the KCM and Kube API server will automatically restart. If you receive the message “The connection to the server x was refused —did you specify the right host or port?”, don’t worry, the Kube API server is restarting. It’s normal to lose the connection with it. To confirm that these components have been configured, you can run:

```

kubectl describe pod kube-controller-manager -n kube-system | grep
'/etc/kubernetes/cloud.conf'

- cloud-config=/etc/kubernetes/cloud.conf
/etc/kubernetes/cloud.conf from cloud-config (ro)
Path: /etc/kubernetes/cloud.conf

```

3.7.2.1 On all nodes (including the master)

9. Add the `/etc/kubernetes/cloud.conf` created on the master node to file to the same path on all nodes.
10. Edit the `/etc/systemd/system/kubelet.service.d/10-kubeadm.conf` file, adding the `-cloud-provider=openstack` and `-cloud-config=/etc/kubernetes/cloud.conf` parameters in the `KUBELET_CONFIG_ARGS` environment variable:

```

# Note: This dropin only works with kubeadm and kubelet v1.11+
[Service]
Environment="KUBELET_KUBECONFIG_ARGS=--bootstrap-kubeconfig=/etc/kubernetes/bootstrap-
kubelet.conf --kubeconfig=/etc/kubernetes/kubelet.conf"
#Environment="KUBELET_CONFIG_ARGS=--config=/var/lib/kubelet/config.yaml"
Environment="KUBELET_CONFIG_ARGS=--config=/var/lib/kubelet/config.yaml --cloud-
provider=openstack --cloud-config=/etc/kubernetes/cloud.conf"
# This is a file that "kubeadm init" and "kubeadm join" generates at runtime, populating
the KUBELET_KUBEADM_ARGS variable dynamically
EnvironmentFile=-/var/lib/kubelet/kubeadm-flags.env
# This is a file that the user can use for overrides of the kubelet args as a last resort.
Preferably, the user should use
# the .NodeRegistration.KubeletExtraArgs object in the configuration files instead.
KUBELET_EXTRA_ARGS should be sourced from this file.
EnvironmentFile=-/etc/sysconfig/kubelet
ExecStart=
ExecStart=/usr/bin/kubelet $KUBELET_KUBECONFIG_ARGS $KUBELET_CONFIG_ARGS
$KUBELET_KUBEADM_ARGS $KUBELET_EXTRA_ARGS

```

11. Restart kubelet service using:

```
sudo systemctl daemon-reload  
sudo systemctl restart kubelet
```

12. Ensure that the node was successfully registered:

```
sudo tail -f /var/log/messages  
  
Oct 27 15:51:32 hysds-kube-node-3 kubelet: I1027 15:51:32.470352      3320  
reconciler.go:207] operationExecutor.VerifyControllerAttachedVolume started for volume  
"weavedb" (UniqueName: "kubernetes.io/host-path/b61b79a2-d9c2-11e8-b5d5-fa163e02ac5c-  
weavedb") pod "weave-net-mfdr9" (UID: "b61b79a2-d9c2-11e8-b5d5-fa163e02ac5c")  
Oct 27 15:51:32 hysds-kube-node-3 kubelet: I1027 15:51:32.470366      3320  
reconciler.go:154] Reconciler: start to sync state  
Oct 27 15:51:32 hysds-kube-node-3 kubelet: I1027 15:51:32.898253      3320  
kubelet_node_status.go:324] Adding node label from cloud provider:  
beta.kubernetes.io/instance-type=3  
Oct 27 15:51:32 hysds-kube-node-3 kubelet: I1027 15:51:32.898300      3320  
kubelet_node_status.go:335] Adding node label from cloud provider: failure-  
domain.beta.kubernetes.io/zone=zone-r1  
Oct 27 15:51:32 hysds-kube-node-3 kubelet: I1027 15:51:32.898313      3320  
kubelet_node_status.go:339] Adding node label from cloud provider: failure-  
domain.beta.kubernetes.io/region=RegionOne  
Oct 27 15:51:32 hysds-kube-node-3 kubelet: I1027 15:51:32.901236      3320  
kubelet_node_status.go:70] Attempting to register node hysds-kube-node-3.novalocal  
Oct 27 15:51:33 hysds-kube-node-3 kubelet: I1027 15:51:33.227417      3320  
kubelet_node_status.go:112] Node hysds-kube-node-3.novalocal was previously registered  
Oct 27 15:51:33 hysds-kube-node-3 kubelet: I1027 15:51:33.458241      3320  
kubelet_node_status.go:73] Successfully registered node hysds-kube-node-3.novalocal
```

You should now have a running Kubernetes cluster configured with the in-tree OpenStack Cloud Provider:

```

(env) centos@hyds-kube-node-1:~$ kubectl get svc,deploy,pod,no --all-namespaces
          gmanipon — centos@hyds-kube-node-1: ~ ssh -i ~/AWS_keypairs/jetstream-iu/hyds.pem centos@kube-master.aria.hysds.io — 200>?

NAMESPACE   NAME           TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)          AGE
default     service/ci    NodePort   10.104.1.33    <none>        80:3070/TCP,8080:31514/TCP,8085:31520/TCP,9001:31440/TCP   11h
default     service/factotum  NodePort  10.103.13.240   <none>        22:30975/TCP,80:30617/TCP,443:30895/TCP,8085:32236/TCP,9001:32441/TCP   11h
default     service/gr-elasticsearch ClusterIP  10.103.243.29   <none>        22:31327/TCP,80:32666/TCP,443:32633/TCP,8878:32205/TCP,8879:30165/TCP,9001:31047/TCP   11h
default     service/gr-redis   NodePort   10.103.162.9    <none>        6379/TCP,9160:32724/TCP,9300:31836/TCP   11h
default     service/kubernetes ClusterIP  10.96.0.1       <none>        443/TCP   44h
default     service/metrics-elasticsearch ClusterIP  10.101.36.156   <none>        22:32307/TCP,80:31114/TCP,443:30311/TCP,5601:30713/TCP,9001:30962/TCP   11h
default     service/metrics-redis   NodePort   10.101.36.155   <none>        3000:32652/TCP,9300:31517/TCP   11h
default     service/mozart   ClusterIP  10.104.1.97     <none>        6379/TCP   44h
default     service/mozart-redis   NodePort   10.100.99.7    <none>        22:30781/TCP,80:32763/TCP,443:32708/TCP,5555:32623/TCP,8888:31085/TCP,8898:31677/TCP,9001:31336/TCP   11h
default     service/mozart-elasticsearch ClusterIP  10.106.14.114   <none>        9200:32630/TCP,9300:30831/TCP   11h
default     service/mozart-rabbitmq  NodePort   10.106.14.239   <none>        5671:32709/TCP,15672:31045/TCP   11h
default     service/verdi-redis   ClusterIP  10.111.72.186   <none>        6379/TCP   11h
default     service/verdi   NodePort   10.104.121.191   <none>        22:32417/TCP,80:32574/TCP,443:32359/TCP,8085:31773/TCP,9001:32008/TCP   43m
kube-system  kube-dns      ClusterIP  10.96.0.10     <none>        53:UDP,53:TCP   44h

NAMESPACE   NAME           DESIRED  CURRENT  UP-TO-DATE  AVAILABLE  AGE
default     deployment/extensions/ci  1        1        1        1        11h
default     deployment/extensions/factotum  1        1        1        1        11h
default     deployment/extensions/gr-elasticsearch  1        1        1        1        11h
default     deployment/extensions/gr-redis   1        1        1        1        11h
default     deployment/extensions/gr-elasticsearch  1        1        1        1        11h
default     deployment/extensions/metrics-redis  1        1        1        1        11h
default     deployment/extensions/metrics-elasticsearch  1        1        1        1        11h
default     deployment/extensions/mozart-elasticsearch  1        1        1        1        11h
default     deployment/extensions/mozart-redis   1        1        1        1        11h
default     deployment/extensions/rabbitmq   1        1        1        1        11h
default     deployment/extensions/verdi-redis   1        1        1        1        11h
default     deployment/extensions/verdi   1        1        1        1        43m
kube-system  deployment/extensions/coredns  2        2        2        2        44h

NAMESPACE   NAME           READY   STATUS    RESTARTS  AGE
default     pod/c1-6b99498b7-jb7hw  1/1    running  0          11h
default     pod/factotum-756485399-k7vkp  1/1    running  0          11h
default     pod/gr-6tdtmmrb5-tjtjg  1/1    running  0          11h
default     pod/polyglot-37644959c-f5d8c57-1qstp  1/1    running  0          11h
default     pod/gr-redis-9b58bd69f-8dr1l  1/1    running  0          11h
default     pod/metrics-58fc5b6d45-8c92s  1/1    running  0          11h
default     pod/metrics-redis-77bc4cc94-nej1b  1/1    running  0          11h
default     pod/mozart-redis-77bc4cc94-nej1b  1/1    running  0          11h
default     pod/mozart-647f9b5f6-f6sfv  1/1    running  0          11h
default     pod/mozart-elasticsearch-77fc7b6-9mdv6  1/1    running  0          11h
default     pod/mozart-metrics-57644959c-x09  1/1    running  0          11h
default     pod/mozart-redis-548774ff6-b71dh  1/1    running  0          11h
default     pod/verdi-659d8d467-vs2w  1/1    running  0          43m
kube-system  pod/coredns-57644959c-c9ppf  1/1    running  0          44h
kube-system  pod/etcd-hyds-kube-node-1.novalocal  1/1    running  0          44h
kube-system  pod/kube-apiserver-hyds-kube-node-1.novalocal  1/1    running  0          44h
kube-system  pod/kube-controller-manager-hyds-kube-node-1.novalocal  1/1    running  0          44h
kube-system  pod/kube-proxy-881fr  1/1    running  0          11h
kube-system  pod/kube-proxy-czgbm  1/1    running  0          12h
kube-system  pod/kube-proxy-kfrdf  1/1    running  0          44h
kube-system  pod/kube-proxy-kwlpz  1/1    running  0          44h
kube-system  pod/kube-proxy-xn5d  1/1    running  0          44h
kube-system  pod/weave-net-1r5gf  2/2    running  1          14h
kube-system  pod/weave-net-67ewk  2/2    running  0          44h
kube-system  pod/weave-net-1gtzz  2/2    running  0          12h
kube-system  pod/weave-net-1tpp5  2/2    running  0          44h
kube-system  pod/weave-net-nmcgg  2/2    running  1          11h
kube-system  pod/weave-net-thwmnt  2/2    running  0          44h

NAMESPACE   NAME           STATUS  ROLES  AGE   VERSION
node/hyds-kube-node-1.novalocal  Ready  master  44h  v1.12.2
node/hyds-kube-node-1.novalocal  Ready  <none>  44h  v1.12.2
node/hyds-kube-node-1.novalocal  Ready  <none>  44h  v1.12.2
node/hyds-kube-node-4.novalocal  Ready  <none>  44h  v1.12.2
node/hyds-kube-node-5.novalocal  Ready  <none>  14h  v1.12.2
node/hyds-kube-node-6.novalocal  Ready  <none>  11h  v1.12.2

(env) centos@hyds-kube-node-1:~$ 

```

Figure 25: Kubernetes Info

3.7.3 Create HySDS Buckets (Swift Containers)

Using the OpenStack dashboard, we now need to create 2 buckets: one for the PGE docker images and one for the datasets.

Containers

hysds-code-bucket

Name	Size	Action
aria-isce_giant-latest.tar.gz	1.41 GB	Download
container-hysds_ariamh_jetstream-iu-k8s.tar.gz	1.41 GB	Download
container-hysds_lightweight-jobs:develop.tar.gz	359.55 MB	Download
container-hysds_lightweight-jobs:master.tar.gz	361.02 MB	Download
hysds-pge-isce_giant-latest.tar.gz	1.41 GB	Download

Figure 26: Code Bucket

Containers

hysds-dataset-bucket

Name	Type	Action
browse	Folder	Delete
products	Folder	Delete

Figure 27: Dataset Bucket

3.7.4 Create the HySDS cluster

Next we provision our HySDS cluster:

1. Log into kube-master via ssh.
2. Clone the `hysds-k8s` repository and checkout the `grfn-jetstream-iu` branch:

```
git clone https://github.com/pymonger/hysds-k8s
cd hysds-k8s
git checkout grfn-jetstream-iu
```

3. Run the `create_hysds_cluster.sh` script to provision all HySDS-related

Kubernetes resources:

```
./create_hysds_cluster.sh
```

Note that the script requires 4 arguments: - AWS Access Key for the ARIA datasets bucket hosted by JPL - AWS Secret Key for the ARIA datasets bucket hosted by JPL - Swift Access Key for the HySDS code and dataset buckets hosted on IU Jetstream - Swift Secret Key for the HySDS code and dataset buckets hosted on IU Jetstream 4. Run the `get_urls.sh` script to list the urls for all HySDS web interfaces and REST APIs:

```
./get_urls.sh
```

3.7.5 Register the lightweight-jobs and ariamh repositories in Jenkins

1. Log into kube-master via ssh.
2. Log into the mozart pod using kubectl:

```
kubectl exec -ti mozart bash
```

3. Register the lightweight-jobs repo to Jenkins:

```
sds ci add_job -k -b master https://github.com/hysds/lightweight-jobs.git s3
```

4. Register the ariamh repo to Jenkins:

```
sds ci add_job -k -b jetstream-iu-k8s https://github.com/hysds/ariamh.git s3
```

You should now see both jobs registered in Jenkins:

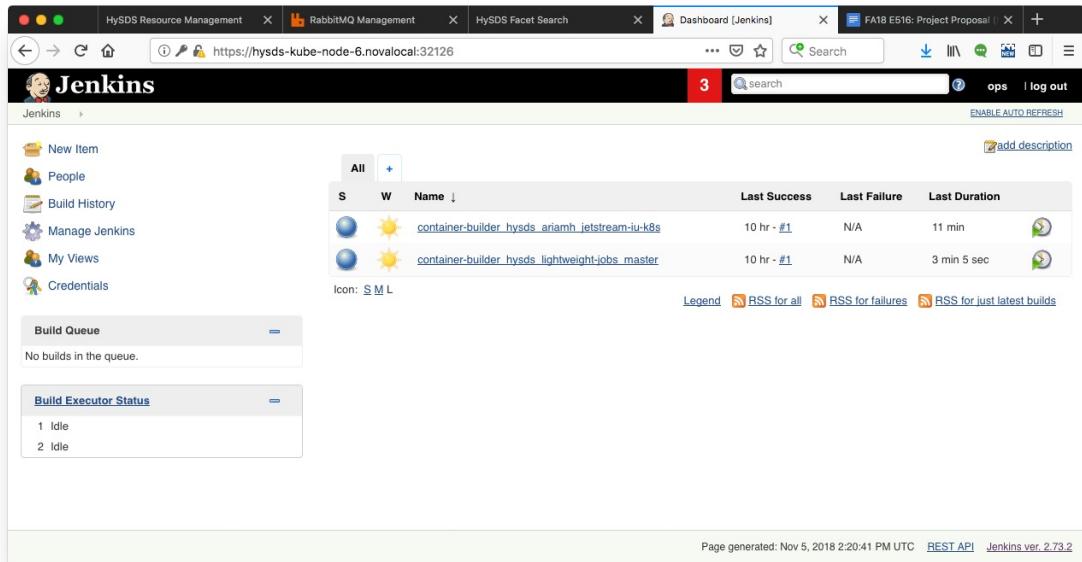


Figure 28: Jenkins Jobs

5. Log into the Jenkins web interface and push the builds for lightweight-jobs and ariamh. You should see the jobs build to completion via the console log output:

The screenshot shows a Jenkins interface with the following details:

- Project Path:** Jenkins > container-builder_ariamh_jetstream-iu-k8s
- Job Name:** container-builder_ariamh_jetstream-iu-k8s
- Build Status:** 3 (Success)
- Console Output:**
 - Started by user **ops**
 - Building in workspace /home/ops/jenkins/workspace/container-builder_ariamh_jetstream-iu-k8s
 - Cloning the remote Git repository <https://github.com/hysds/ariamh.git>
 - Fetching upstream changes from <https://github.com/hysds/ariamh.git>
 - git fetch --tags --progress https://github.com/hysds/ariamh.git +refs/heads/*:refs/remotes/origin/*
 - git config remote.origin.url https://github.com/hysds/ariamh.git # timeout=10
 - git config remote.origin.fetch +refs/heads/*:refs/remotes/origin/*
 - git config remote.origin.tag_fetch +refs/tags/*:refs/remotes/origin/*
 - git rev-parse refs/remotes/origin/jetstream-iu-k8s^{commit} # timeout=10
 - git rev-parse refs/remotes/origin/origin/jetstream-iu-k8s^{commit} # timeout=10
 - Checking out Revision 9e28821cc24bf4d29eecebd99af49d37cbfc4bbc
 - git config core.sparsecheckout # timeout=10
 - git checkout -f 9e28821cc24bf4d29eecebd99af49d37cbfc4bbc
 - Commit message: "remove .netrc binding"
 - First time build. Skipping changelog.
 - [container-builder_hysds_ariamh_jetstream-iu-k8s] \$ /bin/bash /tmp/jenkins4912048385419354008.sh
 - fatal: no tag exactly matches '9e28821cc24bf4d29eecebd99af49d37cbfc4bbc'
 - [CI] Is checkout a tag: 0
 - [CI] Last log: commit 9e28821cc24bf4d29eecebd99af49d37cbfc4bbc
 - Author: Gerald Manipon <pymonger@gmail.com>
 - Date: Sun Nov 4 18:15:42 2018 +0000
 - remove .netrc binding
 - [CI] Skip image build flag: 0
 - [WARNING] ml-trainer does not define a hysds-io.json
 - [WARNING] ml-trainer does not define a job-spec.json
 - [WARNING] sciflo-ifg-stitcher defines hysds-io parameter query without a type
 - [WARNING] sciflo-ifg-stitcher defines hysds-io parameter name without a type
 - [WARNING] sciflo-ifg-stitcher defines hysds-io parameter username without a type
 - [WARNING] sciflo-dense_offset defines hysds-io without 'submission_type'
 - [WARNING] sciflo-dense_offset defines hysds-io parameter query without a type
 - [WARNING] sciflo-dense_offset defines hysds-io parameter name without a type
 - [WARNING] sciflo-dense_offset defines hysds-io parameter username without a type
 - [WARNING] sciflo-cor defines hysds-io without 'submission_type'
 - [WARNING] sciflo-cor defines hysds-io parameter localize_url without a type
 - [WARNING] sciflo-cor defines hysds-io parameter path without a type
 - [WARNING] sciflo-topapp-slcp defines hysds-io parameter query without a type
 - [WARNING] sciflo-topapp-slcp defines hysds-io parameter name without a type
 - [WARNING] sciflo-topapp-slcp defines hysds-io parameter username without a type
 - [WARNING] sciflo-validate-ifg defines hysds-io parameter query without a type
 - [WARNING] sciflo-validate-ifg defines hysds-io parameter name without a type
 - [WARNING] sciflo-validate-ifg defines hysds-io parameter username without a type
 - [WARNING] sciflo-validate-ts defines hysds-io parameter query without a type
 - [WARNING] sciflo-validate-ts defines hysds-io parameter name without a type
 - [WARNING] sciflo-validate-ts defines hysds-io parameter username without a type
 - [WARNING] sciflo-sl-ifg defines hysds-io without 'submission_type'
 - [WARNING] sciflo-sl-ifg defines hysds-io parameter query without a type
 - [WARNING] sciflo-sl-ifg defines hysds-io parameter name without a type
 - [WARNING] sciflo-sl-ifg defines hysds-io parameter username without a type
 - [WARNING] sciflo-sl-slcp defines hysds-io without 'submission_type'
 - [WARNING] sciflo-sl-slcp defines hysds-io parameter query without a type
 - [WARNING] sciflo-sl-slcp defines hysds-io parameter name without a type
 - [WARNING] sciflo-sl-slcp defines hysds-io parameter username without a type
 - [WARNING] sciflo-topapp-ifg defines hysds-io parameter query without a type
 - [WARNING] sciflo-topapp-ifg defines hysds-io parameter name without a type
 - [WARNING] sciflo-topapp-ifg defines hysds-io parameter username without a type
 - [WARNING] sciflo-sl-slcp-mrpe defines hysds-io without 'submission_type'
 - [WARNING] sciflo-sl-slcp-mrpe defines hysds-io parameter preReferencePairDirection without a type
 - [WARNING] sciflo-sl-slcp-mrpe defines hysds-io parameter postReferencePairDirection without a type

Figure 29: Build ariamh

3.7.6 Submit Sentinel-1 Interferogram Jobs

To facilitate running the Sentinel-1 Interferogram pipeline in Kubernetes without having to ingest 1TB of input datasets, we copy the dataset indexes from the ARIA HySDS cluster at JPL. Access to these indexes requires special permissions so please contact me at gmanipon@jpl.nasa.gov for more information. Assuming the indexes have been set up on the grq component of the

HySDS/Kubernetes/Jetstream cluster, you should see SLCs, precise orbits and calibration files available in the tosca interface:

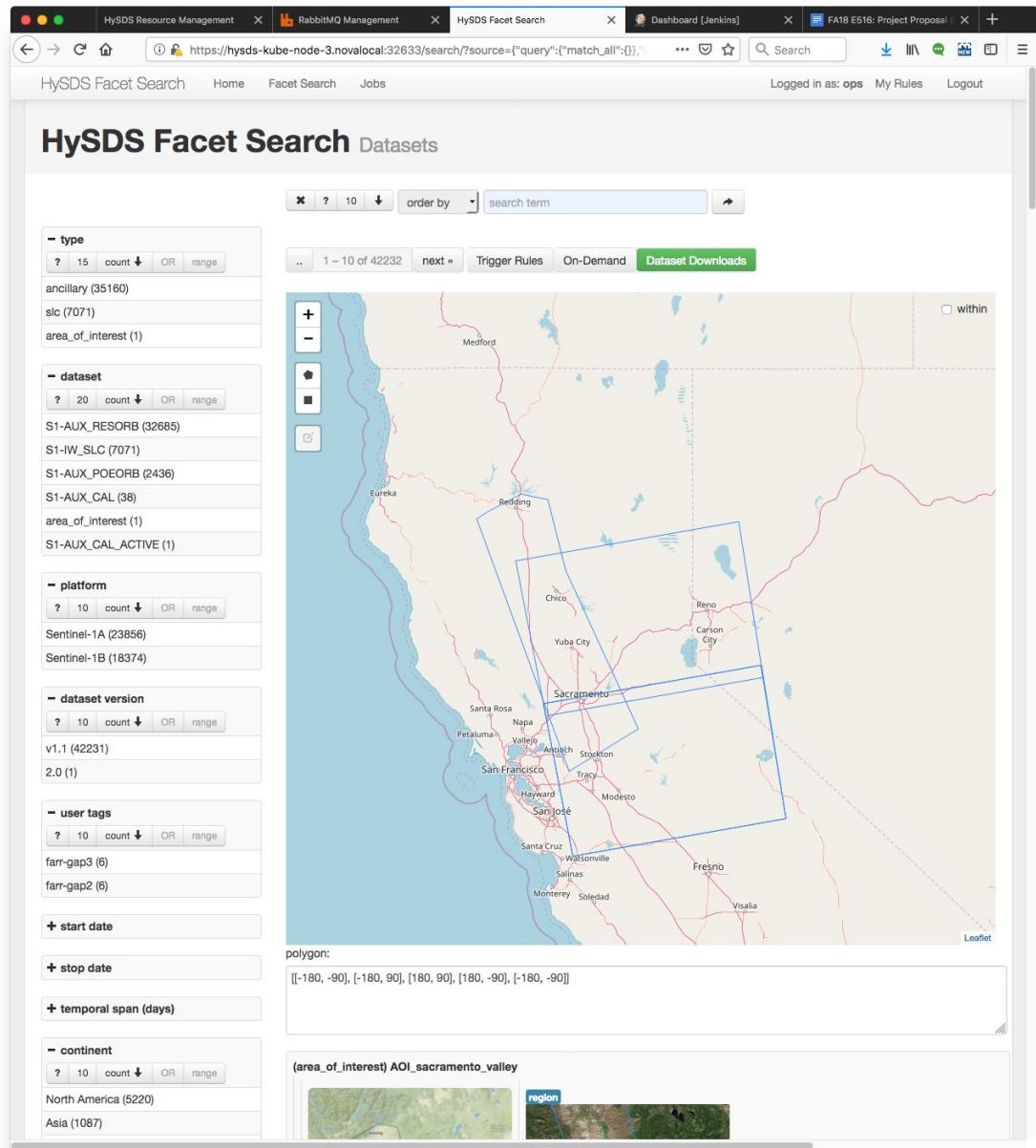


Figure 30: Input Datasets

We can now use the tosca interface to submit Sentinel-1 interferogram jobs.

1. In the tosca interface, draw a bounding box over an area of interest. In our case, we draw a box over the area north of Los Angeles. We then facet down to the S1-IW_SLC dataset type and track 71:

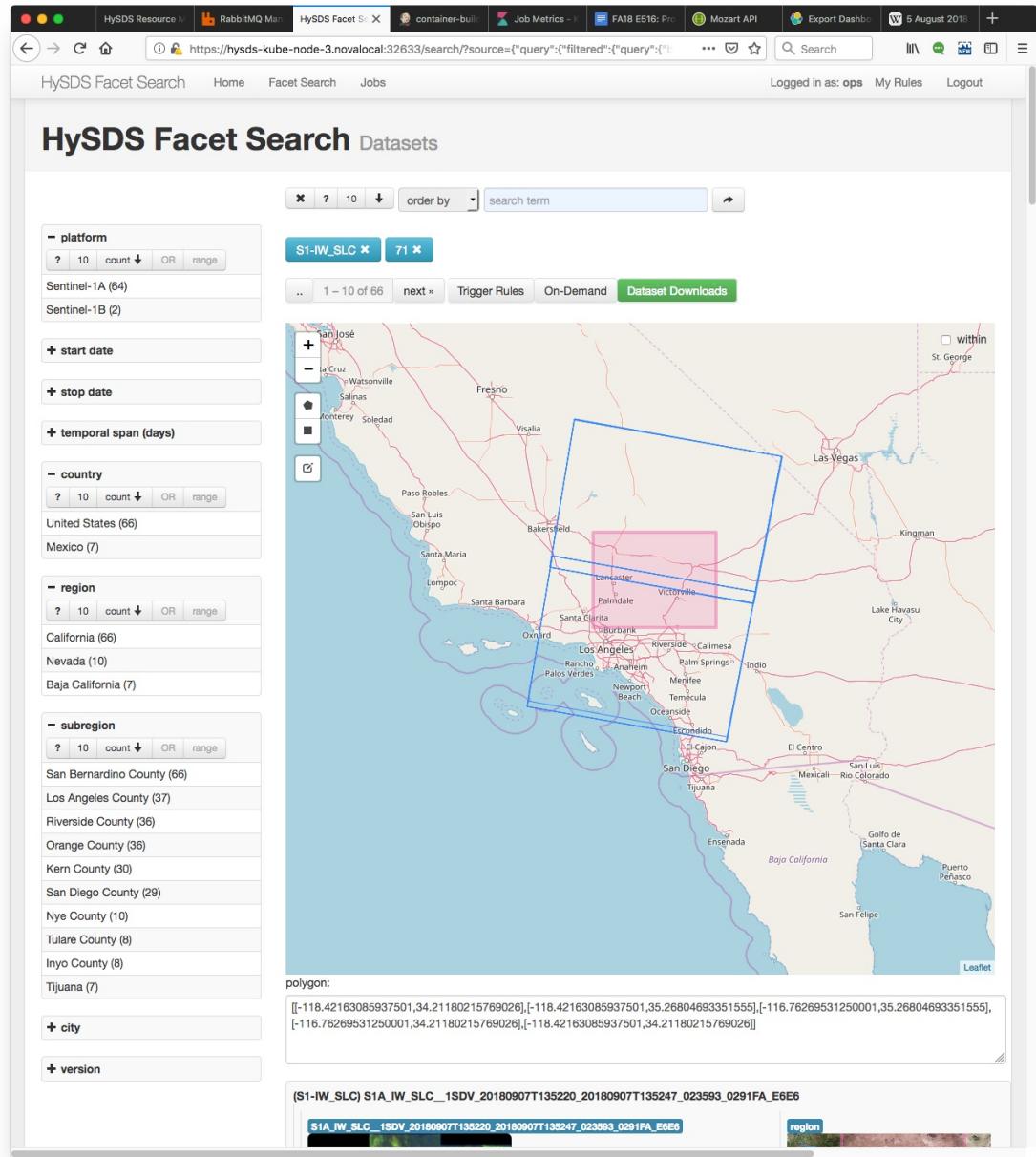


Figure 31: AOI - Los Angeles track 71

2. Click on the “On-Demand” button. This will bring up a modal window that will allow you to submit jobs based on the result set that was faceted down to. Select the “Sentinel-1 Interferogram Product [jetstream-iu-k8s]” action and select the “grfn-job_worker-small” queue:

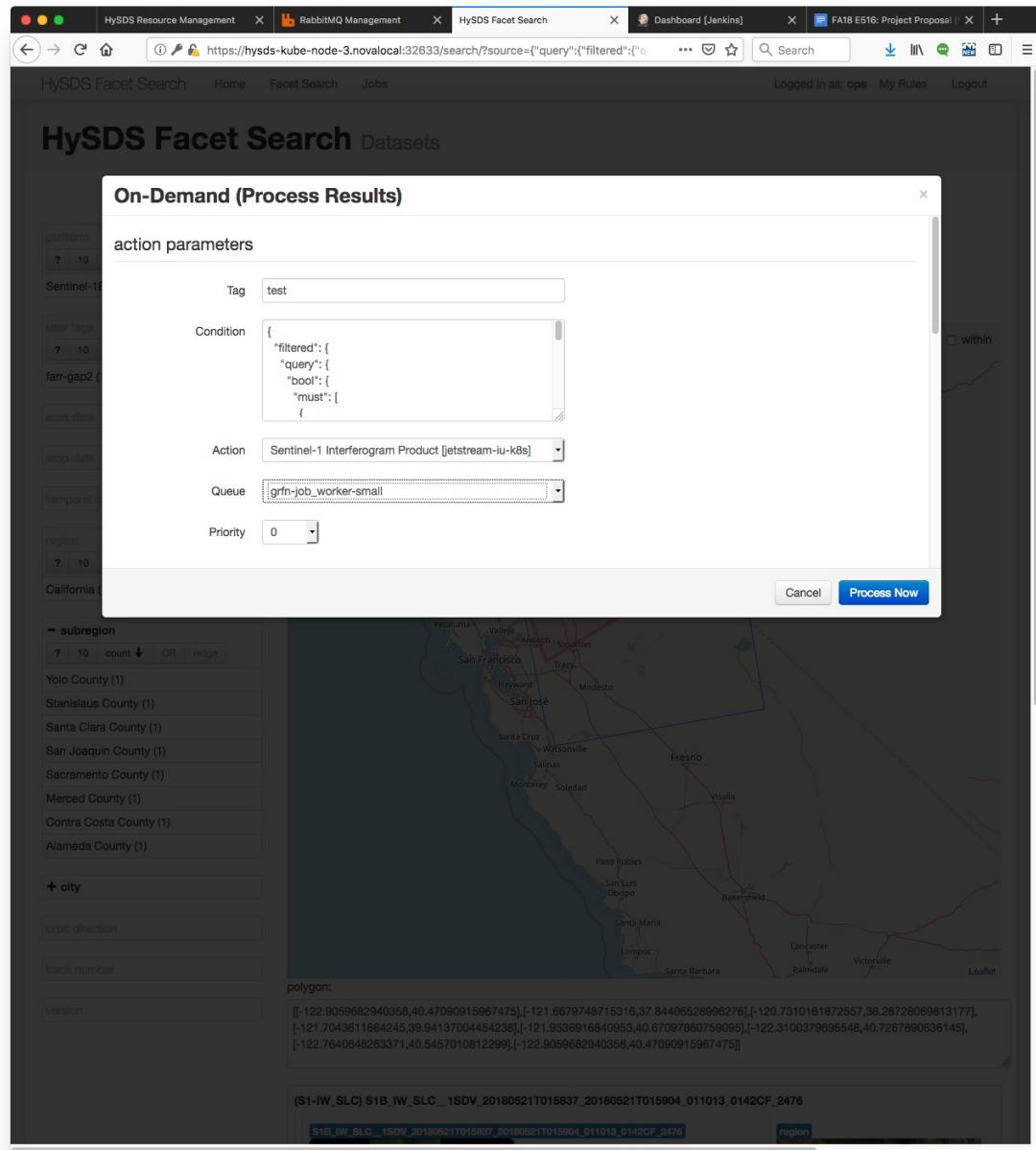


Figure 32: Configure Job

The PGE-specific parameters below can be used unchanged. Click on “Process Now” to submit the job:

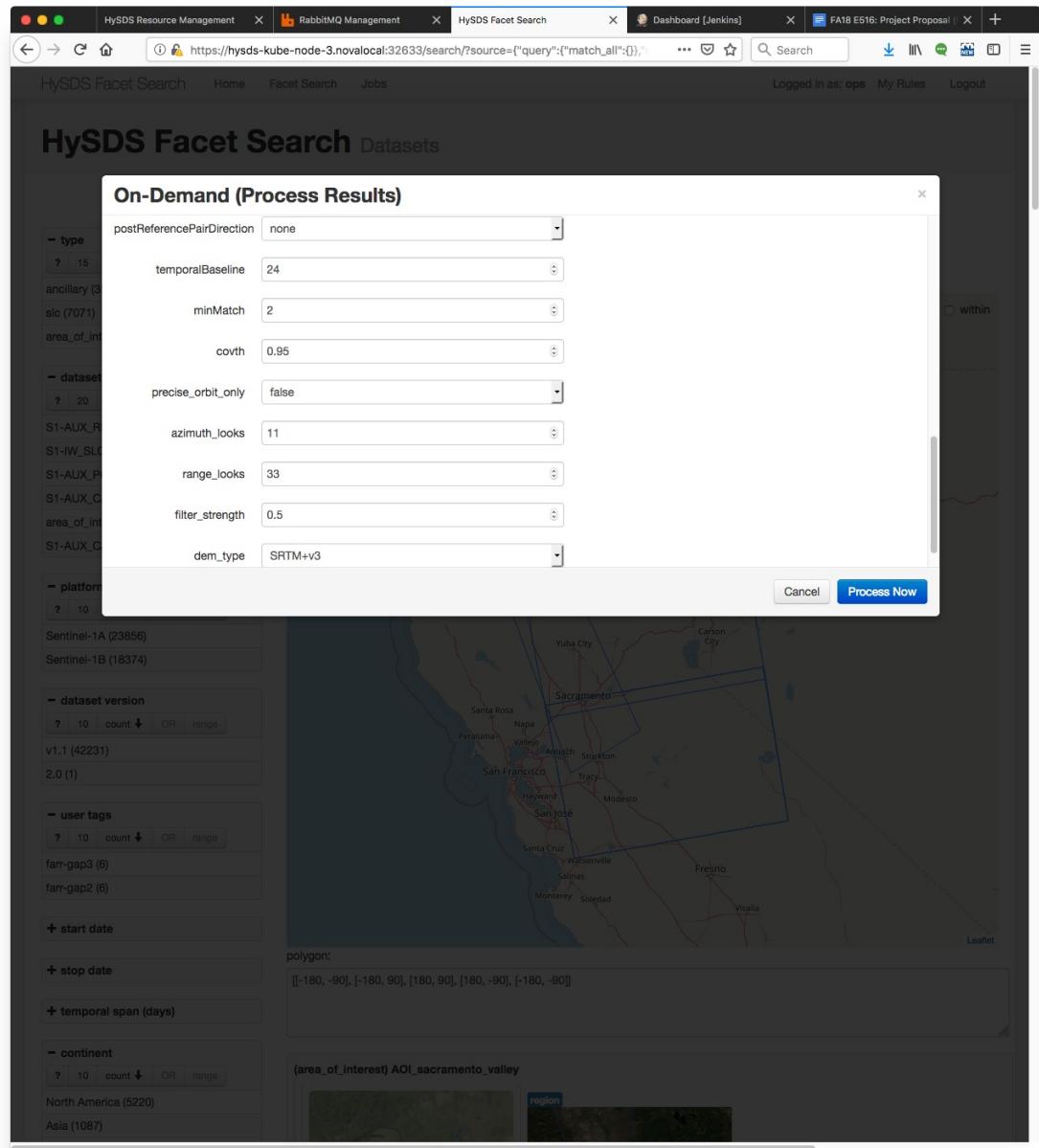


Figure 33: Submit Job

Alternately, the Mozart REST API is a Swagger/OpenAPI-compliant REST interface that can be used to submit jobs:

Mozart API v0.1

[Base URL: /mozart/api/v0.1]
<https://hyds-kube-node-3.novalocal:32708/mozart/api/v0.1/swagger.json>

API for HySDS job submission and query.

queue Mozart queue operations >

job_spec Mozart job-specification operations >

job Mozart job operations > ▾

- GET** /job/info Get complete info on submitted job based on id
- GET** /job/list Paginated list submitted jobs
- GET** /job/status Gets the status of a submitted job based on job id
- POST** /job/submit Submits a job to run inside HySDS

container Mozart container operations >

hysds_io HySDS IO operations >

event HySDS event stream operations >

Models > ▾

```

Job Status Response(JSON) ✓ {
    status*           string
                      example: job-queued
                      job status
                      Enum
                      > Array [ 6 ]
    message*         string
                      message describing success or failure
    success*         boolean
                      if 'false', encountered exception; otherwise no errors occurred
}

```

```

HySDS IO Addition Response(JSON) ✓ {
    message*         string
                      message describing success or failure
    result*          string
                      HySDS IO ID
    success*         boolean
                      if 'false', encountered exception; otherwise no errors occurred
}

```

```

Job Type List Response(JSON) >

```

Figure 34: Mozart REST API

3. Monitor the jobs that were submitted by navigating to the figaro interface:

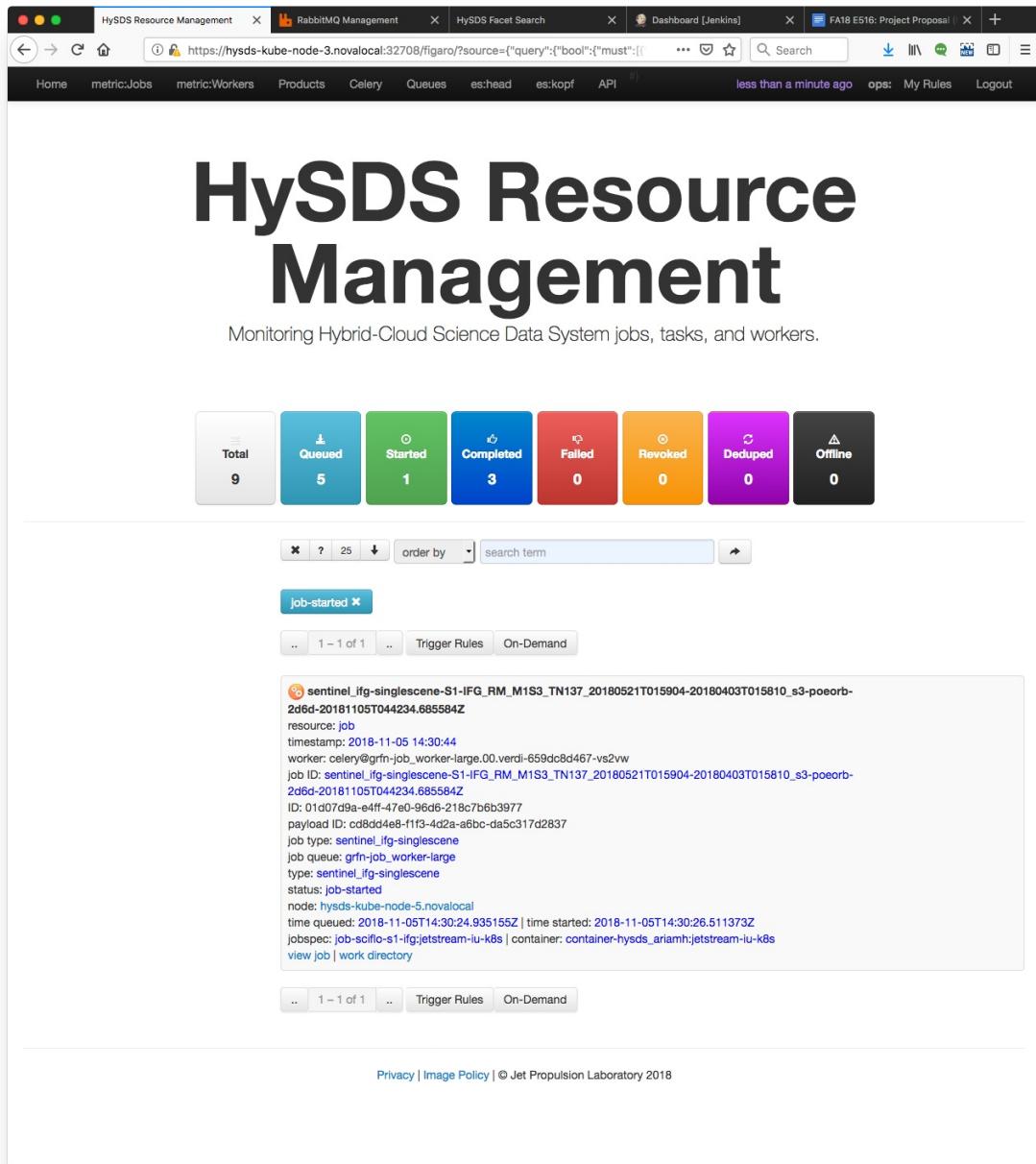


Figure 35: Figaro

The figaro interface provides a faceted view of the HySDS cluster's resource management. This includes job state information as well as information on tasks, workers, and events. Alternately, the RabbitMQ admin interface provides a real-time view of the job queues:

Name	Features	State	Ready	Unacked	Total	Message rates
						incoming deliver / get ack
aria-job_worker-large	D Pri	idle	0	0	0	
aria-job_worker-small	D Pri	idle	0	0	0	
dataset_processed	D Pri	idle	0	0	0	0.00/s 0.00/s 0.00/s
factotum-job_worker-large	D Pri	idle	0	0	0	
factotum-job_worker-small	D Pri	idle	0	0	0	
grfn-job_worker-large	D Pri	idle	5	1	6	0.00/s 0.00/s 0.00/s
grfn-job_worker-small	D Pri	idle	0	0	0	0.00/s 0.00/s 0.00/s
jobs_processed	D Pri	idle	0	0	0	0.00/s 0.00/s 0.00/s
system-jobs-queue	D Pri	idle	0	0	0	0.00/s 0.00/s 0.00/s
user_rules_dataset	D Pri	idle	0	0	0	0.00/s 0.00/s 0.00/s
user_rules_job	D Pri	idle	0	0	0	0.00/s 0.00/s 0.00/s
user_rules_trigger	D Pri	idle	0	0	0	

Add a new queue

HTTP API Server Docs Tutorials Community Support Community Slack Commercial Support Plugins GitHub Changelog

Figure 36: RabbitMQ Admin

4. When the Sentinel-1 interferogram jobs complete, they can be viewed via the tosca interface by facetting on the S1-IFG dataset type:

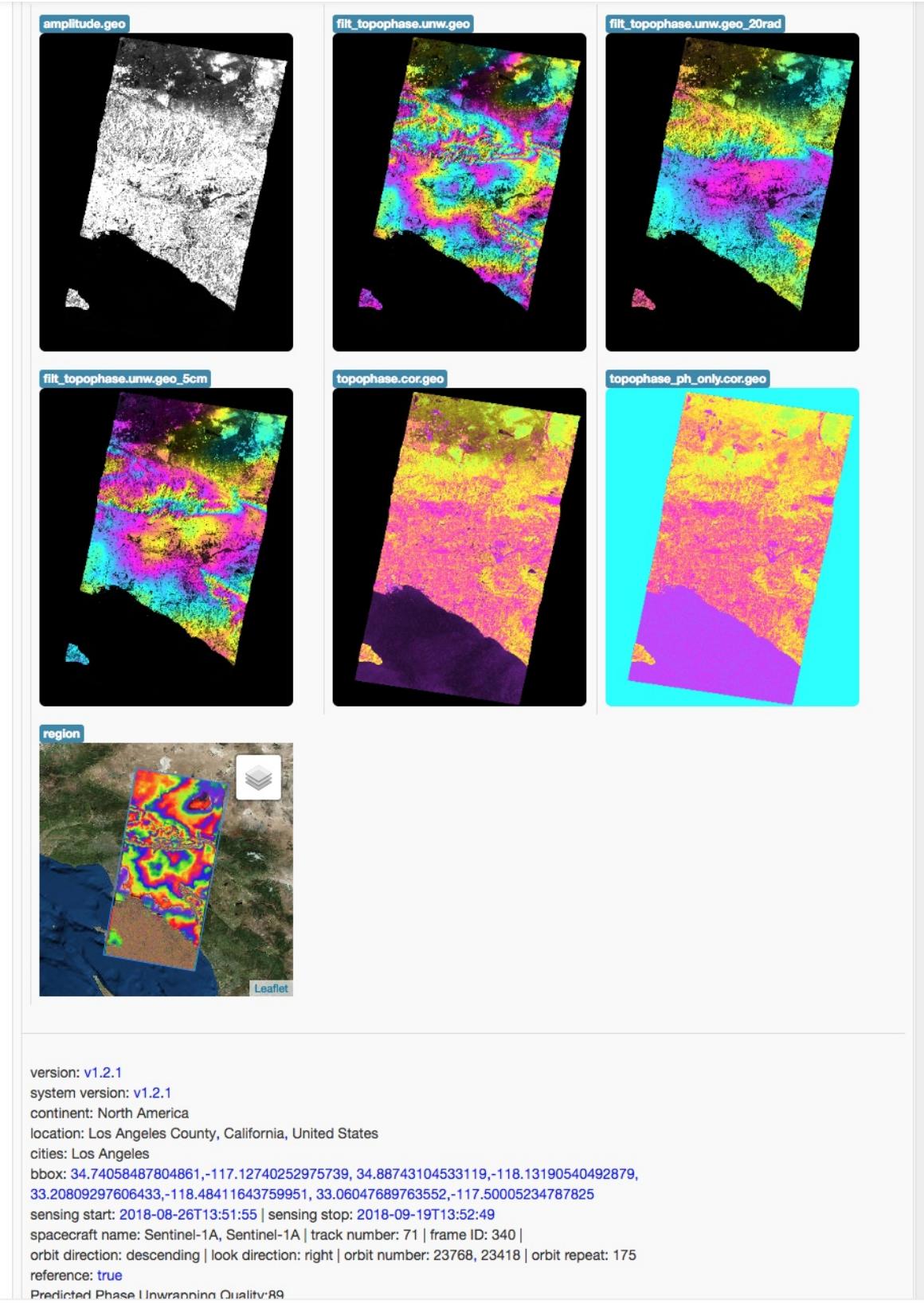


Figure 37: Sentinel-1 Interferogram

3.8 BENCHMARK

After running a few Sentinel-1 interferogram jobs, we can log into the Kibana job metrics interface on the metrics node to view the average execution time of these jobs:

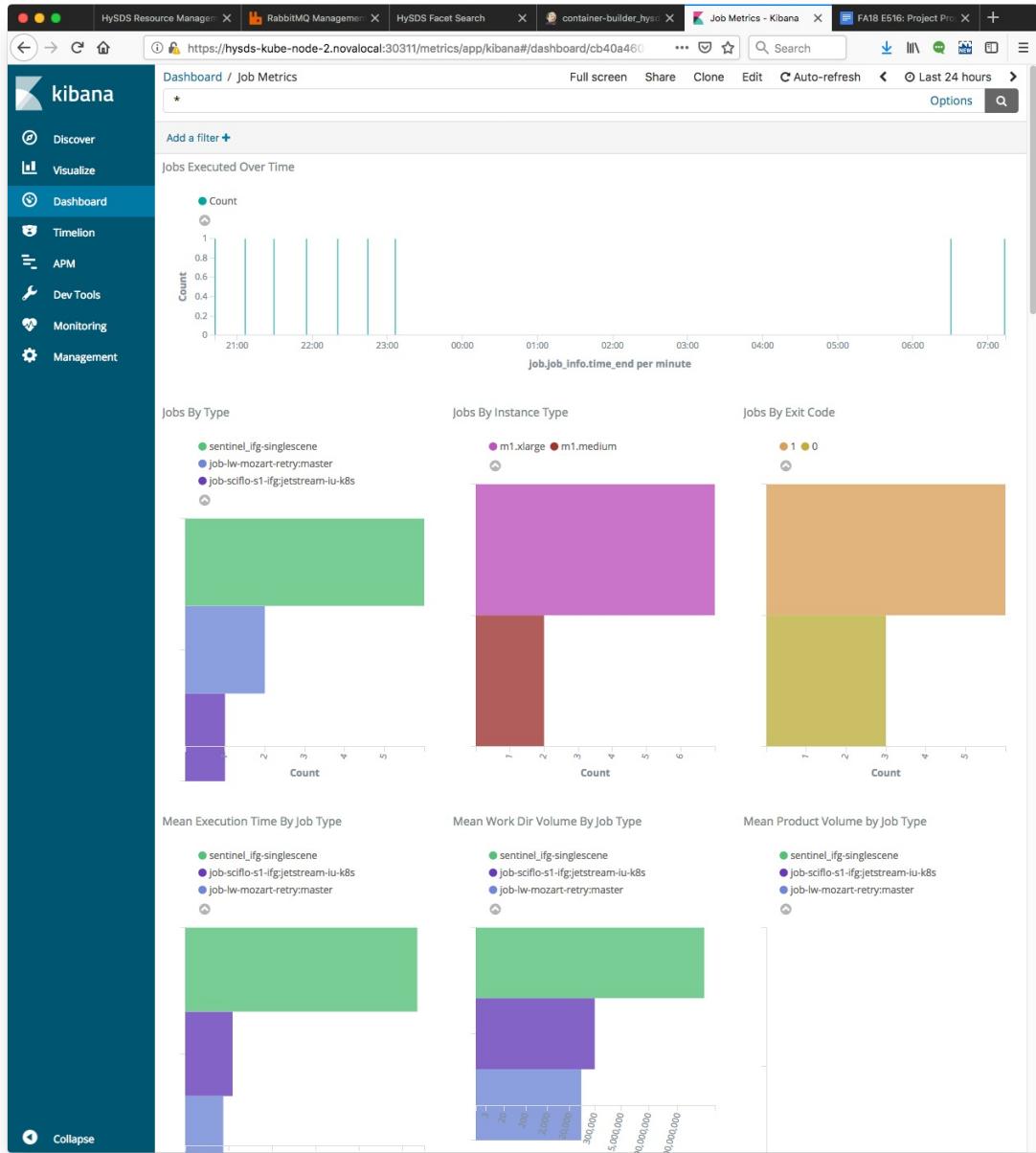


Figure 38: Sentinel-1 Interferogram Job Metrics

We note that it takes an average of 45 minutes to run each job on the HySDS/Kubernetes/Jetstream cluster. In comparison, the ARIA HySDS cluster on AWS takes an average of 25 minutes to run each job. Worker nodes on the ARIA HySDS cluster utilize the c5.9xlarge EC2 instance type (36 vCPUs and 72GB RAM) vs. the worker nodes on the HySDS/Kubernetes/Jetstream cluster which utilize the m1.xlarge flavor (24 vCPUs and 60GB RAM). This discrepancy shows that the ISCE PGE performs better when it is given more

vCPUs and RAM to work with.

3.9 CONCLUSION

The Hybrid Cloud Science Data System was developed in 2011 for the ARIA project to process SAR data to higher-level data products for the purpose of hazard monitoring and urgent response to earthquakes, floods, volcano eruptions, and other natural disasters. It was originally developed to run on OpenStack and later on AWS. As technologies develop and new cloud vendors put forth competitive pricing, there is a constant need to evolve HySDS to utilize these new technologies and cloud vendors. With the advent of container orchestration frameworks such as Kubernetes, HySDS can leverage their built-in support for interfacing with the underlying IaaS services of the various public and private cloud vendors. In this project, we re-architected the HySDS components to run on Kubernetes by decomposing them into Kubernetes pods and services. To test out running HySDS on a Kubernetes cluster on the IU Jetstream OpenStack cloud, we adapted the cluster to run the ARIA Sentinel-1 interferogram pipeline and successfully generated interferograms. In comparing the benchmark execution times of these jobs running on Jetstream infrastructure vs. those that run on AWS, we found that the ARIA worker nodes used a beefier instance type than those used on Jetstream and thus the jobs on the Jetstream cluster took almost twice as long to generate the products: 45 minutes vs. 25 minutes. Future work needs to be done to assess an apples-to-apples comparison of end-to-end HySDS cluster performance and ISCE PGE performance on various OpenStack instance flavors.

3.10 ACKNOWLEDGEMENT

“We thank the XSEDE help desk for assisting with providing access to the IU Jetstream OpenStack API and dashboard, which was made possible through the XSEDE Extended Collaborative Support Service (ECSS) program.”

4 SCALABLE DATA PROCESSING FOR RETAIL

Brad Pope
popebradleyt@gmail.com
Indiana University
hid: fa18-516-17
github: [blue link icon](#)
code: [blue link icon](#)

Learning Objectives

- Describe the unique data storage and compute needs of the Retail Industry
- Implement portions of a cloud computing environment that would meet those needs
- Describe the performance of the prescribed environment on a sample retail dataset
- Highlight some additional architecture opportunities outside the scope that would further help the Retail sector

Keywords: Hadoop, Hive, WebHCat, Java, Retail

4.1 INTRODUCTION

As with other industries, the retail industry today is undergoing a major shift. Traditional brick and mortar stores used to rely on practiced merchandising techniques such as printed flyers, television advertising, and low prices to drive traffic into stores. Today, shoppers are electing for convenience more than ever before. This is driving growth in a variety of time-saving Omni channel purchasing behaviors. Shoppers are voting with their wallets to have more product shipped to their door and for more convenient services such as grocery pick-up.

In tandem with a more demanding shopper, there is also a fundamental shift in

the sheer amount of data collected on the path taken to purchase products. For example:

- * Shoppers now research and initiate more purchases from their smart devices.
- * There are more shopper interaction data points captured in digital advertisements
- * Social media posts can have an impact on brand sentiment and individual product sales
- * E-commerce outlets sell and reselling products through a variety of channels

In short, there is new structured and semi-structured data available for retailers to leverage.

To survive in this changing landscape, retailers need to be open to change and use every advantage they have. Unfortunately, as technology evolved, each retailer responded differently and implemented a piecemeal data strategy to accommodate data elements as they become available. This legacy approach does not scale and ultimately puts them at a competitive disadvantage. We offer a cloud-based highly scalable distributed architecture that will allow retailers to reliably process a variety of data and integrate new data sets as they become available.

4.2 DATASET

In order to appreciate the storage and computational requirements, it is important to understand the idiosyncrasies of a retail data set.

Data available to retailers is heavily nuanced as each retailer collects and houses data in a different way. There are a host of internal metrics required in the ordinary course of daily business. In addition, there are a mounting number of external datasets now required to effectively compete. All of this makes for a challenge when it comes to acquiring, blending and actually using the data.

Retailer generated data (Internal data): This is data that the retailer creates during the normal course of business. This includes transactional data such as what product were sold in each store at any given time including what products sold together. At an operational level, it includes the purchase orders used to get more product to sell from suppliers, inventory levels in warehouses and stores. Operational data is also future looking with anticipated through merchandise forecasts. There are other data sets that track what products should be on

shelves, how much product should be there and where the product should go. In short, retailers internal data sets vary greatly in terms of the metrics gathered and how they are used which adds complexity.

- Granularity describes the level of depth of a dataset. On one side of the scale, transactional data sets are at a shopper, store, item, timestamp level of granularity and include important features such as what products are purchased together. Retailers often make operational data available at the store, product, day or week levels of granularity. For example, units per store per day is a normalized measure of how quickly a product sells in a given set of stores.
- Frequency describes how often a dataset is refreshed. Some measures are important operationally and refreshed continually. Other datasets are more static. For example, forecasts could be refreshed weekly, reference information like competitive stores could be updated monthly and exchange rates for planning purposes could be refreshed annually.
- Latency describes how much of a lag exists with a data set. For example, while a dataset may be updated daily it may have two or three-day latency to get it from the stores where the product is selling. Often times, it takes that much time to flow it to the central repository for further processing and reporting.
- Restatement or trickle data occurs when updated data becomes available for past time periods. When this occurs processes need to be in place to remove previous data with the updates. Controls or versions of data should be in place for critical numbers such as actual sales values that get reported to Wallstreet for their shareholders.

Externally generated data: Not all data that a retailer uses comes from their internal systems as it only gives partial insight to shoppers and competitors. To supplement internal data, retailers are continually assessing shopper preferences. In order to do this many subscribe to services offered by IRI, Nielsen, and InfoScout. This type of data allows them to understand who their shoppers are and if they are getting their fair **share of wallet** or percent of purchases of each category were made in their locations. It also allows them to see what other products those shoppers buy and which of their competitors those shoppers are purchasing them.

There are several external sources of data that have recently entered the market.

- * Social Media: Insights from social media have become highly sought after to understand and influence shopper sentiment.
- * Omnichannel: Omnichannel and path to purchase data varies greatly and can hold insights as to how consumers shop for certain products and brands.
- * E-commerce: Knowledge of purchases made online is desirable to understand which products have better potential to sell online.

The variety and amount of data available in the datasets make pulling and storing the data a challenge. This challenge is exacerbated by the need to blend the vastly different internal and external data together.

4.2.1 Dataset.

Here is a description of the sample dataset:

- Store – Values here represent individual stores. The number of stores varies greatly by retailer. The dataset included 300 stores. Larger retailer chains can have more than 10,000 stores internationally.
- Product – This is a proxy for a product UPC or Item Number. Retailers will sell thousands of products in any given retail location. For our purposes here 23 products were included.
- Period_Key – Date information in a YYYYMMDD format. Many retailer datasets have two years of history available. This dataset contains daily data from 10/23/2015 through 10/15/2017 which equates to 723 unique dates.
- Sales Dollars – US Dollars associated with the sales for each product, store and period.
- Sales Units – Number of units sold for each product, store and period.
- Potential Demand – Potential revenue associated with having the product available for sale (no out of stocks).

4.3 IMPLEMENTATION

Diagram of the overall solution is shown in Figure [39](#).

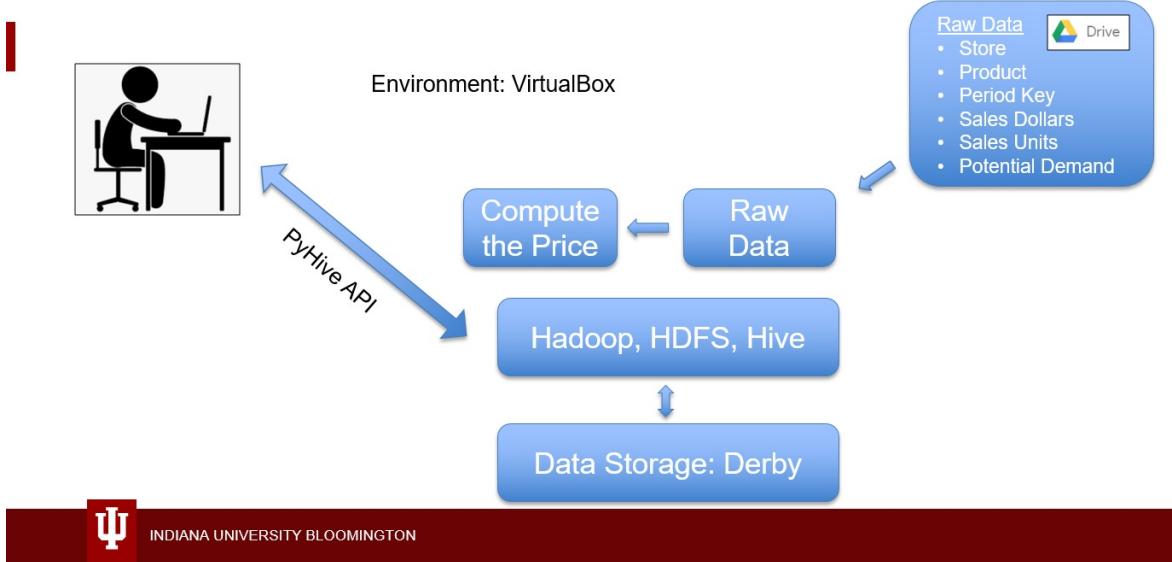


Figure 39: Retail Project Diagram

4.3.1 Hadoop

Apache Hadoop for retailers was chosen for several reasons. First, Hadoop's capabilities are well aligned with the needs of retailers. According to the home page of the Hadoop project at Apache Software Foundation, ***Apache Hadoop offers highly reliable, scalable, distributed processing of large data sets using simple programming models. With the ability to be built on clusters of commodity computers, Hadoop provides a cost-effective solution for storing and processing structured, semi and unstructured data with no format requirements*** [22].

First and foremost, the cost advantages associated with open source software should not be overlooked. Retailers have razor thin margins. There is stiff competition from online retails and a host of aggressive discount retailers such as Aldi, Family Dollar and Lidl with low-cost private label products. In order to save cost and maintain their margin, saving money on infrastructure is critical for retailers.

Retailers also need reliable data processing and distribution at scale. Operational data is the lifeblood of a retailer. If the computer-assisted order system does not have reliable inventory, sales and forecast information from a store, it will not be

able to issue purchase orders to suppliers and keep the product in stores for ongoing business. Rather than relying on the hardware to rely on redundancy and high-availability, Hadoop detects and handles failures at the application layer and delivers a high-availability service on top of a cluster of computers to avoid a whole system failure [23].

The application layer also allows for adding more datasets as they become available which is a key retailer need. The Hadoop framework scales from processing on a single server to thousands of machines and uses the computation and storage available on each. This ability to scale coupled with the flexibility to add new formats of semi and unstructured data is important new data is continually becoming available to retailers [22].

4.3.2 HDFS

The Hadoop Distributed File system (HDFS) is Hadoop's proven ability to distribute very large files on a cluster of commodity hardware [24].

For a retailer with thousands of stores and thousands of products capturing all of the data necessary results in petabytes of data. In addition, data is constantly being created as products move in the supply chain, sell in the stores and forecasting and planning are continually being processed in the background. HDFS accommodates these demands.

First, it is possible to store files of any size on HDFS. The distributed file system breaks down files that potentially petabytes in size into smaller pieces or blocks and each piece could be stored on different machines if needed. In this system, an HDFS master node known as a NameNode and a slave node is called a DataNode. NameNodes maintain and manage all information about all files and directories stored on HDFS including the file system tree and metadata. DataNodes are the actual storage for the blocks. NameNode sends blocks to the DataNodes to store and DataNodes continuously report their status back including the list of blocks they are storing. This makes it possible to store files of any size on a Hadoop cluster.

For this implementation, we are keeping the NameNode and the DataNode on the same VirtualBox with the set of sample data. As a future consideration, the DataNodes would be distributed across many servers. In addition, to fully

leverage HDFS, the retailer data would be stored in larger files instead of millions of smaller files resulting in the NameNode using less memory.

Commodity Hardware means using standard commonly used hardware without the need to specialized high-end systems. As discussed earlier, less cost is important for retailers so being able to use more common server configuration systems to build a reliable cluster with HDFS results is extremely important.

4.3.3 Hive

“Since its incubation in 2008, Apache Hive is considered the defacto standard for interactive SQL queries over petabytes of data in Hadoop. Data analysts use Hive to query, summarize, explore and analyze that data, then turn it into actionable business insight” [25].

Hive works well with HDFS because it organizes the data into databases, tables, buckets and clusters. Partitioning and bucketing these tables allows for efficient storage and data retrieval. This abstract structure allows the system to only load the relevant parts of the table during query processing. Querying less data results in faster query execution times [26].

Just like in other industries, Retail IT professionals often have SQL skills so using the HiveQL which is very similar to SQL would be beneficial for adoption. In addition, the hive structure will do an effective job pruning the large datasets that a retailer has to just the relevant measures and attributes needed in a report or analysis.

Since this is the case, SQL against Hive to compute the average price was leveraged here. As a future consideration with a larger dataset more of the Hive partitioning and bucketing features should be leveraged.

4.3.4 Data Storage

Hadoop and HDFS require a database and, given the smaller sample dataset, Derby was implemented. Derby has a small footprint and relatively straightforward to install, deploy, and use. It also supports a client server model and is based on Java which is already a requirement of Hadoop [27].

As unstructured data is added to Hadoop and HDFS, a future consideration is to use Azure blob storage with HDInsight. Azure blobs are capable of storing a variety of types of files from documents to database backups. Similar blobs are stored in containers. Both containers and blobs do not have hard size limitations.

“Azure Blob storage can be accessed from Hadoop (available through HDInsight). HDInsight can use a blob container in Azure Storage as the default file system for the cluster. Through a Hadoop distributed file system (HDFS) interface provided by a WASB driver, the full set of components in HDInsight can operate directly on structured or unstructured data stored as blobs” [28].

4.3.5 API

There needs to be a way to access the data through an API. As a simpler implementation with python, PyHive is the logical API for illustrative purposes with Hive. Per the Python Software Foundation, Pyhive is “a collection of Python DB-API and SQLAlchemy interfaces for Presto and Hive” [29].

Since the Hadoop, HDFS and HIVE implementation will get complex as different types of data are added, a future consideration is to use WebHCat. WebHCat is a REST API for HCatalog which is the storage management layer for Hadoop. WebHCat also works well with HDInsight making it ideal for an Azure Blob Microsoft implementation. In addition it can interact with more functionality of Hadoop than PyHive.

4.4 BENCHMARK

Using the components above, the calculation for the average price was run on VirtualBox on a local computer with 8MB of RAM. The query itself took 13.1 seconds.

4.4.1 Query and Results

```
Select Item_Key,(POSSales/POSQty) AS AverageRetail FROM (Select Item_Key, sum(POSSales) AS POSSales, sum(POSQty) AS POSQty From
```

```
retaildata GROUP BY Item_Key) byitem ORDER BY Item_Key;
```

The query was run for the average price by store as shown here Figure [40](#).

```
Stage-Stage-2: HDFS Read: 0 HDFS Write: 0 SUCCESS
Total MapReduce CPU Time Spent: 0 msec
OK
1000    6.028177014042909
1001    5.660284780348014
1002    2.425488760155939
1003    7.640841750841222
1004    4.011295764717158
1005    3.6202413320276547
1006    5.797210599721253
1007    3.2290747450916584
1008    5.40005158107212
1009    3.3884679138053797
1010    4.116228383731125
1011    6.332208862600638
1012    3.975876288659689
1013    4.807084735841173
1014    7.415121592073101
1015    10.19820711974094
1016    3.590010245901411
1017    5.629701413427958
1018    4.778877044661191
1019    10.024481382978573
1020    5.524168320762203
1021    5.808968291528611
1022    3.186986834231194
Time taken: 13.107 seconds, Fetched: 23 row(s)
hive>
```

Figure 40: Average Price Results

4.5 CONCLUSION

We've described how retailer data needs are continually expanding. In order to meet those needs, we've shown that it will take a reliable, scalable architecture at a lower price point. The cloud computing model delivered through a combination of Hadoop, HDFS, and Hive described above meets those needs.

We've successfully implemented portions of this environment and illustrated the performance on a sample retail dataset. In addition, several future considerations were called out that will enhance and scale the architecture as additional complexity is added.

5 MANAGE FILES ACROSS CLOUD PROVIDERS

Richa Rastogi

rirastog@iu.edu

Indiana University

hid: fa18-516-18

github: [RichaRastogi](#)

Keywords: Multi-cloud data service, Cloud Computing, Python, Open API, Cloud Providers, MongoDB, Swagger

5.1 ABSTRACT

The goal of this project is to manage files across different cloud providers. There are many cloud providers where we can store data in form of files like Amazon AWS, Microsoft Azure, Google cloud, etc. Here we are going to build an OpenAPI to manage these files, operations like copy, upload, download or delete from any provider. This system is self sufficient to work as a file manager.

5.2 INTRODUCTION

The objective of this project is to manage data across different cloud providers. We are going to build an RESTFUL OpenAPI for managing the data between all the cloud storages. We will analyse how these different clouds work and then build python methods to handle data across them. Final step will be to expose these functionalities as a RESTFUL API. This way we can also take advantage of cloud providers for cheaper solutions for storage by dividing the data across them. Since this project has its own MongoDB and User profiling so it can be used a file manager in itself.

5.3 REQUIREMENTS

This project requires knowledge about Cloud Providers like AWS, Azure,

Google Cloud, etc. * This project is using Amazon AWS and Google cloud as two cloud providers and their storage functionality. We can expand this * This also needs a database so we are using MongoDB through MongoEngine and store the files and User data in it. * Overall functionality can be accessed through console or RESTFUL OpenAPI which is built using Flask and Swagger.

5.4 ARCHITECTURE

Please refer to below diagram #fig:fa18-516-18_Architecture.png for architecture of this project.

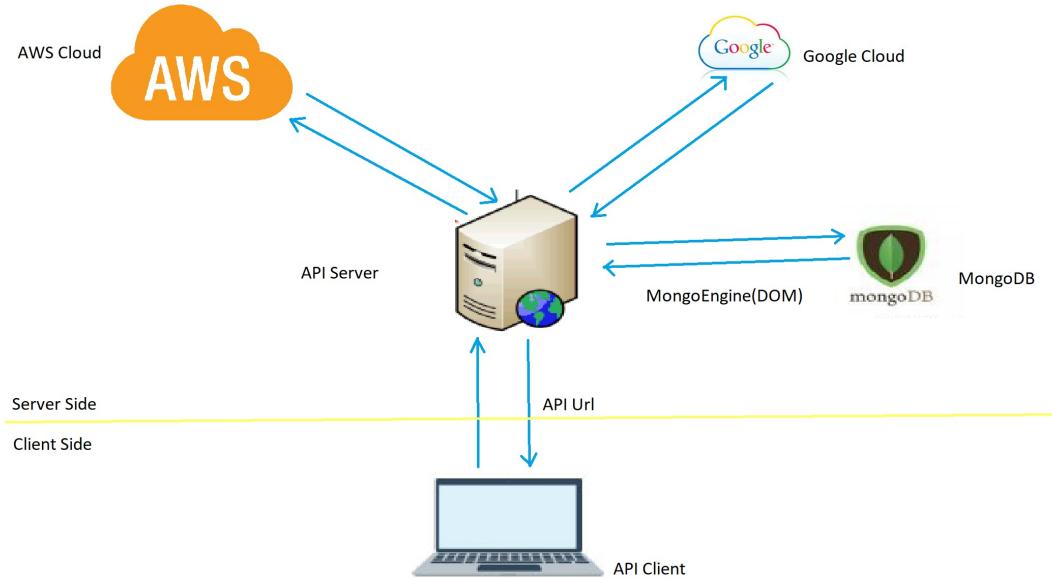


Figure 41: Architecture

5.5 DESIGN

This project involves developing a RESTFUL API to manage files. We can perform following operations like upload, download, list, copy, rsync and delete. We can also use this project to store files in MongoDB and assign specific User Role permissions to access the files.

- The very first thing required for that is to create accounts in AWS and Google(Since those are the two providers we are using).
- Then use their API Keys to connect to these providers through Python code.
- After we have a connection, we use their APIs to access the bucket by providing the name through Yaml file/API input.
- Now since we have the bucket, we can list, download or upload the files.
- There is Command console from where we can execute these functions directly using console or script file.
- On top of that there is an open API built to perform these functions using REST.
- We also have MongoDB storing the downloaded files.

5.6 IMPLEMENTATION

This project is using following technologies for implementation:

- * Python 3.7.0 for python code development
- * Swagger 2.0 for writing API specification. This specification describes REST endpoints for managing files across providers.
- * Python flask framework which consumes the OpenAPI specification and directs the endpoints to Python functions by building a RESTful app.
- * MongoEngine as a Document-Object Mapper for working with MongoDB from Python.

Enable a virtual environment so that all installations can be done specifically in that env.

```
python3 -m virtualenv /home/richa/venv/      //to install venv  
source /home/richa/venv/bin/activate        //to activate venv
```

Now my console looks like:

```
(venv) (3.7.0) richa@richa-VirtualBox:~$
```

5.6.1 AWS access from Python:

- Install apache-libcloud by “pip install apache-libcloud”
- Follow instructions to create an AWS account - <https://github.com/cloudmesh-community/book/blob/master/chapters/iaas/aws/aws.md>
- Select S3 from Services and create a bucket
- To access this bucket, go to IAM and create a user and then create a new Access Key in “Security Credentials”
- While creating this key, system will prompt to download pem file. Save that pem file onto your working machine.

5.6.2 Google Cloud Platform:

- Install “pip install google-cloud-storage”
- pip install google-auth google-auth-httplib2
- pip install –upgrade google-api-python-client
- Create an account on Google Cloud by going to <https://cloud.google.com/>
- Create a new Project from the top of the page.

- Create a new storage bucket in google cloud, select Storage -> Storage -> Browser
- To access this bucket now, follow <https://cloud.google.com/storage/docs/reference/libraries>
- This will download a JSON file in your working VM and use that file for authentication to access Google Cloud Storage.

All the dependencies can be installed easily by running requirements.txt inside project-code so no need to do any pip install.

```
pip install -r requirements.txt
```

AWS and Google Cloud specific functionality python files are under directory structure project-code/cloudmesh/data. cloudmesh-data.yaml is the yaml file holding all the information about these cloud setup. Aws_setup.py and google_cloud_setup.py uses this yaml file to authenticate the cloud providers and setup the connection to the cloud services.

It also has command.py under here to run the functionality from console passing in relevant input. To execute commands from console using cmddata commands, we need to setup cmddata by running in project-code dir:

```
pip install .
```

Now we can run all cmddata commands as given below. We can also test if cmddata is working by running a test command:

```
cmddata test
```

Usage:

```
cmddata data list [--format=FORMAT]
cmddata set provider=PROVIDER
cmddata set dir=BUCKET
cmddata data add PROVIDER FILENAME
cmddata data get PROVIDER FILENAME USER_UUID
cmddata data ls PROVIDER
cmddata data copy FILENAME PROVIDER DEST
cmddata data rsync FILENAME SOURCE DEST
cmddata data del PROVIDER FILENAME
cmddata update user USER file FILENAME
cmddata (-h | --help)
cmddata --version
```

Options:

```
-h --help      Show this screen.
--version     Show version.
--config      Location of a cmddata.yaml file
```

Description:

```
cmddata data ls PROVIDER
    Description: CM command to list all the files in a Provider's bucket

cmddata data add PROVIDER FILENAME
    Description: CM command to upload a file from local directory to the Provider's
bucket

cmddata data get PROVIDER FILENAME USER_UUID
    Description: CM command to download a file from the Provider's bucket to a local
directory
        and then save that file to MongoDB with the username assigned

cmddata data copy FILENAME PROVIDER DEST
    Description: CM command to copy a file from one Provider's bucket to another

cmddata data del PROVIDER FILENAME
    Description: CM command to delete a file from a Provider's bucket
```

Example:

```
cmddata data ls google_cloud
cmddata data add google_cloud abc.txt
cmddata data get google_cloud abc.txt richa
cmddata data copy xyz.txt AWS GOOGLE
cmddata data del google_cloud abc.txt
```

We also have MongoDB installed to save the downloaded files into the database. We are using MongoEngine as Document-Object Mapper to add records and save the file as a FileField into DB. File is stored into MongoDB using GridFS.

5.6.3 MongoEngine GridFS

GridFS is a specification for storing and retrieving files into MongoDB.

Instead of storing a file in a single document, GridFS divides the file into parts, or chunks, and stores each chunk as a separate document. By default, GridFS uses a default chunk size of 255 kB; that is, GridFS divides a file into chunks of 255 kB with the exception of the last chunk. The last chunk is only as large as necessary. Similarly, files that are no larger than the chunk size only have a final chunk, using only as much space as needed plus some additional metadata.

GridFS uses two collections to store files. One collection stores the file chunks, and the other stores file metadata. The section GridFS Collections describes each collection in detail.

When you query GridFS for a file, the driver will reassemble the chunks as needed. You can perform range queries on files stored through GridFS. You can also access information from arbitrary sections of files, such as to “skip” to the middle of a video or audio file.

GridFS is useful not only for storing files that exceed 16 MB but also for storing any files for which you want access without having to load the entire file into memory. See also When to Use GridFS.

This File database table structure is read from project-code/file.yml definitions and it has a primary key as the name of the file so that we can search based on this field. This also has user_uuid field to provide specific user access to the files.

```
class File(Document):
    name = StringField(primary_key=True)
    endpoint = StringField()
    checksum = StringField()
    size = StringField()
    timestamp = DateTimeField(default=datetime.datetime.now)
    last_modified = DateTimeField(default=datetime.datetime.now)
    user_uuid = StringField()
    file_content = FileField()
```

Similarly we have a User table to store Users with their roles and group.

This project also has RESTFUL APIs to perform all the above operations and their Swagger UI looks like below. For File APIs, please refer to screenshot below for Swagger UI for File APIs (refer to FileSwaggerAPI.png).

file

A file is a computer resource allowing storage of data that is being processed. The interface to a file provides the mechanism to appropriately locate a file in a distributed system. File identification includes the name, endpoint, checksum, and size. Additional parameters, such as the last access time, could also be stored. The interface only describes the location of the file. The file object has name, endpoint (location), size in GB, MB, Byte, checksum for integrity check, and last accessed timestamp.

default		Show/Hide List Operations Expand Operations
GET	/file	
POST	/file	
POST	/file/copy	
DELETE	/file/delete	
POST	/file/rsync	
GET	/files	

[BASE URL: /cloudmesh_data , API VERSION: 1.0.0]

Figure 42: FileSwaggerAPI

API Path	Type	Description	Input Parameters
/cloudmesh/files? service={provider}	GET	Returns all files from a specific provider	Query Param: Provider name
/cloudmesh/file? service= {provider}&filename= {filename}&user_uuid= {user}	GET	Returns a file for a specifc provider	Query Param: Provider Name, filename, user_uuid
/cloudmesh/file? service= {provider}&filename= {filename}	POST	Upload a file to a provider	Query Param: Provider name,filename
/cloudmesh/file/copy? service= {provider}&filename= {filename}&dest= {destination}	POST	Copy a file to a provider	Query Param: Filename, Provider, destination
/cloudmesh/file/rsync? service= {provider}&filename= {filename}&dest=	POST	Rsync a file to another directory	Query Param: Filename, Provider,

{destination}	destination
/cloudmesh/file/delete? service= {provider}&filename= {filename}	DELETE Delete a file from a directory Query Param: Filename, Provider

For User APIs, please refer to screenshot below for Swagger UI for User APIs (refer to UserSwaggerAPI.png).

user 1.0.0

[Base URL: /cloudmesh]

User profiles are used to store information about users. User information can be reused in other services. This is useful to access files and upload or download them to cloud machines. Profiles can be added, removed and listed. A group in the profile can be used to augment users to be part of one or more groups. A number of roles can specify a specific role of a user.

The screenshot shows the 'profile' endpoint under the 'user' API. It lists three methods:

- GET /user/profile**: Returns all profiles.
- PUT /user/profile**: Create a new profile. Description: Create a new profile.
- GET /user/profile/{uuid}**: Returns the profile of a user while looking it up with the UUID.

Figure 43: UserSwaggerAPI

API Path	Type	Description	Input Parameters
/cloudmesh/user/profile	GET	Returns all profiles	NONE
/cloudmesh/user/profile	PUT	Create a new profile	Body: Profile Object
/cloudmesh/user/profile/{uuid}	GET	Returns the profile of a user while looking it up with the UUID	Path Param: UUID

5.7 DATASET

REST API Output for getting files list (refer to `FileListAPIOutput.PNG`) API URL - http://localhost:5000/cloudmesh_data/files?service=google_cloud

```

localhost:5000/cloudmesh_ X  Swagger UI  X | NameError: name 'file' is X | +
← → ⌂ ⌄ localhost:5000/cloudmesh_data/files?service=google_cloud
JSON Raw Data Headers
Save Copy Collapse All Expand All
results:
  0:
    0:
      Filename: "renters-insurance-2019.pdf"
      SNo: 1
  1:
    0:
      Filename: "test/"
      SNo: 2
  2:
    0:
      Filename: "test/MapReduce.docx"
      SNo: 3
  3:
    0:
      Filename: "test/User Guide.pdf"
      SNo: 4

```

Figure 44: FileListAPIOutput

Database records for File table. This shows that file_content is stored in another table as per GridFS described above in fs.chunks and fs.files:

```

{'_id': 'MapReduce.docx', 'endpoint': 'AWS', 'checksum':
'2c716d77f0916df41147f16c05c91e10', 'size': '149.5 KB', 'timestamp':
datetime.datetime(2018, 12, 2, 14, 45, 32, 564000), 'last_modified':
datetime.datetime(2018, 12, 2, 14, 45, 32, 564000), 'user_uuid': 'richa', 'file_content':
ObjectId('5c04608cf8724304b52eac8a')}
{'_id': 'aws_lambda.png', 'endpoint': 'AWS', 'checksum':
'a73e3d8449ab6e7366a7b1e7f24dab35', 'size': '99.6 KB', 'timestamp':
datetime.datetime(2018, 12, 2, 14, 46, 9, 241000), 'last_modified':
datetime.datetime(2018, 12, 2, 14, 46, 9, 241000), 'user_uuid': 'richa', 'file_content':
ObjectId('5c0460b1f87243053dd214de')}

```

Database records for User table.

```
{'_id': '11111', 'username': 'richa.rastogi', 'group': 'test', 'role': 'test', 'resource': 'test', 'context': 'test', 'description': 'test', 'firstname': 'richa', 'lastname': 'rastogi', 'publickey': 'test', 'email': 'rrastogi@iu.edu'}
```

5.8 CONCLUSION

The main objective of this project was to gain knowledge and understanding of different cloud providers and create RESTFUL APIs using OpenAPI architecture using either flask or Eve and then to use MongoDB database to manage this data coming in from cloud providers. There were several other technologies been used to bring this whole project together.

This project can be enhanced even further by including many other cloud providers like Openstack, Azure etc. and all these clouds can perform operations within themselves which can reduce costs for certain people since they will not be exceedingly dependent on only just one provider. Since this project has its own access and database system enabled so this can be used as a file manager in itself.

5.9 ACKNOWLEDGEMENT

I am very thankful to Professor Gregor von Laszewski for helping me throughout this project development as I am new to all the technologies used in this project. I also took help from nist and cm projects to understand the OpenAPI development using flask.

5.10 REFERENCES

- [30] <https://github.com/cloudmesh-community/nist/tree/master/services>
- [31] <https://github.com/cloudmesh-community/cm/tree/master/cm4>

6 CLODMESH GRAPHQL APP

Mihir Shanishchara, Vineet Barshikar
mshanish@iu.edu, vbarshik@indiana.edu
Indiana University
hid: fa18-516-21 fa18-516-02
github: [cloudmesh](#)
code: [cloudmesh](#)

Keywords: GraphQL, Cludmesh client, mongoengine, Flask, Electron

6.1 ABSTRACT

Cludmesh cm4 is an ongoing project worked upon by entire class to create a network of computers that can run parallel jobs. Currently it accepts commands via command line. Our project provides an user interface to Cludmesh cm4. We implemented a client-server application which will communicate through GraphQL APIs.

6.2 INTRODUCTION

Our aim with this project was to learn GraphQL and gain deep understanding into how effectively it can be used in place of Rest APIs. Along with this we also wanted to contribute to ongoing Cludmesh cm4 project. Currently cm4 accepts commands via command line. And thats when we realized that we could use GraphQL and create a user interface which would translate user actions into cm4 commands and in turn execute those commands.

Our project has pages for both list of Virtual Machines and list of Images. User can start, restart, shutdown any VM or image. They can also mark them as their favorites so that they appear at the top for them.

We have implemented lazy loading or infinite scrolling due to which irrespective of how big the list of Virtual Machines or Images be, it loads them in batches

which improves page rendering performance.

More info about GraphQL is available as a chapter in the cloud computing handbook.

6.3 REQUIREMENTS

- A cross platform desktop application which can be redistributed to users
- Client App should show data from MongoDB using GraphQL APIs
- Client App should send user action to server and mutate data
- Client and server should be able to handle more than 10000 VMs

6.4 ARCHITECTURE

Figure 45 shows architecture of Cloudmesh GraphQL app.

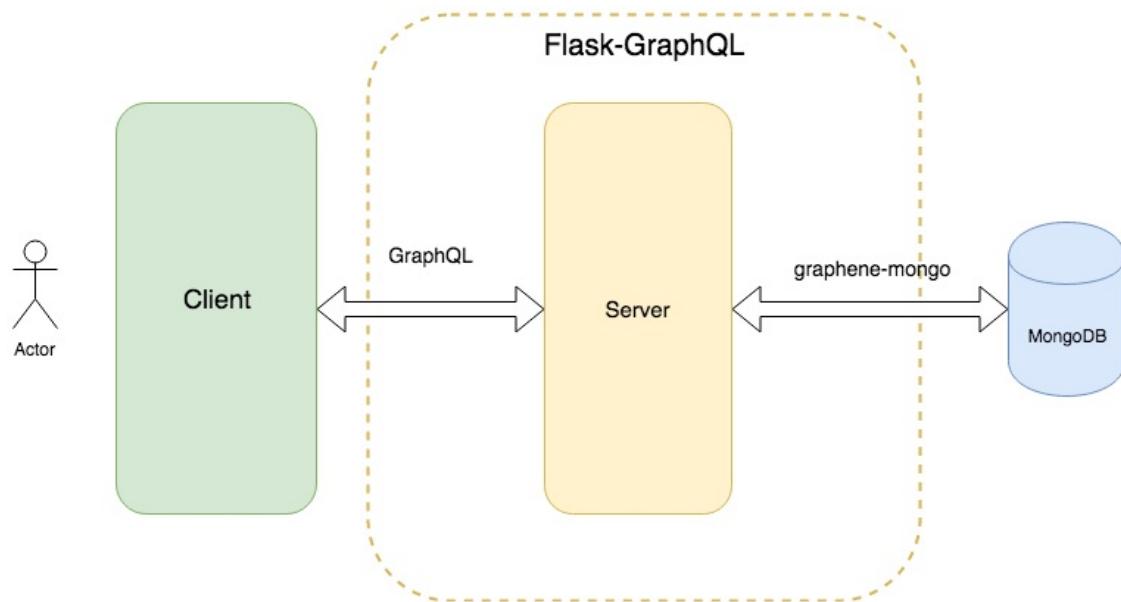


Figure 45: Cloudmesh GraphQL Architecture

Cloudmesh App is divided in two parts

- **Client App:** Client app can be distributed to the users. This app will provide a simple interface to the user, using which user can execute all commands provided by cm4. Client app will communicate with

GraphQL server and based on input from the user, server will execute commands.

- **GraphQL server:** This GraphQL server will be running on one cloud instance to which all client apps can connect.

Client App is designed using following technologies

- ElectronJS [32] : Using ElectronJS we can build cross platform desktop apps with JavaScript, HTML and CSS. ElectronJS combines power of native apps with beautiful web interface.
- BackboneJS [33] : BackboneJS provides an MVC structure with models, collections and views. For code reusability views have been divided into two categories
 - Smart View: Knows how to communicate with server but doesn't know about representation
 - Dumb View: Doesn't know how to communicate with server but knows how to render data
- HandlebarsJS [34]: Handlebars provides set of functions which lets us build generic HTML templates easily. It also provides a way to extend helper function and create custom helpers to use in templates.
 - All custom helper functions for templates are defined at utils/helpers space
- jQuery [35]: jQuery provides set of functions which are very useful for DOM manipulation
- Material UI [36]: Material UI is open source design spec which is mainly developed by Google. We are using web component implementation of Material UI.
- Webpack [37]: Webpack is a module bundler and also it provides a way to specify loaders for different file types. For example handlebars loader is used to load and compile handlebar template before creating bundle.

GraphQL server is designed using following technologies

- Python 3 (Please refer the handbook for introduction)
- Flask (Please refer the handbook for introduction)
- Graphene (Please refer the handbook for introduction)
- Flask-GraphQL [38]: Flask-GraphQL adds support for GraphQL to the flask application.
- Graphene-Mongo [39]: An integration of graphene and mongoengine.

6.5 DATASET

We used python's 3rd party library ***faker***[40] to generate fake data for testing.

6.6 IMPLEMENTATION

Please refer [README.md](#) available under project code to install and start app.

Figure [46](#) shows login page of app

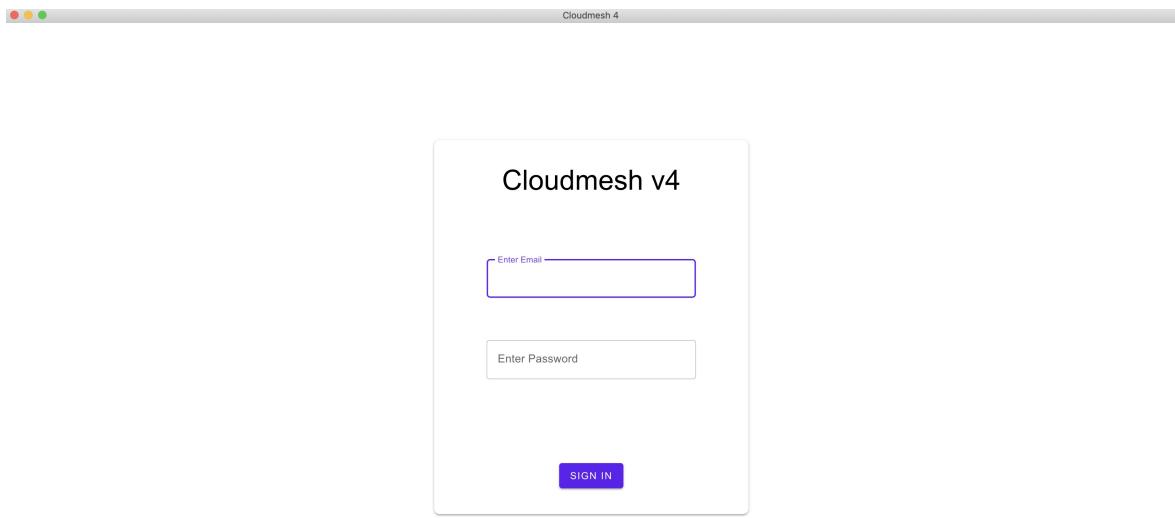


Figure 46: Login Page

Since authentication is not requirement for this project and its already implemented with GraphQL example in handbook, we have not implemented it in project. On click of **Login** button user will see dashboard, which is empty as of now.

Figure [47](#) shows the view rendered on click of **VMs** button

User Name
email@user.com

Cloudmesh

- Home
- Profile

Clouds

- VMS**
- Images
- Register

Settings

Help & feedback

VM List

AWS AZURE CHAMELEON CLOUD GOOGLE CLOUD

UPDATE VM DATA

SORT BY GRID LIST

AWS	AZURE	CHAMELEON CLOUD	GOOGLE CLOUD
lt-35.davis-green.info Region : us-east-1 Image : CC-Ubuntu16.04	srv-06.russell.info Region : us-west-2 Image : CC-Ubuntu12.04	lt-62.brock-patterson.com Region : us-east-1 Image : CC-Ubuntu18.04	srv-41.king.info Region : us-west-2 Image : CC-Ubuntu12.04
srv-73.ware.info Region : us-gov-east-1 Image : CC-Ubuntu12.04	db-14.contreras.biz Region : us-gov-east-1 Image : CC-Ubuntu18.04	email-40.huffman.info Region : us-west-2 Image : CC-Ubuntu12.04	lt-13.wilson.com Region : us-gov-east-1 Image : CC-Ubuntu14.04
lt-11.holland-williams.info Region : us-west-1 Image : CC-Ubuntu14.04	srv-17.butler.com Region : us-east-2 Image : CC-Ubuntu14.04	desktop-99.lyons.info Region : us-gov-east-1 Image : CC-Ubuntu16.04	web-81.day-smith.com Region : us-west-2 Image : CC-Ubuntu14.04
desktop-07.hudson.com Region : us-gov-west-1 Image : CC-Ubuntu12.04	lt-47.chen-good.org Region : us-east-1 Image : CC-Ubuntu18.04	email-81.shepherd.info Region : us-east-2 Image : CC-Ubuntu16.04	db-64.johnson.com Region : us-east-1 Image : CC-Ubuntu16.04

Figure 47: VM List

Default view is card layout. You can switch to table layout using action available at the top-right corner. Figure 48 shows available view options for the VM list

UPDATE VM DATA

GRID

CARD VIEW

SORT BY GRID LIST

Figure 48: View options

App will list only first 40 VMs using query

```
{
  query {
    allAzures (first:40) {
      edges {
        node {
          host,
          name,
          region,
          image,
          state,
          isFavorite
        }
      },
      pageInfo {
        endCursor,
        hasNextPage
      }
    }
  }
}
```

On vertical scroll, the list will be updated with query

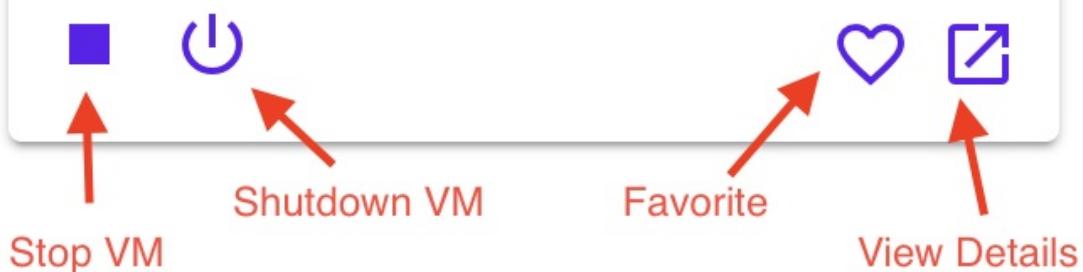
```
{
  query {
    allAzures (first: 40, after: \"YXJyYXljb25uZWN0aW9uOjM5\") {
      edges {
        cursor,
        node {
          host,
          name,
          region,
          publicIps,
          privateIps,
          image,
          state,
          isFavorite
        }
      },
      pageInfo {
        endCursor,
        hasNextPage
      }
    }
  }
}
```

This way the app can load any number of VMs without any performance issue. Each vm has a set of actions assigned. You can start/stop, mark as favorite or view details of VM. Figure 49 explains all the available actions for each VM in the list.

laptop-62.baker.biz

Region : West US

Image : CC-Ubuntu16.04



lt-72.grimes-krueger.com

Region : East US 2

Image : CC-Ubuntu18.04



Figure 49: VM Actions

On click of **start vm** app will execute this mutation query

```
mutation($cardAction:String!,$value:String!,$host:String!,$action:String!) {  
    updateAzure(host:$host, action:$action, actionDetail: $cardAction, value: $value) {  
        azure {  
            state  
        }  
    }  
}
```

```
{  
  "variables": {  
    "cardAction": "start",  
    "action": "state",  
    "host": "desktop-28.woods-porter.com",  
    "value": "false"  
  }  
}
```

On click of ***View Details*** action the app will open pop-up and show VM details. Figure [50](#) shows details view

The screenshot shows a user interface for managing virtual machines across different cloud providers. On the left is a sidebar with navigation links for Cloudmesh, Home, Profile, Clouds (VMS selected), Images, Register, Settings, and Help & feedback. The main area is titled 'VM List' and shows a grid of VM entries. At the top right of the grid is a 'UPDATE VM DATA' button. Below it is a 'SORT BY' dropdown menu with icons for sorting by host and name. The 'AZURE' tab is currently active. A modal window titled 'VM Details' is overlaid on the grid, showing detailed configuration for a specific VM named 'srv-45.young.org'. The modal includes fields for host, name, region, image, publicips, privateips, state, isfavorite, extra, and a large JSON object for hardware and network profiles. At the bottom of the modal are 'CLOSE' and 'UPDATE VM DATA' buttons.

Figure 50: VM Details

User can use **Sort By** dropdown menu available at the top-right corner to sort the VM list by host and name. When you sort a list it will also sort VMs by favorite flag. While implementing the app we found that graphene does not yet support sorting. So we created a query and used **order_by** function provided by mongo.

At any time you can switch cloud provider by just switching the tabs available at top.

Update VM Data button available at top-right corner, is used to load the data from cloudmesh and store it in mongo database. Right now integration with cm4 is not implemented but faker will generate some fake data and store it in mongodb.

6.7 SUMMARY

6.7.1 Sorting support

We observed that graphene-mongo does not have support for sorting. And after exploring a little more, we found that below GraphQL servers do support it

- expressjs-server

- Flask-SQLAlchemy
- Graphene-Django

We were able to find out a workaround but it is not a very convenient way. We had to write to multiple resolver functions for each sort function.

```
def resolve_sortAWSByHost(self, info, **args):
    return AwsModel.objects().order_by("-isFavorite", "host")

def resolve_sortAzureByHost(self, info, **args):
    return AzureModel.objects().order_by("-isFavorite", "host")

def resolve_sortAWSByName(self, info, **args):
    return AwsModel.objects().order_by("-isFavorite", "name")

def resolve_sortAzureByName(self, info, **args):
    return AzureModel.objects().order_by("-isFavorite", "name")
```

6.7.2 Indexing

Once we sorted the data, we observed huge performance degradation in data fetching. Hence it became necessary for us apply correct indexing on mongodb. We applied multiple indexes on the columns which we were using for sorting to see some improvements.

```
collection_vm.create_index([("isFavorite", pymongo.DESCENDING)])
```

6.8 CONCLUSION AND FUTURE WORK

Cloudmesh GraphQL App provides a very nice interface so it can be used by non-technical users as well. There are some performance bottlenecks which we observed for sorting, but it is not actually GraphQL issue because the sorting function was provided by `mongoengine`. We think that after indexing and tweaking some parameters for mongodb it can be improved but we haven't tried it. In future fake data generation can be just replaced with command like `cm4 vm list`. Or even better option would be to integrate cloudmesh inside app so that a standalone app can be distributed to users without need to install cm4 package.

Also from future implementation perspective GraphQL implementations in other languages should be explored. While implementing the app we observed that some of the functions are not implemented by `Graphene`.

6.9 ACKNOWLEDGEMENT

We would like to thank Professor Gregor von Laszewski and TAs for their support and guidance.

6.10 WORKBREAKDOWN

Task	Author
Initial code setup with client server integration	Mihir
Integration of mongoengine	Vineet
Implementation of login page	Mihir
Add routing between pages	Mihir
Use Flask in server code	Vineet
GraphQL query to fetch VMs list	Vineet
Use Faker to generate mock VMs	Mihir
GraphQL mutation to change VM state	Vineet
Add Tabs for various VM types (AWS, Azure, etc.)	Mihir
Add variables for GraphQL APIs	Mihir
Make mutations generic	Vineet
Make custom helper classes for Handlebar templates	Mihir
Lazy loading (infinite scrolling on UI) of VMs	Vineet
Mutation for set/unset Favorite VM	Vineet
Add code to show VM details	Mihir
Add code to mimic DB update from cloudmesh data	Mihir
Add code to show notifications	Mihir
Add helper to prettify JSON	Mihir
Implement sort by dropdown and integrate with API	Mihir
Added table view to the list of VMs	Vineet
Added sorting by IsFavorite and host	Vineet
Implement image page to list image	Mihir

Implement list all image APIs

Mihir

7 OPEN API WITH AWS EMR AND JUPYTER

Ian Sims
isims@iu.edu

Indiana University
hid: fa18-516-22
github: [cloudycode](#)
code: [cloudycode](#)

Keywords: AWS, Open API, EC2, EMR, Jupyter, S3

Learning Objectives

- Use Open API to interact with various AWS products
- Learn to deploy an AWS EMR Cluster
- Interact with Jupyter notebooks stored in S3 buckets

7.1 ABSTRACT

The goal of this project is to create an AWS cloud environment to facilitate analytical work at scale. This includes creating an API to facilitate the creation of an analytical environment as well as APIs to aid a user in determining the types of analytical processes that already exist.

Specifically, this project involves building an API for the creation of an AWS EMR cluster and APIs to give users the ability to determine available analytical datasets on AWS S3. Finally, we will create an API that interacts with the S3 API and gives the ability to parse through Jupyter notebook files to determine which analytical processes utilize a given dataset.

The project architecture can be visualized as follows:

Figure [51](#) shows the proposed architecture for this project

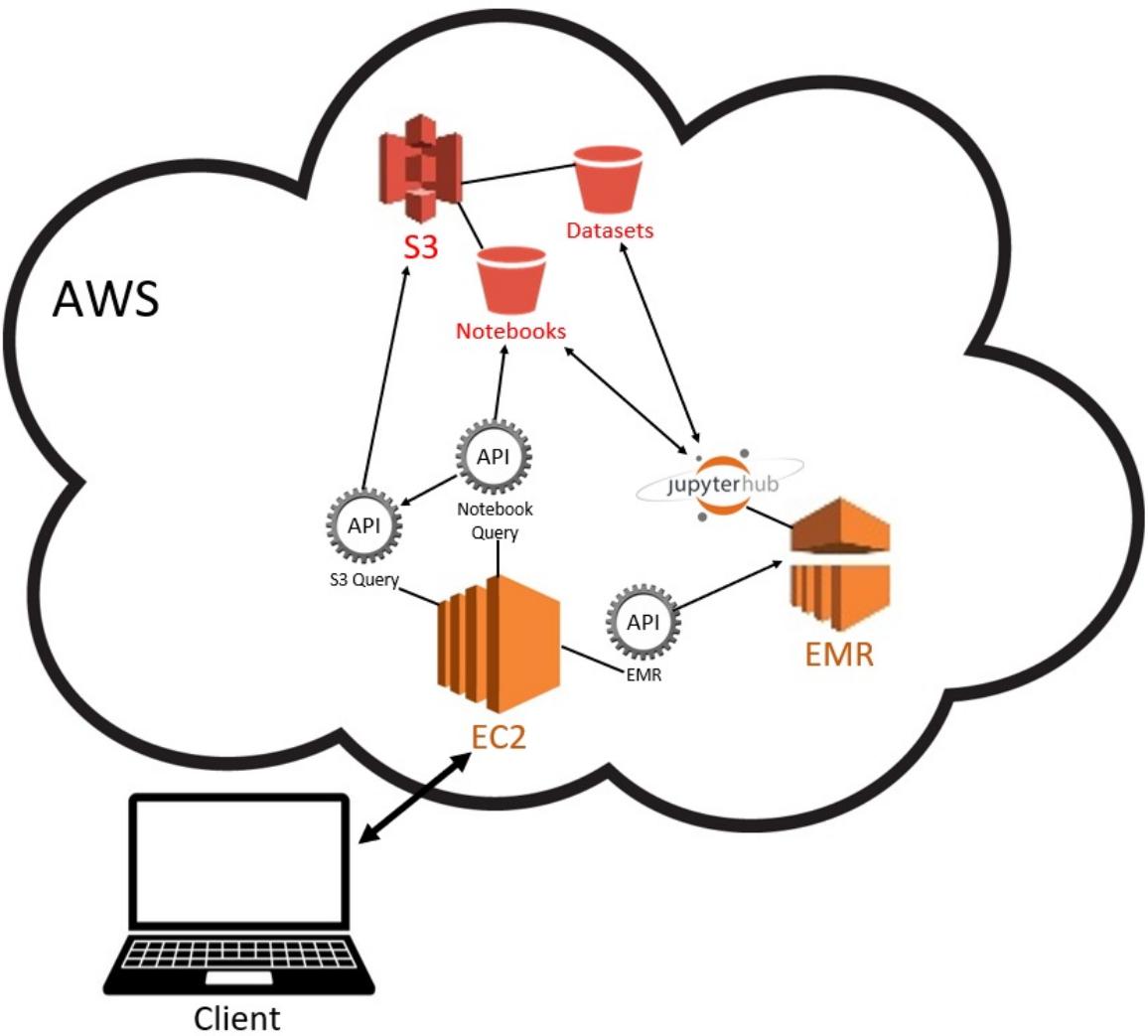


Figure 51: Project Architecture

7.2 INTRODUCTION

The following provides a description of the different technologies and products that were used in the project.

7.2.1 Open API

Rest API's allow for the creation of services that can interact with multiple applications. This project will seek to develop an API for interacting with various AWS products. These products include AWS EMR and S3. In addition, these APIs will be hosted on an AWS EC2 instance.

Open API is an open source project intended to create a consistent format for creating REST services. Open API describes this project as:

“The OpenAPI Initiative (OAI) was created by a consortium of forward-looking industry experts who recognize the immense value of standardizing on how REST APIs are described.” [41]

7.2.2 AWS EMR

EMR is an Amazon product that allows for the creation of clusters of Elastic Compute Cloud (EC2) instances. EMR allows user to take advantage of distributed computing capabilities. As the name suggests this product is designed to allow users to easily scale their cluster to meet their computing needs.

EMR clusters can be created through relatively simple web interfaces or can be created through code using CLI. EMR Clusters can be configured for size and can be provisioned with open-source distributed frameworks such as SPARK and HBase.

For this project we will interact with EMR using an API. This API will allow for the creation and termination of an EMR cluster. It will also allow a user to retrieve the status of an EMR cluster. This EMR cluster will also include the installation of Jupyter Hub to enable the development of notebooks for analytical purposes.

7.2.3 AWS EC2

EC2 is an Amazon product that enables cloud computing. Amazon describes this product as:

“Amazon Elastic Compute Cloud (Amazon EC2) is a web service that provides secure, resizable compute capacity in the cloud. It is designed to make web-scale cloud computing easier for developers.” [42]

For this project APIs will be hosted on an EC2 instance.

7.2.4 AWS S3

S3 is one of Amazon's data storage solutions. For this project we will configure an EMR cluster to read/write to an S3 bucket. This bucket would potentially be used for accessing data for analytical purposes and to store log files associated with the cluster. In addition, the Jupyter Hub instance installed on the EMR cluster will store created notebooks on S3. This will allow for terminating the cluster when it is not in use with the ability to retrieve saved notebooks for future use.

7.2.5 JupyterHub

JupyterHub is an open-source project intended to allow a wide range of users to interact with and organize notebooks for analysis. The open-source project describes JupyterHub as follows:

“JupyterHub brings the power of notebooks to groups of users. It gives users access to computational environments and resources without burdening the users with installation and maintenance tasks. Users - including students, researchers, and data scientists - can get their work done in their own workspaces on shared resources which can be managed efficiently by system administrators.” [43]

For this project we will build an API that creates an Amazon EMR cluster that includes an installation of JupyterHub.

7.3 IMPLEMENTATION

7.3.1 Setting up AWS CLI

After setting up an AWS account account: [AWS Account](#) and an [AWS Key Pair](#), we needed to be able to work with AWS products from the command line. To do this we utilized the AWS Command-Line Interface (CLI) from a Linux environment.

First we set up a [Linux](#) environment using VirtualBox. We then installed [Python](#)

and [PIP](#) on that environment. Finally we installed CLI using the following Bash command:

```
$ pip install awscli
```

The following item had to be configured for CLI:

- AWS Access Key ID
- AWS Secret Access Key
- Default region name (this is the default region that will be used when you create EC2 instances)
- Default output format (the default format is json)

7.3.2 Setting up AWS Admin Access

In order to work from the command line with various AWS products we had to set up admin access. Using CLI we ran the following commands:

```
$ aws iam create-group --group-name Admins
```

```
$ aws iam attach-group-policy --group-name Admins --policy-arn arn:aws:iam::aws:policy/AdministratorAccess
```

Then through the [AWS Console](#) we assigned users to the admin group. Under ‘Group Actions’, we selected ‘Add Users to Group’.

Figure [52](#) shows the AWS Console screen for adding users to a admin security group

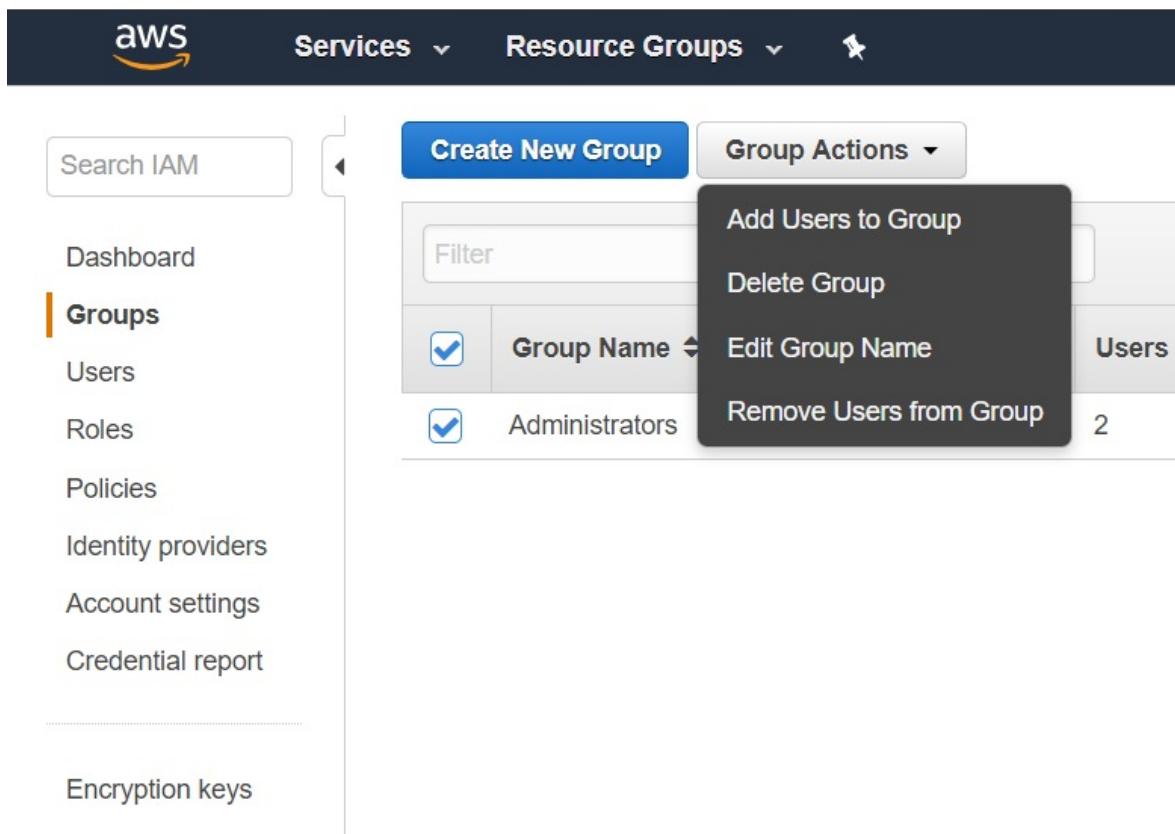


Figure 52: AWS Security [44]

7.3.3 Creating and Configuring EC2 Instance to Host API

7.3.3.1 EC2 Security Group

To set up the EC2 instance for hosting our APIs we first used the Amazon Console to set up a security group.

Navigating to: [EC2 Security Group](#) we selected ‘Create Security Group’

Figure 53 shows the screen to create an AWS security group

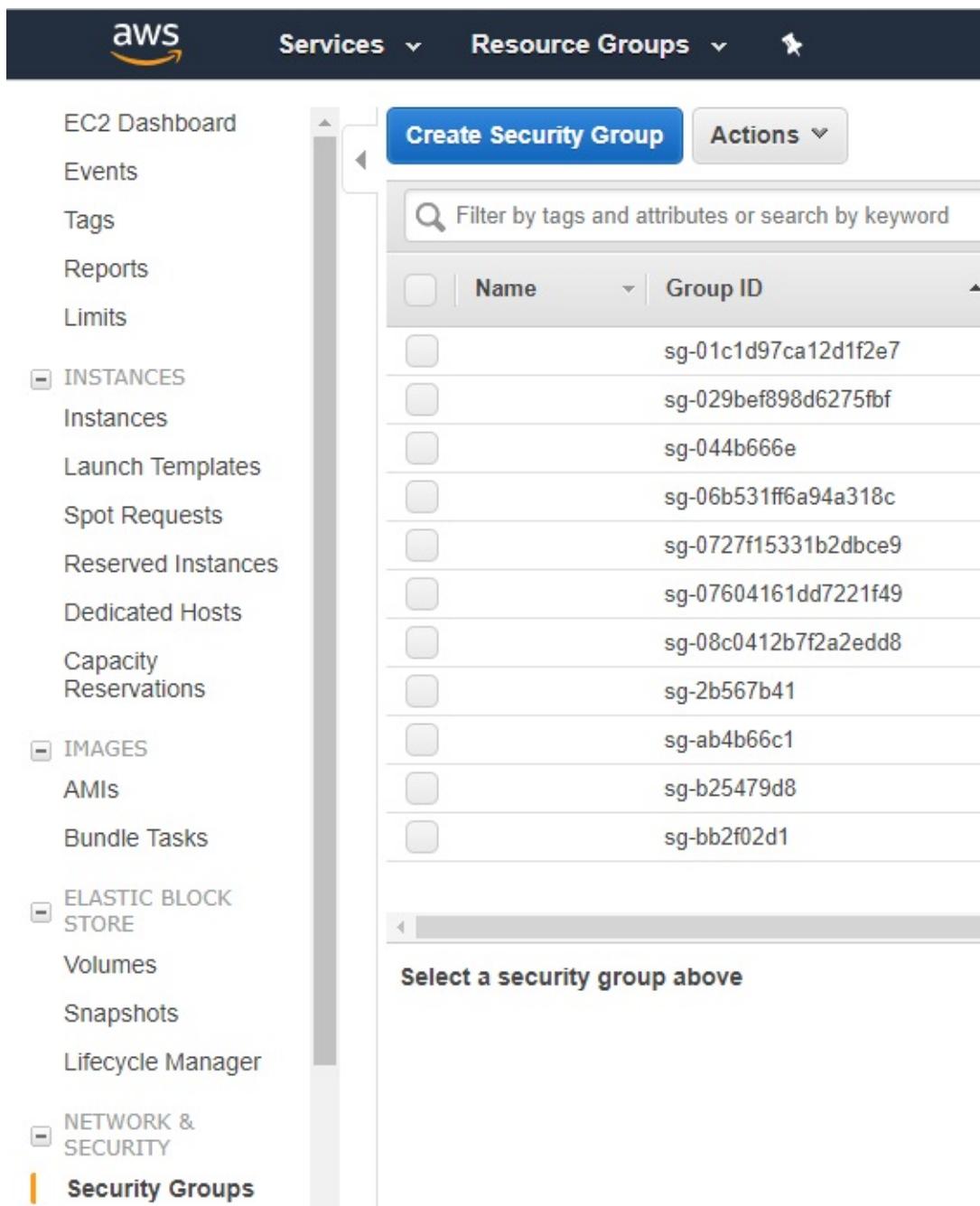


Figure 53: AWS Security [45]

We then gave the security group a name, selected the default VPC and added two rules. One that opens ports 8080, 8081, and 8082 for http traffic and one to allow ssh access from a single ip. Ports 8080, 8081, and 8082 will be used for accessing the APIs.

Figure 54 shows the AWS screen for defining a security group

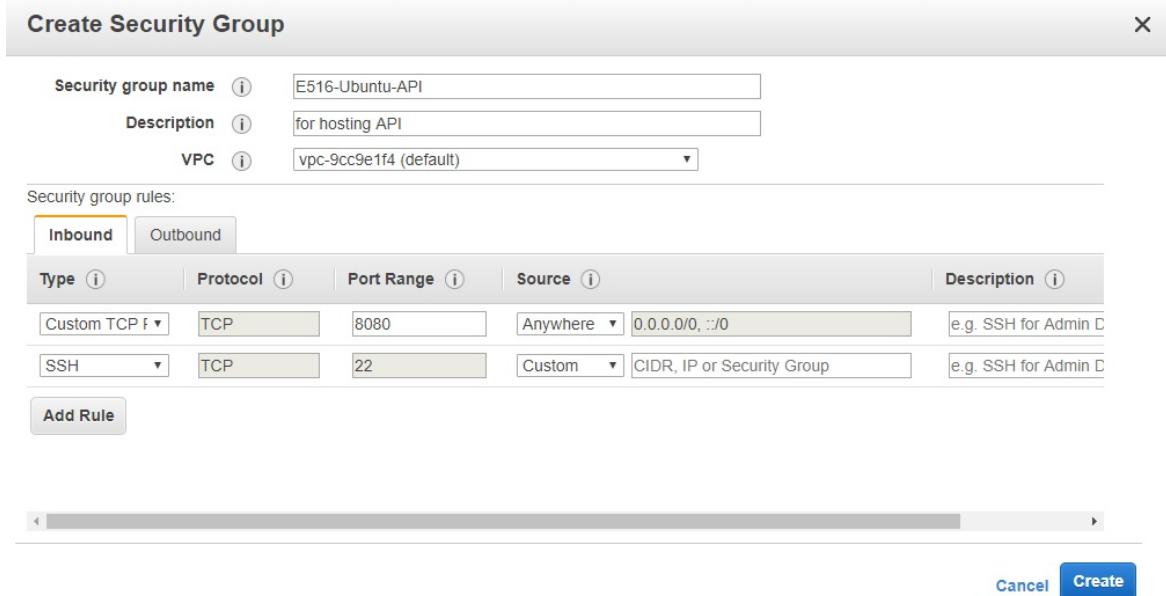


Figure 54: AWS Security [45]

7.3.3.2 EC2 Create Instance

Now it was time to create an EC2 instance using the AWS Console: [Launch EC2](#). We clicked the ‘Launch Instance’ button.

Figure 55 shows the AWS Console screen for launching an EC2 instance

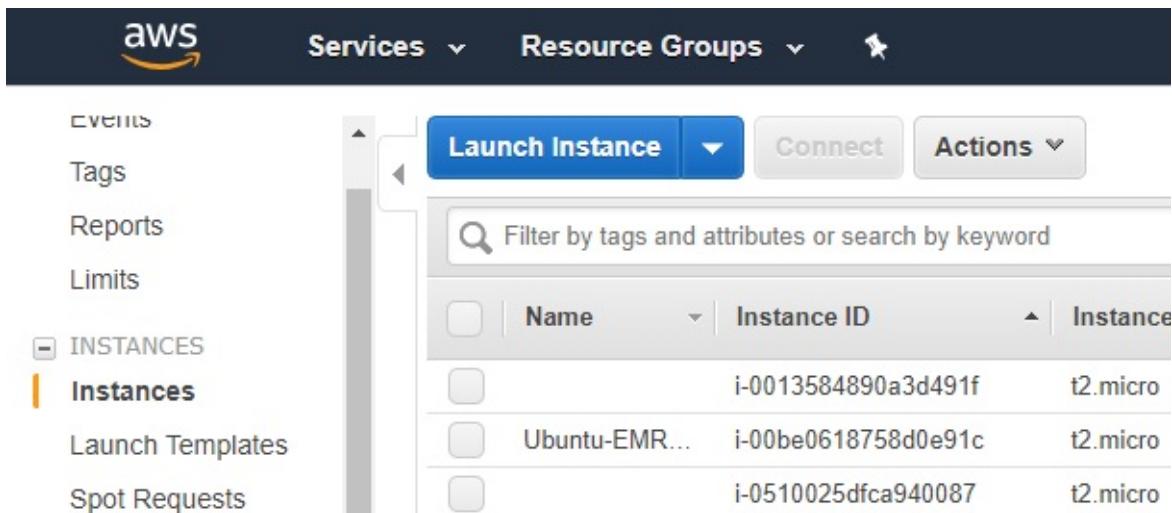


Figure 55: AWS EC2 [46]

We selected the Ubuntu version, using version 18.04:

Figure 56 shows the AWS screen used for selecting an EC2 operating system



Figure 56: AWS EC2 OS [46]

We selected a small instance type and went to "Next: Configure Instance Details".

Figure 57 shows the AWS Console screen for selecting the type of instance

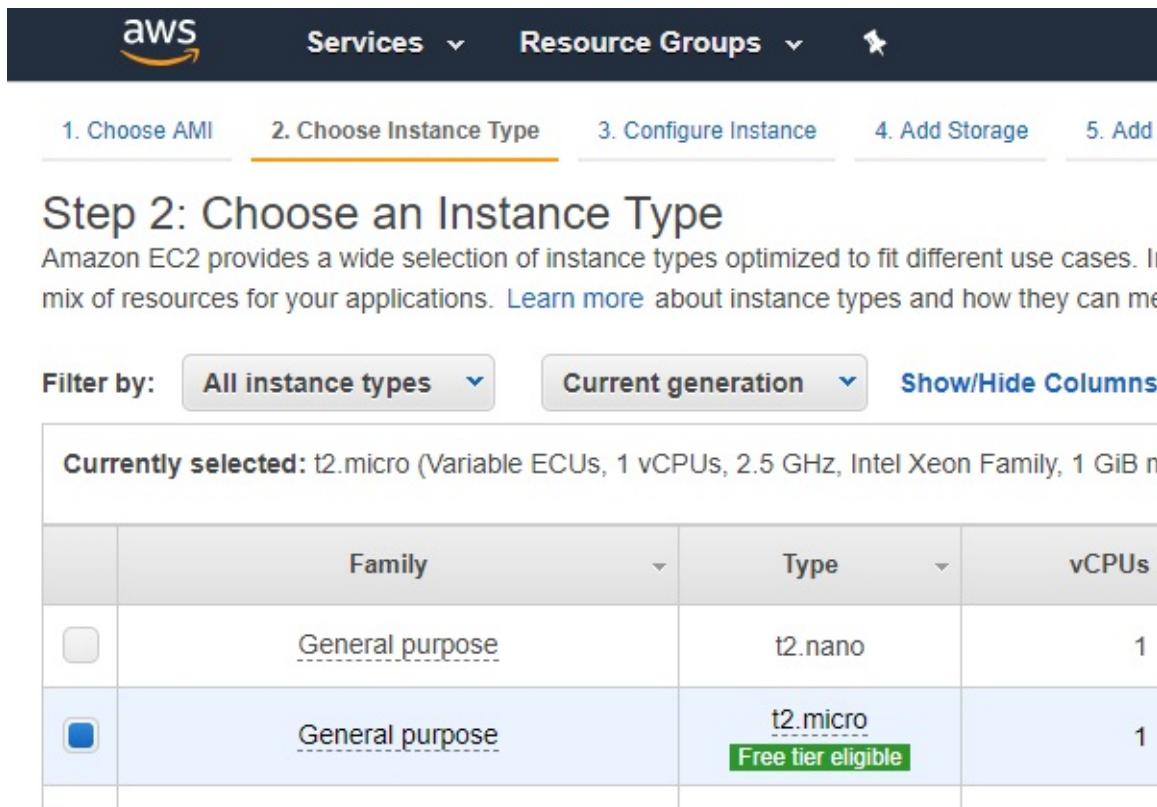


Figure 57: AWS EC2 Type [46]

We made sure the default VPC is selected and then went to 'Configure Security Group'.

Figure 58 shows the AWS Console screen for configuring security on an EC2

instance

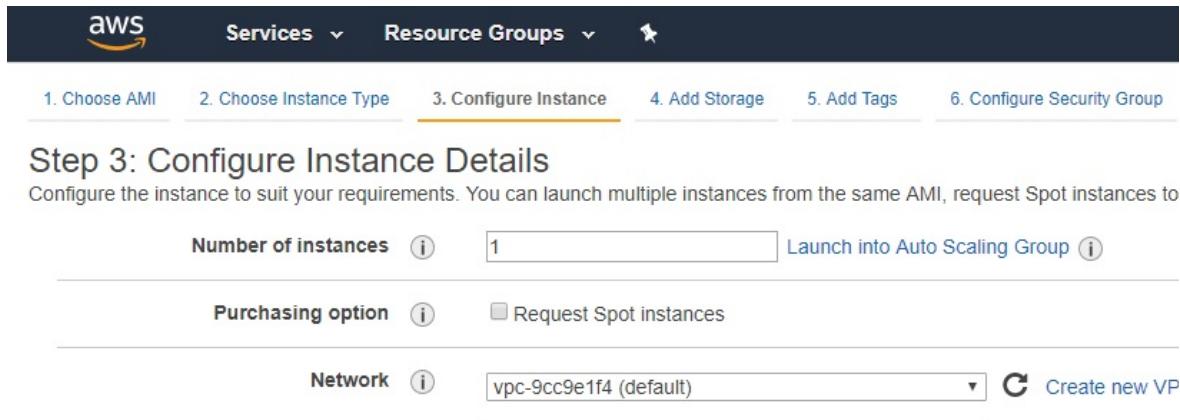


Figure 58: AWS EC2 Security Config [46]

We clicked ‘Select and existing security group’ and selected the group created earlier.

Figure 59 shows the AWS Console screen for selecting a security group for EC2

The screenshot shows the AWS EC2 instance creation wizard at Step 6: Configure Instance. The top navigation bar includes 'Services' and 'Resource Groups'. Below the steps, the title 'Step 6: Configure Security Group' is displayed. A descriptive text explains that a security group is a set of firewall rules that control traffic for your instance. It allows access to HTTP and HTTPS ports and can be created or selected from an existing group. Two radio buttons are shown for 'Assign a security group': one for 'Create a new security group' (unchecked) and one for 'Select an existing security group' (checked). A table lists existing security groups with columns for 'Security Group ID', 'Name', and a small icon. One group, 'sg-0727f15331b2dbce9Ubuntu-EC2', has a blue checked checkbox next to it, indicating it is selected.

	Security Group ID	Name	
<input type="checkbox"/>	sg-ab4b66c1	default	d
<input type="checkbox"/>	sg-07604161dd7221f49dlec2-sg		S
<input type="checkbox"/>	sg-01c1d97ca12d1f2e7E516-EMR-Master		F
<input type="checkbox"/>	sg-08c0412b7f2a2edd8ElasticMapReduce-master		M
<input type="checkbox"/>	sg-06b531ff6a94a318c ElasticMapReduce-slave		S
<input type="checkbox"/>	sg-b25479d8	launch-wizard-1	la
<input type="checkbox"/>	sg-2b567b41	launch-wizard-2	la
<input type="checkbox"/>	sg-044b666e	Neo4j Graph Database - Community Edition-3-4-1-AutogenByAWSMP-T	
<input type="checkbox"/>	sg-bb2f02d1	ssh_http	S
<input checked="" type="checkbox"/>	sg-0727f15331b2dbce9Ubuntu-EC2		U

Figure 59: AWS EC2 Security Select [46]

The EC2 instance could then be launched.

7.3.3.3 EC2 SSH

We then went into to a local Linux environment and set up a key pair to enable ssh to our EC2 instance. We did this using CLI and the following commands:

```
$ aws ec2 create-key-pair --key-name dlec2-key --query 'KeyMaterial' --output text > dlec2-key.pem
```

Allowed access to the key:

```
$ chmod 400 dlec2-key.pem
```

Then locating the ‘Public DNS’ at: [AWS EC2](#), we connected to the EC2 instance with the following command:

```
$ ssh -i "dlec2-key.pem" ubuntu@ec2-18-191-50-79.us-east-2.compute.amazonaws.com
```

7.3.3.4 EC2 Set Up

We then set up a Python virtual environment for our rest services:

```
$ pyenv install -l
$ pyenv install 3.6.6
$ pyenv virtualenv 3.6.6 RestService
$ pyenv activate RestService
```

Installed AWS CLI

```
$ pip install awscli
```

The following items had to be configured for CLI:

- AWS Access Key ID
- AWS Secret Access Key
- Default region name (this is the default region that will be used when you create EC2 instances)
- Default output format (the default format is json)

7.3.4 Create S3 Storage

In order to store analytical data and to backup our Jupyter notebooks We created two S3 buckets using the following commands:

```
$ aws s3 mb s3://e516-analytical-datasets --region us-east-2
$ aws s3 mb s3://e516-jupyter-backup --region us-east-2
```

7.3.5 Codegen Set Up

7.3.5.1 Install Java

We used Codegen to create our rest services and Java is a requirement. We installed Java using the following commands:

```
$ sudo apt update
$ sudo apt install default-jre
$ sudo apt install default-jdk
```

7.3.5.2 Install Codegen

We ran the following commands for installation:

```
$ mkdir ~/e516/swagger
$ cd ~/e516/swagger
$ wget https://oss.sonatype.org/content/repositories/releases/io/swagger/swagger-codegen-cli-2.3.1.jar
```

We then opened the .bashrc file and added an alias for codegen:

```
alias swagger-codegen="java -jar ~/e516/swagger/swagger-codegen-cli-2.3.1.jar"
```

7.3.6 Building the EMR Rest Service

7.3.6.1 Swagger YAML Specs

Using Swagger we built the API specs. This API has POST, DELETE, and GET methods. The POST method will create an AWS EMR cluster and install JupyterHub. The DELETE method allows for the termination of the cluster. The GET method retrieves information about the cluster including the status and a link to the Jupyter Hub web ui.

```
swagger: "2.0"
info:
  version: "0.0.1"
  title: "emrinfo"
  description: "API to spin up an AWS EMR cluster, to check status, and to terminate."
  termsOfService: "http://swagger.io/terms/"
  contact:
    name: "EMR Rest Service"
  license:
    name: "Apache"
host: 18.191.50.79:8080
basePath: /api
schemes:
  - http
consumes:
  - "application/json"
produces:
  - "application/json"
paths:
  /emr/create/{num_of_nodes}:
    post:
```

```

summary: Create EMR cluster.
parameters:
  - in: path
    name: num_of_nodes
    required: true
    type: integer
    minimum: 1
    description: The number of nodes in the cluster
responses:
  200:
    description: OK
/emr/info/{cluster_id}:
get:
  summary: Returns EMR cluster Info.
  parameters:
    - in: path
      name: cluster_id
      required: true
      type: string
      minimum: 1
      description: The cluster id for EMR
  responses:
    200:
      description: OK
/emr/terminate/{cluster_id}:
delete:
  summary: Deletes EMR cluster.
  parameters:
    - in: path
      name: cluster_id
      required: true
      type: string
      minimum: 1
      description: The cluster id for EMR
  responses:
    200:
      description: OK
definitions:
  EMR:
    type: "object"
    required:
      - "model"
    properties:
      model:
        type: "string"

```

7.3.6.2 Deploy EMR Rest Service

As mentioned, we used Swagger Codegen for the creation of our APIs. Once our swagger yaml files was cloned to our EC2 instance we ran the following code to create the needed files:

```
$ swagger-codegen generate \
-i ~/fa18-516-22/project-code/emr-api.yaml \
-l python-flask \
```

```
-o ~/emr-api/server/emr/flaskConnexion \
-D supportPython3=true
```

We then navigated to our controller file and edited it using nano:

```
$ cd ~/emr-api/server/emr/flaskConnexion/swagger_server/controllers
$ nano default_controller.py
```

We then updated our POST, DELETE, and GET methods with the Python functions we created. The POST method accepts a parameter indicating the number of instances desired in a cluster and will create an AWS EMR cluster, configure it and install JupyterHub. This includes setting up the security for the EMR cluster and specifying S3 buckets for it to interact with. It returns a dictionary that includes the created cluster's ID, a link to the GET method for checking the status of the cluster, and a curl command for executing the DELETE method for terminating the cluster.

```
import subprocess

def emr_post(num_nodes):

    aws_cmd = "aws emr create-cluster --name='E516-JupyterHub-Cluster'"
    aws_cmd += " --release-label emr-5.19.0"
    aws_cmd += " --applications Name=JupyterHub"
    aws_cmd += " --log-uri s3://e516-jupyterhub-backup/JupyterClusterLogs"
    aws_cmd += " --use-default-roles"
    aws_cmd += " --ec2-attributes SubnetIds=subnet-d0169eaa,KeyName=dlec2-key,AdditionalInstances=ON_DEMAND,InstanceType=m4.xlarge"
    aws_cmd += " --instance-count " + str(num_nodes)
    aws_cmd += " --instance-type m4.large"
    aws_cmd += " --configurations '[{\"Classification\":\"jupyter-s3-conf\", \"Properties\": {}}]"
    aws_cmd += " --output text"

    c_id = subprocess.run(aws_cmd, shell=True, stdout=subprocess.PIPE)

    cid = c_id.stdout.decode('utf-8')
    cid = cid.rstrip()

    c_status = "http://ec2-18-191-50-79.us-east-2.compute.amazonaws.com:8080/api/emr/info/"
    t_clstr = 'curl -X "DELETE" http://ec2-18-191-50-79.us-east-2.compute.amazonaws.com:8080'

    rtn_dict = {
        "ClusterId": cid,
        "CheckClusterStatus": c_status,
        "TerminateCluster": t_clstr
    }

    return rtn_dict
```

The DELETE method accepts an EMR cluster ID as an input and then

terminates the specified EMR cluster. It returns a dictionary containing the cluster ID, the current status of the cluster, and the url for using the GET method to check the status of the cluster.

```
import subprocess

def emr_delete(cid):

    subprocess.run("aws emr terminate-clusters --cluster-ids " + cid, shell=True)

    c_status = ("http://ec2-18-191-50-79.us-east-2.compute.amazonaws.com:8080/api/emr/info/" + cid).replace(" ", "%20")

    rtn_dict =  {
        "ClusterId": cid,
        "Status": "TERMINATING",
        "CheckClusterStatus": c_status
    }

    return rtn_dict
```

The GET method allows a use to query information about the EMR cluster. The method accepts a cluster ID as an input and returns a dictionary containing the cluster ID, the cluster status, a link to access the JupyterHub UI, as well as the default username and password for JupyterHub.

This function utilizes the Python library ‘boto3’. This is a library created by Amazon to interact with AWS products with Python directly.

```
import boto3

def emr_get(cid):

    client = boto3.client('emr')
    c_info = client.describe_cluster(ClusterId=cid)

    c_status = c_info["Cluster"]["Status"]["State"]
    if "MasterPublicDnsName" in c_info["Cluster"]:
        j_hub = ("https://" + c_info["Cluster"]["MasterPublicDnsName"] + ":9443")
        j_un = "jovyan"
        j_pw = "jupyter"
    else:
        j_hub = ""
        j_un = ""
        j_pw = ""

    rtn_dict = {
        "ClusterId": cid,
        "ClusterStatus": c_status,
        "JupyterHub": j_hub,
        "JupyterUN": j_un,
```

```

        "JupyterPW": j_pw
    }

    return rtn_dict

```

Once the default_controller.py file was updated, we activated this rest service using the following commands.

```
$ cd ~/emr-api/server/emr/flaskConnexion
$ pip install -r requirements.txt
$ python setup.py install
$ python -m swagger_server
```

The POST method could then be accessed using a curl command. In this case we will only specify two instances to be included in the cluster (the example output is included)

```
$ curl -X POST http://ec2-18-191-50-79.us-east-2.compute.amazonaws.com:8080/api/emr/create,
{
  "CheckClusterStatus": "http://ec2-18-191-50-79.us-east-2.compute.amazonaws.com:8080/api/e
  "ClusterId": "j-3A70IXQPD60HK",
  "TerminateCluster": "curl -X DELETE http://ec2-18-191-50-79.us-east-2.compute.amazonaws.co
}
```

The CheckClusterStatus url can then be used in a web browser to check the cluster's status.

Figure 60 shows the results of executing the GET method for the EMR API

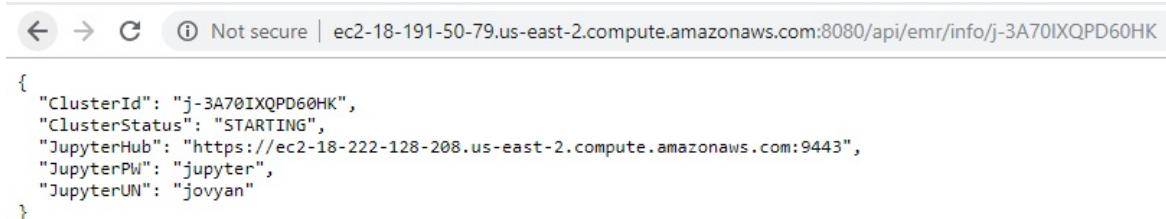


Figure 60: EMR API GET

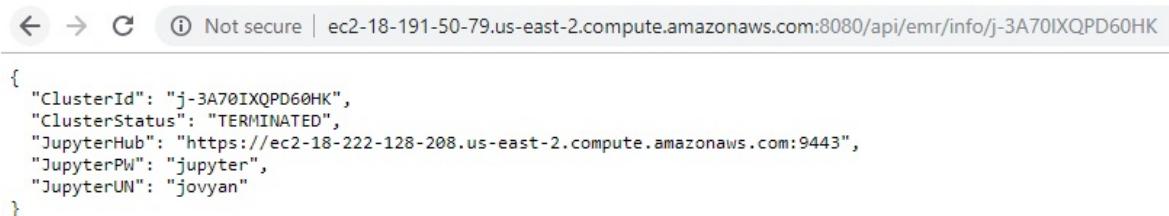
Once the cluster is started, the JupyterHub web interface can be accessed through the given url. The EMR API configured the cluster to store created notebook to S3, so when the cluster is terminated the notebooks will still be persisted. In addition, each time the POST method is used to create an EMR cluster any persisted notebooks will be available to the user in JupyterHub.

The DELETE method can be used in a curl command to terminate the cluster.

```
$ curl -X DELETE http://ec2-18-191-50-79.us-east-2.compute.amazonaws.com:8080/api/emr/terms/{ClusterId}
{
  "CheckClusterStatus": "http://ec2-18-191-50-79.us-east-2.compute.amazonaws.com:8080/api/emr/info/j-3A70IXQPD60HK",
  "ClusterId": "j-3A70IXQPD60HK",
  "Status": "TERMINATING"
}
```

Using the GET method shows that the cluster has been terminated

Figure 61 shows the results of executing the GET method for the EMR API



```
{
  "ClusterId": "j-3A70IXQPD60HK",
  "ClusterStatus": "TERMINATED",
  "JupyterHub": "https://ec2-18-222-128-208.us-east-2.compute.amazonaws.com:9443",
  "JupyterPW": "jupyter",
  "JupyterUN": "jovyan"
}
```

Figure 61: EMR API GET Results

7.3.7 Deploy S3 Rest Service

This rest service was meant to provide an abstraction layer that can be used by another rest service. This API provides a simple function to query the contents of an AWS S3 bucket and return the keys to any files that meet a specified file extension type. The Swagger specifications for this API are included below.

```
swagger: "2.0"
info:
  version: "0.0.1"
  title: "s3info"
  description: "API to query S3 bucket by file extension"
  termsOfService: "http://swagger.io/terms/"
  contact:
    name: "S3 Rest Service"
  license:
    name: "Apache"
host: 18.191.50.79:8081
basePath: /api
schemes:
  - http
consumes:
  - "application/json"
produces:
  - "application/json"
paths:
  /s3:
    get:
      summary: Get a list of files and paths with the given file extension
      parameters:
```

```

    - in: query
      name: bucket
      required: true
      type: string
      minimum: 1
      description: The S3 bucket name
    - in: query
      name: path
      required: true
      type: string
      minimum: 1
      description: The folder path to query
    - in: query
      name: extension
      required: true
      type: string
      minimum: 1
      description: The file extension to search for
  responses:
    200:
      description: OK
definitions:
  S3:
    type: "object"
    required:
      - "model"
    properties:
      model:
        type: "string"

```

Similar to the EMR-API we built this API using Swagger Codegen. The same steps were followed with one exception. The difference was the need to specify a different port to use for the API. There seems to be a bug in Codegen where the specified port in the yaml file is not used. Codegen defaults to always using port 8080. In order to change the port to allow for multiple APIs to run on the same EC2 instance, we ran the following code to replace the port number with our desired port (8081). This was done after Codegen was run the needed files were created.

```
$ cd ~/s3-api/
$ find . -type f -exec sed -i 's/8080/8081/g' {} +
```

The default_controller.py file was then edited with the following Python code. This function used the Amazon ‘boto3’ package to search a specified S3 bucket and path for a given file extension. It then returns a list of all files that meet the specifications.

```
import boto3

def search_s3_by_ext(bucket, path, extension):
```

```

client = boto3.client('s3')
obj = client.list_objects_v2(Bucket=bucket, StartAfter=path )

rtn_list = []
for object in obj['Contents']:
    if object['Key'][-(len(extension)):] == extension:
        rtn_list.append(object['Key'])

return(rtn_list)

```

The API could then be deployed using the following.

```

$ cd ~/s3-api/server/s3/flaskConnexion
$ pip install -r requirements.txt
$ python setup.py install
$ python -m swagger_server

```

The API can be used to query a specified S3 bucket. In the example below we query the S3 bucket ‘e561-jupyter-backup’ and specify a path for a particular user ‘jupyter/jovyan’. We also specify to search for files with an extension of ‘.ipynb’ (notebooks). The service returns a list of files (including the path) that meet the specifications.

Figure [62](#) shows the results of executing the GET method for the S3 API

```

[ "jupyter/jovyan/iris-Copy1.ipynb",
  "jupyter/jovyan/search-notebook-contents.ipynb",
  "jupyter/jovyan/search-s3-contents.ipynb",
  "jupyter/jovyan/setup.ipynb",
  "jupyter/jovyan/test-datasets/create-test-data.ipynb",
  "jupyter/jovyan/test-datasets/iris.ipynb",
  "jupyter/jovyan/test-datasets/nile.ipynb",
  "jupyter/jovyan/test-datasets/titanic.ipynb"
]

```

Figure 62: S3 API GET

7.3.8 Deploy Notebook Rest Service

Our final API is a service that allows for the searching of Jupyter notebook content. This API also connects to the S3 API as an abstraction layer.

Jupyter notebooks allow for custom tagging on individual ‘cells’. These tags are saved with the notebook. An example of a ‘tag’ applied to an individual cell is given below. In this case the tag is called ‘parameters’.

Figure [63](#) shows an example of a Jupyter notebook tag

```
In [15]: ► parameters x
mybucket = 'e516-jupyter-backup'
path = 'jupyter/'
file_extension = '.ipynb'
```

Figure 63: Notebook Tag

Once again we used Swagger for the API specifications. This service has a GET method that accepts parameters for an S3 bucket, a path, and a text field to search for in the notebooks. The Swagger specifications are provided below.

```
swagger: "2.0"
info:
  version: "0.0.1"
  title: "notebookinfo"
  description: "API to query jupyter notebook parameters"
  termsOfService: "http://swagger.io/terms/"
  contact:
    name: "Notebook Rest Service"
  license:
    name: "Apache"
host: 18.191.50.79:8082
basePath: /api
schemes:
  - http
consumes:
  - "application/json"
produces:
  - "application/json"
paths:
  /notebook:
    get:
      summary: Get a list of files and paths with the given file extension
      parameters:
        - in: query
          name: bucket
          required: true
          type: string
          minimum: 1
          description: The S3 bucket name
        - in: query
          name: path
          required: true
          type: string
          minimum: 1
          description: The folder path to query notebooks
        - in: query
          name: search_on
          required: true
          type: string
          minimum: 1
          description: The parameter text to look for
      responses:
        200:
```

```

        description: OK
definitions:
  NOTEBOOK:
    type: "object"
    required:
      - "model"
    properties:
      model:
        type: "string"

```

Codegen was used to generate the needed files and the default_controller.py file was edited to include the following Python code. This function uses the Python library ‘requests’ to call our S3 API. The S3 API provides a list of .ipynb files in the specified bucket and path. We then loop through that list and search each cell to see if there is a ‘parameter’ tag. If there is a ‘parameter’ tag the specified search value is looked for in the cell’s source code. The function returns a list of all the unique notebooks that contain the specified text in any ‘parameter’ cell.

```

import boto3
import json
import requests

def search_nb_param_inpt_data(bucket, path, search_on):

    #call s3 API to get list of files including paths
    url = 'http://ec2-18-191-50-79.us-east-2.compute.amazonaws.com:8081/api/s3'
    payload = {'bucket': bucket, 'path': path, 'extension': '.ipynb'}
    r = requests.get(url, params=payload)

    s3 = boto3.resource('s3')

    #loop through all .ipynb under path
    nb_has_val = False
    nbs_found = []
    for file_path in r.json():

        content_object = s3.Object(bucket, file_path)
        file_content = content_object.get()['Body'].read().decode('utf-8')
        json_content = json.loads(file_content)

        #loop through each cell of nb
        for cell in json_content['cells']:
            if 'source' in cell:
                if 'metadata' in cell:
                    if 'tags' in cell['metadata']:
                        if 'parameters' in cell['metadata']['tags']:
                            if search_on in cell['source']:
                                nbs_found.append('s3://' + bucket + '/' + file_path)

    rtn_nb_full_path_list = list(set(nbs_found))

    return(rtn_nb_full_path_list)

```

Again we needed to specify a different port (8082) with the following code.

```
$ cd ~/s3-api/  
$ find . -type f -exec sed -i 's/8080/8082/g' {} +
```

The API could then be deployed using the following.

```
$ cd ~/notebook-api/server/notebook/flaskConnexion  
$ pip install -r requirements.txt  
$ python setup.py install  
$ python -m swagger_server
```

The API can be used to query the contents of Jupyter notebooks. In the example below we query the S3 bucket ‘e516-jupyter-backup’ and specify a path of ‘jupyter’. We also specify to search the parameter cells for ‘test1/iris.csv’. This text relates to a dataset that is stored on S3. In this way a user could query a set of analytical notebooks to see who is performing analysis using a particular dataset. The service returns a list of files (including the path) that meet the specifications.

Figure [64](#) shows the results of executing the GET method for the Notebook API



```
[  
  "s3://e516-jupyter-backup/jupyter/jovyan/iris-Copy1.ipynb",  
  "s3://e516-jupyter-backup/jupyter/jovyan/test-datasets/iris.ipynb"  
]
```

Figure 64: Notebook API GET

7.3.9 Running all APIs

In order to have all three of these Rest services running on the same EC2 instance. The following commands were run. The ‘&’ returns to the prompt after each service is deployed.

```
$ cd ~/emr-api/server/emr/flaskConnexion  
$ python -m swagger_server &  
$ cd ~/s3-api/server/s3/flaskConnexion  
$ python -m swagger_server &  
$ cd ~/notebook-api/server/notebook/flaskConnexion  
$ python -m swagger_server &
```

The services can be terminated with the following command.

```
$ pkill -f swagger
```

7.4 CONCLUSION

This project results in a usable Jupyter notebook environment that is scalable over time. The intial APIs lay a foundation for a robust notebook environment in which a user can create analytical processes and search for existing ones.

8 MORPHOLOGICAL IMAGE-BASED PROFILING OF SKIN LESIONS FOR SCIENTIFIC COMMUNITY

Anna Heine

avheine@iu.edu

Indiana University

hid: fa18-523-52

github: [cloud](#)

code: [cloud](#)

Keywords: fa18-532-52, lesion, medical

8.1 ABSTRACT

One major area that is being utilized highly today within big data platforms is that of medical image collection. Medical image datasets are often used as training tools as a source of comparison when analyzing diagnostic images from current or past cases of diseases. The issue in this field is that current datasets lack the diversity and number of samples that is necessary to make sufficient predictions based on analysis. With valuable datasets, medical advances can be made to deliver more personalized medicine, to create predictive diagnostic models, research treatment methods, improve the overall value of healthcare, and much more. The main goal of medical big data analysis is to find associations and correlations within complex data. The HAM10000 dataset is a significant resource for analysis because it includes over 10,000 dermatoscopic images that have been carefully stained and optimized to display skin-lesion biopsies. The grafts entered into this dataset were verified and therefore proven worthy based on expert consensus, a follow-up, and in-vivo microscopy. Each individual donation is tracked by its patient identification number, image identification number, donor age, donor sex, localization on the body, and a final inclusion reasoning.

8.2 INTRODUCTION

The big data revolution has changed multiple industries around the world, one of which largely includes the field of medicine. The role of big data in the medical sector is high-grade as it aims to be predictive in order to diagnose patients and even discover new treatment methods. This means that the data obtained and used in these models must be wide enough to include a variety of patients and diseases. Disease is a major unknown for many reasons in medicine. Big data has recently been shown to be beneficial in disease management as it provides aggregate information around multiple aspects of a disease. For example, some datasets include functionality and characteristics of DNA and RNA, proteins, cell types, tissues, organs, and more [47]. The ability for these models to evolve and grow are what will advance the predictive analysis so desired in the field today. The sources of big data are varied. Some of which include administrative claim records, health records, the internet, medical imaging, and clinical trials [48].

Despite the vast number of sources, medical big datasets can be quite different than other types of big data. The main difference is located in the form of accessing the data. Because of privacy laws and ethical stances, some medical data is hard to come by. The fact that data has been breached in other industries such as in business and consumerism models, most people are hesitant to include large amounts of people's private health records in one location. That being said, medical datasets have to be managed so that the records being added follow a specific, structured protocol. These datasets are also costly, as they may need to be examined and measured multiple times by trained personnel to ensure correctness. The nature of the data is therefore, dangerously susceptible to human error, as most of it usually contains information we are not very sure about to begin with. Although data scientists have found methods and reasonings that allow for the production of medical datasets, they are often unreliable because the amount of data contained is just too small to make any valuable inferences or claims.

Though there are many challenges medical big data will need to dissolve in order to gain popularity and trust, the field is already proven to hold valuable insights. For example, the field will expand the use of big data into other industries by providing a basis for accumulating data of various sources and materials. Medical big data can provide answers to uncertainties even when lacking substantial evidence. The influx of continued large medical datasets is

contributing to the advance of trusted future predictive healthcare learning models. The overall goal is not to automate the position of a trained physician, but to make the diagnostic process much easier and more efficient as well as aid in several areas of medical research.

The realm of skin cancer research is more or less crying out for big data analysis. Melanoma, specifically, affects around 73,000 new people each year which will result in about 9,000 deaths [49]. There is no general biomarker for melanoma, which causes for imprecise diagnostic margins. With this prevalence of disease in society, the amount diagnoses per melanoma skin lesion sometimes reaches up to 36 because of false-negative uncertainties [49]. However, with big datasets and the use of computer algorithms, there has been a significant increase in diagnostic accuracy -less than 5% error rates.

8.3 REQUIREMENTS

This project requires Python 2.7 or greater to integrate the Pandas and NumPy packages. It requires at least 420 KB of storage to hold the HAM10000 dataset. The project also requires the Anaconda platform to access Jupyter Notebook with Numpy and Pandas packages installed. The cloud service analytics were performed on KNIME's Cloud Analytics Platform.

8.4 DESIGN

The design of the project was to obtain the dataset and test it on a web services. First, we tested the dataset on Jupyter Notebook to get a baseline of the components. After creating some visualizations, we then incorporated the data into KNIME's Analytics Cloud Platform.

8.5 ARCHITECTURE

Before we made any implementations on this project, we realized that we would be using multiple software packages. Therefore, we wanted to ensure that they were all in the same place. To do this, we installed Anaconda. Anaconda is an open-source, free distribution software with Python and R available packages already installed. Its main purpose is to provide simple tracking of these

packages with an easy, user-friendly dashboard design[50]. Each version of its packages are managed by its package management system, Conda. In Anaconda, we used Jupyter and Jupyter Notebook to perform most of my data analysis. Jupyter allows for data manipulation across many programming languages. Jupyter is also an open-source web application that allows for distribution of documents, code, and other projects for collaboration. It has many uses: data cleansing, statistical data modeling, visualization, machine and deep learning, data transformation, and more. Once a project has been created, your work can be output as HTML, images, video, or LaTeX[51]. The Notebook specifically can contain both code and text. These formats have the ability to create descriptions and visible output for graphs of many types. Jupyter runs via client-server, which means it does not need Internet access to be run. Each notebook contains a kernel which controls the execution of the code inside it. For example, if you wanted to execute Python code, the notebook would execute via the ipython kernel. To manually execute a file, the user must either choose to run each cell one at a time by clicking Run on the left of each line or by clicking on Run All in the Cell menu[52].

NumPy is a package that interacts with Python to provide numerical operations. NumPy uses its own defined library to make things like arrays and matrices and invoke operations on them. The most basic object in NumPy is the ndarray object. The ndarray holds arrays of homogeneous data types which are compiled for efficiency. NumPy arrays have a fixed size array and are homogeneous, which is different from regular Python arrays. NumPy also has vectorized and broadcasting behavior which both work to amplify performance and decrease compile time. Vectorized code is simple and easy to read code that typically is prone to less bugs. The code sometimes is mistaken for mathematical notations, but of course, results in Python-looking code. Broadcasting describes the step-by-step behavior of operations. It is beneficial in taking the outer operation of two arrays to make a combined array. However, both arrays must be of same dimension[53].

Pandas is a Python package also automatically downloaded with Anaconda. It provides many data analysis features that are widely used in data visualization. Pandas is able to incorporate many types of data formats as well. For example, Pandas can read in tabular data from SQL or Excel, it can obtain ordered or unordered data, arbitrarily matrixed data, and it can even read in data that has no

labels. Pandas is able to handle missing values and also non-floating point data by labeling it as NaN. One important feature we specifically used was to convert data that incorporated NumPy structures into DataFrame objects. This feature allowed Pandas to easily read and control the data into an acceptable format that was able to successfully create a readable graph. Another convenient tool we used while manipulating my dataset was Pandas ability to slice and create subsets of my data. This allowed me to create new tables and graphs from carefully selected data where we saw possible correlations. To create detailed visualizations, Pandas incorporates matplotlib API [54]. Matplotlib allows the import of visualization libraries that can be read by Pandas. Matplotlib is a 2D plotting library that can be used in Python scripts and the Jupyter notebook, which we have previously mentioned. Some examples of the types of plots matplotlib includes is histograms, bar charts, scatterplots, errorcharts, power spectra, and more[55].

8.6 INSTALLATION

To install Jupyter itself you must already have Python 2.7 or Python 3.3 or greater. It is recommended to go ahead and download Anaconda, like we did, so that all of your packages are in one place. To download the latest version of Anaconda, follow the code below:

```
import webbrowser  
webbrowser.open('https://www.anaconda.com/download/')
```

From there, you will have to choose which operating system to download from. That is all you have to do to install Anaconda. You know have an open platform with many packages for use. One of these packages is Jupyter Notebook. To run Jupyter Notebook the following line into the command prompt:

```
jupyter notebook
```

Now you can begin using Jupyter's Notebook to create visuals and write code for that manipulates your data. NumPy and Pandas are also automatically downloaded with the latest version on Anaconda.

8.7 DATASET

The dataset we have chosen is often used in training tools for medical professionals and is one of the only few available skin lesion datasets. The HAM10000 (Human Against Machine with 10000 training images) dataset contains dermatoscopic images from different populations that include all general diagnostic categories that have been discovered in this type of medicine. The diagnostic categories in this dataset include diseases such as: Bowen's disease (akiec), basal cell carcinoma (bcc), benign keratosis-like lesions (bkl), dermatofibroma (df), melanoma (mel), melanocytic nevi (nv), and vascular regions (vasc). The confirmation of the samples that were entered into the dataset are given: histopathology (histo), follow-up examination (follow_up), expert consensus (consensus), or confirmation via in-vivo confocal microscopy (confocal). Each image within the dataset can be tracked by their lesion-id [56].

8.8 IMPLEMENTATION

The first part in analyzing the HAM10000 dataset is to acquire it as well as the Anaconda platform. Once you access the Jupyter Notebook and import the necessary Python packages, you are ready to begin analyzing the data. Jupyter is a great tool to use in data analysis because you can easily manipulate your code line-by-line. By performing multiple plotting algorithms, you get a great visualization of relationships amongst the dataset. Of course, there are simple methods to analyze singular features and columns in your dataset. The method below shows multiple statistical analyses on the age feature.

```
db['age'].describe()

count    9919.000000
mean     52.067749
std      16.686741
min      5.000000
25%     40.000000
50%     50.000000
75%     65.000000
max     85.000000
```

The first algorithm that was used in my Jupyter-based script was a DataFrame comparison between the localization and age features. This comparison shows the number of patients who recorded diagnosed skin lesions for different locations in addition to their differing ages. The Pandas DataFrame is a class that has the ability to take in a mutable, two-dimensional data structure that contains labeled axes[54]. It is known as the primary Pandas data structure. There are

many examples of methods that can be used with this structure on Pandas documentation. To configure a visual of this correlation, the following code was imposed:

```
#name the database file to your liking after completed download
db=pd.read_csv('534data.csv')
df3 = pd.DataFrame(np.random.randn(1000, 2), columns=['age', 'localization']).cumsum()
df3['localization'] = pd.Series(list(range(len(db))))
df3.plot(x='localization', y='age')
```

The generated plot is as shown below Figure 65 :

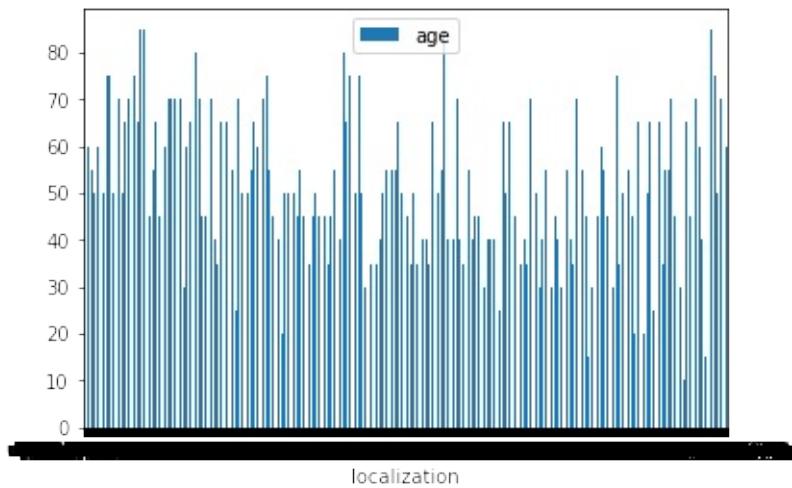


Figure 65: Correlation of localization and age

The visualization makes it obvious that there can be many different localizations per age group. This visualization also supports the need for varied datasets in the clinical domain. It ultimately makes it quite difficult to make assumptions on singular lesion samples because the locations are varied no matter the age.

Another algorithm that was imposed on Jupyter Notebook to create a visual analysis was an autocorrelation analysis. Autocorrelations are important in data analysis because they give important information regarding the quality of your dataset. Specifically, it checks randomness within your values which can also give clues to missing or empty values that you may have missed when cleaning the data. Over time, the data is compared to see if it lies near zero. If the data is considered random, the time series autocorrelation will be near zero for all time-lag separations. If the series is considered non-random, then the autocorrelations will be non-zero. The graph that is generated shows two horizontal lines that indicate 95% and 99% confidence bands. Using the following code, we have

generated the following autocorrelation plot Figure 66.

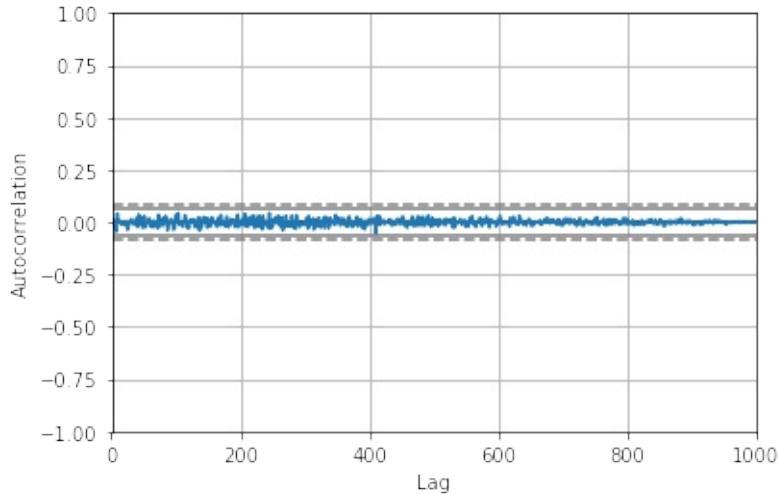


Figure 66: Autocorrelation

Since the values in the above autocorrelation plot are within the 95%, and some very close to the 99% correlation lines, it can be said that the dataset is far from random. This is promising because it reinforces the quality and trust within the dataset.

Another technical tool that we used to analyze my dataset was the KNIME Cloud Analytics Platform [57]. The KNIME cloud can be integrated via the Azure Marketplace or on Amazon AWS. KNIME stands for KoNstanz Information MinEr and is an open source data analytics software that creates services and applications for data science projects. KNIME allows its users to create visual workflows with a user-friendly drag and drop graphical interface that depletes the need for any programming. However, KNIME does allow implementation of other scripting languages such as Python [57] or R [57] that creates connections to abilities within Apache Spark or other machine learning tools. KNIME allows imports of datasets from a variety of formats, some of which include CSV [58], PDF [58], JSON [59] and more. The workflows and visualizations that KNIME produces allows export in many of these formats as well. It also supports several unstructured data types from images, documents, and certain networks. KNIME operates by a node system that includes embedded modules that help its users build their workflow. With this node system, users can make changes at every step of their analysis to ensure the most current version. KNIME also provides detailed visualizations from a set of

defined graphs and charts which can lead to predictive analyses and machine learning implementations. Users can shape their data by a variety of mathematical models such as statistical tests, standard deviations, and means. Users can even select specific features for use in possible machine learning datasets and apply filters to mark out some of the data if needed.

KNIME is a platform that can perform intense data analytics on a graphical user interface and incorporate a user-friendly workflow. It incorporates large or small data sets and even projects as broad as deep learning. KNIME is diverse in that its users do not necessarily need to know any coding languages to use it. KNIME is a process-oriented, single base workflow with basic input/output manipulations. KNIME is an open source platform that uses thousands of its documented nodes within the node repository for use in the KNIME workbench. A node is a single processing point of data manipulations within your workflow. A workflow is described as a sequence of steps a user follows in their platform that is used to complete their final product. The collection of nodes that creates a KNIME workbench is able to be executed locally or within the KNIME web portal on its own server. The workflow that KNIME follows first begins with data collection, data cleaning, data integration, and finally, feature extraction. This workflow allows for large files such as a CSV to be accessed through the web portal and it can therefore be manipulated through several wizards [60].

KNIME has the ability to be integrated with other technologies for larger open-source projects. These cloud services allow for user's projects to be analyzed even further. For example, KNIME can be used with Amazon AWS [61] and Azure [62]. KNIME's platform can be hosted on Microsoft Azure Cloud Services. Azure allows KNIME to perform its analytical, machine learning, and deep learning tasks on its integrated server. This application can be downloaded from Azure's Marketplace. KNIME can also be incorporated with Amazon AWS. When KNIME is connected to AWS resources, users can leverage the memory available while connected to the relational database service to construct SQL queries visually. KNIME's Analytic Platform is a free service for all who use it. However, if you are using KNIME on a cloud service such as Azure or AWS, there are often subscription fees associated. Students and other organizations can receive discounts or allocated amounts for a specified time of use. KNIME is a data analysis software platform that allows for easy read and manipulation of large datasets that can ultimately be used to make inferences and

predictions. Its user-friendly interface allows for a broad integration of users and sometimes more efficient workflows. KNIME has several applications for its users such as data modeling, machine learning, predictive analysis, and more. After visualization, users can extract specific features from their data and implement it into a model of their choice, which can then be exported as a CSV file.

Creating visualizations in the KNIME platform is extremely easy. The steps to create visual analyses on KNIME's analytic platform are to first drag and drop the CSV Reader node into the workflow. Once in the workflow, import the dataset from where you saved it locally by right-clicking on the CSV Reader node, and choosing Configure. Once the dialog box pops up, choose the number of rows and columns to include (we included them all). Then, choose which algorithm you would like to impose on the dataset by searching the node directory again. For my first analysis, we chose to do a basic graph showing the amount of different diagnostic descriptions per area (localization). The figure below shows this correlation, Figure 67.

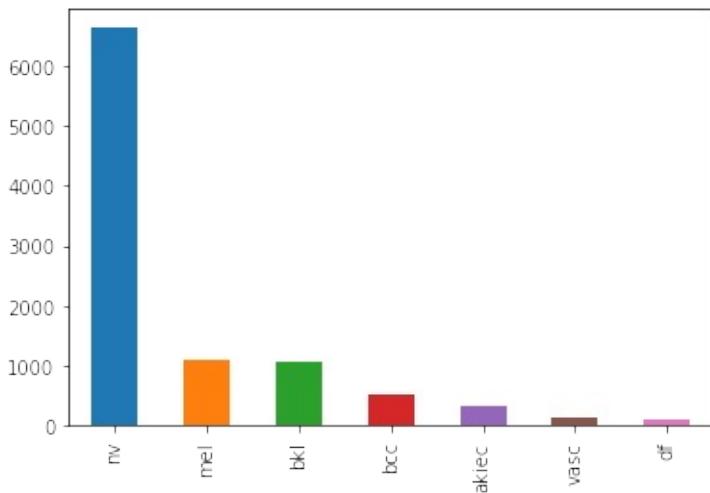


Figure 67: Descriptions by location

The above image shows proof that although some diseases are more relevant than others, it is important to have variable datasets such as these. Having so improves the accuracy of other diagnoses because of the increased comparison amongst individual units of data. It emphasizes that no matter the location, there can be many different types of apparent disease types. Another created image is the visual of density among three data features. We chose to generate this graph

with all three features along separate axes. The figure below indicates these results Figure [68](#).

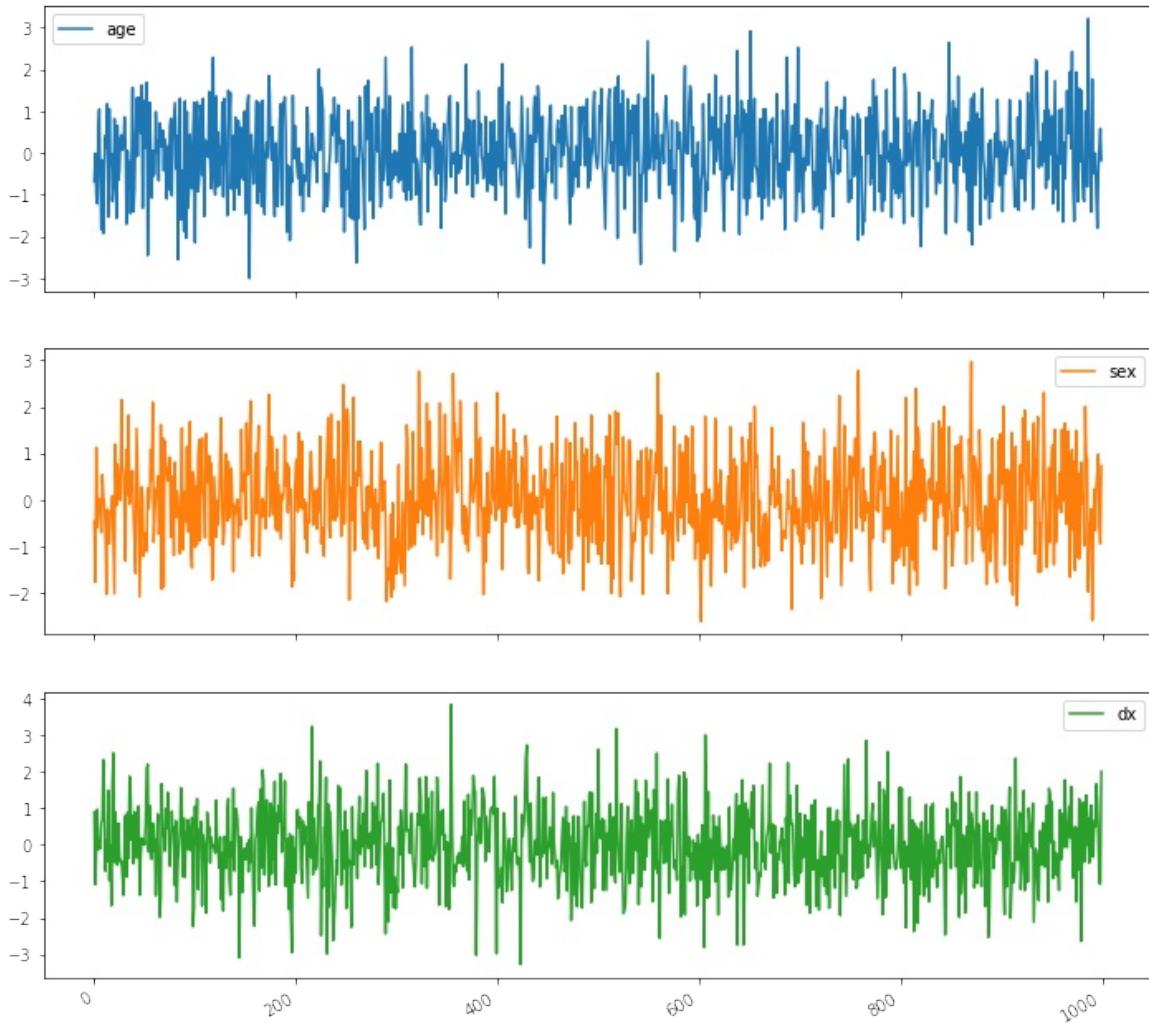


Figure 68: Subplots

Using KNIME's analytical platform, we was able to use defined algorithms to further expand my visualizations. One algorithm we explored was that of parallel coordinates. Parallel coordinates is a graph that exemplifies the many differences between multiple variables and features. In the graph I've generated, the age and sex features are placed on their own axes. The lines that connect the two features together are the actual values of the two features. Each line can be seen as an accumulation of the points that lie upon each axis [63]. The downside of this algorithm is that sometimes the dataset can make the graph look somewhat dense. The figure below shows the parallel coordinates of age and sex within the HAM10000 dataset, which is also quite dense Figure [69](#).

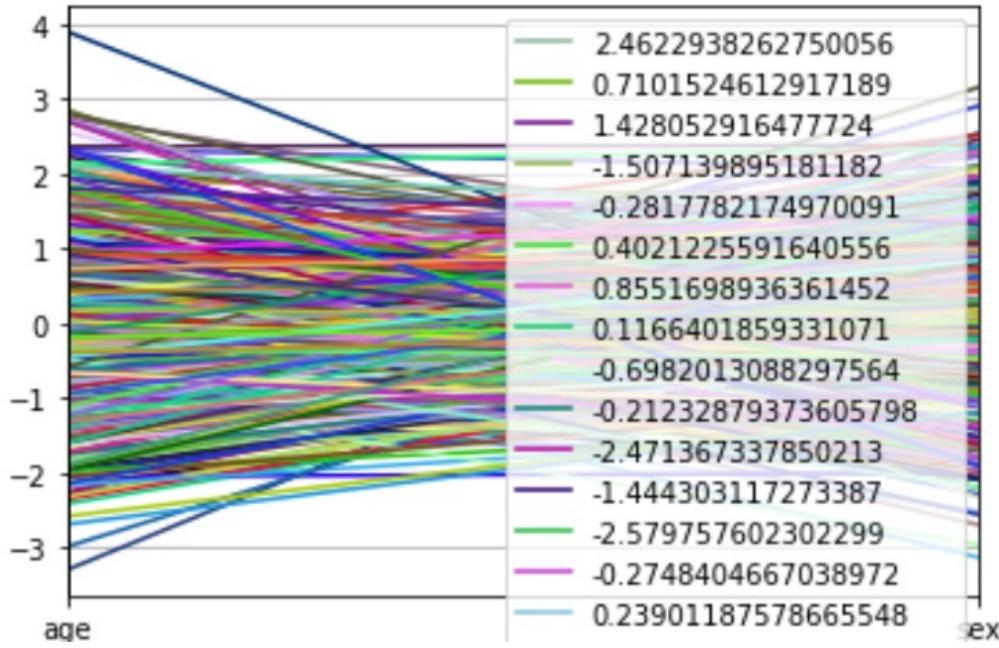


Figure 69: Parallel Coordinates of Age and Sex

KNIME allowed me to create this visualization without the use of any code, as their GUI is simple enough that programming is not always necessary. However, we also created this picture through code in Jupyter Notebook. The following script shows how we was able to do this.

```
from pandas.plotting import parallel_coordinates
df=pd.DataFrame(np.random.randn(1000, 2), columns=['age', 'sex'])
parallel_coordinates(df, 'dx')
```

Another algorithm we incorporated was logistic regression. This type of analysis compares a nominal independent and dependent variable. The regression model is used to predict a possible outcome between the two given variables[64]. The logical question we thought of when performing this analysis was if the location of skin cancer on a person's body had any influence or correlation on which type of cancer it was. The following image Figure 70 was created to show the logistic regression in this dataset. It shows that the relationship between cause of diagnosis (description/dx) and location is very much varied. There are, of course, some outliers, but it seems as though one can conclude that many different types of skin cancers can occur in numerous locations. This can be useful for medical professionals or students who are using this dataset as a training tool. It can also be said that conclusions can not automatically be made on what type of cancer is in a given region just by initial glance or statistics.

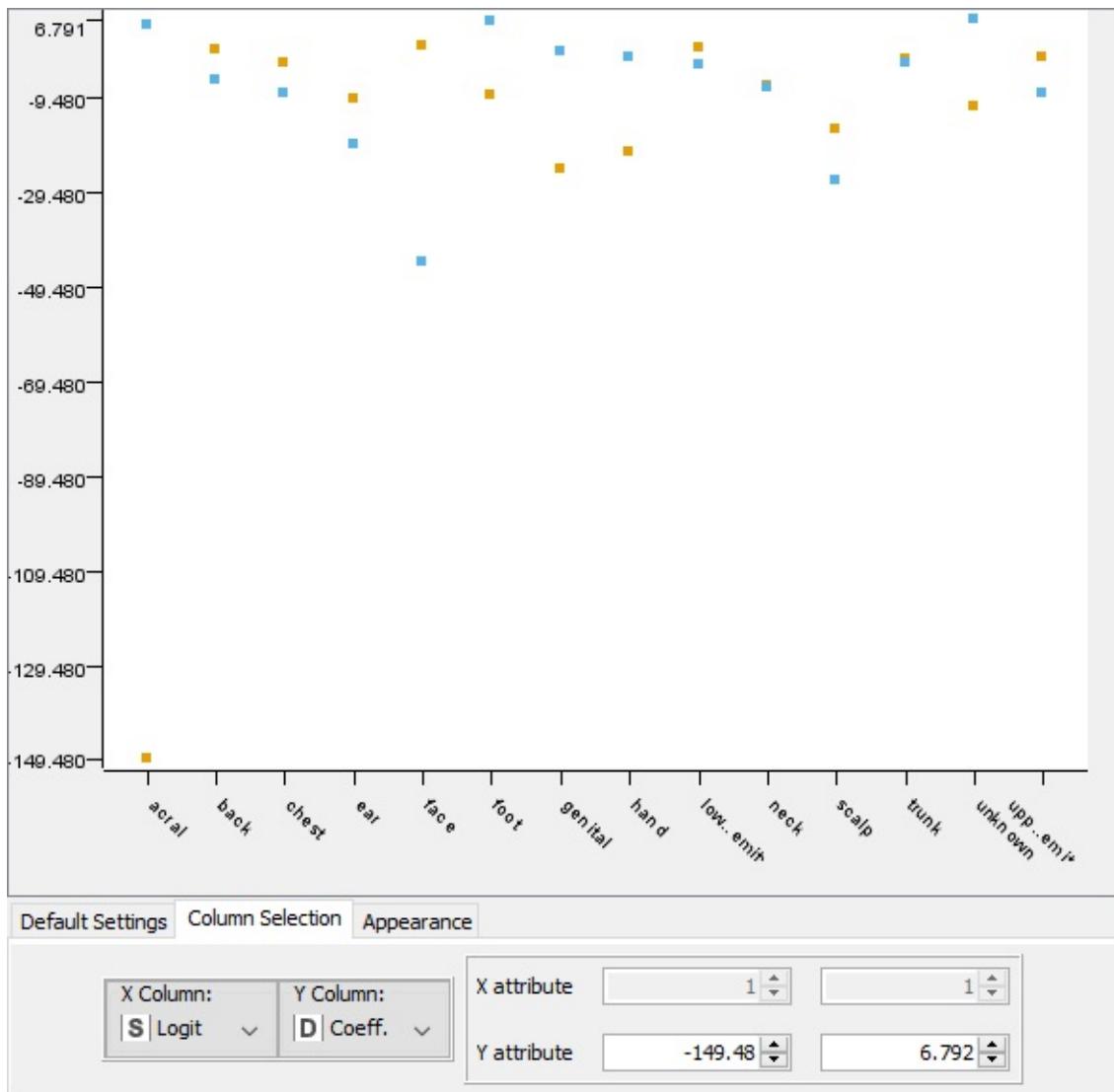


Figure 70: Logistic Regression of dx and localization features

It turns out that our analysis is correct in that different types of cancers can occur in varied locations. The figure below, Figure 71, actually shows the amount of different localizations within the dataset. Therefore, the relationship between these two features is not very strong. Everyone knows that one of the most common risks for skin cancer is sun exposure and ultraviolet light. Melanoma (mel) and basal cell carcinoma (bcc), most often caused by these factors, can occur in areas such as the face, arms, chest, back, scalp, ears, neck, and so on [65]. However, skin cancers can also occur in areas where the sun cannot reach [fa18-523-52-abchealth]. These occurrences can be caused by a genetic mutation or a change in gene regulation that causes healthy cells to divide with errors. These mutated cells could then invade other parts of the body and spread. It has

also been seen that skin cancers can possibly be caused by pollutants or toxins in the environment [fa18-523-52-abchealth].

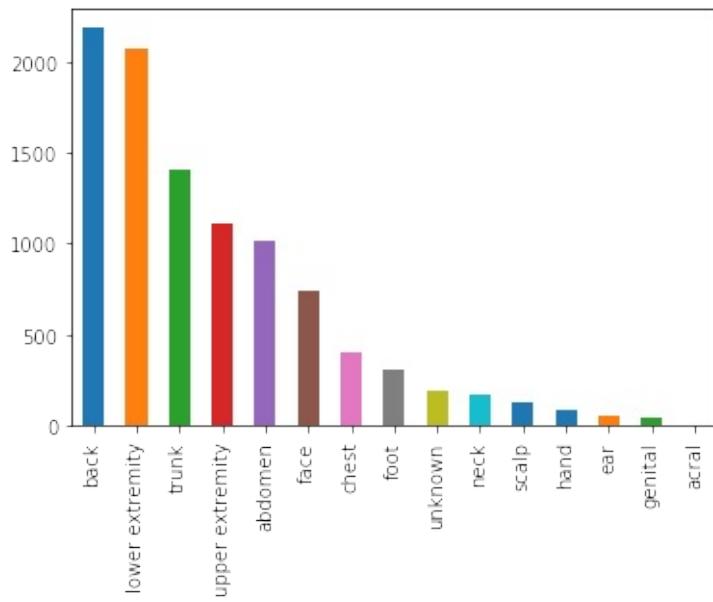


Figure 71: Skin Cancer Localizations Within the HAM10000 Dataset

A good algorithm to test the relationship between diseases and the sexes is a regression tree. A regression tree is an algorithm that predicts the results of two, usually categorical, variables. The regression model is quite easy to interpret. Using KNIME's analytical platform, we conducted a predictive regression tree diagram of the sexes within the HAM10000 dataset. The results show that the dataset includes a higher number of diseased males than females. The figure below shows the results in the model of the actual regression tree Figure 72. This analysis is also important to know because there is no evidence that strongly supports a specific sex to have a greater or lesser chance of developing a disease. It is easy to see that there are 54.7% of males with diseases in the dataset and 48.8% of females with diseases.

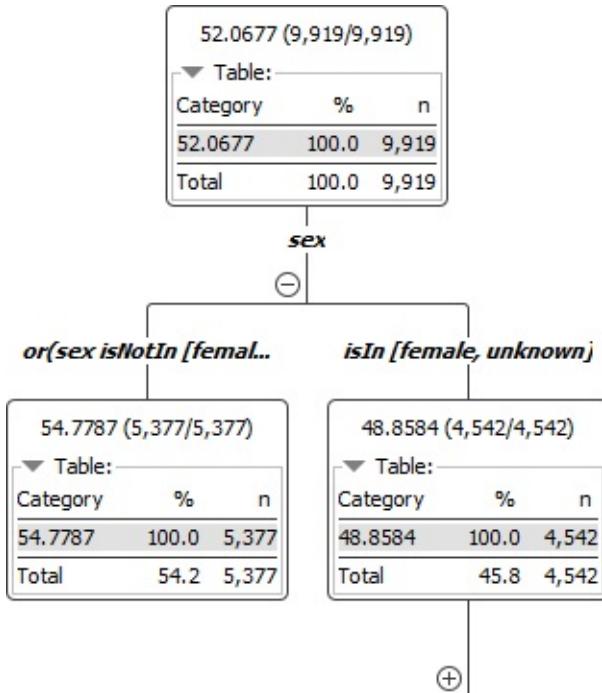


Figure 72: Regression Tree of the Sexes

8.9 BENCHMARK

The presence of big data analysis in healthcare is increasing exponentially. The burden placed on medical professionals to read and understand each patient's medical history as well as handling the current visit is sometimes too much to ask. The use of big data in the healthcare sector would make the time spent analysing each patient's medical history much shorter, resulting in an increased amount of face-to-face communication time between the patient and professional. Predictive analysis is a large advantage taking place in healthcare's use of big data today. Predictive analyses can be used to identify risks for future diseases among patients and therefore, doctors can use that information to prescribe certain medications, health-promoting activities, and other preventative measures. As mentioned earlier, electronic health records are now being used as one of the most reliable forms of big data in healthcare. It is obvious, especially in the modern age, that we realize people's health is ever-changing and there are multiple factors that weigh in for almost every medical situation. Use of big data in healthcare setting is also being seen in medical research. Analysis of recovery methods for certain drugs and treatments in cases of cancer can be used to

discover the best-working plan [66].

8.10 CONCLUSION

The use of big data analytics in healthcare is variable, but has an exciting future. The teaming up of data scientists with medical professionals will expand society's knowledge for determining the identity of a disease or even a patient's future. Big data in healthcare has the ability to give us inferences to outcomes and diseases we may have not even seen coming. The combination of these two fields will definitely speed up research and increase the rate of discovery from both sides. This starry-eyed future, however, comes at a cost. Acquiring reliable datasets are expensive, the data must be kept confidential, and it must also be extremely accurate regarding its inclusion requirements. The HAM10000 dataset, which satisfies all of the above credentials, is one of many big datasets being used for analysis among medical professionals. The analyses performed on this dataset is just the beginning of what can truly be done to explore further information into the relationship between the donated skin lesions and its multiple features.

8.11 ACKNOWLEDGEMENT

Thanks are given to Professor Geoffrey Fox and Professor Gregor von Laszewski for providing the inspiration to create the analyses displayed in this project.

9 ORCHESTRATING MICROSERVICES FOR A CREDIT SCORING APPLICATION IN KAFKA

Chaitanya Kakarala

ckakara@iu.edu

Indiana University

hid: fa18-523-53

github: [cloudycode](#)

code: [cloudycode](#)

Keywords: Kafka, Zookeeper, Microservices, Python.

9.1 ABSTRACT

This Project deals with orchestration of micro services in a Credit Scoring application using a Kafka cluster. A user keys in his personal identification information in a user interface created in Python and upon submitting the same multiple micro services written in python will be executed. These light weighted and autonomous services interact with one another using a Kafka broker which works in a subscribe-publish model. The user will then see his credit report and the factors that impact his credit.

9.2 INTRODUCTION

With the increase in the amount of data being processed there is a great need of developing the applications with better performance. One such technique to make the application perform better is breaking the application into smaller units. These units are light weighted and autonomous in nature. These small applications provides scalability and because of their autonomous nature they can be plugged into any system. These small applications are also known as Microservices. Since we are breaking the big or complex application to multiple small or light weighted microservices, a mechanism to efficiently communicate between these microservices is required. Apache Kafka is one such mechanism

which provides a message streaming platform so that the microservices can either subscribe or publish the data.

9.3 APACHE KAFKA

Apache Kafka [67] is an open source streaming platform that streams the data in the form of messages. The messages are nothing but a collection of bytes and kafka has nothing to do with the content of these messages. Each of these messages will be tagged with a topic name and these messages will be published into a partition (disk space) of the topic it is associated with. These partitions can be made available across different machines which makes kafka a horizontally scalable streaming platform. Optionally each of these messages can be given with a key and whose hash values determines the partition it should be saved in. Hence the messages with the same key value will be stacked together in the same partition. Figure 73 describes four partitions of a single topic.

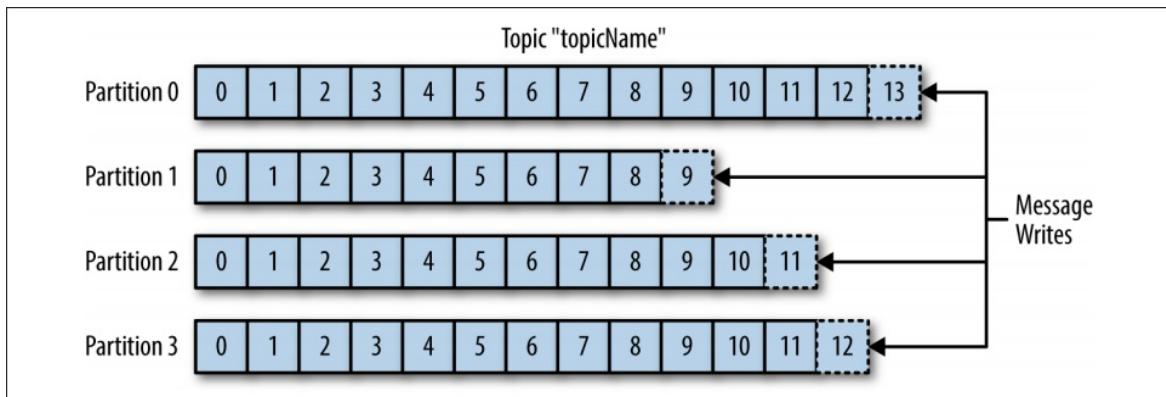


Figure 73: Representation of topic with multiple partitions [68].

There are two users of the kafka system. They are producers and consumers. Producers sends the messages and they are also known as publishers. Producers while sending the messages does not care about the partition the message is going to save. However, publishing the message with a key value ensures all the messages with same key stored in the same partition. On the other hand consumers consumes those messages by the producers. The consumers saves the offset of each message it reads and process the same. Saving the offset helps restarting from the point of failure in case of an issue rather starting all over again. These consumers can be grouped together called as consumer group and each consumer in the consumer group could be hosted in a different machine

which makes the consumer aspect scale horizontally. Consumer groups also restricts the partition to be read by multiple consumers if required. The data retention in each partition can be controlled in different ways. For example the message in a partition can be removed after one month or the partition can always be maintained at the capacity threshold set to 1 GB. Figure 74 illustrates on how consumer group works.

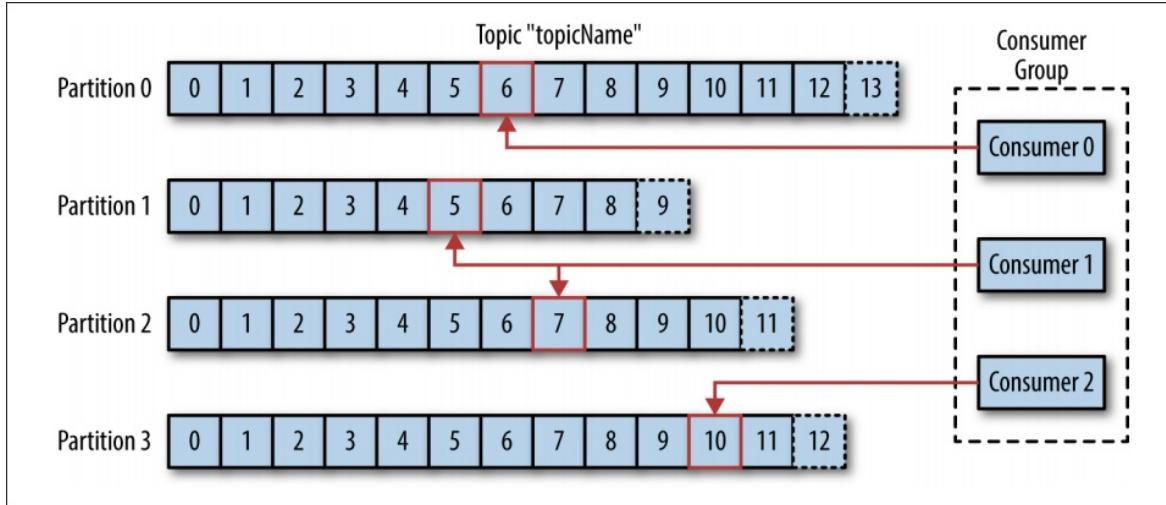


Figure 74: A consumer group reading from a topic [68].

A single kafka server is called as kafka broker. Each broker receives messages from producers and save them into their respective partitions. The broker also responds to the consumer requests and saves the offset of the consumed messages. Kafka is designed to have multiple brokers and collection of all such brokers is called as kafka cluster. A leader broker can be defined in each cluster and the data replicates from leader broker to the other brokers to provide high data availability and persistent data. Kafa also supports the communication between the clusters in different data centers. Figure 75 explains how multiple brokers are replicated in a kafka cluster.

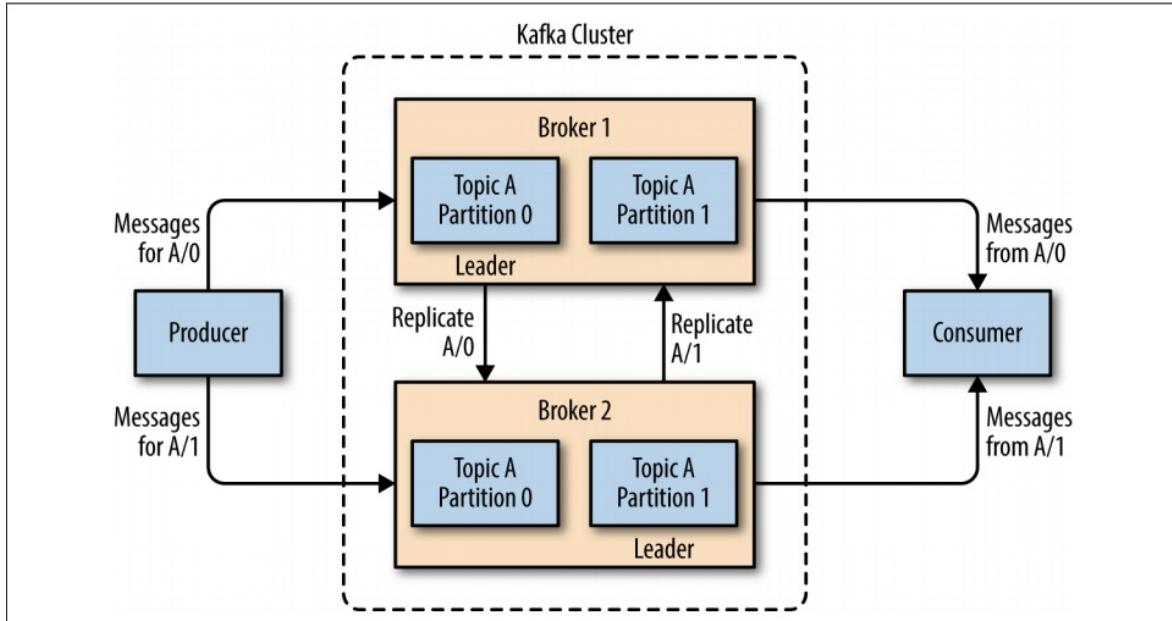


Figure 75: Representation of partitions in a cluster [68].

9.4 REQUIREMENTS

A Credit reporting agency would like to create an application to help their customers in finding their credit scores and the factors influence their score. In order to provide the above information, the application require the below personal identification information from their customers. They are:

- Name Salutation (Optional),
- First name,
- Middle Name (Optional),
- Last name,
- Name Suffix (Optional),
- Address Line 1,
- Address Line 2 (Optional),
- City,
- State,
- Zip Code,
- SSN.

Upon collecting the above information from the user, the below rules need to be applied to cleanse and standardize the user input.

- None of the name related information should contain any integers in them. They should contain only characters.
- Address lines should be standardized such as Lane to Ln and Circle to Cir.
- City and State should be characters.
- Zip code has to be integer.
- SSN should be a 9 digit integer.

After cleansing and standardizing the user input, below logic has to be applied for determining the score.

- The maximum score one can get is 850
- 10 point reduction should be applied for every credit inquiry
- Existence of a public record should result 200 point reduction
- Every missed payment will result in 100 point reduction
- If the available credit to total debt ratio is less than 10%, there will not be any reduction in score
- If the available credit to total debt ratio is between 10% to 20% , there will be a 20 point reduction in score
- If the available credit to total debt ratio is between 20% to 30% , there will be a 30 point reduction in score
- If the available credit to total debt ratio is between 30% to 40% , there will be a 40 point reduction in score
- If the available credit to total debt ratio is between 40% to 50% , there will be a 50 point reduction in score
- If the available credit to total debt ratio is greater than 50% , there will be a 100 point reduction in score
- The minimum score that one can get is 350.

The application has to be designed in such a way that the code can be packaged and implemented in any machine. The services have to be light weighted and autonomous in nature. In case of any issue with the code the user inputs must be guarded and the application should start from the last point of failure.

9.5 ARCHITECTURE

The application is designed using below technologies:

- Kafka: Kafka is used as a message streaming tool for establishing a data pipeline between multiple microservices.
- Python: Python is the programming language used for creating the microservices and kafka library is used to publish and subscribe messages to kafka clusters. Tkinter library is used for creating user interfaces.
- Zookeeper: Zookeeper is used for maintaining a centralized configuration information by providing distributed synchronization and providing group services. Apache kafka uses zookeeper for maintaining the configurations.

Figure [76](#) describes the high level Kafka Architecture

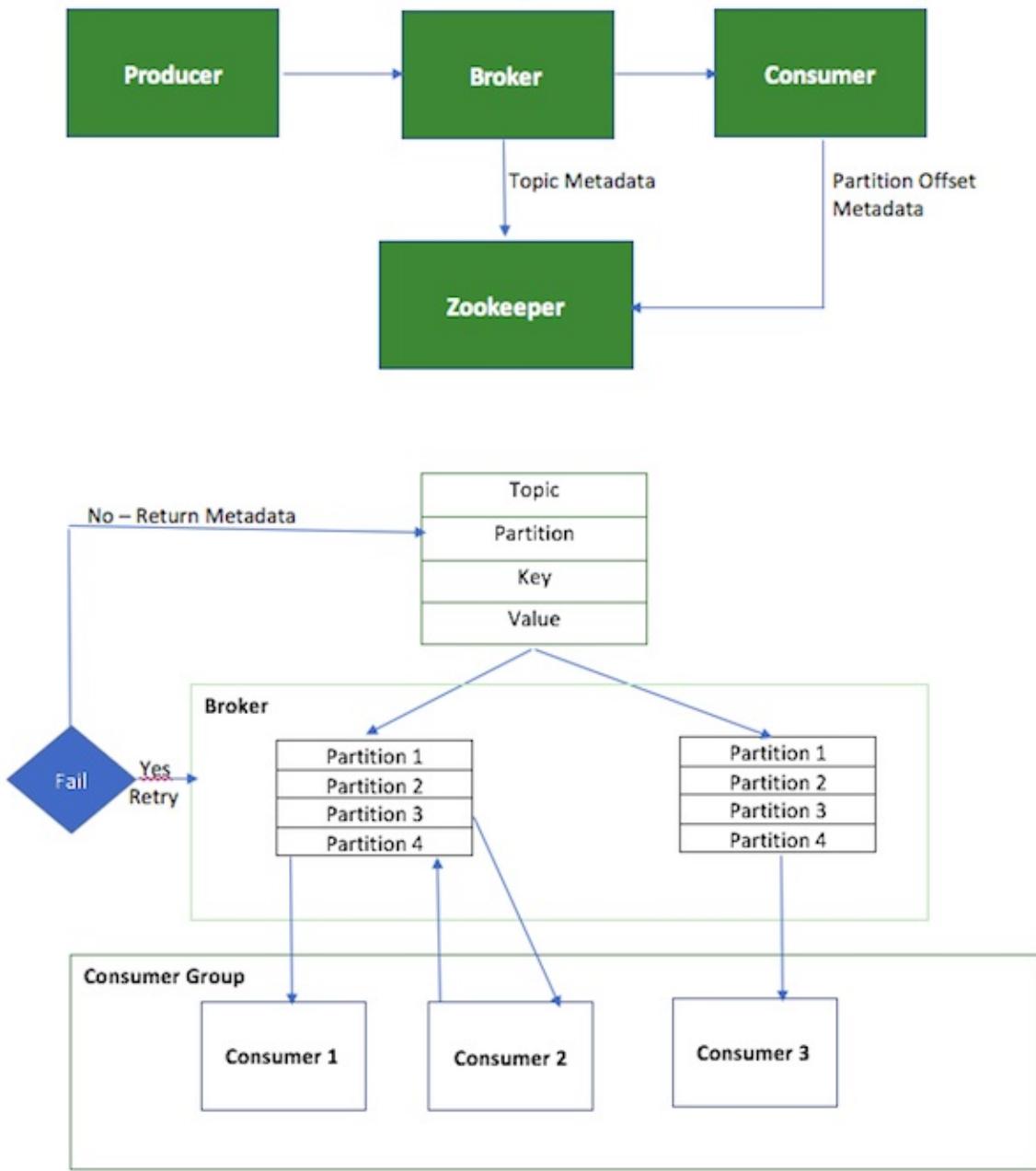


Figure 76: Kafka Architecture

9.6 DESIGN

An application is designed to have different services in python which interacts with one another using kafka streams. Below are the different microservices created as part of this application.

- scoringUI: This microservice launches a user interface for the user to input the personal identification information. The service reads each of the user inputs and create a JSON payload. The JSON payload created is sent to kafka partition under ‘nameParsing’ topic. The service is coded in python and using TKinter and Kafka libraries.
- nameParsingMicroService: This microservice is a consumes the messages from ‘nameParsing’ topic and then parse the payload. The service also applies the parsing logic to clean all the name related fields. Upon cleansing the name fields, the service creates a payload with parsed name and all the address related fields. The payload is then sent to kafka partitions under ‘addressParsing’ topic.
- addressParsingMicroService: This microservice consumes the messages from ‘addressParsing’ topic and applies parsing logic on the address related fields to define the parsed address. The service then creates a payload with parsed name and address to kafka partitions under ‘scoring’ topic.
- scoringMicroService: This microservice consumes the messages from ‘scoring’ topic from kafka cluster and applies scoring logic on the subject. This service reads the dataset called ‘creditDatabase.csv’ for obtaining the required attributes for a given SSN to calculate the score. The service then launches the user interface for the user to check his score and the factors contributing towards the same. Figure [77](#) describes how the above stated microservices are orchestrated in Kafka.

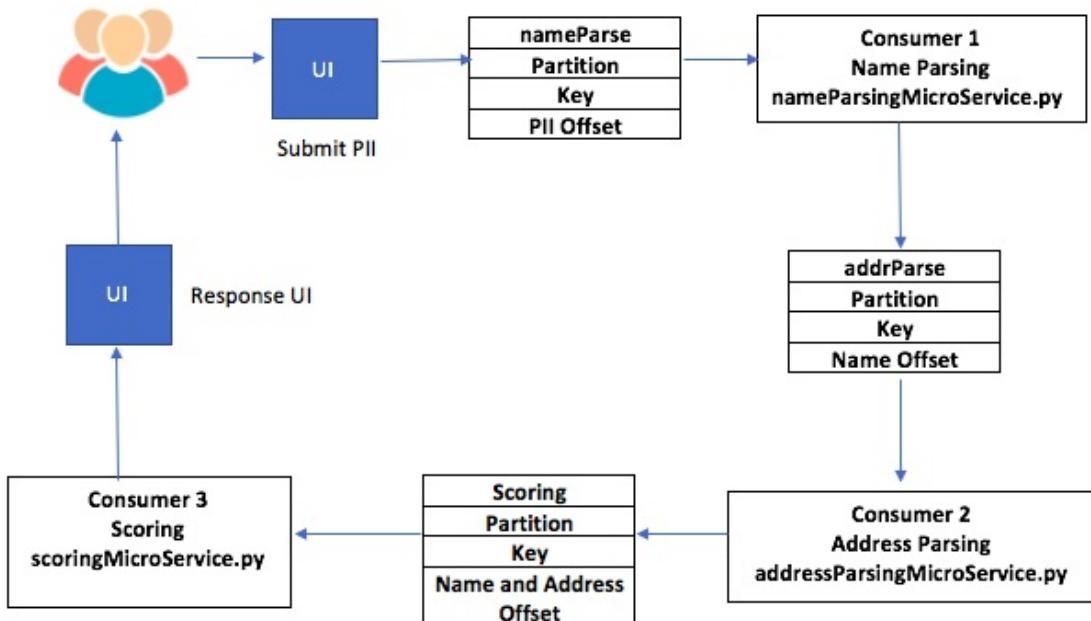


Figure 77: Scoring Application Design

Zookeeper saves the offsets of the messages consumed and produced to maintain the configuration information. Kafka requires this information in the event of restarting the application from a point of failure.

9.7 DATASET

A credit database file in csv format is created for this project. Below are the attributes of the dataset.

- SSN : Key field that uniquely identifies a person
- ACTIVE SINCE : This is the date in YYYY-MM-DD format on which date the credit was established for the person
- INQUIRIES : Number of inquiries on the record in the past 3 years.
- CREDIT LIMIT : Total credit limit of the person across all his accounts.
- TOTAL BALANCE : Sum of all current balances reported by different financial institutions.

- PUBLIC RECORD : This attribute is set to “Y” if there are any public records listed for this person. Default Value is “N”
- MISSED PAYMENTS : Number of missed payments by this person from the date of credit establishment.

9.8 IMPLEMENTATION

- Kafka Installation:
 - Kafka tar file can be obtained from [69]. Please download and save it on the server. Please be aware that kafka requires Java to be installed on the server.
 - Untar the downloaded file using below commands

```
tar -xzf kafka_2.11-1.1.0.tgz
```

- If the java version in your server is having a LTS (Long Time Support) then below fix is needed in kafka-run-class.sh located in bin folder under the kafka home directory. This is a known fix and kafka is working to address this issue for future releases [70].

Change below line

```
JAVA_MAJOR_VERSION=$(($JAVA -version 2>&1 | sed -E -n 's/.*/ version "([^. -]*).*/\1/p')
```

to

```
JAVA_MAJOR_VERSION=$(($JAVA -version 2>&1 | sed -E -n 's/.*/ version "([^. -]*).*/\1/p')
```

- install below libraries of python
 - json
 - CSV
 - os
 - re
 - kafka
 - tkinter
 - tk_tools
 - datetime
 - time

A python library can be installed using pip install commands. Here is the

example command line to install kafka library

```
pip install kafka
```

- Start the zookeeper server using the below command. You need to be in kafka home directory to be able to successfully execute the below command.

```
bin/zookeeper-server-start.sh config/zookeeper.properties
```

- Start the Kafka server using the below command. You need to be in kafka home directory to be able to successfully execute the below command.

```
bin/kafka-server-start.sh config/server.properties
```

- Execute below services in different instances
 - nameParsingMicroService.py
 - addressParsingMicroService.py
 - scoringMicroService.py

```
./nameParsingMicroService.py  
./addressParsingMicroService.py  
./scoringMicroService.py
```

- Execute the UI service

```
./scoringUI.py
```

This command opens a user interface to key in the personal identification information. Provide the details and hit “Check your Score” button. Figure 78 is the sample input screen.

The screenshot shows a user interface for a credit scoring application. At the top right, it says "Know Your Credit Score" and "Credit Scoring Application". On the left, there's a vertical list of form fields: Salutation, First Name, Middle Name, Last Name, Name Suffix, Address Line1, Address Line2, City, State, and Zip. To the right of each field is a corresponding input field containing the entered information. Below these fields is a note: "SSN-Defaulted to the one available in Dataset" followed by a text input field containing "123-45-6789". At the bottom right is a button labeled "Check your Score".

Salutation	Mr.
First Name	Chaitanya
Middle Name	K
Last Name	Kakarala
Name Suffix	Jr.
Address Line1	1234 Hollywood St
Address Line2	
City	Chicago
State	IL
Zip	60504
SSN-Defaulted to the one available in Dataset	123-45-6789
Check your Score	

Figure 78: Scoring Application User Interface

If the SSN provided in the user input does not found in ‘creditDatabase.csv’ dataset then you will see a “Credit Record Not Found” exception. If not, a user interface similar to Figure [79](#) will be opened with the credit information.

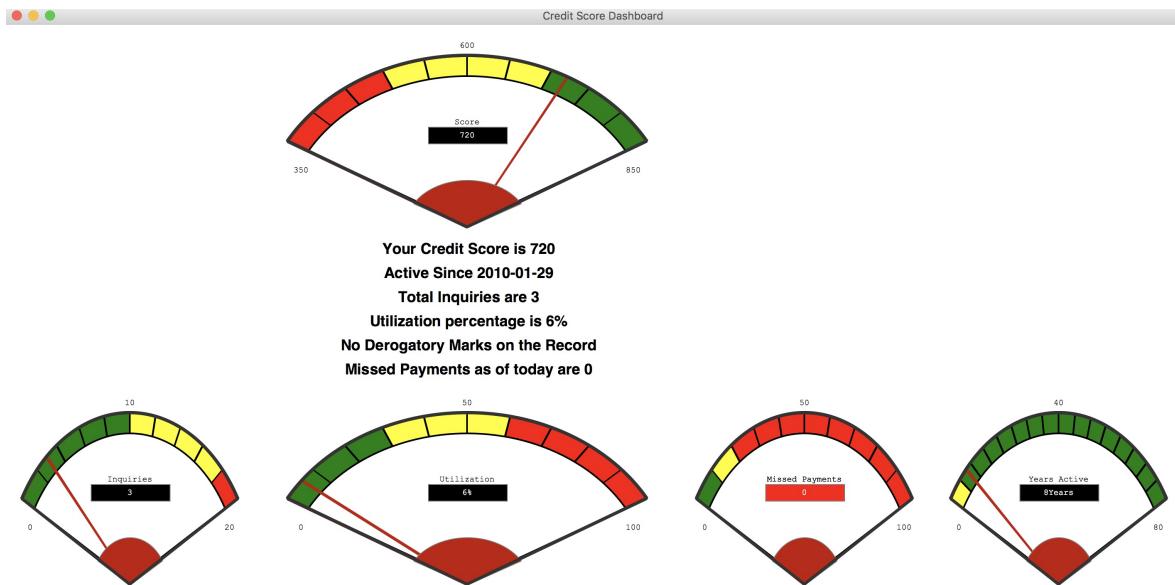


Figure 79: Scoring Application Results Interface

9.9 CONCLUSION

This project successfully demonstrates the orchestration of microservices using a kafka cluster. The four microservices created in python as part of this credit scoring application interacts with one another using kafka cluster.

9.10 ACKNOWLEDGEMENT

The author would like to thank Professor Gregor von Laszewski and associate instructors for their help and guidance.

10 APACHE NIFI

Daniel Hinders
dhinders@iu.edu
Indiana University
hid: fa18-523-56
github: [dhinders](#)

Learning Objectives

- Learn about NiFi
 - Install NiFi and setup a data stream with various processors
 - Integrate NiFi with Kafka
-

Keywords: ETL, Data Stream, NiFi, NSA,

10.1 APACHE NIFI INTRODUCTION

NiFi is a customizable tool for building flexible data flows while preserving data provenance and security [71]. NiFi provides the ability to build or alter an ETL flow with a few clicks. NiFi builds Gets, Converts, and Pulls in a GUI and allows the user to build and customize the flow [72]. This flexibility and usability is key to NiFi's value in a big data world where stovepipes and inflexibility are frequently challenges.

As pointed out in [73] NiFi is a tool for:

- *Moving data between systems, including modern systems such as social media sources, AWS cloud server, Hadoop, MongoDB, and so on*
- *Delivering data to analytics platforms*
- *Format Conversion, extracting/parsing data*
- *Data or files routing decisions*
- *Real-time data streaming*

NiFi is not recommended for:

- **Distributed Computation**
- **Complex Event Processing**
- **Join/ Aggregated Functions**

10.2 NIFI HISTORY

NiFi was first developed at the National Security Agency but was released as an open source project to the public.

“NiFi was submitted to The Apache Software Foundation (ASF) in November 2014 as part of the NSA Technology Transfer Program” [74].

Since then, Apache Foundation has used its volunteer organization to grow and mature the project [72].

10.3 NIFI FEATURES

NiFi incorporates a straightforward User Interface (UI) to engineer traceable data provenance with configurable components. NiFi offers up the ability to custom build processors and incorporate them into a highly customizable flows. Through

“data routing, transformation, and system mediation logic” [71],

NiFi seeks to automate data flow in a big data environment and gives architects the ability to keep data flowing between evolving systems quickly. Amongst a host of features NiFi offers, one sticks out as particularly important because of the challenges associated with what the feature addresses: data errors, data inconsistency, and data irregularity handling. NiFi provides users the ability to incorporate in the flow processes to catch these non-happy path realities in big data. As new situations are discovered, a user can quickly build ***if-then*** forks in the process to catch, store, or resolve the data issues.

NiFi's main features are:

- **Guaranteed delivery:** use purpose-built persistent write-ahead log and content repository to ensure guaranteed delivery in an effective way[71] [75].
- **Web-based user interface:** easy to use web-based GUI with drag and drop features that allows users to build, schedule, control, and monitor data flow[75] [71].
- **Provenance:** provide the ability to track data flows through the systems with audit trail and traceability functionalities [75][71].
- **Queue Prioritization:** provide the ability to configure and prioritize job flow and determine the order of events [75][71].
- **Secure:** provide and support multiple security protocols and encryptions, as well as authorization management [75][71].
- **Extensibility:** provide flexibility by allowing pre-built and built-your-own extension to be integrated [75][71].
- **Scalability:** supports scale-out by clustering architecture as well as scale-up and scale-down [75][71].

10.4 NiFi ARCHITECTURE

The NiFi homepage Figure 80 shows the main components in NiFi architecture.

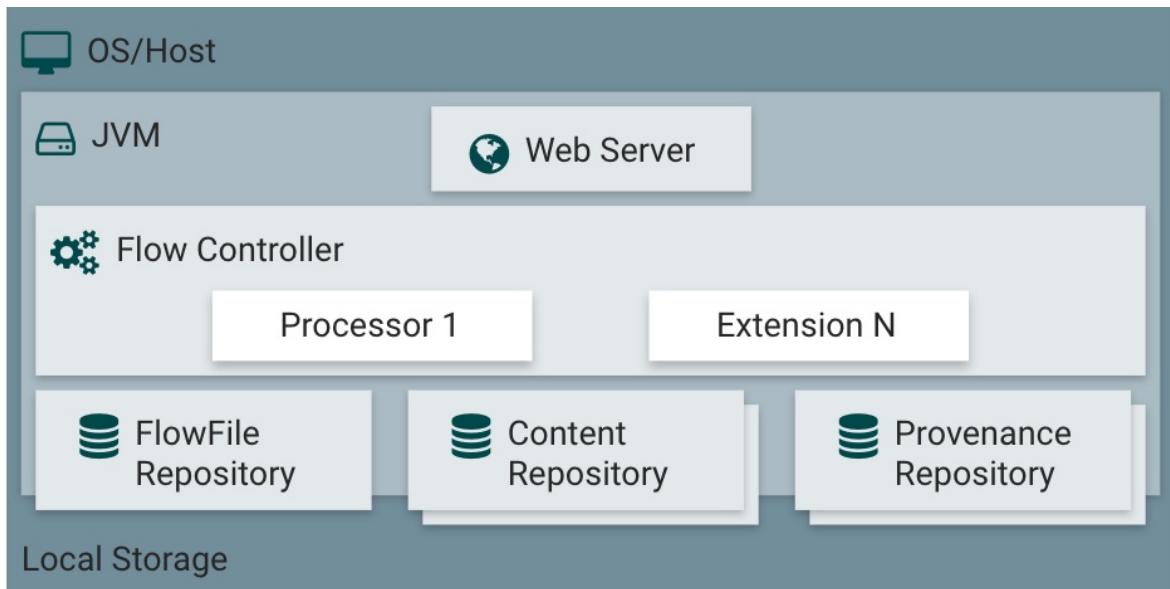


Figure 80: NiFi Architecture [75]

From the top down, NiFi is web browser accessible by a NiFi hosted Web Server. NiFi processor operations are managed through the Flow Controller and the three repositories; FlowFile, Content, and Provenance work to process data on and off disk and in a NiFi flow. NiFi is hosted in the Java Virtual Machine environment or JVM [75].

10.4.1 Web Server

NiFi's easy-to-use graphic user interface(GUI) is hosted on the Web Server within the JVM [75].

10.4.2 Flow Controller

NiFi central operations hub is the Flow Controller. Threads are managed and allocated to the processors and the FlowFiles are passed through and managed through the Flow Controller [76].

10.4.3 FlowFile Repository

Files in an active NiFi flow are tracked in a write-ahead log so that as data moved through the flow NiFi can keep track of what is known about files as they pass through[75].

10.4.4 Content Repository

The real data for a flow file is in the NiFi content repository. NiFi uses simple blocks of data in a file system to store this FlowFile data [75]. Multiple file systems can be used in order to increase speed with multiple volumes being utilized.

10.4.5 Provenance Repository

In NiFi, the provenance repository stores historic event data. The provenance data about flows is indexed to enable search of the records [76].

10.4.6 Processors

NiFi provides more than 260 processors and more than 48 controller services for users to integrate into a flow from the graphic user interface(GUI) of Nifi[77]. Processors are base on underlying controller services in the java virtual machine. Controller services can be centered around a security implementation, database CRUD (create, read, updates, and deletes), and many other foundational areas. Users can create custom processors from existing controller services or create a customer controller service as well [77].

10.4.6.1 Processor Examples

- Get
 - Examples: GetFTP, GetMongo, GetTCP, etc [75]
 - Similar input type processors: Consume, Extract, Fetch, Listen, etc

Nifi provides dozens of **Get** processor options and many other similar input type processors. A **Get** processor is commonly used to pick up a file or data and launch a FlowFile. The **Get** file processer setup typically gives configuration options to point to a host, set timing increments for polling and timeouts, set proxy settings, and more [75].

- Convert
 - Examples: ConvertJSONToSQL, Convert Record, ConvertExceltoCSVProcessor, etc [75]
 - Similar transformation type processors: Evaluate, Merge, Split, etc.

Once data is in the flow, NiFi provides dozens of processors to manipulate or transform data. The **Convert** processors can be configured to the expected schema or type from the **Get** processor and transform, edit, thin, enrich, or many other functions on the data in the flow [75].

- Put
 - Examples: PutFile, PutFTP, PutSQL, PutElasticSearch, PutAzureBlobStorage, etc [75]

- Similar output type processors: Publish, etc.

A critical part of a flow in NiFi is pushing the right data out of the flow into the right spot. There are dozens of ***Put*** processors that can be configured to set the directory to write files too. Additional configuration options are specific to the destination type to include SSL configuration, cache options, batching options, and many other configuration options based on the destination type [75].

10.4.7 NiFi Clusters

NiFi can also be integrated with ZooKeeper to operate within a cluster as shown by Figure 81 shows how ZooKeeper manages NiFi's nodes by determining the primary node, Zookeeper Coordinator, and failover node . Each of the nodes performs the same tasks but processes different dataset(s) [fa18-523-56-www-nifi-homepagetechdoc].

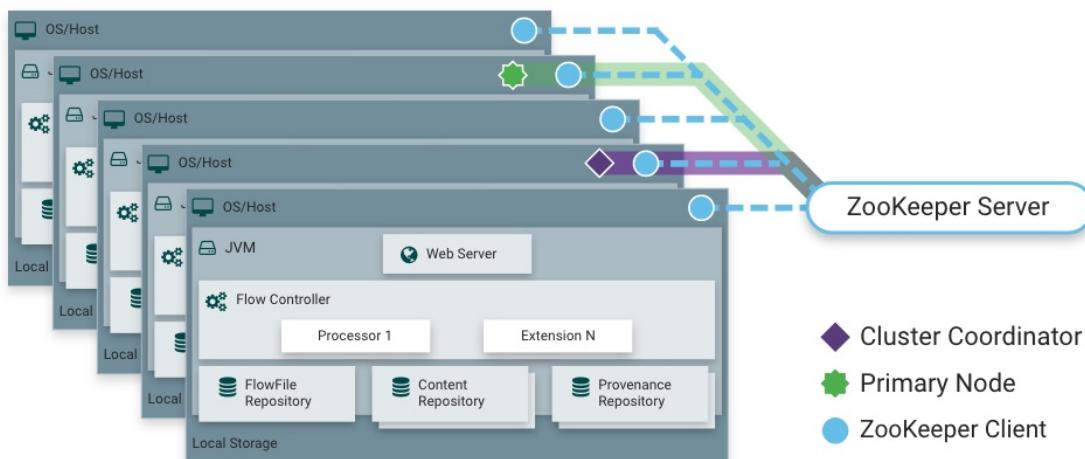


Figure 81: NiFi Cluster Architecture [75]

10.5 INSTALL NIIFI

10.5.1 Apache NiFi - Windows

(Note: Assumes Windows OS and recent verison of Java is installed)

1 NiFi can be downloaded from Apache NiFi homepage[78]. Select the latest version and the bin.zip option for the Windows instillation.

2 Unzip the install package.

3 Navigate to the configuration directory:

```
nifi-1.8.0-bin\nifi-1.8.0\conf
```

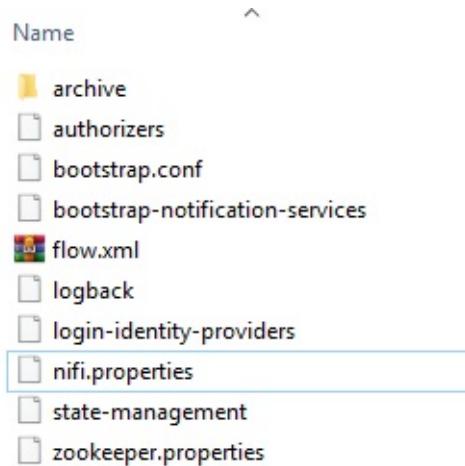


Figure 82: nifi_config

Open nifi.properties file with a text editor and edit nifi.web.http.port= to the desired port

```
# web properties #
nifi.web.war.directory=./lib
nifi.web.http.host=
nifi.web.http.port=9090
nifi.web.http.network.interface.default=
nifi.web.https.host=
nifi.web.https.port=
nifi.web.https.network.interface.default=
nifi.web.jetty.working.directory=./work/jetty
nifi.web.jetty.threads=200
nifi.web.max.header.size=16 KB
nifi.web.proxy.context.path=
nifi.web.proxy.host=
```

4 Start up NiFi by navigating to

```
nifi-1.8.0-bin\nifi-1.8.0\bin
```

Name	Date modified
bin	11/9/2018 4:18
conf	11/9/2018 5:40
content_repository	11/9/2018 4:33
database_repository	11/9/2018 4:33
docs	11/9/2018 4:18
flowfile_repository	11/9/2018 5:27
lib	11/9/2018 4:18
logs	11/12/2018 5:0
provenance_repository	11/9/2018 4:33
run	11/9/2018 4:31
state	11/9/2018 4:33
work	11/9/2018 4:32
LICENSE	11/9/2018 4:17
NOTICE	11/9/2018 4:17
README	11/9/2018 4:17

{#fig:nifi_bin}

Run the windows batch file

```
run-nifi.bat
```

Wait about 5 minutes for NiFi to load

5 Open the NiFi GUI by opening a browser and navigate to

```
http://localhost:9090/nifi
```

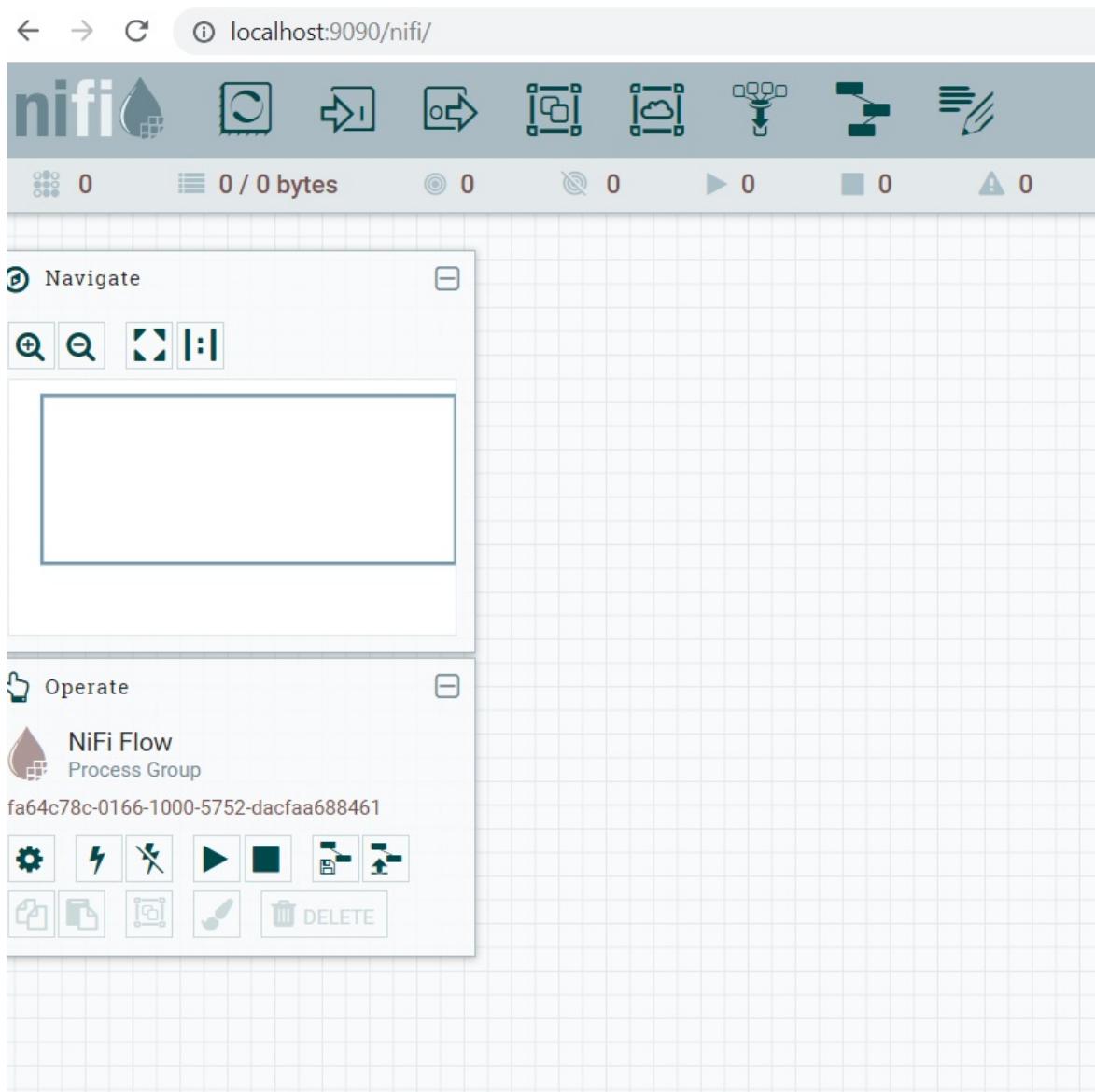


Figure 83: nifi_nifihome

10.6 BUILDING A NIIFI FLOW

1 Add a TailFile Processor by clicking and dragging the processor icon from the top tray to add a processor. Type into the filter “tail” and select the TailFile processor and click ADD

Displaying 1 of 286

Type	Version	Tags
TailFile	1.8.0	file, log, tail, restricted, text, sou...

Figure 84: nifi_processor_tailfile

2 Configure the TailFile Processor by right-clicking on the process and click configure.

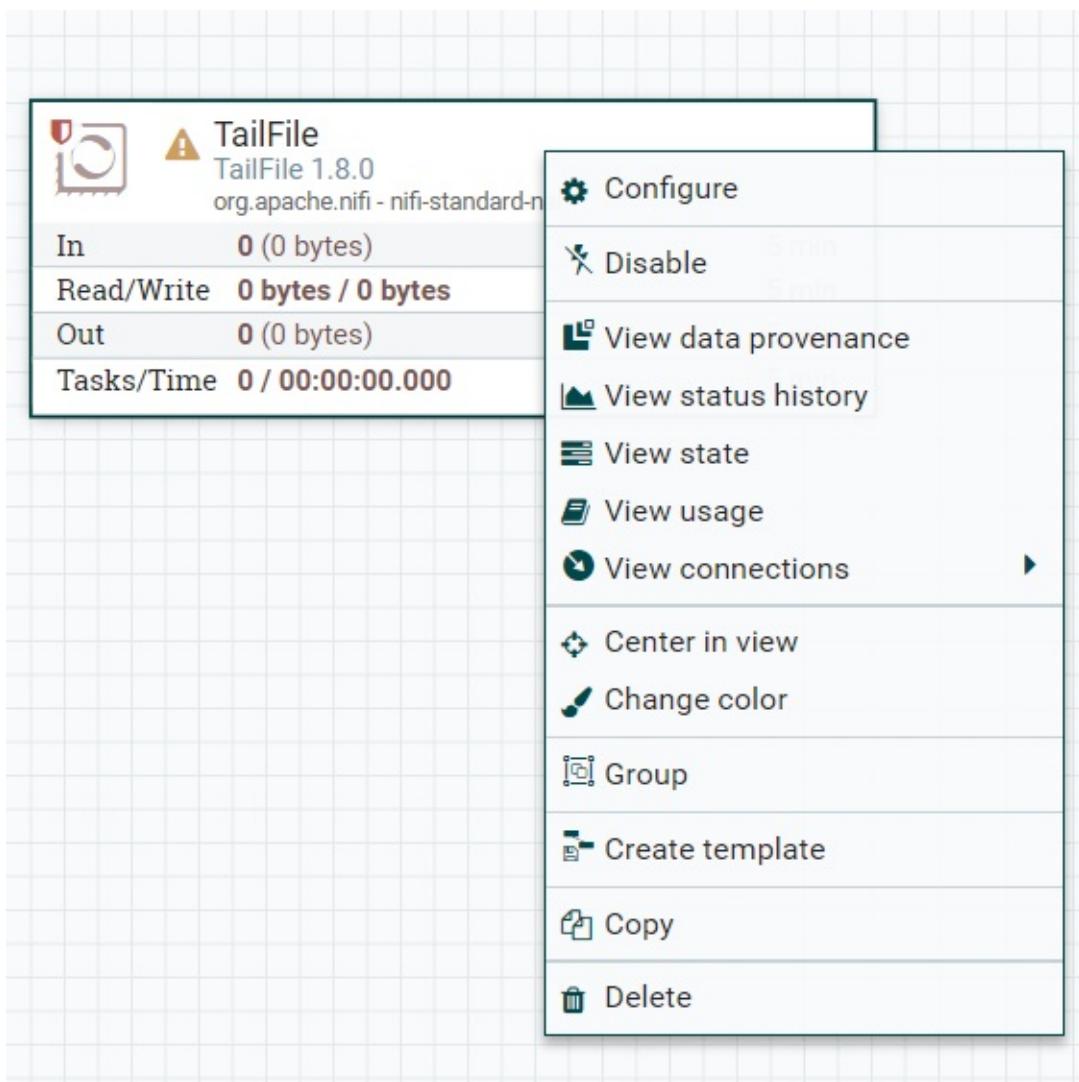


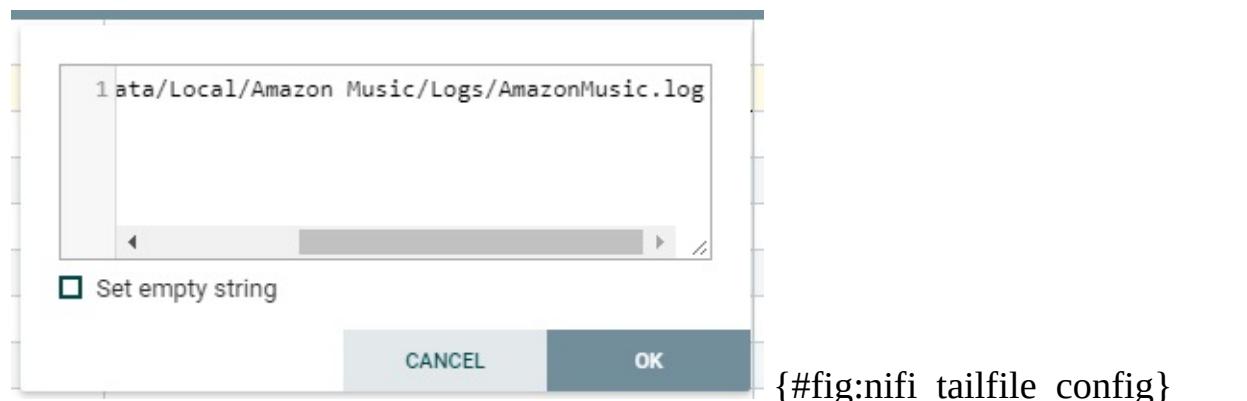
Figure 85: nifi_processor_config

Click the properties tab and click the value for the property “File(s) to Tail”

A box will appear to paste the location of the file to tail. For this example I will use a log file for a music player because it will provide a lot of data.

Use / slash when inputting file path

/AppData/Local/Amazon Music/Logs/AmazonMusic.log



{#fig:nifi_tailfile_config}

Click OK and then click APPLY

3 Add a processor called SplitText

Open the configuration options for the processor and on the settings tab in the options for Automatically Terminate Relationships check the boxes “failure” and “original”

Configure Processor

SETTINGS	SCHEDULING	PROPERTIES	COMMENTS
Name SplitText <input checked="" type="checkbox"/> Enabled	Automatically Terminate Relationships ? <input checked="" type="checkbox"/> failure If a file cannot be split for some reason, the original file will be routed to this destination and nothing will be routed elsewhere <input checked="" type="checkbox"/> original The original input file will be routed to this destination when it has been successfully split into 1 or more files <input type="checkbox"/> splits The split files will be routed to this destination when an input file is successfully split into 1 or more split files		
Id 190be21a-0167-1000-3657-ee3781173047	Type SplitText 1.8.0	Penalty Duration ? 30 sec	Yield Duration ? 1 sec
Bulletin Level ? WARN			

Figure 86: nifi_splittext_config2

NOTE

This provides direction if there is a failure at this step if a file can't be split any what to do with the original file after it is split. This flexibility that NiFi provides requires extra configuration choices but provides the NiFi admin extensive control over every aspect of the flow being built.

Click the properties tab and change the property Line Split Count to a value of “1”

SETTINGS	SCHEDULING	PROPERTIES	COMMENTS												
Required field															
<table border="1"><thead><tr><th>Property</th><th>Value</th></tr></thead><tbody><tr><td>Line Split Count</td><td>1</td></tr><tr><td>Maximum Fragment Size</td><td>No value set</td></tr><tr><td>Header Line Count</td><td>0</td></tr><tr><td>Header Line Marker Characters</td><td>No value set</td></tr><tr><td>Remove Trailing Newlines</td><td>true</td></tr></tbody></table>				Property	Value	Line Split Count	1	Maximum Fragment Size	No value set	Header Line Count	0	Header Line Marker Characters	No value set	Remove Trailing Newlines	true
Property	Value														
Line Split Count	1														
Maximum Fragment Size	No value set														
Header Line Count	0														
Header Line Marker Characters	No value set														
Remove Trailing Newlines	true														

Figure 87: nifi_splittext_config

This will split each line of the log file into one row that will be processed independently in the rest of the flow.

4 Add a processor called RouteOnContent

Open the configuration options for the processor and on the settings tab in the options for Automatically Terminate Relationships check the box “unmatched”

Click the properties tab and change the property Match Requirement to “content must contain match” click OK

Click the + in the upper right corner to add a property.

NOTE

This property will be used to select a word or phrase from the rows of the log file. When the word is seen in the row the content of the row will be routed down stream. For this example we will use "AddToLibrary" when a user in the music player adds a song to the library. We will use "ClientImplWinHTTP.cpp:525" which is the tag in the log when a song plays in the music player

After naming the new property click OK

Click on the value and use the NiFi expression language insert the tags to use to select the rows for processing.

NOTE

The NiFi expression language can be found on the Apache NiFi website[@fa18-523-56-www-nifi-expressionlanguageguide].

We will use these tags for our new properties:

```
\bAddToLibrary\b  
\bClientImplWinHTTP.cpp:525\b
```

SETTINGS	SCHEDULING	PROPERTIES	COMMENTS
Required field			
Property		Value	
Match Requirement	?	content must contain match	
Character Set	?	UTF-8	
Content Buffer Size	?	1 MB	
addsong	?	\bAddToLibrary\b	
playsong	?	\bClientImplWinHTTP.cpp:525\b	

Figure 88: nifi_routeoncontent_config

Once all properties have been added click APPLY

5 Link the processes by hovering over the TailFile processor click on the arrow that appears and drag to connect it to the SplitText Processor

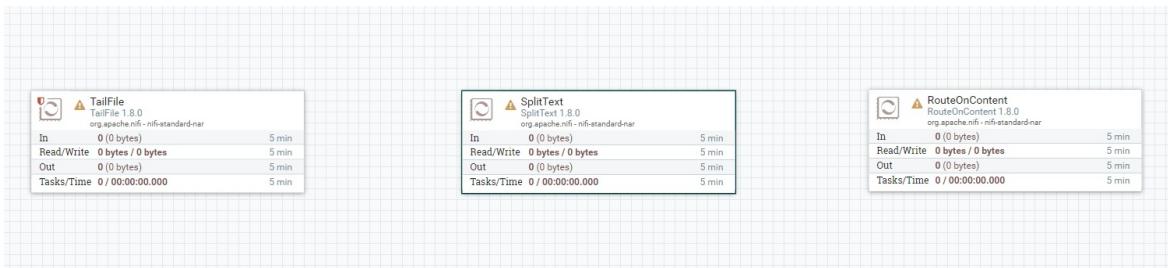


Figure 89: nifi_flow1

A window will appear to create the connection. Click ADD to connect the processors

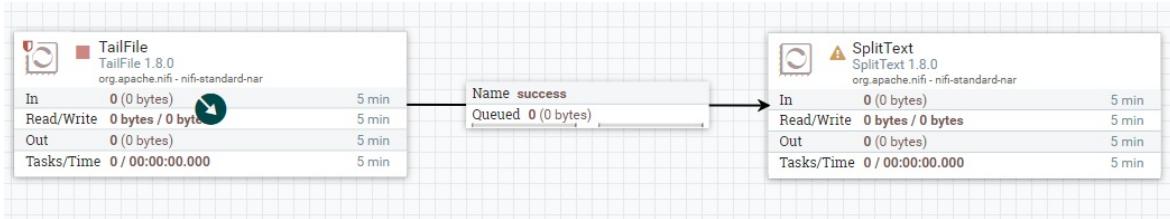


Figure 90: nifi_flow1

Connect the SplitText Processor to the RouteOnContent processor

Configure the connection in the column For Relationships, check the box “splits”

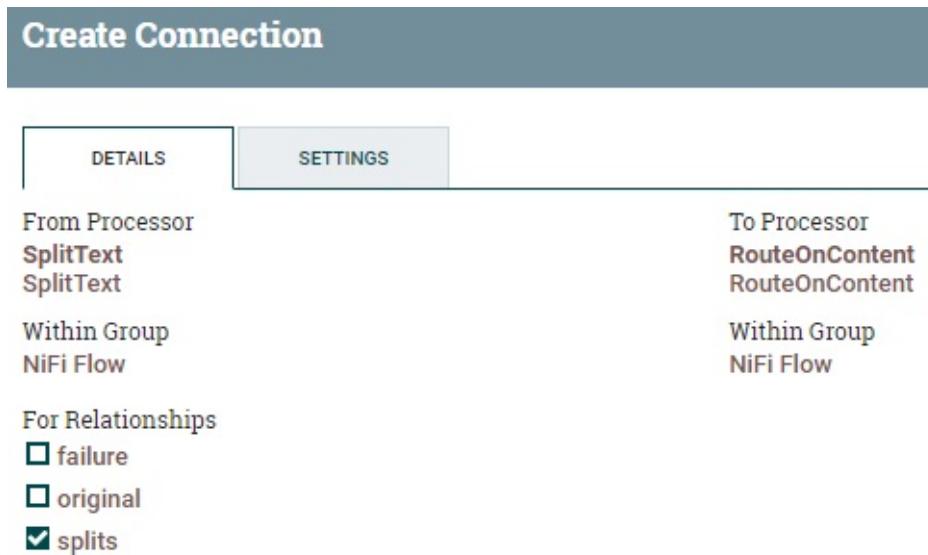


Figure 91: nifi_connection1

NOTE

This configuration for this connection will route the rows we selected from the log file that were split out. Another path could be created to handle the original files or the failures.

6

Right click on the RouteOnContent processor to open the configuration options for the processor and on the settings tab in the options for Automatically Terminate Relationships check the boxes “playsong” and “addsong” and click APPLY.

The flow is now complete, it will read a log file, select and split inputs into rows based on parameters and routes the selected rows for output. But we have chosen

to terminate the output at the RouteOnContent processor so that we run this simple flow first before connecting the flow to an external consumer.

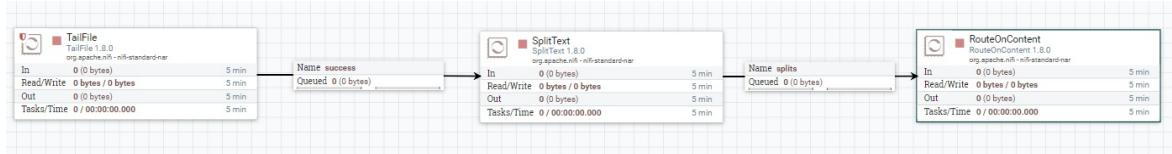


Figure 92: nifi_flow3

Select all five components in the flow with the shift key held down, then right-click on any component and select Create template.

Type a name for the template and click CREATE.

Click and drag the Template icon from the top tray to add to the workspace and a dialog box will ask what template to add and you can select the template that was just saved.

7 To run our completed flow we need to turn on individual components or start them all at once.

Select all components with the shift key held down and click on the play button on the operate panel on the left side of the workspace.

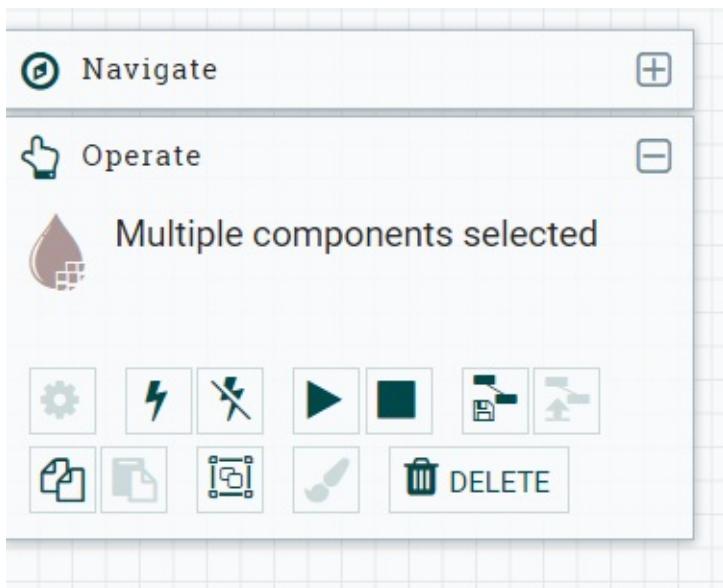


Figure 93: nifi_operate

After a few seconds right-click anywhere in the workspace and click refresh.

There is will statistics on each processor for data flowing through the flow.

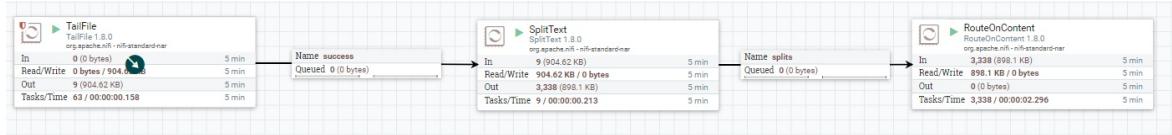


Figure 94: nifi_flow4

10.7 LINKING NIIFI FLOW TO APACHE KAFKA

NiFi provides numerous endpoint processors pass data out of the NiFi flow to a new host. One example is Apache Kafka, these directions for setting up Kafka are for Windows.

1 Start setting up Apache Kafka by download the latest version of Kafka from the Apache Kafka website. The download will include Apache ZooKeeper.

2 Configure Zookeeper and Kafka by navigating to the configuration directory:

```
kafka_2.11-2.0.1\kafka_2.11-2.0.1\config
```

Open zookeeper.properties file with a text editor and edit clientPort= to the desired port and the dataDir= to the desired folder for logs

```
clientPort=2181  
clientPortAddress=localhost
```

Open server.properties file with a text editor and edit zookeeper.connect= to the port selected for zookeeper and the dataDir= to the desired folder for logs

```
zookeeper.connect=localhost:2181
```

3 Start the ZooKeeper server by opening cmd prompt and set directory to

```
\kafka_2.11-2.0.1\kafka_2.11-2.0.1\bin\windows
```

type:

```
zookeeper-server-start.bat c:\(path)\kafka_22.11-2.01\config\zookeeper.properties
```

Hit enter and the ZooKeeper server will start up

```
[2018-11-23 12:51:04,692] INFO tickTime set to 3000 (org.apache.zookeeper.server.ZooKeeperServer)
[2018-11-23 12:51:04,692] INFO minSessionTimeout set to -1 (org.apache.zookeeper.server.ZooKeeperServer)
[2018-11-23 12:51:04,693] INFO maxSessionTimeout set to -1 (org.apache.zookeeper.server.ZooKeeperServer)
[2018-11-23 12:51:04,721] INFO Using org.apache.zookeeper.server.NIOServerCnxnFactory as server connection factory (org.apache.zookeeper.server.ServerCnxnFactory)
[2018-11-23 12:51:04,725] INFO binding to port localhost/127.0.0.1:2181 (org.apache.zookeeper.server.NIOServerCnxnFactory)
```

Figure 95: nifi_zookeeper_startup

4 Start Kafka server by opening another cmd prompt and set directory to

```
\kafka_2.11-2.0.1\kafka_2.11-2.0.1\bin\windows
```

type:

```
kafka-server-start.bat c:\(path)\kafka_2.11-2.0.1\config\server.properties
```

Hit enter and the Kafka server will start up

```
2018-11-23 17:31:48,439] INFO [Partition warn-0 broker=0] warn-0 starts at Leader Epoch 0 from offset 0. Previous Leader Epoch was: -1 (kafka.cluster.Partition)
2018-11-23 17:31:48,452] INFO [ReplicaAlterLogDirsManager on broker 0] Added fetcher for partitions List() (kafka.server.ReplicaAlterLogDirsManager)
```

Figure 96: nifi_kafka_startup

5 Create two Kafka topics to so that we can put files data from the NiFi flow into the Kafka topics

To add the addsong topic open a new cmd prompt and set directory to

```
\kafka_2.11-2.0.1\kafka_2.11-2.0.1\bin\windows
```

type:

```
kafka-topics.bat --create --zookeeper localhost:2181 --replication-factor 1 --partitions 1
--topic addsong
```

[79]

Hit enter and the new topic will be created

```
\windows>kafka-topics.bat --create --zookeeper localhost:2181
--replication-factor 1 --partitions 1 --topic addsong
Created topic "addsong".
```

Figure 97: nifi_kafka_addtopic

Repeat to add the process type:

```
kafka-topics.bat --create --zookeeper localhost:2181 --replication-factor 1 --partitions 1
```

```
--topic playsong
```

6 Add NiFi Processors to push data from the NiFi flow to Kafka

Select all three processors with the shift key held down and click on the stop button on the operate panel on the left side of the workspace.

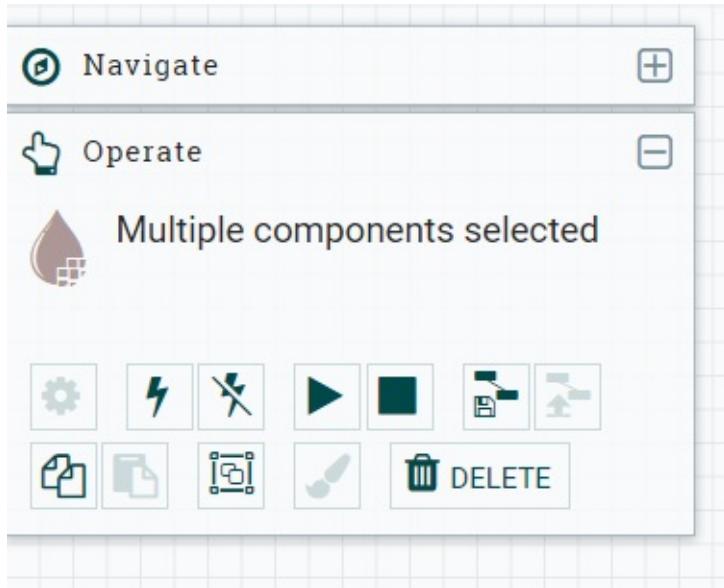


Figure 98: nifi_operate

Add a PublishKafka processors to the flow. On the settings tab check the success check box. On the properties tab provide the port for the Kafka Broker.

```
localhost:2181
```

On the properties tab provide the Topic Name: addsong

Repeat this process by adding another PublishKakfa processor and configure it the same except set the Topic Name: playsong

Configure Processor			
SETTINGS	SCHEDULING	PROPERTIES	COMMENTS
Required field			
Property			Value
Kafka Brokers	localhost:2181		
Security Protocol	PLAINTEXT		
Kerberos Service Name	No value set		
SSL Context Service	No value set		
Topic Name	playsong		
Delivery Guarantee	Best Effort		
Kafka Key	No value set		
Key Attribute Encoding	UTF-8 Encoded		
Message Demarcator	No value set		
Max Request Size	1 MB		
Acknowledgment Wait Time	5 secs		
Max Metadata Wait Time	5 sec		
Partitioner class	DefaultPartitioner		
Compression Type	none		

Figure 99: nifi_publish_kafka_config

6 Configure the RouteOnContent processor, on the settings tab in the options for Automatically Terminate Relationships uncheck the boxes “playsong” and “addsong” and click APPLY.

Configure Processor			
SETTINGS	SCHEDULING	PROPERTIES	COMMENTS
Name	RouteOnContent	<input checked="" type="checkbox"/> Enabled	Automatically Terminate Relationships <small>?</small>
		<input type="checkbox"/> addsong	
		<input type="checkbox"/> playsong	
		<input checked="" type="checkbox"/> unmatched	FlowFiles that do not match any of the user-supplied regular expressions will be routed to this relationship
Id	19715052-0167-1000-08f3-5a1a614d988f		
Type	RouteOnContent 1.8.0		

Figure 100: nifi_routeoncontent_config_kafka

7 Link the RouteOnContent processor and a PublishKafka processor by hovering over the RouteOnContent process and click on the arrow that appears and drag to connect it to the PublishKafka Processor

The Create Connection settings will appear, For Relationships, check the

appropriate topic being used by the PublishKafka processor that is linked, addsong, or playsong. Repeat the link from the RouteOnContent processor and the other PublishKafka processor and check the correct relationship.

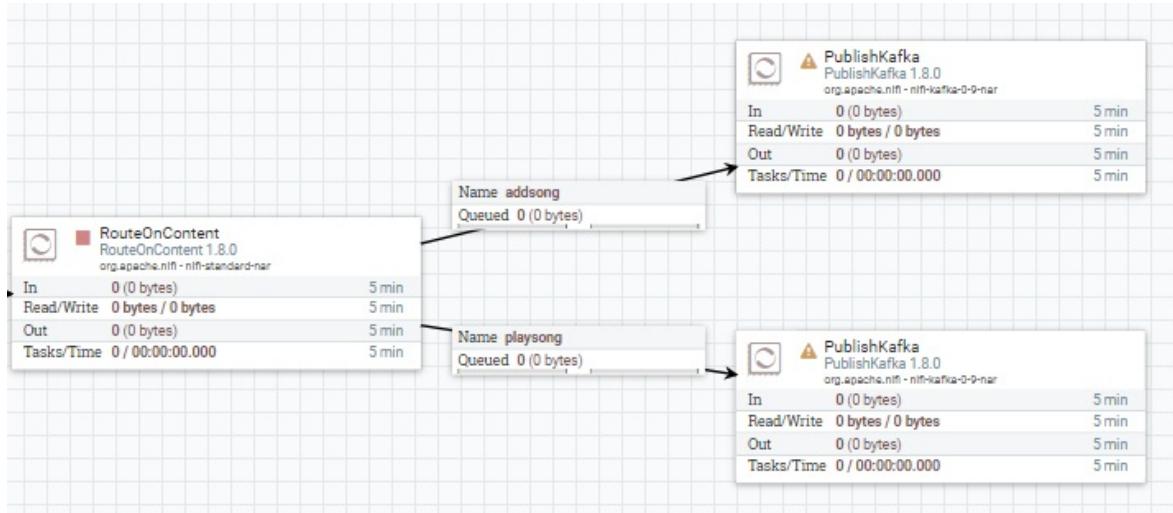


Figure 101: nifi_kafka_linked

NOTE

Hover over the yellow exclamation point on one of the PublishKafka processors. The error will say "'Relationship failure' is invalid because Relationship 'failure' is not connected to any component and is not auto-terminated" This is a good example of the robust validation in NiFi to ensure flows have the proper fail-over properties in place.

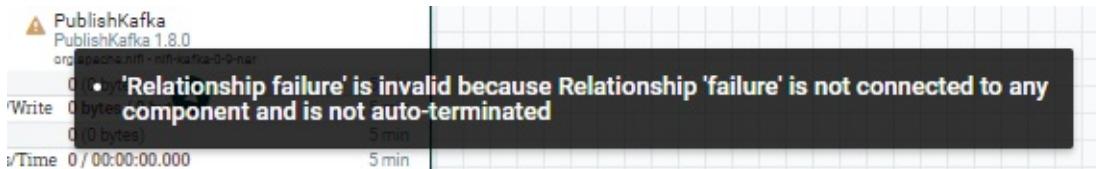


Figure 102: nifi_kafka_error

8 To create a pathway for any data that fails in the PublishKafka processors hover hovering over the PublishKafka processor and click on the arrow that appears and drag to connect it with itself.

The Create Connection settings will appear, For Relationships, check the failure box.



Figure 103: nifi_kafka_fail

Repeat for the other PublishKafka processor.

The flow is ready to run.

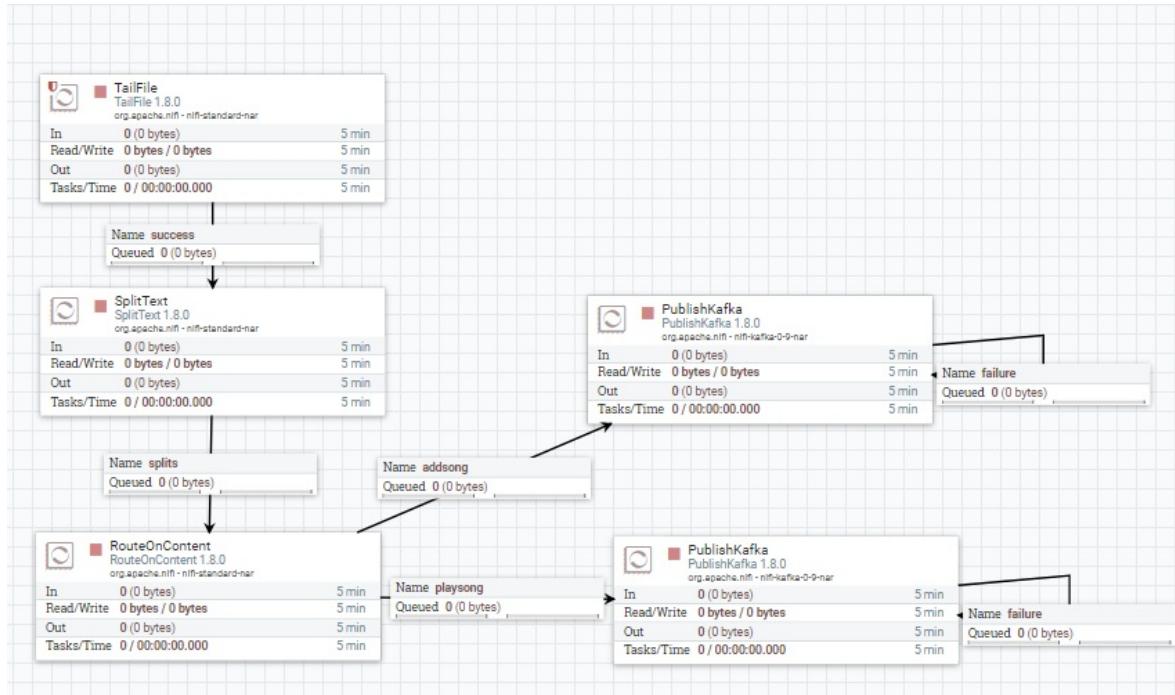


Figure 104: nifi_nifi_kafka_finalflow_notrun

Select all processors while holding down the shift key, right click on any of the processors and click start.

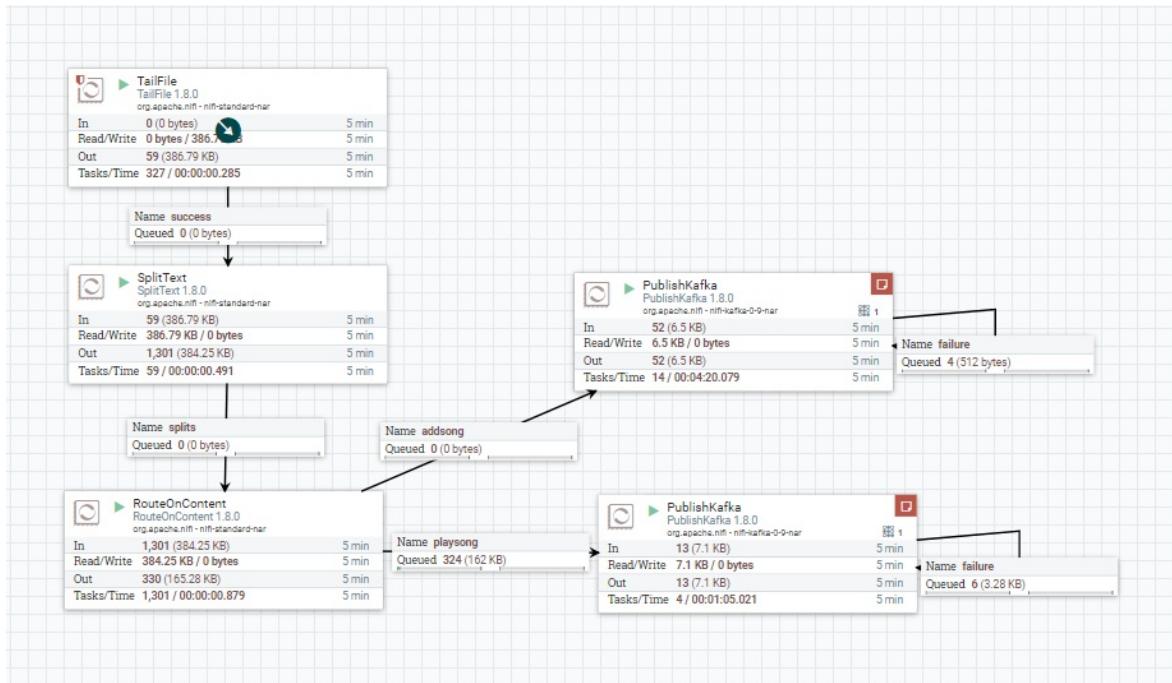


Figure 105: nifi_nifi_kafka_finalflow_run

Data from the selected rows of the log files are flowing all the way through the NiFi flow to the Kafka cluster.

10.8 CONCLUSION

NiFi is extensible tool that is flexible enough to solve many big data challenges. With the easy to use drag-and-drop user interface and configurable options it is a great tool for even non-programmers to tackle a big data challenge.

11 HISTORICAL STORM DATA ANALYSIS WITH COSMOS DB

Divya Rajendran, Pramod Duvvuri

divrajen@iu.edu, vduvvuri@iu.edu

Indiana University, Indiana University

hid: fa18-523-57, fa18-523-58

github: [cloud](#)

code: [cloud](#)

Keywords: e534, fa18-523-57, fa18-523-58, [Data Visualization](#), Data Analysis, Cosmos DB, [Machine Learning](#)

11.1 ABSTRACT

The decisions people make these days are all data-driven. The amount of data we collect has increased exponentially and this makes it harder to visualize and understand. We can overcome this hurdle by sampling and visualizing subsets of the data. Sampling refers to the process of collecting random subsets of the data. The only thing one should remember is to pick a good sample of the data or to repeat the process of sampling and visualizing the data subset. In this project, we visualized the historical storm dataset and obtained inferences from our visualizations.

11.2 INTRODUCTION

With the disastrous effects of the increasing number of storms all around the world, we thought of taking storm dataset for entire Asia Pacific region and visually analyze the effects of various variables on the storms. We want to check any patterns in the storm data and visualize them geographically so that it might help us to make useful inferences on the storm data. We have identified a data set on data.world[80], a community where we find open datasets from various organizations containing varying data attributes according to the need. We try to identify the various correlations among the attributes and understand if they can help explain the change in frequency of the storms.

11.3 IMPLEMENTATION

11.3.1 Data set

The data set for the various storms in Asia Pacific region for the years between 1956 and 2017. We have chosen an initial subset of data to visualize locally and then implement the same visualizations on the entire data. The dataset contains attributes like region, latitude, longitude, type of category, name, date and hour, speed, and pressure. There are more than 190,000 rows in the dataset. We clean the data for any duplicates, null values, and any redundant data before we generate any visualizations.

Our dataset contains five different regions surrounding the oceans like Indian Ocean, Atlantic Ocean, Southern Pacific Ocean, West Pacific Ocean, and Eastern Pacific Ocean and the regions are named according to the ocean covering the locations. We have four different storm types like S for Tropical Storms, H for Hurricanes, D for Depressions, and U for low pressure and high-speed storms.

11.3.2 Related Work

We identified an earlier work on tropical storm data using R by Stoltzman consulting LLC [81] and have used it as a base reference to visualize our data using Python.

11.4 TECHNOLOGIES USED

1. Python [82] is an object-oriented programming language we have utilized in this project
2. The data is being stored on a Microsoft Azure Cosmos DB [83] instance and we will be using Mongo API to connect and retrieve the data. The data is stored in the cloud, we are using an instance of Microsoft Cosmos DB, a NoSQL database, for storage. The output of our visualization is shown in Jupyter notebook.
3. Matplotlib [84] is a 2D visualization library containing a module plotly which gives us publication-ready images and we aim to utilize this

- library to show the basic correlation between attributes
4. Seaborn [85] is a visualization tool based on Matplotlib used to draw attractive statistical plots and we aim to show the change in the number of storms per year and other advanced correlations between different attributes
 5. Folium [86] is an interactive mapping tool which plots the data on a map based on the latitude and longitude values
 6. For Machine Learning, we use the Python package Scikit-Learn [87]

11.5 DATA VISUALIZATION

This section contains visualizations of the data and the inferences we draw from them and mostly contains our analysis and inferences from them. We attempt to answer a few research question through these visualizations.

1. We start with the basic relationship between the number of unique storms per Year

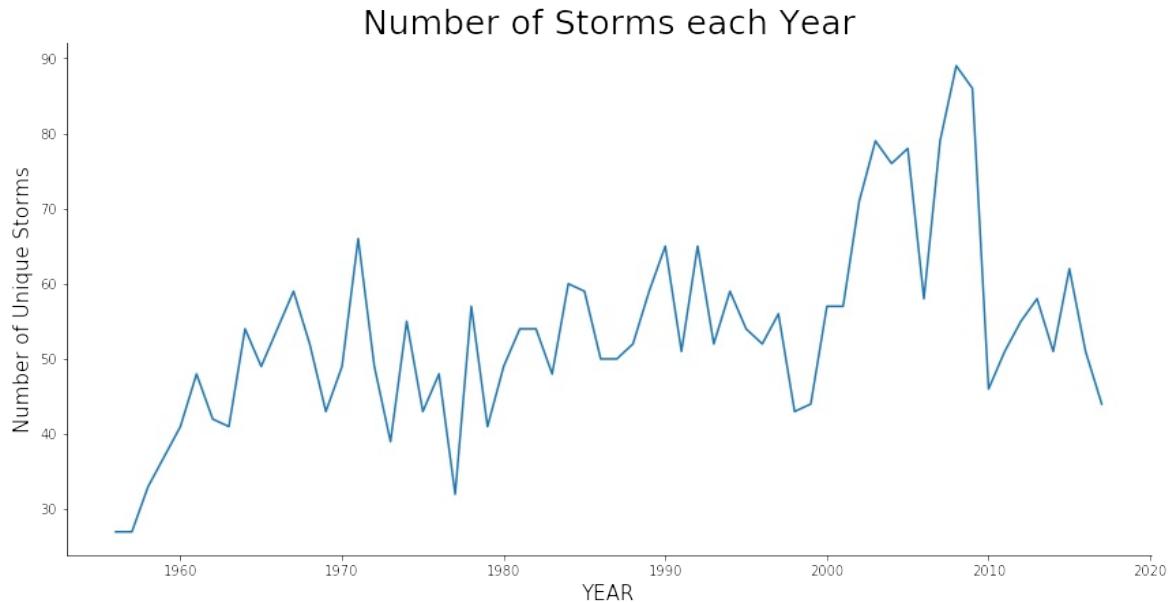


Figure 106: Storms per Year

We have plotted a linear relationship model between the number of storms and the storm year. We can see from the plot that the rate of increase in the number of unique storms through the years 2000 to 2010 and then reduce to the same rate as between the years 1950 to 1990. But, we cannot assume that the number

of storms increased every year, we need to look at other factors such as the correlation between other attributes describing the storms.

2. Number of Storms per basin

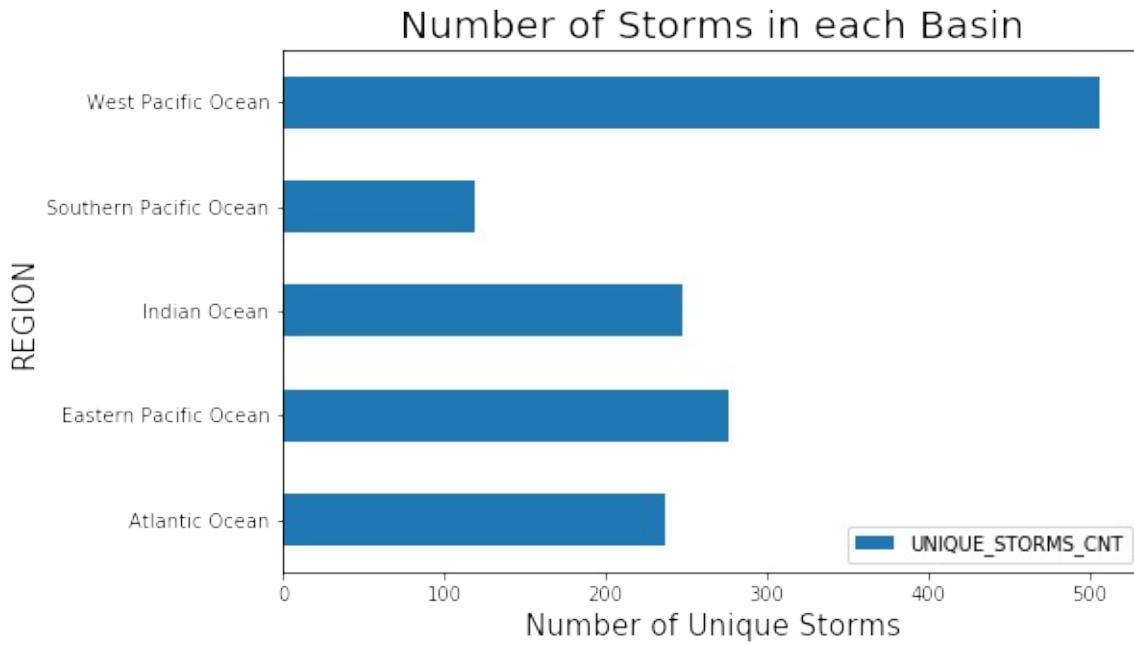


Figure 107: Storms per Region

We plotted a horizontal bar graph between the number of storms occurred in each region and identified that the Western Pacific Ocean region had a higher number of storms than in other regions

3. Number of storms per year per region

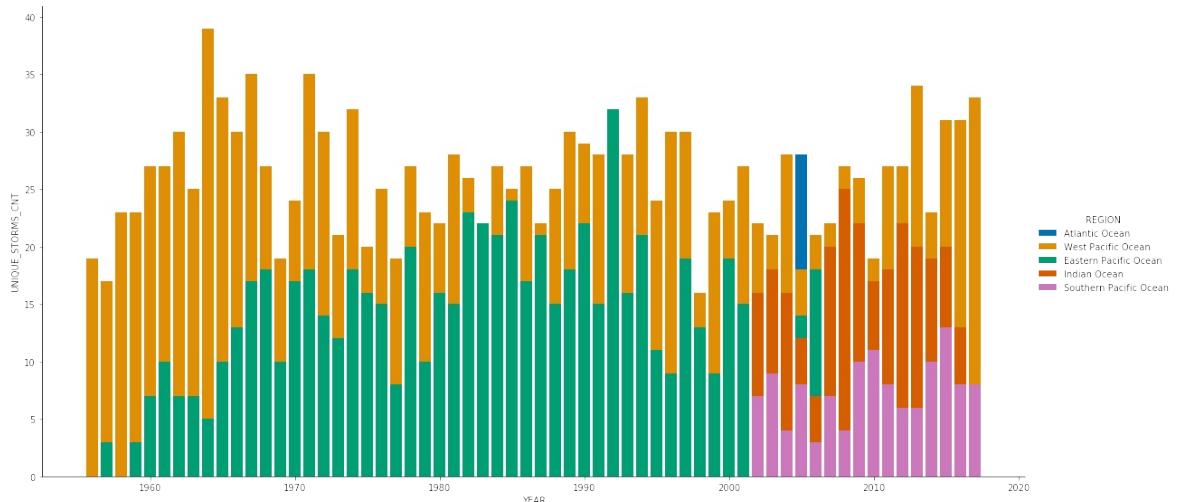


Figure 108: Storms per Year and Region

We plotted a bar graph between the number of storms per year colored by the region in which the storm was observed. The storms were increasing each year in every region and we found that the storms in the region “West Pacific Ocean”, were higher than the storms in the region “Eastern Pacific Ocean” during the years 1950 to 1965 and decreased during the years 1965 to 1994. As per our plot, the regions “West Pacific Ocean” and “Eastern Pacific Ocean” had a higher number of storms during 1950 and 2000

4. Number of storms per region per type of storm

Number of Storms each Year grid by Region and Type of Storm



Figure 109: Storms per Region and type over the years

We took another step and plotted a pairwise plot between the storm count per region and category type over the years and we observed that in all the five regions, we had a sizable number of storms of types D, H and S when compared to the storm type U

5. The relationship between wind and pressure

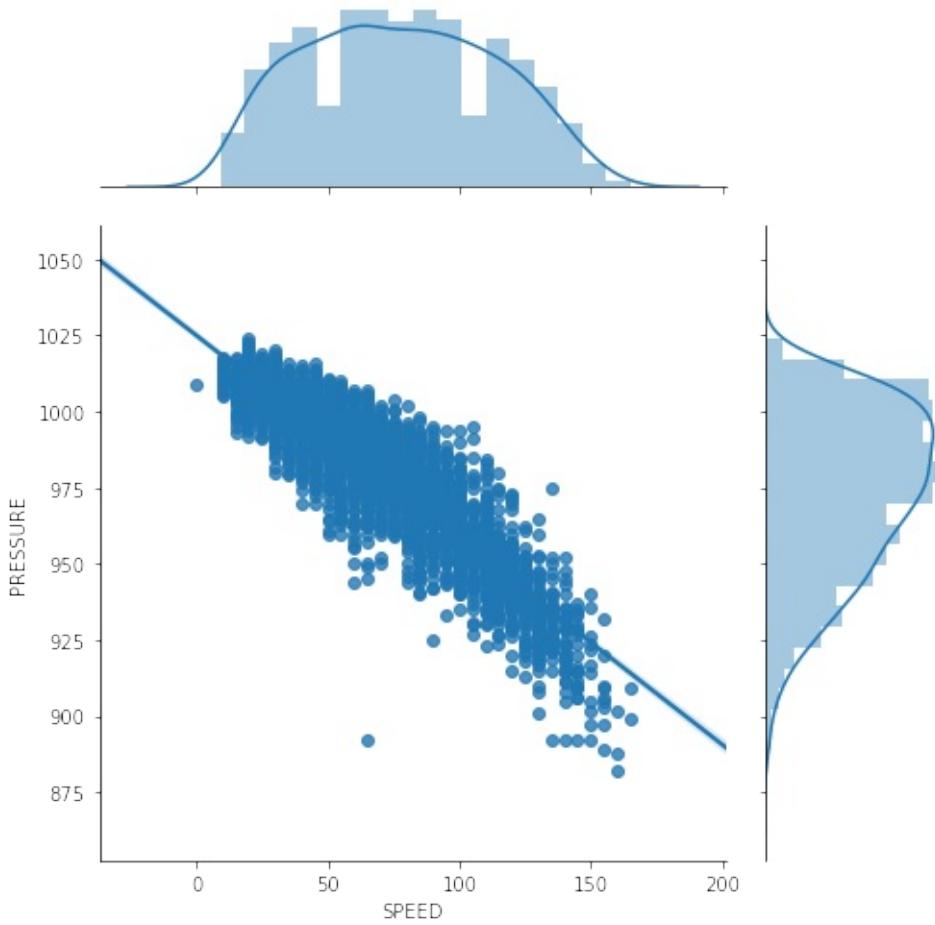


Figure 110: relationship between wind and pressure

When we plot the linear relationship plot between speed and pressure along with their distributions, we can see that the pressure distribution looks more normal. We clearly observe that the pressure is inversely proportional to speed, that is, higher the speeds lower the pressure and vice-a-versa

6. Correlation between Speed, pressure and category type is identified by visually analyzing them

Effect of Wind Speed and Pressure on High Category Storms

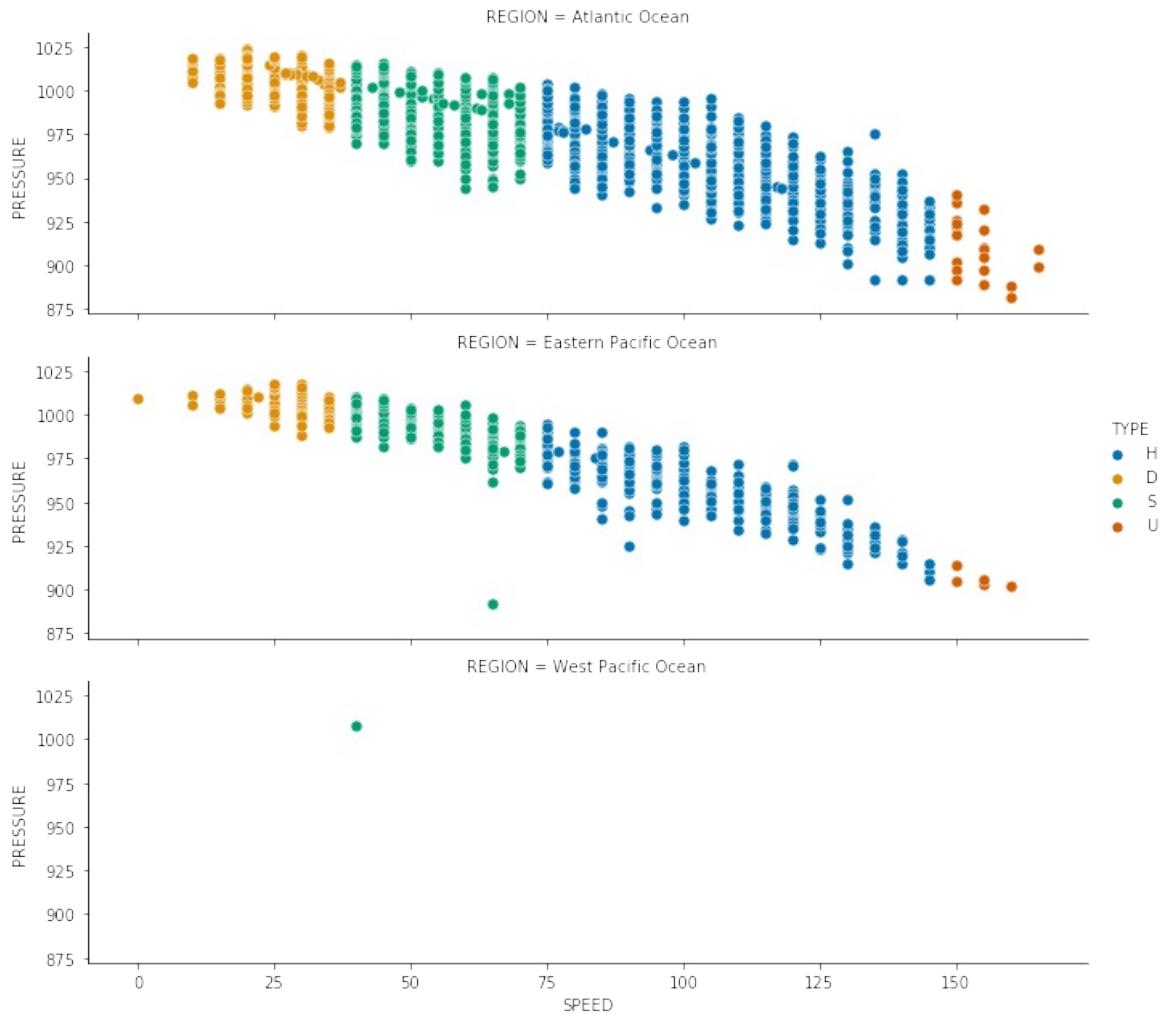


Figure 111: Correlation between speed, pressure and type of storm

When we plotted the faceted plot of wind speed, pressure, and category of storm type we observed that the type “D- Depression” storms are of higher pressure and lower speed and storm type “U” is of lower pressure and higher speed. We also observed that the storms of type “S - Tropical Storms” are more prevalent in the region “West Pacific Ocean” and are of lesser speeds when compared to other types of storms.

7. Heat Map of Eastern Pacific region

HEAT MAP for storm types for the Eastern Pacific Ocean region

```
: heat_map(locations, 'Eastern Pacific Ocean').save("images/heatmap3.html")
heat_map(locations, 'Eastern Pacific Ocean')
```

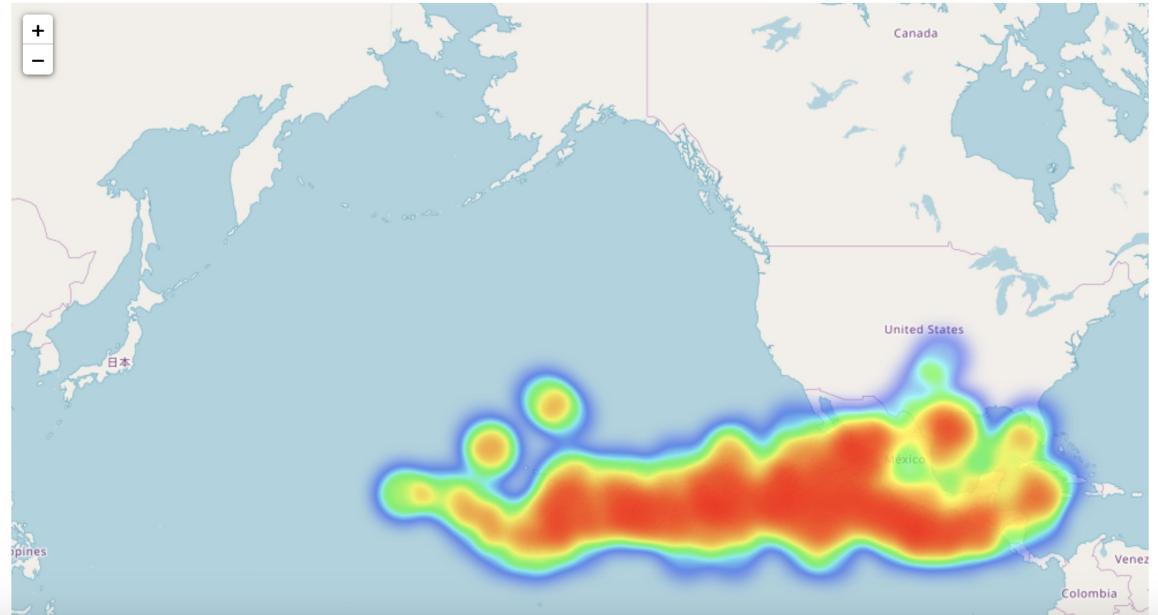


Figure 112: Heat Map of Eastern Pacific Region

When we plotted a heat map for Eastern Pacific region, we see that the major portion of storms are concentrated in the ocean itself and then as they make landfall the storms decrease in number.

We have created an interactive map showing the distribution of different storm categories and regions. The interactive HTML files are available in this [folder](#) and the code for these interactive maps is available in our Jupiter notebook.

11.6 MACHINE LEARNING

We used a few sophisticated supervised machine learning algorithms after data cleaning and feature engineering for storm type classification. When the output variable is labeled it is called supervised learning, in our scenario, the output variable is the type of storm. The below diagram best describes the process flow of our project.

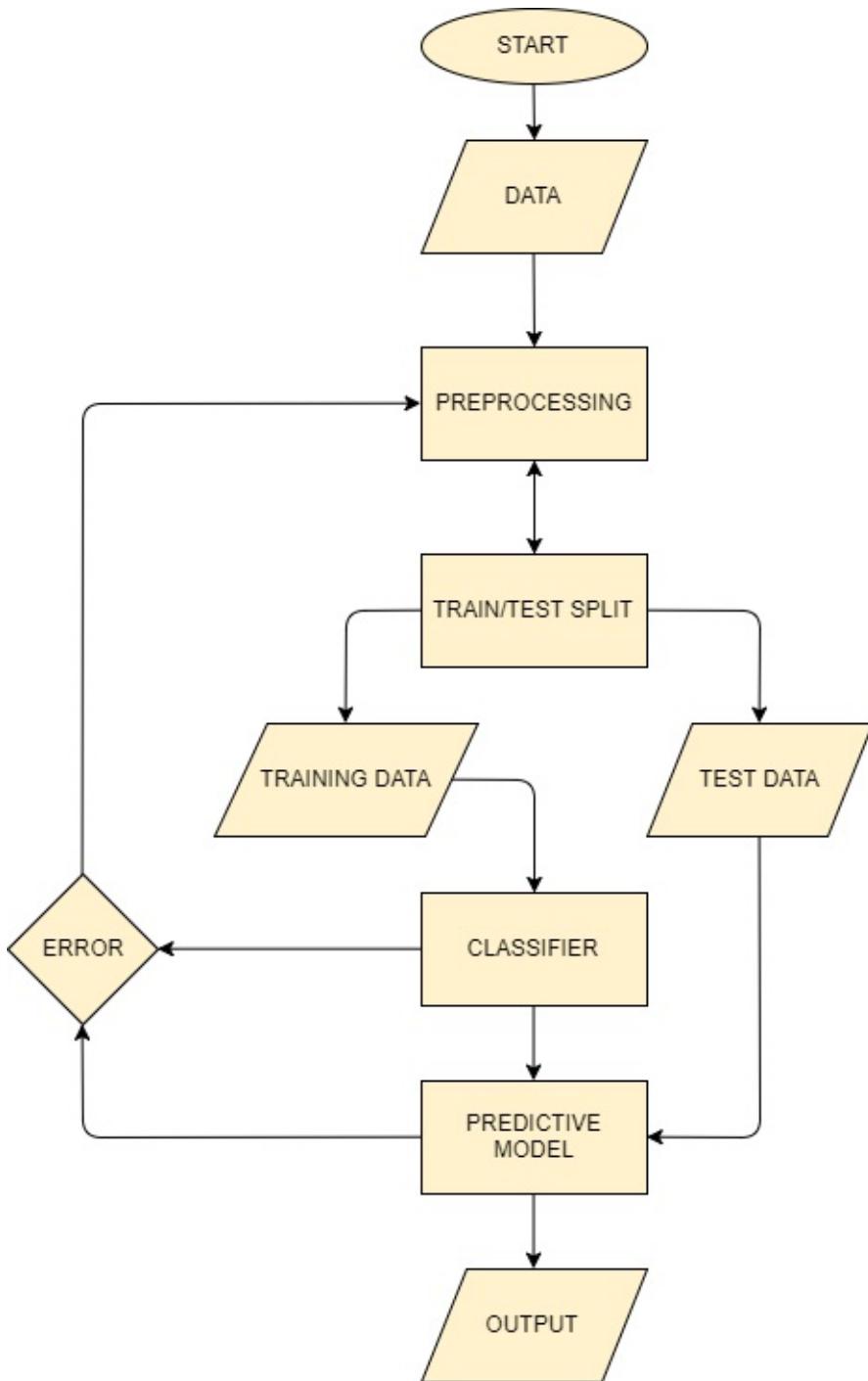


Figure 113: Flow Diagram

11.6.1 Feature Engineering

This step is common to both data visualization and machine learning. The process of choosing only the necessary features to build the machine learning model is known as ***Feature Engineering***. This is an integral part in solving any

machine learning problem because our data tends to consist of features which are not helpful in predicting the output.

11.6.2 Cross Validation

Cross-Validation is a technique that is commonly used in solving most machine learning problems today. The process of repeatedly splitting our data into train-test sets and training a model is known as Cross-Validation. This ensures our machine learning model performs as expected when new data is encountered in the training test. The parameter we need to specify is how many times should the process be repeated. In our project, we set this parameter value to be 5 and 10 to see the accuracy results.

11.6.3 Multinomial Naive Bayes

Multinomial Naive Bayes (MNB) is a simple classifier that uses the Bayes Theorem and assumes independence between the features and is used for data that follows a Multinomial distribution. We shall use scikit-learn's implementation in our project. The highest accuracy we got was when we used the 70/30 train-test split which gave us an accuracy of split gave us an accuracy of 13.4 % (rounded to the nearest decimal point). MNB tends to perform better when used with nominal data or data that consists of text and hence performs poorly on our data since it has no features which are nominal.

11.6.4 Logistic Regression

Logistic Regression [88] is a machine learning algorithm that is used for classification, it uses the natural logarithm function to find the relationship between the independent variable and the target variable, it uses test data to find the coefficients. The function can then predict the future results using these coefficients in the logistic equation. We have used various train/test splits to analyze the accuracy of our model. The highest accuracy we got was when we used the 70/30 train-test split which gave us an accuracy of split gave us an accuracy of 98.4 %.

11.6.5 Random Forest

Decision Trees are versatile Machine Learning algorithms that can perform both classification and regression tasks and even multi-output tasks. A Decision Tree is the fundamental concept behind a ***Random Forest*** [89]. In general, aggregation of answers tends to give a better result, the same technique used in machine learning is called ensemble learning. Instead of using a single classifier we use a group of classifiers or predictors, these group of classifiers usually tend to give a better prediction. When a group of decision trees is trained on a random subset of the training data we get better results. Such groups of decision trees or ensemble are known as a ***Random Forest***. This process is also known an ***ensemble learning***. The highest accuracy we got was when we used the 70/30 train-test split which gave us an accuracy of split gave us an accuracy of 98.9 %.

11.6.6 Support Vector Machines (SVM)

Support Vector Machines [90] are sophisticated predictive models in machine learning that are used for ***Supervised Learning***. SVMs are highly efficient in performing linear classification. An SVM uses a hyperplane or a set of hyperplanes to perform the task of classification. We have used two different implementations of SVMs in our project. We also implement SVMs with Stochastic Gradient Boosting in our project, this method is discussed in the next sub-section. The highest accuracy we got was when we used the 70/30 train-test split which gave us an accuracy of split gave us an accuracy of 98.4 %.

11.6.7 SVM with Stochastic Gradient Descent (SGD)

Stochastic Gradient Descent [91] (SGD) is a linear classifier under convex loss functions such as (linear) SVMs and Logistic Regression. The efficient application of SGD in large-scale and sparse machine learning problems in recent times has made it popular. For the parameters, we use ***elasticnet*** for the penalty as it might bring sparsity to the model. We also use 100 as the value for the parameter max_iter. The highest accuracy we got was when we used the 70/30 train-test split which gave us an accuracy of split gave us an accuracy of 98.6 %.

11.6.8 XGBoost

XGBoost [92] is another machine learning algorithm which uses ***ensemble***

learning like Random Forests. This process of uses a group of gradient boosted trees or ensemble to predict the label. XGBoost is known to be very fast, robust and optimized library. It provides the option of training these group of trees used in the ensemble in a parallel manner to achieve high accuracy, which is very effective in solving data science problems these days. The highest accuracy we got was when we used the 70/30 train-test split which gave us an accuracy of 98.9 %.

11.7 SUMMARY

From our visualizations, we identified that storms increased each year with different points of origination and that there were many storms in the Western Pacific ocean region than in other regions. We identified that the pressure and speed have an effect on storm type and they separate storms into Hurricanes, Tropical Storms, Depressions, and low-pressure storms. We identified that the higher speed storms are more likely to occur in the West Pacific Ocean region. We hope to study the rise in the storms across different basins on a much deeper level to understand what categories of storms increase in frequency over time.

After applying machine learning algorithms we saw that all the algorithms achieved an accuracy of over 98 % using Cross-Validation with five and ten folds. Before we make any strong conclusions we would like to gather more data and hope our trained model performs well on this new and unseen data examples.

11.8 FUTURE WORK

In this project, our main aim was to only visualize the dataset and analyze these visualizations. We would like to identify and uncover more interesting patterns in our data. We hope to use unsupervised machine learning algorithms to find clusters in our data. This might help us identify further patterns in the data that could not have been possible with just plain visualization of the data. The main problem that we have identified in building such a model is the availability of resources required to run the algorithm without any interruption. Using local computer resources for running such a model would not be feasible. Hence, we hope to deploy such a model in the cloud and utilize its power and resources for

running such algorithms.

We would also like to build a model that is able to do forecasting. The data we have collected is from 1952-2008. We would like to explore the possibility of building a model that predicts areas that are more prone to such devastating hurricanes and storms. Ideally, we would like to split the above-mentioned 56-year data into two parts, train a model on one half and see if it is able to make accurate predictions about the storm and hurricanes in the second half.

11.9 ACKNOWLEDGEMENT

The authors would like to thank Dr. Gregor Von Laszewski for giving the opportunity to work on this project and for providing valuable feedback during the duration of our project. The authors would also like to thank Dr. Geoffrey Fox for providing feedback on the project report. The authors would also like to thank the Associate Instructors of this class for their help and prompt responses to our questions on Piazza. The authors would also like to thank Microsoft for providing a free trial of Azure Cloud services.

11.10 WORK BREAKDOWN

The authors Divya Rajendran and Pramod Duvvuri have contributed equally in preparing this report. Divya Rajendran has worked on Data Cleaning, Data Visualization, and Data Analysis. Pramod Duvvuri has worked on setting up the Cosmos DB instance and Machine Learning. Our contributions towards the completion of the project have been updated in our respective notebook markdown files in our respective git repositories.

12 KICKSTARTER PROJECTS ANALYSIS

Izolda Fetko, Nishad Tupe, Vishal Bhoyar
ifetko@iu.edu, ntupe@iu.edu, vbhoyar@iu.edu
Indiana University
hid: fa18-523-60, fa18-523-64, fa18-523-72
github: [blue icon](#)
code: [blue icon](#)

Keywords: MongoDB, PyMongo, Crowdfunding, VM's, IaaS & DBaaS Cloud, Data Analysis, Predictions

12.1 ABSTRACT

Crowdfunding is a certain way of raising funds where individuals come together and collectively support projects by investing in them. Kickstarter is one of the leading crowdfunding platforms in the world that helps individuals/businesses in various categories such as art, film, music, theatre, games, design, and other, to raise necessary funds to complete their projects. This paper explores the utilization of big data technologies and cloud services such as MongoDB, virtual machines, MongoDB Atlas, DigitalOcean, AWS, and PyMongo while analyzing the **Kickstarter Projects** dataset obtained from the Kaggle datascience platform.

12.2 INTRODUCTION

Many creative individuals and groups around the globe are trying to develop products, projects, and businesses based on their unique ideas and talents. In many cases, an idea or a talent is not enough to accomplish this. Capital funds are one of the main elements of a successful start-up process. Although there is a significant number of government programs in the United States designed to help small businesses and creative individuals to obtain necessary funds for their projects, a high number of start-ups nonetheless remain capital constrained [93]. This is the reason why the **Kickstarter** platform was developed and is still

successfully operating since 2009 [93].

Kickstarter is a web-based crowdfunding platform that helps support creative arts around the globe [93]. It uses the fundable project model where the would-be business owners or project managers submit their project along with the necessary project information and its funding goal.

“The funding goal establishes a base target for the project and a deadline (generally 30-days) to achieve the funding” [93].

The prospective artists and entrepreneurs typically display their projects in a video form where they have the opportunity to outline the project along with its benefits and funding requirements [93]. The video is shown on the **Kickstarter** project webpage along with the number of backers, the amount of funds received/pledged as well as the fundraising goal. Since each project is timed, the time remaining in the **Kickstarter** promotion is shown on the project page as well. The platform is based on an ***all or nothing*** funding model which means that the pledged funds are not available to the entrepreneurs until the end of the funding period, which is determined by the goal-setting statement [93]. The financial transactions occur only in cases where projects meet their initial targets and funding expectations [93]. If the initial target is exceeded, the project receives all funding as well as the excess of its goal.

Our project includes several big data technologies while exploring the **Kickstarter** dataset collected in the 2009-2018 timeframe and made public via Kaggle, a well known datascience platform. The base technology used to store and query the dataset is the widely known NoSQL database called MongoDB. Three cloud services chosen to be benchmarked against each other are MongoAtlas, DigitalOcean, and AWS. Our team had developed a Python program and a packaged shell script that builds the MongoDB environment on a Unix platform, performs data analysis through visualizations to gain hidden insights

and builds a logistic regression model to predict the success/failure of the **Kickstarter** projects. MongoDB database is hosted on several cloud environments. The time needed to query the dataset was used as a quantitative measure to analyze the relative benchmarking. Moreover, our team provides an ease-of-use review of each service as an addition to the performance benchmarking. This report introduces the reader to the related work in the

MongoDB realm, as well as to the chosen dataset prior to presenting the project design and research methods, architecture, technologies, and results.

12.3 LITERATURE REVIEW

MongoDB and Python are both open source technologies. One can get started quickly by building an application on MongoDB using any of the languages that leverage MongoDB's driver. This database offers a native driver called PyMongo to fit Python developer community needs.

“MongoDB stores data in documents, however, they are not like Microsoft Word or Adobe PDF documents but rather JSON documents based on the JSON specification” [94].

There are several advantages of storing data in document format and some of them are flexible schema and ability to store arrays, which are faster to process using native commands of Python scripts [94]. In June 2018, students of the Indiana University, Bloomington Izolda Fetko, Rashmi Ray, and Nishad Tupe explored the France accidents dataset using MongoDB, PyMongo, and Tableau to provide safety recommendations. They used the newest release of the MongoDB driver called the **Mongo BI connector** that allowed BI tools such as Tableau to interact with MongoDB. The BI connector converts the Tableau's SQL-like commands on structured data into the native MongoDB commands while fetching data [95]. The team also used PyMongo and other web technologies to build the website that could store, update, process the records live and can be accessed by the global audience. As MongoDB is a NoSQL engine, it scales easily for multiple tables as a single JSON object, and makes query retrieval speed faster than RDBMS, while avoiding complex joins. However, the research also showed that Tableau and MongoDB is a lousy marriage predominantly because Tableau was built before the NoSQL and Big Data were popular and is not yet mature to process relational data [96]. Tableau's slow processing of joins, when connected directly to MongoDB, was one of the main reason authors decided to join the CSV files using Pandas data frame and use MongoDB as a backend tool. This tool allows a stable platform for the user-friendly BI tools such as Tableau to perform analytics on large datasets with millions of rows and also to stand as a robust database on which one can build applications [95]. In the final step, authors created a website and

provided links to various dashboards using technologies such as HTML, CSS, JavaScript, Bootstrap, Flask, JQuery, and Chartist [95]. Although the goals of the aforementioned and our current project differ, they do share a lot of similarities such as using the exact same big data technologies MongoDB and Python.

Another paper written by [97], presents an interesting benchmarking of the MongoDB database on several cloud instances. In his report, [97] notes that each virtual instance created for this purpose had MongoDB and PyMongo installed on it. The similarity between this report and our project is that both exploit the benefits of the PyMongo driver and the Amazon Web Services, more specifically the **EC2** instances. In addition to the AWS, [97] used Linode, a hosting company that offers a virtual private server (VPS); Rackspace cloud and its open source technology **OpenStack**; and Windows Azure and its virtual machines [97]. In the final section of his paper, [97] concludes that MongoDB's performance varies from cloud to cloud due to various factors. One of the most important factors that he lists are the fast I/O access and ability of the database to cache all indexes in RAM [97].

The report by [98] shows similarities with our report in the sense that is conducting benchmarking of MongoDB using the Amazon Web Services. However, in his report, [98] takes a step further and compares the MongoDB performance to other databases such as CouchDB and Apache Spark. He concludes that MongoDB performs well on the cloud with ultra-low latency which makes it a great choice for applications with flexible schema requirements [98].

The article written by [99], presents NoSQL database benchmarking with the use of the Yahoo Cloud Benchmark (YCSB) and Amazon Web Services with an installed Linux operating system [99]. The purpose of his article is to help developers choose the right database for their application. He tested Cassandra 2.0, MongoDB 2.4.6, HBASE 0.92 and concluded that developers need to evaluate different solutions in their search, and test their performance first prior to making any decisions [99]. According to him, all databases are good in some way, but may perform differently in different scenarios, hence, it is important to choose them based on the most needed properties and project requirements [99]. The similarity between this project and our project is the use of the Amazon Web Services as a benchmarking tool for MongoDB.

12.4 DATASET DESCRIPTION

12.4.1 Kaggle API

The technology used to easily obtain the ***Kickstarter Projects*** dataset is the newly offered Kaggle Public API. API stands for ***Application Programming Interface*** through

“which interactions happen between an enterprise and applications that use its assets” [100].

Kaggle Public API was launched in February 2018 and can be used for ***creating datasets, kernels***, or simply ***connect with Kaggle*** [101]. Although still in its beta phase, it allows a more user-friendly data download as well as a seamless workflow for its community members. To be able to use this technology, users need to ensure that they had installed the latest Python 3 version on their machines as well as the pip package manager [102]. Accessing the Kaggle API is done by using a simple command line; however, this is not possible until a Kaggle account is created [102]. Once the initial step had been completed, an API token can be created, which triggers a download of a JSON file that contains the user credentials necessary to access the API [102]. Once the sign-up had been finalized, various command lines can be used to access the list of competitions along with the files and submissions associated with them [102]. A different set of commands can be used for dataset downloads and dataset creation, while the final group of commands listed on the Kaggle API GitHub page are the commands to manage Kernels, more specifically Kernel pull and push [102].

The importance of the Kaggle’s public API is significant. It minimizes the need for its users to manually download large datasets, hence saving them time when working on important projects. It is also helping students in expanding their knowledge and programming experience through practical examples and real-life data that can be later implemented in their professional work.

12.4.2 Kickstarter Projects Dataset

The **Kickstarter Projects** dataset is publicly available on the Kaggle website and can be accessed using this [path](#). The instructions on how to obtain the dataset using the Kaggle API can be found [here](#). The dataset is available in the CSV format and contains more than 370 thousand projects submitted to Kickstarter between 2009 and 2018 [103]. The dataset variables allow versatile data analysis which is presented in the ***Observations and Visualizations*** segment of our project. Other than the project ID, the dataset contains information on the project name; main category and category of campaign; currency used to support the project; fundraising goal (the amount of funds needed to complete the project); project launch (date); project crowdfunding deadline (date); state – current state of the project; actual funds pledged to the project along with the number of backers; the country of origin; and the total amount of funds pledged by currency [103].

12.5 DESIGN AND METHODS

One of the primary aims of the project was to utilize the minimal cost cloud resources. Thankfully, nowadays, every cloud provider provides a free tier service. To leverage this, our team created VM's on Amazon Web Services (AWS), DigitalOcean cloud service with more or less the same basic configuration of memory and hard disk. The AWS and DigitalOcean are primarily an **IaaS** service, while MongoDB Atlas is a **DBaaS**. This gave us the opportunity to test the solution on various cloud providers and benchmark their performance with pros and cons. Although the cloud VM's gave us the lot more control and customization over the OS and database level resources, MongoDB Atlas cloud services provided a stable database clustered environment for high availability of data. There was no overhead of configuring MongoDB instance as it is a **DBaaS** service. We decided to leverage Unix bash script to perform a task automatically. The following bullet points outline the steps taken during our project:

- Dataset Download
- Install MongoDB
- Import MongoDB
- Conduct Python Analysis

Once the stable infrastructure foundation on the cloud based VM's was

achieved, we moved onto loading the data using two methods:

- Python script based method for DBaaS
- MongoImport for cloud VM's

Before beginning the analysis, it is crucial to extract the essential features. We used Python datetime library to get the date, year, month; to clean up the *Nan* rows carefully and also to calculate the duration of a project. The next step encompassed in data enhancement by extracting the features that can give a foundation to perform various analysis and develop a machine learning model. During the aforementioned process, our team had utilized the following methods to complete the data analysis and draw insights from the dataset.

- Cloud and MongoDB set up

The first step was to create a stable infrastructure to perform analysis. This method involved creating the cloud VM's, and use the bash shell scripts to complete the installations. We had also done some testing of the MongoDB connectivity from various machines to the cloud instances. Finally, we had created and prepared the architecture diagram.

- Exploratory Analysis and Visualization

Data Visualizations that let one discover trends or patterns in a dataset are called the *Exploratory Data* analysis. Once the data is in a good shape, it is easier to gain insights by using visualizations that often become handy tools for finding interesting patterns.

- Correlation or Heatmap analysis

The correlation analysis is a statistical method used to evaluate a relationship between two continuous variables [104]. For example, we had used this type of analysis to find relationships between project categories and states.

- Time-Series Analysis

The time series analysis is a statistical technique which is related to data

that is distributed in a series of particular time periods or intervals [105]. We had used it to get a better understanding of the distribution of the projects over years and months.

- Logistic Regression Model

The logistic regression analysis is a predictive analysis used to describe a relationship between one dependent and one or more nominal, ordinal, interval, or ratio-level independent variables [106]. We had used this model in our machine learning algorithm to predict successful vs. failed project status.

- Perform MongoDB queries

The gist of our project is to show the MongoDB ability to query in real time, hence we used the MongoDB aggregation framework to analyze the **Kickstarter** dataset as well.

12.6 TECHNOLOGIES

12.6.1 Technologies and Tools Used

- Python version 3.6 and various libraries such as Seaborn, Matplotlib, Pandas, and Scikit-learn
- PyMongo Driver, Bash Shell
- Cloud service - MongoDB Atlas, 3 node replica cluster
- Cloud service - DigitalOcean, Ubuntu 18.04 ,MongoDB 3.6.3
- Cloud service - AWS , Amazon Elastic Compute Cloud (EC2), Ubuntu 18.04, MongoDB 3.6.3

12.7 CODE ORGANIZATION

Our code is checked-in on GitHub and can be accessed by using this [link](#). It is organized as described in the following section.

12.7.1 bin

- [MongoDB_queries.txt](#)
- [bash_envsetup_script.sh](#)
- [csv_to_mongo.py](#)
- [kickstarter_analysis.ipynb](#)
- [main.py](#)
- [mongo_uninstall.sh](#)

12.8 ARCHITECTURE

Since the beginning of our project, the team aimed to create a scalable Python code that can run seamlessly on different cloud environments. We mainly used cloud computing services from Amazon Web Services (**AWS**), **DigitalOcean** as **IaaS** and **MongoDB Atlas** as a **DBaaS** platform. Though **AWS** is giant a cloud provider with multiple cloud services, we found that **DigitalOcean's** user-friendly virtual machine (VM's) management interface is equally attractive. The architecture diagram (Figure 114), more specifically the upper left dotted box, shows the client machine or application tier where the source code(**.py**) was stored and used for performance benchmarking.

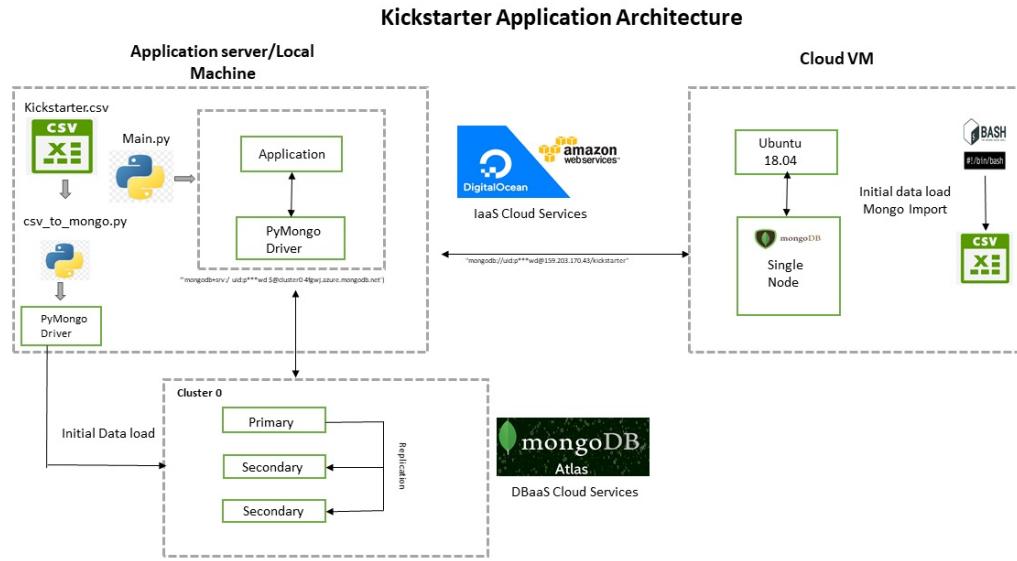


Figure 114: Architecture Diagram

The team also kept a copy of the source code and other scripts on the cloud VM's file systems. We used PyMongo driver as one of the primary components

for communicating with the MongoDB database. PyMongo does not only provide the MongoDB driver access to Python libraries, but is also a recommended choice when wrangling data with MongoDB [107].

During the initial loading phase of the **Kickstarter** dataset, our team imported the raw data using the MongoDB import command line in the **DigitalOcean** and **AWS** cloud VM's. A custom bash shell script that installs, configures the MongoDB environment and then imports the CSV data to MongoDB was also written. To load the data in the MongoDB Atlas cluster, we used a simple Python function **load_csv.py**. The primary reason for this is because MongoDB Atlas is a **DBaaS** cloud service and typically in **DBaaS** users do not have control or access to OS resources such as a file system. In most basic form, our M0 cluster consists of one primary node and two secondary nodes. The primary nodes are mainly responsible for writing operations, while the secondary nodes replicate primary's **oplog**. This way the secondary node's dataset reflects the primary's dataset in cases where the primary node is unavailable [108]. Only the eligible secondary nodes will

“hold an election to elect itself the new primary” [108].

This replica set arrangement ensures high availability of the data. Optionally, one can configure the arbiter node which does not hold any data but keeps the track quorum in the replica set. Since arbiter nodes do not hold any data, they act as a suitable repository to keep the heartbeat information at a cheaper cost [108]. On the contrary, our team had found that the **IaaS** services provide greater control and customization to the OS resources but add overhead to perform the configuration and other tasks which can be complex and may induce a lag time for writing the code due to incomplete pre-requisites. On the cloud VM's, the team hosted a single instance MongoDB database on Ubuntu 18.04 platform. As post install steps, our team had installed the Python Anaconda Distribution and other necessary libraries essential to complete the analysis. The communication between the MongoDB and Python application happens by connecting string. The connect string, one must have a valid username and password and necessary privileges to access and modify the database. To accept the remote connection, one of the vital steps is to set a value of **bind_ip** to 0.0.0.0 in the **mongodb.conf** file that resides on the VM. For all cloud providers, our team had used free-tier services. We observed notable advantage of using MongoDB Atlas free tier service often called as **M0 cluster**. By default, the **M0 cluster** comes with three

node replica sets and 512 MB storage. The replica set is a group of ***mongod*** processes which provide redundancy and high availability to the application while accessing the MongoDB data.

12.9 OBSERVATIONS AND VISUALIZATIONS

Our team completed an exploratory analysis of the ***Kickstarter*** dataset. The results of the analysis can be observed in the following sections.

12.9.1 Exploratory Analysis

Our queries have determined that the ***Kickstarter Projects*** data includes six different project states such as failed, successful, canceled, live, suspended and undefined, as shown in Figure 115. Due to the lack of funding, more than 300000 projects in the overall pool of projects had failed, while 200000 projects achieved the funding goal with a successful status. Approximately 70000 projects were canceled while others with less significant counts were marked as live, suspended and undefined.

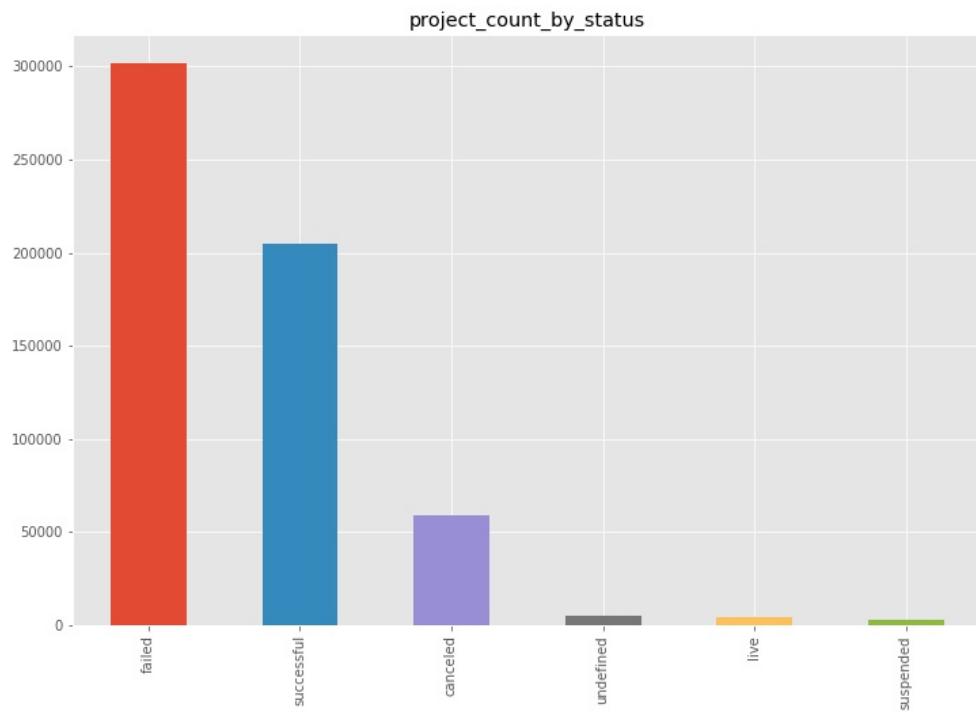


Figure 115: Project States Count By Year

A more detailed view of the project counts was created for the main two state categories - successful and failed projects. The data visualization in Figure 116 shows that the highest number of projects were submitted in 2015. Nearly 70000 projects failed to achieve their funding goal in the targeted time-frame. This was the highest count of failed projects in the history of **Kickstarter**. The number of successful projects has steadily increased since 2009 and shows a normal distribution over the years with lot less variance compared to failed projects. Overall, it can be concluded that the number of failed projects was much higher than the number of success projects in each year.

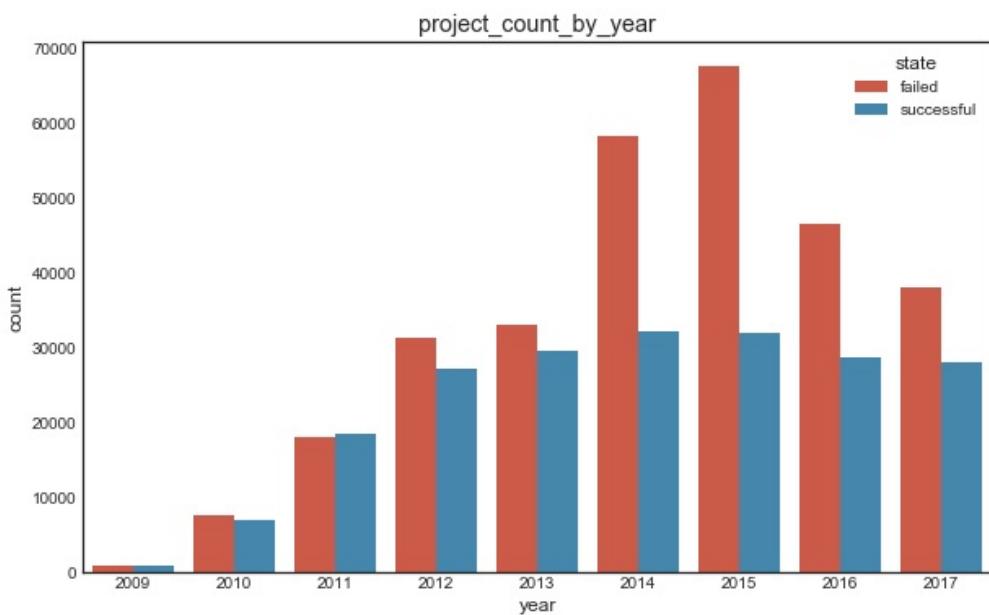


Figure 116: Project Counts By Year(Succces & Failed)

The Figure 117 visualization is showing the best markets (countries) for funding projects. Based on this analysis, Austria appears to be the best country for project funding. More than 800000 projects with pledged US dollars launched in Austria achieved their goal within deadline. It is followed by China and New Zealand, which have less successful projects but almost equal amount of failed projects as Austria.

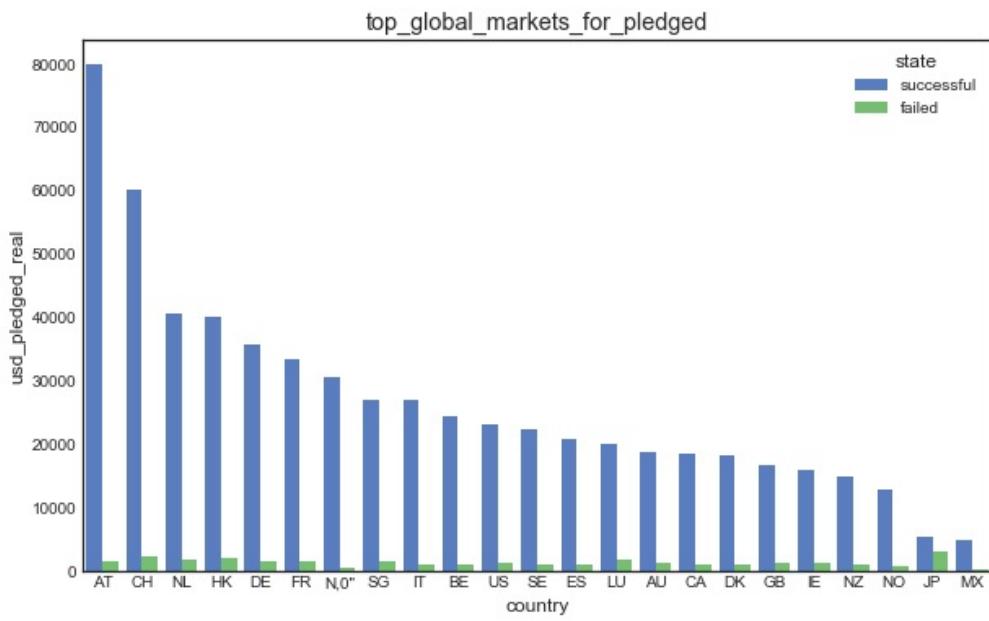


Figure 117: Top Markets For Pledged Funds

The heatmap visualization in Figure 118 is showing project state count against the main category. The scale represents highest count with yellow color and lowest count with dark blue color. The chart is showing highest successful project counts for the **Technology** main category followed by **Dance** and **Games** categories. The count of successful projects in the **Technology** main category was more than 80000.

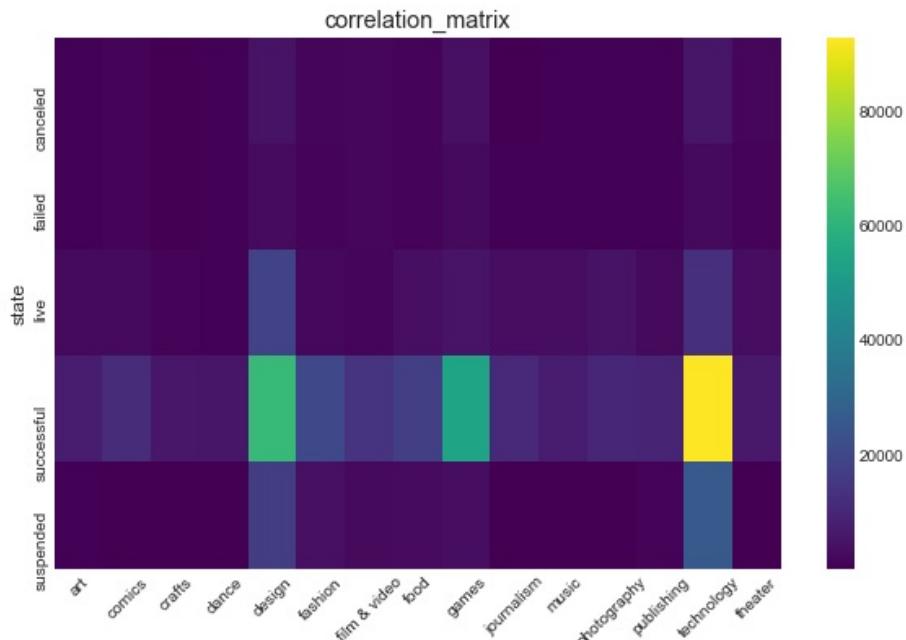


Figure 118: Heatmap Amount Pledged By Category & State

The projects recorded as ***failed*** are the ones that were not able to achieve funding goals between the project launched date and the project deadline date. There are some projects which started collecting funding however, were not able to collect the targeted amount. The visualization in Figure 119 is showing the targeted funding goal data and actual funding collected for main categories. The ***Film & Video*** category has the highest funding goal and only 10% fund achieved by projects. The funding collection of the ***Technology*** category was the highest compared to the total fund goal.

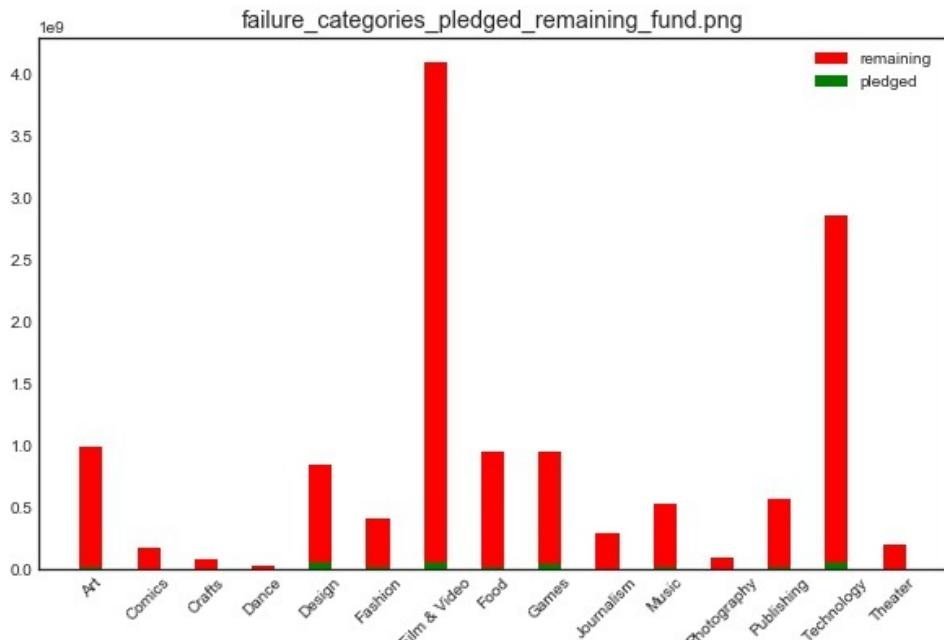


Figure 119: Pledged & Remaining Funds Failed Categories

The project fund duration is a measure that represents the day counts between the project launch date and the project deadline date. The duration box plot in Figure 120 shows the average duration for successful and failed projects. The average duration of the successful projects was less than 31 days and the average duration of the failed projects was more than 40 days. The median between successful and failed projects the duration at approximately 36 days.

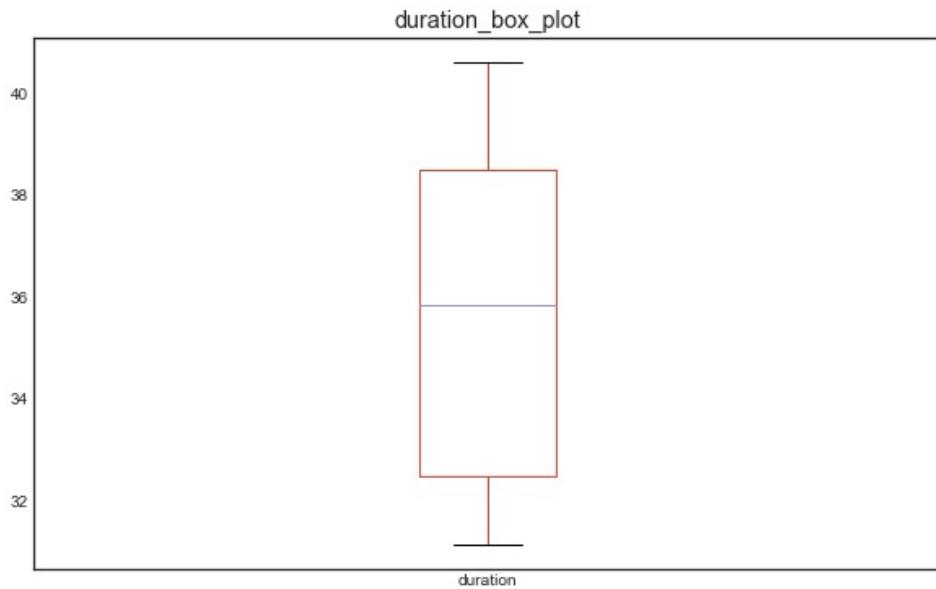


Figure 120: Project Duration Box Plot

12.9.2 Time Series Analysis

The time-series analysis conducted by the team revealed certain trends in the **Kickstarter** dataset which will be presented in the following segment. One of the trends noticed was that the projects mostly get launched in the warmer months with July being the month with the highest number of projects. The trend of the launched projects by month can be seen in Figure 121.

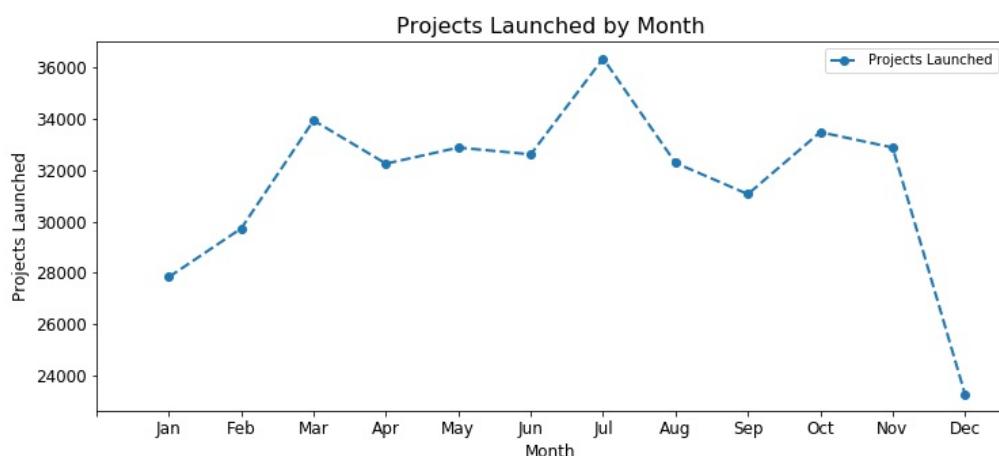


Figure 121: Projects by Month

Drilling further into the project by month data, as shown in Figure 122, one can notice that proportionally to the number of launched projects, July is the month with the highest number of cancelled and failed projects. The number of live projects is higher in winter months, mostly in November and December, however, the overall number of projects drastically decreases most likely due to the holidays.

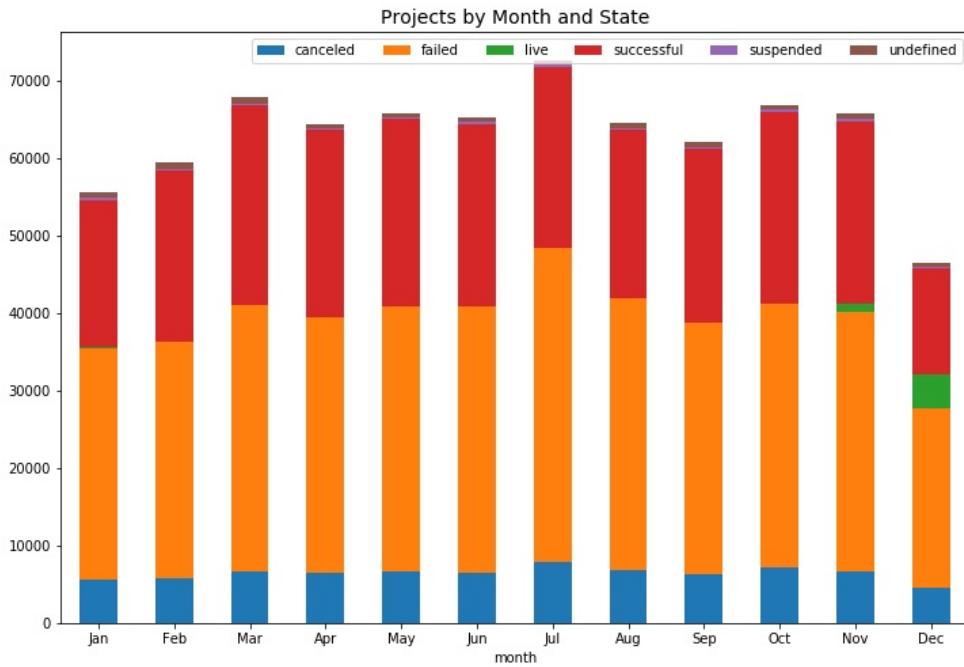


Figure 122: Projects by Month and State

Looking at the same categories over the years time series, from Figure 123 one can conclude that overall, 2015 was a great year for **Kickstarter** with the highest number of submitted projects, but also, proportionally, one of the highest number of failed projects.

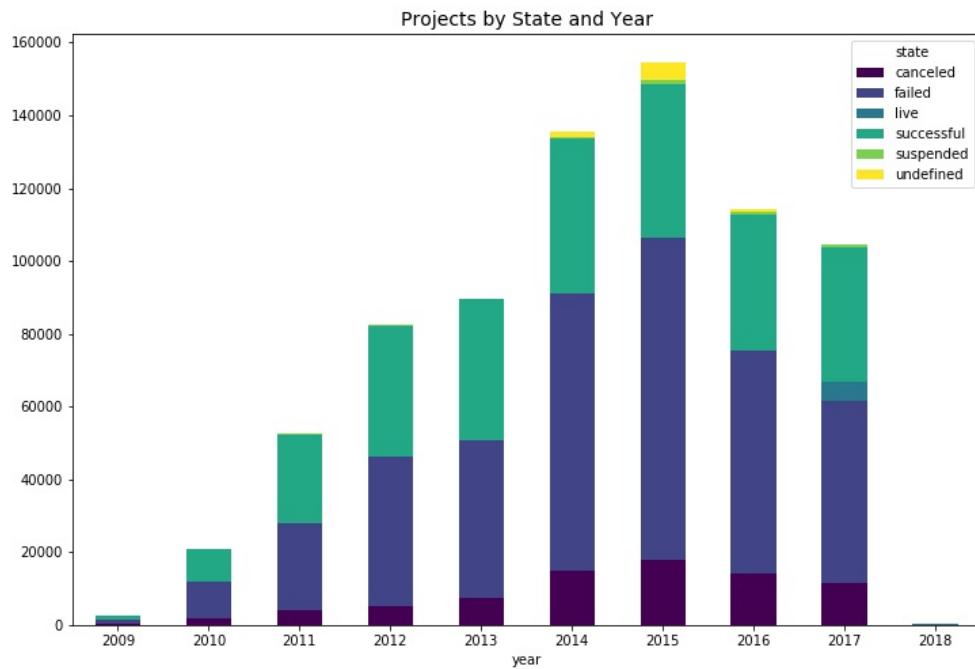


Figure 123: Projects by Year and State

When it comes to the countries from which the projects were submitted in the 2009-2017 timeframe, one can conclude that **Kickstarter** was popular and available only in the United States in the first four years of its activity. In 2013, for the first time, one can notice other countries making project contributions such as Canada and New Zealand. As one can notice in Figure 124, in the following years, the number of contributing countries rises, however, the leading one still remains the US.

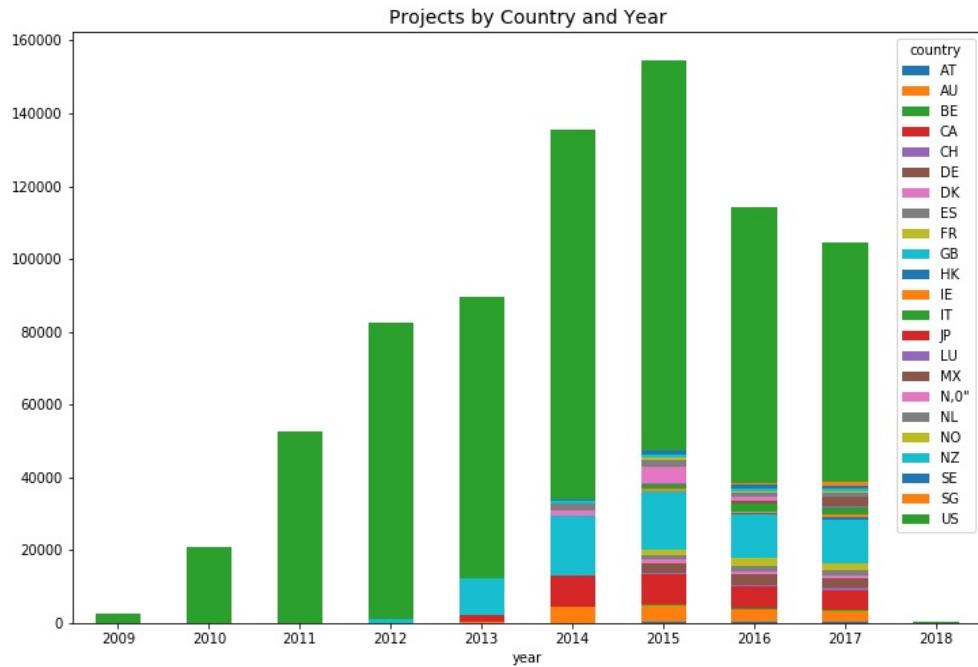


Figure 124: Projects by Year and Country

12.9.3 Logistic Regression

In real life classification problems are more prevalent than regression. Logistic regression helps us solve classification problems by employing the sigmoid function. Logistic regression tells us about the probability as a cut off point, it uses logistic function to build the model that can predict binary dependent variable[109].

The probability of belonging to a class is less than 50%. The values are assigned to class 0 and in our case, the classes were predefined as *success* or *failed*. Thanks to the scikit-learn logistic regression model which takes care of the substantial mathematical part, we moved forward with building a model that can predict future projects' states.

To build this model, we used the following steps:

- Cleaning the dataset
- Extract projects with success and failed states only

- Explore dataset to predefine the classes
- Identify top 5 categories
- Extract features and label data
- Build the model
- Predict and measure accuracy

The following figure (Figure 125) shows categories with the highest number of succeeded and failed projects.

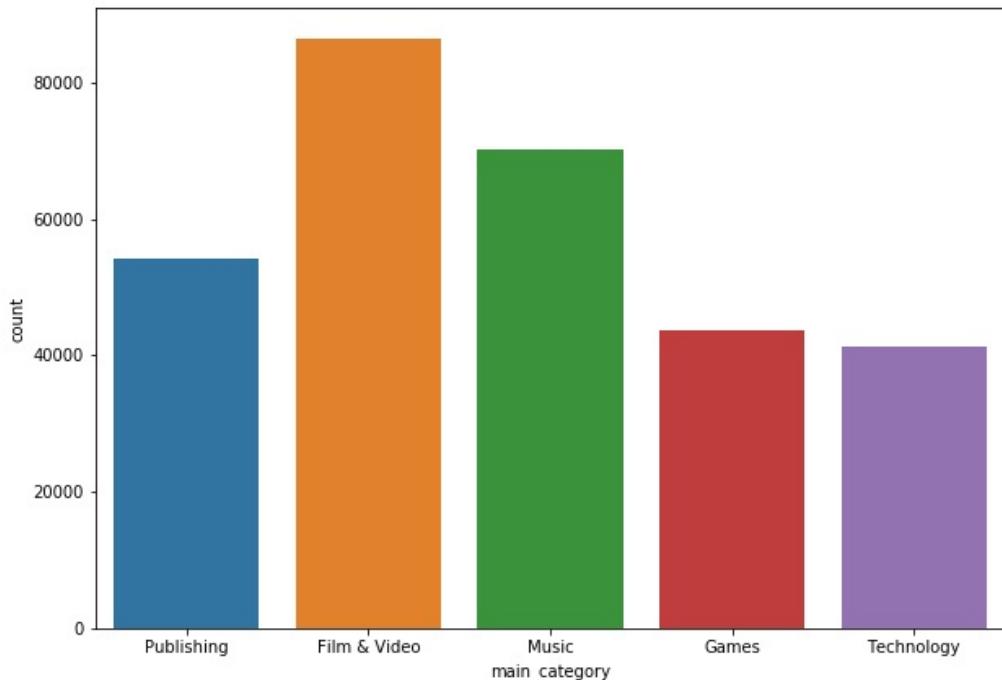


Figure 125: Top Five Categories

We decided to use features such as '***main_category***', '***goal***', '***backers***', '***duration***', '***successful***' out of which ***successful*** feature was generated using pandas get_dummies method.

```
proj_state = pd.get_dummies(data=df_sf_t['state'], drop_first=True)
```

The following figure (Figure 126) shows the categorization of our input dataframe into two classes - successful and failed.

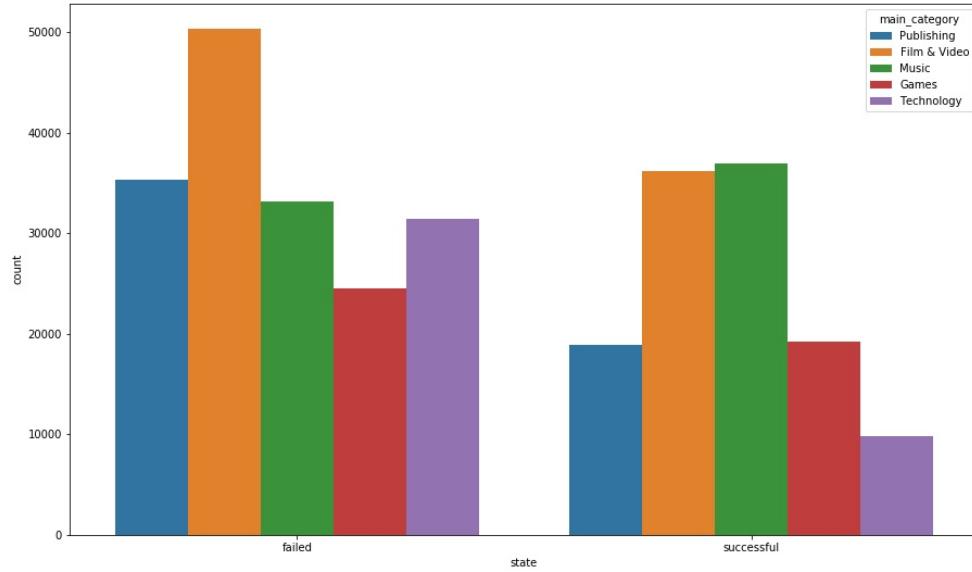


Figure 126: Category Classes

All the main categories were mapped to numerical values so as to input them as numeric vectors to the machine learning algorithm. This can be seen in the following table.

```
cats = {'Publishing':4, 'Film & Video':1, 'Music':2, 'Games':5, 'Technology':3}
```

- Labeled Data Sample

main_category	goal	backers	duration	successful
4	1000.0	0	58	0
1	30000.0	15	59	0

Once the data frame with labeled data was ready we followed the standardized machine learning steps to build the model which involved creating the classifier object; splitting the train test dataset; fitting the model; predicting values; and measure model accuracy. The following table shows the confusion matrix of our test dataset.

- Confustion Matrix (y-test)

n=88758	Predicted No	Predicted Yes
Actual No	TN = 50106	FP = 2331
Actual Yes	FN = 7043	TP = 29278

We used the five fold cross-validation method to measure the accuracy of the algorithm. We had found the average accuracy score was close to 90%. To get more accuracy, this model can be extended to additional features and also a larger set of classes. Our accuracy scores can be seen in the following table:

- Accuracy Scores of train data

Cross Val=5	Accuracy %
CV-1	89.84
CV-2	89.66
CV-3	89.48
CV-4	89.54
CV-5	89.37
Average	89.58

- Accuracy of actual Model

algorithm	Accuracy %
Logistic Regression	89.43

12.9.4 MongoDB Queries

Our team had also had the opportunity to write MongoDB queries directly working in Mongo Atlas with a purpose of comparing the ease-of-use related to querying of the **Kickstarter** data between MongoDB and Python. One of the queries included the count of projects by category.

```
> db.project.aggregate([
    {$group:{_id:{category:"$main_category"}, count:{$sum:1}}},
    {$sort:{count:1}}
```

```
])
```

The output of this query can be seen in the following section.

```
{ "_id" : { "category" : "Dance" }, "count" : 3768}
{ "_id" : { "category" : "Journalism" }, "count" : 4755}
{ "_id" : { "category" : "Crafts" }, "count" : 8809}
{ "_id" : { "category" : "Photography" }, "count" : 10779}
{ "_id" : { "category" : "Comics" }, "count" : 10819}
{ "_id" : { "category" : "Theater" }, "count" : 10913}
{ "_id" : { "category" : "Fashon" }, "count" : 22816}
{ "_id" : { "category" : "Food" }, "count" : 24602}
{ "_id" : { "category" : "Art" }, "count" : 28153}
{ "_id" : { "category" : "Design" }, "count" : 30070}
{ "_id" : { "category" : "Technology" }, "count" : 32569}
{ "_id" : { "category" : "Games" }, "count" : 35231}
{ "_id" : { "category" : "Publishing" }, "count" : 39874}
{ "_id" : { "category" : "Music" }, "count" : 51918}
{ "_id" : { "category" : "Film & Video" }, "count" : 63585}
```

From the results we can see that the highest number of the overall submitted projects was in the **Film and Video** category. This makes sense when we take into consideration the number of smart phones and camera devices in the world. These technologies have made this art more available to amateurs. In a similar manner, the team had written a query to count the number of projects by year.

```
> db.project.aggregate([
    {$group:{_id:{year:"$year"}, total_number_of_projects:
        {$sum:1}}},
    {$sort:{_id:1}}
])
```

The results in this query have shown that the number of projects varied throughout the years, and that the highest number of projects was submitted in 2015. Finally, a query that resembles the previous two is the total count of projects by project state, which revealed that the vast majority of projects that get submitted also get funded.

```
> db.project.aggregate([
    {$group:{_id:{state:"$state"}, count:{$sum:1}}},
    {$sort:{count: -1}}
])
```

A slightly more complex query was used to compute the total amount of the pledged funds, number of backers (investors), and total funding goal by project category. Our team had determined that with more complex queries, the output appeared less attractive and more difficult to read as it can be seen in the output after the query code.

```
> db.project.aggregate([
    {$group:{_id:{category:"$main_category"}, tot_amt_pledged:
        {$sum:"$pledged"}, tot_backers:{$sum:"$backers"}, tot_goal:
        {$sum:"$goal"}}}
])

{
  "_id": {
    "category": "Theater",
    "tot_amt_pledged": 44713012.92,
    "tot_backers": 513536,
    "tot_goal": 300569615.61
  },
  "_id": {
    "category": "Dance",
    "tot_amt_pledged": 13906929.44,
    "tot_backers": 16127,
    "tot_goal": 38890776.3
  },
  "_id": {
    "category": "Music",
    "tot_amt_pledged": 207294846.96,
    "tot_backers": 2708475,
    "tot_goal": 833613962.47
  },
  "_id": {
    "category": "Film & Video",
    "tot_amt_pledged": 404574432.02,
    "tot_backers": 4197577,
    "tot_goal": 562378004.68
  },
  "_id": {
    "category": "Photography",
    "tot_amt_pledged": 39501225.45,
    "tot_backers": 428078,
    "tot_goal": 140161865.51
  },
  "_id": {
    "category": "Publishing",
    "tot_amt_pledged": 145090176.71,
    "tot_backers": 2231589,
    "tot_goal": 1161588016.96
  },
  "_id": {
    "category": "Crafts",
    "tot_amt_pledged": 17760300.12,
    "tot_backers": 240342,
    "tot_goal": 102116446.5
  },
  "_id": {
    "category": "Games",
    "tot_amt_pledged": 770331916.1,
    "tot_backers": 11336829,
    "tot_goal": 1786609751.2
  },
  "_id": {
    "category": "Comics",
    "tot_amt_pledged": 74643647.75,
    "tot_backers": 1458090,
    "tot_goal": 219016009.29
  },
  "_id": {
    "category": "Art",
    "tot_amt_pledged": 101547027.64,
    "tot_backers": 1188200,
    "tot_goal": 1149463908.6
  },
  "_id": {
    "category": "Fashion",
    "tot_amt_pledged": 149422709.86,
    "tot_backers": 1407993,
    "tot_goal": 566253100.85
  }
}
```

In a similar fashion, the average metrics by category were obtained thanks to the following query:

```
> db.project.aggregate([
    {$group:{_id:{category:"$main_category"}, avg_amt_pled:
        {$avg:"$pledged"}, avg_backers:{$avg:"$backers"}, avg_goal:
        {$avg:"$goal"}}, {$sort:{avg_amt_pled: -1}}}
])
```

The results can be seen the in following output:

```
{
  "_id": {
    "category": "Design",
    "avg_amt_pledged": 27119.75,
    "avg_backers": 241.30,
    "avg_goal": 46733.63
  },
  "_id": {
    "category": "Technology",
    "avg_amt_pledged": 22586.16,
    "avg_backers": 164.4,
    "avg_goal": 119712.32
  },
  "_id": {
    "category": "Games",
    "avg_amt_pledged": 21865.17,
    "avg_backers": 321.78,
    "avg_goal": 50711.29
  },
  "_id": {
    "category": "Comics",
    "avg_amt_pledged": 6899.31,
    "avg_backers": 134.77,
    "avg_goal": 20243.65
  },
  "_id": {
    "category": "Fashion",
    "avg_amt_pledged": 6549.03,
    "avg_backers": 61.45,
    "avg_goal": 24818.25
  },
  "_id": {
    "category": "Film & Video",
    "avg_amt_pledged": 6362.73,
    "avg_backers": 66.02,
    "avg_goal": 84334.01
  },
  "_id": {
    "category": "Food",
    "avg_amt_pledged": 5340.16,
    "avg_backers": 54.17,
    "avg_goal": 48693.75
  },
  "_id": {
    "category": "Theater",
    "avg_amt_pledged": 4097.22,
    "avg_backers": 47.06,
    "avg_goal": 27542.34
  },
  "_id": {
    "category": "Music",
    "avg_amt_pledged": 3992.73,
    "avg_backers": 52.17,
    "avg_goal": 16056.36
  },
  "_id": {
    "category": "Dance",
    "avg_amt_pledged": 3690.80,
    "avg_backers": 42.80,
    "avg_goal": 1407993.0
  }
}
```

```

"avg_goal" : 10321.33 }
{ "_id" : { "category" : "Photography" }, "avg_amt_pledged" : 3664.65, "avg_backers": 39.71, "avg_goal" : 13003.23}
{ "_id" : { "category" : "Publishing" }, "avg_amt_pledged" : 3638.72, "avg_backers": 55.96, "avg_goal" : 29131.46 }
{ "_id" : { "category" : "Art" }, "avg_amt_pledged" : 3606.97, "avg_backers": 42.20, "avg_goal" : 40829.18}
{ "_id" : { "category" : "Journalism" }, "avg_amt_pledged" : 3218.08, "avg_backers": 42.20, "avg_goal" : 88783.58 }
{ "_id" : { "category" : "Crafts" }, "avg_amt_pledged" : 2016.15, "avg_backers": 27.28, "avg_goal" : 11592.19 }

```

Although querying in MongoDB is relatively simple, the team had concluded that compared to Python the output results are less attractive, as it can be seen in the query outputs previously shown. We have also concluded that the overall usage is less user-friendly.

12.10 CONCLUSION

In this project, our team was able to deploy a successful Python application that has the capability of running on various cloud services using MongoDB as a stable backend. The team had observed that the performance was better on DigitalOcean and Amazon platforms (44 seconds) which have a dedicated memory and CPU, as well are more hardware level customization options such as CPU,Memory etc. compared to the MongoDB Atlas cluster where it took 96 seconds to run the aforementioned Python script. Various factors need to be taken into consideration when benchmarking performance on different services. Some of them include shared infrastructure load on which the cloud VM's are mostly available, dedicated resources such as CPU and memory, which play a very important role. Overall, we have concluded that each platform has its own advantages such as ease of use, customization of the environments, however, one of the very important aspects also includes the cost of the offered services. Our team had used the minimal cost approach in this project, and ran the code on the available free tier level services, which could have also affected the performance. When it comes to the crowdfunding data, we have found that the overall success rate of the **Kickstarter** projects in the 2009-2018 time frame was 36%, and that each year, the number of failed projects was much higher compared to their successful counterparts. We had also identified the most successful categories - **Technology, Games, Publishing, Music, Film & Video**. From those categories, **Technology** and **Games** were among the categories with the highest amount of pledged funds, including **Design**. The highest number of

successful projects was achieved in 2014 while the number of suspended projects was the highest in 2015. Overall, 2011 was the year of the highest success rate. By conducting the monthly time series analysis, we had identified certain trends where we could see that warmer months yielded more project launches while the colder months were sequentially less project heavy. Although the platform was created in the United States, countries such as Austria , China, and Netherland shown to be the top countries for generating pledged funds. Project duration was also analyzed and the team had concluded that the average duration of the successful projects was less than 30 days, while the upper limit for the maximum project length was 45 days. Moreover, thanks to our implemented machine learning methods, we were able to predict the success and failure of the projects based on the extracted features such as goal, backers, categories, and duration. However, we also believe this model can be expanded with additional feature engineering and stimulate sponsors to fund projects that are more likely to be successful with the sole purpose of keep encouraging the start-ups.

12.11 ACKNOWLEDGEMENT

The authors would like to thank the Big Data Applications and Analytics (I-523) course teaching staff, mainly professor Gregor von Laszewski for their support and guidance during this project. Also, we would also like to extend our appreciation to Kaggle for providing us with the ***Kickstarter Projects*** dataset, as well as to other online sources for allowing us to gather meaningful insights and programming support.

12.12 WORKBREAKDOWN

12.12.1 Nishad Tupe

- Project cloud architecture and implementation research
- MongoDB Atlas, Digital Ocean Cloud VM set ups
- Bash script for Dataset download, installation & import of MongoDB
- Python data loading scripts and exploratory visualizations
- Python Logistic regression implementation

12.12.2 Vishal Bhoyar

- Project requirements and design research
- Exploratory Analysis of data
- Amazon Cloud VM set up
- PyMongo research

12.12.3 Izolda Fetko

- Project dataset and literature research
- Testing MongoDB Queries on Cloud VM's
- MongoDB Aggregation framework research
- Python Time Series Analysis

12.13 NISHAD TUPE, VISHAL BHOYAR, IZOLDA FETKO

- Performance benchmarking
- Project Papers

13 TWITTER TEXT MINING WITH PYTHON AND MONGODB

Jay Stockwell
jaystock@iu.edu
Indiana University
hid: fa18-523-61
github: [blue link icon](#)

13.1 ABSTRACT

The paper's objectives are to provide a thorough analysis, and detailed, informative document pertaining how the Twitter API tool, in conjunction with Python and MongoDB Atlas can provide an effective solution for collecting and analyzing tweets. Additionally, other objectives are to provide a thorough understanding pertaining to how the twitter API works, how twitter data is constructed, how Python's tweepy library works to collect data, and how MongoDB Atlas cloud service provides a robust environment to store twitter data for future analysis. We will also discuss the implications of using the Twitter app and its data within the realm of Big Data and Natural Language Processing.

13.2 INTRODUCTION

Twitter allows individuals and organizations to post short messages called tweets. Individuals can follow other sites within Twitter to receive their tweets and to stay informed in subjects that appeal to them. Twitter is also thought of as an instant messaging and micro-blogging application that allows users to communicate their opinions and thoughts in a completely open and uninhibited manner. One major component of Twitter is the hashtag. Users can append a hashtag '#' in front of a word or phrase within their tweet in order to categorize them so other users can find and contribute to your tweet [110]. The hashtag can quickly draw attention to, and promote your tweet very effectively. The implications of how the data on Twitter can be used are simply astounding. In particular, Twitter is used daily by businesses to find new customers, market new products, find marketplace trends, and many other uses.

Twitter data, and social media data in general, represent a huge component of Natural Language Processing (NLP). NLP in a nutshell is the ability of machines to decipher and understand human language [111]. This can sometimes pose a challenge because language can tend to have multiple meanings and contexts, and can prove to be very ambiguous to machines which tend to rely on highly structured and precise programming languages [111]. Recent advances in NLP technology have eliminated some of the challenges facing NLP, specifically with part-of-speech tagging and Natural Language Understanding, and have greatly improved how semi-structured and unstructured text data is analyzed and leveraged today. NLP is largely based on the deep learning concept, which is prevalent in the emerging world of Artificial Intelligence. Deep learning algorithms look for patterns in data and use those identified patterns to help machines derive understanding from textual data. Another aspect of NLP that has major implications for Twitter is Sentient Analysis. Sentient Analysis can help organizations ascertain whether a particular product is seen in a positive or negative way based on user comments or reviews. Sentient models can analyze the language components and provide valuable data about products and/or services that can affect corporate decision making and product marking and placement strategies.

Data Scientists can mine the data on Twitter and discover new insights and uses of the data through the applications of machine learning algorithms, such as the deep learning algorithms mentioned earlier. For example, users can examine through the use of the Twitter API, how to mine for tweets that contain the hashtags #dog, #cat, or both. Users can also examine how to clean up the data in python and also examine and contrast two different machine learning classification algorithms to ascertain just how effective it is to classify tweets into specific categories. This type of analysis can have far-reaching implications as aforementioned previously, and specifically we can try to answer many questions about an individuals tastes, interests, and hobbies based solely on the information in their tweets. Users can also examine how effective twitter data is being stored within cloud database, in particular MongoDB Atlas to determine if that platform provides an effective storage infrastructure for json-styled, semi-structured data.

13.3 TWITTER API AND PYTHON IMPLEMENTATION

Twitter provides a dedicated API platform for developers to allow for the development of custom applications to collect and use tweets. This API can be used for both personal and business purposes. Users can tap into the vast social network for numerous reasons such as collecting specific tweets and placing in a datastore, integrating tweets from twitter within your own website or application, monitor your own twitter accounts to see how individuals are engaging [112]. Twitter provides several different API's for individuals to use based on their final goals of how they intend to leverage the data. For most basic needs, the standard API will work. However, twitter offers an Enterprise API for organizations that depend on Twitter for their day-to-day business.

In order to gain access to Twitter data through the API, users are required to create a Twitter account. The signup process requires users to provide basic user information, as well as information about your purpose and intent of your objectives with Twitter. Upon completing the sign up process, Twitter reviews your information and then assigns 4 keys that will be required for authentication between your application and Twitter. These 4 keys consist of consumer tokens and secrets that provide application authentication, and access tokens and secrets that provide the actual user or account authentication [113].

One method of connecting to, and collecting Tweets is through Python. Python is an open source, general purpose programming language that has gained tremendous traction in the data science field recently due to its ease of use and the availability of many different toolkits that can be implemented to complete data science related tasks. Python toolkits such as numpy, sklearn, pandas, and matplotlib can be leveraged to apply statistical analysis, machine learning algorithms, and quick visualizations using your data [114].

As Python has gained momentum with the Data Science field, another field in which the language is gaining ground is Big Data. Python is now compatible to work with the popular big data tool Hadoop [115].

“Python consists of Pydoop package which helps in accessing HDFS API and also writing Hadoop MapReduce programming. Besides that Pydoop enables MapReduce programming to solve complex big data problems with minimal effort [115]”.

Python is very scaleable within this type of environment. Another great component is that Python has a very large, robust user community that users can reach out to for guidance on all sorts of coding related topics. Due to the popularity of the language, experienced users are extremely willing to participate in the community to facilitate its growth and continued usage within the programming world.

Python can be leveraged using a basic shell program, which can accomplish basic tasks, or with an interpreter or IDE (Integrated Developer Environments). An IDE is a dedicated program, integrated with several tools such as debuggers, build and execution tools, syntax highlighting, and source control tools, that are specifically used to writing software code [116]. Some examples of well known Python specific IDE programs are PyCharm, Spyder, and Thonny. When tasked with developing more sophisticated Python code for data science purposes, it is a good practice to use a dedicated code editor well suited to the task.

In regards to working with Twitter, there is a dedicated library entitled Tweepy which can be loaded into Python and installed along with any other required libraries and toolkits. Users will need to use the pip command at a terminal screen in order to install tweepy. PIP is a package management system used by Python to install and manage larger software packages [117].

```
pip install tweepy
```

After installation is complete, users will need to use the import function to fully load the tweepy program into the python script that's currently in development.

```
import tweepy
```

The next step is authentication between the Twitter API and python. As mentioned earlier, there are 4 required keys needed for this step for which Tweepy is responsible for completing. Tweepy currently supports OAuth authentication and authentication is handled via the tweepy AuthHandler class in python [118]. OAuth authentication is considered the standard method for token authentication by 3rd-party applications such as Facebook and Twitter. OAuth works on behalf of the end user to provide a token which authorizes specific information to be shared between applications [119]. There are four variables in the python code that will be used to hold the security information

```
consumer_key=YOUR_CONSUMER_KEY  
consumer_secret=YOUR_CONSUMER_SECRET  
access_token_key=YOUR_ACCESS_TOKEN  
access_token_secret=YOUR_ACCESS_TOKEN_SECRET
```

Once tweepy is set up and the user passes through the authentication step, users can now begin further development on python code that can be implemented against Twitter data.. Tweepy collects tweets in real-time, so the script will be collecting tweets that were released just prior, or during the implementation of the script. The tweepy program in Python can only collect 100 tweets at a time, so the script can be built to contain a function which runs in an iterative loop to collect a set number of tweets as defined by a ‘MaxTweets’ parameter. Once that parameter value is met, the function terminates.

Next, users need to install the latest version of Python, which is version 3.7.1. Python is an open source general purpose programming language that is very useful for data science projects. There are numerous libraries available to install within Python that will help you accomplish your end goal. One of the most popular libraries available for working with twitter data in python is tweepy. Tweepy is designed to handle multiple aspects of twitter tweet collection including authentication, connection, session management, and reading and routing incoming messages [118].

13.4 DATA

Twitter data, when extracted, is in JSON format. JSON (Javascript Object Notation) is a data interchange format that is both easy for humans to read and write, and easy for machines to parse and implement [120]. A json object is basically a set of key value pairs ending contained within two braces. This format borrows heavily from the Javascript programming language, hence the Javascript component of the term [121]. The format is considered to be structured, as you would expect to see relational data within a table, but it’s considered to be thought of as semi-structured which “..the structure is either flexible or not fully predefined. It is sometimes also referred to as a self-describing structure [121].” The JSON datatype has proven to be very popular and has garnered wide usages with web sites due to the effective way in which JSON data can be exchanged between client and server web applications [121]. Below is sample JSON object:

```
{"first name": "Jay", "last name": "Stockwell", "hometown": "Louisville", "hobby": "playing guitar"}  
[<] [>] [x]
```

A json array is essentially a collection of key values contained within two brackets

```
{  
  "first name": "Jay",  
  "last name": "Stockwell",  
  "hometown": "Louisville",  
  "hobby": "playing guitar"  
  "favorite cars": ["Porsche", "Lamborghini", "Jeep"]  
}
```

Twitter JSON data is comprised of many different components, all of which are used to describe individual tweets.

"At Twitter we serve many objects as JSON, including Tweets and Users. These objects all encapsulate core attributes that describe the object. Each Tweet has an author, a message, a unique ID, a timestamp of when it was posted, and sometimes geo metadata shared by the user. Each User has a Twitter name, an ID, a number of followers, and most often an account bio [112]."

The tweets follow a parent-child construction. All tweets contain a user object which can also contain a geo-tagged child object describing the geographic location of where the tweet originated. The tweet also contains an entities object that consists of information such as assigned hashtags, URLs, user mentions, and any sort of media material [112]. The data record also contains a flag to indicate if the tweet has been retweeted, or has been forwarded by someone else to another person. A retweet is typically comprised of someone else's comments that a user would like to share.

13.5 TWITTER CLOUD STORAGE

There are many options available today to store twitter data within a cloud storage platform. Cloud storage consists of storing data within logical pools that can span multiple servers and multiple locations throughout many locations [122]. Cloud services can be accessed through a dedicated cloud service platform, and website APIs such as cloud desktop storage and gateways. [122].

Some of the main players in today's cloud computing market are Amazon Web Services, Microsoft Azure, and Google iCloud.

Amazon Web Services (AWS) provides a portfolio of services that can help organizations deal with the voluminous amounts of data that's available in Twitter. AWS provides a cloud storage service called S3 which is capable of storing data of any type from a variety of sources including web sites, mobile apps, and even IoT sensors [123]. Used in conjunction with Amazon Glacier, and Amazon Glue, one could build a secure data lake in the cloud that could be set up to contain streaming twitter data. Once the data is in the data lake, one can take advantage of cutting-edge advanced machine learning and analytics capabilities available through additional Amazon services such as AWS Athena (Interactive analytics), AWS Kinesis (Real-Time Analytics), and AWS Sagemaker and Deep Learning AMIs [123].

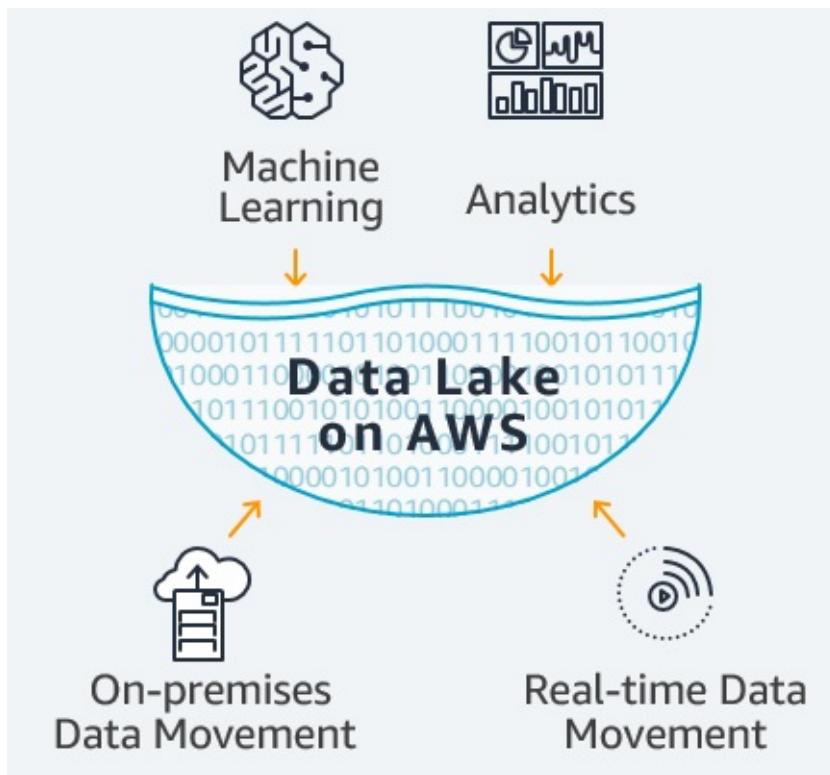


Figure 127: AWS S3 Data Lake [123]

A lot of organizations today leverage the power and scalability of the S3 platform and come to rely on the system for its day-to-day data needs. That reliability was tested during an AWS S3 outage that occurred on February 27th, 2017 that effected Amazon's entire US-EAST-1 region. This outage caused

widespread website outages and vast disruptions to Amazon's clients. The issue was caused by an Amazon employee typing an incorrect command which caused several key subsystems to go offline [124]. This was an embarrassing incident for Amazon that just goes to show how important cloud services have become in recent years, and how much some clients depend on these services.

In addition to the S3 platform, Amazon includes other services that allow users to set up and manage virtual machines and virtual cloud platforms. Users can create a cloud cluster leveraging one of these services and run a different database application that is more suitable to their data needs. MongoDB Atlas, an open source database storage system that provides support for JSON document style data, is one of those types of applications that can be stood up and managed on an Amazon Virtual Private Cloud (VPC). Mongo Atlas provides a platform from which users can create and manage their own clusters, as well as set up automation jobs to manage provisioning and deployment tasks [125]. Users are required to create an account with Mongo DB Atlas, which provides a minimum of 512 MB of storage for a free cloud database, but can be updated to 10 GB or more along with dedicated RAM, backups, and performance optimization with the creation of a dedicated cluster [125]. In addition to running on Amazon VPC, MongoDB Atlas can also run on Microsoft Azure and Google Cloud.

Connecting to MongoDB Atlas from python is fairly straightforward and requires the pymongo python library to be installed. The pymongo library contains several tools that enables users to connect to their MongoDB data, either on their local machine, or a database hosted on a MongoDB Atlas cluster. Users must use the pip method from a command prompt or terminal screen and run the following command:

```
python -m pip install pymongo
```

After the pip command is completed and pymongo is installed, the library must be imported into python using the following simple command:

```
import pymongo
```

The Mongo Client component is one of the pymongo tools allowing for python to connect to a MongoDB database. Connecting to a local instance of MongoDB is simple and requires that the software is fully installed and the MongoDB

service is up and running. Also, port 27017 must be available on your machine in order for the two applications to communicate properly. The python command for connecting to a local MongoDB instance is as follows:

```
client = pymongo.MongoClient("localhost", 27017)
```

Connecting to a MongoDB Atlas cluster involves a few more steps. First, before you can start up a cluster, there are a few requirements that need to be met. Users must have the correct TLS/SSL support in place and be able to have the correct SNI driver installed in order to connect [126]. Users must also add their IP address to the MongoDB Atlas Whitelist. This ensures that the only connections allowed will be listed in MongoDB's whitelist [126]. Next, users must create a user account with Admin access in order to be able to administer the cluster. After those steps are complete, and the cluster has been started up, users must choose a connection method to use in order to connect to another application. The connection dialog contains different connection strings for different versions of applications including Java, C, C++, Perl, Ruby, and Python. To set up a connection between MongoDB Atlas and Python, users will need to select a URI connection string based on their version of Python. Next, they will need to paste that string within their python script within the following pymongo command:

```
client = pymongo.MongoDB(mongodb://USERNAME:PASSWORD@cluster0-sample-mongodb.net:27017/<DAT
```

After this is complete, and you're able to connect without any issues, you can begin to create databases, collect and insert data into the MongoDB database, as well as other data related tasks. Just a note on connecting to MongoDB, users will sometimes receive error messages pertaining to DNS connection errors. To resolve this, users need to install the dnsPYTHON library using the pip method and importing the resolver tool from the dnsPYTHON library within python.

```
python -m pip install dnsPYTHON
```

```
from dnsPYTHON import resolver
```

Once fully connected to MongoDB, you begin to 'listen' or search for tweets that contain a specific word, phrase, user, #hashtag, or any other piece of information that you're interested in mining for. There's a couple of ways to set up the search criteria, but typically a variable is created that contains the search words such as listed below:

```
count = 50
q="cats"
search_results=api.search.tweets(count=count, q=q)
```

You can configure tweepy to work in a search or streaming manner. Tweepy has a class entitled StreamListener that will access the Twitter API and pull all tweets are created using the specified criteria [127]. Once executed, the script will continue streaming until the script is stopped. The api.search method will provide a collection of tweets based upon a count variable specified earlier. One thing of note is the Tweepy will only return 100 tweets at a time, so the script will need to contain an iterative loop function to run continuously until the value specified in the count variable is met.

After the script is run, the data can now be inserted into the MongoDB Atlas database. The pymongo library includes an insert function that will allow the user to insert either one record, or many records into the database. In order to use the insert many function, you will need to create a collection database object. A MongoDB collection is essentially a collection of documents, or in relation to Twitter data, a collection of json documents [128]. A collection can be thought of as being similar to a SQL relational table [129]. The collection, if it doesn't already exist, can be created within the insert statement itself. Below are two examples:

```
# Insert one record only
db = mydb.insert_one(mydict)

# Insert multiple records
db = mydb.insert_many(mydict)
```

The python script can be run multiple time to continuously gather tweets. However, it is important that we don't collect the same tweets every time we run the script. Therefore, it is adviseable to create an index object after the initial connection to the MongoDb database and database table. This will ensure that each object is properly indexed and contains a unique ID field. Once the script is run subsequent times, an argument can be put in place to check the ID field and ensure that duplicate tweets are not being added.

Twitter Data contained within the MongoDB database can be queried from within Python using specific command line arguments and functions from the pymongo library. The argument findone() will return the very first record in the database and will include every item associated with that record. With Twitter

data, the `findone()` argument will return all of the components of the first tweet collected such as key, timestamp, description, user, etc. If you want to zero in on one specific item within the tweet, you can add some additional information to the `findone()` argument. For example, if a user wanted to see the key of the first tweet, they would add ‘key’ to the `findone()` argument as `findone.key()`. The argument needs to also contain the data collection that we are querying.

To return all of the records in the database, a cursor object must be created in the python script that will allow the user to read and analyze all of the records within the collection [130]. A cursor can be created by setting up a `find()` argument within a collection. You can also limit the results by including a `limit` argument set to the desired number of records.

MongoDB Atlas was designed to handle large datasets by spreading the data among many servers with the cloud computing platform [131]. MongoDB can also be connected to a Hadoop instance to deal with the upcoming challenges that are presented by Big Data. This creates an ideal environment to house streaming Twitter data as MongoDB is perfect for document data types such as JSON. MongoDB is also thought of as being a great platform for working with Big Data because of the high scalability offered by MongoDB [129]. However, there have been some points of contention that have arisen with NoSQL databases such as MongoDB. Some users that have used MongoDB to store twitter data have reported issues related to duplicate data, which was addressed earlier with the creation of an index. Another issue is that because of the structure of Twitter data and the document storage type, it is very hard, if not impossible to do SQL style joins within the data [132]. Social media data is highly unstructured, and individuals that are accustomed to working with relational SQL based databases may find this type of document styled data very hard to understand, analyze, and construct effective queries against.

Conversely, querying and analyzing twitter data in python is a compelling and direct way to open many new doors to insights.

13.6 DATA SCIENCE ALGORITHMS FOR TWITTER DATA

There are numerous algorithms available to use in Python when running a script against a Twitter dataset. It’s a good idea to perform some dedicated research

and make a determination on which algorithms will accomplish your objectives. Deep Learning algorithms such as Neural Networks are very effective when implemented against twitter data, and can be compared using accuracy scores and performance metrics to other similar algorithms such as the Naive Bayes classification model.

The Naïve Bayes classification method is based on Bayes Theorem probability theory to classify data into distinct classes. The method assumes independence between all of the attributes used in the algorithm and it works on the assumption that a particular feature in a class is unrelated to the rest of the features included in the analysis [133]. Naive Bayes is a fairly simple algorithm to set up, and it runs very effectively against very large datasets such as those used in Big Data.

Below is the mathematical formula that comprises the algorithm. The formula follows the Bayes Rule of Probability:

$$P(c|x) = \frac{P(x|c)P(c)}{P(x)}$$

Likelihood Class Prior Probability
 ↓ ↓
 Posterior Probability Predictor Prior Probability

$$P(c|X) = P(x_1|c) \times P(x_2|c) \times \dots \times P(x_n|c) \times P(c)$$

Figure 128: Bayes Rule of Probability [133]

The foundation of the formula is that it calculates the posterior probability $P(c|x)$ from the other variables in the equation, $P(c)$, $P(x)$, and $P(x|c)$ [133].

The Naive Bayes classification method has gained popularity recently with usage as an effective spam filter, text analysis, and medical diagnosis tool. Users are able to utilize the multinomial Naïve Bayes model as it appears to be more suitable for text classification by explicitly modeling the word counts and makes adjustments to any underlying calculations involved.

Another popular algorithm to use with Twitter data is neural networks. Neural networks function as an information processing model that closely mirrors how biological nervous systems , specifically neurons, process information [134].

Neural networks make predictions by learning the relationships between your data features and other previous observations. This approach works by feeding data into an input layer that goes through a series of transformations within hidden layers until a final transformation occurs and a final output is produced. Deep neural networks go a step further due to the larger number of nodes or layers that the data passes through in a complex, multi-step process of pattern and correlation recognition [135]. Working with more layers provides more opportunities for the data to train on the features and patterns recognized in the previous layer, which in turn produces a more and more complex hierarchy known as a feature hierarchy [135].

Neural networks are popular in natural language classification tasks because of the ability for neural networks to create an embedded layer. Through vectorization, the words in the embedded layer become mathematical arrays. These vectors can be very useful in text classification. Deep neural networks have gained traction with sentiment analysis in recent years due to the increased user requirements for this type of analysis, and how this analysis can handle increasing levels of complexity with very large unstructured data sets, some of which is not readily apparent [136].

13.7 RUNNING A TWITTER SCRIPT IN PYTHON

There are multiple ways that users can set up python and perform some basic text mining against Twitter. The scenario below is to design a script that pulls tweets with the hashtags #cats and #dogs. Working with these hashtags, Using python, users can generate a simple script that can be used to classify if a person is a cat or dog person based on their tweets. Users evaluate the performance and accuracy of the two classification algorithms used, as well as evaluate any issues and challenges that arose during their analysis.

13.7.1 Python Libraries

Below is a sample list of python libraries that can be used for twitter analysis:

```
import tweepy
import matplotlib.pyplot as plt
import matplotlib
!pip install tweepy
import sys
```

```

import jsonpickle
!pip install jsonpickle
import os, json
import pandas as pd
import numpy
import re
from wordcloud import WordCloud, STOPWORDS
import warnings
import sklearn
import sklearn.metrics
from sklearn import metrics
import sklearn.naive_bayes
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
import sklearn.svm
import sklearn.neighbors
import sklearn.neural_network
from sklearn.feature_extraction.text import CountVectorizer
import seaborn as sns

```

From the `sklearn` library, users can create a function to gauge the precision scores from the two algorithms that will be used later in the script. The precision scores will help assess whether a record was classified incorrectly, for example a record that is labeled a positive when it should have been labeled negative [137].

```

def print_score(Ytrue,Ypred):
    s = (sklearn.metrics.precision_score(Ytrue,Ypred),
          sklearn.metrics.recall_score(Ytrue,Ypred),
          sklearn.metrics.f1_score(Ytrue,Ypred))
    print('Precision: {:.3}\nRecall: {:.3}\nF-Score: {:.3}'.format(*s))

```

The next step in the code is authentication with the Twitter API. The concepts behind the authentication process were discussed earlier in the paper. If Twitter is unable to authenticate your account with the provided credentials, a “Can’t Authenticate” message is displayed and the script terminates.

```

"""twitter credentials. Removed for security reasons."""
Credentials file format:
consumer_key=YOUR_CONSUMER_KEY
consumer_secret=YOUR_CONSUMER_SECRET
access_token_key=YOUR_ACCESS_TOKEN
access_token_secret=YOUR_ACCESS_TOKEN_SECRET

"""Replace the API_KEY and API_SECRET with your application's key and secret."""
auth = tweepy.AppAuthHandler(consumer_key, consumer_secret)

api = tweepy.API(auth, wait_on_rate_limit=True,
                  wait_on_rate_limit_notify=True)

if (not api):
    print ("Can't Authenticate")
    sys.exit(-1)

```

Now that the user is authenticated, they can specify the search criteria for the specific tweets needed for identification and collection. A variable called ‘SearchQuery’ contains the #cats hashtag that users may want to search on, the maxTweets variable contains the maximum number of tweets to be collected, the ‘tweetsperQry’ is set to 100 as this is the maximum amount of tweets the Tweepy API is allow to gather during one iteration using the search function of the API. The ‘fname’ contains the name of the file in which the tweets will be collected.

```
"""Information about the type of tweets we want to find as well as how many"""
searchQuery = '#cat' # this is what we're searching for
maxTweets = 2000 # Some arbitrary large number
tweetsPerQry = 100 # this is the max the API permits
fName = 'cat_tweets.txt' # We'll store the tweets in a text file.
```

Since the Tweepy API only allows 100 tweets per iteration, the script contains a functions that runs in a loop until the maxTweets value has been reached. If any issues arise, the function will terminate.

```
SinceID = None
max_id = 1
tweetCount = 0
print("Downloading max {0} tweets".format(maxTweets))
with open(fName, 'w') as f:
    while tweetCount < maxTweets:
        try:
            if (max_id <= 0):
                if (not sinceId):
                    new_tweets = api.search(q=searchQuery, count=tweetsPerQry)
                else:
                    new_tweets = api.search(q=searchQuery, count=tweetsPerQry,
                                            since_id=sinceId)
            else:
                if (not sinceId):
                    new_tweets = api.search(q=searchQuery, count=tweetsPerQry,
                                            max_id=str(max_id - 1))
                else:
                    new_tweets = api.search(q=searchQuery, count=tweetsPerQry,
                                            max_id=str(max_id - 1),
                                            since_id=sinceId)
            if not new_tweets:
                print("No more tweets found")
                break
            for tweet in new_tweets:
                f.write(jsonpickle.encode(tweet._json, unpicklable=False) +
                       '\n')
            tweetCount += len(new_tweets)
            print("Downloaded {0} tweets".format(tweetCount))
            max_id = new_tweets[-1].id
        except tweepy.TweepError as e:
            # Just exit if any error
            print("some error : " + str(e))
```

```
        break

    print ("Downloaded {0} tweets, Saved to {1}".format(tweetCount, fName))
```

The iteration loop has completed, and two sample two datasets were created that contains data for the hashtags of #cats and #dogs. Working with twitter data that was contained in dataframes was easier to analyze. The pandas dataframe function was used to take the twitter data from the iteration script and load into two separate dataframes.

```
dogs = pd.read_json("C:/Users/sample/dog_tweets.txt",lines=True)
cats = pd.read_json("C:/Users/sample/cat_tweets.txt",lines=True)
```

A crucial step when working on a new set of data is to perform some type of data preprocessing and exploration. This gives you a good sense of that data contents, and if you need to perform some cleansing of the data such as removing null values and incorrect characters or words.

The first two lines provide a record count. Next, using the dogs dataset, a new column was created entitled tweety and extracted all the URLs and usernames from the collected tweets into that new column. All of the URLs, RT's (retweets), and usernames or twitter handles were removed.

```
dogs['tweety'] = ''

#add tweety first part
for i in range(len(dogs['text'])):
    try:
        dogs['tweety'][i] = dogs['text'].str.split(' ')[0]
    except AttributeError:
        dogs['tweety'][i] = 'other'

#Preprocessing tweety. select tweety contains 'RT @'
for i in range(len(dogs['text'])):
    if dogs['tweety'].str.contains('@')[i] == False:
        dogs['tweety'][i] = 'other'

# remove URLs, RTs, and twitter handles
for i in range(len(dogs['text'])):
    dogs['text'][i] = " ".join([word for word in dogs['text'][i].split()
                               if 'http' not in word and '@' not in word and '<' not in word])
dogs['followers_count'][1]
```

Within Natural Language Processing, there are textual components that do not provide any significance to the analysis. These are referred to as stopwords. Some examples of stopwords are *is*, *the*, and *and*. Stopwords should be removed

during the processing step to eliminate unnecessary information from being included in the analysis [138]. The nltk python library has contains a corpus library of which contains a comprehensive list of stopwords that can be used within the script.

```
import nltk
import nltk.corpus
nltk.download('stopwords')
stopwds = list(nltk.corpus.stopwords.words('english'))
## stopwds
```

13.7.2 Visualizations

Visualizing twitter data in python can be accomplished with the use of a variety of different python packages and tools. One of the more popular and useful libraries for creating plots, charts, and other visualizations is Matplotlib. The Matplotlib python library is a 2D graphical tool that creates publication-like visualizations within a variety of platforms including python, ipython, Jupyter notebooks, and other web applications [139]. “Matplotlib makes easy things easy and hard things possible” [139]. Matplotlib installation is straightforward and can be completed with the following code:

```
import matplotlib
import matplotlib.pyplot as plt
```

Matplotlib contains a tool called pyplot which allows for very simple plotting as well as creating bar charts, line charts, and histograms. The pyplot tool can be utilized to create line and bar charts to show the distribution of twitter users by source. Below is a bar graph depicting the number of followers by source for the #dogs twitter dataset:

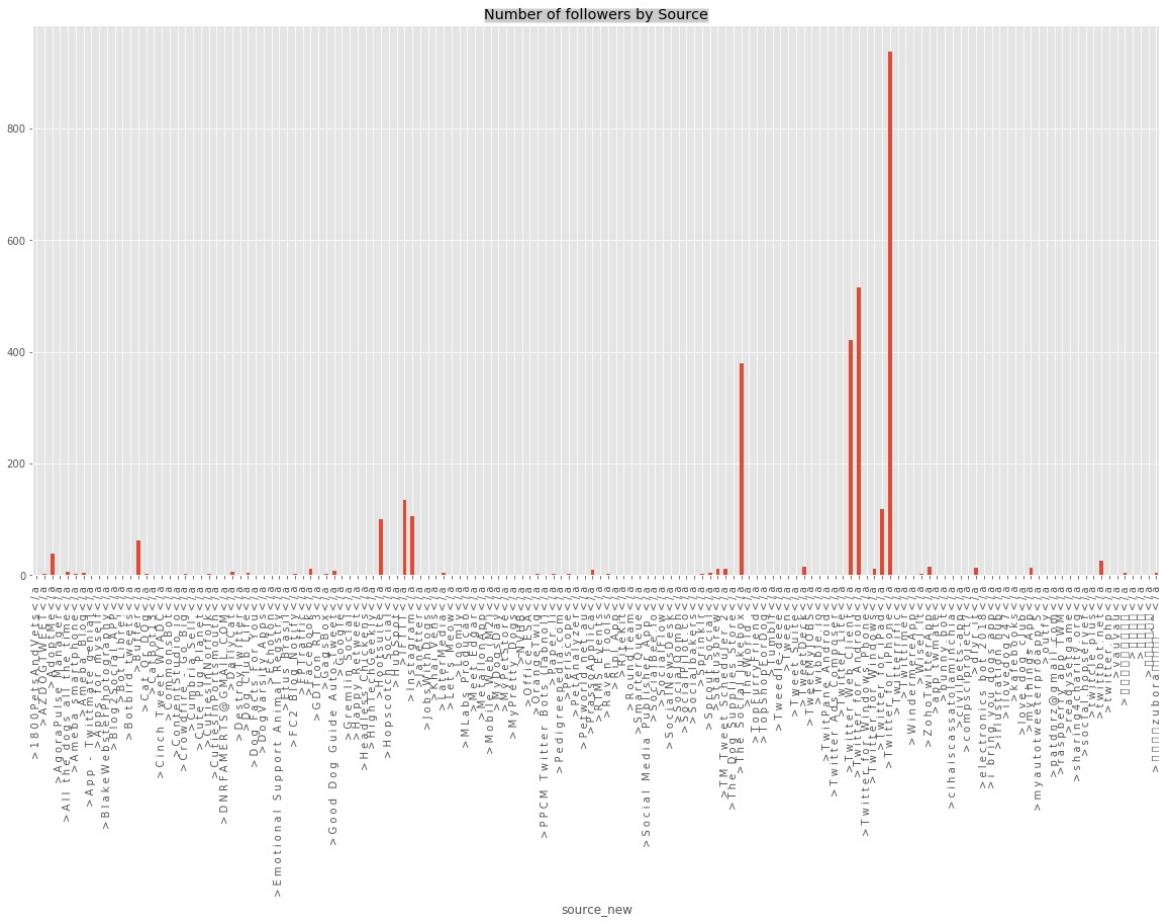


Figure 129: Twitter Source Bar Chart

From looking at the bar chart, one could surmise that the top source for all of the tweets is the iPhone, followed by Twitter for Android, and then the Twitter Web Client application. This is no surprise given the ubiquitous nature of smart phones today.

The python code used to generate the bar chart is below:

```
dogs['source_new'] = ''

for i in range(len(dogs['source'])):
    m = re.search('>(.*)</a', dogs['source'][i])
    try:
        dogs['source_new'][i]=m.group(0)
    except AttributeError:
        dogs['source_new'][i]=dogs['source'][i]

dogs['source_new'] = dogs['source_new'].str.replace(' ', ' ', case=False)

dogs['source_new'].head()
```

```
tweets_by_type = dogs.groupby(['source_new'])['favorite_count'].sum()
plt.title('Number of followers by Source', bbox={'facecolor': '0.8', 'pad': 0})
tweets_by_type.transpose().plot(kind='bar', figsize=(20, 10))
```

Another effective visualization for analyzing and visualizing twitter datasets are wordclouds. Wordclouds depict groupings of words in different sizes depending on how frequently they are displayed within tweets. The Wordcloud python library can be installed using the pip method and imported into the python script with the following commands:

```
python -m pip install wordcloud

from wordcloud import WordCloud, STOPWORDS
```

Below is the python code to generate two different wordcloud visualizations; one depicting text and another depicting language.

```
""" Dogs Text Wordcloud"""
def wordcloud(tweets,col):
    stopwords = set(stopwds)
    wordcloud = WordCloud(background_color="white",stopwords=stopwords,random_state = 2016)
        .generate(" ".join([i for i in dogs[col]]))
    plt.figure( figsize=(20,10), facecolor='k')
    plt.imshow(wordcloud)
    plt.axis("off")
    plt.title("Dog DataSet")
wordcloud(dogs, 'text')

""" Dogs Language Wordcloud"""
def wordcloud(tweets,col):
    stopwords = set(stopwds)
    wordcloud = WordCloud(background_color="white",stopwords=stopwords,random_state = 2016)
        .generate(" ".join([i for i in dogs[col]]))
    plt.figure( figsize=(20,10), facecolor='k')
    plt.imshow(wordcloud)
    plt.axis("off")
    plt.title("Dog DataSet")
wordcloud(dogs, 'lang')
```

Below are two wordcloud charts; one illustrating the top words or phrases seen within the Twitter #dogs dataset, and another illustrating the top languages used within the same dataset.



Figure 130: Text Wordcloud

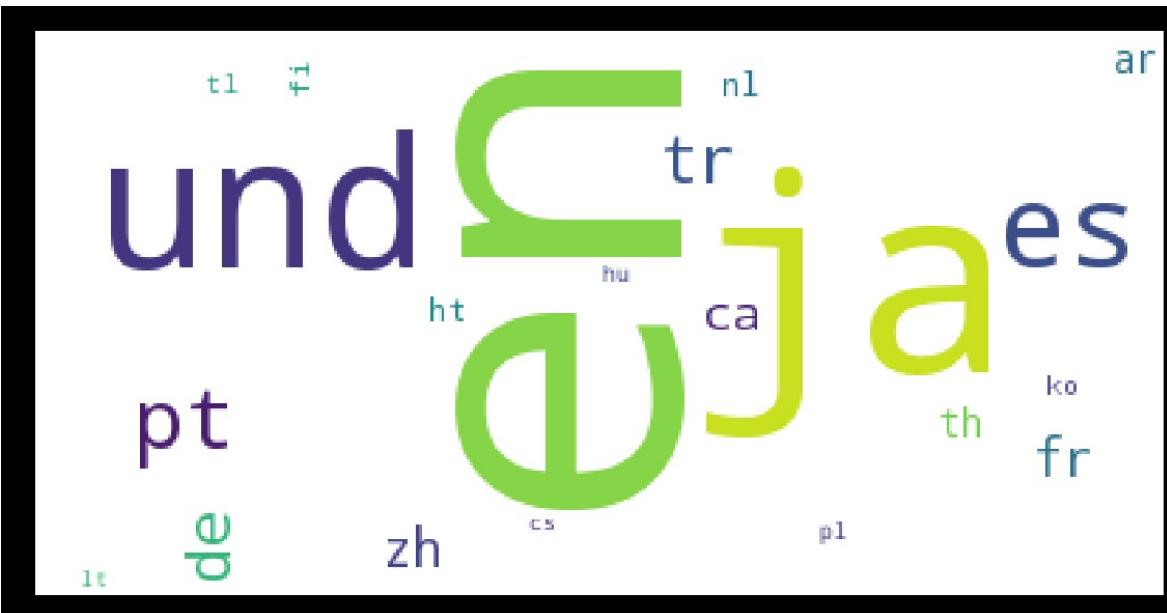


Figure 131: Language Wordcloud

The word **dog** is displayed prominently as expected, and there are also some other interesting words such as **love**, **pet**, **first** displayed as well. There are some other words that were not expected such as **millionaire**, **crypto**, and **trading binance**. This could lead to further research as to how these terms are related to tweets containing the #dog hashtag. The Language wordcloud illustrates that the majority of the tweets are in the English and Japanese language.

13.7.3 Machine Learning Algorithms

The Naive Bayes and Neural Networks algorithms can be set up and executed very seamlessly in Python using a variety of libraries. Users can import and implement the Naive Bayes and Neural Network algorithms from the sklearn library. Sklearn contains numerous machine learning algorithmic functions and toolkits, and is a manageable task as far as setup and implementation is concerned. After creating the train and test datasets in python with the desired twitter data, setting up and running the algorithm in Python is straightforward.

In order to effectively run the algorithms, there are a few model preparation steps needed to be completed. For the cat and dog datasets, the hashtags should be removed and replaced with a filler, non-significant value.

```
cats_txt = [x.replace('#cat', "BLAH") for x in cats['text']]  
dogs_txt = [x.replace('#dog', "BLAH") for x in dogs['text']]
```

In addition to setting up the algorithm, it is advisable to add in a vectorization component from the sklearn library to incorporate within the algorithm. The vectorization process is used to transform textual information into numerical information that machine learning algorithms such as Naive Bayes and Neural Network can understand and process more efficiently [140]. the stopwords component can also be pulled into play at this step to ensure these elements are excluded from this step. Below is the python code to accomplish this:

```
vectorizer = sklearn.feature_extraction.text.CountVectorizer(cats_txt+dogs_txt, analyzer='w'  
stop_words=stopwds, min_df=5)  
vectorizer.fit(cats_txt+dogs_txt)  
cat_tdm = vectorizer.transform(cats_txt).toarray()  
dog_tdm = vectorizer.transform(dogs_txt).toarray()  
vectorizer.get_feature_names()
```

The next step would entail combining the arrays into one matrix, and then assigning adding in values the class labels.

```
catdog_tdm = numpy.concatenate((cat_tdm, dog_tdm), axis=0)  
Ycat = numpy.array([0 for i in range(len(cats_txt))])  
Ydog = numpy.array([1 for i in range(len(dogs_txt))])  
Y = numpy.concatenate((Ycat, Ydog), axis=0)
```

At this point, the data must e split into two distinct data sets; one for training and another to test the algoritm. The training dataset is used by the machine learning

algorithm to learn about the the data and its possible outcomes. The test data is used to test the algorithm after it has run through the training phase. Typically the training data set is larger in size. Below is the python code for splitting data into train and test datasets:

```
xtrain,Xtest,Ytrain,Ytest = sklearn.model_selection.train_test_split(catdog_tdm, Y, test_s:  
[1]
```

In this case, the test data comprised 20% of the dataset while the train data is 80%.

As mentioned earlier, the Naive Bayes Multinomial algorithm was determined to be one of the more effective algorithms for tweet classification. The python code is below.

```
mnb = sklearn.naive_bayes.MultinomialNB()  
mnb.fit(Xtrain,Ytrain)  
Ypred = nb.predict(Xtest)  
  
print("\nNaive Bayes Performance")  
print_score(Ytest,Ypred)  
metrics.accuracy_score(Ytest, Ypred)  
metrics.confusion_matrix(Ytest,Ypred)  
print(Ytest.value_counts())
```

The mnb.fit portion of the code above is fitting the data into the Naive Bayes Algorithm, and the nb.predict function is running the algorithm against the test data. There are some accuracy scores generated as well as a confusion matrix to help gauge whether or not the model was fitted properly.

The Neural Networks algorithm is another algorithm that has been linked to high performance and results when used with Twitter data. The python code is also straightforward in setting up and requires network hidden layers to be defined as well as the iterations. The python code is below:

```
nn = sklearn.neural_network.MLPClassifier(hidden_layer_sizes=(30, 30, 30),max_iter = 500)  
nn.fit(Xtrain,Ytrain)  
Ypred = nn.predict(Xtest)  
  
print("\nNeural Network Performance")  
print_score(Ytest,Ypred)  
metrics.accuracy_score(Ytest, Ypred)  
cm = confusion_matrix(Ytest, Ypred)
```

13.8 CONCLUSION

In conclusion, using python to extract and analyze twitter data appears to be an effective tool that can accomplish an enormous amount of tasks. Python's powerful collection of libraries and tools really make a difference in regards to how users can gather results to accomplish their objectives. Python appears to be extremely flexible in allowing many different variables and options to be set. Regarding twitter data, Tweepy appears to be a very fluid and straightforward tool to setup and program. Creating a twitter api account and setting up the authentication in Python is also very uncomplicated. One caveat in terms of searching for actual tweets is that the number of tweets returned when using the tweepy package is set to 100. Users that want to return more than 100 tweets in their results may need to add in some additional programming to create a iterative loop function in order to get the desired number of tweets. This can add a level of complexity to the script.

Storing twitter data in the MongoDB Atlas cloud storage system can be an effective solution. Setting up a MongoDB Atlas account is effortless as well as starting up and managing a new cluster. The free tier is offered at no cost and includes up to 512 MB, however, this may inefficient for some, if not most, users who are working with very large amounts of twitter data. In order to perform an effective analysis, large amounts of data is required, and the free tier will probably not meet those requirements. MongoDB Atlas does offer services at higher pricing tiers that include both scalable RAM and scalable storage. These options may prove to be attractive for enterprise level entities. MongoDB Atlas has shown some issues at the data level pertaining to duplicate records being captured, however, those issues can be alleviated within Python by including some code that ensures that the ID fields of the tweets are unique.

13.9 FUTURE RESEARCH

Collecting social media for sentiment analysis is another area of interesting research that has arisen. Sentiment analyze can provide data on a more subjective level, as opposed to objective such as what is provided here. Many new questions arose such as how can using NLP to determine the state or emotion that a certain tweet is conveying, can this be accomplished in python somewhat efficiently, and what are the specific uses for this data.

13.10 ACKNOWLEDGEMENTS

The author gratefully acknowledges the input and guidance of Dr. Gregor von Laszewski, specifically in regards to paper syntax rules, bibtex reference usage, and the use of images within the markdown document.

14 PREDICT EPL RESULTS USING TWEETS

Manek Bahl, Sohan Udupi Rai
mbahl@iu.edu, surai@iu.edu
Indiana University
hid: fa18-523-62, fa18-523-69
github: [blue link icon](#)
code: [blue link icon](#)

Learning Objectives

- Learn about extracting Tweets and storing them on MongoDB
 - Run Machine Learning and NLP algorithms on the data to predict soccer results
-

Keywords: Twitter API, MongoDB, Machine Learning, NLP

14.1 ABSTRACT

There are a lot of factors that influence the outcome of a football game. Team statistics such as recent form, win-loss ratio, goals scored, goals conceded etc have been proven to be useful in dictating the results of a game. These statistics can be directly obtained from the Official EPL website, however, there are certain other factors such as injuries, general mood of the fanbase and sentiment of the fanbase, preparation by the team etc. which aren't documented in the official website. Analysis of Twitter data could prove useful in extracting these undocumented features, using NLP models. In this paper, we analyze the usefulness of Tweets in predicting the outcomes of games in the English Premier League. We create three different models to predict the outcomes, the first using the statistics available from the EPL website, the second using only the twitter data and the third with combination of both. If the model improves with the addition of features obtained from the twitter data, we would be able to prove that Twitter data does increase the predictive power of the model.

14.2 INTRODUCTION

The English Premier League is one of the most famous and competitive football leagues in the world. The season begins in August and ends in May in which 20 teams compete for the championship. Each team plays each of the other teams exactly twice, Home and Away, over the course of the season. A win awards the team with 3 points while a draw earns 1 point and a loss gives 0 points. At the end of the season, the team with the maximum points is crowned as the champion.

Twitter is a social media website where people express their opinions about a wide range of topics such as Politics, Sports, Religion, Entertainment etc. It was launched in 2006 and currently has 335 Million users worldwide. Due to the large number of users, Twitter has proven to be a reliable source of the general opinion of the public regarding any matter. When it comes to tweets regarding the Premier League, Twitter is being used not only by the public, but also by the players and the club officials. Therefore, we think that tweets could contain valuable information which could be used to improve the prediction models.

We are choosing the English Premier League due to its globally widespread fanbase which is active on Twitter. This provides us with a rich corpus of data regarding the team which could be used to analyze and interpret the overall fan sentiment for a team and other factors such as player unavailability, current coaching style etc. which are subjective and hence not obtainable from any other statistical sources.

For each team playing in the upcoming weekend fixtures, we are using Twitter scraping API in Python called Selenium, to monitor and extract all tweets over the course of the entire week. This is repeated for first 12 gameweeks, to build the corpus of tweets. These tweets are fed to Natural Language Processing tool to create a feature which serve as the predictor in the dataset. The corresponding results of the matches will serve as the class labels. With dataset with the statistics and tweet feature combined we run machine learning predictive algorithms to predict the results of the 13th gameweek.

14.3 RELATED WORK

There were a few papers in which an attempt has been made to predict soccer results using machine learning techniques but just using the statistical data. However, owing to the intense competition in the English Premier league and the unpredictable nature of soccer results a great accuracy was never achieved. In Timmaraju et al. they used game statistics such as number of goals, shots on target, corners etc [141]. They added another feature which was based on the recent form of the team in the last 5 matches. They were able to achieve the maximum accuracy of 66%.

We however, didn't find any other paper which included external feature such as tweet sentiment. We are not too concerned about the accuracy of the models, but we tried to leverage the usability of tweet data and check if it improves our model in any way.

14.4 IMPLEMENTATION

14.4.1 Data

The tweets were extracted using web scraping tool with Python called Selenium and were stored in MongoDB over the course of almost 3 months from September 2018 to November 2018. We were able to collect more than 100,000 tweets combined for 20 premier league teams using the team specific hashtags. Examples of hastags used are: #AFCB for Bournemouth, #Arsenalfc and #Gunners for Arsenal, #MANU and #RedDevils for Manchester United etc. In total we had 50 hashtags overall for different teams.

The statistical football data for this season was taken from English Football Data Website [142]. The data contains all individual match results for this season with number of goals scored by each team, number of shots, number of shots on target, corners etc. The data also contained win-loss odd prediction for each time according to various betting websites. The results were given as Home team Win, Away Team Win or Draw.

The above 2 datasets are seperately imported by our main code and the following processing is done to get them into the required format:

The tweet text imported from the database (or the CSV file in the case of this

demo) are fed to the TextBlob method from the python library ‘textblob’ and the ‘polarity’ attribute of the ‘sentiment’ attribute is extracted to get the overall sentiment of each tweet. These individual tweet sentiment scores are then grouped and averaged for each team into weekly batches from saturday through friday. We now have the average sentiment of tweets for each team leading upto each of weekend fixtures. This is added as one of the ‘predictors’ for the dataset to be fed to the machine learning algorithms.

Next, the statistical dataset (along with the outcomes of each match) is manipulated using for loops and dictionaries in Python so that the goals, shots, shots on target, fouls, corners, yellow cards and red cards are converted to their corresponding per match averages prior to the game. The odds data obtained are kept as obtained. These per game average values generated along with the odds data serve as the other ‘predictors’ in the dataset.

The outcome of each of the game labelled as ‘w’ for a win, ‘d’ for a draw and ‘l’ for a loss act as the ‘class label’ for the prediction algorithm in the dataset. The rows for data corresponding to gameweek 13 is seperated and stripped off the class label column, this acts as the test dataset. The rest of he data (including the ‘result’ column) acts as the train dataset for the prediction algorithms.

14.4.2 MongoDB

Developed by MongoDB Inc., MongoDB is a NoSQL Database system which stores data as documents. The documents are stored in JSON format which means that various documents can have their own different format and data structure. The structure of the existing documents as well can be changed later in time. Another advantage that this structure provides is that, it makes mapping of different objects in the application code much easier. High availability and horizontal scaling are in built features of MongoDB due to it being a distributed database. It was written in C, C++ and Javascript and can be used on various operating systems such as Windows, iOS, Linux and Solaris.

However, owing to default security configuration, MongoDB has been rendered to a lot of data thefts as it allows full database access to all [143].

14.4.2.1 Local Installation

We used the MongoDB Community Edition on Windows and MacOS systems. The below steps were followed for MacOS using Homebrew:

Updated the HomeBrew's version by the following shell command

```
brew update
```

Then use the below command to install the mongodb binary files

```
brew install mongodb
```

To run MongoDB the below command was used

```
<path to the binary files>mongod
```

Once MongoDB is up and running we should be able to see the following line in the shell or terminal window [144].

```
[initandlisten] waiting for connections on port 27017
```

14.4.3 Tweet Extraction

14.4.3.1 Tweepy and its challenges

Twitter provides dedicated API for developers and researchers to be able to extract tweets for any given hashtag or for a particular username and perform various kinds of analysis. For various use cases, Twitter provides different types of access levels with the business version being called the Enterprise API which large scale business use for their analysis. To obtain the developers access, one must own a twitter account and apply for a developer's API access. The form usually consists of basic user information and justifying the purpose of obtaining a developer's access for the API. This information is then reviewed by Twitter and assigns 4 unique keys for each user which are needed to use the API. The keys consist of consumer key, consumer secret, access token key and access token secret. Once these are obtained a dedicated python version of Twitter API called Tweepy can be used.

Tweepy is an easily installable python package which leverages the Twitter API using Python. Tweepy can be easily installed as other python packages using pip command in the command line.

```
pip install tweepy
```

The package can then be imported in python IDE and authentication between python and Twitter API can then be established using the keys obtained earlier.

Tweepy though can be used only to extract tweets only for 7 days. For tweets before 7 days, Tweepy cannot be used. However, for our project we need tweets starting from August 10, so using Tweepy is not feasible for our project.

14.4.3.2 Selenium

Since the data we needed from Twitter could not be extracted using official twitter API due to the 7-day restriction, we had to try another approach. We scraped data from Twitter using the selenium package in python. Selenium essentially is a package which is used to automate the interaction of python with a web browser of our choice.

To use selenium, we needed to install a WebDriver for the specific web browser, Firefox in our case. The below shell script can be used to install the WebDriver for Firefox.

```
wget https://github.com/mozilla/geckodriver/releases/download/v0.19.1/geckodriver-v0.19.1-linux64.tar.gz
tar xvfz geckodriver-v0.19.1-linux64.tar.gz
mv geckodriver ~/.local/bin
```

Once the WebDriver is installed, we can install Selenium package just as installing any other python package.

```
pip install selenium
```

Then in the python code, we can just point to the location of the WebDriver to successfully run selenium and scrape data [145].

```
from selenium import webdriver
browser = webdriver.Chrome(r'<location of the WebDriver>')
```

14.4.4 Natural Language Processing

Natural Language Processing is an integral part of this project as we need to extract the general sentiment of the team's fans from the tweets they have posted. There are various tools available that can be integrated with Python

which can be used to provide the overall sentiment of a sentence. One such tool is TextBlob. It is a python specific library which is built on the existing NLTK but has a much simpler interface to use and is much faster.

```
pip install -U textblob python -m textblob.download_corpora
```

It is adept in performing a lot of NLP tasks such as POS tagging, word tokenization, Noun Phrase extraction, spelling correction and obviously sentiment analysis which we are interested in. Sentiment analysis is the process of extracting the attitude, mood or the emotion of the writer and judging whether the sentence written is overall positive, negative or neutral. The sentiment function gives us the overall sentiment of the sentence in the range [1,-1], where 1 is the positive sentiment and -1 is the negative sentiment.

```
#Example:  
from textblob import TextBlob  
text = 'Big Data is exciting.'  
blob = TextBlob(text)  
print(blob.sentiment.polarity)  
>> 0.15
```

We use TextBlob on all the tweets captured for all teams and create a new feature vector which is the averaged sentiment score of all tweets for a team for a given week [146].

14.4.5 Machine Learning Approaches

Once our data is ready, we needed to feed this data to various machine learning prediction algorithms to predict the outcome of each match in the test dataset. Built-in python package Scikit-Learn was used to run these algorithms.

14.4.5.1 Random Forest Classifier

In Random forests, multiple random decision trees are created using only a small random subset of the features. Samples of the training dataset are taken with replacement, but the trees are constructed in a way that reduces the correlation between individual classifiers. Specifically, rather than greedily choosing the best split point in the construction of the tree, only a random subset of features is considered for each split. Using Random Forests also removes the necessity of pruning trees which would otherwise be required to prevent over-fitting of the

model to the training dataset.

There were few parameters that we had to set to achieve best results which are shown in the snippet below.

```
model_RF = RandomForestClassifier(n_estimators=num_trees, max_features=max_features,max_depth=5)
model_RF.fit(X, Y)
y_pred_RF=model_RF.predict(Xtest)
```

14.4.5.2 Logistic Regression

Logistic regression is a classification algorithm used to assign observations to a discrete set of classes. Unlike linear regression which outputs continuous numeric values, logistic regression transforms its output using the logistic sigmoid function. The function maps any real value into another value between 0 and 1. Since this is a 3-class problem, we need to use the multinomial logistic regression.

14.4.5.3 XGBoost

Boosting is a sequential technique which works on the principle of ensemble. It combines a set of weak learners and delivers improved prediction accuracy. At any instant t , the model outcomes are weighed based on the outcomes of previous instant $t-1$. The outcomes predicted correctly are given a lower weight and the ones miss-classified are weighted higher. This technique is followed for a classification problem while a similar technique is used for regression. XGBoost(eXtreme Gradient Boosting) is an advanced implementation of gradient boosting algorithm. XGBoost is well known because of the in-built cross-validation and regularization feature that helps reduce overfitting. It also deals with missing values in a very effective way.

Below snippet was used to run XGBoost with the parameters that were defined by us:

```
model_xgb = XGBClassifier(max_depth= 3)
model_xgb.fit(X2, Y)
y_pred_XGB = model_xgb.predict(Xtest2)
```

14.5 RESULTS

We ran our three machine learning algorithms with dataset with and without the feature extracted from twitter sentiments. We can see the accuracy of each of the algorithms in the table below.

	With Tweets	Without Tweets
Random Forest	60%	50%
XGboost	40%	40%
Logistic Regression	50%	50%

We can see that there is no definite impact created by adding the feature containing the tweet sentiments. In some of the cases it improves the prediction accuracy but in other cases it doesn't. However, considering all factors Random Forest gives us the best prediction accuracy of 60% with tweets.

14.6 CONCLUSION AND FUTURE WORK

Based on the above results, there is no evident increase in the accuracy by adding the tweet sentiments. This might have been due to the quality of tweets that were extracted. So as part of future work we need to plan to somehow be able to remove unwanted tweets and keep only meaningful ones.

14.7 ACKNOWLEDGEMENT

We would like to thank Professor Gregor von Laszewski for all his support in this project and his material which helped us install MongoDB. We would also like to acknowledge the TAs for their extended support and guidance throughout.

14.8 WORK BREAKDOWN

Sohan and Manek worked on this report equally on this report. The tweet extraction part was divided equally between the two of us on our respective MongoDB localhost. The responsibility of data manipulation and the NLP was

handled by Sohan and Manek completed the machine learning part of the project.

15 DATA ANALYSIS OF YELP REVIEWS

Prajakta Patil, Sahithya Sridhar
patilpr@iu.edu, sahsrid@iu.edu
Indiana University, Bloomington
hid: fa18-523-65, fa18-523-67
github: [https://github.com/patilpr](#)

Keywords: Yelp, Natural language processing (NLP), Sentiment analysis, Scikit-learn

15.1 ABSTRACT

Yelp is quite a useful platform for users to read and upload reviews for businesses such as restaurants. The main objective of our work is to perform analysis on text reviews to find how well they correlate with star ratings given by users to restaurants. It is found that star ratings 1 and 5 are well correlated with textual reviews unlike ratings 2, 3 and 4. Our analysis can help users make better decisions by suggesting them to either look at star ratings or go through actual text reviews before selecting a business for their next visit [147].

15.2 INTRODUCTION

Users visit Yelp website to either look for a restaurant or write a review for one after their visit. Yelp also has reviews and related information for various other businesses as shown later in this project. Yelp had 148 million reviews on its website at the end of 2017. These reviews are in both text and star rating formats. Users also post pictures of food and the restaurant which in turn is used by other users for making decisions. This is also useful for businesses to improve their services. When a restaurant is very popular, it tends to have lot of reviews and in general, a higher star rating. In such cases, people are drawn to choose that restaurant if it fits their choice of cuisine, budget and other parameters they might have in mind. However, when there are multiple restaurants that are similar, it becomes little difficult to pick one. We took a dataset of Yelp reviews

and tried to find correlations between text reviews and star ratings. We performed additional analysis to show various patterns in data. The technologies used to perform this analysis is Natural Language Processing and Sentiment Analysis [147].

15.3 LITERATURE REVIEW

Li et al [148] performed text analysis on Yelp reviews to study if they can be categorized as ***Useful***. This analysis was needed for businesses that didn't have many reviews and hence no reviews were categorized as either ***Useful /funny/cool***. The researchers used SVM and Random Forest models to perform this analysis. They extracted features such as the number of words, sentences, average sentence length, TFIDF, star rating, sentiment of the review as either positive or negative, number of votes received by a review, how long has the user been providing Yelp reviews etc. Using SVM, the researchers were able to achieve an accuracy of 0.67 and 0.69 in the case of Random Forest model for predicting if a review is Useful [148].

Hajas et al [149] tried to understand how external factors such as how changing tastes result in a cyclic behavior for user reviews for restaurants. They took Yelp reviews from 11 college campuses across USA and modeled restaurant ratings as a function of restaurant quality using a second-degree liner differential function. They were able to show a cyclical behavior in a restaurant's rating based on its quality - restaurants that initially started out with good quality dropped their investment in maintaining it which resulted in bad reviews. This in turn forced them to invest again in quality to win back lost business [149].

"We provide heat maps of where most reviewed restaurants are located. The aim of this is to show that most reviewed restaurants are usually clustered together. This is an intuitive result, given that consumers have the option of choosing another (equally good) restaurant if the originally chosen restaurant is crowded [149]".

Koven et al [150] focused on methods to predict useful reviews. Their motivation was that there is lot of work done in trying to find bad/fake reviews. But, it was useful to find genuine and useful reviews. They created various

attributes for Yelp reviews such as reviewer's average star rating, relationship between reviewers, topic and personality analysis, geographic distribution of reviews etc. They used the J48 algorithm and reached an accuracy of 79.8% in predicting if a review is useful or not. They were also able to predict bad reviews with roughly 5% false positives [150].

15.4 DATASET

The dataset used for analysis is taken from Kaggle website. This data contains about 5.2 million user reviews for 174k businesses from 11 metropolitan areas. The data is in CSV and JSON format. It has the following features business_id, name, neighborhood, address, city, state, postal_code, latitude, longitude, stars, review_count, attributes, categories about businesses across 11 metropolitan areas in four countries. It was originally put together for the Yelp Dataset Challenge to perform analysis on Yelp's data [147].

15.5 DATA PROCESSING

The data is processed to get it ready for analysis. Using the NLTK library, we removed the stop-words and the punctuations. All the words were converted to lowercase. We also removed unnecessary symbols and spaces. We didn't remove numbers as it might help users with the prices which is an important factor while choosing a restaurant. All these operations helped reduce the size of the data while retaining useful information [147].

15.6 ANALYSIS METHODS

Following methods are used for analysis to make observations and conclusions from data:

- Linear Regression

Linear Regression is performed to find out if there is any correlation between reviews and ratings (stars). The reviews were characterized into polarity. This is done using the python library called **TextBlob**. The polarity value is between -1 to 1. Polarity above 0 means the text emotion is positive while polarity below 0

means the text emotions are negative. 0 polarity means neutral emotions. We also calculated another parameter called ***Subjectivity***. Subjectivity reflects the user's personal emotions and opinions. Subjectivity is between 0 to 1. A subjectivity of 0 means it's a fact-based opinion and 1 means it's a subjective opinion. We also used this method to find out if there are correlations between reviews and classification of the review i.e. whether it is categorized as ***useful, cool or funny***. As the data we worked with was large, we used a sampling method to use only 10% of all the data for this analysis.

- Logistic Regression, Multinomial Naïve Bayes and Random Forest

These algorithms are used to predict the rating based on text reviews. This will help understand and predict how well the text review and ratings match. Low accuracy means that there is more to the ratings than just the text review. High accuracy means that the ratings accurately capture the sentiments in the text review.

- Count, Term Frequency Inverse Document Frequency (tfidf) vectorizer, and Linear SVC

These help with understanding which words are most commonly used in reviews for a restaurant and help in gaining insight on why certain places might be famous or infamous for.

15.7 RESULTS

Linear regression analysis between polarity and star ratings showed a correlation coefficient of 0.61. This means that emotions in the review and star ratings are moderately correlated. This is shown in Figure [132](#). We can see that the regression line graphically represents this correlation between review text and star rating given by user. One important result is that we got p value less than 0.001 which means that the results are highly significant and that they are very unlikely to have occurred by chance. Review classifications such as useful, cool and funny has negative correlation with review emotions. This is likely because people might have used ironic language while providing a review for the business. All results from the linear regression analysis are shown in Table.1

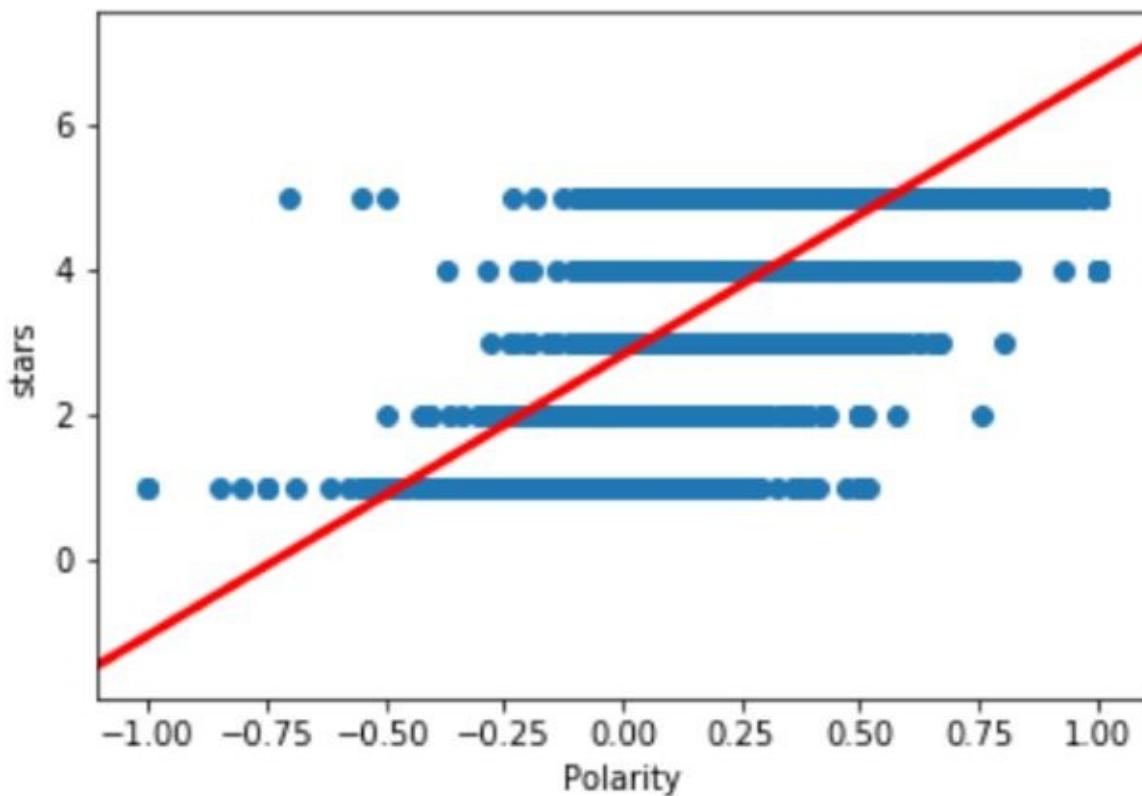


Figure 132: Correlation between Star rating and Polarity

Table.1 Correlation between review sentiment and star rating and review class

Polarity (Review Sentiment)	Correlation coefficient-r value	p value
Star Rating	0.61	< 0.001
Useful	-0.064	< 0.001
Funny	-0.043	3.97
Cool	-0.011	3.66

We have used Logistic Regression, Multinomial Naïve Bayes and Random Forest models to check for accuracy in classifying reviews into star-ratings from 1 to 5. Count vectorizer was used to encode data features in a matrix in which these algorithms can learn using scikit-learn. We trained all the models on the data set and checked their performance for guessing the rank on the test data. We achieved a highest overall precision of 0.55 with Logistic Regression. All 3 models had higher precision when predicting 1 and 5-star ratings and lowest precision when predicting star rating 3 in general.

Table.2 Logistic Regression

Star Rating	Precision	Recall	F1-Score
1	0.78	0.72	0.75
3	0.60	0.43	0.50
5	0.87	0.94	0.90
Average	0.55	0.82	0.81

Table.3 Multinomial Naïve Bayes

Star Rating	Precision	Recall	F1-Score
1	0.71	0.74	0.73
3	0.56	0.34	0.42
5	0.85	0.93	0.89
Average	0.78	0.80	0.78

Table.4 Random Forest

Star Rating	Precision	Recall	F1-Score
1	0.70	0.60	0.65
3	0.46	0.16	0.23
5	0.78	0.94	0.86
Average	0.72	0.75	0.72

We created a confusion matrix to understand these correlations in simpler way. We grouped our data by star rating. Confusion matrix thus created is shown in form of heatmap in Figure 133. We can see that **useful** and **funny** is correlated with text length. **cool** and **subjectivity** are correlated to polarity.

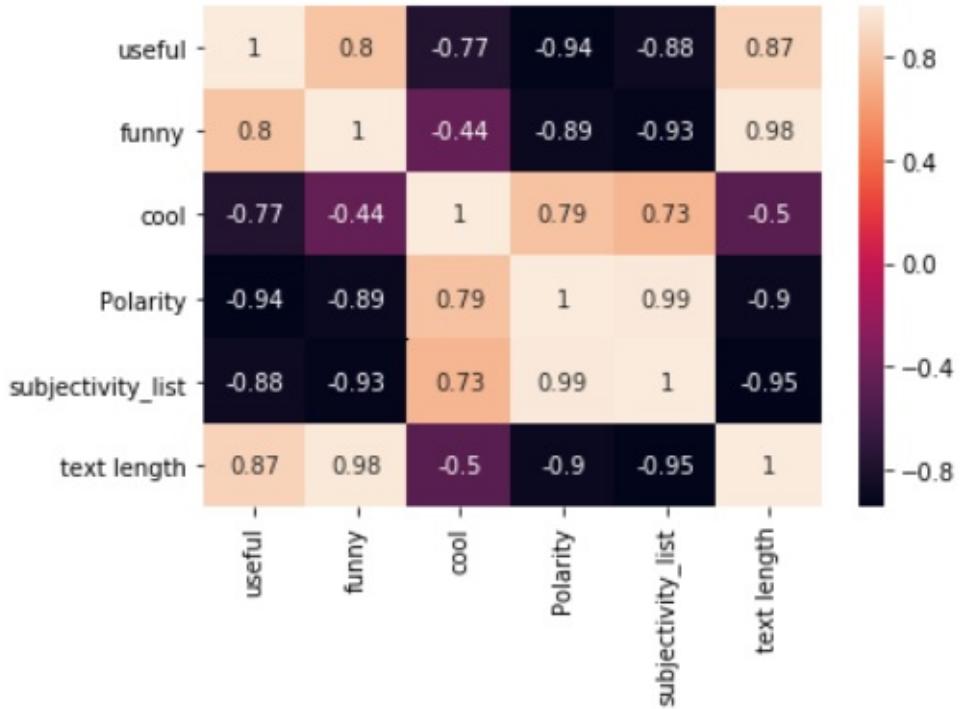


Figure 133: Confusion matrix for predicting emotions

We wanted to understand patterns from this data. First, we looked at reviews by states. Shown in Figure 134, we see Arizona is the state with most reviews for businesses followed by Nevada and California. We then looked at number of reviews by cities. As seen in Figure 135, Las Vegas is the city with most number of reviews with 26810 reviews followed by Phoenix and Toronto. Figure 135 shows only top 10 cities but, our dataset has reviews for 11 metropolitan areas and had reviews for 1093 cities. We also looked at what type of businesses are present on Yelp. In Figure 136 we can see that various types of restaurants receive most reviews followed by shopping and home services. Another interesting observation we wanted to make was which weekday was preferred by users to go out and hence provide reviews for these businesses. Figure 137 shows that most users liked to go out on Saturday followed by Sunday and Friday. Monday and Tuesday saw the lowest number of checkins for businesses.

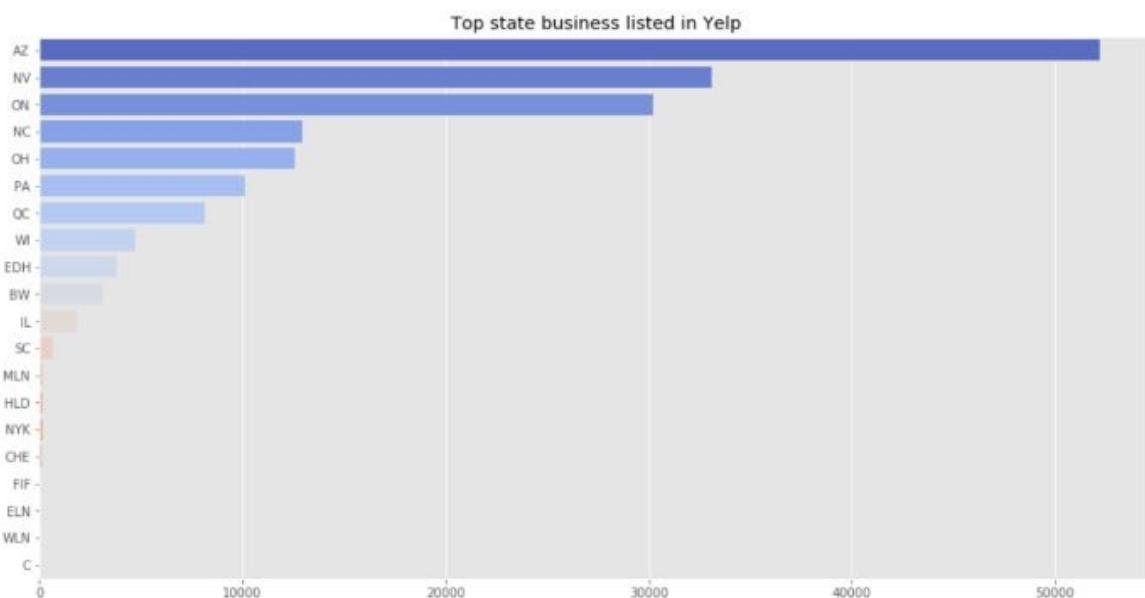


Figure 134: Top states for businesses in Yelp data



Figure 135: Top cities with most user reviews

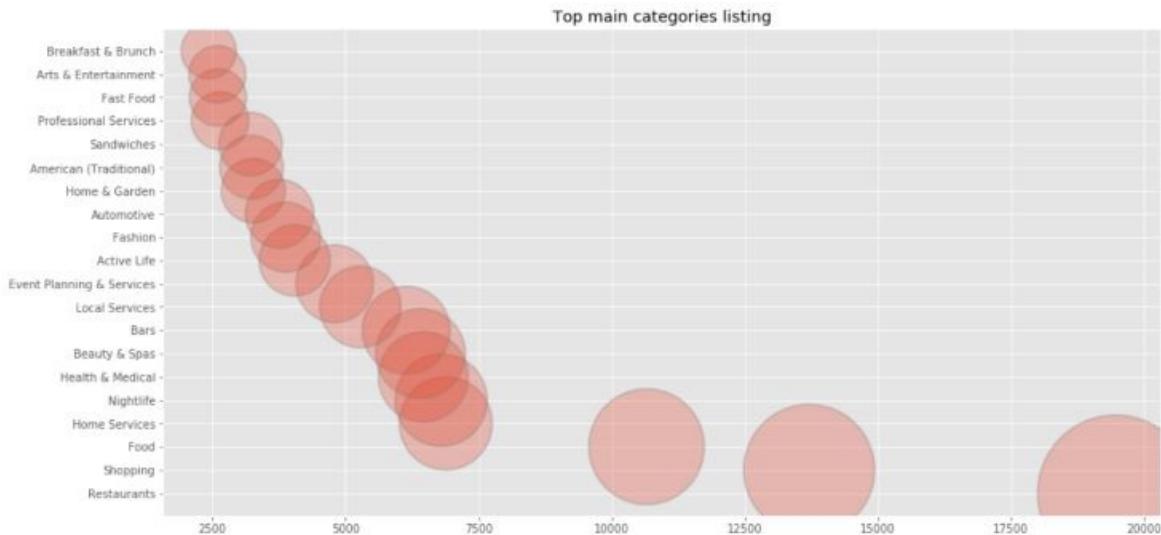


Figure 136: Categories of businesses in Yelp data

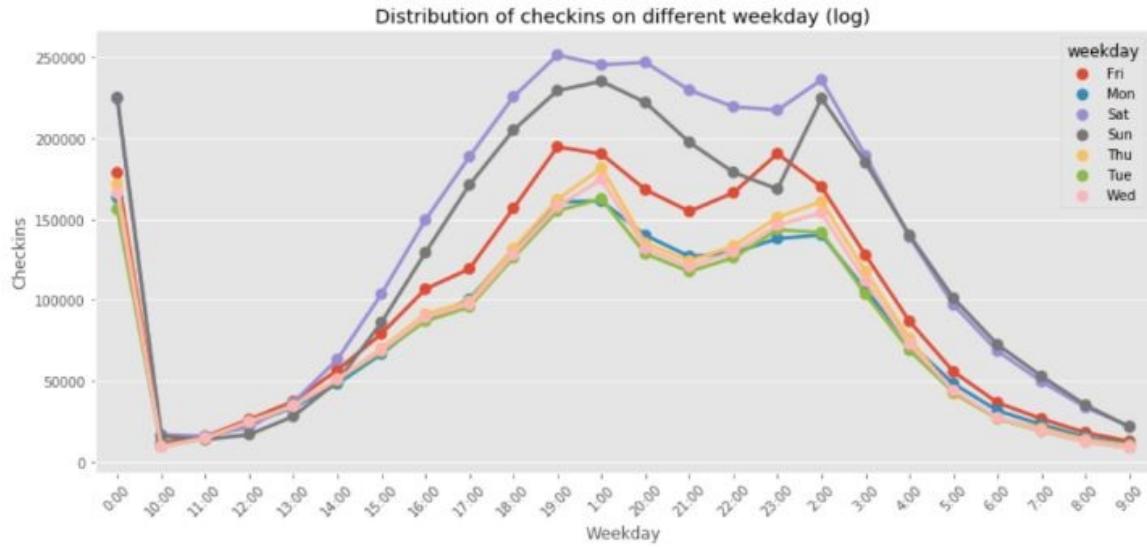


Figure 137: Distribution of Checkins for weekdays

We used pandas ***groupby*** function to calculate mean star ratings for businesses. For purpose of this project we decided to look at details of restaurants only. Figure 138 shows top rated restaurants with descending average descending star rating. We are showing top 10 businesses only as data has more than 1000 places. ***Earl of Sandwich*** was top rated restaurant with average rating of 4.25. Top 10 places are dominated by restaurants that serve pizza and burger. One other thing to notice is that most of these restaurants are franchises. However, regional franchises have higher star ratings than ones that have pan US presence.

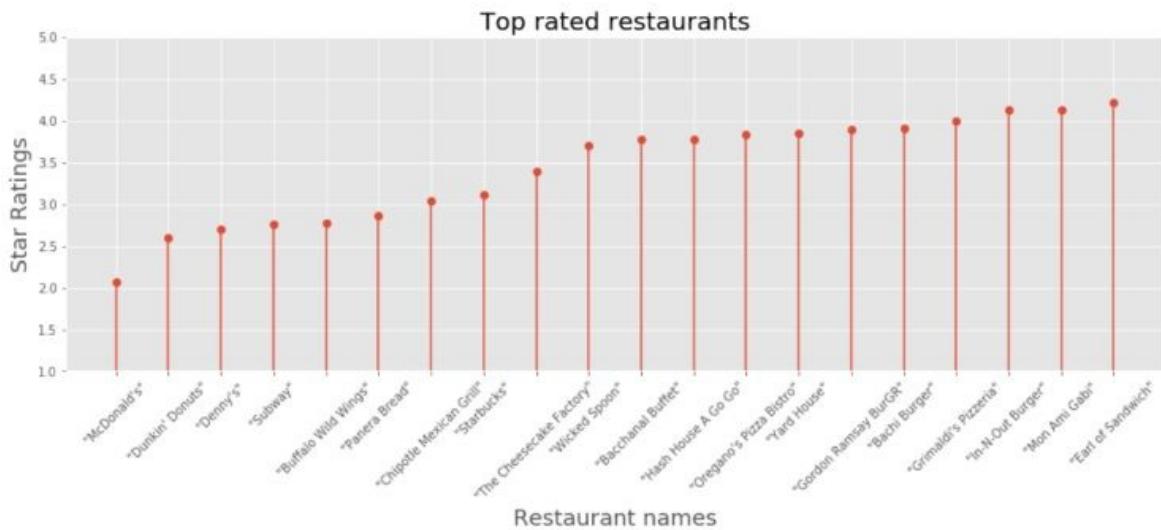


Figure 138: Top rated restaurants on Yelp

Figure 139 shows distribution of ratings offered to businesses. We see that rating 5 was most used rating by reviewers for describing their experience at given restaurant whereas the rating 2 was the least used. Looking at distribution of review length for each star rating shows there was no significant difference in review length distribution. However, comparing Figure 140 with Figure 139 shows same trend. Meaning, more restaurants got 5-star rating but they also got some textual review. We also looked at average number of reviews given by users. We can see in Figure 141 that most users provided less than 5 reviews. There is small percentage of users who have provided more than 30 reviews.

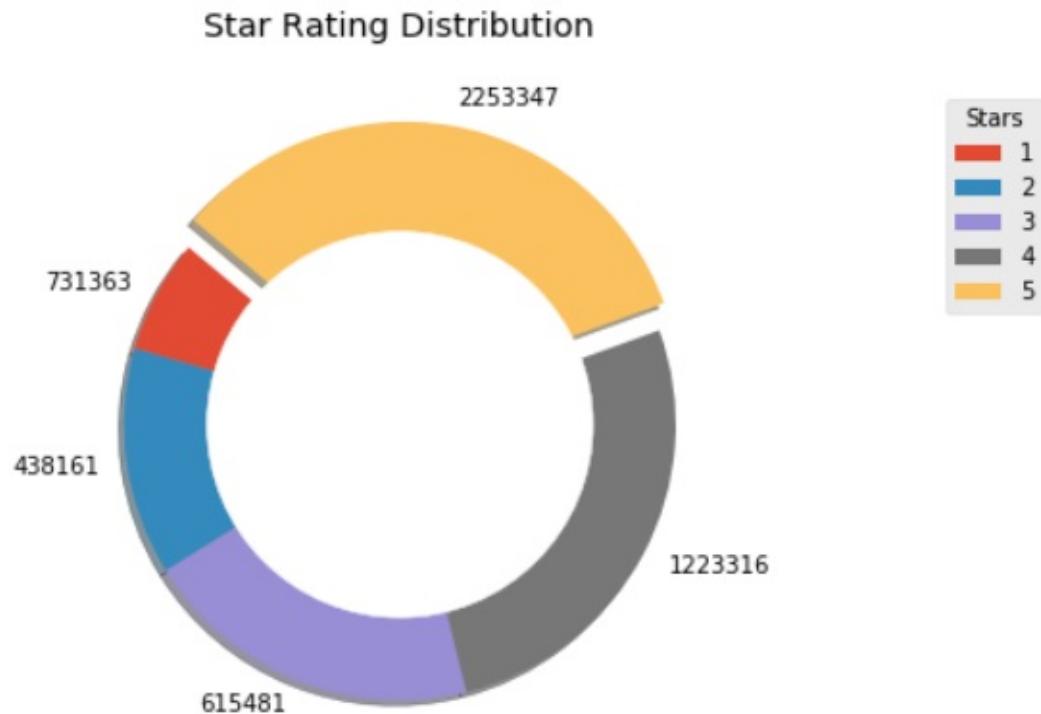


Figure 139: Distribution of Star ratings

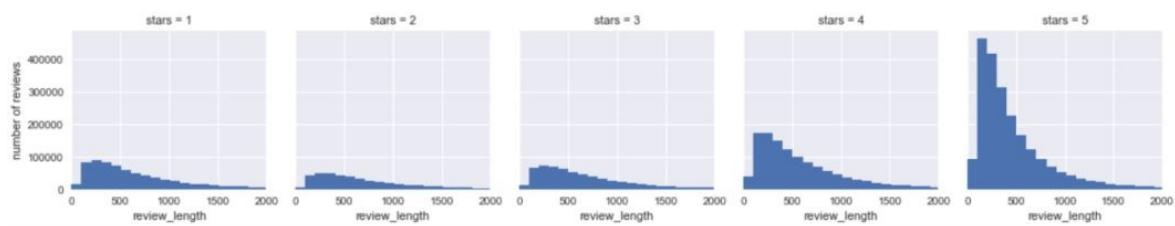


Figure 140: Review length distribution by Star rating

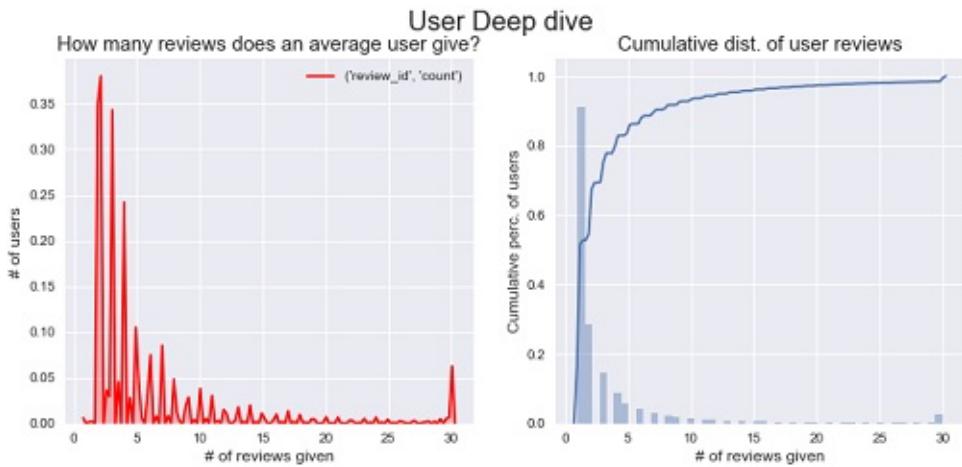


Figure 141: Distribution of number of reviews by users

We were interested in seeing usage of words to describe user experience about a certain place. We chose 11 words that we thought could have been most used in reviews. This list covered words that help express both positive and negative emotions. Figure 142 shows that word *great* was used a lot by users to describe positive experience. Word *bad* was used mostly to describe negative experience.



Figure 142: Most commonly used words for good and bad reviews

15.8 DISCUSSION

We observed a higher accuracy in predicting the star rating of 1 and 5 than all other ratings i.e. 2, 3 and 4. We think that this limitation is a result of understanding user emotions and applying language models to decode it. It is difficult to find how many users write a very objective review and give star ratings accordingly. It is possible that someone has either a better or worse than expected experience and provides a review based purely on emotions at that point of time. Additional analysis is needed to understand how a person from a certain cultural background finds food from other cultures i.e. whether there is a bias in their feedback because they are not used to certain taste.

15.9 CONCLUSIONS

We have been able to show how textual reviews and star ratings are generally moderately correlated. While choosing a restaurant, a user can rely more on star rating 5 to find a good place and a star rating 1 to know that a certain place is bad. If the restaurant has a star rating of 2 or 3, they will have to spend more time in reading the actual textual reviews to make their decision. In general a good business gets a lot more reviews than a bad one. Most users give very few reviews and they do so more in case if their experience was good. Businesses serving pizzas and burgers are one of the most liked places.

15.10 PROJECT MEMBERS AND WORK BREAKDOWN

- Prajakta Patil - fa18-523-65: Introduction, Literature review, Dataset, Data Analysis using Python, Results, Discussion
- Sahithya Sridhar- fa18-523-67: Abstract, Data processing, Analysis methods, Data Analysis using Python, Discussion, Conclusions

16 DO I BUY OR SELL? - USING BIG DATA TO PREDICT STOCK PERFORMANCE

Ritu Susan Sanjay

rssanjay@iu.edu

Indiana University, Bloomington

hid : fa18-523-66

github: [blue icon](#)

code: [blue icon](#)

Keywords

Stock performance prediction, SciPy, Numpy, Pandas, Python, R, Classification, Support Vector Machine, Quandl, Supervised Learning, Dow Jones

16.1 ABSTRACT

Big data is everywhere. Newspapers and magazines are teeming with details on how Company A implemented the perfect business plan or how XYZ pharmaceuticals developed the drug to reverse some previously incurable disease. Better still is the internet with more resources to utilize big data. Numerous copies of books have been published on this topic and Forbes rates big data related jobs as the most desirable. Every single industry today has finally turned its attentions to better understanding how to use data to serve customers better and reap profits. In light of this, there are two industries which have the most to benefit from data science : a) the finance and investment industry and b) the medical industry. This paper focuses on how big data can be used to predict the performance of stocks against the market.

16.2 INTRODUCTION

The financial sector is deemed to be one of the largest producers of data - roughly 2.5 quantillion bytes everyday[151]. However, many reports state how

the investment sector could always make better use of their massive databank. Analytics has been widely adopted in the financial services industry and has helped traders make better decisions and thus see more favorable returns.

“Algorithmic trading now uses a great deal of historical data in conjunction with big data and complex mathematical formulas > to help investors maximize the returns in their investment portfolios” [152].

In years past, traders and banks made decisions based on market trends and calculated risks on each investment. Today, however, they are at an advantage : computers can do the same things, only it is done on a massive scale and is far more accurate. The algorithm developed makes use of the vast amount of resources it has at its disposal. The accuracy of the predictions made by a computer depend on the input data i.e. the closer the data is to the real-time the more accurate its analysis of the current market. In his book, Our Final Invention, author James Barrat mentions that, if humans are ready to tackle AI and develop it to the ultimate Artificial Super Intelligence, then the Wall Street might be the first place to look for such an advancement [153]. Coming back to the point made earlier : the banking and investment sector holds some very promising advancements for data science and artificial intelligence.

“It was not at all what the experts predicted. most of them did not foresee that an economic powerhouse could suffer so much > damage in such a short period of time. They did not expect the fast-growing gross domestic product (GDP) to go so spectacularly into reverse, the real estate bubble to burst as violently as it did, and industrial production and capacity utilization to fall so steeply. Nor did they expect the stock market to plunge so dramatically from its all-time high- although it would recover some ground subsequently....But the real issue is not what has happened, but what happens next” [154].

On this note, it is then necessary to state that the stock prediction can be a very tempting prospect for most data scientists. The financial market has a very short feedback cycle and hence quick validation of predictions is possible. However, it is extremely difficult to develop a model that can provide higher levels of

accuracy in most of its predictions. It is very much possible that the model developed would make predictions for a maximum of one day after which you might have to revamp it all over again [155]. Nonetheless, this paper will attempt to develop a model that predicts the trends for stock prices over time.

16.3 IMPLEMENTATION

16.3.1 Data

Knowledge Discovery of Databases is the sequential process of transforming raw data into actionable insights [156]. The most important step in the KDD lifecycle is: preprocessing the data. A model applied to a dataset is only as good as the data it is applied to. In this regard, most times raw data is available in the most crude formats, with noise, duplicates, missing values or at times even necessitate parsing of large webpages and documents.

Data for this project was derived from Quandl and Yahoo Finance. The stocks considered are those listed on the Dow Jones Index: 3M (MMM), American Express (AXP), Apple (AAPL), Boeing (BA), Caterpillar (CAT), Chevron (CVX), Cisco (CSCO), Coca-Cola (KO), DowDuPont Inc (DWDP), Exxon Mobil (XOM), Goldman Sachs (GS), Home Depot (HD), IBM (IBM), Intel (INTC), Johnson & Johnson (JNJ), JPMorgan Chase (JPM), McDonald's (MCD), Merck (MRK), Microsoft (MSFT), Nike (NKE), Pfizer (PFE), Procter & Gamble (PG), Travelers Companies Inc (TRV), United Health (UNH), United Technologies (UTX), Verizon (VZ), Visa (V), Walmart (WMT), Walgreens Boots Alliance (WBA), Walt Disney (DIS) [157].

The historical for the stocks are available to be downloaded via an API on Quandl. The Quandl platform makes it easy to download financial and alternative data for analysis [158]. A python script can easily download all the historical data for the companies in the Dow index. However, some datasets are only available to premium users; data for both Walgreens Boots Alliance (WBA) and DowDuPont Inc (DWDP) are available for a small fee. The data descriptions for the 30 datasets is as follows [159]:

- Date - date in YYYY-MM-DD format
- Open - opening price of stock in dollars

- High - highest price for day in dollars
- Low - lowest price for day in dollars
- Close - closing price for day in dollars
- Volume - volume of traded stocks
- Dividend - unadjusted dividend on any ex-dividend date else 0.0
- Split - shows any split on a the given day else 1.0
- Adj_Open - adjusted (for splits and dividends) using the CRSP
- Adj_High - adjusted (for splits and dividends) using the CRSP
- Adj_Low - adjusted (for splits and dividends) using the CRSP
- Adj_Close - adjusted (for splits and dividends) using the CRSP
- Adj_Volume - adjusted (for splits and dividends) using the CRSP

The historical data for the DJIA needs to be downloaded manually from yahoo finance. The dataset description for the same is as follows:

- Date - date in YYYY-MM-DD format
- Open - opening price of stock in dollars
- High - highest price for day in dollars
- Low - lowest price for day in dollars
- Close - closing price for day in dollars
- Volume - volume of traded stocks
- Adj_Close - adjusted to include any influential corporate actions and distributions

The final dataset required is the given day's data about the stock. This can be available in one of two ways: parsing the Yahoo Finance website for each stock or by parsing the CNN Business website for a complete summary table for all the stocks. Both methods are implemented via python scripts and the pandas toolkit. To parse the Yahoo Finance website for data, the first step involved is to download static files of the html pages. Then it is a simple case of parsing the HTML documents. The similar process can be followed to download the summary statistics from the CNN website.

Note that it is also possible to download historical data from the U.S. Securities and Exchange Commission (SEC) website if quandl is not preferred [160].

16.3.2 Design

The data is obtained and compiled with python scripts. R code is then used to mine this data. The goal of this project is to figure out whether a stock is underperforming or outperforming. This being a classification problem, the support vector machine algorithm was used.

16.3.2.1 Data Visualization

Figure [143](#) shows the trend changes on the DOW index over the past three decades.

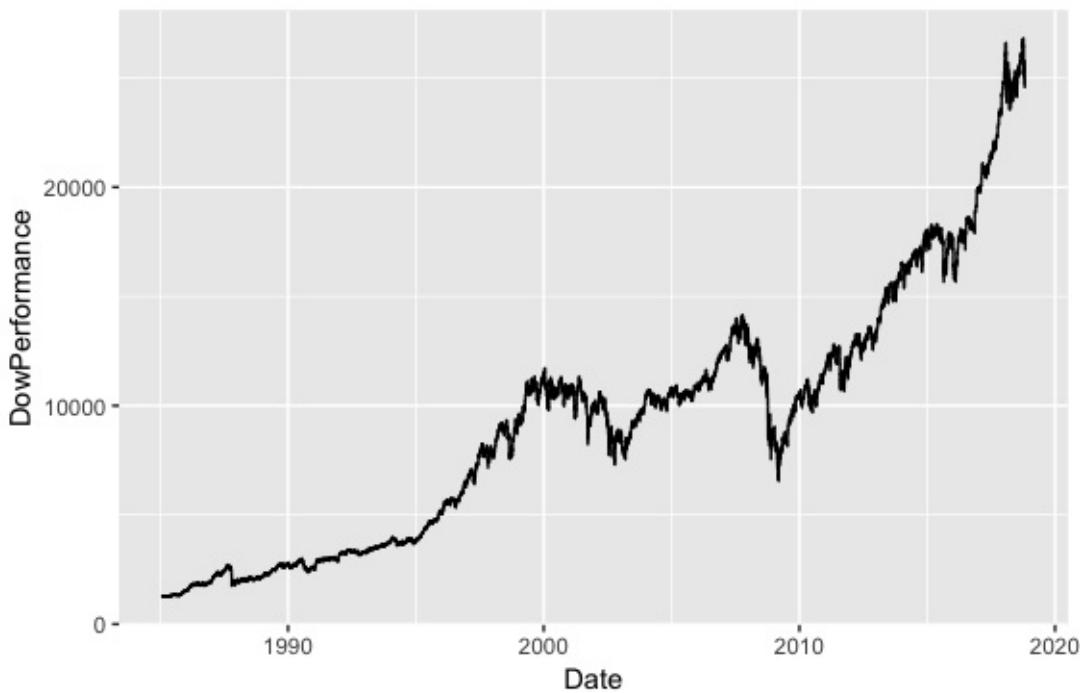


Figure 143: DJI Performance [161]

Figure [144](#) shows the trend changes in the twenty-eight stocks over thirty years.

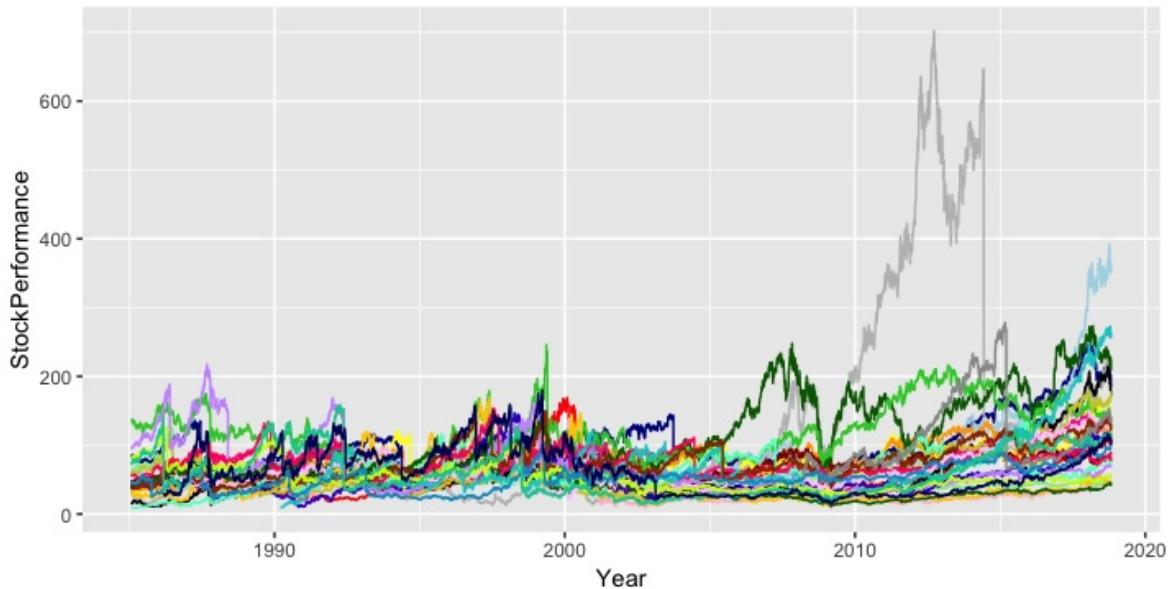


Figure 144: Stock Performance [162]

This project attempts to formulate a model that predicts whether a model underperforms or outperforms the market. Four main factors of a stock are taken into consideration: the Open, High, Low and Close values of any particular day. However, it is unclear whether, it would be beneficial to use the present day's values or the changes in the four factors from the previous day's close. Therefore, two models have been proposed: the first model uses the four factors as they were, while the second model determines the difference from the previous day.

But first it is always advantageous to determine the relationship between the data variables to check if there could be the possibility of another model with greater prediction accuracy.

Figure [145](#) shows the correlation between the open, low, high and close variables. As is obvious, there appears to be a very high correlation between all four variables as would be expected.

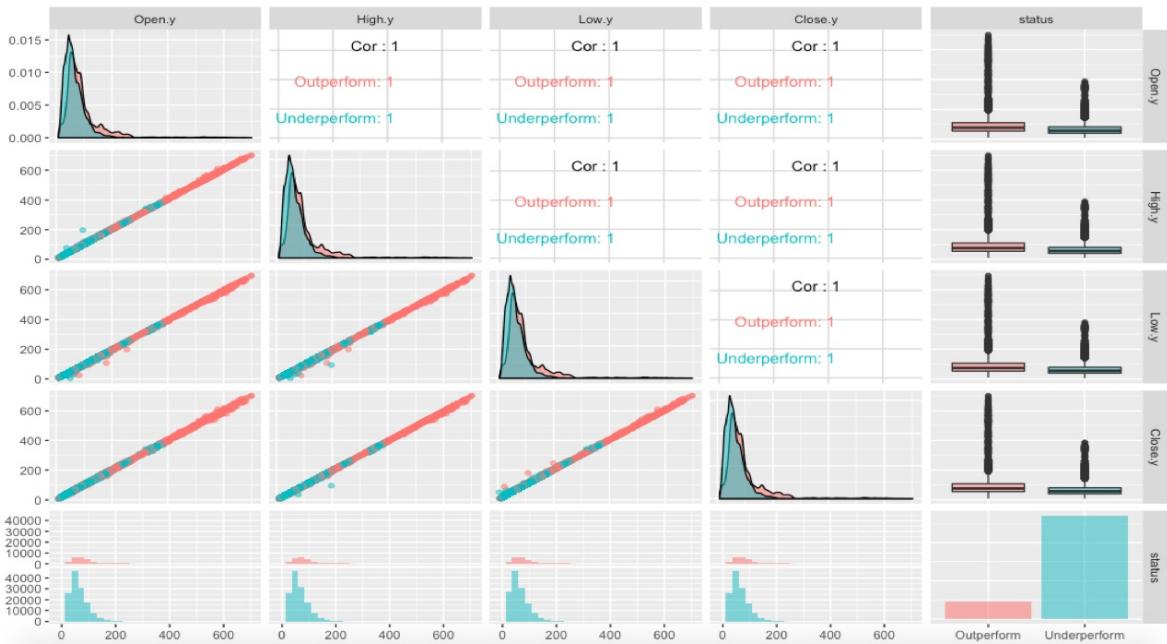


Figure 145: Data Correlation 1 [163]

Figure 146 shows the correlation between the previous day close with the day's open, high and close variables. As in the above case, we observe a high correlation between the features

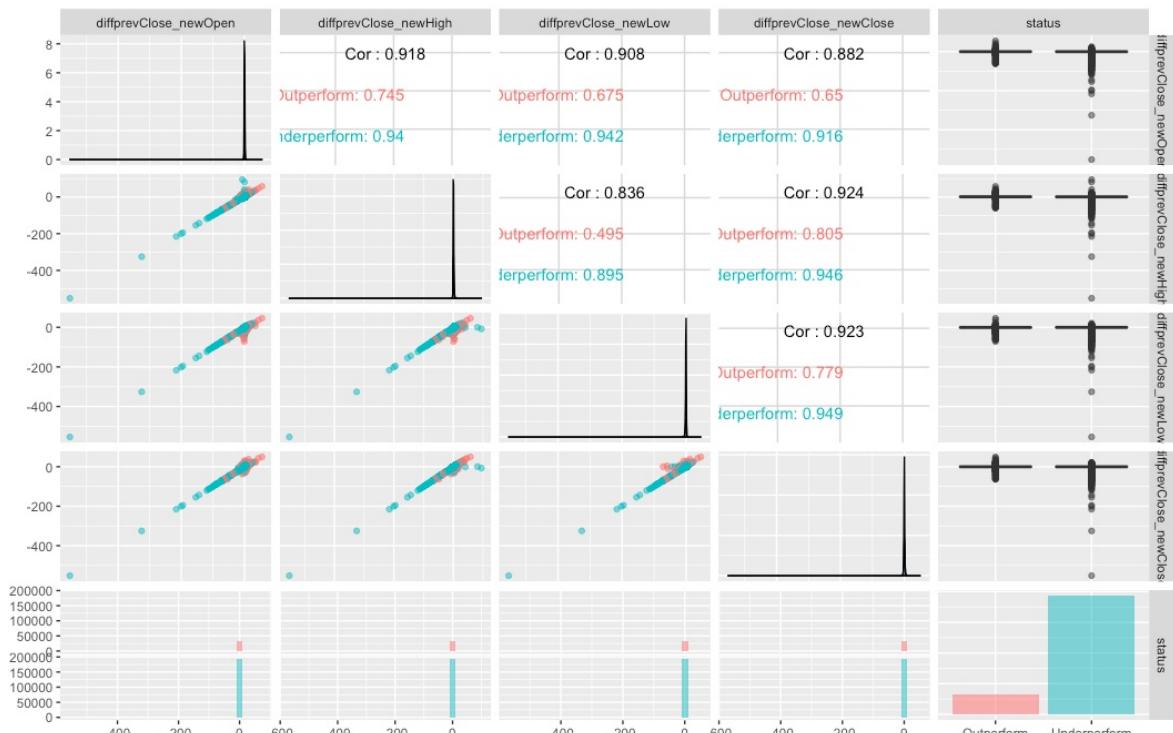


Figure 146: Data Correlation 2 [164]

In order to make sure all possibilities for the said model have been considered, another relationship was considered. Over the net quite a number of traders rave about the OHL technique. Note that this strategy is only to be used for intraday trading and not for long term analysis. However, it would be interesting to determine whether this strategy would provide the desired results. the strategy has two basic rules and only three of the factors from the originally proposed model are used: Open, Low and High. The first step would be to find the difference between the Open and Low (diff1) , and the Open and High (diff2). If diff1 is greater than diff2 then it indicates a downward trend, while if diff2 is greater than diff1 then it indicates an upward trend. As such inorder to visualize the data, we check its correlation matrix [165]. Figure 147 shows the correlation between the day's open and the low and high variables.

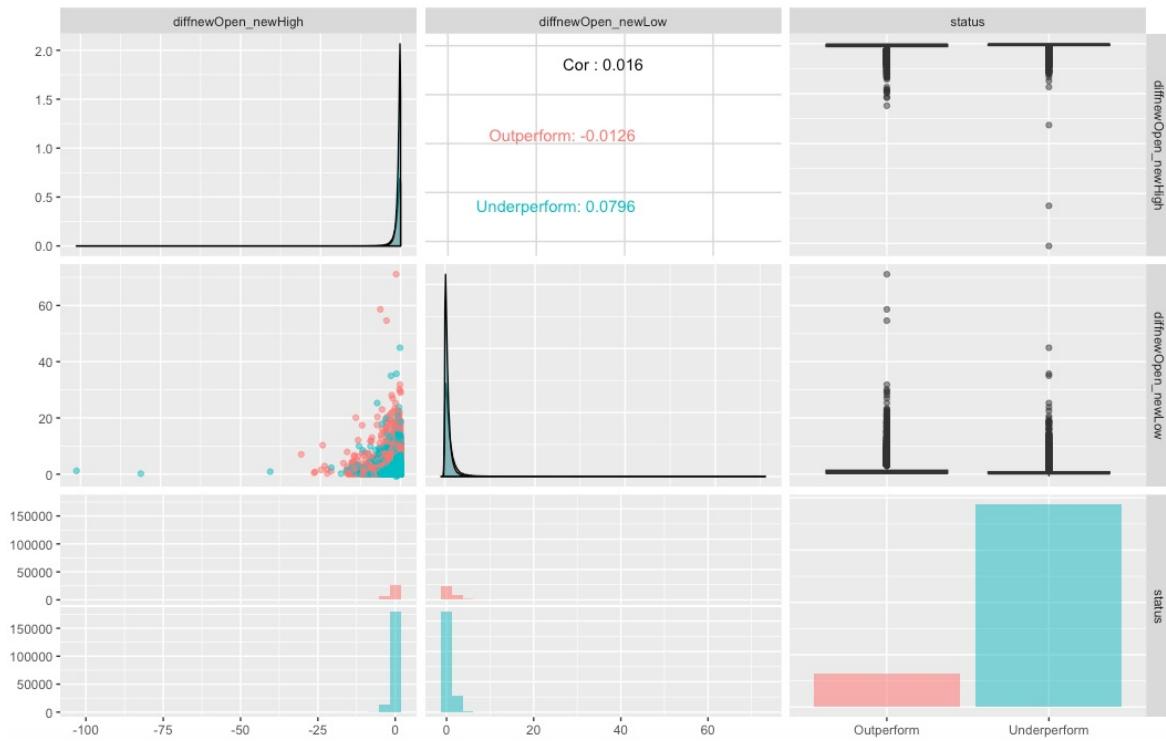


Figure 147: Data Correlation 3 [166]

It is interesting to note that there is little to no correlation between the two variables. Statisticians regularly comment how it is always ideal to choose features which are independent of each other or in other words have no correlation between them. More details about this proposition may only be gauged by building an svm model.

16.3.2.2 Support Vector Machine

“The SVM learning problem can be formulated as a convex optimization problem, in which efficient algorithms are available to find the global minimum of the objective function. SVM performs capacity control by maximizing the margin of the decision boundary. Nevertheless, the user must still provide other parameters such as the type of kernel function to use and the cost function C for introducing each slack variable” [156].

Support Vector Machines are most suitable when there are a large number of features and the number of rows in the training data is quite large. It is usually considered the best model in situations where there is high sparsity or problems where the number of zero values among the features is high. Furthermore, using the kernel trick ensures optimization of algorithms and problems [167].

16.3.2.2.1 Model Fitting

Choosing the right kernel can prove to be tedious, given the nature of the data. Radial kernel was chosen for all three models as it includes the most important feature : being a stationary kernel. The kernel is also smooth, and therefore a visual representation is aesthetic in nature.

16.3.2.2.1.1 Model-1: Using Open,Low, High and Close

Figure [148](#) shows the summary after applying the svm model-2 to the training dataset.

```

> summary(svm_model2)

Call:
svm.default(x = dataset1, y = dataset2, type = "C-classification", kernel = "radial",
            data = subdata_train)

Parameters:
  SVM-Type: C-classification
  SVM-Kernel: radial
    cost: 1
   gamma: 0.25

Number of Support Vectors: 44471
( 22914 21557 )

Number of Classes: 2

Levels:
  Outperform Underperform

```

Figure 148: SVM Model2 Summary [168]

Figure [149](#) is the confusion matrix generated by the code for the svm model-2.

```

> pred2 <- predict(svm_model2,dataset1)
> system.time(pred2 <- predict(svm_model2,dataset1))
  user  system elapsed
206.742  1.080 208.119
> table(pred2,dataset2)
  dataset2
pred2      Outperform Underperform
  Outperform        1433         326
  Underperform     21074      135043

```

Figure 149: SVM Model2 CM [169]

Figure [150](#) shows the confusion matrix from applying the model to the test data.

```

> test_pred1 <- predict(svm_model2,type='response',newdata = dataset3)
> table(test_pred1,dataset4)
  dataset4
test_pred1      Outperform Underperform
  Outperform        639          129
  Underperform      9055        57838
>

```

Figure 150: SVM Model2 Test Pred [170]

16.3.2.2.1.2 Model-2: Using difference between previous day Close and present day Open,Low, High and Close values

Figure [151](#) shows the summary after applying the svm model-3 to the training dataset.

```
> summary(svm_model3)

Call:
svm.default(x = dataset5, y = dataset6, type = "C-classification", kernel = "radial",
             data = subdata2_train)

Parameters:
  SVM-Type: C-classification
  SVM-Kernel: radial
    cost: 1
   gamma: 0.25

Number of Support Vectors: 47297
( 24974 22323 )

Number of Classes: 2

Levels:
  Outperform Underperform
```

Figure 151: SVM Model3 Summary [171]

Figure [152](#) is the confusion matrix generated by the code for the svm model-3.

```
> pred3 <- predict(svm_model3,dataset5)
> system.time(pred3 <- predict(svm_model3,dataset5))
  user  system elapsed
186.539  0.914 187.551
> table(pred3,dataset6)
  dataset6
pred3      Outperform Underperform
  Outperform        532        216
  Underperform     21927     135200
```

Figure 152: SVM Model3 CM [172]

Figure [153](#) shows the confusion matrix from applying the model to the test data.

```

> table(test_pred3,dataset8)
      dataset8
test_pred3    Outperform Underperform
  Outperform        208        115
  Underperform     9534      57804

```

Figure 153: SVM Model3 Test Pred [173]

16.3.2.2.1.3 Model-3: Using OHL strategy

Figure 154 shows the summary after applying the svm model-4 to the training dataset.

```

> summary(svm_model4)

Call:
svm.default(x = dataset9, y = dataset10, type = "C-classification",
  kernel = "radial", data = subdata3_train)

Parameters:
  SVM-Type: C-classification
  SVM-Kernel: radial
  cost: 1
  gamma: 0.5

Number of Support Vectors: 47067
( 24619 22448 )

Number of Classes: 2

Levels:
  Outperform Underperform

```

Figure 154: SVM Model4 Summary [174]

Figure 155 is the confusion matrix generated by the code for the svm model-4.

```

> pred4 <- predict(svm_model4,dataset9)
> system.time(pred4 <- predict(svm_model4,dataset9))
  user  system elapsed
142.235  0.515 142.761
> table(pred4,dataset10)
      dataset10
pred4    Outperform Underperform
  Outperform        657        371
  Underperform     21949      134899

```

Figure 155: SVM Model4 CM [175]

Figure [156](#) shows the confusion matrix from applying the model to the test data.

```
> test_pred4 <- predict(svm_model4,type='response',newdata = dataset11)
> table(test_pred4,dataset12)
    dataset12
test_pred4      Outperform Underperform
  Outperform        244         144
  Underperform     9351       57922
```

Figure 156: SVM Model4 Test Pred [176]

16.3.2.2.1.4 Model Comparision

Figure [157](#) shows the model performance rates for both models.

		Performance Rates						
		Accuracy Rate	Error Rate	True Positive Rate	False Positive Rate	Specificity	Precision	Prevalence
Model-2	Training	0.8645	0.1355	0.8650	0.1853	0.8147	0.9976	0.9889
	Testing	0.8643	0.1357	0.8646	0.1680	0.8320	0.9978	0.9886
Model-3	Training	0.8597	0.1403	0.8605	0.2888	0.7112	0.9984	0.9953
	Testing	0.8574	0.1426	0.8584	0.3560	0.6440	0.9980	0.9952
Model-4	Training	0.8586	0.1414	0.8601	0.3609	0.6391	0.9973	0.9935
	Testing	0.8597	0.1403	0.8610	0.3711	0.6289	0.9975	0.9943

Figure 157: SVM Model Performance [177]

In order to evaluate the better model from among the three, the performance features (namely, accuracy rate, error rate, true positive rate, false positive rate, specificity, precision and prevalence) have been calculated. Interestingly, the difference in the rates of all three models is quite negligible. However, considering how the Open, Low , High and Close variables are frequently used for predictions, then Model-2 would be the best possible fit for the data.

16.4 RESULTS

Figure [158](#) shows the results for the prediction on recent data.

```

> print(results)
      prediction
1 "AAPL"    "2"
2 "MMM"     "2"
3 "AXP"     "2"
4 "BA"      "2"
5 "CAT"     "2"
6 "CVX"     "2"
7 "CSCO"    "2"
8 "KO"      "2"
9 "XOM"     "2"
10 "GS"     "2"
11 "HD"      "2"
12 "IBM"    "2"
13 "INTC"    "2"
14 "JNJ"    "2"
15 "JPM"    "2"
16 "MCD"    "2"
17 "MRK"    "2"
18 "MSFT"   "2"
19 "NKE"    "2"
20 "PFE"    "2"
21 "PG"     "2"
22 "TRV"    "2"
23 "UNH"    "1"
24 "UTX"    "2"
25 "VZ"     "2"
26 "V"      "2"
27 "WMT"    "2"
28 "DIS"    "2"

```

Figure 158: SVM Model Results [178]

Note that the the values ‘2’ and ‘1’ have been used to represent ‘Underperform’ and ‘Outperform’ respectively. The results shown above indicate that out of all the twenty-eight stocks in the list, only model predicts that only United Health Group (UNH) is expected to outperform the market. Checking the news coverage and media for UNH performance yields that it has indeed been outperforming the market.

The reason for there being a difference in predictions between the real-world and the project model may be attribute dthe various features that have not been considered for the model. Features such as total debt-to-equity ratio, market cap, volume etc. would provide better prediction. Also, many internal financial factors have not been considered like the financial reports realeased by each company. Thus, this model proves that it can be quite challenging (but quite possible) to create a model which predicts the performance of the companies in

accordance with the market changes. Nevertheless, this could prove to be a most promising project for future work.

16.5 CONCLUSION

The stock market is non-linear in nature, owing to which it is quite challenging to predict a trend with great accuracy [179]. Nevertheless, it has attracted one too many investors and analysts in attempting to predict its performance.

This report presents the study in using data mining algorithms (Support Vector Machines) to predict whether a stock underperforms or outperforms the market. The imperfection in the results obtained via this project can be attributed to the change in political changes, current economical shifts and lastly the ever-changing investor prospects.

For future work, there are quite a number of improvements to be made. For example, kafka can be used to stream real-time data from websites. Also, it would be interesting to implement neural networks to improve the efficiency of the prediction. Text mining social media feeds could be implemented to bring aspects of changes in economy, politics and current trends to the model.

16.6 ACKNOWLEDGEMENTS

The author would like to thank professor Dr. Geoffrey C. Fox, Dr. Gregor von Laszewski for providing the opportunity to embark on this project. This project would not be possible without the guidance and inputs from the entire piazza team.

17 LOAN DEFaulTER ANALYSIS

Uma Kota, Jatinkumar Bhutka
umabkota@iu.edu, jdbhutka@iu.edu
Indiana University Bloomington
hid: fa18-523-71 fa18-523-59
github: [blue link](#)
code: [blue link](#)

Learning Objectives

- Run different Machine learning Algorithms to classify the loan defaulters and find the most optimal model
 - Learn how to use the Microsoft Azure platform
-
-

Keywords: fa18-523-71, fa18-523-59, Exploratory Data Analysis, Python, Jupyter Notebook, Microsoft Azure, Classification, Credit, K-NN, Random Forest, PCA, XGBOOST.

17.1 ABSTRACT

Classifying loan defaulters from potential clients is one of the most important problems in banking sector today. Especially, when there is little or no data regarding the customers' credit history, it's difficult to predict whether the client will be able to repay their loan amount. In 2018, Home Credit , an international non-bank, consumer finance group released a variety of data in a kaggle competition to urge the datascience enthusiasts all over the world to come up with new models and predictions. If an optimal solution is achieved, it would help us understand the clients potential of repaying the loan and also help many potential clients get their loans sanctioned . To classify the defaulters, we have tried various approaches including PCA and Random Forests for feature selection and Random Forest, k-nn and XGBOOST for modelling with XGBOOST giving us the best results. To build the models, Jupyter Notebook

IDE with a python 3.7 kernel was used on the cloud computing service Microsoft Azure [180].

17.2 INTRODUCTION

Defaulting is a case of inability of a debtor to meet the legal obligation of debt repayment on a loan or security when due just as in the case of a business where it fails to make coupon payments to his bondholders of issuing bonds. It has become increasingly important these days to manage the credit risk of the borrowers and hence see the revision of loan policies these days by many banking and financial institutions. To assess a borrower's credit risk, monitor borrower's behaviour are some of the key focus of interests in collecting the statistical data on the consumer's behaviour is concerned. This study, specifically helps the financial institutions examine the borrower's credit payment performance with respect to some of the variables. Banking and Financial institutions come up with some exogenous acceptance rule wih a view to investigate how a customer's payment performance will be, therby, ranking the potential customers according to the predicted default possibilities. The Related Research section gives a brief review of the ongoing and existing relevant research followed by models and predictions used to identify the potential clients of their ability to repay the loan.

17.3 RELATED WORK

One of the main studies on ameliorating the acceptance and the rejection criterion of the potential customers is 'Investigates lenders and borrowers back in 1990 pertaining to the activities of consumer rationing for the United States' credit market. His investigation suggested of the rejections in loans due to the credit history, their age, income and the amount of collateral offered by the borrower to secure loan in case of delinquency. In due course of time, Crook (1996) replicated Jappelli's (1990) by saying more education for a household head would allow the potential borrower to be more capable in forecasting his loan repaying capability, thereby, leveraging the lender's thoughts on his decisions. With a view to measure the risk of portfolio for a sample loan, Roszbach K. and Jacobson T. (1998) built a statistical model with a view to provide a demo on the evaluation of the alternate lending policies and concluded

that the decision on credit grant is not affected by the income. A little different to the above studies, the recent studies examine the effects of different variables on the customer's payment performance characteristics [181].

17.4 DATA

Home Credit released seven tables of data from different sources for this competition, but this project is confined to the main dataset which has features attributing to the loan, loan applicant and time of the loan.

The main dataset has 307511 rows with each row representing individual loan application and 122 features (variables) including the TARGET which indicates if the loan is repaid (indicated by a 0) or not repaid (indicated by 1). For our project, we have taken the first 150000 rows for train and the next 30000 rows as the test dataset.

17.5 TECHNOLOGIES USED

- Python, being a high-level object-oriented programming language, we have extensively coded in Python 3.0 as it is used for general purpose programming [182].
 - Matplotlib: Matplotlib is one of the most widely used visualization libraries in python that provides 2D plotting library to generate plots, enabling quality figures in interactive environments across diverse platforms.
 - Seaborn: Seaborn is an extensively used data visualization package available in python that is built on top of matplotlib. It provides an interface to draw appealing graphs and draw enlightening factual illustrations.
 - Sklearn: Scikit-learn is one of the leading machine learning library featuring different classification, regression and clustering algorithms. Main purpose of using scikit-learn was to conduct random-forests, gradient boosting and k-means.
- Data Science Virtual machine on Azure
 - The Microsoft Data science virtual machine is windows Azure Virtual machine. VM has preinstalled tools for data analytics

- and machine learning. You can create your own VM [183].
- The entire data is being loaded in azure cloud and can be accessed from the cloud directly [@ www-microsoftazure-freeaccount].
 - **Jupyter Notebook** Azure notebook on azure cloud services supports Jupyter Notebook that allows you to interact in many ways. It supports python 2 and 3 [184].
 - XGBoost, Preinstalled machine learning tool on VM for fast and accurate boosted tree implementation [183].

17.6 EXPLORATORY DATA ANALYSIS

The dataset with high dimensionality of 122 features required a series of univariate and bivariate analyses to be conducted on it for us to understand the features and their relationship with each other.

- To begin with, we had to find the types of data the dataset comprises. We found out that of the 122 columns the dataset is made of 106 numeric and the rest of the 16 columns are categorical. Also, of the 106 numeric variables, 32 of them were Boolean.
- Number and percentage of missing values for each column were found to conduct missing value treatment on the dataset. 67 of the 122 columns had one or more missing values.
- The distribution of labels for the dependent variable, TARGET, was seen by plotting a countplot of the variable. It can be seen in the figure that the instances of repaid and not repaid are balanced. The instances where the customer has not repaid are very sparse making this problem a case of rare event modeling.

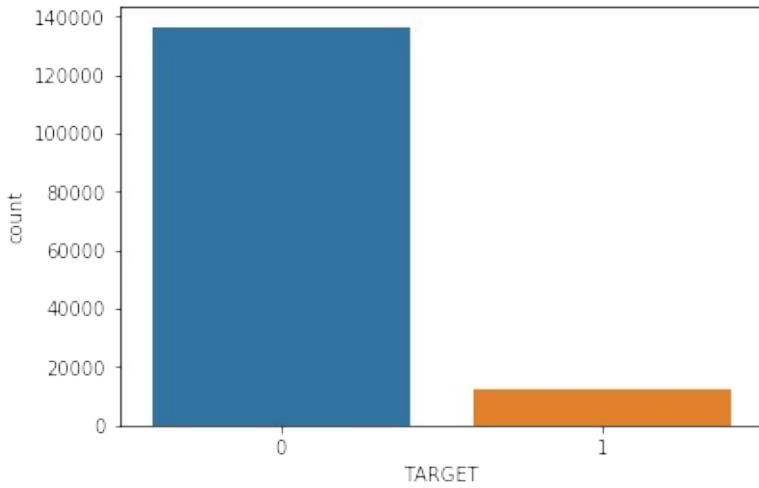


Figure 159: Label Distribution of Target

- Distributions of the intuitively important variables were also plotted to check for outliers and skewedness and it was found that many variables exhibited skewed distributions with outliers. In the figure Figure 160, the histogram plot, shows the distribution of AMT_INCOME_TOTAL which captures total income of the applicant. In the figure Figure 161, the barplot shows the range of the feature and the outliers that lie outside the IQR (Inter Quartile Range)

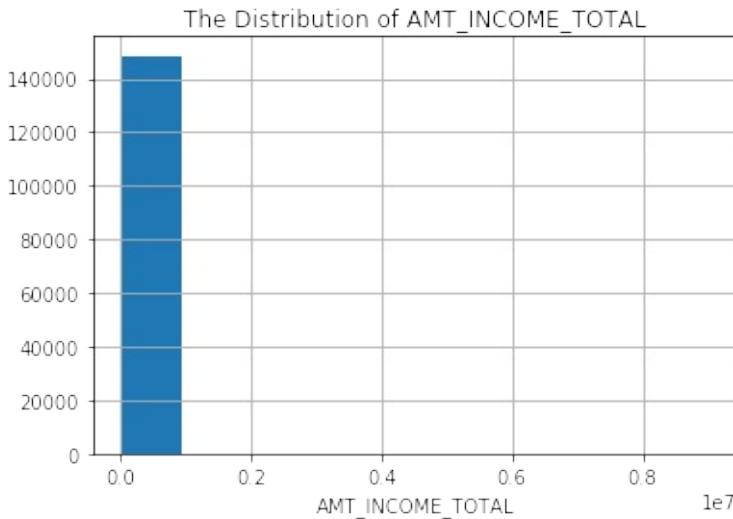


Figure 160: Distribution of AMT_INCOME_TOTAL

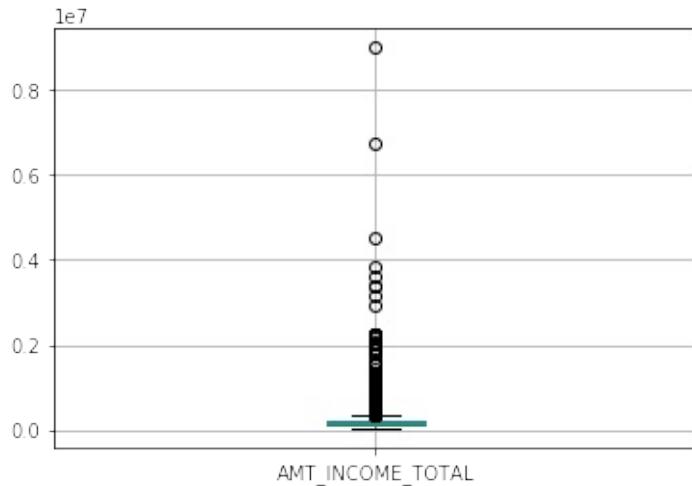


Figure 161: Outliers in `AMT_INCOME_TOTAL`

Distribution of the categorical variable, `OCCUPATION_TYPE`, can be seen in Figure [162](#)

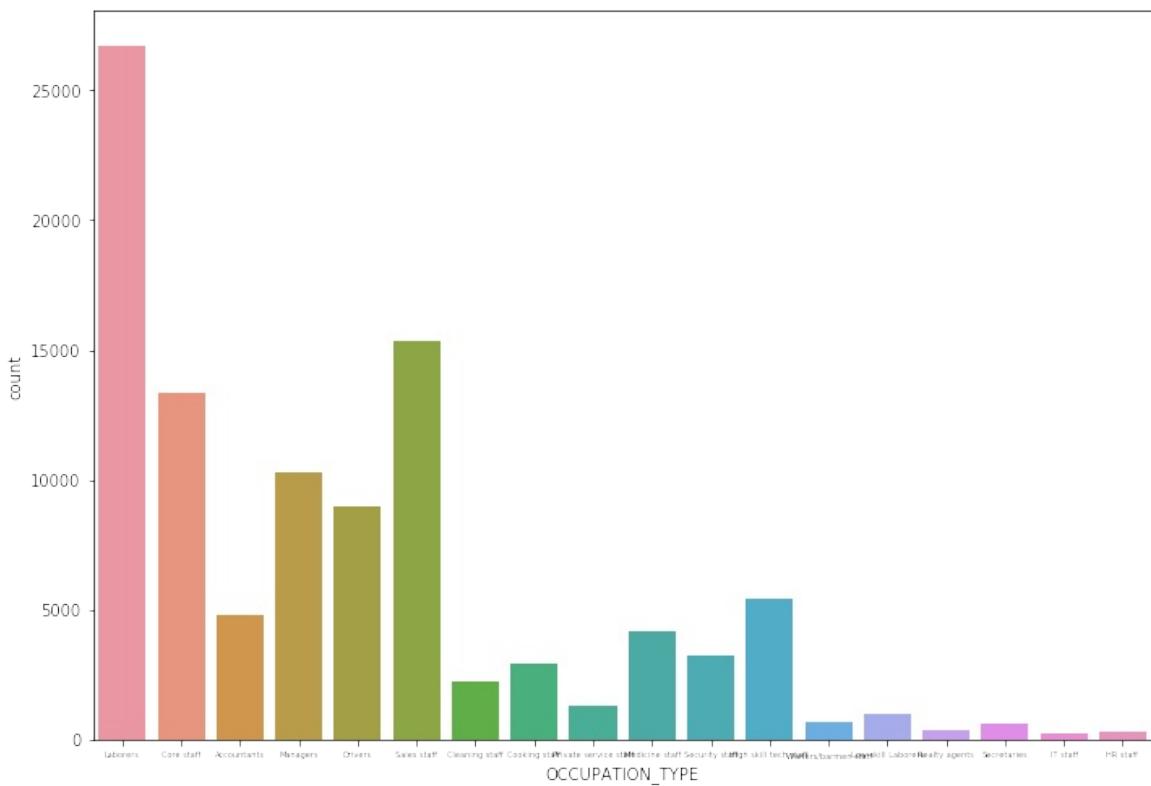


Figure 162: Distribution of `OCCUPATION_TYPE`

- Various bivariate analyses were also conducted to understand the relationship between the dependent and independent variables and

relationships in between independent variables and to make sure no multi-collinearity exists in the dataset before we conduct statistical modeling. Figure [163](#) shows the variance of the EXT_SOURCE3 variable with respect to the dependent variable and Figure [164](#) shows the same for the variable OCCUPATION_TYPE

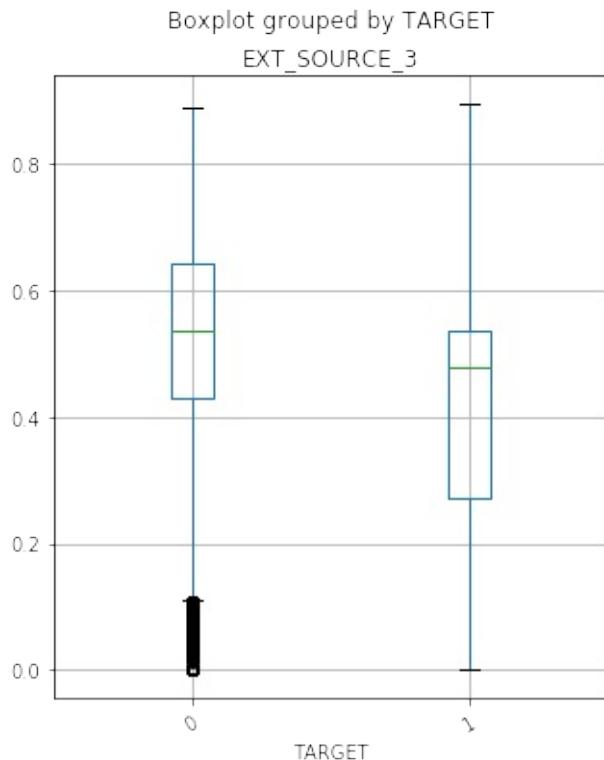


Figure 163: Relation between EXT_SOURCE3 and TARGET

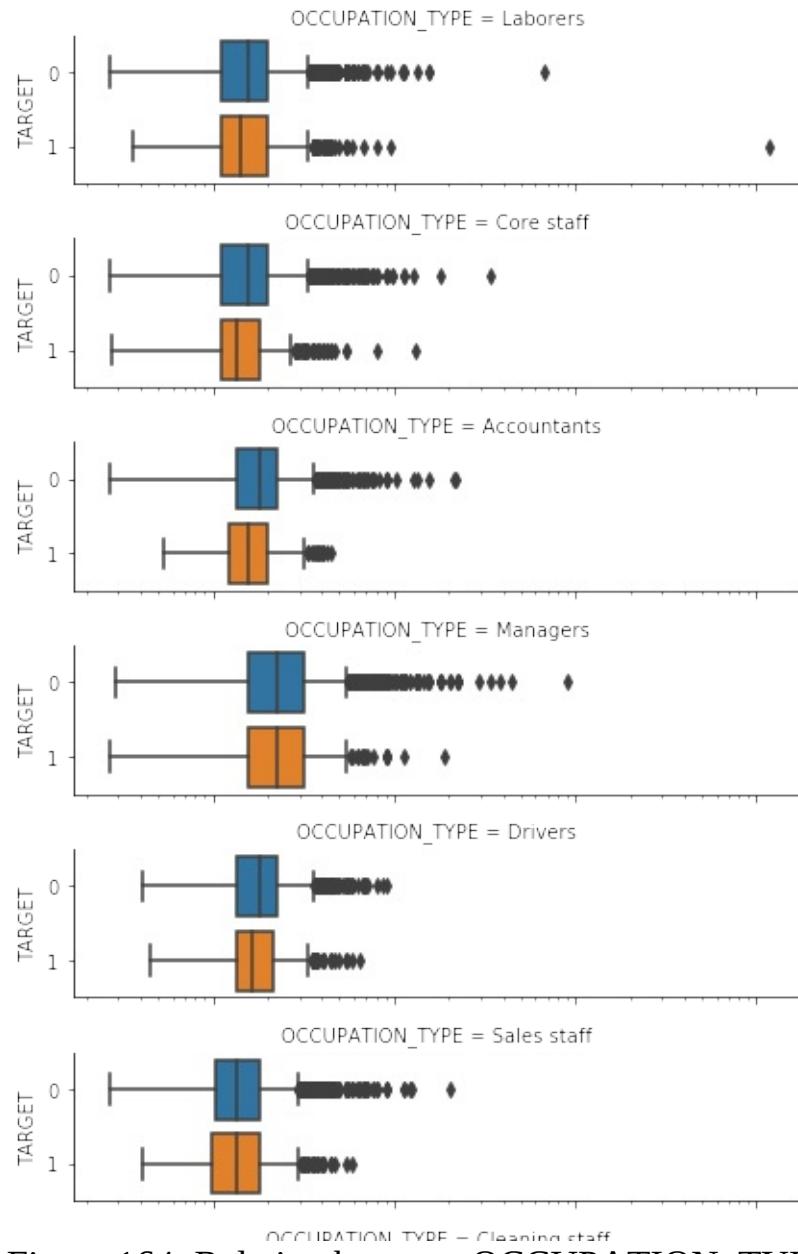


Figure 164: Relation between OCCUPATION_TYPE and TARGET

Figure [165](#) shows the relationship between gender, occupation type and income

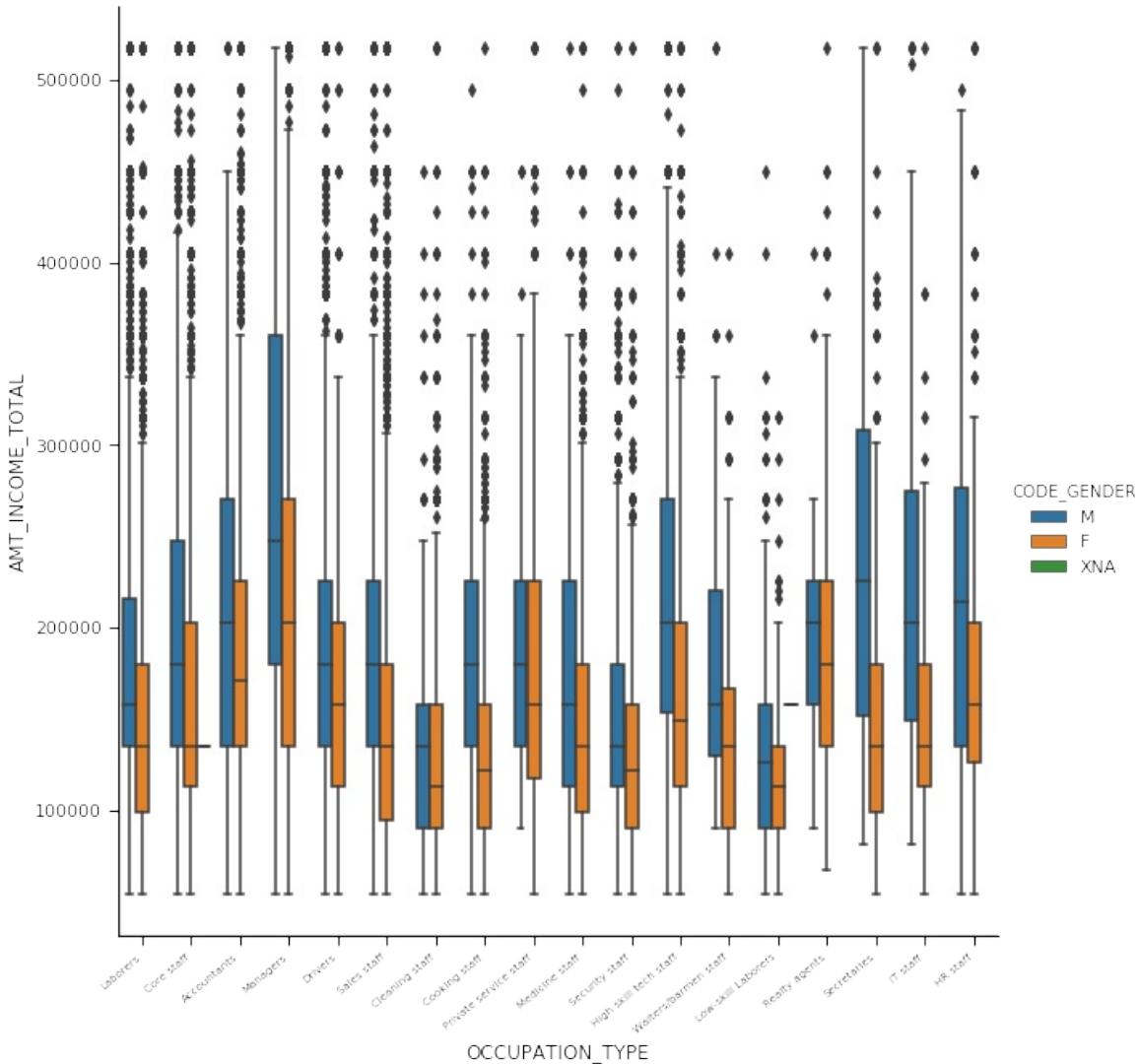


Figure 165: Relation between OCCUPATION_TYPE, AMT_INCOME_TOTAL and CODE_GENDER

- Correlations between the variables were found before heading to feature selection.

17.7 FEATURE PREPROCESSING

- Missing value treatment:
 - During the exploratory data analysis, it was found that 67 variables had one or more missing values.
 - For variables with less than one percent missing values, instances with missing values were removed from the dataset.

- For variables with missing values between one and twenty percent, their missing values were replaced by the median value.
 - Apart from EXT_SOURCE1, variables with more than thirty percent missing values were removed from the dataset as they didn't seem intuitive and replacing such high values of data would only skew the actual values.
 - While modeling for one of the XG-BOOST models, all the variables were considered irrespective of percentage of missing values as the model accepts missing values.
- Outlier treatment on selected numerical variables:
 - Outliers were found in most of the numerical columns and they may affect the model results
 - For selected columns, Winsorization was conducted. It's a clipping approach to make all the data stay between the first and ninety-ninth percentile [185].
- One hot encoding:
 - As most machine learning algorithms cannot apprehend categorical variables, they have to be mapped to the number space.
 - One way of doing this is by representing the categorical variables as series of binary vectors.
 - Each vector represents a label, where 1 indicates its presence and 0 its absence [186].
 - This was achieved by using pandas.get_dummies on all the categorical variables
- Removing multi-collinearity:
 - The independent variables with a Pearson correlation of more than 0.9 were removed to curb redundancy in model.
- Scaling the data:
 - Using the Scikit-learn sklearn.preprocessing.MinMaxScaler module, the values were scaled such that they are all in between 0 and 1 [185].

17.8 FEATURE ENGINEERING

- New Features:

- 3 new features, income_fm, days_emp and ann_nr , were added to the list of features. They were calculated using the existing features.
- Principal Component Analysis:
 - PCA is a procedure that changes potentially correlated variables into a smaller set of independent principal components.
 - The variability of dependent variable is mostly explained by the first component and as we proceed further, the variability explained decreases [187].
 - PCA was applied on the ten most intuitive features from the whole set.
 - The top 6 PCs were seen to explain most of the variability
- Random Forest
 - A Random forest classifier was trained on the train data and the importance of each variable was found using the model.
 - Taking the importance scores into consideration, top 15 features were selected to make a model on

17.9 MODELING TECHNIQUES

17.9.1 Random Forest

Random Forests is an ensemble technique that fits several decision tree classifiers on different subsets of dataset. Each tree votes for a specific class and the class with most number of votes is chosen, in this way it gives a better prediction accuracy than a decision tree while simultaneously controlling overfitting [188], [189].

17.9.2 K nn

KNN classifies a point based on feature similarity, i.e., by considering the classes k points around it belong to, It is a called a lazy learning algorithm as it requires no prior knowledge about the distribution of data and uses all the training data while classifying the test data [190] .

17.9.3 XGBOOST

XGBoost is an implementation of the Gradient Boosted Decision Trees algorithm. We go through cycles that repeatedly builds new models and combines them into an ensemble model [191].

In every cycle, the errors in every observation are taken to build a new model on reducing them. The new predictions from such models are added to the ensemble of models [191].

17.9.4 Model 1

Using the top 15 features given by scikit-learn's RFClassifier new test and train datasets are made. The new train dataset is used to fit a new Random Forest Classifier. When run on test, the classifier gave an accuracy score of 0.90 but when we look in to the confusion matrix we can see that the F1 score is very low (0.017) true positive cases in case of the loan defaulters

```
array([[22281,     23],  
       [ 2043,    18]], dtype=int64)
```

Figure 166: Confusion Matrix of Model1

17.9.5 Model 2

Using the new features created by PCA during feature analysis, another model is built by implementing 1-Nearest Neighbours (1 was chosen after tweaking for the best accuracy metrics), an f1 score of 0.13, although the model improved, the no. of true positive cases for loan defaulters was less.

```
array([[20605,  1699],  
       [ 1798,   263]], dtype=int64)
```

Figure 167: Confusion Matrix of Model2

17.9.6 Model 3

XGBOOST model was applied to the complete cleaned final dataset that had

missing value treatment and outlier treatment done on it. With XGBOOST, while the accuracy dipped the f1 score improved with the number of true positive cases for the defaulter increasing double fold.

```
[[20868 1436]
 [ 1489  572]]
```

Figure 168: Confusion Matrix of Model3

17.9.7 Model 4

The final dataset was replaced by the dataset generated by PCA and XGBOOST was applied on the Principal Components. The accuracy and f1 score decreased with new dataset.

```
[[20886 1418]
 [ 1578  483]]
```

Figure 169: Confusion Matrix of Model4

17.9.8 Model 5

Another dataframe is created using the features that seem intuitive and XGBOOST is applied to the new dataset, a rise in accuracy and true positives of loan defaulters was observed.

```
[[20659 1645]
 [ 1433  628]]
```

Figure 170: Confusion Matrix of Model5

17.9.9 Model 6

The dataset where no missingvalue and outlier treatment is conducted is taken and the top 15 features from RF are subsetted to become the new training set for the XGB model, this model resulted in an accuracy score of 0.88 and highest number of true positive cases of loa defaulters of all the 6 models

```
[[25797 1820]
 [ 1694  689]]
```

Figure 171: Confusion Matrix of Model6

17.10 CONCLUSION

Microsoft Azure as a cloud platform helps in running the jupyter notebooks on cloud anywhere and works very robustly. The model 6 with XGBOOST trained on the dataset of the features with high importance according to Random Forest showed to give the best results out of the six models, The evaluation metric we considered for the selection of models was confusion matrix.

17.11 ACKNOWLEDGMENT

We would like to thank professor Dr. Gregor von Laszewski for helping us with the problem statement and dataset selection and also for all the guidance he has provided us during the coursework. We would also like to thank the associate instructors for helping us and guiding us.

18 UTILIZING MACHINE LEARNING TO PREDICT BREAST CANCER TUMOR DIAGNOSIS

Evan Beall

ebbeall@iu.edu

Indiana University

hid: fa18-523-80

github: [cloudbeall](#)

code: [cloudbeall](#)

18.1 ABSTRACT

Cancer is a life altering disease that affects people all over the world. In the United States alone it is estimated that throughout a person's lifetime they have around a 39.66% chance of developing cancer and a 22.03% chance of dying from cancer at all invasive sites [192]. Cancer research is a trillion dollar industry and gains more funding and researchers each year. One of the most highly funded and visible forms of cancer is breast cancer. Breast cancer research has taken the spotlight by being propped up by organizations such as the Suzan G Komen Foundation, National Breast Cancer foundation, etc. The amount of private investors, charitable donations, and pharmaceutical companies that have been funding research to treat this disease has been astronomical for decades.

Due to the fact that breast cancer research takes such a spotlight in the cancer research world, we decided to take a look at possible ways that cloud computing and machine learning could help physicians with diagnosing breast cancer patients. We have implemented a MongoDB Atlas database and combined it with several packages within the powerful Python 3.6 environment to perform analysis on a breast cancer fine needle aspirate dataset. This dataset was developed at the University of Wisconsin. It involved patients who had fine needle aspirate biopsies performed on suspicious tumors found in breast tissue [193]. The resulting dataset contains characteristics of these tumors. We have stored this dataset in the cloud via MongoDB Atlas and then taken this data and run a k-means clustering algorithm on it via the Python environment.

We have explored a method that is easily understandable and potentially deployable for other cancer diagnosis. Utilizing cloud technology to host the database like MongoDB Atlas would allow researchers to collaborate across the globe to give the k-means algorithm that we have created more training data and potentially make the algorithm more accurate for breast cancer diagnosis. This structure of database hosting and python algorithm could also potentially be applied to other types of cancers. This tool could be developed to be used by physicians treating cancer patients with other malignancies. Research would need to be performed on how the different nuclei characteristics change across malignancies and training datasets would need to be created.

18.2 KEYWORDS

Breast Cancer, Python, Big Data, Machine Learning, Xcyt, MongoDB Atlas

18.3 INTRODUCTION

Cancer is one of the most devastating conditions financially, emotionally, and physically to the person who is diagnosed and those around them. This disease can be classified as a human problem as it affects everyone around the world. With the odds of developing this disease over one's lifetime being over one in three, it is likely that everyone will be impacted by this disease at some point in life. In the United States alone "the Agency for Healthcare research and Quality estimates that the direct medical costs (total of all health care costs) for cancer in the US in 2015 were \$80.2 billion [fa18-523-80-acsecon]." This figure includes all health care costs that are associated with cancer. There are several aspects that go into the total cost of health care for cancer patients, but one of the main ones is the cost of cancer treatment.

Breast cancer has high visibility in the public eye and has some of the highest funding of any other types of cancer out there. In the United States, there is a branch of the government known as the National Cancer Institute that helps to organize governmentally funded research into Cancer. The current budget that has been allocated to the NCI for the 2019 fiscal year is \$5.74 billion [194]. This budget represents an increase of 79 million dollars from 2018[194]. Several large pharmaceutical companies and researchers get large grants to research new

compounds and molecules to fight of these diseases. The oncology pharmaceutical industry is over a trillion- dollar industry and each year there are thousands of clinical trials that are ongoing.

Oncology clinical trials function by exposing patient populations to new types of therapies and recording the results. In the past, cancer treatment and research has been mainly focused on casting a wide net to attempting to treat all individuals with generalized treatment regiments. The focus of large pharmaceutical companies has been to research drugs that can be used in a one drug fits all methodology. However, in recent years this mindset has begun to change. Many of the large pharmaceutical companies are moving to a more tailored approach in combating cancer. This has been seen in the form of genomic testing, targeted therapy, and specific receptor-based therapies. Clinical trials are setup to collect data throughout the time that the patient is on trial. Some clinical trials are set up to run for decades at a time with hundreds of visits per patient. These trials can include tens of thousands of patients. All of this together means that there is an enormous amount of data collected during each of these trials. This data is stored in electronic data capture systems or EDCs. The amount of data that needs to be worked on and analyzed for each of these trials means that there is a need for better ways to accomplish this task. Utilizing data science techniques such as cloud computing and machine learning seems like a natural fit into the world of oncology research.

We will delve into the possible implementation of machine learning techniques to analyze one aspect of an Oncology clinical trial. The Wisconsin Breast Cancer dataset that is utilized in this analysis give characteristics of tumor tissue. The dataset assigns a value to these characteristics between one and ten. The dataset also provides the final tumor diagnosis of benign or malignant. Having this diagnostic outcome will allow us to use this dataset as a training dataset to teach the machine learning algorithm we design. The goal will be to create a database to store this data within MongoDB Atlas and utilize Python to create an algorithm that can predict malignant vs benign tumors. To determine how accurate the algorithm can predict tumor diagnosis in breast cancer patients we will compare our k-means algorithm results with those provided within the dataset. This report will also run further analysis of other tumor characteristics that are present in the dataset. If an algorithm like this can be proven to be accurate enough, its implementation in clinical trial research and oncology

treatment in general could be life-saving. It could also help to eliminate the human error that is seen when a physician alone is making a diagnosis. By utilizing these tools as an aid to help physicians determine breast cancer diagnosis it could allow for earlier detection and more accurate treatment.

18.3.1 Dataset

The dataset used in this analysis is the Breast Cancer Wisconsin Data set. This dataset was created by Dr. William H. Wolberg. Dr. Wolberg is a physician at the University of Wisconsin Hospital [193]. The dataset utilized for this analysis is a collection of fine needle aspirate breast cancer tumor samples [193]. Fine needle aspirate biopsies are biopsies that are taken by inserting a fine needle into the affected site. The extracted tissue is then suspended into an aqueous solution and mounted on a slide. Dr. Wolberg then used a graphic program called Xcyt to analyze the features of each of these biopsies [193].

This dataset includes 569 unique patients with 12 attributes per patient. The attributes provided by this dataset includes de-identified patient identifiers, pathological diagnosis, and tumor characteristics. The specific characteristics included are clump thickness, uniformity of cell size, uniformity of cell shape, marginal adhesion, single epithelial cell size, bare nuclei, bland chromatin, normal nucleoli, mitosis, and class [195]. Each of these characteristics are assigned a numerical value from 1 to 10 except for the “Class” characteristic. The Class characteristics contain the results for malignant vs benign, these are assigned values 2 or 4 respectively. The below figure displays the output of the `pandas.DataFrame.describe()` method for the training dataset Figure [172](#):

Column1	Scn	A2	A3	A4	A5	A6	A7	A8	A9	A10	CLASS
count	699	699	699	699	699	699	699	699	699	699	699
mean	1071704	4.41774	3.134478	3.207439	2.806867	3.216023	3.486409	3.437768	2.866953	1.589413	2.689557
std	617095.7	2.815741	3.051459	2.971913	2.855379	2.2143	3.621929	2.438364	3.053634	1.715078	0.951273
min	61634	1	1	1	1	1	1	1	1	1	2
25%	870688.5	2	1	1	1	2	1	2	1	1	2
50%	1171710	4	1	1	1	2	1	3	1	1	2
75%	1238298	6	5	5	4	4	5	5	4	1	4
max	13454352	10	10	10	10	10	10	10	10	10	4

Figure 172: describe

The characteristics that are included in the data are of the cell nuclei that was present in the digitized images. The dataset is publicly available for analysis. The following image contains which time points that the data points were collected by Dr. Wolberg for this dataset in Figure [173](#).

Table 1: Gruops of instances

Group 1	369 instances	January 1989
Group 2	70 instances	October 1989
Group 3	31 instances	February 1990
Group 4	17 instances	April 1990
Group 5	48 instances	August 1990
Group 6	49 instances	January 1991
Group 7	31 instances	June 1991
Group 8	86 instances	November 1991
Total: 701		

Figure 173: Datapoints [196]

18.3.2 Xcyt

Xcyt is the program utilized by Dr. Wolberg and his team to generate the breast cancer dataset that we are using in this analysis. The Scyt program is broken into three parts. The initial portion of the program involves an operator selecting nuclei from FNA biopsy specimen by drawing rough lines around each nuclei in question[193]. The program then applies a curve fitting program to determine the exact boundaries of each cell nuclei that was identified by the operator[196]. This program then computes ten unique characteristics from the specimen nuclei. For each of these the mean, extreme value, and standard error are output. As a result, each patient that is analyzed by Xcyt has a 30-dimensional vector created. The data analyzed by his team included both this 30-dimensional vector and another dataset that included the overall nuclei characteristics[193]. The second portion of this dataset is the data that is included within the Breast Cancer Wisconsin Data set that will be utilized for this analysis. The following image shows an example of what an image going into the Xcyt program would

look like Figure 174:

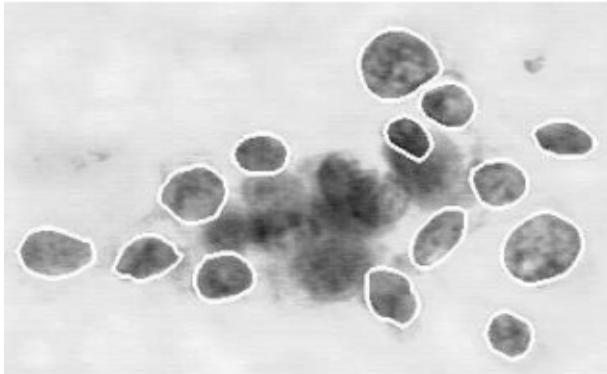


Figure 174: Xcyt Input[196]

The next part of the Xcyt program is the diagnosis portion. Xcyt utilizes a linear programming-based diagnostic method to determine if the suspected malignancy contains benign or malignant cells based off of the characteristic generated by the initial model. The system used by Xcyt is a variant of the multi-surface method called the multi-surface method-tree[197]. This diagnostic method creates strict planes based on malignant or non-malignant based on selected characteristics from the initial part. In the initial analysis the most accurate output occurred when extreme area, extreme smoothness, and mean texture were plotted. These resulted in nearly 97.5% accuracy in identifying malignant vs benign tumors[197].

The final part of the Xcyt program involves prediction of recurrence of the tumor. This portion utilizes a kernel density estimation to compare the amount of benign vs malignant nuclei that are found within a sample. By comparing these two figure's densities a bayesian computation is applied to compare the height of malignant densities divided by the sum of the densities of malignant and benign nuclei.

18.4 IMPLEMENTATION

18.4.1 Requirements

- Python 3.6 or higher
 - This instance was performed in the Spyder IDE
- Python Packages

- Python Standard Library
- Pandas
- Numpy
- Matplotlib
- pymongo
- Dataset downloaded in .csv file format
- MongoDB Atlas Account

18.4.2 MongoDB Atlas Database

The cloud-based database used in this project is the MongoDB database called MongoDB Atlas. To deploy this type of database the user will need to create a cluster and then set up a database and collection within that cluster. This will allow the user to start entering documents into the collection. The type of cluster that was used in this project is the MongoDB Atlas free tier instance. This instance uses the M0 Shared Cluster Tier via an AWS server[198]. This cluster tier is free to deploy, will remain free forever and comes with 512 MB of free data storage [198]. The below image shows the type of MongoDB Atlas database that was constructed for this project Figure [175](#).

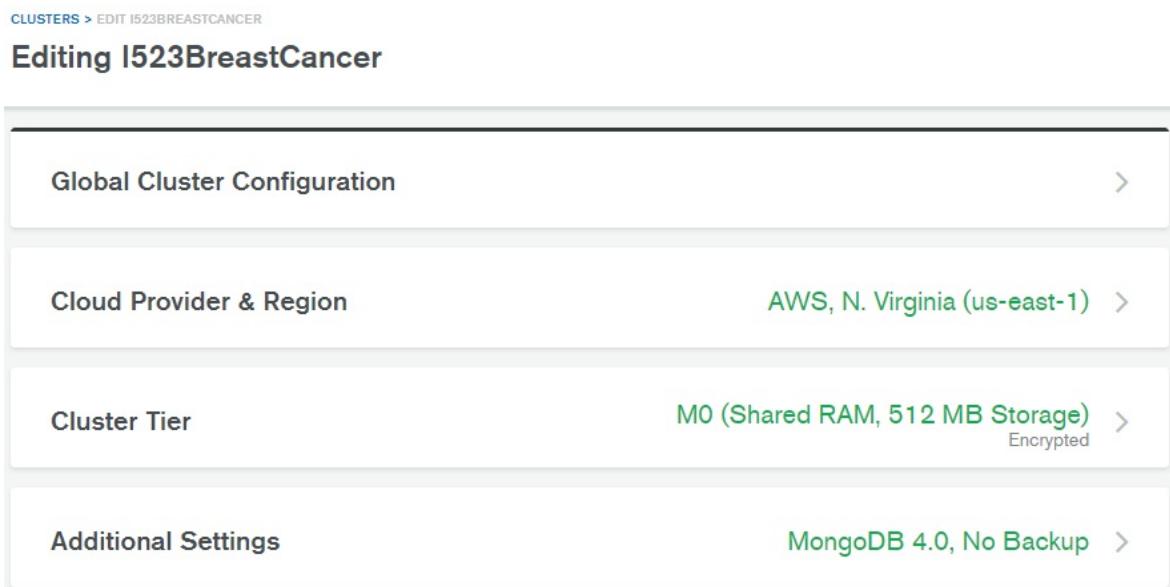


Figure 175: Mongodb Breast[198]

We have set up a database cluster via MongoDB Atlas that has the data to be analyzed already inside. Prior to connecting to this data however, the user will need to make sure that the python package pymongo and dnspython are both

installed. This can be accomplished via the following terminal commands:

```
python -m pip install pymongo  
python -m pip install dnspython
```

After both of these libraries are installed the user can utilize the following code will allow a user to access this database to run analysis:

```
import pymongo  
import pandas as pd  
  
connection = pymongo.MongoClient('mongodb+srv://I523Admin:<I523Admin>@i523breastcancer-fdclg.mongodb.net/test?retryWrites=true')  
  
db = connection.Wisconsin_Breast_Cancer.I523  
df = pd.DataFrame(list(db.find()))
```

The above code will connect to the instance of the MongoDB Atlas cluster and to the specific database and collection that has been created to house the data. It will then utilize python package pandas to import the data into a dataframe so that the data can be handled locally. Below is a graphical image of what the database we have set up looks like in MongoDB Atlas Figure [176](#):

The screenshot shows the MongoDB Atlas interface. In the top left, it says "mongoDB Atlas All Clusters". Below that, "CONTEXT" dropdown is set to "Project 0". Under "PROJECT", there are links for Clusters, Alerts (0), Backup, Users & Teams, Settings, Stitch Apps, Docs, and Support. The main title is "I523BreastCancer". Below the title, tabs for Overview, Real Time, Metrics, Collections (which is underlined), and Command Line Tools are visible. It shows "1 DATABASES 1 COLLECTIONS". A button "+ Create Database" is present. Below that is a search bar "NAMESPACES" with a dropdown showing "Wisconsin_Breast_Cancer". The "Collections" section shows "I523" and "1523". On the right, a "Find" section has a "FILTER" button with the value {"filter": "example"}. Below it, a table titled "QUERY RESULTS 1-20 OF MANY" lists several document snippets, with the last one being: _id: ObjectId("5c04c5d4a24ede595c765de2"), Scn: "1000025", A2: "5", A3: "1", A4: "1", A5: "1", A6: "2", A7: "1", A8: "3", A9: "1", A10: "1", CLASS: "2".

Figure 176: Wisconsin database[198]

While Dr. Wolberg and his team used the novel program of Xyct to perform their machine learning algorithm on a more complex dataset, this report will instead use the popular k-mean clustering algorithm on a more high-level dataset within the Spyder Python 3.6 environment. Once this environment has been downloaded and the dataset has been download in .csv format. The next step would be to import the pandas, numpy, and matplotlib packages will need to be imported into the code. This will be accomplished using the following code:

```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
from pandas import DataFrame, read_csv
from matplotlib.backends.backend_pdf import PdfPages
import pymongo
```

The dataset that is in the working directory can now read using the `pandas.read_csv()` and placed in a dataframe. Alternatively, to grab the data from the MongoDB Atlas database you would run the code included above. Both of

these methods will get you the data that is needed for this analysis. However, the MongoDB Atlas method is deployable by researchers across the globe. Additionally these users would be able to add more data to the database whenever they needed to. By utilizing this simplistic method, it allows this program to be easily reproduceable and deployable by researchers that are not well versed in machine learning methodology. This would also allow for researchers to shape the analysis to their own data and create customizations to the base program. The 16 missing values in the dataset are all found in column A7 in the dataset, we have addressed this missing values by utilizing the fillna() method. This method is assigning the median value for the other values in column A7 and placing it as a place holder for the missing values via the following code:

```
df = pd.read_csv('breastcancer.csv', na_values = '?')
df = df.fillna(df['A7'].median())
```

This analysis will be split into several different parts. The first part will include a statistical output for each of the characteristics that have been derived by the dataset. The frequency of each of these characteristics will then be plotted on basic bar graphs. The above discussed steps are carried out via the visualization.py item in the github repository. Some of the derived statistical features will include mean, median, and standard deviation of each of the characteristics. The following is a sample of code that shows how the visualizations were created:

```
with PdfPages('finalproject.pdf') as pp:
    for i in range (9):
        fig1 = plt.figure()
        sp1 = fig1.add_subplot(1, 1, 1)
        sp1.set_title(columns[counter])
        sp1.set_xlabel("Results")
        sp1.set_ylabel("Number of Cases")
        plt.xticks((1,2,3,4,5,6,7,8,9,10))
        sp1.hist(df[(column_header[counter])], bins=10, color="blue", edgecolor =
"black", alpha=0.5, label = [1,2,3,4,5,6,7,8,9,10])
        pp.savefig(fig1)
        counter = counter + 1
pp.close()
```

In the second part of the analysis, a basic k means algorithm will be created and implemented on the data set. The code will select two data points randomly using the np.random() method as the initial ‘means’ of the dataset these can also be seen as the initial cluster points. These clusters will be utilized for each datapoint to calculate the euclidian distance. This dataset will continue to iterate

in this way by selecting randomly selected datapoints as new cluster points each time. Each time the final cluster assignment will be compared to the one before this. The condition for the system to stop this iteration will occur when the system identifies no change in cluster assignment or the system has iterated 1500 times. This is put in place to keep the cluster method from running continuously. This is accomplished by running the clustering.py program contained in the github repository.

The final portion of this will compare how successful the k means algorithm we have implemented predicted the diagnosis condition. In this step we take the cluster assignment and compare it to the final column included in this dataset that contains the actual diagnosis of the patient. By doing this we are able to calculate the error rate for each cluster and the overall error rate for both clusters. This will give us a real world definition of how accurately the k mean algorithm is functioning. This step is the main part of what we were trying to accomplish. It combines pulling data from the MongoDB Atlas database with computing within Python to diagnose the tumor tissue. This is accomplished by running the analysis.py program contained in the github repository.

18.5 RESULTS

Applying the k-means algorithm to this data set happened in several different steps. The following analysis will be split into those different steps to breakdown how the final results were achieved.

18.5.1 Visualizaton

In the first section of the code the data is read from a .csv file using the pandas read_csv() method. This data was then inserted into a dataframe. Once here the mean, median, standard deviation, and variance were all calculated per characteristic. An example of these results can be seen below Figure [177](#):

```
Row A 5
Mean: 3.22
Median: 2.0
Standard Deviation: 2.21
Variance: 4.9
```

Figure 177: statistics

Each characteristic in this dataset is assigned a value of 1-10. Using these variables several visualizations were created to allow for the research scientists and physicians to have an idea of how the cells in their research tumors look. This can be utilized in future studies to account for variances in tumor malignancies across cancer diagnosis. An example of these visualizations can be seen below in Figure 178:

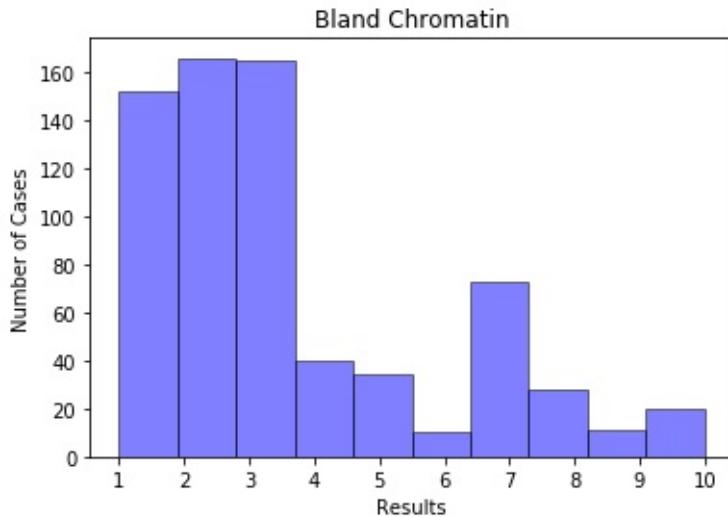


Figure 178: chromatin

18.5.2 K-Means Clustering

The next part of this analysis included the creation and recalculation steps for the k-means clustering steps. During this step cluster points were chosen as random data points within the dataset. The euclidean distance was then calculated from each of these clusters to determine a malignancy diagnosis. Upon either reaching 1500 iterations or the cluster assignment not changing across iterations the clustering would complete. The system would then produce the calculated k means for each characteristic in the study. The following table shows the calculated k-means for each characteristic depending on their malignancy diagnosis.

18.5.3 Clustering Assessment

The final portion of the study compares the results of the k-means clustering algorithm with that of the correct diagnosis that was provided in the initial dataset. This portion of the study calculates the error rate for both the malignant

and benign clusters. This then goes on to calculate the total error rate across both clusters. The following images identifies the total amount of datapoints in each cluster and the error rates that were calculated. The error rates calculated in this figure are compared the given diagnosis of each of the patients in the dataset. This image also includes the time (in seconds) that it took to perform the analysis see Figure [179](#):

```
Amount of Benign tumor samples: 464  
Amount of Malignant tumor samples: 235  
  
U2 Error Rate: 2.37  
U4 Error Rate: 7.23  
  
Total Error Rate: 4.01  
  
Total time taken to perform analysis: 9.832224130630493
```

Figure 179: error

18.5.4 Benchmarks

In Dr. Wolbergs experiments they were utilizing a 30-dimensional vector for each data point in this data set. The deployment used in this study only utilizes a 10-dimensional vector for each analysis. In this experiment it takes around 9.65 seconds for the analysis to import the data, read the data, sort the data into initial clusters, iterate over the dataset to ensure cluster assignments are correct and then run a comparison between estimated cluster assignments and the real cluster assignment. The only values that could be found from Dr. Wolberg's implementation of the Xcyt program were the length of time that the system would take to generate the characteristic map from the FNA imaging. This was estimated to take around five to ten minutes. This however is not a comparable time stamp as the system was estimating the characteristics and not running the diagnostic report.

18.6 LIMITATIONS

Limitations of this type of analysis are based around several factors. The first and biggest one is the necessity to utilize the initial portion of the Xcyt program to generate a characteristic map of FNA samples. This program is currently

accessible for free via Nick Street at University of Iowa[199]. These packages have been outdated for many years, however, Broad Institute has maintained the packages for implementation in R. To utilize these datasets in the future some manipulations to port the datasets generated via R to python for machine learning would be required. Without this information, the machine learning algorithm implemented in this report will not run. The assumption is that the initial portion of the Xcyt program output could be utilized going forward and combined with the easy to implement and customize python algorithm that is displayed in the report above.

Another limitation of this type of algorithm is its ability to apply to a variation of cancer types. FNA biopsies are not able to be produced on cancers that are not solid tissue samples. This limits the effectiveness of this analysis as it only focuses on a type of biopsy only available to a selection of cancer tissues. While this is a limitation, implementing this algorithm for other solid tumor cancers could be extremely helpful for diagnostic purposes.

18.7 CONCLUSION

Dr. Wolberg team's implementation of the novel Xcyt program to diagnose malignancies in breast cancer patients is a revolutionary way that machine learning could be implemented in cancer research. This type of methodology was implemented on a very specific instance of cancer research. The machine learning algorithm was able to identify tumor nuclei characteristics that indicated malignant tumor cells at a rate of 97.5%[197]. However, this type of machine learning program is extremely specific and not very reproduceable. This program utilized an analysis method that created a 30-dimensional vector for each patient. The team was then able to run analysis on each comparison point to find the highest rate of correct results. Finding that very specific combinations of characteristics would lead to higher rates of success. In the end they discovered that the specific combination of extreme area, extreme smoothness, and mean texture gave the highest success rate. This is a time-consuming way to run analysis as it took the algorithm nearly 5-10 minutes to create the 30-dimensional vector for each sample[193]. On top of this, the researchers would need to then use trial and error to discover which combination of these vectors would give the highest success rate.

By cutting this down to only the high-level data assessment with fewer characteristics to analyze the process on the front and back end can be sped up significantly. The Xcyt program implemented by Dr. Wolberg's team utilized very specific characteristics that may or may not apply across other cancer diagnosis. However, by utilizing the higher-level dataset with only 10 points of comparison we could make this method more scalable and applicable across all oncological diagnosis. The k-means clustering algorithm that was implemented in this report showed around a 96% rate of success. This is only 1.5% lower than that seen by Dr. Wolberg's team.

By lowering the amount of complexity of assessment, we have increased the reproducibility across other cancer diagnosis and decreased the time investment for researchers to run analysis. By weighing the fact that the decrease in efficacy is only 1.5% we could allow researchers to make a judgement call as to if this decrease in efficacy is worth the increase in efficiency. This tool should not be used as the sole method for diagnosis but should be used in combination with physician determination in combination with other diagnostic tools. Taking all of that into account, making this type of analysis more easily reproduced and available to all researchers would more than make up the 1.5% loss in efficiency. Python tools are much more approachable and easier to deploy the proprietary program Xcyt that was created by Dr. Wolberg's team. Since Python is an open sourced tool and each package that was used in this analysis is also available to everyone. This means this method is easily accessible to all researchers willing to take the time to set up analysis points for their teams.

19 CREDIT SCORING ALGORITHM AND ITS IMPLEMENTATION IN PRODUCTION ENVIRONMENT

Nhi Tran

nytran@iu.edu

Indiana University

hid: fa18-523-83

github: [cloud](#)

code: [cloud](#)

19.1 ABSTRACT

A credit scoring algorithm is essential to help any bank determine whether to authorize a loan to consumers. Most of the decisions require fast results and high accuracy in order to improve the bank customer satisfaction and profit. Choosing a correct machine learning algorithm will result in high accuracy in the prediction and a reasonable implementation of the algorithm will result in the fast service a bank can provide to its customers.

19.2 KEYWORDS

fa18-523-83, machine learning, predict algorithm, classification, docker, deployment, API

19.3 INTRODUCTION

For every machine learning problem, there are primarily two main areas that everyone focuses on: which machine learning algorithms to use and how to implement and integrate the machine learning code into a new or existing production infrastructure.

Most of the time, machines make predictions by learning and observing the data patterns from previously existing data with known results. That process is called training. Once the training is over, the machine learning code can predict the

unknown results by applying the trained models into new data.

The next thing to do after prediction would be determining how to retrieve and apply the result of the prediction into a new or existing production application and how to ensure the continuous deployment into the production environment without resulting deployment code defects.

The business problem that will be focused on is how to determine whether or not a customer will be experiencing a financial distress in the next two years. By predicting the business problem, banking companies can use the results as part of their business rules to decide whether to approve their products to the customers. Once the algorithm is finalized and trained into a model, the deployment of code will be performed and benchmarked to provide an overview of how long it will take to prepare and run the code from one environment to another.

19.4 DESIGN

19.4.1 Dataset

We are utilizing an existing dataset from the ‘Give Me Some Credit’ competition on Kaggle to train and test algorithms and determine which algorithms would be the best to predict the probability of someone experiencing financial distress in the next two years [200]. However, due to Kaggle competition dataset can no longer be downloaded after the competition ended, the same data provided by the competition was reuploaded by a Kaggle user in Kaggle Dataset section [201]

The Kaggle competition contains a training set, a test set, and a data dictionary. The training set contains 150,000 records of previous customer data with an existing label indicating whether or not each customer had serious bank delinquency within two years. The test set contains about 100,000 records without any label data, which will not be part of the analysis but 50 test records will be used as part of the benchmarking report.

Data descriptions, as mentioned in [202], are:

- SeriousDlqin2yrs: label data, contains ‘Yes’ or ‘No’ indicator
- RevolvingUtilizationOfUnsecuredLines: total balance of unsecured lines such as credit cards and personal lines
- age: bank customers’ age
- NumberOfTime30-59DaysPastDueNotWorse: number of times each customer has been 30-59 days past due but no worse in the last 2 years
- DebtRatio: monthly debt payments divided by monthly gross income
- MonthlyIncome: bank customers’ monthly income
- NumberOfOpenCreditLinesAndLoans: number of open loans and lines of credit
- NumberOfTimes90DaysLate: number of times each customer has been 90 days or more past due
- NumberRealEstateLoansOrLines: number of mortgage and real estate loans
- NumberOfTime60-89DaysPastDueNotWorse: number of times each customer has been 60-89 days past due but no worse in the last 2 years
- NumberOfDependents: number of dependents in family excluding themselves (spouse, children etc.)

19.4.2 Data Visualization

Using the `describe()` function in Python **pandas** package, the statistics for each attribute within training dataset is populated. The statistics include count, mean, standard deviation, minimum, quantiles, and maximum. The function helps with seeing outliers and identifies values or columns that need to be cleaned up. For example, in the training set, some areas that will need to be carefully examined are **age** with a minimum value of **0**, **MonthlyIncome** and **NumberOfDependents** contains **NaN** value in their quantiles.

	age	MonthlyIncome	NumberOfDependents
count	150000.0	120269.0	146076.0
mean	52.29520	6670.22123739	0.7572222678605657
std	14.77186	14384.6742152	1.1150860714872997
min	0.0	0.0	0.0
25%	41.0	NaN	NaN
50%	52.0	NaN	NaN

75%	63.0	NaN	NaN
max	109.0	3008750.0	20.0

Data Visualization in Python can be done using graphing packages such as **matplotlib**, **seaborn**, etc.

Using **matplotlib**, Figure 180 shows that there is a small count of **0** value as outliers and the distribution without those outliers will be a right-skewed distribution. Therefore, it is better to replace those outliers with the median value of the distribution.

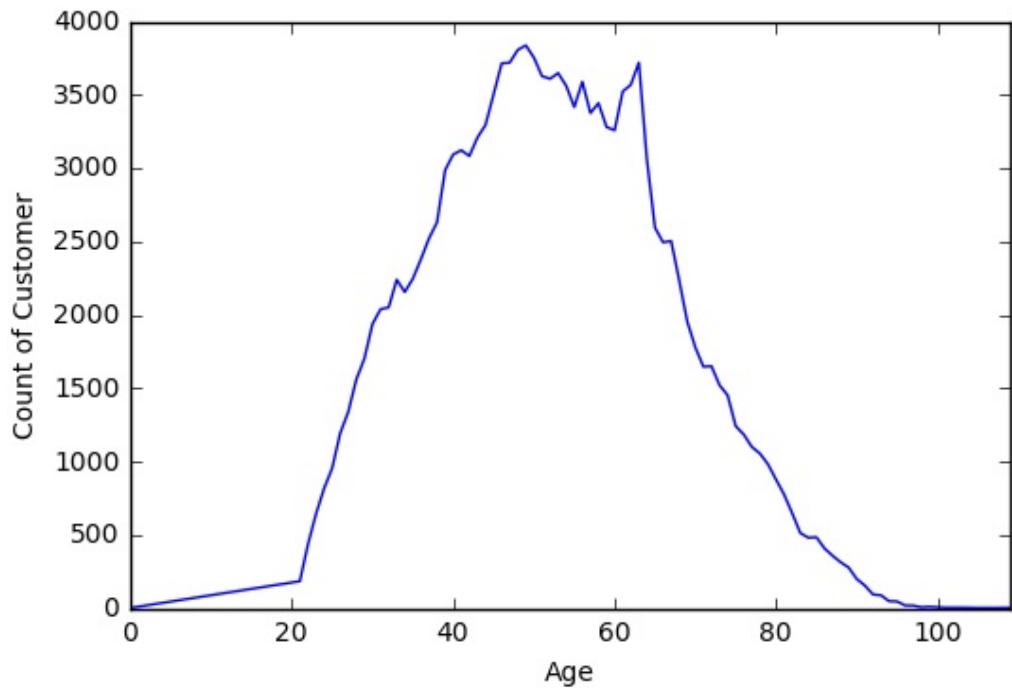


Figure 180: Count of Customer by Age

Figure 181 shows the overall distribution of the count of customers that experienced past due, the count of customers that have open credit and real estate lines, and the count of customers that have dependents other than themselves. Most of the distributions are right-skewed, the majority of them do not have any past due or dependents.

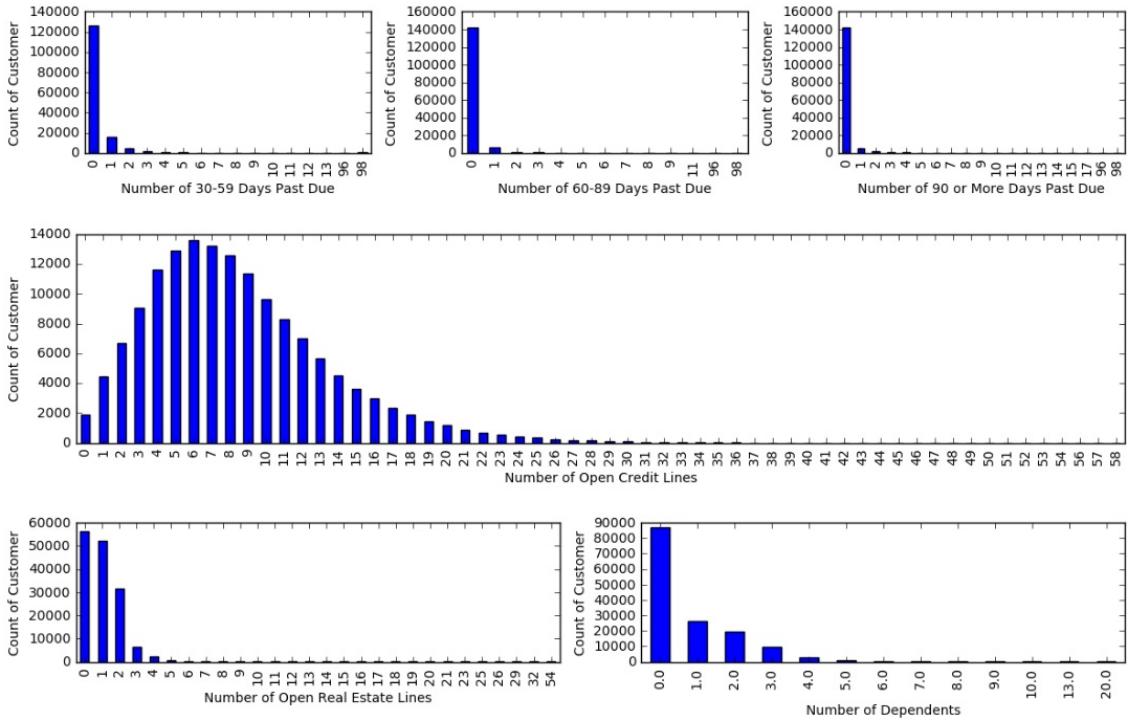


Figure 181: Count of Customer by Multiple Attributes

Figure 182 shows the distribution of the label that will be what the machines are trying to predict. If the label distribution is not even, the model might overfit and give a higher chance of predicting the label that has a higher population. In this case, the amount of customers that had no delinquency is 14 times more than the number of customers that had delinquency. This issue will need to be handled during the **Data Cleaning** or **Model Training** process to avoid overfitting.

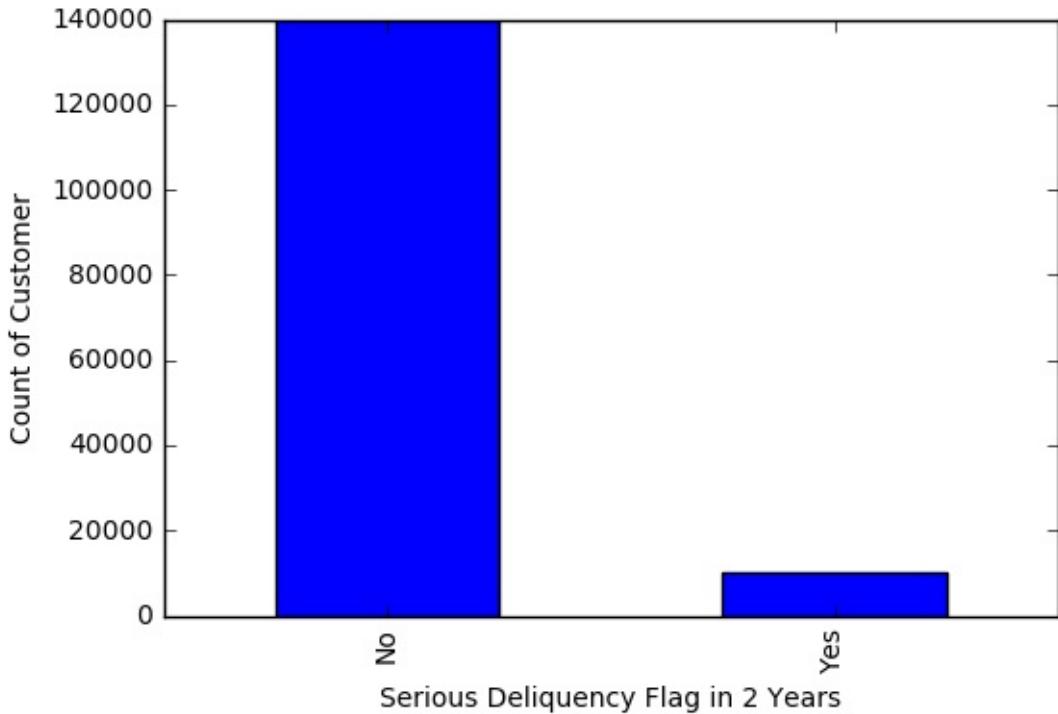


Figure 182: Label Distribution

19.4.3 Data Preparation

From the **Data Visualization** step, the first basic data preparation is replacing missing value with median value. Another way to handle this issue is to drop records that have missing data, however, due to low data volumes and imbalanced class data distribution, it is better to replace missing data instead.

Using **seaborn**, **heatmap** can be used to visualize the correlation of all attributes. Correlation is a measurement of the strength of association between two variables and the relationship's direction [203]. Correlation is an important indicator in the **Feature Selection** process to help determine which attributes should be used as part of the training set and which attributes should be irrelevant. There are multiple correlation methods to calculate the correlation coefficient, the method that is used for this training set is called **Spearman**. Figure [183](#) shows the first observation of the correlation between all variables.

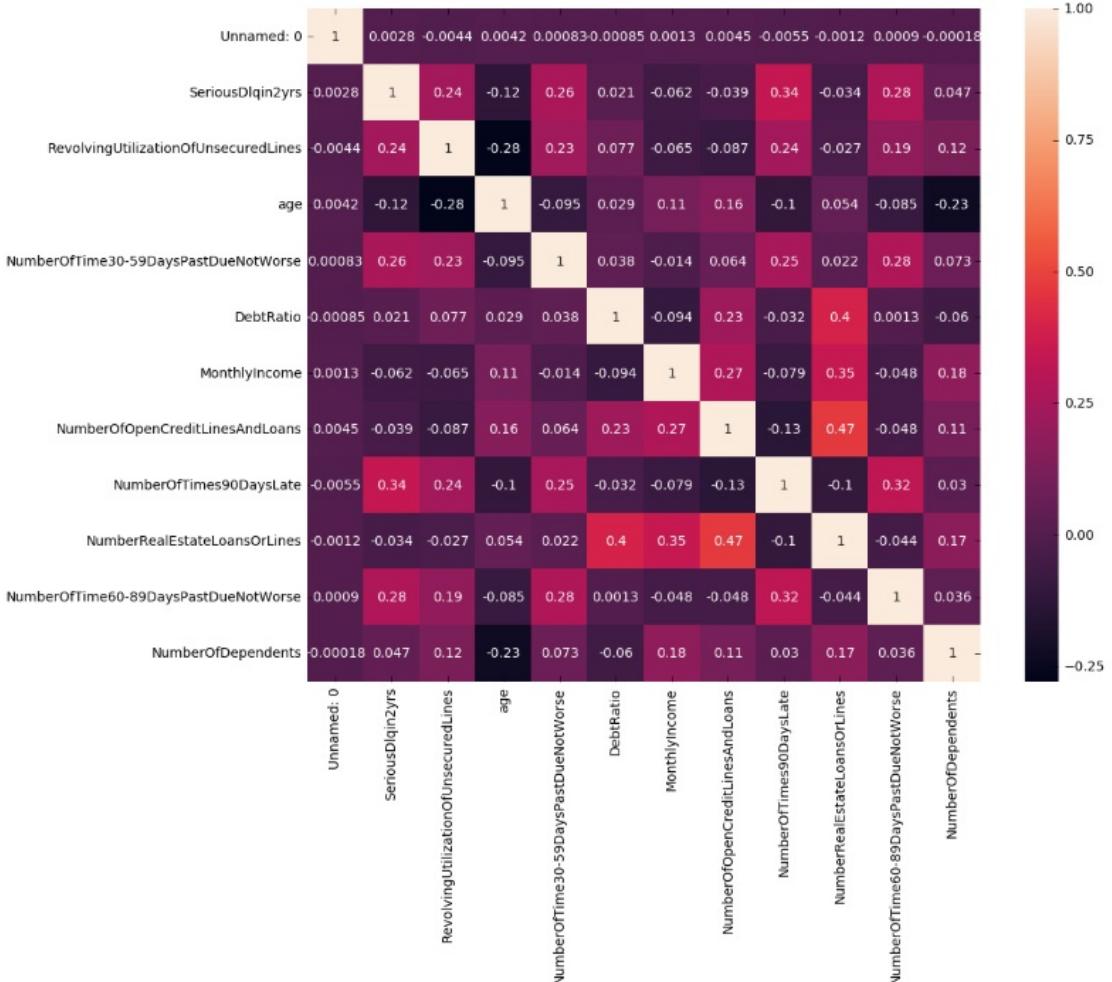


Figure 183: Correlation First Run

From Figure 183, **DebtRatio** has a very low correlation to the class label. It means that the debt ratio of a customer does not impact whether a customer will be likely to default in the next 2 years. At the same time, the **DebtRatio** variable is highly correlated with the number of open credit lines and real estate loans, which could potentially interrupt the training algorithms and result in a false prediction. Therefore, it is better to exclude **DebtRatio** out of the final dataset.

Another observation from +fig:correlation1 is that all number of past due variables have a high impact on the class label and between themselves at the same time. Since they are all past due type of data, adding them all into a new variable called **TotalNumberOfPastDue** and dropping all individual past due variables could be a good idea to avoid conflict between those variables.

Similar to past due variables, **NumberOfOpenCreditLinesAndLoans** and

NumberOfRealEstatesLoansOrLines are highly correlated to each other, therefore, adding them together into a new variable called **TotalNumberOfOpenLines** is a way to solve the problem.

Once all of the new variables are added and individual variables are removed, it is good to run the **heatmap** again to determine whether or not there is more cleaning to be done. Figure 184 is the second **heatmap** run with all prepared variables.



Figure 184: Correlation Second Run

The final observation is that **MonthlyIncome** and the new variables **TotalNumberOfOpenLines** are also highly correlated to each other. Since **MonthlyIncome** has a higher correlation than **TotalNumberOfOpenLines**, the last step to have a final training dataset is to drop **TotalNumberOfOpenLines**.

19.4.4 Model Training

Because the training data has imbalanced classes, there are multiple ways to handle this issue:

Methods that will be used [204]:

- Changing Performance Metric: instead of using only **Accuracy** as the main performance metric to evaluate models, try to include other metrics such as **Confusion Matrix**, **Precision**, **Recall**, **F1 Score**, **Kappa**, **ROC Curves**.
- Resample Dataset: add copied of instances from low volume class as an over-sampling method, or delete instances from high volume class as an under-sampling method, or re-run the algorithm on shuffled data (k-fold cross-validation). Under-sampling and cross-validation are used as part of the training process for this problem and will be included as part of the result evaluation section.
- Different Algorithms: run data on multiple algorithms and evaluate and choose the best algorithms that fit the provided data.
- Penalized Models: Use penalized methods to cause the model to pay more attention to the minority class to evaluate.

Methods that will not be used [204]:

- Increase Training Data: gaining more data to provide more balance and insights in data.
- Generate Synthetic Samples: use popular algorithms such as Synthetic Minority Over-sampling Technique to generate synthetic data.

The goal is to determine whether someone will experience financial distress in the next two years, therefore, there will only be two values in the label. With a binary classification problem on supervised data, it is best to use classification algorithms such as Random Forest, Logistic regression, XGBoost, Support Vector Machine, Neural Network, and Support Vector Machine.

- Random Forest: an ensemble of Decision Tree algorithm, builds and merge multiple decision trees together to get average results for prediction [205]. Using Python, the algorithm can be used from ***RandomForest()*** function in ***sklearn*** package.
- Logistic Regression: uses an equation with weights for coefficient values of input values to make a prediction. For Binary Logistic Regression, a threshold between 0 and 1 is needed to determine the category of the prediction [206]. This algorithm is ***LogisticRegression()*** function in ***sklearn*** package.
- XGBoost: xgboost is

“a scalable and accurate implementation of gradient boosting machines and it has proven to push the limits of computing power for boosted trees algorithms as it was built and developed for the sole purpose of model performance and computational speed” [207].

There is an XGBoost API that can be called via ***sklearn*** using the function ***XGBClassifier***.

- Neural Network: a neural network is a machine learning algorithm that is inspired by the human brain. Neural network identifies the pattern of input data through multiple layers of artificial neural layers [208]. The function ***MLPClassifier*** from ***sklearn*** is one of the Neural Network algorithms for classification problem.
- Support Vector Machine: an algorithm that finds patterns and predicts output data by finding hyperplane(s) in an N-dimensional space fa18-523-83-www-svm]. The function ***LinearSVC*** from ***sklearn*** is one of the Support Vector Machine algorithms for classification problem.

19.4.5 Result Comparison

There are two sets of run that were evaluated in the provided dataset: normal run and balanced run. On a normal run, due to imbalanced data distribution, it is important to compare metrics for algorithms with defaulted parameters and

algorithms with penalized configured parameters.

In order to evaluate and determine which algorithms to use, it is important to understand what each metric mean and what the expectation of each metric is. In the evaluation, there are five metrics that will be used to compare between models [209]:

- Classification Accuracy: the ratio between correct prediction over the total sample sizes. It only works well if we have a balanced class label distribution.
- Area Under Curve (AUC): often used for binary classification problem. AUC measures the area under the curve of False Positive Rate vs True Positive Rate plot. AUC ranges from 0 to 1 and a higher AUC indicates a better-performed model.
- Precision: the number of correct positive results divided by the number of positive results predicted. Precision indicates how many instances that were predicted correctly.
- Recall: the number of correct positive results divided by the number of records that should have been identified as positive. Recall indicates how much data was misclassified.
- F1 Score: the F1 score is the balance between precision and recall, a better model gives a higher F1 score.

The metrics for all models are being performed by `cross_validation()` function in **sklearn** with a parameter of `cv=10` to allow the data to be re-shuffled and re-validated ten times. It is to ensure a high-quality result and optimize the data volume. The results are the average of all ten runs.

	LR	LR Pnlzd	RF	LSVC	LSVC Pnlzd	XGBoost	XGBoost Pnlzd
fit_time	0.64	0.43	2.50	32.10	31.87	5.34	5.10
score_time	0.03	0.02	0.26	0.02	0.02	0.35	0.30
test_f1	0.00	0.00	0.11	0.01	0.01	0.03	0.00

train_f1	0.00	0.00	0.86	0.01	0.01	0.03	0.00
test_accuracy	0.93	0.93	0.93	0.87	0.93	0.93	0.93
train_accuracy	0.93	0.93	0.98	0.87	0.93	0.93	0.93
test_recall	0.00	0.00	0.07	0.06	0.00	0.01	0.00
train_recall	0.00	0.00	0.77	0.06	0.00	0.02	0.00
test_precision	0.00	0.00	0.27	0.06	0.07	0.49	0.20
train_precision	0.00	0.00	0.99	0.04	0.05	0.60	0.25
test_roc_auc	0.64	0.58	0.68	0.59	0.58	0.80	0.79
train_roc_auc	0.64	0.58	1.00	0.58	0.58	0.80	0.80

Accuracy will not be a relevant metric to determine the performance of each model. It is included to emphasize a point that a model can provide a really high accuracy but can still perform poorly. In term of **F1 Score**, **Random Forest** seems to perform well comparing to the rest of the models. In term of **AUC**, **Random Forest** has a high **AUC** value in training dataset and a lower **AUC** value in test dataset whereas **XGBoost** performed consistently in both train and test run.

The same algorithms will be applied to the balanced dataset. **Sklearn** has a function that allows data to be shuffled with a specification of how many samples are needed for each label to ensure a result of the balanced-label dataset. This evaluation will not require penalized algorithms.

	LR	Random Forest	XGBoost	MLPClassifier	Linear SVC
fit_time	0.16	0.34	0.66	1.61	1.59
score_time	0.02	0.06	0.06	0.02	0.01
test_f1	0.61	0.66	0.72	0.51	0.26
train_f1	0.61	0.97	0.73	0.51	0.26
test_accuracy	0.60	0.68	0.73	0.55	0.50
train_accuracy	0.60	0.97	0.73	0.55	0.50
test_recall	0.61	0.63	0.72	0.58	0.36
train_recall	0.61	0.96	0.73	0.59	0.36

test_precision	0.60	0.70	0.73	0.59	0.40
train_precision	0.60	0.98	0.74	0.60	0.47
test_roc_auc	0.64	0.74	0.79	0.59	0.55
train_roc_auc	0.64	1.00	0.80	0.60	0.56

Accuracy is now an important factor to determine how well a model performed. In this case, **XGBoost** has the highest accuracy out of all models as well as a high and consistent **AUC** value. **Random Forest** also returns a great result but seems to be inconsistent due to its higher result in train data comparing to test data.

After both comparisons, **XGBoost** is the model that yields high and consistent performance. The **XGBoost** model on balanced dataset will be saved and used in the credit score application to predict the banking credit problem we are trying to solve.

19.5 IMPLEMENTATION

19.5.1 Technologies Used

19.5.1.1 Python Packages

- click: allows arguments to be passed to python code.
- sklearn: contains off-the-shelf machine learning algorithms packages with evaluation/report functions to cross-validation, shuffle data, and report metrics
- xgboost: contains xgboost algorithm
- pandas: allows easy-to-use abilities to read, process, slice, write and store data via dataframe and pickle
- flask: allows the ability to run and process APIs via web services via Python

- jupyter : jupyter notebook package that allows user to run and see Python result without having to construct a full syntax Python application
- kaggle: Kaggle API provides the ability to pull and push data from Kaggle website using command tool implemented in Python 3. The Kaggle API GitHub page provides a detailed installation, API credentials and commands instruction [210].
- flask: Flask API is a Python package that allows RESTful web service from Python.
- matplotlib: python package that allows graphing in Python
- seaborn: another python package that allows more new graph types in Python

19.5.1.2 Docker

Docker creates containers that are standalone images that bundle all dependent configuration files, packages, tools, libraries, and applications. A user can run any application through images without having to worry about the rest of the dependent components.

19.5.1.3 AWS EC2

AWS EC2 provides cloud services to host server, run and deploy the code.

19.5.1.4 Other Technologies

- GitHub
- Ubuntu 18.04 Bionic Beaver or 18.10 Cosmic Cuttlefish Server

19.5.2 Prerequisites

In order to run the code and reproduce the run, the following prerequisites need to be met:

- Ubuntu 18.04 Bionic Beaver or 18.10 Cosmic Cuttlefish Server: all codes were tested on Ubuntu 18.04 and 18.10 Server
- AWS Account : an AWS account is required to be able to launch a cloud server instance for deployment and benchmarking results. The AWS account can be created via the AWS EC2 page [211]. Credit card information is required during registration but the server instance can be launched for free.
- Setting up EC2 Server Instance: once the AWS is created, AWS EC2 instances can be created. Once logged in, the user can ***Launce Instance*** with the following configurations [211]:
 - Step 1: Choose an Amazon Machine Image (AMI) Cancel and Exit: select ***Ubuntu Server 18.04 LTS (HVM), SSD Volume Type***
 - Step 2: Choose an Instance Type: default to free tier option
 - Step 3: Configure Instance Details: use all default options
 - Step 4: Add Storage: use all default options
 - Step 5: Add Tags: use all default options
 - Step 6: Configure Security Group: allow SSH (Port 22) and HTTPS (Port 443) with Source ***0.0.0.0/0*** and ***::/0***
 - Step 7: Review Instance Launch: after hitting launce, make sure to choose an existing or create a new key pair and download the key. This will be the only time user can download a key pair. Ensure the key permission is 400.

To connect to AWS EC2 server, run the following command:

```
ssh -i <key-file-and-directory> ubuntu@<AWS-public-DNS>
```

- Make: ensure that ***make*** is installed. If not, use the following command to install ***make***:

```
sudo apt-get install make
```

This will allow the ***make*** command from ***Makefile*** to be run. The rest of the prerequisites packages and software can be run using ***make*** command.

- Project's Git Command: install git command by running the following command:

```
sudo apt-get install git-core
```

- Project's Github Repository Cloned: ensure all project is cloned from GitHub using the following command:

```
sudo git clone https://github.com/cloudmesh-community/fa18-523-83.git
```

- Kaggle Account: a Kaggle account is required to pull data from Kaggle.
- Kaggle API Credentials File: after the Kaggle account is created, go to the **Account** tab of user profile and select **Create API Token** to generate and download `kaggle.json` and save as `kaggle.json`. This file will be moved to a specific folder once **`kaggle`** package is installed [210].

19.5.3 Project Code Structure and Components

The directory structure of the project are:

```
tree
.
├── bin
│   └── report_time.sh
├── data
│   ├── interim
│   ├── processed
│   └── raw
├── Makefile
└── notebook
    ├── 00-Visualization-Notebook.ipynb
    └── 01-Evaluation-Notebook.ipynb
├── README.md
└── src
    ├── data
    │   ├── preprocessing.py
    │   ├── to_json.py
    │   └── train_data.py
    ├── docker
    │   ├── app.py
    │   ├── Dockerfile
    │   └── requirements.txt
    └── evaluation
        ├── result_balanced.csv
        └── result_imbalanced.csv
```

19.5.3.1 LICENSE

LICENSE file indicates which licensing the project and its code are under. In this case, it is MIT license.

19.5.3.2 README

README file provides a short description of the project and code components and instruction on how to run the code.

19.5.3.3 Makefile

Makefile contains a list of shortcut commands that can easily run via a ***make*** command when the user is in the directory where the Makefile is.

19.5.3.4 Dockerfile

This file contains the instruction and components to build the docker image. The file content is:

```
FROM python:3.6
WORKDIR /app
COPY . /app

RUN pip install --trusted-host pypi.python.org -r requirements.txt
EXPOSE 80
CMD python ./app.py
```

Dockerfile includes six major steps: FROM, WORKDIR, COPY, RUN, EXPOSE, CMD. The steps instruct docker to know which programming language and version to use, temporary working directory, package dependencies, port to use, and the python main script to run.

19.5.3.5 requirements.txt

This file contains all the necessary packages to be part of building the docker image.

The four packages that are required for Flask API app are:

```
flask  
sklearn  
xgboost  
pandas
```

19.5.3.6 app.py

app.py is the main app that utilizes flask API to take JSON file data from API **POST** command and apply the machine learning model to output prediction into a JSON file.

19.5.3.7 Other python codes

- preprocessing.py: python code that reads raw data and cleans the data into a good form to feed into machine learning algorithms
- to_json.py: python code to convert file from dataframe format to JSON format
- train_data.py: python code to train and build XGBoost model and output the model for the credit flask API app
- Python notebooks for visualization and evaluation:
 - 00-Visualization-Notebook
 - 01-Evaluation-Notebook

19.5.4 Code Running Instruction

19.5.4.1 Environment and Files Preparation

After all the prerequisites are met and the Ubuntu server is up and running, the following steps can be used to reproduce the environment and files preparation process starting at the directory that the **Makefile** is in (to output **make** command and runtime status into a log file, append `2>&1 | sudo tee report.log` next to every make command, e.g. `make clean 2>&1 | sudo tee report.log`)

Step 1: Environment preparation

Option A: run one command to update, upgrade and install all required software and packages:

```
make prepare-environment
```

Option B: run smaller commands in the right order listed if there is a connection failure during Option A

To update and upgrade Ubuntu environment:

```
make update-environment
```

To install other packages:

```
make packages-install
```

To install docker:

```
make docker-install
```

Step 2: Move `kaggle.json` file downloaded as prerequisite earlier into `~/.kaggle` folder. If the folder does not exist, either create one or run:

```
kaggle
```

An error will occur and the folder will be generated. Also, it is good to change the file permission to 660 by running **`bash`** command:

```
$ sudo chmod 660 ~/.kaggle/kaggle.json
```

Step 3: File and model preparation

Option A: Run one **`make`** command to achieve all:

```
make file-prep
```

Option B: Run the following **`make`** command in the correct order:

To download files:

```
make download-file
```

To preprocess files:

```
make preprocessing-files
```

To prepare a 50-records JSON test file:

```
make prepare-test-file
```

To create XGBoost Balanced Dataset model:

```
make create-model
```

19.5.4.2 Analysis (Optional)

Run the following command:

```
sudo jupyter notebook
```

A web browser should pop up to direct to the GUI of Jupyter Notebook.

19.5.4.3 Deployment

Step 1: Build a docker image

```
make docker-build-image
```

Step 2: Run docker image

```
make docker-run-image
```

19.5.4.4 Test Run Application

POST JSON test data to API Flask app

In a new terminal, run the following command:

```
make post-test-data
```

The application will return the label of **0** or **1** along with the correct **ID** to indicate whether the user ID will be more likely to experience financial distress in the next two years (1) or not (0).

19.5.4.5 Clean Up

The following command to clean up files after the deployment:

```
make clean
```

To stop docker service, run:

```
make docker-stop
```

19.6 RESULTS

All benchmarks were performed on:

Local Desktop - VMWare running Ubuntu 18.10 with 2GB Memory and 1 Processor

Local Laptop - VMWare running Ubuntu 18.10 with 8GB Memory and 4 Processors

AWS Server - Instance Type: t2.micro (1 CPU and 1GB Memory) running Ubuntu Server 18.04 LTS (HVM), SSD Volume Type

19.6.1 Deployment Benchmarks

	Desktop	Laptop	AWS Server
prep file	23.48 secs	10.71 secs	5.28 secs
docker build	33.62 secs	1 min 13.40 secs	56.43 secs

19.6.2 Application Benchmarks

	Desktop	Laptop	AWS Server
50 records test	0.12 sec	0.06 sec	0.09 sec

19.7 LIMITATIONS

Data volume is limited due to the provided dataset is from a Kaggle competition; increasing in data volume would improve the prediction results.

Prediction results could also be improved by running and evaluating more other classification algorithms. Algorithm parameters can be enhanced to provide more customization to all algorithms to observe this specific pattern of the

training set.

In term of network and security, we did not focus on security and authentication/authorization aspect of the code, most of the ports and network are set to allow any IP access to the EC2 server.

Also, there are other applications that can be paired with Docker that have not been explored such as docker compose and Kubenetes or AWS ECS to enhance the implementation (DevOps) process and mimic a more ideal production environment.

19.8 CONCLUSION

There are many algorithms that would give different results depends on different types of training data being fit into the algorithms. Just because one algorithm performs well does not mean it will give high results for all different datasets.

There are multiple tools and software that integrate all of the machine learning algorithms and turn them into real-time and responsive decision machine engine.

With every machine learning algorithm codes, there are many components involved and it would take multiple steps and complex code to deploy from one environment to another. ***Makefile*** and ***Docker*** provides convenient deployments that only require simple command and speed up implementation in machine learning code.

19.9 ACKNOWLEDGEMENTS

The author would like to thank professor Gregor von Laszewski and his class's TAs for helping with materials, markdown writing techniques, and review of this project.

19.10 REFERENCES

20 OCR EXTRACTION IMPLEMENTATION WITH TESSERACT

Joao Paulo Leite

jleite@iu.edu

Indiana University

hid: fa18-523-88

github: [blue icon](#)

code: [blue icon](#)

Keywords: OCR, Tesseract, Python, Computer Vision

20.1 ABSTRACT

The focus is to create an extraction engine that will work well for semi-structured and unstructured documents. As the volume of these documents are ever increasing, organizations are tasked with storing and organizing these documents. An OCR tool that enables these organizations to extract valuable information from these documents is the first step in being able to analyze the data that is currently locked within these documents. To accomplish this, Google's Tesseract OCR Engine is leveraged to provide full-page OCR data. The goal is to have a configurable extraction engine that allows users to pinpoint the meta-data to be extracted and output said meta-data.

20.2 INTRODUCTION

Optical Character Recognition (OCR) technology first appeared in the 1940's and grew alongside the rise of the digital computer. It was not until the late 1950's when OCR machines became commercially available and today this technology presents itself in both hardware devices as well as software offerings [212]. OCR is the first step in enabling the extraction of actionable data by transforming print on an image(document) to machine encoded text. The analysis of the output provided by OCR engines allows for this key data to be used for downstream processes and reporting. Documents fall into three categories: structured documents, semi-structured documents and unstructured

documents.

Gartner, a leading technology analysis firm, has stated the following:

"...the amount of data stored in companies will increase by 800 percent by 2018, 80 percent of which would include unstructured data that are harder to tame and manage. The biggest challenges for companies will include: collecting, managing, storing, searching and archiving this content [213]."

As unstructured documents continues to grow, big data systems are being introduced as a solution to analyze and organize this data. As a precursor, an OCR extraction solution can extract actionable data from documents and provide structure to unstructured content.

Nuance, an OCR engine provider, has stated:

" ...most companies understand the business potential behind big data, and that they either have a strategy in place, or have at least started to evaluate various tools and technologies to help them capitalize on all that big data has to offer.

As they continue to look for new sources of meaningful data, an increasing number are now realizing that their documents may contain extremely valuable information. For example, consider an insurance company that may be sitting on reams of paper documents related to its customers. These documents are full of important information including clients' policies, earnings and other financial details, health records, job histories, family records and much more. With the right tools, this information could be analyzed to anticipate insurance events, better attract and retain customers, reduce risk and even devise strategies to minimize malpractice suits or other ways to prevent fraud [214]."

20.3 OVERVIEW OF OPTICAL CHARACTER RECOGNITION

The main principle in Optical Character Recognition (OCR) is to automatically recognize character patterns. This is accomplished by teaching the system each class of pattern that can occur and providing a set of examples for each pattern. At the time of recognition, the system performs a comparison between the unknown character provided and the previously provided examples, assigned the appropriate class to the closest match[212]. This system is designed to solely transform text on a document into machine encoded text and additional systems must be built to further extract relevant information from the document, that is to say, the process of OCR is the first step in transforming structured, semi-structured and unstructured documents into valuable and relevant information.

20.4 CONTEXT BASED EXTRACTION ENGINE

This project utilizes Google's Open Source Tesseract OCR engine to provide HOCR output that is leveraged to begin the process of extracting information from unstructured data provided by Tesseract. The extraction engine's logic works by indentifying potential candidates(data which follows a specific format) and the scoring of each candidate based on context around said candidate. At the end of this process, the candidate which obtained the highest score will be selected.

For the extraction engine, there are 8 distinct phases:

1. Image Thresholding
2. OCR Process
3. Transform HOCR Data
4. Define Candidates
5. Set Context
6. Group Context
7. Score Context
8. Output Results

20.4.1 Image Thresholding

Before submitting the image into Tesseract, image clean up is performed to create a bitonal image and to remove any noise that may be present. This process consists of three steps; standardizing image DPI, smoothing the image and

removing noise from the image [215].

Standarizing Image DPI to 300 DPI:

```
def set_dpi(path):
    image = IMG.open(path)
    len_x, wid_y = image.size
    factor = max(1, int(1800 / len_x))
    size = factor * len_x, factor * wid_y
    # size = (1800, 1800)
    image_resized = image.resize(size, IMG.ANTIALIAS)
    temp_f = tempfile.NamedTemporaryFile(delete=False, suffix='.jpg')
    temp_fn = temp_f.name
    image_resized.save(temp_fn, dpi=(300, 300))
    return temp_fn
```

Converting to Bitonal Image via Adaptive Thresholding:

```
def remove_noise(name):
    image = cv2.imread(name, 0)
    filtered = cv2.adaptiveThreshold(image.astype(np.uint8), 255, cv2.ADAPTIVE_THRESH_MEAN_C,
                                     np.ones((1, 1), np.uint8)
                                     opening = cv2.morphologyEx(filtered, cv2.MORPH_OPEN, core)
                                     closing = cv2.morphologyEx(opening, cv2.MORPH_CLOSE, core)
                                     image = smooth(image)
                                     original_image = cv2.bitwise_or(image, closing)
                                     return original_image
```

Smoothing Image:

```
def smooth(image):
    ret1, th1 = cv2.threshold(image, BINARY_THRESHOLD, 255, cv2.THRESH_BINARY)
    ret2, th2 = cv2.threshold(th1, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)
    blur = cv2.GaussianBlur(th2, (1, 1), 0)
    ret3, th3 = cv2.threshold(blur, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)
    return th3
```

20.4.2 OCR Process

Google's Tesseract OCR engine is an open source engine that has the ability to output HOCR. HOCR is an open standard of data representation for formatted text obtained from an OCR engine. This standard includes text, style, layout information, recognition confidence and other info in a XML structure.

Create HOCR data:

```
def Run(self):
    DATA = pytesseract.image_to_pdf_or_hocr(image, lang=None, config='hocr', nice=0, e
```

Sample HOCR Output:

```
<span class="ocr_line" id="line_1_7" title="bbox 110 358 1198 378; baseline 0 0; x_size 24">
    <span class="ocrx_word" id="word_1_20" title="bbox 110 358 162 372; x_wconf 95"><em>Ma
    <span class="ocrx_word" id="word_1_21" title="bbox 169 358 225 372; x_wconf 77"><em>Ne
    <span class="ocrx_word" id="word_1_22" title="bbox 846 364 905 378; x_wconf 93"><em>Inv
    <span class="ocrx_word" id="word_1_23" title="bbox 912 364 980 378; x_wconf 96"><em>nur
    <span class="ocrx_word" id="word_1_24" title="bbox 1168 363 1198 378; x_wconf 44"><em>O
</span>
```

20.4.3 Transform HOCR Data

Once the HOCR results are generated, we must transform the results into useable data for our extraction process. The first step is to target the ocrx_word data from the results and parser it into separate words. After this initial parsing is complete, we separate each individual data point within a dictionary object with the values: value, confidence, left, top, right and bottom.

Parsing HOCR results with Beautiful Soup:

```
def Run(self):
    soup = bs4.BeautifulSoup(DATA, 'html.parser')
    words = soup.find_all('span', class_='ocrx_word')
```

Creating word data structure:

```
def transform_hocr(self, words):
    # Convert HOCR to usable structure
    for x in range(len(words)):
        word[int(words[x]['id'].split('_')[2])] = {}
        word[int(words[x]['id'].split('_')[2])]['Value'] = words[x].get_text()
        word[int(words[x]['id'].split('_')[2])]['Confidence'] = words[x]['title'].split(';')
        word[int(words[x]['id'].split('_')[2])]['Left'] = words[x]['title'].split(';')[0]
        word[int(words[x]['id'].split('_')[2])]['Top'] = words[x]['title'].split(';')[0].split(
            ',')[0]
        word[int(words[x]['id'].split('_')[2])]['Right'] = words[x]['title'].split(';')[0].split(
            ',')[1]
        word[int(words[x]['id'].split('_')[2])]['Bottom'] = words[x]['title'].split(';')[0].split(
            ',')[2]
```

20.4.4 Define Candidates

After transforming the HOCR results, we use the generated word dictionary to find values that match the defined regular expression that was provided by the user. All candidates which match the regular expression are stored within the candidates dictionary object with the values: value, confidence, left, top, right

and bottom.

Finding candidates:

```
def find_candidates(self, RE_ATT):
    y = 1
    for z in RE_ATT:

        for x in range(len(word)):

            m = re.match(r' ' + z + ' ', word[x + 1]['Value'], )

            if m:
                candidates[y] = {}
                candidates[y]['Value'] = word[x + 1]['Value']
                candidates[y]['Confidence'] = word[x + 1]['Confidence']
                candidates[y]['Left'] = word[x + 1]['Left']
                candidates[y]['Top'] = word[x + 1]['Top']
                candidates[y]['Right'] = word[x + 1]['Right']
                candidates[y]['Bottom'] = word[x + 1]['Bottom']
                print(candidates[y])
                y = y + 1
```

20.4.5 Set Context

Using the location input define by the user, we will set the context of each candidate based on the proximity(top, bottom, left and right) in pixels. Each word which falls within the proper proximity is stored in the context dictionary with the values: value, candidate ,word number, confidence, left, top, right, bottom, line number and same line as candidate.

Set Context:

```
def set_context(self, candidates, word):
    line = 1
    z = 1
    for x in range(len(candidates)):

        for y in range(len(word)):

            if (int(word[y + 1]['Bottom']) > int(candidates[x + 1]['Bottom']) - float(ABOVE)
                (int(word[y + 1]['Bottom']) < int(candidates[x + 1]['Bottom']) + (float(BELOW))
                (int(word[y + 1]['Right']) > int(candidates[x + 1]['Left']) - float(LEFT)
                (int(word[y + 1]['Right']) < int(candidates[x + 1]['Left']) + (float(RIGHT))

            context[z] = {}
            context[z]['Value'] = word[y + 1]['Value']
            context[z]['Candidates'] = candidates[x + 1]['Value']
            context[z]['Word'] = str(y + 1)
            context[z]['Confidence'] = word[y + 1]['Confidence']
```

```

        context[z]['Left'] = word[y + 1]['Left']
        context[z]['Top'] = word[y + 1]['Top']
        context[z]['Right'] = word[y + 1]['Right']
        context[z]['Bottom'] = word[y + 1]['Bottom']

        if z == 1:
            context[z]['Line'] = line
        elif context[z - 1]['Bottom'] == word[y + 1]['Bottom']:
            context[z]['Line'] = line
        else:
            line = line + 1
            context[z]['Line'] = line

        if int(word[y + 1]['Bottom']) > int(candidates[x + 1]['Bottom']) - 15 and \
           int(word[y + 1]['Bottom']) < int(candidates[x + 1]['Bottom']) + 15:
            context[z]['SameLine'] = "1"
        else:
            context[z]['SameLine'] = "0"

    z = z + 1

```

20.4.6 Group Context

Once the context for each candidate has been defined, we will group the context based on proximity. If multiple context words are in sequence, we will group those so that they are arranged as a phrase.

Grouping Context:

```

def define_groupcontext(self, context):
    # TRANSFORM CONTEXT INTO GROUPED CONTEXT
    # Context words that are on the same line and in sequence are grouped together
    z = 1
    for x in range(len(context)):

        if x == 0:
            groupcontext[z] = {}
            groupcontext[z]['Value'] = context[x + 1]['Value']
            groupcontext[z]['Word'] = context[x + 1]['Word']
            groupcontext[z]['Candidates'] = context[x + 1]['Candidates']
            groupcontext[z]['Weight'] = '0'
            groupcontext[z]['Confidence'] = context[x + 1]['Confidence']
            groupcontext[z]['Left'] = context[x + 1]['Left']
            groupcontext[z]['Top'] = context[x + 1]['Top']
            groupcontext[z]['Right'] = context[x + 1]['Right']
            groupcontext[z]['Bottom'] = context[x + 1]['Bottom']
            groupcontext[z]['SameLine'] = context[x + 1]['SameLine']

        elif int(groupcontext[z]['Word']) + 1 == int(context[x + 1]['Word']):

            groupcontext[z]['Value'] = groupcontext[z]['Value'] + ' ' + context[x + 1]['Value']
            groupcontext[z]['Word'] = context[x + 1]['Word']

```

```

groupcontext[z]['Confidence'] = context[x + 1]['Confidence']
groupcontext[z]['Top'] = context[x + 1]['Top']
groupcontext[z]['Right'] = context[x + 1]['Right']
groupcontext[z]['Bottom'] = context[x + 1]['Bottom']

else:
    z = z + 1
    groupcontext[z] = {}
    groupcontext[z]['Value'] = context[x + 1]['Value']
    groupcontext[z]['Word'] = context[x + 1]['Word']
    groupcontext[z]['Candidates'] = context[x + 1]['Candidates']
    groupcontext[z]['Weight'] = '0'
    groupcontext[z]['Confidence'] = context[x + 1]['Confidence']
    groupcontext[z]['Left'] = context[x + 1]['Left']
    groupcontext[z]['Top'] = context[x + 1]['Top']
    groupcontext[z]['Right'] = context[x + 1]['Right']
    groupcontext[z]['Bottom'] = context[x + 1]['Bottom']
    groupcontext[z]['SameLine'] = context[x + 1]['SameLine']

```

20.4.7 Score Context

After grouping the context, using the context values provided by the user, we will score each grouping based on how strongly it matches the context values. We utilize a fuzzy algorithm that allows us to accommodate for any OCR errors or misspellings. The weight given to each context word is also judged based on the weighted value provided by the user. This gives the ability for the user to define which context words should carry more weight in the scoring algorithm. For grouped context that fall within the same line as the candidate, the user can define a value to be added to the overall weight.

Score Context:

```

def weightcontext(self, KW_ATT):
    # Match Context and Weighting

    for z, value in KW_ATT.items():

        for x in range(len(groupcontext)):

            groupcontext[x + 1]['Weight'] = 0

            if int(groupcontext[x + 1]['Weight']) < fuzz.WRatio(groupcontext[x + 1]['Value'],
                groupcontext[x + 1]['Weight'] = float(fuzz.WRatio(groupcontext[x + 1]['Value'],
                    value[0])) / 100

            if groupcontext[x + 1]['SameLine'] == '1':
                groupcontext[x + 1]['Weight'] = groupcontext[x + 1]['Weight'] + float(

```

20.4.8 Output Result

Outputting a resulting text file with the winning candidate as well as the entire results array. The text file name will be the same as the input image file.

Outputting Results:

```
def outputresults(self, groupcontext, fp):
    # Output Results
    for x in range(len(groupcontext)):

        if groupcontext[x + 1]['Candidates'] in results:

            if int(results[groupcontext[x + 1]['Candidates']]) < int(groupcontext[x + 1]['Weight']):
                results[groupcontext[x + 1]['Candidates']] = groupcontext[x + 1]['Weight']

        else:
            results[groupcontext[x + 1]['Candidates']] = groupcontext[x + 1]['Weight']

    if (len(results.keys()) == 0):

        f = open(fp + '.txt', 'w')
        f.write("Could not find any valid candidates")
        f.close()

    else:
        sorted_by_value = sorted(results.items(), key=lambda kv: kv[1], reverse=True)
        f = open(fp + '.txt', 'w')
        f.write("WINNING CANDIDATE (CANDIDATE , WEIGHT): " + str(sorted_by_value[0]) + "\n")
        f.write("ALL CANDIDATES: " + str(sorted_by_value))
        f.close()
```

20.5 EXAMPLE

20.5.1 Tkinter GUI

Provided with this extraction engine is a simple GUI that allows the user to input the various data points needed. Because this engine is meant to process any document, this configuration step is crucial to the success of any attempted extraction.

The steps to configure are as follows:

1. Candidate Regular Expression
 - The user can define multiple regular expressions that

represents the data which is being extracted

2. Keyword Weight Ratio

- The user will assign a weight for each of the context keywords they provide. This weight determines the strength of that keyword and directly affects the scoring algorithm.

3. Search Area Definition

- The user will assign a search area in relation to the candidate. With semi-structured documents, typically the most relevant context can be found to the left and above the candidate, but this system does allow to look to the right and below as well.

4. Same Line Weight Boost

- The user can boost the weight of keywords which are found on the same line as the candidate.

5. Context Keyword Definition

- The user can add a list of keywords that will be used to score the context found around candidates.

6. Run

- When the user presses the run button, they will be prompted to select an image to be processed. Once complete, a prompt will appear and a results text file will have been generated in the same directory as the .py script.

Figure 185 This is an example of the GUI screen.

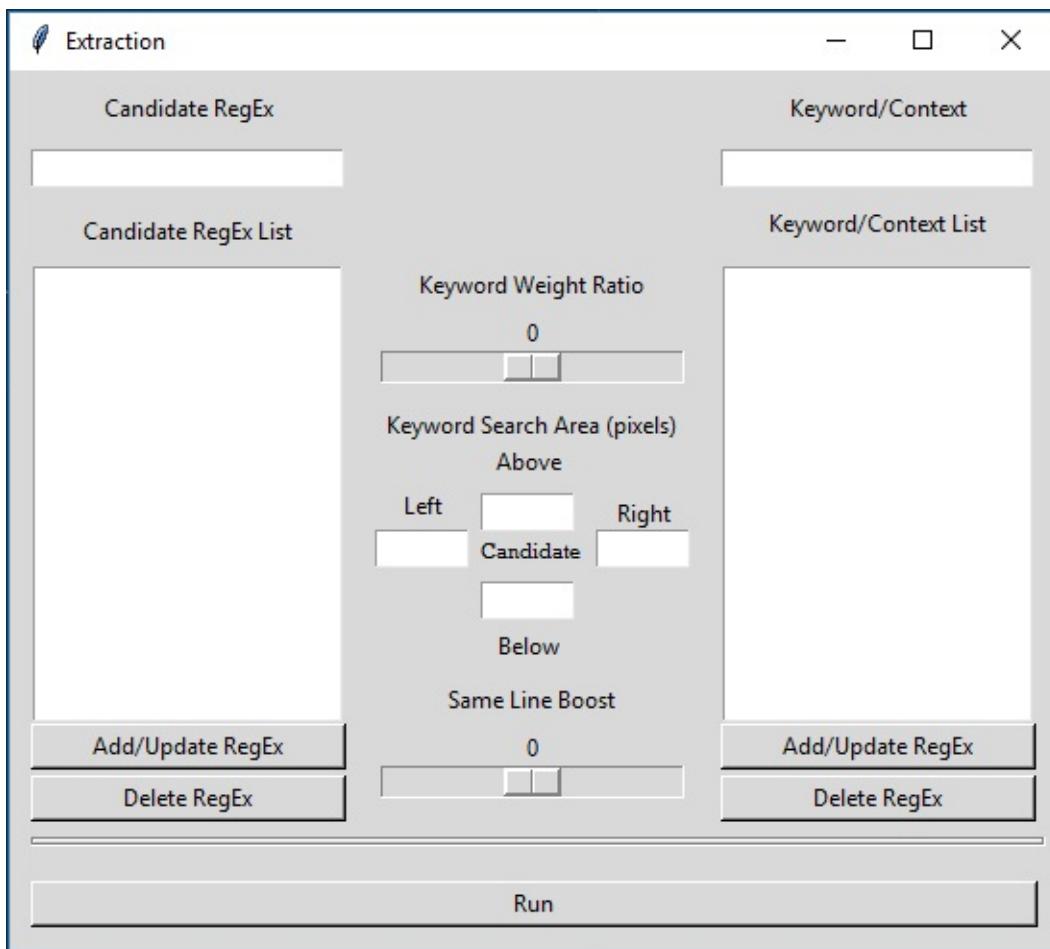


Figure 185: alt text

20.5.2 Sample Images

Provide in the project file under images are two invoice documents that were obtained online. These two invoice images are from different companies and have different context and layouts.

Figure [186](#) This is the first Invoice image.

INVOICE



My Company Ltd., 486 Jasmin Drive, City AX1 BC2, United Kingdom

TO

Martin Neville
12/A Sample Street
London
SE10AA
United Kingdom

Invoice number: 009

Issue date: 28.12.2016

Due date: 11.1.2017

Reference: 009

Payment method: Transfer

Co. Reg. No.: 999123
VAT Reg. No.: GB999999973

Thank you for your business

DESCRIPTION	QTY.	UNIT PRICE (\$)	AMOUNT (\$)
Specification	13 hr	120.00	1,560.00
Design	29 hr	90.00	2,610.00
Programming	26 hr	90.00	2,340.00
SUBTOTAL:			\$6,510.00
VAT 20%:			\$1,302.00
TOTAL (USD):			\$7,812.00
TOTAL DUE (USD)			\$7,812.00

Issued by, Signature:

A handwritten signature in blue ink, appearing to read "Martin Neville".

Figure 186: alt text

Figure 187 This is the second Invoice image.

<http://mrsinvoice.com>




Your Company LLC Address 123, State, My Country P 111-222-333, F 111-222-334

BILL TO: John Doe Alpha Bravo Road 33 P: 111-222-333, F: 111-222-334 client@example.net	Invoice # 00001
SHIPPING TO: John Doe Office Office Road 38 P: 111-333-222, F: 122-222-334 office@example.net	Invoice Date 12/12/2001
	Name of Rep. Bob
	Contact Phone 101-102-103
	Payment Terms Cash on Delivery

Amount Due: \$4,170

NO	PRODUCTS / SERVICE	QUANTITY / HOURS	RATE / UNIT PRICE	AMOUNT
1	Tyre	2	\$20	\$40
2	Steering Wheel	5	\$10	\$50
3	Engine Oil	10	\$15	\$150
4	Brake Pad	24	\$1000	\$2,400
			Subtotal	\$275
			Tax (10%)	\$27.5
			Grand Total	\$302.5

Figure 187: alt text

20.5.3 Sample Configuration - Invoice Number

With these invoices in mind, we will configure the system to extract the invoice number from both using only one definition(configuration).

Figure 188 This is a sample configuration for invoice number.

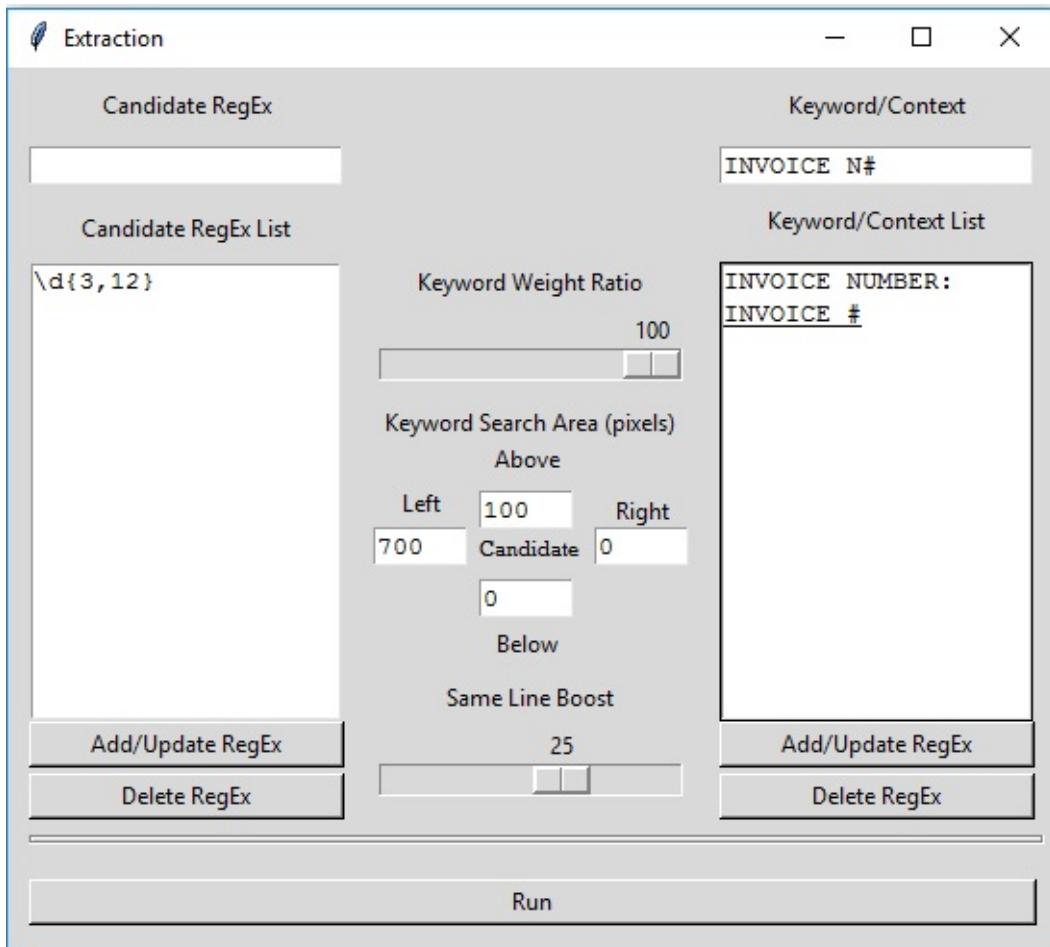


Figure 188: alt text

Output for Invoice-1:

```
WINNING CANDIDATE (CANDIDATE , WEIGHT): ('008', 125.0)
ALL CANDIDATES:
('008', 122.0)
('555666777', 61.0)
('555-987654.', 58.0)
('546516516', 58.0)
('899123', 55.0)
('120.00', 52.0)
('486', 42.0)
```

Output for Invoice-2:

```
WINNING CANDIDATE (CANDIDATE , WEIGHT): ('00001', 112.0)
ALL CANDIDATES:
('00001', 112.0)
('123,', 60.0)
('111-222-333,', 60.0)
('101-102-103', 59.0)
('111-222-334', 48.0)
('122-222-334', 45.0)
```

```
('111-333-222, ', 0)
```

20.5.4 Sample Configuration - Total Amount

With the same images, we can also configure the software to extract the total amount. With various amounts on the page as well as shared context(Subtotal vs Total), this example shows the power of the context engine.

Figure 189 This is a sample configuration for total.

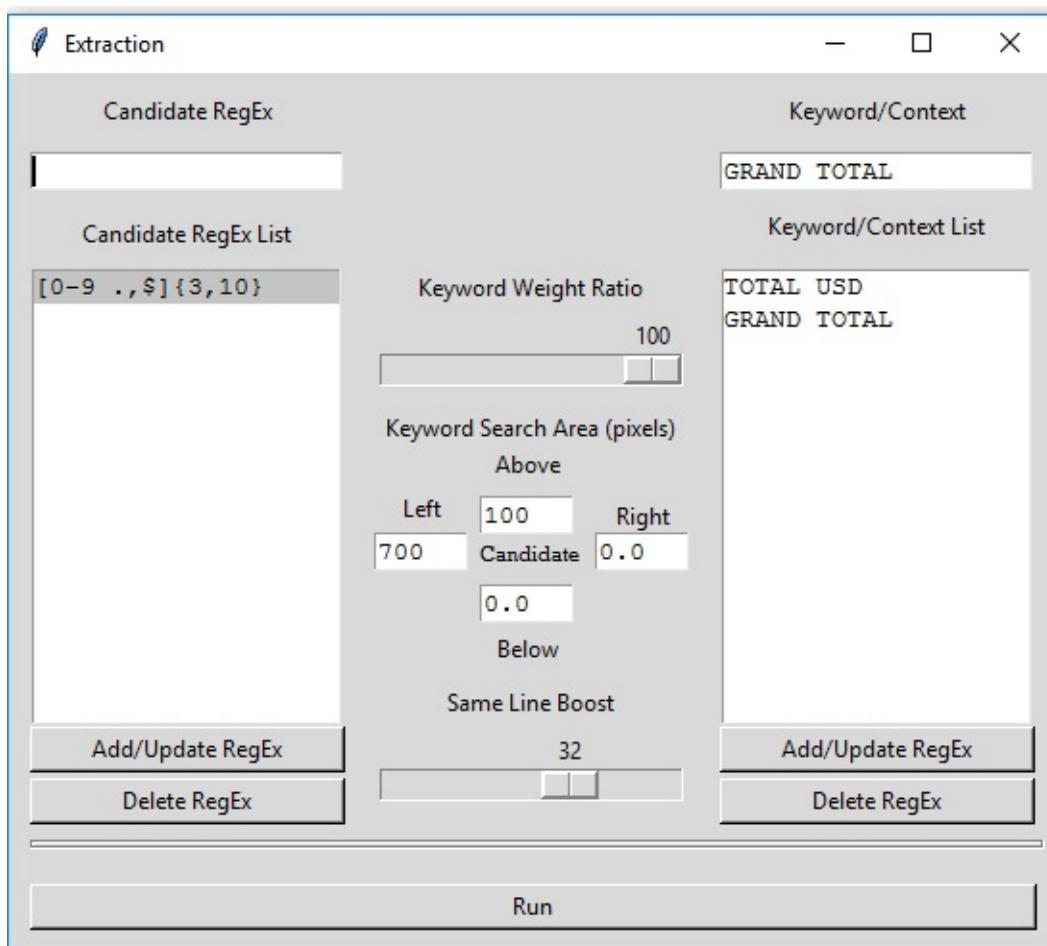


Figure 189: alt text

Output for Invoice-1:

```
WINNING CANDIDATE (CANDIDATE , WEIGHT): ('$7,812.00', 127.0)
ALL CANDIDATES:
(''$7,812.00', 127.0)
(''$6,510.00', 91.0)
(''$546516516', 82.0)
(''$555666777', 70.0)
```

```
('486', 67.0)
('11.1.2017', 65.0)
('008', 62.0)
('28.12.2018', 62.0)
('120.00', 62.0)
('$1,302.00', 62.0)
('90.00', 55.0)
('899123', 53.0)
('555-987654.', 52.0)
('1,560.00', 42.0)
('2,610.00', 42.0)
('2,340.00', 42.0)
```

Output for Invoice-2:

```
WINNING CANDIDATE (CANDIDATE , WEIGHT): ('$302.5', 132.0)
ALL CANDIDATES:
('$302.5', 132.0)
('$275', 91.0)
('$4,170', 74.0)
('111-222-333,', 70.0)
('$27.5', 70.0)
('101-102-103', 68.0)
('123,', 62.0)
('111-222-334', 62.0)
('$40', 53.0)
('$50', 53.0)
('$150', 49.0)
('00001', 44.0)
('$2,400', 44.0)
('$20', 43.0)
('$10', 43.0)
('122-222-334', 32.0)
('111-333-222,', 0)
('$15', 0)
('$1000', 0)
```

20.6 TOOLS AND TECHNOLOGY

The tools and technology deployed for this project are going to be covered in this section.

20.6.1 Installation

Requirements:

- numpy==1.12.1
- Pillow==5.3.0
- beautifulsoup4==4.6.3
- fuzzywuzzy==0.17.0
- pytesseract==0.2.5

- opencv-python==3.4.4.19

20.6.2 Terresact

Python-tesseract is an optical character recognition (OCR) tool for python. That is, it will recognize and “read” the text embedded in images. Python-tesseract is a wrapper for Google’s Tesseract-OCR Engine [216].

Code Example:

```
import pytesseract  
hocr = pytesseract.image_to_pdf_or_hocr('test.png', extension='hocr')
```

Install:

```
$ pip install pytesseract
```

20.6.3 Beautiful Soup

Beautiful Soup is a library that makes it easy to scrape information from web pages. It sits atop an HTML or XML parser, providing Pythonic idioms for iterating, searching, and modifying the parse tree [217].

Code Example:

```
import bs4  
soup = bs4.BeautifulSoup(DATA, 'html.parser')  
words = soup.find_all('span', class_='ocrx_word')
```

Install:

```
$ pip install beautifulsoup4
```

20.6.4 FuzzyWuzzy

Fuzzy Wuzzy provides fuzzy string matching in an easy to use package. It uses Levenshtein Distance to calculate the differences between sequences in a simple-to-use package [218].

Code Example:

```
from fuzzywuzzy import fuzz
```

```
fuzz.ratio("fuzzy wuzzy was a bear", "wuzzy fuzzy was a bear")  
Output:91
```

Install:

```
$ pip install fuzzywuzzy
```

20.6.5 Python

Python 3 is the high-level programming language that was used to develop this project.

20.6.6 Numpy

NumPy is the fundamental package for scientific computing with Python [219].

Code Example:

```
import numpy as np  
core = np.ones((1, 1), np.uint8)
```

Install:

```
$ pip install Numpy
```

20.6.7 OpenCV

OpenCV (Open Source Computer Vision Library) is released under a BSD license and hence it's free for both academic and commercial use. It has C++, Python and Java interfaces and supports Windows, Linux, Mac OS, iOS and Android. OpenCV was designed for computational efficiency and with a strong focus on real-time applications [220].

Code Example:

```
import cv2  
img = cv2.imread('messi5.jpg', 0)
```

Install:

```
$ pip install opencv-python
```

20.6.8 Python Imaging Library

The Python Imaging Library adds image processing capabilities to your Python interpreter. This library provides extensive file format support, an efficient internal representation, and fairly powerful image processing capabilities [221].

Code Example:

```
from PIL import Image as IMG  
image = IMG.open(path)
```

Install:

```
$ pip install Pillow
```

20.6.9 Tkinter

Tkinter is Python's a standard GUI (Graphical User Interface) package. It is a thin object-oriented layer on top of Tcl/Tk ???.

Install:

```
$ pip install Tkinter
```

20.7 CONCLUSION

The context based extraction approach that was implemented offers a way to gain insight into the data that is stored within semi-structured and unstructured documents. Improvements can be made in the future with machine learning techniques to move away from user-defined context and instead utilize training sets to provide the context necessary.

20.8 ACKNOWLEDGEMENT

The authors would like to thank the Big Data Applications and Analytics (I-523) course teaching staff, mainly professor Gregor von Laszewski for their support and guidance during this project.

21 AUTOMATION ON DRUG INTERACTIONS PROFILING

Yixing Hu, Kelvin Liuwie, Chandler Mick, Omkar Tamhankar
yixihu@gmail.com, kliuwie@gmail.com, chmick@iu.edu, otamhank@iu.edu
Indiana University
hid: fa18-423-02 fa18-423-03 fa18-423-05 fa18-423-06
github: [link](#)
code: [link](#)

Learning Objectives

- Creating a program that can parse and query large data sets
 - Exploring various “free” cloud servers to run our program
 - Evaluating which cloud server provides the optimal working efficiency
 - Learning how to use cloud computing
-

Keywords: Chameleon Cloud, Microsoft Azure, Amazon Web Services, EC2, Cloud, Drugs Interactions, FAERS, Python, Ubuntu

21.1 INTRODUCTION

We created a Python program that took raw data from the United States Food and Drug Administration’s Adverse Event Reporting System (FAERS) and filtered out important components for a user to query the information simply. The main issue that we noticed was that the important information that the Food and Drug Administration (FDA) was collecting was in large datasets that incredibly difficult to breakdown analyze for the average users of the information. Users of this information could be regulators, doctors, or even concerned consumers. Therefore, we believe that our program simplifies the analysis of these big data sets that are provided by the FDA and makes them fit for spreadsheets, such as Excel, for more in-depth review of the drug data.

One of the most important issues in the pharmaceutical industry today is determining the quality of a drug on the market [222]. Currently a

pharmaceutical company needs only prove in human clinical trials that a drug is safe when taken in isolation. However, humans are not basic research subjects and are often taking a number of medications. As the state of the healthcare system continues to fall victim to political cycles, it is always important to make sure that information on the safety and effects of drugs on the market need to be readily and easily available for review. We hope that our project represents an example of what could be implemented in the marketplace to present the information to those who need to be aware of the effects of the drugs that they are interacting with.

21.2 DATASET

The dataset is an Extensible Markup Language (XML) file from the FAERS on the FDA's website. FAERS is a collection of reports from doctors, nurses, and patients who have reported to the FDA side-effects of using certain commercial drugs [223]. This data is used by the FDA to formulate warnings and regulations against drugs to warn consumers of the potential side effects that come about from using pharmaceutical drugs.

The publicly available XML file is a 112 MB file with over 16 million rows of data. Therefore, the file cannot be easily evaluated and assessed without creating some sort of program to trim out the unnecessary data and store the valuable data for running the query. The data that we analyzed with our program covers the months of July 2018 to September 2018. However, a user would most likely want to analyze data from previous quarters, which is possible with our program.

The XML file contains 67 tags for each “adverse event report.” An adverse event report is a negative side effect to a drug from anything including migraines to something as serious as death. Every tag includes information about the patient, the side effect, the drug, treatment duration, etc. For this project, we decided the most important tags that would want to be accessed would be the reactionmeddrapt, medicinalproduct, and activesubstancename. These relate to the reactions from the drug(s), the name of the drug, and the substances in the drug, respectively.

21.3 IMPLEMENTATION

21.3.1 Code Structure

Figure 190 depicts the workflow of our program as a whole. `compile.py` is taking the quarterly data from the website and parsing the data and compiling the data into a .csv file. Then, `query.py` is allowing a user to query the data that they are looking for from the database.

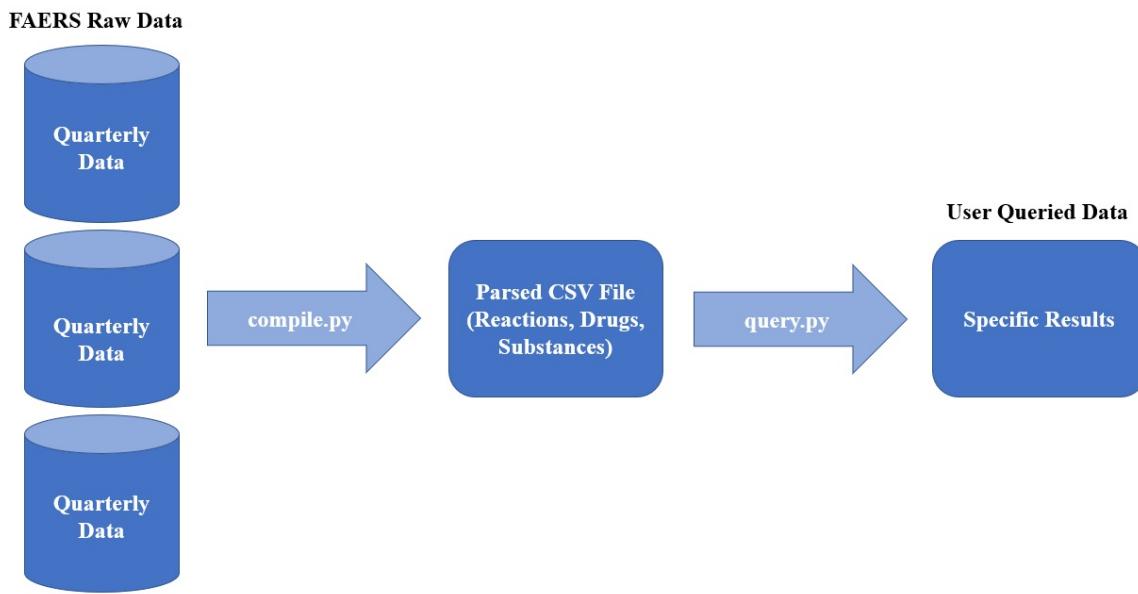


Figure 190: This image depicts the workflow of our program

21.3.1.1 Code README file

To access our program's README file and gain an understanding of the look and process of our code, go to [this website](#).

21.3.1.2 parser.py

Our `parser.py` file takes an XML file containing the initial FAERS raw data from the website and converts it to a dataframe format to be stored in a CSV file. This initial parse separates the three tags that we are looking to store for our program (`reactionmeddrapt`, `medicinalproduct`, and `activesubstancename`). The `parser.py` program flows into the `compile.py` program where the dataframe format allows the data to be easily manipulated.

21.3.1.3 compile.py

Our compile.py file takes an XML file containing the initial FAERS raw data from the website and converts the selected data to a CSV file. Using the parsing.py function, the function takes our dataframe data and outputs the three tags that we used in our project. This program takes the original, large dataset and forms a smaller, compact dataset with the data that we deemed important.

21.3.1.4 query.py

Our query.py file takes the compiled CSV file and allows a user to return certain data based on inputted values. These inputted values can be certain drugs, substances within the drugs, or the reactions of the drugs. This user queried data can be used by the user to evaluate the specific results that they are looking for. This CSV file can be transferred to a spreadsheet for simple data analysis.

The main purpose behind this code is to allow healthcare practitioners to be able to find patients who have had reported adverse side effects to drug or substance combinations similar to their patient. For example, if a patient were to have a history of intense fatigue and headaches while taking Crestor and Repatha, a healthcare practitioner could use our query to search for a drug combination that does not have the reported adverse side effects of headaches and fatigue yet combats the same target health issue.

21.3.2 Technologies Used

21.3.2.1 Python Packages

- pip: package manager that is used to download pandas. Ensure that the most recent version is downloaded.
 - `pip install --upgrade pip`
- pandas: Python package that allows our program to put our data in datatable format.
 - **In AWS:** `pip3 install pandas --user`
 - **In Chameleon Cloud:** `pip3 install pandas --user`
 - **In Azure:** `pip3 install pandas`

21.3.2.2 Amazon Web Services (AWS) EC2

AWS EC2 [224] is a cloud service that acted as a virtual machine that stored, ran, and transferred our code and its results. A free trial version with limited performance features was used to test our code.

21.3.2.3 Chameleon Cloud

Chameleon Cloud [225] is a cloud service that acted as a virtual machine that stored, ran, and transferred our code and its results. This cloud compute was provided to us for free for academic purposes by the National Science Institute.

21.3.2.4 Microsoft Azure

Microsoft Azure [226] is a cloud service that acted as a virtual machine that stored, ran, and transferred our code and its results. A free trial version with limited performance features used to test our code.

21.3.2.5 Cyberduck

Cyberduck [227] is a FTP and SFTP service that was used to transfer our program files from our local machine to the virtual machines. Cyberduck is free with any platform.

21.3.2.6 Ubuntu 18.04

Ubuntu 18.04 [228] is the operating system that we ran all of our program code on through the virtual machine and a local machine.

21.3.2.7 Mac Operating System (OS)

Mac OS [229] is the operating system that we ran on our local machines to test the program code.

21.3.3 Prerequisites

To run our program, the following would be needed to use the technologies mentioned above.

21.3.3.1 AWS Account

An AWS account is necessary to run their EC2 server. To get an account, go to [this website](#) and follow the instructions. The free trial version was used for our tests.

21.3.3.2 Chameleon Cloud Account

A Chameleon Cloud account is necessary to access their server. To get an account, go to [this website](#) and sign up for the free service. However, you will need PI Eligibility or have the permission of a PI. A PI must be a researcher or faculty of an academic institution, part of a federal agency, or related to an independent research facility associated with educational purposes. Essentially, the server is meant to provide free access for educational and development purposes, as opposed to for-profit purposes.

21.3.3.3 Microsoft Azure Account

An Azure account is necessary to access their cloud server. To get an account, go to [this website](#). Initial access is free, but Microsoft incorporates a “pay as you go” structure to provide users with greater needs of computing power access if necessary.

21.3.4 Architecture

Overall, the architecture of our tests is depicted in Figure [191](#). Data files and program code were loaded to the virtual machines through Cyberduck. Then, AWS EC2, Chameleon Cloud, and Microsoft Azure were tested to find out which service offered the best environment to run our program code efficiently. Then, this data was accessed by the user through a local machine using console commands.

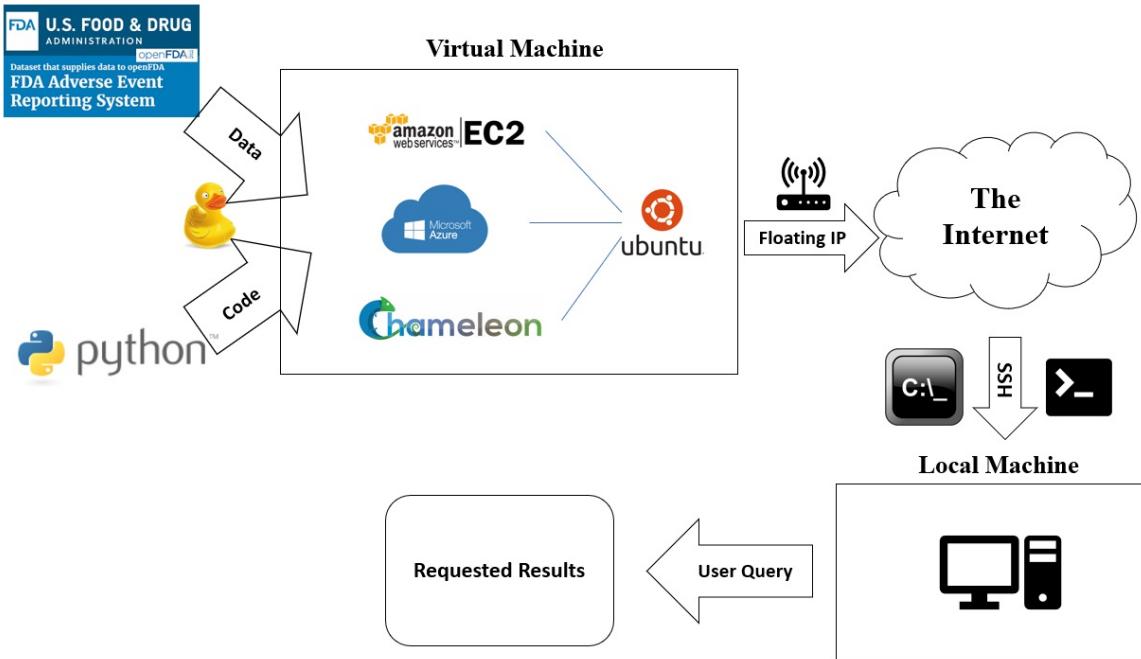


Figure 191: This image depicts the architecture of our tests

21.4 RESULTS

21.4.1 Data Visualization

21.4.1.1 Machine Information and Power

Figure [192](#) summarizes the power of the machines that we tested.

Machine Name	RAM	Processor	Operation System
iMac 2015	16 GB	1.6 GHz Intel Core i5	MacOS HighSierra 10.13.6 (17G3025)
Chameleon Cloud Instance: fa18-423-03	8 GB	Intel Xeon CPU E5-2650	Ubuntu 16.04
Chameleon Cloud Instance: fa18-423-06	16 GB	Intel Xeon CPU E5-2650	Ubuntu 16.04
Terrans Force T5	16 GB	4.00 GHz Intel Core i7	Windows 10 Professional

Figure 192: Machine information data table

21.4.1.2 FAERS Data files

Figure [193](#) reveals the sizes of the XML files taken from the FAERS website.

Data Files Processed	
.xml File Name	File Size
1_ADR18Q3	681.5 MB
2_ADR18Q3	634.9 MB
3_ADR18Q3	661.6 MB

Figure 193: FAERS Data Files

21.4.1.3 compile.py Run Times

Figure [194](#) lays out the performance of each of the tested machines compared by their processing speed on the compile.py program.

compile.py Results	
Machine Name	Compile Run Time (s)
iMac 2015	161.288
fa18-423-03 Instance	189.22
fa18-423-06 Instance	171.918
Terrans Force T5	59.8858

Figure 194: Compile Data Set

Figure [195](#) graphically analyzes the data presented above, showing the Terrans Force local machine as a clear winner.

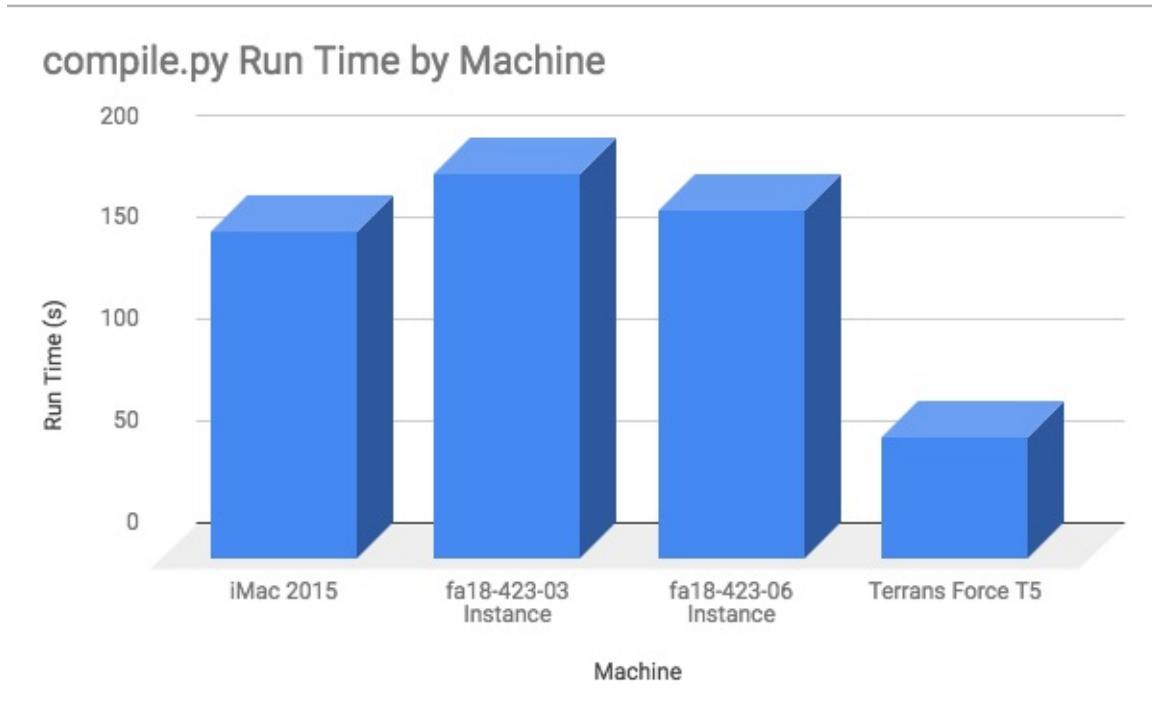


Figure 195: Compile Time Chart

21.4.1.4 query.py Run Times

Figure [196](#) lays out the performance of each of the tested machines compared by

their processing speed on the query.py program.

query.py Results						
Queries Run	Resulting File Size (MB)	iMac 2015 Run Time (s)	fa18-423-03 Instance Run Time (s)	fa18-423-06 Instance Run Time (s)	Terrans Force T5 Run Time (s)	
1-DRUG-HUMIRA	2.9	19.41563		21.0688	20.1966	10.1492
1-SUBSTANCE-CALCIUM	1.8	18.0769		19.4326	18.5107	9.30793
1-REACTION-HEADACHE	4.1	20.044		22.3272	21.5999	10.4781
2-REACTION-HEADACHE_FATIGUE	1.4	17.5384		17.4173	16.2001	9.31584
2-SUBSTANCE-AMOXICILLIN_IBUPROFEN	0.19	16.976		16.7262	16.8246	8.55467
2-DRUG-CRESTOR_LUTATHERA	0.001	17.7784		17.7807	15.4248	8.34523
3-SUBSTANCE-AMOXICILLIN_IBUPROFEN_ASPIRIN	0.071	17.4185		18.324	18.1837	8.73495
3-REACTION-HEADACHE_SWELLING_FEELING COLD	0.009	16.7977		16.5282	16.8835	8.68725
3-DRUG-NEURONTIN_ALEMTUZUMAB_ACICLOVIR	0.001	16.7925		16.5576	16.8368	8.456

Figure 196: Query Data Set

Figure 197 graphically analyzes the data presented above, showing the Terrans Force local machine as a clear winner.

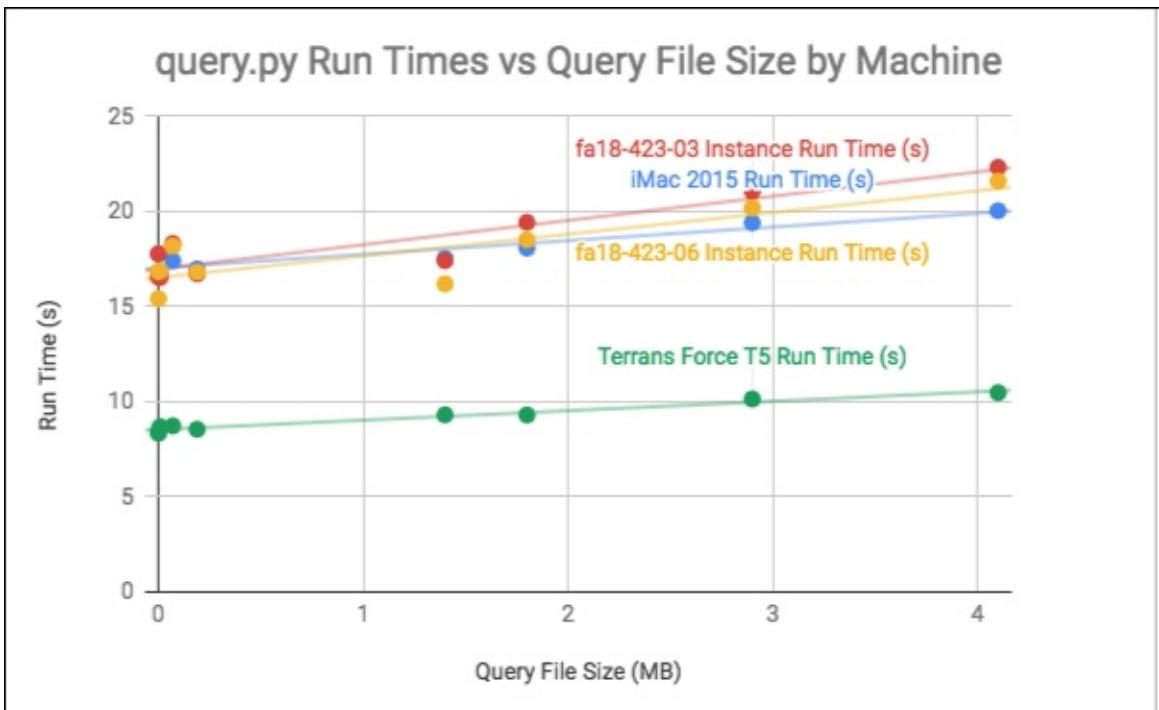


Figure 197: Query Time Chart

21.4.2 Discussion

As shown by our results, there are a few similarities and differences to note when deciding what platform to use when executing our program. The Terrans Force T5 computer by far outperformed all the other machines both in run times for compile.py and query.py. Notably, the Terrans Force T5 outperformed the other three machines on every single query execution. Two of the other machines had the same amount of RAM as the T5, therefore its superior run times may be on account of the more powerful Intel processor that it held. The

other three machines exhibited similar run times. Interestingly, the two instances we ran on Chameleon Cloud had 8 GB of RAM and 16 GB of RAM, respectively, but there was no significant difference in run time between them.

Although, these results are difficult to extrapolate to all computer forms, based on our results, our program runs best on local machines that have both high RAM and powerful processors. However, users may not want to take up their own hard disk space with these large data files, therefore we recommend running on a virtual machine with minimum 8 GB of RAM.

21.4.2.1 AWS Error

Figure [198](#) depicts the error message that comes from trying to initiate the compile.py program in AWS EC2.

```
[ec2-user@ip-172-31-44-177 ~]$ python3 compiler.py
This must be an initial run
Traceback (most recent call last):
  File "compiler.py", line 26, in <module>
    new_data = parsing(file)
  File "/home/ec2-user/parser.py", line 13, in parsing
    for event, elem in ET.iterparse(file_name, events=('start','end')):
  File "/usr/lib64/python3.7/xml/etree/ElementTree.py", line 1227, in iterator
    pullparser.feed(data)
  File "/usr/lib64/python3.7/xml/etree/ElementTree.py", line 1269, in feed
    self._parser.feed(data)
MemoryError
```

Figure 198: AWS Error

21.4.2.2 Azure Error

Figure [199](#) depicts the error message that comes from trying to initiate the compile.py program in Microsoft Azure.

```
yixihu@BigData1:~/bigdata$ ls
1_ADR18Q3.xml  compile.py      parser.py
__pycache__    faers_xml_2018q3.zip query.py
yixihu@BigData1:~/bigdata$ python3 compile.py
This must be an initial run
Traceback (most recent call last):
  File "compile.py", line 29, in <module>
    new_data = parser1.parsing(file)
NameError: name 'parser1' is not defined
yixihu@BigData1:~/bigdata$ nano compile.py
yixihu@BigData1:~/bigdata$ python3 compile.py
This must be an initial run
Traceback (most recent call last):
  File "compile.py", line 29, in <module>
    new_data = parsing(file)
  File "/home/yixihu/bigdata/parser.py", line 6, in parsing
    tree = ET.parse (file_name)
  File "/usr/lib/python3.6/xml/etree/ElementTree.py", line 1196, in parse
    File "/usr/lib/python3.6/xml/etree/ElementTree.py", line 597, in parse
MemoryError
Error in sys.excepthook:
Traceback (most recent call last):
  File "/usr/lib/python3/dist-packages/apport_python_hook.py", line 63, in apport_excepthook
    from apport.fileutils import likely_packaged, get_recent_crashes
  File "/usr/lib/python3/dist-packages/apport/__init__.py", line 5, in <module>
    from apport.report import Report
  File "<frozen importlib._bootstrap>", line 971, in _find_and_load
  File "<frozen importlib._bootstrap>", line 955, in _find_and_load_unlocked
  File "<frozen importlib._bootstrap>", line 665, in _load_unlocked
  File "<frozen importlib._bootstrap_external>", line 674, in exec_module
  File "<frozen importlib._bootstrap_external>", line 771, in get_code
  File "<frozen importlib._bootstrap_external>", line 482, in _validate_bytecode_header
MemoryError

Original exception was:
Traceback (most recent call last):
  File "compile.py", line 29, in <module>
    new_data = parsing(file)
  File "/home/yixihu/bigdata/parser.py", line 6, in parsing
    tree = ET.parse (file_name)
  File "/usr/lib/python3.6/xml/etree/ElementTree.py", line 1196, in parse
    File "/usr/lib/python3.6/xml/etree/ElementTree.py", line 597, in parse
MemoryError
```

Figure 199: Azure Error

The services that were offered through Amazon Web Service and Microsoft Azure were “free trials,” where we were limited to a certain number of hours or days that we were able to access. However, this also limits the amount of computing power we able to access through their programs. AWS only allows 1 GB of memory for their free trial version, a mark that was well over the datasets that we were working with. Microsoft Azure has a similar pricing structure where you actually have different tiers for computing power (RAM) and storage space [230].

The issues with Microsoft Azure and Amazon Web Services Cloud Servers proved to be issues that were brought about by having a dataset that was too large for the free trial versions of the software. We firmly believe that our

program would be able to work with ease using our implementation procedures and spending the extra money to access a higher quality version of the same services that we tested. Services such as Azure and AWS offer a wide range of virtual machine options so it is up to the user how much they are willing to pay for speed and processing power.

21.4.3 Clinically Relevant Findings

Repatha produced by Amgen is a PCSK9 inhibitor designed to attack and inactivate PCSK9 proteins and result in lowered LDL cholesterol [231]. This drug was approved by the FDA in January 2015, therefore it is a relatively new drug for treating hyperlipidemia. According to PDR.net, Repatha has no interactions with other drugs [232]. When run our program, however, we have found that Repatha has an interaction with Pravastatin another cholesterol lowering medication. However, we did not find any adverse event reports tying Repatha and Lipitor (Atorvastatin) a different cholesterol lowering medication. Figure 200 depicts the final CSV file that a user would have access to after running our program to completion. Figure 201 shows our results tying the adverse events of patients taking both Repatha and Pravastatin, but no other drugs.

reaction	drug	substance
['Injection site pain', 'Musculoskeletal discomfort', 'Inappropriate schedule of drug administration', 'Arthralgia', 'Hypoesthesia']	['REPATHA', 'PRAVASTATIN.1']	['EVOLOCUMAB', 'PRAVASTATIN']
['Blood cholesterol abnormal']	['REPATHA', 'PRAVASTATIN.1']	['EVOLOCUMAB', 'PRAVASTATIN']
['Liver function test increased']	['REPATHA', 'PRAVASTATIN.1']	['EVOLOCUMAB', 'PRAVASTATIN']
['Joint swelling', 'Glomerular filtration rate decreased', 'Blood pressure abnormal', 'Arthralgia', 'Peripheral swelling']	['REPATHA', 'PRAVASTATIN.1']	['EVOLOCUMAB', 'PRAVASTATIN']

Figure 200: Resulting CSV file from running program

Interactions: Pravastatin and Repatha
Injection site pain
Musculoskeletal discomfort
Inappropriate schedule of drug administration
Arthralgia
Hypoesthesia
Blood cholesterol abnormal
Liver function test increased
Joint swelling
Glomerular filtration rate decreased
Blood pressure abnormal
Peripheral swelling

Figure 201: This image depicts the side effects of Repatha and Atorvastatin

PSCK9 inhibitors are generally prescribed in combination with a statin drug so healthcare providers should be aware of the possible interactions between these two classes of drugs [231]. Therefore, the novelty produced by our program is that healthcare professionals are able to take the most up-to-date adverse events published by the FDA and check for drug interactions of recently released drugs. This database in combination with our program gives healthcare professionals the power to skip the latency period for enough adverse events to occur for drug companies to relay that information to the healthcare providers. Ultimately, this can save patients from facing unnecessary interactions, whether they be major or minor side effects.

21.5 CONCLUSION

In the beginning of our process, we had a goal to find a way to take the massive datasets provided by the FDA and turn them into files that can be easily queried by a user for further data analysis. At the same time, we wanted to be able to explore the possibility and usefulness of Cloud Servers to run our program so it does not have to be limited to being accessed via a local machine. While we can firmly say that we have made a program that provides the services we set out to create, we can say to a degree of certainty that we have found that it can be ran on a cloud server. The service that we used successfully was Chameleon Cloud however for the average user, the two services that we found to be favorable were Amazon Web Service's EC2 and Microsoft Azure's Cloud Shell. In terms of usability and reliability, we believe that a paid subscription to AWS EC2 would be the better of the two.

21.6 ACKNOWLEDGMENTS

Thank you to Dr. Gregor von Laszewski for his continued help and support through our development and understanding of Big Data and cloud servers. His guidance proved essential to the culmination of our configuration and testing.

Results presented in this paper were obtained using Chameleon testbed supported by the National Science Foundation.

21.7 WORK BREAKDOWN

- Yixing Hu: Microsoft Azure Server Testing, Query Code Editor
- Kelvin Liuwie: Parsing, Compile, and Query Code Writer
- Chandler Mick: Report Writer, Chameleon Cloud Testing, Query Code Editor
- Omkar Tamhankar: AWS Server Testing, Chameleon Cloud Testing, Medical Advisor

21.8 APPENDIX

21.8.1 Chameleon Cloud

Overview:

This document explains how to access Chameleon Cloud and how it was used to run the program in the cloud server [225].

Setup

Go to <https://www.chameleoncloud.org> and sign up for the free service. However, you will need PI Eligibility or have the permission of a PI. A PI must have be a researcher or faculty of an academic institution, a part of a federal agency, or an independent research facility associated with educational purposes. Essentially, the server is meant to provide free access for educational and development purposes, as opposed to for-profit purposes.

After obtaining proper access, you are free to create an instance on the server to create your virtual machine.

Creating an Instance

When creating our instance, the easiest way to get to the instances database was to use this website:

<https://openstack.tacc.chameleoncloud.org/dashboard/project/instances/>

Here, you should see all the instances created in your project. The webpage should resemble Figure 202:

Instances

		Instance Name	Image Name	IP Address	Size	Key Pair	Status	Availability Zone	Task	Power State	Time since created	Actions
<input type="checkbox"/>	ruili-test-01	CC-CentOS7	192.168.0.207	m1.small	ruili-test-01	Active	nova	None	Running	3 days, 16 hours	<button>Create Snapshot</button>	
<input type="checkbox"/>	cm-dmdemeul-1	CC-Ubuntu16.04-20181018	192.168.0.129	m1.medium	-	Active	nova	None	Running	1 month	<button>Create Snapshot</button>	
<input type="checkbox"/>	vjoshi-3	CC-Ubuntu16.04-20180831	192.168.0.128 Floating IPs: 129.114.33.64	m1.small	macpro	Shutoff	nova	None	Shut Down	1 month, 4 weeks	<button>Start Instance</button>	
<input type="checkbox"/>	vjoshi-2	CC-Ubuntu16.04-20180831	192.168.0.127 Floating IPs: 129.114.111.160	m1.small	macpro	Shutoff	nova	None	Shut Down	1 month, 4 weeks	<button>Start Instance</button>	
<input type="checkbox"/>	vjoshi-1	CC-Ubuntu16.04-20180831	192.168.0.126 Floating IPs: 129.114.111.160	m1.small	macpro	Shutoff	nova	None	Shut Down	1 month, 4 weeks	<button>Start Instance</button>	

Figure 202: InstanceDatabase

From here, you must select the “Launch Instance” button. This should bring up a webpage that resembles Figure 203. Here, you must enter a name for your instance. We used a group member’s HID as an example (fa18-423-06). Flavor describes the amount of resources that will be allocated to the project. Resources such as RAM, disk space, etc. It is important to have at least 16 GB of RAM to run our project.py program because of the size of the files that are being processed. The “Instance Boot Source” must be set to “Boot from image.” We used Ubuntu 16.04 for our “Image Name” due to the ease of running commands through the terminal later.

Launch Instance

Details * Access & Security Networking * Post-Creation Advanced Options

Availability Zone
nova

Instance Name *
fa18-423-06

Flavor * ⓘ
m1.xlarge

Instance Count * ⓘ
1

Instance Boot Source * ⓘ
Boot from image

Image Name *
CC-Ubuntu16.04 (696.2 MB)

Specify the details for launching an instance.
The chart below shows the resources used by this project in relation to the project's quotas.

Flavor Details

Name	m1.xlarge
VCPUs	8
Root Disk	160 GB
Ephemeral Disk	0 GB
Total Disk	160 GB
RAM	16,384 MB

Project Limits

Number of Instances	6 of 100 Used
Number of VCPUs	7 of 200 Used

Figure 203: InstanceDetails

Next, click on the “Access & Security” tab. Here, you want to create a key pair to link the virtual machine to your computer. To do this, you must click on the “+” button next to the drop down. You should get a pop up that looks like Figure [204](#):

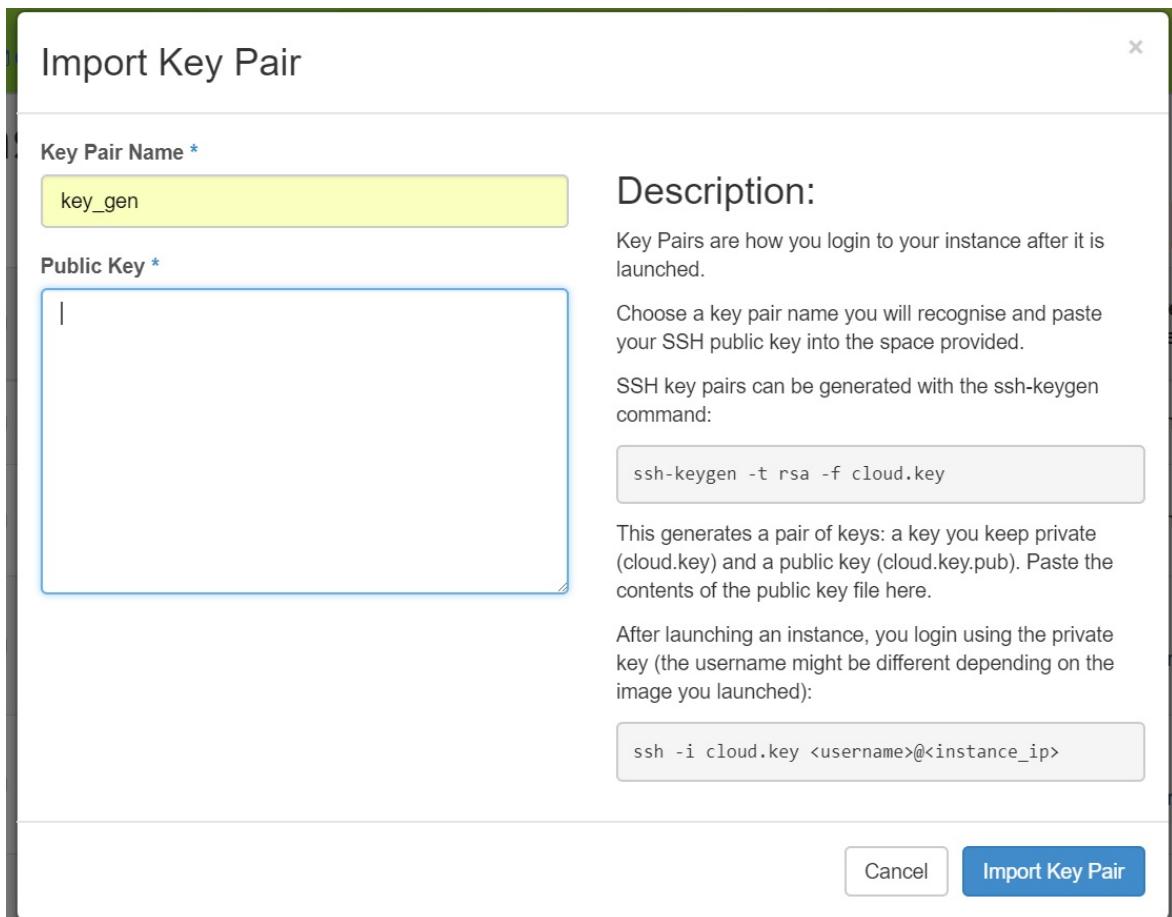


Figure 204: KeyPair

You can name the key however you like. To generate and access your public key, enter the following commands on Terminal:

```
ssh-keygen -t rsa -f <insert your key name>
```

This should result in a file being created containing information on your public and private keys. To access the public key enter the following command:

```
$ cat ~/.ssh/<public key file name>
```

Copy and paste the returned key value into the text box labeled "Public Key" in Chameleon Cloud. Now, you are ready to launch the instance. So, click the "Launch Instance" button on the bottom of the dialog box. The instance should now appear in the database and take a few minutes to spawn and become active. Now, you must associate a floating IP address with the instance to access the instance. So, click on the "Associate Floating IP" button (see Figure 205).

Instances

		Instance Name	Image Name	IP Address	Size	Key Pair	Status	Availability Zone	Task	Power State	Time since created	Actions
<input type="checkbox"/>	fa18-423-06	CC-Ubuntu16.04	192.168.0.214	m1.xlarge	key_gen	Active	nova		None	Running	0 minutes	<button>Create Snapshot</button>
<input type="checkbox"/>	ruili-test-01	CC-CentOS7	192.168.0.207	m1.small	ruili-test-01	Active	nova		None	Running	3 days, 16 hours	Associate Floating IP Disassociate Floating IP
<input type="checkbox"/>	cm-dmdemeul-1	CC-Ubuntu16.04-20181018	192.168.0.129	m1.medium	-	Active	nova		None	Running	1 month	Edit Instance Edit Security Groups Console View Log Pause Instance Suspend Instance Resize Instance Lock Instance Unlock Instance Soft Reboot Instance
<input type="checkbox"/>	vjoshi-3	CC-Ubuntu16.04-20180831	192.168.0.128 Floating IPs: 129.114.33.64	m1.small	macpro	Shutoff	nova		None	Shut Down	1 month, 4 weeks	
<input type="checkbox"/>	vjoshi-2	CC-Ubuntu16.04-20180831	192.168.0.127 Floating IPs: 129.114.111.160	m1.small	macpro	Shutoff	nova		None	Shut Down	1 month, 4 weeks	

Figure 205: Associating Floating IP

Here, you may either add a custom IP address, or just use one of the ones given in the drop down. We chose the latter. Either way, you must remember this IP address to later access the instance through commands in Terminal. Then, leave the “Port to be associated” section be. Now, you are ready to associate the IP address so click the button to do so. Now, the instance is created and ready to run our program!

21.8.2 Amazon Web Services (AWS) EC2 Usage

Overview: This document explains how to access AWS EC2 and how it was used to run the program in the cloud server [224].

Setup Go to <https://aws.amazon.com/>. Click on “Sign In to the Console”. If you do not already have an account set up, create an account. In the “AWS services” search bar, search “EC2” and click on the EC2 option. Under “Create Instance” click “Launch Instance”. Click the “Select” button next to “Amazon Linux 2 AMI (HVM), SSD Volume Type”. Click on “Review and Launch” then “Launch”. In the drop-down menu select “Create a new key pair”, type in a key pair name, click “Download Key Pair”, select the check box below, and click “Launch Instance”. In the green box at the top of the screen click on Instance ID link, this will open the instance.

Accessing the instance Open command prompt, if using Windows, or Terminal, if using Mac. Type in:

```
ssh -i ~/key-pair-location* ec2-user@IPV4 Public IP*
```

This will connect you into the EC2 instance.

Uploading files into the instance Download Cyberduck 2. Open the application. In the dropdown menu, select “SFTP (SSH File Transfer Protocol)”. In the Server field, enter the Public DNS. In the Username field enter “ec2-user” [233]. In the SSH Private Key field select “Choose” from the dropdown menu and select your .pem key pair file. Click on “Connect”. Once connected through Cyberduck, you can simply drag and drop files into the instance.

Disclaimer Our .xml files were greater than 1 GB in size. AWS EC2’s free tier only offers up to 1 GB memory therefore we would receive memory errors when running our python files on the server.

21.8.3 Microsoft Azure Cloud Shell Usage

Overview: This document explains how to access Microsoft Azure and how Azure is used to run programs in the cloud server [226].

Setup Go to <https://portal.azure.com/> to register an Azure account. Once you get an account, you can go to Azure Portal to access all the tools you need. The search bar at the very top could help you access any resource you need in Azure.

Create a Python web app through Azure app services

First, clone the folder containing code files and data files from Github with Bash/Terminal in your local machine. Before further operation, run the Python files locally to check if the program runs correctly. Then, go back to the Azure portal which was accessed previously. There, you will open the Cloud Shell for the rest of your operations. The very first step in the Cloud Shell is to create a deployment user, if you do not already have one. The command should read:

```
az webapp deployment user set --user-name <username> --password <password>
```

where `<username>` and `<password>` are replaced with the username and password you are planning to use. After this command is ran successfully, there should be a

JSON output with your password shown as null. The username and the password should be recorded for future use. After that, you have to create a resource group to make it easier to manage, by using the command:

```
az group create --name myResourceGroup --location "East US"
```

For this command, “East US” could be replaced by other regions where Azure is available, but for the connection stability, it would be best to use the closest region. Then, an Azure App Service plan should be created. The command should read:

```
az appservice plan create --name myAppServicePlan --resource-group myResourceGroup --sku B1 --is-linux
```

When those steps are done, it is the time to create the web app. Still working on the Azure Cloud Shell, type the following command lines:

```
az webapp create --resource-group myResourceGroup --plan myAppServicePlan --name <app_name> --runtime "PYTHON|3.7" --deployment-local-git
```

These command lines should return an output starting with:

```
Local git is configured with url of  
'https://<username>@<app_name>.scm.azurewebsites.net/<app_name>.git'
```

This URL of git should be kept as you will need it later for connection.

After those steps, go back to local bash for the following operations. The first step is to add an Azure remote to the local Git repository by using the command:

```
git remote add azure <deploymentLocalGitUrl-from-create-step>
```

And then use:

```
git push azure master
```

to push to Azure from Git repository. This command should take a bit longer to process. Once the processing is done, you can go back to the Azure Portal to access the App Services to find the one being created. Click on that service and go to the side bar to click on the SSH under Development Tools. Then, you can access all the files you have pushed onto the cloud server.

22 SCALABLE MICROSERVICES TO LABEL YELP IMAGES USING KUBERENETS

Keerthi Naredla

knaredla@iu.edu

Indiana University Bloomington

hid: sp18-616-02 but in fa18 hid-sp18-602

github: [cloudnative](#)

code: [cloudnative](#)

Keywords: docker, vision, kubernetes, yelp, kafka

22.1 ABSTRACT

Automated Machine Learning is the emerging technology now from startups to major tech firms. One of the most popularly used in the era of AutoML is Google Cloud AutoML which has a suite of products including Vision, NLP, Speech, Translation and so on. Again in each of these products emphasis is made on an important feature set, for instance in Cloud Vision, there are label detection, face recognition etc. This project is an application of Cloud Vision API to label images from Yelp dataset. The project is built in microservices architecture and deployed using kubernetes. The message broker Kafka is used to communicate between the microservices. This paper briefly discusses the technologies used for the implementation of the project, the project setup, results and deployment benchmarks.

22.2 INTRODUCTION

The applications are growing complex, not just functionality wise but also the data generated and used are highly increasing. This makes it absolutely necessary to break down such complex application, instead of a monolithic application so that product deployed is much more maintainable, scalable, reliable, independent of the failure of other functional components of the

application, portable, easy to deploy on different cloud platforms, and so on. Most of these suit of best practices called “Twelve-Factor Apps” can be bought into the production of Software applications using technologies such Docker, Kubernetes and Microservice architecture [234]. In this project, an application is built using a combination of these three technologies. In the below subsections, these concepts are briefly introduced to better understand the working of the application.

22.3 IMPLEMENTATION

22.3.1 Yelp Dataset

Yelp provides an open-source dataset for the challenge with students and university grads. The yelp dataset is huge of nearly 2.66 gigabytes of dataset comprised of all the text details present in yelp, this may include all kinds of business and their details. Apart from this, yelp also holds photos dataset which is 7.50 gigabytes of the compressed dataset, which are purely images of count nearly 200,000, for purposes like image analysis, apply machine learning and computer vision technologies. The text-based dataset is usually in JSON/ CSV, SQL format that can be downloaded from their website. In addition to this Yelp also gives access to their data through Yelp-Fusion.

22.3.1.1 Yelp Fusion API

On Yelp Developers blog different open source tools that leverage access to retrieve, test, implement on yelp dataset. Yelp Fusion is a REST API, which can be integrated simply by generating and importing API token key into your program. This gives access to a variety of feature data such as business, locations, reviews, images, metadata and also a specific search URL available to query on each of these features. This is very efficient on fetching data on the fly without additional storage disk space and based on your requirements steps like data scraping, cleaning, pre-processing can also be skipped. But the number of time data fetch is limited, if the data set is changes or some part of it is removed then it might affect your analysis. However, using Yelp Fusion REST API is easy and feasible with minimal system requirements, and on average no or very less downtime[235].

22.3.2 Cloud Vision API

Cloud Vision API is the most popular API that Google has till date. It is very easy and efficient to analyze the content of the image, which has state-of-the-art tools for Image detecting features like face, text, label and document text, web detection. It is further made easy to use, through Cloud AutoML suite. Using Vision AutoML, it is just one click away to upload images and run pre-determined, custom machine learning models. It is built based on Google's powerful technology of learning-to-learn, neural network architecture. In fact, building custom ML model is just a few steps [236]. First, uploading training dataset with images labeled into google bucket or human-support to label images and the ML model is trained according to the provided dataset. And then test data is passed, and accuracy of prediction, classification of test data set is determined. However, this feature of Cloud AutoML is accessible to only limited customers, but the basic feature of labeling the images such as data in Google is quite possible through REST API and are available to use in different programming languages [237].

22.3.3 Docker

Modern applications are often built using different technologies with different versions based on the application requirement. Deploying much such application on a single Operating System could be a huge risk at times especially when there is a conflict in dependencies between applications. For instance, if you consider installing 2 versions of Nginx on the same OS, there would be a conflict with the namespace, network port making it logically cumbersome to install the same software with 2 different versions. Although the concept of containerization is old, the ability to package applications with their dependencies into the container, making applications independent and isolated from other applications is huge progress to deploy applications, this is achieved with Docker container technology. Moreover, it is very easy to run multiple containers using Docker. The docker image which is a snapshot of the container is the basis to build a container. The Docker container image is a packaging format that contains all the dependencies necessary for the application along with required initial steps such as setting environment variables, or even running command to start the application. These images are hosted on public or private repository such as docker hub, google storage. There are 2 ways to build a Docker image, first is to

build Dockerfile, and the second way is to make changes on previous docker image. In most cases, a Dockerfile is built on a base image that is useful for the application and then all the required dependencies, commands are added. Once the docker image is ready it takes simple commands to build the container, push the final docker image to the storage repository, and pull the docker image whenever required. Thus, it is easy and robust to create, distribute and run applications using Docker Containers with docker images and docker command-line tools. There are less or no restrictions for docker usage, as containers can be built on any machine with Docker installed it is highly in use by DevOps [238].

22.3.4 Kubernetes

Although Docker makes it easy to deploy and run applications using container technology when it comes to application configuration, service discovery, managing updates, secrets management, and monitoring containers on the cluster, a better technology is required to leverage all of these tasks. Here comes Kubernetes, an open-source platform that provides a high level of abstraction and orchestration of containers deployed on one or more clusters, which in turn are treated as a single logical machine. Usually, a cluster has single Kubernetes master nodes that keep on running despite explicitly deleting, and zero or more worker nodes [239]. The master node is responsible for managing the cluster, whereas the worker nodes work like a VM, it consists of one or more pods, Volume, network ID and tools to handle container operations.

A pod is the smallest unit of Kubernetes and it consists of one or more containers. All the containers in the pod have shared the same filesystem and IP address, this makes the communication between containers in a pod easy. Each of these pods created based on the scheme which is usually in YAML or JSON file format. The scheme covers important aspects of spec which specifies the Pod behavior, container name, container ports. A pod without Services or Replication Controller cannot be accessed by the external client, neither scaling and distribution of the application are possible [240].

Services provide an external interface for one or more pods. The Service schema definition has 3 important parameters: kind, metadata, and spec. The kind is set to Service to indicate a Kubernetes Service, which is deployment, pod in case of Kubernetes deployment, pod definition files. The label app and the name

constitute the metadata. The spec mapping includes a ports mapping for port 80 with name HTTP. The selector is the key mapping in the spec and specifies a mapping to be used for selecting the Pods to expose via the Service. Therefore, the service diverts the network traffic to all its pods with the same label as the label selector specified in the Service spec, in a round-robin manner. There are 3 different types of Service: Load Balancer, Internal IP, Node port. If a Service type is ClusterIP, then the service is accessible only within the cluster via its internal port. Whereas if the service type is Node port then the service is accessible from outside the node port, which further routes the traffic to internal port Cluster IP of the service, that is automatically created. Similarly Load Balancer service type also automatically creates Node port and cluster IP. It gives access for the external user to ping the IP. In addition to this Load Balancer has the responsibility to balance the load between all the Pods in Service [241].

Another important aspect in scaling applications is the replication controller, which manages the replication level of pods by setting “replicas” in replication controller definition or on the command line with the –replicas parameter. This ensures that the number of pod replicas are running at any given time. If a replica fails or is stopped deliberately a new replica is started automatically. With these 2 crucial features scaling and replication factor, Kubernetes keep microservices up and running all the time. Hence, Kubernetes is production-ready, which provides dynamic container cluster orchestration in real time.

Kubernetes as a cluster manager provides the feasibility for deploying Microservices by breaking an application into smaller, manageable, scalable components that could be used by groups with different requirements; Fault-tolerant cluster in which if a single Pod replica fails (due to node failure, for example), another is started automatically; Horizontal scaling in which additional or fewer replicas of a Pod could be run by just modifying the replicas label in the Replication Controller or using the replicas parameter in the kubectl scale command;

22.3.5 Apache Kafka

Apache Kafka is an open source, real-time distributed streaming platform. It originally started as a research project at LinkedIn and later launched an open-

source project with wide-range of active contributors. There are three main use cases for Kafka such as publish, subscribe to topics od data to receive and send, Kafka streams API used for microservices that require a real-time continuous flow of records, Kafka as a Storage system can be used for multiple purposes especially logging across distributed systems would be a good implementation. Deploying Apache Kafka on kubernets has recently evolved and there is a lot of scope for scalability and high availability with kubernetes. Further, Kafka has to connect with Zookeeper service discovery concept and it is must that Zookeeper is up and running for Kafka to work as expected. And this has further benefits, as Zookeeper can be used for other microservices for their own purposes like service discovery, logging and so on.

22.4 DESIGN

The main aim of the application is to label photos from Yelp dataset retrieved on passing location and search term such as food, dinner, using cloud vision API. The application is divided into 3 microservices frontend, backend, and broker. Each of these functionalities is explained below along with initial setup.

22.4.1 Initial Setup

As mentioned above, the application requires 2 important API Cloud Vision API which has to enabled for the specific project id, the application can be started, in google cloud console. The best part for a software developer to test the working application is to launch directly using gcloud command-line tool, as it doesn't require authentication setup. For Cloud SDK installed on the local environment, setting up the authentication is crucial. For this, it is first required to create a service account and download service account key which is usually in JSON file format. Then set the environment variable `GOOGLE_APPLICATION_CREDENTIALS = [PATH]`, where `PATH` is the file path of the JSON file downloaded from Google Console Dashboard.

After setting the environment variable, it is important to activate gcloud command line tool, with the command “gcloud init”. Then it is required to set up the cluster to run the project. The cluster is built on Google Cloud, for billing, is activated on your Service Account. To set up the cluster, it is required to choose the compute zone. The command “gcloud config set compute/zone <zone-

name>” sets up the particular zone you would want your cluster to locate at. This is important because it provides less latency when connecting to the cluster from gcloud, or via Cloud SDK. The zone-name would like east1-b,central1-a and so on, and it is better to choose according to location, although the features offered doesn't differ much. The actual cluster can be created using the command “gcloud container cluster create <cluster-name> –num-nodes <num> –scope cloud-platform”. This command specifies cluster name and the number of nodes created for each zone of that cluster. After creating a cluster, the cluster should be authorized with the service account, in order to get access to all the api's. Hence the command " gcloud container clusters get-credentials <cluster-name>“. If the cluster is perfectly created then you should be able to get the correct information for the cluster created by activating”kubectl cluster-info”. Since initially, there are no pods, services, deployments created on running the command kubectl get pods you find no pods created. The execution section gives more details on how to run the project.

22.4.2 Frontend Microservices

The front end of the application plays a key role as the load balancer service for the entire application. It is basically a dynamic web-page, which allows the user to enter the location, for example, San Francisco, CA and business like food, dinner. Based on these inputs, photos are fetched. The technologies used for the web-page along is python, html, javascript, and CSS. The below paragraph explains briefly about each of these technologies and how they have been used in te frontend of this application.

- Python Python web-development framework Flask is used to make the server calls to storage.py. Flask package is a micro framework it can be implemented on top of any backend service with no restrictions like particular tools, libraries, or extensions are required. Moreover, Flask supports RESTful requests for dispatching GET, POST. Hence it is very useful in our project to easily make use of API to communicate and receive from the backend.
- Javascript JavaScript is a core-technology mainly used to build dynamic web-pages. There are different frameworks such as vue.js, AngularJS, ReactJS, to design creative and interactive web pages. In

this frontend javascript is extended via a material design file in storage.googleapis.com. This really reduces the time, effort and gives a world-class view for the webpage.

- HTML Hypertext Markup Language is useful to add content to the webpage. To make the web pages dynamic and add style to the page, Javascript, CSS is used. The HTML is the base of the webpage no matter any web-technologies out there, they must be added on top of HTML base page.
- CSS

Cascading Style Sheets (CSS) is used for styling the HTML webpage, with respect to display and it is added either internal with `<style></style>` tags or external using CSS file. The CSS for this project is taken from google style package, google apis for fonts, material icons and for other design can be selectively picked from material designs stored at google storage.

In this frontend, the main.py is developed using python flask and it uses Kafka producer and consumer to communicate with the backend service. As the frontend service is deployed as a load-balancer service an external IP is provided which enables the user to access outside of the cluster through a web-browser.

22.4.3 Kafka Message Broker

The kafka message broker is used to communicate between frontend and backend microservices through producer and consumer concept. The kafka-python package is used to build producer and consumer. The docker-compose file for kafka and zookeeper can be deployed on kubernetes.

22.4.4 Backend Microservice

The mainapp provides the actual functionality of the application, starting from scraping the data to generating the desired output. The major functionalities involved in the service is divided into main.py, yelp_label.py and vision.py. Each is further modularized with function, which is briefly explained in below paragraphs.

The `yelp_images.py` has 4 functions `query_api()`, `get_business()`, `search()` and `request()`; As the given location and term are passed to `query_api`, it sends the information to `search` function. The `search` function sends a GET request (GET <https://api.yelp.com/v3/businesses/search>) with parameters term, location and search limit equal to 10 in the json object. The JSON object response for the GET request is a dictionary type and is returned to calling the function. The `query_api()` stores each business list into businesses dictionary. This business ist consist of several other parameters like rating, price, id, categories, location details, review count and so on. But yelp-fusion offers an exclusive option to get more details of each business via getting Request (GET <https://api.yelp.com/v3/businesses/id>) with business id. Therefore, for each business, business id is extracted and passed to `get_business`. In this function, a request is sent with business path <https://api.yelp.com/v3/businesses/> and business id is appended to form the appropriate URL for the request. The response object for this request consists of details specific to that business only, which includes open hours, photos and reviews. For this project, we require only photos of each business, hence the URLs of the photos for all the businesses are appended to a list. All the photos are returned to the calling function in `main.py`.

In `vision.py`, the authorization to access Vision API is set through Google Credentials python package. The authorization done at this step isn't specific to vision API rather it is common to access almost any Google API. Now, for image passed to the `detect_labels()`, a post request is sent to Vision API to annotate the image, by particular mentioning the label detection feature and the image URL for that image. It is important to specify the feature or any further details of the image as it improves Machine Learning model for the analysis of the photo by adding weightage to each of the parameters. Along with the label which is a description parameter of string type, response object consists of details of the image such as score, confidence, location, and so on. The label annotation is returned to the calling function for the sent image url.

To summarize, `main.py` brings together all the above functionalities, it retrieves the data from `yelp_images.py`, passes photos to `vision.py` to label each one of them, after completion it produces to labeled images to frontend service.

22.5 RESULTS

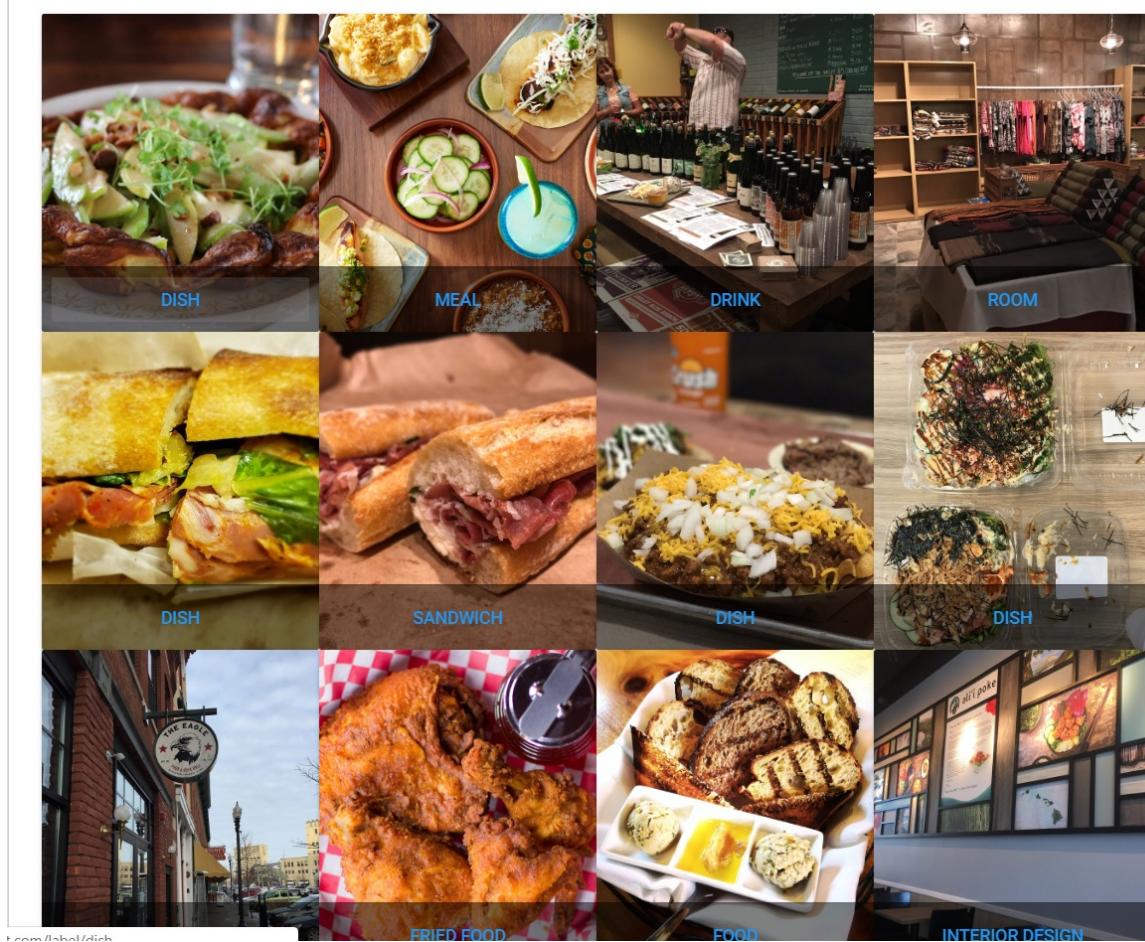


Figure 206: yelp_images_labeled

```
+@fig:yelp_images_labeled is the result of this project when user
provided input of business=food and location=Indianapolis
```

```
![yelp_images_labeled](results/result.PNG){#fig:yelp_images_labeled}
```

The results of the project are to display labeled images from Yelp photos dataset. And this achieved by populating the browser with the label and the image_url pair received by Kafka consumer on topic labels. As the cloud-vision api is a pre-trained model on a huge dataset of Google, the label detection is with an average measure of 0.8161305 scores, and 0.8161305 topicality. For this application, images of top 10 business for user input business category and location are generated and the image.annotate request is reiterated 3 times to make sure highly accurate label is detected for the image, if not in a single request.

22.6 BENCHMARK

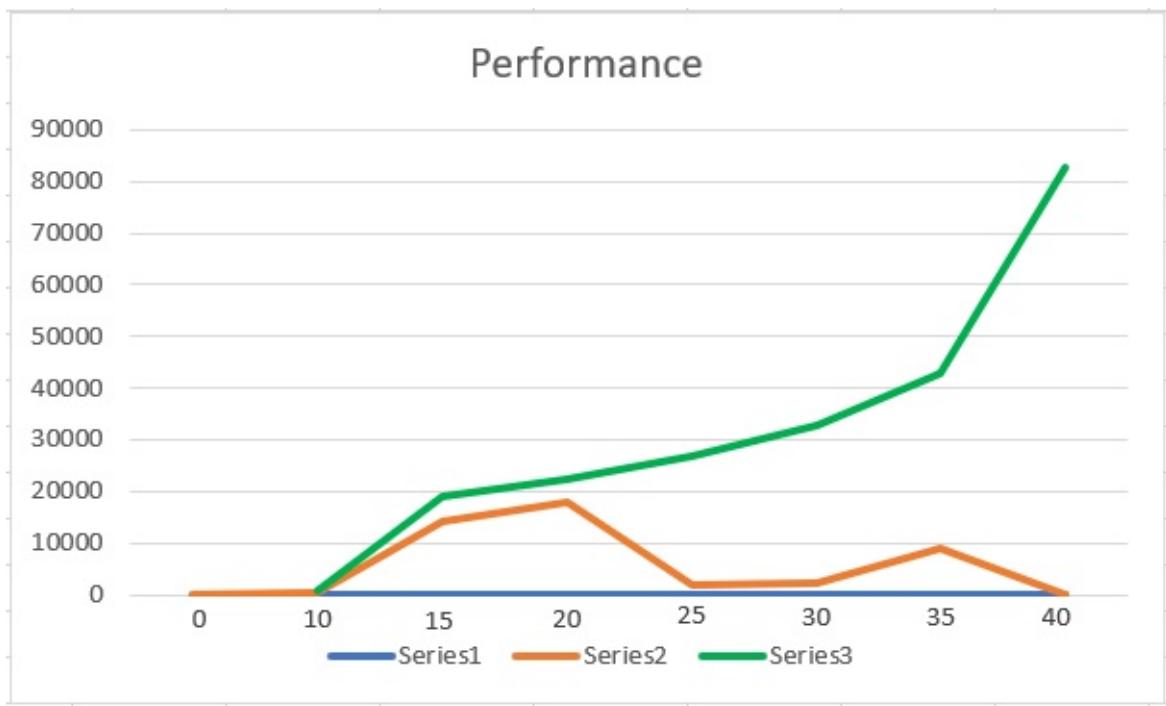


Figure 207: with_and_without_kubernetes

```
+@fig:Throughput calculated based on timestamp with and without kubernetes on Google Cloud Platform
```

```
![yelp_images_labeled](results/result.PNG){#fig:yelp_images_labeled}
```

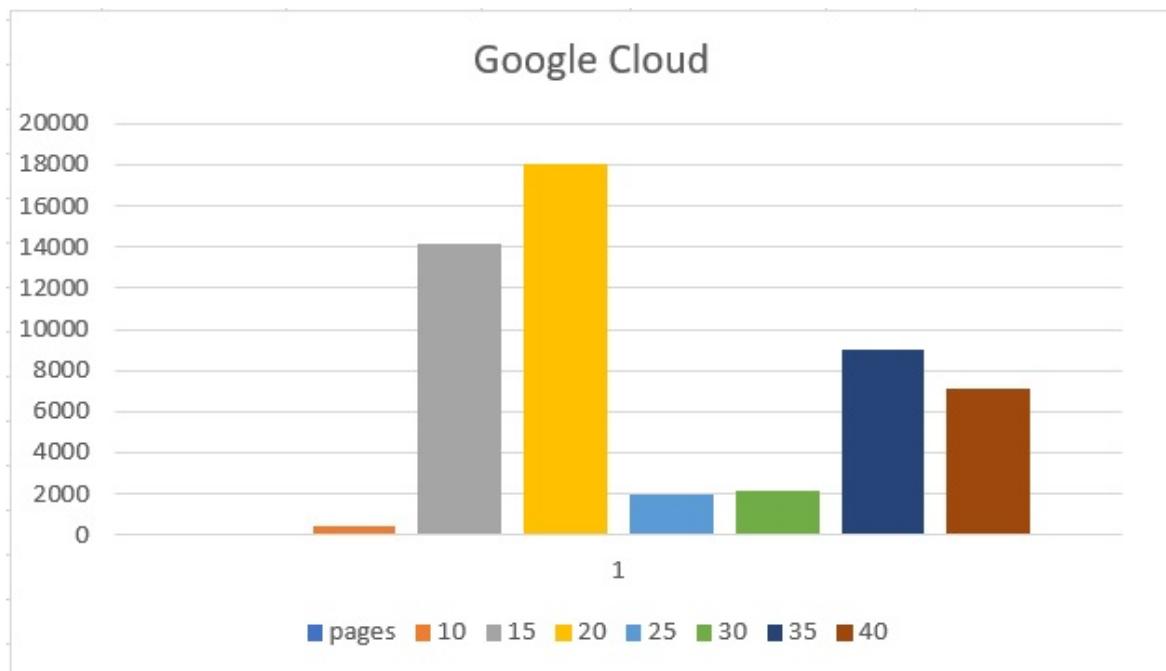


Figure 208: google_cloud_log

```
+@fig:Throughput calculated based on timestamp in container logs when deployed on Google
```

Cloud

```
![yelp_images_labeled](results/result.PNG){#fig:yelp_images_labeled}
```

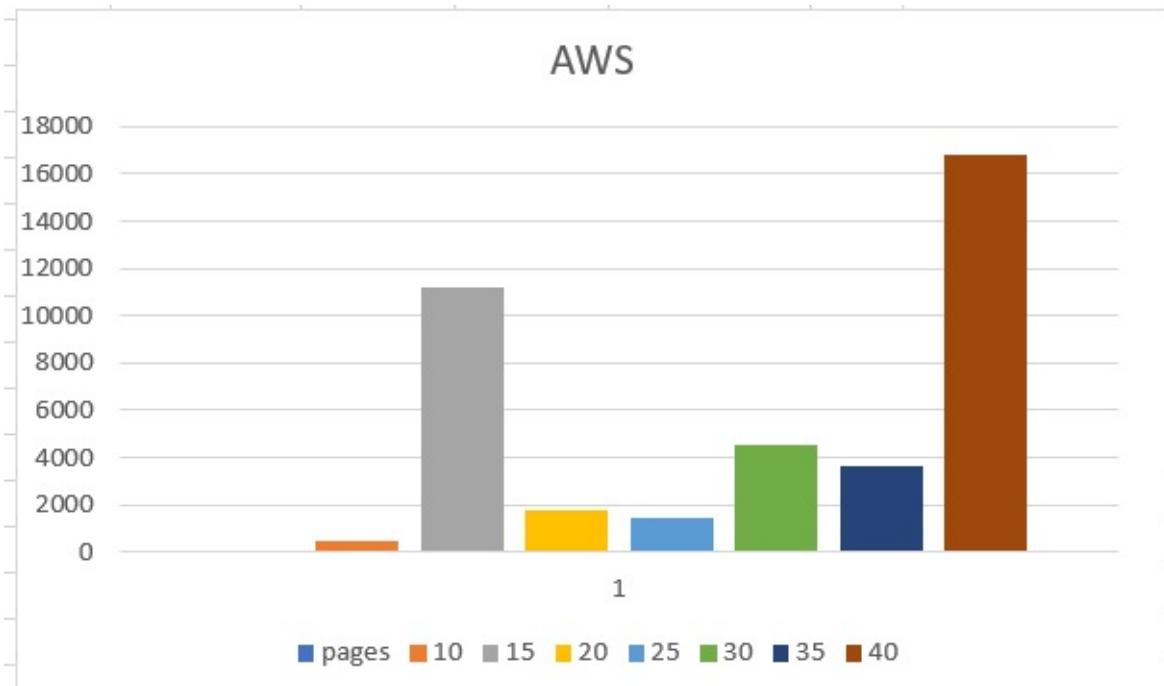


Figure 209: aws_log

```
+@fig:Throughput calculated based on timestamp in container logs when deployed on AWS
```

```
![aws_log](images/aws.PNG){#fig:aws_log}
```

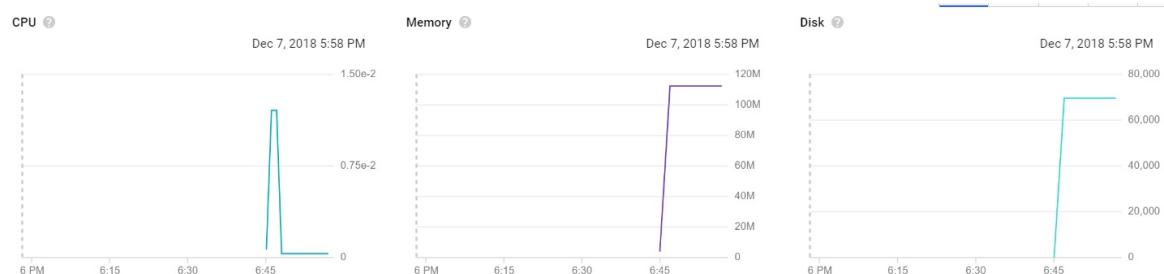


Figure 210: google_cloud_stat

```
+@fig:Stat from kubernetes dashboard when application is deployed on Google Cloud
```

```
![google_cloud_stat](images/cloudStat.PNG){#fig:google_cloud_stat}
```



Figure 211: aws_stat

```
+@fig:Stat from kubernetes dashboard when application is deployed on AWS
![aws_stat](images/awsStat.PNG){#fig:aws_stat}
```

The application is deployed using Kubernetes on Google Cloud Platform and AWS platform. The benchmark analysis is based on the comparison of throughput which is time taken for the to label images by the backed service and display to the user. The x-axis of the first 2 graphs is the timestamp difference and on y-axis the number of business from which images are retrieved for the given category. Apart from that, when the application is deployed on average it takes at least of 55 seconds to generate the external IP address. The graph showing usage of the resources is captured from Kubernetes dashboard. The benchmark for this project extensively depends on Cloud Vision API as the batch processing for the images is limited and this can effect the throughput for the label-detection and showcasing the results on the browser.

22.7 CONCLUSION

The application to retrieve images from Yelp-Fusion API and detect label using Cloud Vision API is developed using microservices and Kafka. With Kubernetes not just orchestration of Docker components but the flexibility, scalability of the deployment of microservices becomes efficient.

22.8 ACKNOWLEDGEMENT

The author would like to thank Dr. Gregor von Laszewski for his support and guidance throughout the course of project development.

23 NEW APPROACHES TO MANAGING METADATA AT SCALE IN RESEARCH LIBRARIES

Timothy A. Thompson

timathom@indiana.edu

Indiana University

hid: hid-sp18-705

github: [blue icon](#)

code: [blue icon](#)

as you can see from the epub your section headings are wrong make sure they show up in the TOC correctly

I checked the epub, and all section headings appear correct in the TOC.

urls in text must be replaced with bibtex entries

Fixed.

are last two figures self created? or do they need citations.

The last two figures are self created. Original graphviz files are in project-code/graphviz.

quotes are wrong. If highlight use italic if quote use

“This is a quote” ???.

Fixed.

it is unclear what code you developed, please add a section with

Artifacts Developed by Author

- list in that in bullet form what they are and what they do

I have added the section requested and attempted to detail all functionality.

Keywords: i523, Research Libraries, Library Catalogs, Blockchain, BigchainDB

23.1 ABSTRACT

The analysis of big data often relies on distributed storage and computation; however, access to big data—and to the platforms capable of managing and processing it—continues to be largely centralized. Centralization is particularly evident in the case of the metadata produced, managed, and disseminated by academic and research libraries. Libraries typically create and share their catalog records by uploading them to a centrally managed database, which can then be searched by other libraries for records that can be copied and added to an institution's local catalog. This centralized approach, which operates on the basis of membership fees, has the advantage of scalability and availability, but it comes at the cost of a loss of autonomy. Although technical innovation is possible within the current paradigm, the growing maturity of peer-to-peer protocols and decentralized solutions points toward an alternative approach, one that would allow libraries to share their data directly without having to pay an expensive intermediary.

23.2 INTRODUCTION

The problem of entity resolution (also known as record linkage or data matching [242]) is one that has a direct impact on the work of information professionals in research libraries. In library units responsible for catalog management, many workflows center on a procedure known as copy cataloging, which aims to expedite the processing of new acquisitions. Copy cataloging involves searching a shared database for records created by another cataloging agency, but that describe identical publications that have been acquired by one's local institution [243]. In the current environment, a single company, the Online Computer Library Center (OCLC), is the only viable platform for global cooperative cataloging [244], [245].

OCLC provides data aggregation and warehousing services that allow libraries to effectively share their data, but its business model does not encourage peer-to-peer interaction and innovation among individual libraries. This vendor-driven

paradigm entails the acceptance of a business model that, in effect, charges libraries for serving their own data back to them, with some added value through quality control and normalization. Once a library's data has been sent to OCLC, it also becomes subject to potential licensing restrictions, as well as the expectation that future dissemination of the data will include attribution of OCLC [246], [247].

23.3 NEW APPROACHES TO METADATA MANAGEMENT

Libraries have a tradition of experience with record matching and automation [248], but now stand to benefit from the increasingly mainstream availability of algorithms and routines developed within the context of data science and machine learning. Sophisticated algorithms for string comparison and probabilistic record linkage have long been available, but are not widely used by libraries, with the exception of large-scale projects such as the Social Networks and Archival Context Project (SNAC) [249] and the Virtual International Authority File (VIAF) [250], itself a project of OCLC. The former has employed methods based on Naive Bayes classification algorithms to aggregate and disambiguate data from across a wide range of libraries and archives (the reported accuracy of the approach fell with the range of 80-90 percent) [251]. More recent approaches to record matching have improved on probabilistic methods such as Naive Bayes by using Artificial Neural Networks, improving accuracy rates in some cases to 98 percent or more [252].

As machine learning tools and methods have become more accessible, however, large-scale, real-time access to library metadata has not necessarily followed suit. The catalog of a large academic library may contain around 10 million records [253]. By comparison, as of August 2018, the OCLC catalog database, WorldCat, contained 427,501,671 bibliographic records in 491 languages [254]. As long as service providers such as OCLC maintain centralized control over the aggregated metadata of research libraries, large-scale computational analysis—and the innovation it could produce—will remain proprietary and locked away.

The situation is further complicated by professional and cultural norms within libraries. Although decentralization may be appealing as an ideal, librarians who manage bibliographic metadata are also immersed in a discourse that centers on the idea of control: they use terms such as authority control, controlled

vocabularies, and intellectual and physical control of collections [255]. The idea of control is closely related to the idea of trust: when workflows and systems are centralized, it becomes easier to enforce norms and standards, but it also becomes more likely that potential contributors may be excluded, especially when they are unable to afford the price of membership in a proprietary system.

New decentralized technologies such as blockchains could allow research libraries to form robust peer-to-peer networks that would enable data sharing on a larger scale. Public blockchains such as Ethereum and Bitcoin are limited in the amount of data that can feasibly be stored on chain, but alternative platforms that address this limitation have recently emerged. When discussing decentralization, there are a range of new technologies that should be considered, and blockchain may or may not be the most appropriate for a particular use case—or blockchain technology may need to be used in conjunction with other technologies in order to enable decentralized exchange. Several efforts are underway to develop systems for decentralized file storage using distributed hash tables, one of the most prominent being the Interplanetary File System (IPFS) [256]. In a way similar to the software versioning protocol Git, IPFS uses hash values to capture the state of a file at a particular point in time and then serves it on a peer-to-peer network. IPFS hashes might be referenced as links in blockchain transactions in order to decouple the storage layer from the accounting layer [257].

In this regard, the blockchain-based database service BigchainDB, implemented in Python, provides a robust storage data solution while preserving the benefits of blockchain features such as data immutability and an asset-based transactional model. By running a consortium blockchain network of BigchainDB nodes [258], libraries could be empowered to abandon centralized models and begin managing their data collectively.

23.4 BLOCKCHAINS FOR RESEARCH LIBRARIES

Some in the library profession have been skeptical of blockchain applications for their domain, arguing that they have been overhyped as a panacea, when in reality they are nothing more than slow, expensive, append-only databases [259]. Even core developers working to support the Bitcoin blockchain have argued that the constraints imposed by blockchain technology, such as immutability and

decentralized consensus, make it appropriate for a very limited set of applications—namely, currency and the exchange of value [260]. For individuals and organizations who are investigating blockchains as a technical solution, it is important from the outset to establish a framework for evaluating their applicability and appropriateness [261].

For example, a blockchain-based solution may be appropriate in a scenario in which there is a lack of trust among participants, or in which processes and collaboration would be more efficient if the need for trust were eliminated [261]. In the case of a shared catalog for research libraries, trust is an issue because not all participants can be trusted to provide data that conforms to expected levels of quality. A commercial, centralized solution mitigates these concerns by requiring participants to pay a membership fee. A blockchain solution addresses issues of trust by enforcing a decentralized consensus mechanism, which may take different forms, but which is designed to ensure that participants can trust the network to maintain a consistent state across all transactions [262].

The Proof-of-Stake consensus algorithm, employed by some blockchain networks as an alternative to Bitcoin’s resource-intensive Proof-of-Work mechanism, is similar to the membership fee model in that validator nodes are elected based on their share of *stake* in the network, measured by their willingness to commit or stake an allocation of network tokens as a proof of honesty [263]. For research library applications, a variation of Proof-of-Stake known as Proof-of-Authority may be the most appropriate solution [258], [263]. In contrast to public blockchains such as Ethereum and Bitcoin, or fully private blockchains restricted to a single organization, so-called consortium blockchains may be the preferred approach, one in which consensus

“is controlled by a pre-selected set of nodes” [258].

The model implemented by the BigchainDB project fits the parameters of a consortium blockchain that implements a Proof-of-Authority approach to consensus [264].

23.5 DESIGN REQUIREMENTS

A blockchain-based catalog for research libraries should support the creation of

a decentralized marketplace for library metadata. Rather than paying a centralized exchange to distribute their catalog records, libraries could buy and sell records in a peer-to-peer exchange. Catalog records could thus become a source of revenue rather than a costly expenditure. Many blockchain systems support the creation of so-called smart assets, or the creation of tokens to represent real-word assets. A new token could be minted to facilitate the exchange of metadata objects, and payment and settlement channels could be created using smart contracts on a public blockchain such as Ethereum.

However, a public blockchain solution does not fully satisfy the requirements of decentralization for this use case. A data asset cannot be represented exclusively by a token—it also needs to be stored in a decentralized system optimized for read and write transactions. Public blockchains such as Ethereum have been designed for exchange, not storage. At the current price of the Ethereum blockchain's native token, Ether (ETH), at approximately \$200.00, storing 1 gigabyte of data on the blockchain would cost over \$7,000,000.00 [265]. A decentralized system for library metadata must be able to scale and store big data out of the box. BigchainDB is a production-ready solution that might meet the requirements for this use case: it supports the creation of assets and the direct storage of metadata objects on its blockchain [266].

23.6 SCOPE

Findings are presented from an initial exploration of BigchainDB as a blockchain database solution for a shared library catalog. An overview of the BigchainDB architecture and data model is provided, and some of BigchainDB's features and functionality are probed. A preliminary analysis of library metadata standards and requirements is included, and the question of whether they can be accommodated using BigchainDB is examined.

23.7 BIGCHAINDB

23.7.1 Evolution

BigchainDB was created to address the scalability and storage limitations of traditional blockchains such as Bitcoin and Ethereum and to create a hybrid

solution that builds a blockchain layer on top of an existing big data system [267]. Development of the BigchainDB framework initially focused on integration with the RethinkDB system, but now works exclusively with MongoDB [267], [268].

The early focus of BigchainDB development was to create an architecture that would allow existing big data databases to be enhanced with blockchain features [269]. The original BigchainDB whitepaper, released in June 2016, focused on the scalability limitations of traditional blockchain networks such as Bitcoin and claimed that it should be possible to develop a blockchain-based distributed database that would enable

“1 million writes per second throughput, storing petabytes of data, and sub-second latency” [269].

This is in contrast to the storage restrictions and 7 transaction-per-second (tps) limit of the Bitcoin network [269]. The advantages of adding a blockchain layer to an existing distributed database would be to incorporate

“decentralized control, immutability, and creation [and] movement of digital assets” [269].

The primary challenge in designing a decentralized system is how to defend against both arbitrary failure and malicious actors. In so-called Sybil attacks, an attacker attempts to generate false identities in order to gain majority control over a network [270]. To address Sybil attacks, BigchainDB proposes a governance model that would create a federation of trusted nodes. Because all participants are known, any attempt by one participant to gain control over the network would be obvious. A more pervasive vulnerability comes in the form of the so-called Byzantine Generals’ Problem [269]. Nodes in a distributed network must be able to reach consensus about the final order of transactions at each state of the system, even in the presence of node failure or malicious attempts to manipulate system state in order to gain an unfair advantage—for example, in double-spending, in which a transaction is replayed so that the same asset can be used again (a particular problem in the case of financial transactions) [269], [271].

In its original design, BigchainDB relied on the consensus algorithm of its

underlying database to manage benign node failure and incorporated additional constraints to verify the integrity of the voting process by which nodes in the network approved transactions—and the blocks containing them—as valid [269]. However, in its initial version, BigchainDB did not claim to be Byzantine Fault Tolerant (or BFT—the term used to indicate that a system can withstand unexpected node behavior, whether benign or malicious, up to a certain threshold [269]). In the original design, all nodes belonged to a single logical database. This made the system overly centralized and vulnerable to attack: a malicious actor who gained control over a single node would have been able to drop the entire database, which was shared among all nodes in the network [267], [268].

BigchainDB 2.0, released in June 2018, underwent a complete redesign and incorporated full Byzantine fault tolerance through integration with Tendermint, an application for managing consensus and state machine replication in blockchain systems [272], [273]. As a result of implementing Byzantine fault tolerance through Tendermint, BigchainDB’s original goal of supporting 1 million tps was no longer viable.

23.7.2 Benchmark

A recent benchmark of BigchainDB 2.0 throughput performed by the BigchainDB development team indicated that the system was able to process approximately 300 tps [274]. Benchmarking was carried out on a four-node network on Microsoft Azure-hosted virtual machines located in the same data center. Three separate experiments were run to test different options, and a full report of configurations and results is available for review [275]. The primary experiment tested how long it would take to commit 1 million transactions of 765 bytes each to the BigchainDB blockchain under default settings. Results showed an average rate of 299.0 tps and a median rate of 309.0 tps. All 1 million transactions were finalized in 56 minutes with no failures [275].

23.7.3 Architecture

The architecture of a BigchainDB 2.0 network is shown in Figure 212, created by the BigchainDB development team. Each node in the network is self-contained and includes its own MongoDB database and Tendermint application

server. Tendermint is used to manage consensus, communication, and state replication among nodes, whereas the software that is unique to BigchainDB is responsible for

“registering and tracking the ownership of ‘assets’” [272].

In BigchainDB 2.0, as is the case in general with systems that are Byzantine Fault Tolerant, $3f + 1$ nodes are necessary to run a network, where f is the number of faulty nodes to be tolerated [276]. Therefore, at least four nodes are required in order to run a BigchainDB network: if one of the four nodes becomes unresponsive or attempts to approve an invalid transaction, the network will continue to function based on the majority consensus of the other three nodes [276].

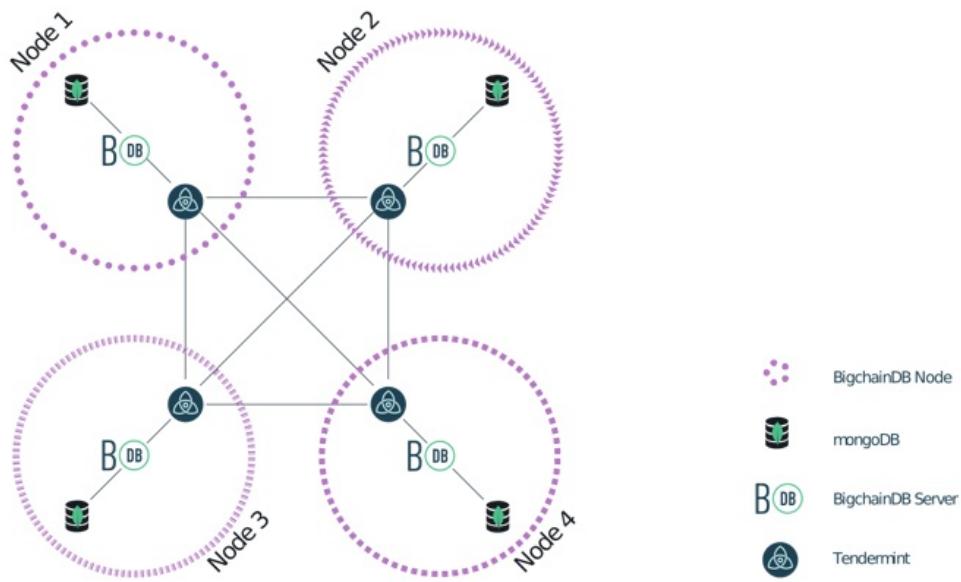


Figure 212: High-Level Architecture of BigchainDB 2.0 [277]

A BigchainDB client can potentially connect to any node in the network. Each MongoDB instance contains a full replication of the data stored in the network [276]. The BigchainDB project officially supports three client drivers to connect to a node server (in Python, Node.js, and Java) [278].

23.7.3.1 BigchainDB Server

The BigchainDB Server, written in Python, implements the logic to model, validate, and store transactions in the BigchainDB blockchain [272]. The server also incorporates a Python implementation of the Crypto-Conditions specification, which is a standard for enforcing complex boolean conditions for fulfillment (or transfer of assets) using cryptographic signatures [279].

All objects in BigchainDB are modeled as *assets*. Two transaction types are available for managing assets: CREATE and TRANSFER [280]. Each transaction must be cryptographically signed with the private key of its *owner* (the agent who created an asset through a CREATE transaction or to whom an asset was assigned through a TRANSFER transaction). Public/private keypairs are implemented using the Edwards-curve Digital Signature Algorithm Ed25519 [280]. A transaction is encoded using a dictionary or associative array that can be serialized as a JSON object. The BigchainDB Transactions Specification defines the structure and usage of a BigchainDB transaction object [280]. The key/value pairs that all valid BigchainDB transactions must include are listed here:

```
{  
  "id": ctnull,  
  "version": version,  
  "inputs": inputs,  
  "outputs": outputs,  
  "operation": operation,  
  "asset": asset,  
  "metadata": metadata  
}
```

Conditions for fulfillment and asset transfer are defined in the values of the *inputs* and *outputs* keys. An object representing the asset itself is stored as the value of the *asset* key and cannot be modified once an asset has been created and committed to a block in the BigchainDB blockchain. The *metadata* key is used to store an arbitrary object that records additional information about the asset or its state: in contrast to the asset object, the metadata object *can* be modified with each TRANSFER transaction [280].

23.7.3.2 Tendermint

Tendermint provides an application interface and BFT consensus algorithm for

replicating application state across the nodes in a decentralized network [273]. Tendermint Core implements the consensus algorithm, which ensures that all nodes agree on a single order for transactions. Tendermint's Application Blockchain Interface (ABCI) provides a language-agnostic interface for blockchain applications to use when validating and processing transactions [273].

Figure 213 is a sequence diagram, created by the BigchainDB development team, that illustrates the role of Tendermint in processing BigchainDB transactions. After a client prepares and signs a transaction, typically using a BigchainDB driver, the transaction is submitted to the BigchainDB server for initial validation. The server then sends the transaction to Tendermint, which includes it in a local memory pool. Tendermint returns its own validation request to the server and, upon confirmation, proposes a new block and begins a round of voting as part of its consensus algorithm. Each node in the network votes on the order and validity of transactions in the block, and if consensus is reached, the block is committed to the application's blockchain [267], [281]. BigchainDB stores a queryable copy of each block in MongoDB, while Tendermint appends each block to its canonical blockchain, which is stored in an internal LevelDB database and used for replicating transaction state to network peers [267], [273].

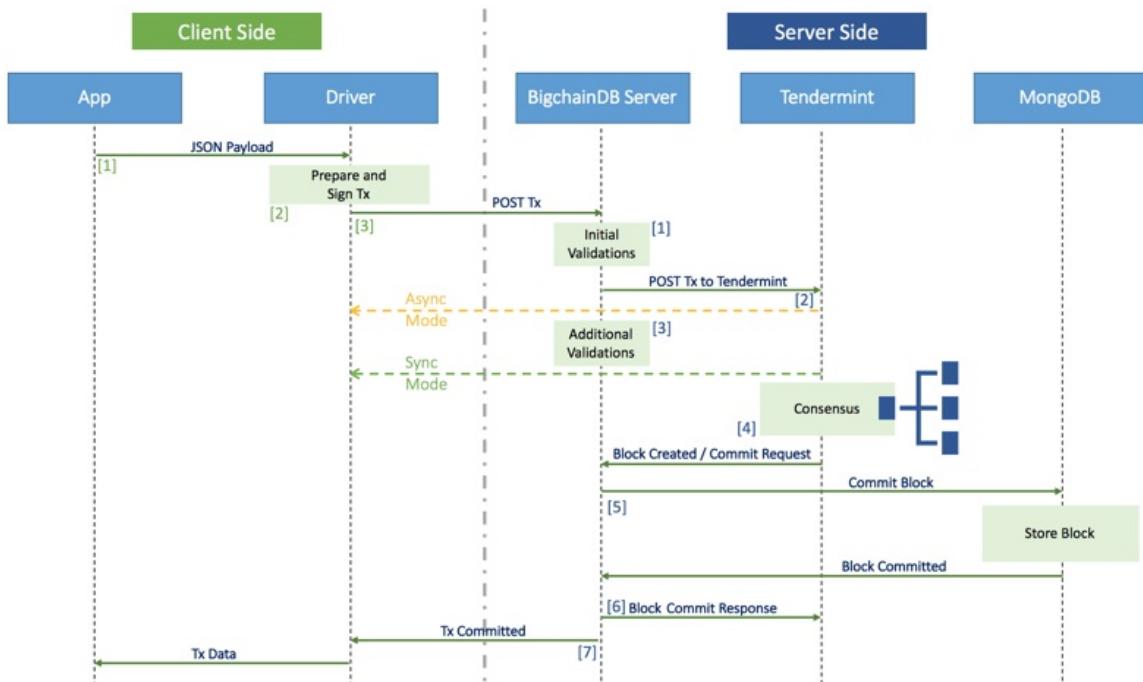


Figure 213: BigchainDB Sequence Diagram [281]

23.7.3.3 MongoDB

MongoDB is an enterprise-grade NoSQL database optimized for storing JSON objects as documents. It supports both high availability (replication) and scalability (sharding) [282]. Early versions of BigchainDB used a single MongoDB replication set and were able to take advantage of these core MongoDB features. In BigchainDB 2.0, because each node maintains a separate MongoDB instance, replication and sharding are not supported out of the box [272]. In order to enforce practical immutability and decrease the likelihood of data tampering, the BigchainDB server limits access to MongoDB and does not expose any interfaces for deleting or making arbitrary modifications to database documents [267]. Although it is technically possible for a system administrator to modify the MongoDB database directly, each BigchainDB transaction is signed with a public/private cryptographic keypair—thus any tampering would result in a modified signature, which would be detectable by other nodes in the network and would violate its social contract [267].

BigchainDB does take advantage of MongoDB's query facility for read-only queries. Through its own HTTP API, it exposes a simple search interface for querying MongoDB, but it also allows node administrators to create custom indexes and leverage the full range of MongoDB query functionality [277].

23.8 DATASET

The dataset used is intentionally small and meant to test a potential use case for BigchainDB as a library catalog application. Currently, library catalog records are stored in a set of industry-specific formats maintained by the Library of Congress: the MAchine Readable Cataloging (MARC) formats for bibliographic and authority data (standardized as ISO 2709 and ANSI/NISO Z39.2) [283], [284]. In recent years, the Library of Congress has undertaken an effort to update library metadata standards and adopt standards and formats maintained by the World Wide Web Consortium (W3C)—specifically those related to linked data and the Semantic Web, such as the core data model known as the Resource Description Framework (RDF) [285], [286]. A new domain-specific data model and ontology for library metadata, expressed using the W3C's OWL standard for semantic ontologies, is currently being developed and implemented [287]. The data used here for testing with BigchainDB follows this model, known as the

Bibliographic Framework Initiative (BIBFRAME) [285].

In the basic model proposed by BIBFRAME, descriptions of library resources are divided into three entity types or classes: Work (the abstract concept of the resource), Instance (the embodiment of a Work in a particular publication), and Item (a physical copy of an Instance) [285]. As an example, a catalog record from the Indiana University Library catalog was chosen. This record describes the Lilly Library’s partial copy of the Gutenberg Bible. The data is divided into six files in the `project-data` directory:

```
ocm05084045.xml  
gutenberg-iul-item.rdf  
gutenberg-iul-instance.json  
gutenberg-iul-item.json  
gutenberg-iul-record.json  
gutenberg-work.json
```

The file `ocm05084045.xml` represents the original MARC-format record from the Indiana University Library catalog, encoded as XML [288]. The file `gutenberg-iul-item.rdf` provides an example of a partial conversion of the original MARC record to the BIBFRAME model using the RDF/XML serialization [289]. The remaining files represent the data used to create assets for storage in BigchainDB and are encoded in BIBFRAME using the JSON-LD serialization of RDF [289]. Several preprocessing steps of data conversion and cleanup were necessary. The original MARC/XML catalog record was converted to BIBFRAME RDF/XML using a suite of XSLT stylesheets provided by the Library of Congress [290]. The RDF/XML documents were then converted to JSON-LD using the Python RDFLib library (see the `convert_rdf.py` script in the `project-code` directory for a brief example). The JSON-LD produced by RDFLib was then broken into separate files to allow for the creation of individual assets in BigchainDB.

The JSON-LD output produced by RDFLib was further modified to support the inclusion of named graphs, a feature of RDF 1.1 that makes it possible to encode assertions about a collection of RDF statements [286]. Using named graphs, it is possible to record administrative metadata about the creation or generation of the resource description itself (when it was created, by whom, using what standards, etc.). It should be noted that the usage of JSON-LD—which has some unconventional syntax features—with BigchainDB led to the discovery of a bug in the BigchainDB validation code that triggered a fatal Internal Server Error (HTTP 500). This bug was subsequently reported and has since been fixed by a

BigchainDB core developer [291].

23.9 IMPLEMENTATION

23.9.1 Use Case

Currently, most large library catalogs are stored in enterprise relational databases such as Oracle. The catalog is one module in a suite of services known as an Integrated Library System (ILS), which also includes modules for circulation and ordering or acquisitions. The cataloging module in an ILS includes a public-facing interface for search and retrieval and a staff-facing interface for data entry and management. One advantage of using a distributed system such as BigchainDB for library cataloging functions would be to allow libraries to share their data more easily with peer institutions. BigchainDB's asset-based data model might also allow libraries to perform inventory and lending functions more efficiently. However, many functional components would need to be considered before determining whether a blockchain platform such as BigchainDB would be appropriate for the library catalog use case. One such component is focused on here: namely, the management of roles and permissions for data entry.

The Python component implements an extension to BigchainDB that adds support for Role-Based Access Control (RBAC) functionality [292]. The code is based on a Node.js example created by the BigchainDB development team to demonstrate the RBAC extension [293]. Support for RBAC is important for library cataloging because library personnel roles are typically divided between professional librarians (catalogers) and paraprofessional technicians. Librarians are expected to create *original* descriptions of library resources, whereas paraprofessionals are responsible for copying existing data from a shared database such as OCLC WorldCat. Public blockchain systems do not usually impose write restrictions (allowing anyone to write to the database), so support for RBAC is an important consideration when evaluating BigchainDB.

23.9.2 Installation

BigchainDB was designed to be a federated network of distributed nodes. In an ideal setup, each node would be maintained in a different location by a different

operator [267]. The official BigchainDB documentation provides instructions for those interested in setting up a production network [294]. However, for testing purposes, a full network is not necessary. A BigchainDB test network is currently available at [295], but the testnet installation does not include the RBAC extension required here.

The official BigchainDB distribution includes a Docker Compose script that can be used for spinning up a single node. It also includes a shell script and a set of Ansible playbooks that can be used to provision a four-node network of virtual machines using Vagrant and VirtualBox. Both methods were tested, but frequent errors were encountered when trying to install and run a network of virtual machines. The Docker approach was chosen for simplicity and because it was possible to build the container from the RBAC branch of the BigchainDB Server codebase. The distribution includes a Makefile, and Docker containers for BigchainDB Server, Tendermint, and MongoDB can be easily run with a simple `make run` command.

23.9.3 Data Management

All BigchainDB CREATE transactions must include a JSON-serializable object to represent the asset being recorded on the blockchain. The `asset` field of a CREATE transaction takes an object with the required key `data`. The content of the `asset` field is treated as immutable—it cannot be changed once a CREATE transaction has been committed, or when ownership of an asset is subsequently changed using a TRANSFER transaction. The following shows how a Work asset might be represented in BigchainDB:

```
{
  "data": {
    "@context": {
      "rdfs": "http://www.w3.org/2000/01/rdf-schema#",
      "schema": "http://schema.org/"
    },
    "@type": [
      "http://id.loc.gov/ontologies/bibframe/Work",
      "http://id.loc.gov/ontologies/bibframe/Text"
    ],
    "rdfs:label": "Bible. Latin. Vulgate. 1454."
  }
}
```

Because this data cannot be changed, it makes sense to represent it simply using its RDF type (in this case, it is a BIBFRAME Work with a subtype of Text), as

well as a human-readable label. Any BigchainDB transaction may also include an optional `metadata` key that takes as its value an arbitrary JSON-serializable object. This flexible design makes it possible to effectively update data by using a TRANSFER transaction to indicate that the state of an asset has changed—and recording that change in the `metadata` object. The code in `rbac_demo.py` creates separate BigchainDB assets to represent the BIBFRAME types Work, Instance, and Item.

The data and code also illustrate how JSON-LD named graphs may be used to include both descriptive and administrative metadata about the same BigchainDB asset in a single transaction. The `rbac_demo.py` script inserts a second named graph into the `gutenberg-work.json` file so that it contains two named graphs: one representing the Work entity and one representing a separate Record entity (which is not part of the core BIBFRAME model). Within the named graph for the Record entity, there is an RDF property (`foaf:topic`) that links to the URI for the named graph representing the Work entity. Figure 214 illustrates this pattern, indicating how BigchainDB metadata objects may be used to create internal linkages among assets conforming to the BIBFRAME data model.

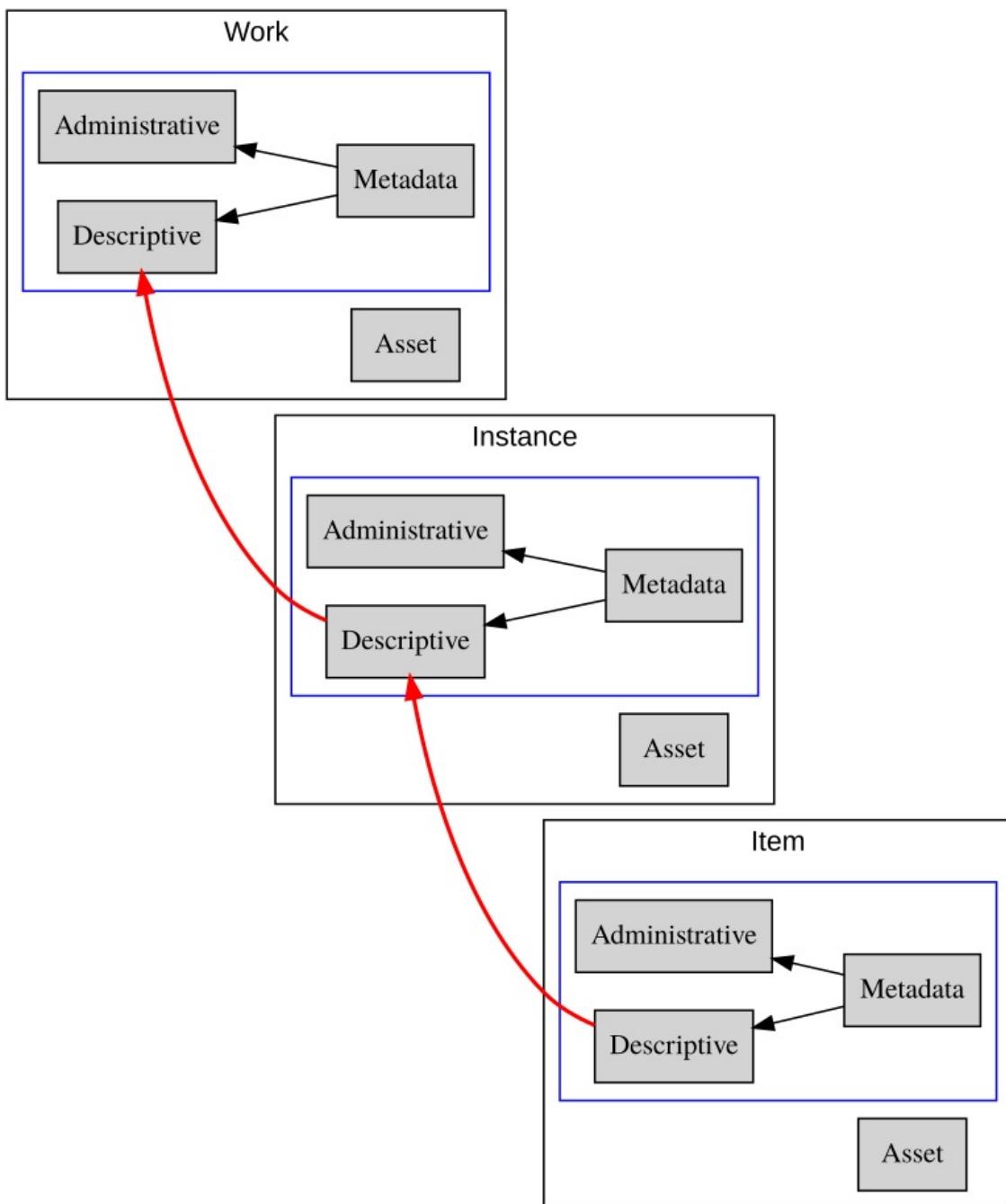


Figure 214: Graph of asset and metadata objects in BigchainDB

23.9.4 Role-Based Access Control in BigchainDB

The file `rbac.py` contains a single Python class, `BigchainRbac()`, that provides an interface to create new assets, users, types, and type instances for Role-Based Access Control in BigchainDB. In the BigchainDB RBAC extension, roles and

permissions, like everything else, are modeled as assets [292]. Two basic assets are necessary for bootstrapping: an asset to represent the application in which RBAC is being used and an asset to represent an admin group for admin users who can create other groups and users and assign permissions. BigchainDB RBAC employs two reserved keys, `link` and `can_link` that are used to create a dependency graph of users and asset types or groups. This graph is illustrated in Figure 215. Proper usage of these linkage keys is validated when a transaction is sent to the BigchainDB server, and a transaction is rejected if it violates the logic of the permissions scheme [292].

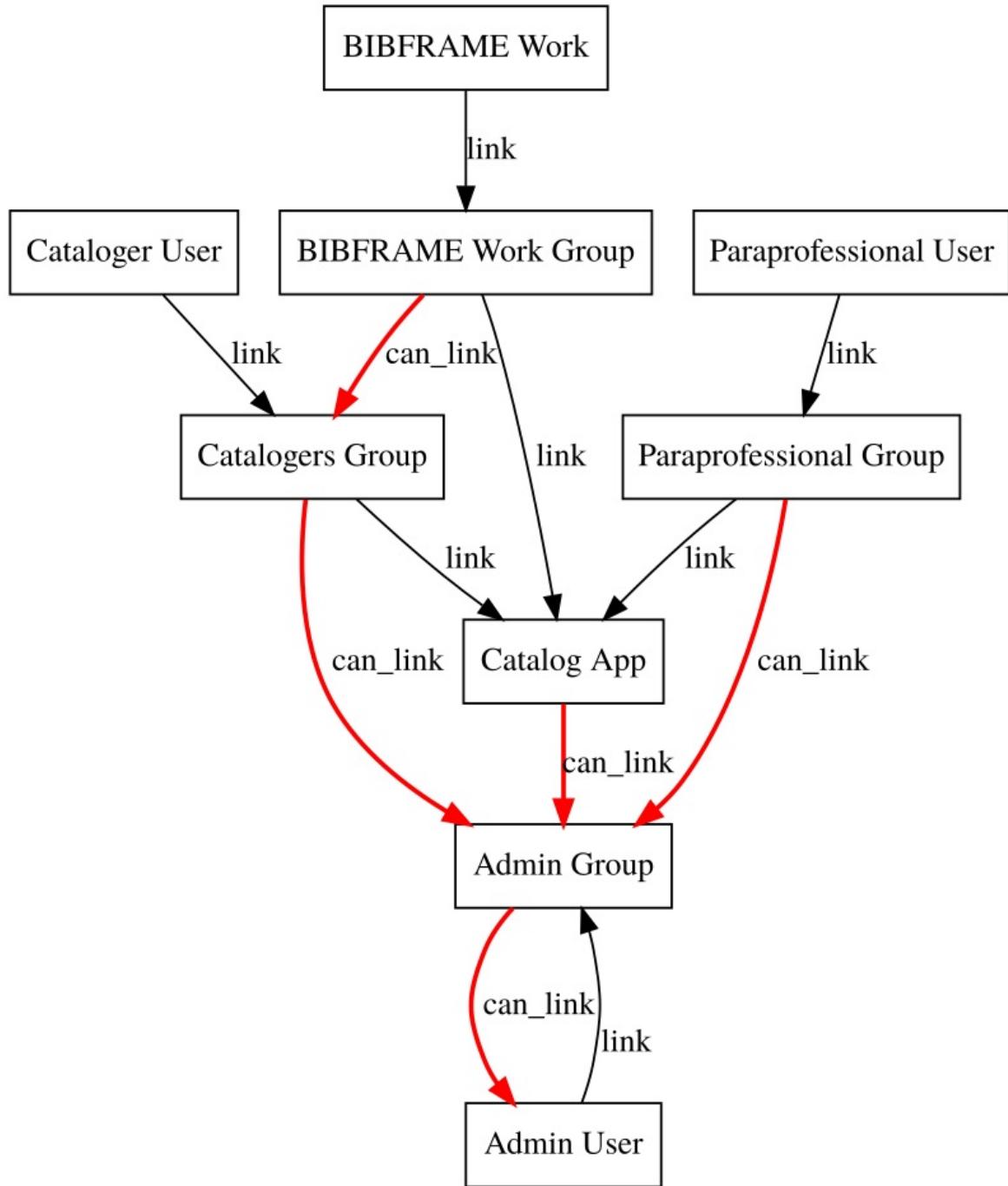


Figure 215: Graph of permissions in BigchainDB using Role-Based Access Control

The application logic proceeds according to the following steps:

1. User keypairs are generated for three different user types: admin users, catalogers, and paraprofessionals. In BigchainDB, a user's identity is

defined by a public/private keypair, and each transaction must be signed by one or more private keys, depending on the conditional logic that has been specified (especially in the case of TRANSFER transactions).

2. An admin user creates an asset to represent the admin user group and the application itself. The `can_link` key is included in the metadata object of the transactions. In the admin group asset, the value of `can_link` is a list of all admin user keypairs. In the app asset, it is the transaction ID for the CREATE transaction of the admin group asset—or, to put it more simply, the admin group ID. This linkage entails that only members of the admin group can link other groups to the app asset.
3. The admin user can then call the `create_type()` method of `BigchainRbac()` to instantiate new roles for the application. In `rbac_demo.py`, two roles have been created: one to represent ***catalogers*** and one to represent ***paraprofessionals***. In the definition of each type asset, the `asset` object includes the `link` key, which takes as its value the app asset ID. This linkage sets the scope of permissions to the current application context. The admin user can then create new user assets and assign them to a group. The `can_link` key of the group asset points to the admin group ID because only the admin user can assign new users to this group. By contrast, the `link` key of the user asset points to the user's respective group ID (such as that of the catalogers group). It is important to distinguish between a user—identified by a public key—and a user asset, which is the result of a BigchainDB CREATE transaction. In the RBAC scheme, when a user ***asset*** is created, it must then be assigned to the user's public key through a TRANSFER transaction. This can be described as placing the asset in the user's wallet [292].
4. The most important step occurs when an admin user creates a type to represent a resource group and then assigns permissions to restrict which users can CREATE instances of that group. In `rbac_demo.py`, for example, an asset is created to represent BIBFRAME Work resources. Then, the `create_instance_type()` method is invoked in order to link a resource asset to its type. Figure 215 shows that individual BIBFRAME Work assets are linked to the BIBFRAME Work Group, which has a `can_link` relationship to the Catalogers Group. Therefore, only users

associated with a cataloger user asset can CREATE BIBFRAME Work assets.

5. Finally, it is worth mentioning one concern that arises when working with BigchainDB as a data entry and data management system. The system's asset-centric approach requires that an asset have one or more owners. But in a shared scenario, how can one cataloger edit or update a record originally created by a different cataloger if they are not the owner of the asset? One solution would be for the original creator/owner to TRANSFER the asset to the person who wants to input new information or revise existing information. This may not be practical, however, since the original creator may no longer be present. Another solution might be to CREATE the asset with two original owners (that is, the CREATE transaction must be signed with two private keys): the cataloger as well as the admin user. The admin user could then TRANSFER the asset when needed. This is the approach taken by the demo developed here. However, in practice, it would have to be handled in an automated way, so that it would not be necessary for a single admin user to execute multiple TRANSFER transactions whenever the need arose. In general, best practices for managing public/private keypairs in the context of this use case is an area that calls for further examination.

When the `rbac_demo.py` script is executed, it populates a local BigchainDB instance with RBAC assets and tests whether catalogers and paraprofessionals can issue a CREATE transaction for a BIBFRAME Work asset. For successful transactions, the program simply outputs an HTTP URL that can be used to request the result of each transaction. However, one transaction attempts to CREATE a Work resource with a paraprofessional user asset. This transaction should fail with a `validationError`, as shown in the following:

```
BIBFRAME Work (IUL Paraprofessionals):
(400, {"message": "Invalid transaction (ValidationError):
Linking is not authorized for:
6GcYiCCNFsDbBicna6YCVq8RmSjGyB7MGJw9CHjDjqwh", "status": 400} \n'...)
```

23.10 CONCLUSION

BigchainDB is a new blockchain-based solution for managing big data. A

preliminary examination was undertaken to explore whether BigchainDB could be adapted for use as a library catalog database. The primary advantage of using a blockchain system instead of a conventional or distributed database is that it enables peer-to-peer interaction and exchange in a more fundamental way. Participants in a BigchainDB network are required to collaborate and coordinate their work very directly. Doing so would allow data scientists to access library metadata at scale and to more easily process, clean, and study it, potentially connecting it to other datasets. This could be particularly useful in large-scale text mining projects. In the current centralized scenario, library metadata remains locked away and can only be used for machine learning and complex analysis by the central service providers who control it.

A blockchain data model also allows for richer data modeling. Rather than a repository of records, a library catalog can be modeled as a collection of assets: actual books, journals, and multimedia resources, as well as the records that describe them. This could facilitate operations beyond cataloging, such as interlibrary lending, and make it easier to determine which library holds copies of which books, or subscribes to which journals. BigchainDB is flexible enough to accommodate the semantic data models, such as BIBFRAME, that are currently being developed by research libraries. BigchainDB itself is not a semantic data store, so additional solutions for semantically enriched information retrieval (such as graph databases) would also need to be explored. However, BigchainDB's incorporation of MongoDB allows users to take advantage of the indexing and querying capabilities internal to MongoDB. Extensions to BigchainDB provide support for features such as Role-Based Access Control, which would be important for enabling library cataloging workflows. More systemic analysis would be needed before recommending BigchainDB as a solution for library catalogs, but preliminary results indicate that the possibility would merit further exploration.

23.11 ARTIFACTS DEVELOPED BY AUTHOR

All original Python code is included in the `project-code/demo` directory:

- `demo/rbac.py`: Python class (`BigchainRbac()`) that provides an interface for the role-based access control extension that is available for BigchainDB.

- `demo/rbac_demo.py`: demo Python script that calls methods from the `BigchainRbac()` class to illustrate how the role-based access control features of BigchainDB could be employed. The use case is that of data and workflow management for a library catalog. This script sends a series of transactions to a BigchainDB server node.
- `demo/convert_rdf.py`: simple Python script provided to illustrate how data in the `project-data` directory was converted using the Python RDFLib library.

Two original graphviz files (used to generate the last two figures in the project report) are included for reference in the `graphviz` directory:

- `graphviz/assets-metadata.dot`: illustrates how data for a library catalog might be modeled and linked in BigchainDB.
- `graphviz/rbac-graph.dot`: illustrates the dependency tree and validation scheme used to enforce role-based access control constraints in BigchainDB.

23.12 ACKNOWLEDGMENT

The author would like to thank Dr. Gregor von Laszewski and the i523 and i516 teaching assistants for their support and suggestions in writing this report.

24 SCIKIT LEARN ALGORITHMS WITH REST API

Arijit Sinha, Ritesh Tandon
arisinha@iu.edu, ritandon@iu.edu
Indiana University Bloomington
hid: hid-sp18-520, hid-sp18-523
github: [cloudycode](#)
code: [cloudycode](#)

Keywords: docker, Scikit, AWS, Azure, Linear Regression, Boosted Decision

24.1 ABSTRACT

In current world, data is getting generated and stored with different storage systems. We need to use this data for a better understanding, analyze and can estimate the future scenarios with certain probability. There are many algorithms, which have developed and implemented for providing better accuracy on the future scenarios.

24.2 INTRODUCTION

Scikit learn is a library created for machine learning algorithms, which uses different datasets gathered over years to learn and predicts future scenarios. This library have methods for implementing supervised and unsupervised machine learning algorithm. Supervised algorithms are on the dataset, which has the target variable, which need to be predicted or estimated. This datasets can be acted with below different approaches Classification is based on the classes and the labeled data, we need to predict the unlabeled data. Regression is based on the continuous variable or data, we need to predict the future state of data is known as regression Unsupervised algorithms are on the dataset, where we do not see the target variable for prediction, we learn its behavior set of vector input variable to identify the clustering group of similar behavior data or sample [296]. Kaggle is a known location for different kind of datasets gathered by various institutes across globe.

24.3 SCOPE OF WORK

Below are the 3 algorithm from Scikit learn

- Implement Linear Regression
- Implement Boosted Decision
- Implement Hyper-tuned Boosted Decision

24.4 REASON

We are planning to use Regression learning algorithm because the target variable is numerical and continuous in nature. We will be creating ML pipeline using linear, regularized linear, tree and forest learning algorithm. We will compare and evaluate different models based on RMSE of learning algorithm [296].

24.5 TECHNOLOGY STACK

Python will be used for Data loading, preprocessing and cleaning. Using Scikit learn library, we will implement variety of algorithms to conduct above process and finally will predict the sale price of its products.

REST services has been implemented to provide a prediction of price of the products:

- REST data preprocessing: It will be the service, which does the data processing with removal or imputing the data with mean or median. Removal of the columns which doesn't any correlation with target variable
- REST data prediction: it will be the service, which will do multiple predictions using multiple algorithms as below
- Rest API with Linear Regression – Display the outcome of product and predicted price
- Rest API with Boosted Decision – Display the outcome of product and predicted price
- Rest API with Hyper-tuned Boosted Decision – Display the outcome of product and predicted price

Cloud technology utilized will be Microsoft Azure, AWS, it has been implemented at Local machine and Docker. We have acquired the dataset from Kaggle and read the data dictionary details on different websites which includes below describes attributes. We have 14204 instances and 13 attributes in the dataset, which will be spitted into Training and Test Data set. This dataset is available on public websites BigMart Dataset, With this dataset, we will predict the sale price of various products based on the learning of historical data in the datasets using different algorithm. The dataset has various data with respect to

- Item Fat Content
- Item Identifier
- Item MRP
- Item Outlet Sales
- Item Type
- Item Visibility
- Item Weight
- Outlet Establishment Year
- Outlet Identifier
- Outlet Location Type
- Outlet Size
- Outlet Type
- source

[297].

24.6 DATASET DETAILS

It has following 12 attributes with continuous and categorical values with Unique Values

- Item Fat Content has 5 unique values
- Item Identifier has 1559 unique values
- Item MRP has 8052 unique values
- Item Outlet Sales has 3494 unique values
- Item Type has 16 unique values
- Item Visibility has 13006 unique values
- Item Weight has 416 unique values

- Outlet Establishment Year has 9 unique values
- Outlet Identifier has 10 unique values
- Outlet Location Type has 3 unique values
- Outlet Size has 4 unique values
- Outlet Type has 4 unique values

24.7 DATA VISUALIZATION

The histogram in Figure 216 shows the distribution of data of different variables from the dataset. These are the important feature influencing the predicted output

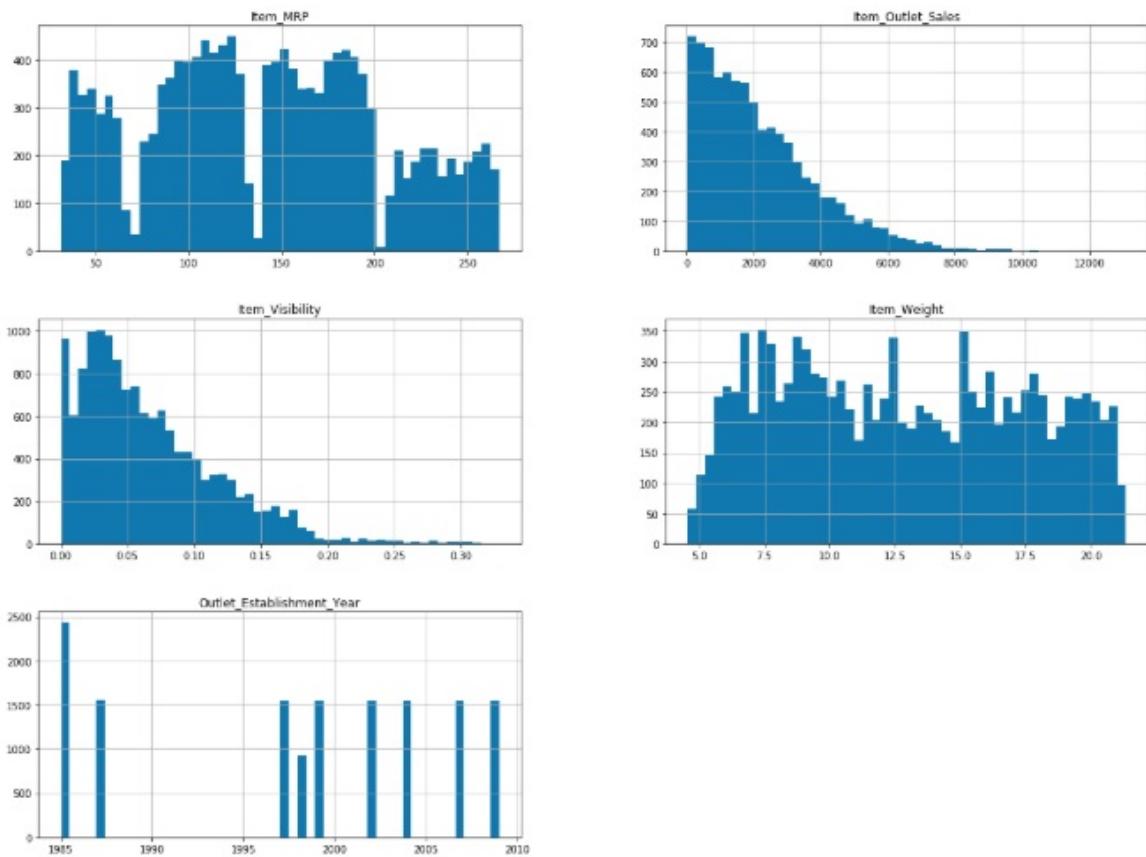


Figure 216: Histogram of Important Attributes

In Figure 217 shows the plot about the corelation between variables in the dataset. This will display the features which are higly correlated in darker color.



Figure 217: Correlation Matrix between Variables

24.8 DATA EXPLORATION

Analyzed and plotted the categorical and continuous feature summaries to see which feature is closely related with target variable. This helped us with deciding which feature are influencing the prediction.

24.9 DATA PREPROCESSING

- Missing values (2439) of item weight is replaced with mean.
- Missing values (4016) of outlet size observations, which been replaced with mode.

24.10 AZURE ML STUDIO

Azure ML studio provides the GUI interface for creating the Machine Learning Train models and Predictions. It provides a provision to integrate with Azure Cloud and expose the Web Services

24.11 TRAIN MODEL WITH AZURE

Created on Azure ML Studio, 3 Learning Algorithms used

- Boosted Decision Tree
- Linear Regression
- HyperTuned Boosted Decision Tree

As seen in the results below in Figure 218 and Figure 219 regarding the RMSE result scores, Hyper-tuned Boosted Decision Tree has provided better results.

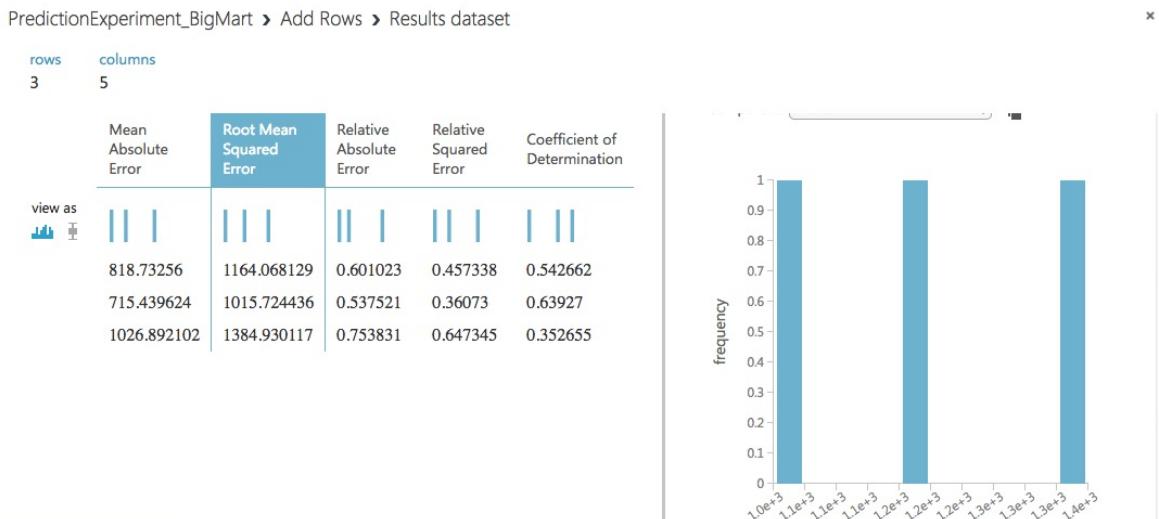
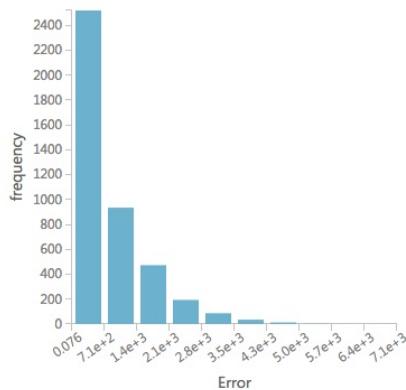


Figure 218: RMSE Comparision Results

PredictionExperiment_BigMart > Evaluate Model > Evaluation results

◀ Error Histogram



◀ Error Histogram

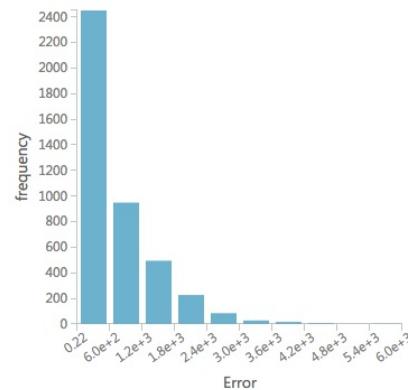


Figure 219: Hypertuned Algorithm RMSE Comparision

24.12 PREDICTIVE MODEL

Update the Trained model with Test dataset for predicting the Item Outlet Sales data. Verified and updated the data cleaning process which we have implemented for Train dataset. After converting the categorical data with indicators, we can apply the trained model.

Created predictive model using the above Hyper-tuned Boosted Decision Tree.

From the score function, have extracted only 2 columns

- Item Identifier
- Item Outlet Sales

In below Figure 220 shows WebService input and WebService Output using the selected model on the dataset with predicted value.

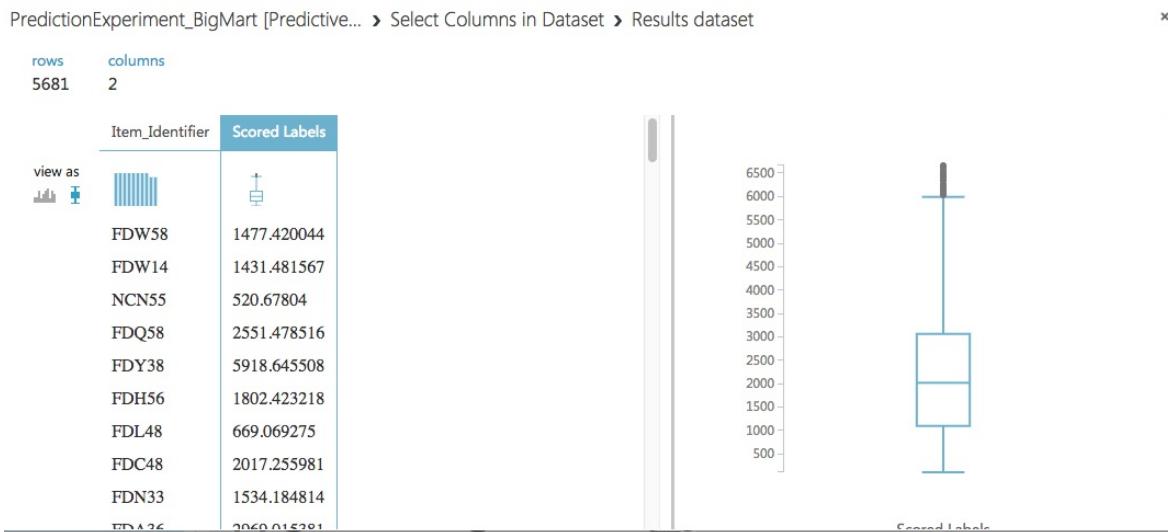


Figure 220: Output of Prediction from Models

24.13 WEB SERVICE DEPLOYMENT

Once the Prediction model has been executed successfully, it can be deployed as web service from Azure ML Studio.

Figure 221 shows the generated API key, which will be used for Azure Cloud deployment.

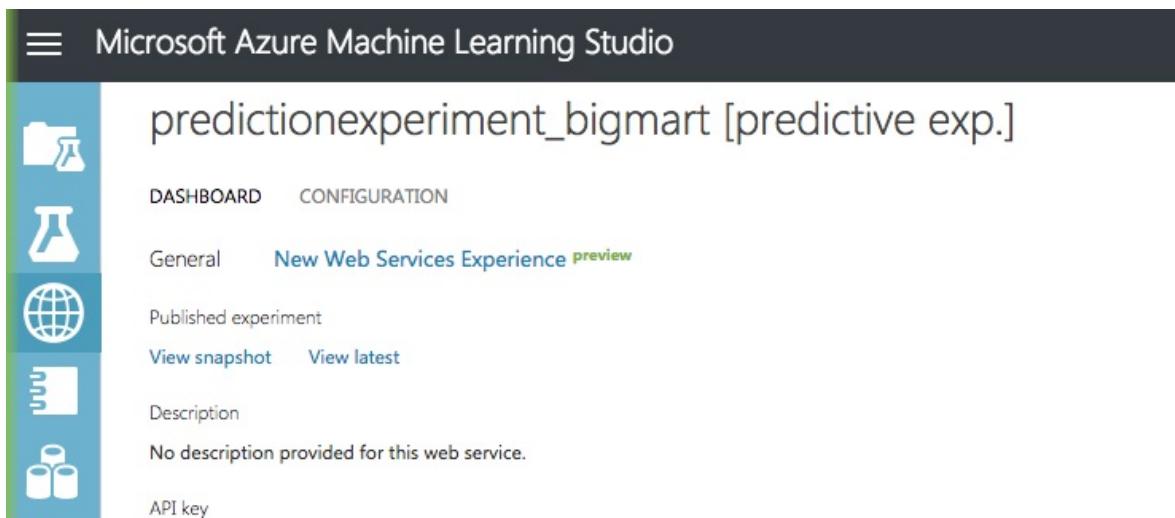


Figure 221: Azure MLStudio Webservice Deploy

It will provide an option to Test web service locally with below options

- Click on Test button enabled at the bottom of the screen
- Download the CSV file from the tool to test the Web API with prediction model.

24.14 AZURE CLOUD DEPLOYMENT

Once Web Service is created locally, It creates a hyper link with name of the web service. On clicking this hyperlink service dashboard opens up that let the developer set up and configure web service on Azure cloud for consumption as shown in the below Figure [222](#).

It provides test tab on the dashboard where we can provide inputs and get the prediction values given by model after clicking on Test Request response button.

This step will assure that, the web services are working as expected.

The screenshot shows the 'Test' tab of the Azure ML Studio interface. At the top, there are tabs for 'Quickstart', 'Dashboard', 'Batch Request Log', 'Configure', 'Consume', 'Test' (which is selected), and 'Swagger API'. Below the tabs, the title is 'PredictionExperiment_BigMart [Predictive Exp.]' and the sub-section is 'default'. A 'View in Studio' button is located in the top right corner.

The main area is divided into two sections: 'input1' and 'output1'. Under 'input1', there are ten input fields with dropdown menus:

- Item_Identifier: FDW58
- Item_Weight: 20.75
- Item_Fat_Content: Low Fat
- Item_Visibility: 0.007564836
- Item_Type: Snack Foods
- Item_MRP: 107.8622
- Outlet_Identifier: OUT049
- Outlet_Establishment_Year: 1999
- Outlet_Size: Medium
- Outlet_Location_Type: Tier 1
- Outlet_Type: Supermarket Type1

Under 'output1', there are two displayed results:

- Item_Identifier: FDW58
- Scored Labels: 1477.42004394531

A 'Test Request-Response' button is located at the bottom left of the input section.

Figure 222: WebService Configuration Details

After clicking consume tab from Dashboard, It will display option for Response Request Web Template link.

Copy the request response link generated on the page.

Once clicked on the link, it will redirect to Azure cloud configuration using Response Request Web application.

Azure ML Request, Response Service Web App In Azure cloud, it need to created as

- Create the request response service web app
- Create Resource Group
- Add Model Management services

- Click on the URL link generated under resource group
- Update the Settings with API POST URL
- Update the API key generated from Web service from Azure ML studio.

Expose the Web Service from Azure cloud

Click on the below link to access the prediction web service *
<https://predictbigmart.azurewebsites.net/>

Entered the values used to for testing locally, from the Figure 223, we see that amount should matched which confirms service is functioning as expected for predicting values.

 PredictionExperiment_BigMart [Predictive Exp.]

Input1 Parameters

Item_Identifier <input type="text" value="FDW58"/>	Outlet_Identifier <input type="text" value="OUT049"/>
Item_Weight <input style="width: 100px; height: 10px; border: 1px solid #ccc; border-radius: 5px; vertical-align: middle;" type="range" value="20.75"/> 0 100	Outlet_Establishment_Year <input style="width: 100px; height: 10px; border: 1px solid #ccc; border-radius: 5px; vertical-align: middle;" type="range" value="1999"/> 0 2,999 1999
Item_Fat_Content <input type="text" value="Low Fat"/>	Outlet_Size <input type="text" value="Medium"/>
Item_Visibility <input style="width: 100px; height: 10px; border: 1px solid #ccc; border-radius: 5px; vertical-align: middle;" type="range" value="0.0075"/> 0 100 0.0075	Outlet_Location_Type <input type="text" value="Tier 1 Tier 2 Tier 3"/>
Item_Type <input type="text" value="Snack Foods"/>	Outlet_Type <input type="text" value="Supermarket Type1"/>
Item_MRP <input style="width: 100px; height: 10px; border: 1px solid #ccc; border-radius: 5px; vertical-align: middle;" type="range" value="107.86"/> 0 100 107.86	<input style="background-color: #0078D4; color: white; border: none; padding: 5px;" type="button" value="Submit"/>

Result

Label	Value
output1	
Item_Identifier	FDW58
Scored Labels	1477.42004394531

Powered by Azure Machine Learning

Figure 223: WebService Result

Batch Mode for Web Service Execution Download the CSV generated from Azure ML Studio

- Open the CSV, it will be open with Web API built in
- Use Sample data link on the API
- Select the range of columns and provide as input to API
- Select the cell from where the prediction values needs to be displayed.
- Click on Prediction button. It will generate the prediction values for all the selected Input entries with Item Identifiers.

Figure 224 shows the API got created on the CSV format file, for generating the predicted values using the features from the dataset. These predicted values matches with the values generated from the local or webservice created.

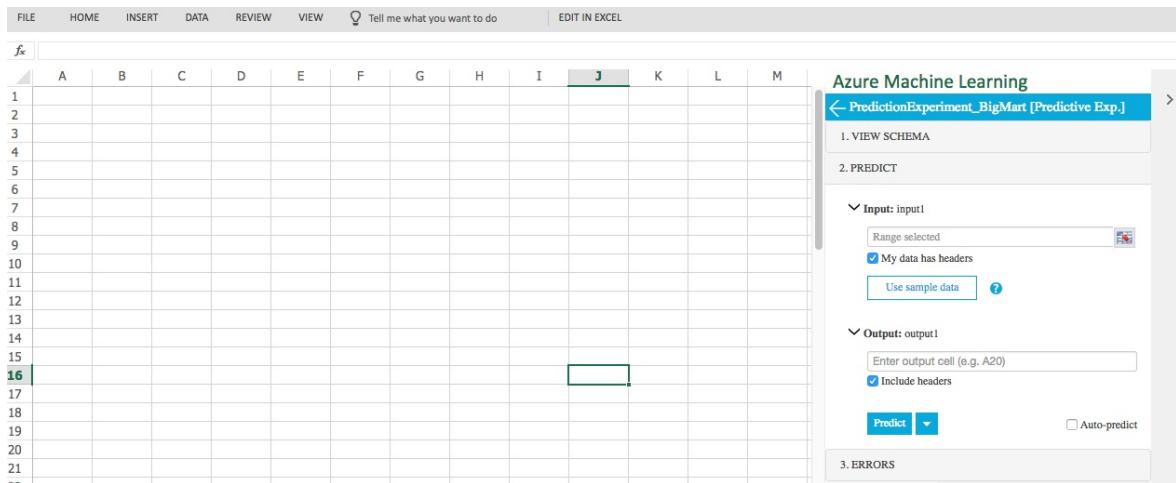


Figure 224: Built API on CSV File

24.15 DRAWBACK OF USING AZURE ML STUDIO

Azure ML studio provides easy to use drag and drop objects for all steps (from model initialization to deployment) such as data preprocessing, training model, fitting model, evaluating model, tuning model and prediction. It is every easy to use and handy tool for data scientists and data engineer. However, it has a drawback; developers are not able to generate the code that is generated at the backend for further experimentation and testing.

24.16 AZURE WITH NOTEBOOK INSTANCE

Notebook instance of Azure ML Studio was used to execute python code file for benchmarking. Code containing data preprocessing, training model, fitting model, evaluating model, tuning model and prediction steps was executed on Azure notebook instance. It took 15.2 ms for overall batch prediction using notebook. Computation time using model saved on disk was recorded as 1.95 mili seconds as shown in below code.

24.16.1 Code

```
Predict on original Test Set using Random forest model with Hypertune

%%time
overallprediction=clfgrdmof.predict(testr[predictors])
CPU times: user 15 ms, sys: 761 µs, total: 15.7 ms
Wall time: 15.2 ms

print(overallprediction)
[1648.57907925 1413.2015807 700.49019691 ... 11922.98505294 3487.44068317
 1308.81770146]

import pickle
filename = 'finalized_model.pkl'
pickle.dump(clfgrdmof, open(filename, 'wb'))

# load the model from disk
loaded_model = pickle.load(open(filename, 'rb'))
Test1 = loaded_model.predict(testr[predictors])

%%time
print(Test1)
[1648.57907925 1413.2015807 700.49019691 ... 1922.98505294 3487.44068317
 1308.81770146]
CPU times: user 1.68 ms, sys: 277 ms, total: 1.96 ms
Wall time: 1.95 ms
```

24.17 AWS WITH NOTEBOOK INSTANCE

AWS Sagemaker service was used to benchmark performance of model on AWS cloud. Python notebook file containing code for data preprocessing, training model, fitting model, evaluating model, tuning model and prediction was executed on AWS notebook instance. Execution time of prediction was captured for comparison. It took 8.2 ms for overall batch prediction using notebook. Computation time using model saved on disk was recorded as 371 micro secs.

24.17.1 Code

```
Predict on original Test Set using Random forest model with Hypertune

%%time
overallprediction=clfgrdmof.predict(testr[predictors])
CPU times: user 12 ms, sys: 0 ns, total: 12 ms
Wall time: 8.2 ms

print(overallprediction)
[1610.61752003 1401.48146969  578.3748237 ... 1854.50378336 3521.16741962
 1286.86433971]

import pickle
filename = 'finalized_model.pkl'
pickle.dump(clfgrdmof, open(filename, 'wb'))

# load the model from disk
loaded_model = pickle.load(open(filename, 'rb'))
Test1 = loaded_model.predict(testr[predictors])

%%time
print(Test1)
[1610.61752003 1401.48146969  578.3748237 ... 1854.50378336 3521.16741962
 1286.86433971]
CPU times: user 0 ns, sys: 0 ns, total: 0 ns
Wall time: 371 µs
```

24.18 LOCALMACHINE INSTANCE

Python notebook file was executed on local instance for comparing performance with cloud. Data preprocessing, training, fitting and prediction steps were performed on local python instance. Start time and end time was recorded and execution time was calculated. It was observed that execution and prediction

took more time compared to execution time on cloud. It took approximately 10 minutes on local instance.

24.18.1 Local Execution - Prediction Time

```
Local Start Time:  
2018-12-08 01:39:40.654383  
Local End Time:  
2018-12-08 01:39:40.684909  
[1710.00908356 1425.18637329 566.05070462 ... 1825.71847362 3565.54132359  
1278.2714367 ]  
[1710.00908356 1425.18637329 566.05070462 ... 1825.71847362 3565.54132359  
1278.2714367 ]  
[1710.00908356 1425.18637329 566.05070462 ... 1825.71847362 3565.54132359  
1278.2714367 ]
```

24.19 DOCKER

Docker Image was created on local environment. Execution of docker image of model predict prices for passed dataset. This code can be reproduced using Docker commands. Once replicated locally, running localhost endpoint url created by docker image on port- 8080 will predict prices of all items of dataset.

Link for localhost

- <http://localhost:8080/cloudmesh/prediction>

24.20 CODE REPRODUCING STEPS

Environment - Ubuntu 16.04 , python

24.20.0.1 Testing using local python instance

step 1 - Download project-code folder from github to local drive on Ubuntu

step 2 - On terminal go to project-code folder

step 2 - Run below command to execute locally

```
python Project-BIGMartPrediction.py
```

24.20.0.2 Testing Using Docker Image

step 1 - Download project-code folder from github to local drive on Ubuntu

step 2 - On terminal go to project-code folder

step 3 - Run below command

```
make service
```

step 4 - Run below command to build docker image by name
cloudmeshprediction

```
make docker-build
```

step 5 - Run below command to start service. This will create service endpoint and will start the service on local environment

```
make docker-start
```

step 6 - Open new terminal and run below command

```
curl -H "Content-Type: application/json" http://localhost:8080/cloudmesh/prediction
```

24.21 PERFORMANCE COMPARISON

Environment	Description	Elapsed Time
Azure Cloud	Notebook Instance	15.2 ms
AWS Cloud	Notebook Instance	8.2 ms
Local	Python Instance	0.03 sec

From above comparison we note that AWS cloud is faster giving performance of 0.0082 seconds whereas prediction on Azure has took 0.0152 seconds. lowest being the local environment with 0.03 second.

24.22 CONCLUSION

In this project, below two models have been implemented and hyper-tuned.

- Boosted Decision
- Linear Regression
- HyperTuned Boosted Decision

Best model was decided based on accuracy and that was used for prediction of sales price of items across all store outlets.

Model was deployed on Azure Cloud, AWS Cloud and Local machine for predicting prices. Docker image was created for reproducing. Performance was benchmarked on different cloud and local environment. On comparison we observed that prediction performance was best on AWS cloud.

24.23 APPENDIX

Web service was deployed on Azure Cloud. Weservice endpoints was exposed to predict price of passed Item Identifiers.

Deployment and usage playback Video of using exposed webservice endpoint is uploaded at below location.

- <https://www.youtube.com/watch?v=xrLto4XPn1o&t=518s>

24.24 ACKNOWLEDGEMENT

The authors would like to thank Dr. Gregor von Laszewski for his support and suggestions to write this paper.

24.25 WORKBREAKDOWN

Arijit and Ritesh worked on data processing, Data exploration and designing the Machine learning Algorithms. We both have brainstromed on visualizing data. Arijit worked on Azure cloud and creating web service and Ritesh worked on Docker image creation on local machine. We both worked on comparing performance benchmarks. Compared the time running on AWS cloud to prepare the time comparision chart and project report.

25 VISUALIZING GLOBAL COMMODITY STATISTICS USING AZURE, PYSPARK AND PYTHON

Abhishek Rapelli, Harika Putti, Pavan Kumar Madineni

arapelli@iu.edu, haputti@iu.edu, pmadinan@iu.edu

Indiana University, Indiana University

hid: fa18-523-79, fa18-523-81, fa18-523-82

github: [cloud](#)

code: [cloud](#)

Keywords: Exploratory Data Analysis, Python, PySpark, HDFS, HBase, Watson Analytics

25.1 ABSTRACT

Economists, governments, trading organizations and even general public may be curious to know about the international trading patterns between different nations across the globe on various commodities. For example, one may be curious to get answers to some interesting questions like which is the largest importer of steel in the world? Which country has the highest growth in sheep production? What is America's chocolate consumption this year? These kinds of questions can be answered by analyzing the huge volume of data that is collected from the trading transactions that happens globally among various countries. The idea is to do exploratory data analysis to find patterns and insights that may be helpful in answering such questions or raise more such new questions from observations.

25.2 INTRODUCTION

This dataset contains import and export volumes for 5,000 commodities across most of the countries on the globe over the past 30 years. Trade across the globe plays a prominent role in the economy of a nation. We are looking ahead to check for any peculiar patterns in the commodities traded by different countries

in accordance with time in the past 30 years. We have imported the dataset from the UNData site published by the United Nations Statistics Division. The dataset is a 1.2 GB sized dataset and we aim to illustrate meaningful correlations among different volumes of commodities traded by different countries in different years.

25.3 IMPLEMENTATION

25.3.1 Data

The dataset was provided by United Nations Statistics division on the UNData website. It consists of data for over 30 years until 2016 covering all nations in the world and around 5000 plus commodities. It has over 8.5 million transactions/records with 10 columns. The total size of the dataset was 1.2 Giga bytes. The 10 variables names are: country, year, commodity code, commodity name/type, flow (import or export), trade in USD, weight in kg, quantity name, quantity and category. There are some missing values as well as outliers in the data. Hence, the data is not ready to be used directly and needs good amount of cleaning and preprocessing.

25.3.2 Related work

We have found an earlier work on this dataset, that focused on specific goals about finding the trading patterns for a nation. It has taken only a sample of this dataset than whole data and was more focused on specific countries[298].

25.4 TECHNOLOGIES USED

- We have coded entirely in python 3.0 which is a high-level object-oriented programming language used for general programming [299].
- Seaborn is one of the most widely used data visualization packages available in python that is built on top of matplotlib. It provides an interface to draw appealing graphs and draw enlightening factual illustrations. We are primarily using seaborn to draw valuable insights about the trade statistics of different countries out of this huge 1.2 GB data [300].

- Altair is a statistical visualization library available in python based on Vega and Vega-lite. It provides different visualization features that enable to create a wide variety of statistical visualizations very rapidly. In addition to seaborn we are also using Altair to create interactive visualizations to dive deep into the correlations between the trade among different countries [301].
- Folium is one of the famous visualization libraries available in python used predominantly to visualize geospatial data. Folium enables one to create the map of any location given its latitude and longitude values [302].
- Matplotlib is one of the most widely used visualization libraries in python that enables producing quality figures in interactive environments across diverse platforms. The major benefit of this library is it can be used to generate visualizations in python scripts, ipython shells, jupyter notebook and also on many other development environments [303].

The entire data is stored in Azure cloud and is accessed directly from cloud rather than from the local machine. All the interactive visualizations are created using data from cloud in jupyter notebook.

25.5 VISUALIZATIONS

We will look at different kinds of visualizations like boxplots, frequency distributions, histograms, bar plots, density plots, etc. of each variable/feature depending on if it is categorical or numerical variable. We will also look at heat maps on correlation matrix between the variables. We may also use geographical plots to visualize the trade volume of each country on map. We shall be using python packages for visualization like Matplotlib, Seaborn, Plotly, Geographical plots majorly.

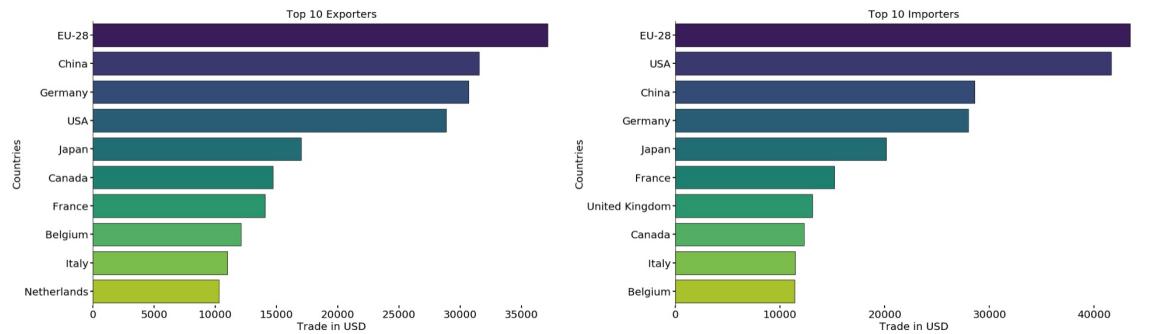


Figure 225: Top Importers and Exporters by Country

Figure 225 shows the line graph of top importers and exporters based on Trade in USD. European Union leads the list of both top 10 importers and exporters, but it has to be noted that EU-28 is a region comprising 28 countries. Most of the countries are heavy import based countries with China being an exception. Countries like USA in particular have a huge trade deficit i.e. the exports are much smaller when compared to imports. Only a few countries like China and Germany have decent trade surplus i.e. these countries import less than their exports which acts like a tremendous boost to their economy.

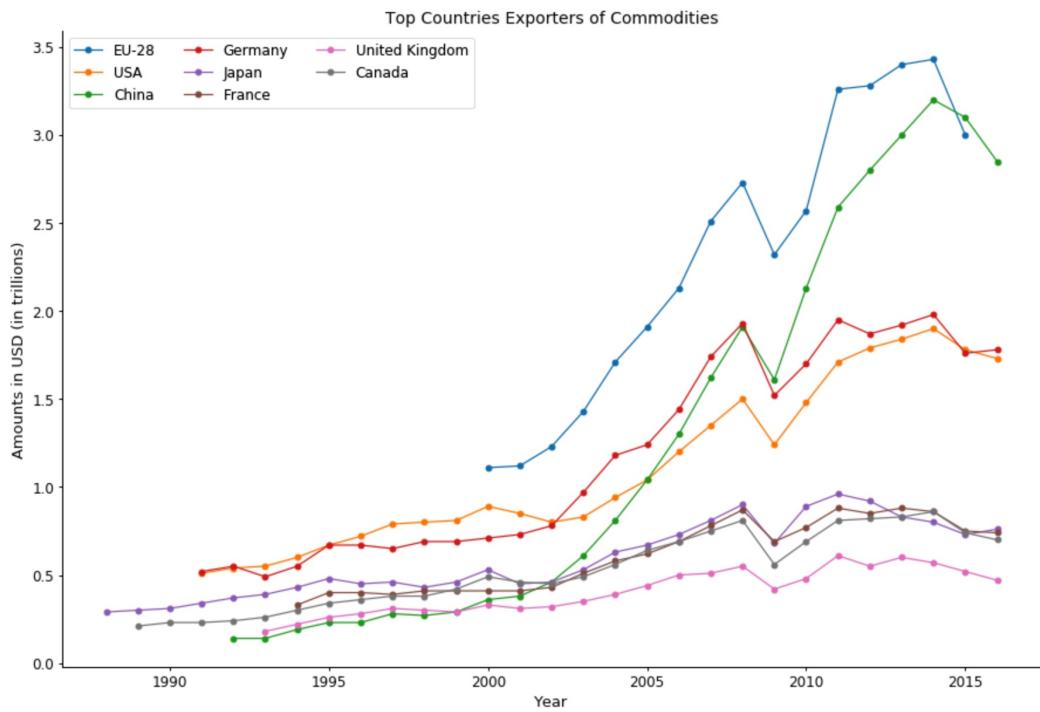


Figure 226: Top Countries Exporter of Commodities

Figure [226](#) shows the line graph of top 8 countries leading in Export.

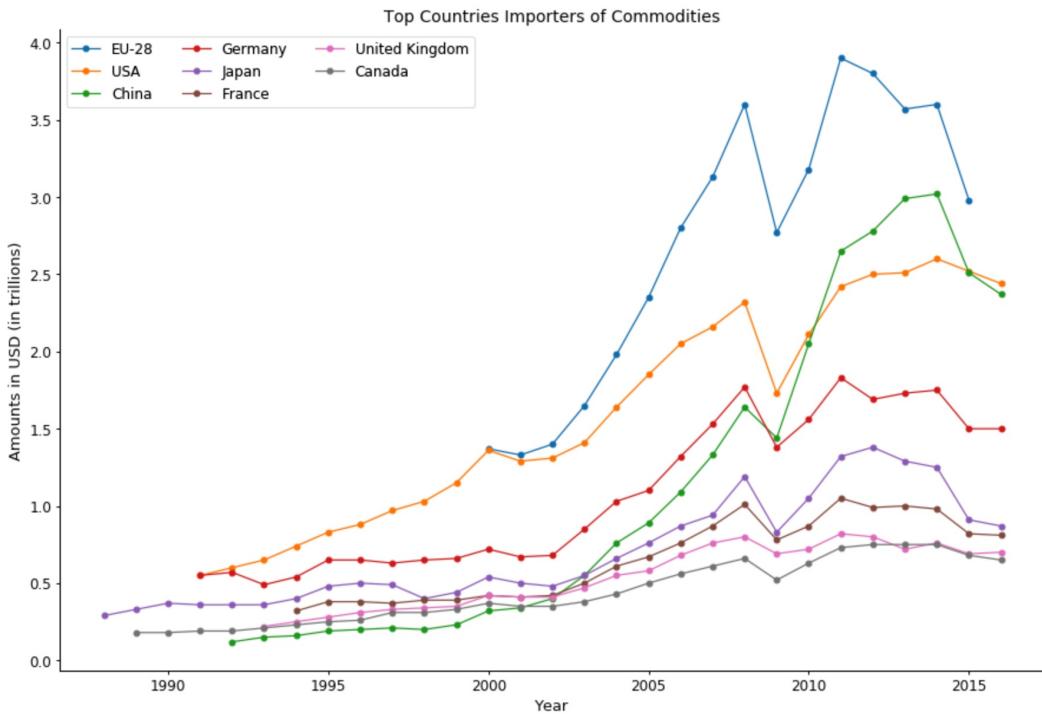


Figure 227: Top Countries Importer of Commodities

Figure 227 shows the line graph of top 8 countries leading in Import in terms of money. EU is the leading country in imports as well. While USA is also a top contender, China is the only country with a very rapid increase in imports, even surpassing USA after 2010. Around 2015, the imports seem be competitive between China and the US, cut to now where the USA and china seem to have an open trade war. It would be interesting to see which country takes the lead in the future.

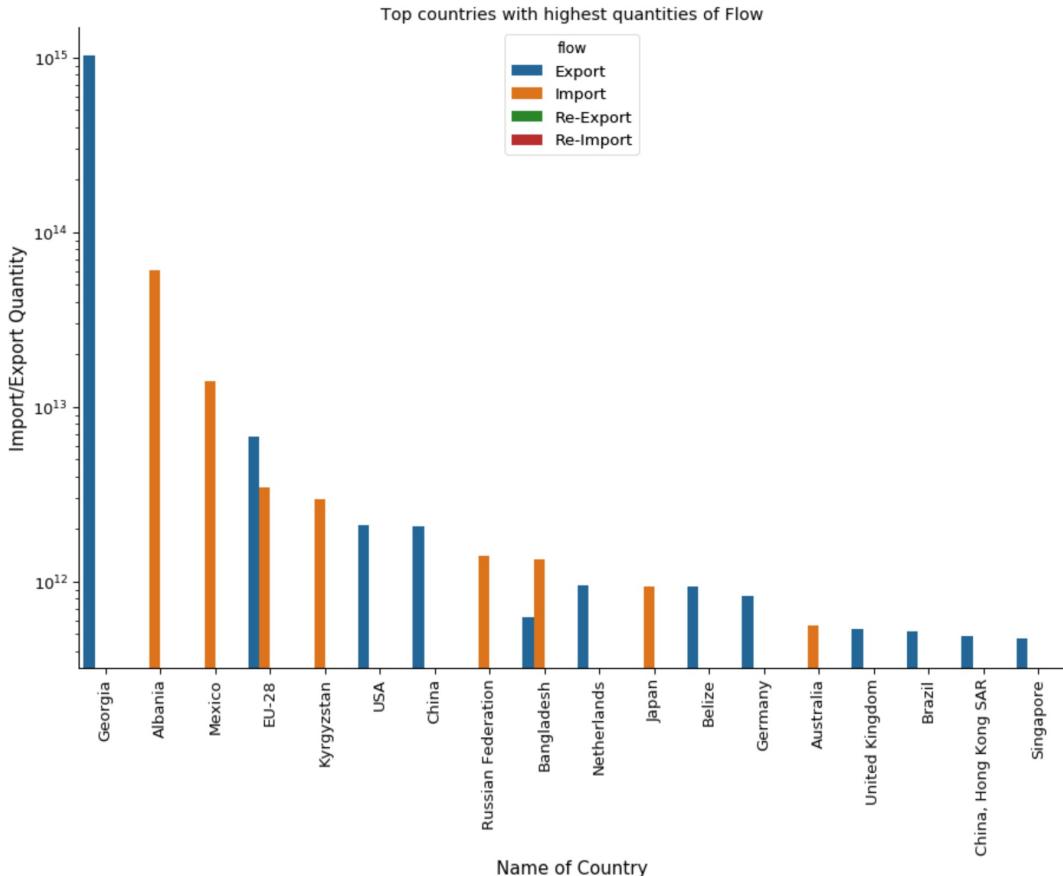


Figure 228: Top Countries With Highest Quantities

Figure 228 shows the bar graph of top countries with top quantities of import and export. The graph shows the highest importers as well as exporters in terms of quantity. Georgia surprisingly is a leading expert in exporting large quantities followed by Albania which is a top importer.

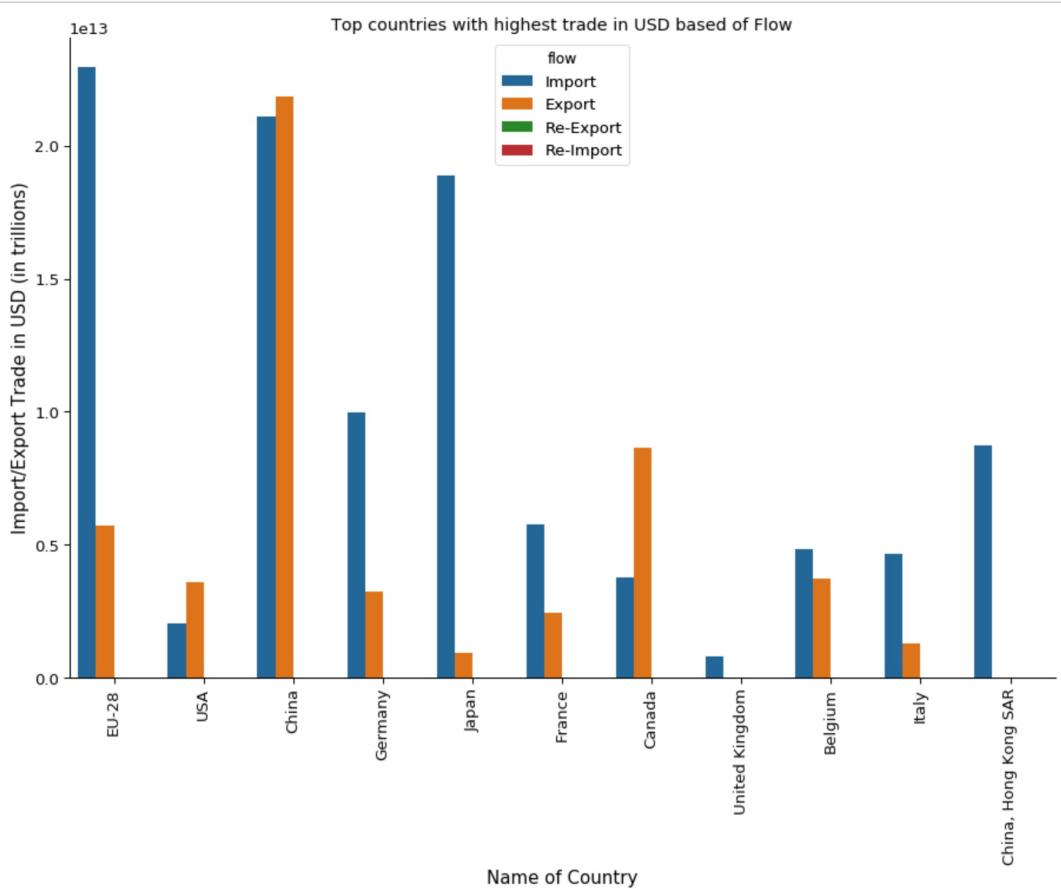


Figure 229: top countries with highest trade in USD

Figure 229 shows the line graph of top countries with highest trade in USD in import and export.

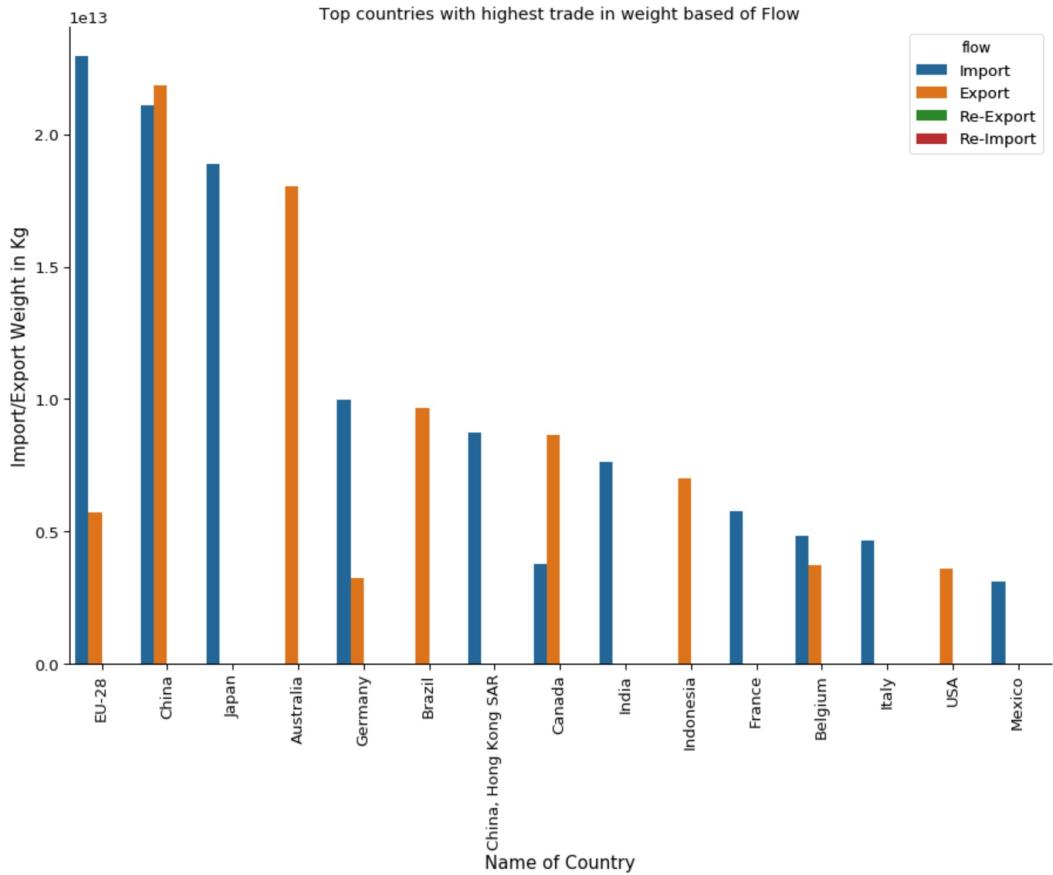


Figure 230: Top Countries with Highest Weight Flow

Figure 230 shows the line graph of top countries with highest weights of Export and Import.

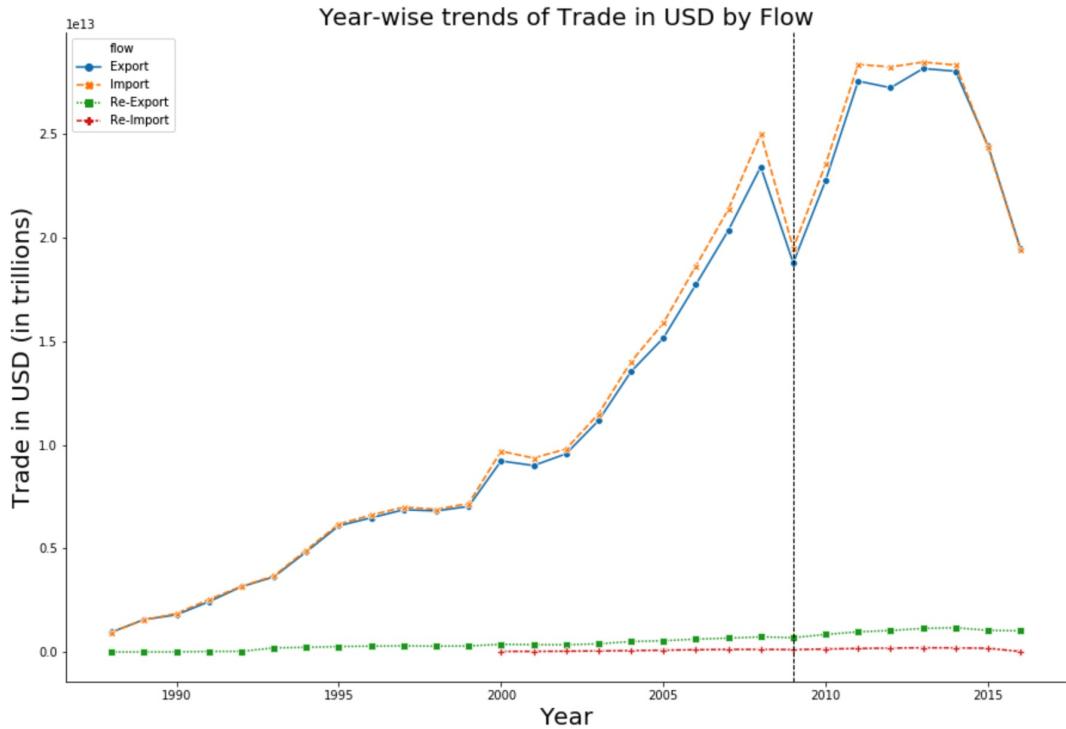


Figure 231: Yearwise trends in trade

Figure 231 shows the line graph of Year-wise trends in trade in USD. It can be clearly seen from the graph that there is a huge dip in the trade in US dollars in the year 2009 which is otherwise very consistently progressive. The trade has recovered immediately by the next year undeterred by this dip in 2009.

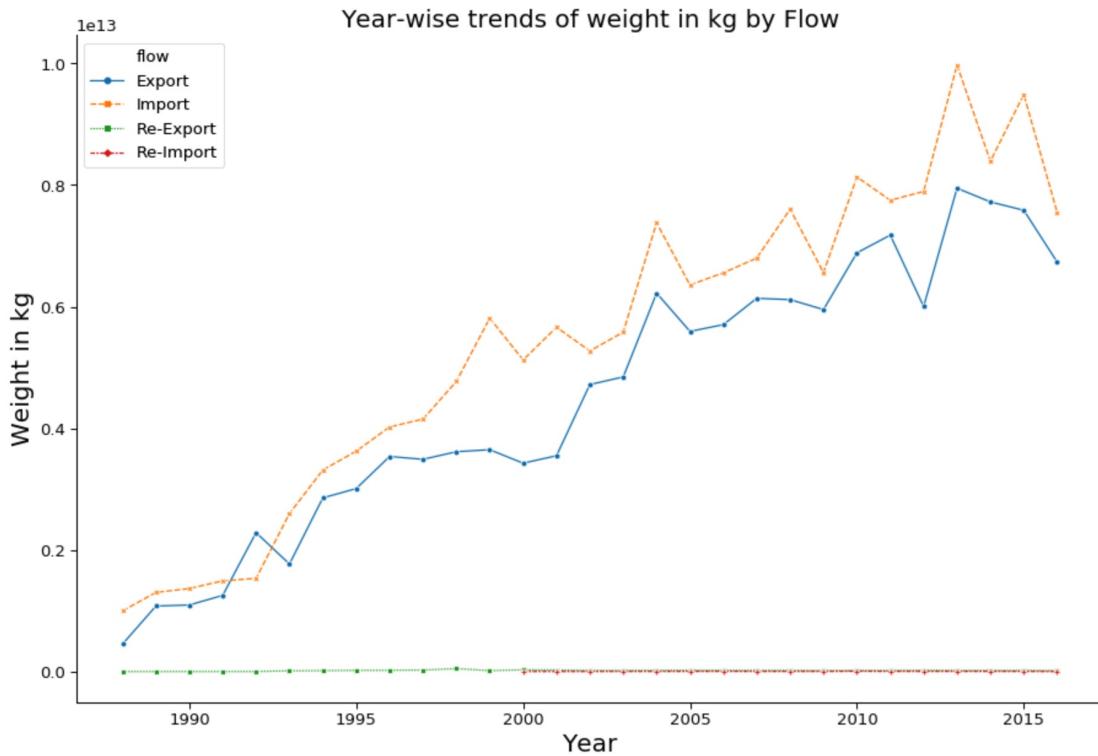


Figure 232: Yearwise trends in weight

+???

trends in weight shows the line graph of Year-wise trends in weight in kg.

25.6 SUMMARY

There are a lot of valuable insights that could be discerned from this data. This is a very large dataset that has scope for varying analyses to be done. A few important details learnt out of these visualizations are; China is way ahead of all the other countries as well as other regions in terms of cumulative trade imports and exports. Most traded items among all the countries in general are sheep, goats, bovine animals, poultry and swine which clearly show that these animals are traded for human consumption. China is clearly the leader in terms of both volume as well as consistency in world trade over the years. An astonishing 40 percent of Chinese commodity exports is constituted by woven fabric. To sum up, similar analyses can be performed to extract similar trade statistics for each country or a group of countries as a region, but we have selected only a few aspects of trade to perform analysis for this project.

25.7 FUTURE WORK

Though in this project we will be focusing more on big data storage and retrieval, data preprocessing and exploratory data analysis, there is scope to do more work on this by building predictive models using various machine learning, deep learning or forecasting models, depending on the problem statement or goal we set.

25.8 ACKNOWLEDGEMENT

We would like to thank the United Nations Statistics Division for kindly publishing this dataset on UNData site. We would also like to thank the professor Dr. Gregor von Laszewski for his valuable inputs in helping us to choose this dataset and also for all the guidance he has provided. We would like to appreciate the effort of the associate instructors for their timely help on Piazza.

25.9 WORK BREAKDOWN

The authors Abhishek Rapelli, Harika Putti and Pavan Kumar Madineni have worked together and contributed nearly equally in preparing this report. All three of us have organized multiple collaborative sessions to extensively search for a large dataset that would meet the course requirements as well as which would also facilitate in doing quality visualizations. Individually, we have each done the following:

- Abhishek Rapelli: Microsoft Azure deployment
- Harika Putti: Parsing and writing the project code
- Pavan Kumar Madineni: Analyzing results and report writing

26 BIG DATA IN EDUCATION

Tong Wang, Yeyi Ma

wangton@iu.edu, yeyima@umail.iu.edu

Indiana University Bloomington

hid: fa18-523-73, fa18-523-74

github: [https://github.com/yeyima](#)

Keyword: Education

26.1 ABSTRACT

The advancement of technology has affected education in a big way. In recent times, the educators and administrators mainly rely on the Student Information Systems (SISs), Learning Management Systems (LMSs), and Admissions Management Systems to streamline classroom, school and campus operations[304]. Therefore, as institutions continue to adopt such systems, user data is generated, which helps the education stakeholders including the parents to make more informed decisions on how the learning process can be improved. As a result[305], highlights that big data has fundamentally changed the way education is offered, and institutions that invest in the big data and successfully derive value from their data will have a distinct advantage over others. Moreover[306] anticipates that in the near future there will be a performance gap between the institutions with big data systems and those that have not implemented such systems. This is because as time goes by, more relevant data will be generated, and the emerging technologies and digital channels will offer better acquisition and delivery mechanisms. For instance, teachers can mine learning patterns to see how students master specific subjects and experimental designs. Therefore, this paper analyzes how big data is relevant in the education system.

Introduction

Today, many organizations are collecting, storing, and analyzing massive amounts of data. This data is commonly referred to as “big data” because of its

volume, the velocity with which it arrives and the variety of forms it takes. Therefore, big data can be described as the process which is used to convert continuous, analog information into discrete, digital and machine reliable format. Examples of big data include web data, text data, time and location data, smart grid, and social network data. Such data can be utilized in educational institutions by using the data-driven approaches that make it possible to study learning in real-time and offer systematic feedback to students and teachers[305]. As a result, it is crucial to explore how the big data is used to mine learning information for insights regarding student performance and learning approaches.

26.2 FORMS OF BIG DATA IN EDUCATION INSTITUTION

26.2.1 Administrative Data

These type of data refers to the demographic, behavioral, and achievement data collected through schools, government agencies, and their contractors. It is obtained by sampling many participants, which can be longitudinally at prescribed, or regular intervals. In most cases, this type of data is made up by attendance records, test scores, transcripts, and surveys. Examples of the administrative data that can be taken as big data include international test scores, a national assessment of educational progress data, and the behavior data that is collected by the U.S. Department of Education's Civil Rights Data Collection.

Learning Process Data

They are continuous or near-continuous, fine-grained records, mostly in the form of digital interactions of students behaviors to illuminate learning processes. The learning process data is considered big data since it involves many participants and a large number of variables in regards to a single person. Examples of learning process include online assessments and courses such as the massive open online courses.

26.3 THE VALUE OF SYSTEMATIC, REAL-TIME DATA

The use of a computerized system in schools helps the instructors to assess the learning process in real-time. Therefore, by the use of data mining and analytic

data software, it is possible to get the immediate feedback in regards to how the learning process is taking place. In the process, the analytic software analyzes the underlying patterns, which help advice on the best practices to undertake for improving or maintaining high performance. An example of the application of the real-time data is in Connected Chemistry that enables students to understand more in-depth the key concepts in molecular theory and gasses[306]. Therefore, the use of mining and analytic software allows the teachers to mine learning patterns to analyze how the students understand the various areas of chemistry.

Moreover, Joseph Beck and Jack Mostow are known individuals who have examined the benefits of the big data in education. In their experimentation, they applied the intelligent tutor software to study how the students comprehend a story. In the process, they analyzed the reading time, word knowledge, reading mistakes, and the help requests. Beck and Mostow concluded that when students re-read an old story, it assists them understands the words better than reading a new story.

The other example that showcases how big data relates to education is research carried out by James Theroux of the University of Massachusetts at Amherst. The study focused on students who did average courses, aiming to analyze their perceptions of the uses of online learning compared to the traditional approach. Theroux found out that majority of the students preferred learning via online platforms against the conventional methods. Their preference was mainly because it is a modern method that relates to them. Therefore, the big data technologies can help the institutional faculties to assess the degree to which the students understand the various concepts when learning via online as to other methods. Similarly, the same issue was raised by scholars Robert Perkins and Margaret McKnight of 139 who also concluded that most students prefer using online platforms for leaning as they enjoyed their collaborative and interactive nature[306].

Therefore, the big data is very applicable in schools. For instance, the data collected in real-time can help evaluate the performance of a learning process. The data-driven approaches make it possible to study learning in real-time and offer regular feedback to students and teachers[305]. One advantage of the real-time data is that it provides more accurate results than other methods since the data is not manipulated or distorted before it is analyzed. Thus, schools need to have learning analytics modes to scrutinize the learning process in real-time to

identify areas of improvements.

26.4 LEARNING ANALYTICS

In recent times, learning analytics has become an area of research and application. However, unlike the traditional data mining, it draws on a broader array of academic disciplines, incorporating concepts and techniques from information science and sociology. Therefore, the approach primarily focuses on the analysis and reporting of the data. In other words, it aims to look at the application of known methods to answer the crucial questions that affect the performance in schools. Thus, learning analytics can monitor and predict student's learning performance and brings out the potential issues that the administration needs to address.

26.5 REQUIREMENTS

When establishing the requirement of the big data implementation, it is important to adopt a user-center design approach. It is an appropriate method rather than specifying the elements at the beginning for various reasons. According to Williamson[306], when requirements are collected at the beginning of the project, it may result in later problems in the system development. On the other hand, a user-centered approach enables the development team to establish the communication between users and developers and to gather different requirements on each side. However, when determining the conditions for big data implementation, the procedures are guided by ISO/IEC/IEEE 42010:2011 “Systems and software engineering- Architecture description.” The following are various requirements that need to be established to implement the system successfully.

26.5.1 Prediction

The first requirement of big data in school is to develop a model that can help infer a single aspect of the data. For instance, it can assist detect students behaviors when they are involved in activities such as gaming, engaging in off-task behavior. Moreover, the predicting model is also applied to study the behavior of the students in an online learning environment by examining their

participation in discussion forums and taking practice tests. Therefore, it is important to have a prediction model to forecast and understand student educational outcomes such as success on posttests after tutoring.

26.5.2 Clustering

The second requirement needs one to find data points that can assist in grouping the collected information. For instance, one can group the students based on their learning difficulties and interaction patterns. For example, in an online learning set-up, the analytics can examine the student's cognitive interview and also how they post in the discussion board.

26.5.3 Relationship Mining

It involves establishing the relationship between variables in a dataset and then creating the rules to be used in later stages. For example, it is important to identify how students within a specific cluster relate. One of the methods used is the association rule mining, which assists in determining the behaviors students. The other method applied to discover relationship is sequential pattern mining which aids in capturing the connections between occurrences of subsequent events. For instance, it is used in detecting students regressing to make errors in mechanics when they are writing with more complex and critical thinking techniques.

26.6 ARCHITECTURE

When designing big-data applications, one needs to have particular methods, tools, and technologies that can handle data efficiently from the initial stage of data collection to the final stage of the interpretation. As a result, a meaningful big data architecture should consider the characteristics of the data extracted. For this project, the architecture is divided into two main section: data management and data analytics. The data management process acquire, govern, integrate, secure and store data that is prepared to be used in data-analytics methods. The data management system is composed of five different categories: distributed file system, cluster management, data store, governance and security, and data ingestion. On the other hand, data analytics involves data modeling, analyzing and interpreting to transform raw data into valuable knowledge. It is divided into

distributed data, processing and programming, visualization, data analysis, and pre-processing data components.

26.6.1 Data Management

26.6.1.1 Distributed File System (DFS)

As shown in the diagram above, DFS is the lowest level of the architecture. It is used for the storage and maintains large amounts of data across multiple nodes of commodity hardware. However, it is designed in such a way that it conforms to master and slave node architecture.

26.6.1.2 Cluster Management

The components help in deploying, scheduling and orchestrating the jobs across the other parts of the architecture to ensure there is a readily available and highly scalable computing infrastructure.

26.6.1.3 Distributed Data Processing and Programming

It is used to process the big amount of batch data and millions of data in real time.

26.6.1.4 Data Store

It is used for the storage of the diverse and large number of data generated throughout the architecture. To increases the efficiency of the architecture, it is important to have a distributed, scalable, schema-free and fault-tolerant data stores.

26.6.1.5 Visualization

Visualization entails representing the massive amount of data in graphical formats for easy interpretation.

26.6.1.6 Data analysis

Data analysis is the process of deploying analytic data software to analyze the raw data and present useful, meaningful information to the end users. Therefore, an analytic model is used to infer knowledge by finding patterns and drawing conclusions with the aid of specialized tools and algorithms.

26.6.1.7 Data pre-processing

This step ensures that the data stored and used for interpretation is reliable. During this stage, the data is cleaned to eliminate noises, errors, or incomplete data to improve the overall success of data analysis.

26.6.1.8 Governance and Security

These entails the rules laid out to ensure the data is secure. Within the big data architecture, there exist several governance policies. For instance, Apache Atlas provides a scalable and extensible set of core foundational governance services. On the other hand, Apache Ranger gives the details of how to successfully implement a secure data framework to monitor and manage the data across the system.

26.6.1.9 Data Ingestion

It helps in transferring data from outside sources to the internal systems within the architecture. Since the big data involves a large volume of data and velocity characteristics of big data, data ingestion needs to be resilient and fault-tolerant.

26.6.1.10 Application

The application layer helps present the analyzed data to the under-users. An example of application is Nutch, which is a production-ready web crawler.

In conclusion, the architecture represents a key part of the big data. For that reason, it is crucial for an educational institution to try and create a unified information architecture, to ensure it leverages all types of data. Moreover, a well-designed architecture promotes a unified vision for information management and analytics. Therefore, it is important to get a better

understanding of the role of the implementation of the big data, which would provide a better context for deriving requirements for architecture.

26.7 IMPLEMENTATION

26.7.1 Factors to Consider Before Implementing the Big Data

The big data has become a game changer in shaping the future of education. However, certain factors need to be considered before applying the big data project.

26.7.1.1 Understanding Industry Point-of-View on Big Data

The education institutions needs first to assess the capabilities and trends to sidestep mistakes made in other sectors, particularly in the business world. For instance, it is advisable that before an institution or organization implement the big data to first study reports from well-regarded publications such as IDC and Forrester's.

26.7.1.2 Developing a Proof of Concept (PoC)

It is helpful first to create a business case that showcases all the values and the costs associated with the implementations of the big data.

26.7.1.3 Understanding the Current Tools and Technology

After the institution approves of continuing with the implementation of the big data, the next step entails analyzing the available tools and technology that can help meet architectural guidelines requirements.

26.7.1.4 Developing the Measures of Successful Big Data

It is important to have a target that will measure the benefits of the implementation of the big data system.

26.7.2 Steps of Successful implementation of the Big Data

26.7.2.1 Identification of the Goals

Both in business and education world requires one first to determine the purpose of the big data. For instance, in an education set-up, the institutional management needs to clarify whether the big data is to increase the efficiency of the learning process, improve institution's marketing, or to improve the overall performance of the institution. Therefore, based on the purpose, then it is easy to choose a methodology, hire employees, and select the right sources of data.

26.7.2.2 Leverage a Proven Big Data Strategy

There exist four prove strategies, which are based on the objective of the big data as well as the availability of the data.

26.7.2.2.1 Performance Management

This approach involves applying the transactional data such as the client purchase history and their turnover. However, this approach is not suitable for an education set up.

26.7.2.2.2 Data Exploration

This strategy primarily relies on the use of data mining and research to obtain solutions to the various problems an organization might be facing. In the process, the analytic teams can identify new segments of data.

26.7.2.2.3 Social Analytics

Social analytics approach employs the non-transaction data on social media platforms such as Facebook, Twitter, and Instagram. It is an applicable method in an education setting since most of the students use such platforms as their modes of communication.

26.7.2.2.4 Decision Science

This approach relies on the analysis of non-transactional data such as online discussion forums. Therefore, it utilizes text and sentiment analysis to determine

students/customer's opinions in regards to new services and schemes.

26.7.2.3 Identify infrastructural changes

To meet the requirements of the big data, the institution may need to change some of its infrastructures. For instance, the traditional storage mediums might not facilitate the running of complex algorithms and analysis. It may also involve the reconfiguration of the hardware platform to meet the new computation needs.

26.7.2.4 Training of the Staffs

It is crucial that all the staffs working within the institutions are conversant with the big data. Therefore, at the start, it is important to hire outside consulting team to guide on how best to utilize the big data. Moreover, it is essential to hire new strong system programmers who can work in a parallel processing environment and a network communications specialist.

26.8 BENCHMARKING

Big data benchmarks are used to evaluate and compare the performance of big data systems and architecture. A successful benchmarking entails five steps: planning, generating data, generate tests, execution, and analysis and evaluation.

The process provides a realistic and accurate measure of the big data systems and therefore, it can enable one to make efficient decision-making. For instance, a successful big data support education institutions to make decisions for planning system features, tuning system configuration and validate the deployment strategies. The following are the various steps of benchmarking:

26.8.1 Planning

One needs to make a plan on how the big data benchmarking will take place. Therefore, it is important to first come up with the goals of the benchmarking process. For instance, it should be able to generate application-specific workloads and tests such that it's easier to draw meaningful conclusions. Also,

in the planning tests, it is imperative to lay down the methodology to use in the benchmarking process.

26.8.2 Generating Data

There are four properties associated with big data: volume, velocity, variety, and veracity. Therefore, benchmarking should aim to cover those areas, and thus, it is crucial to apply real-world data or generates synthetic data for application-specific workloads. However, in most cases, most real-owners are not willing to share their real-data because of the security issues. Consequently, in big data benchmarks, the consensus is generated synthetic data as inputs of workloads by real data sets.

26.8.3 Generating Tests

In big-data benchmarks, one should first identify the typical workload behaviors for an application domain. Besides, it is important to take into considerations the diversity of workloads to cover different types of application domains, and at the same time to generate tests based on the workloads automatically. However, due to complexities involved in big data, it is challenging to develop big data benchmarks that reflect the real workload cases.

26.8.4 Execution

The execution stage needs to address various challenges. For instance, the big-data benchmarks need to adapt to different data formats. For example, the text data can be stored in the form of text files, web pages or pdf. The data also needs to be portable to represent different software stacks to provide a fully functional solution. Besides, the execution must reflect the user's experiences in using benchmarks.

26.9 CONCLUSION

From the discussion, it is a crucial time for the education institution to implement big data. Massive quantities of educational data can now be stored, analyzed and shared. For instance, various big-data algorithms can help track individual students in school and as a result keep detailed information about their

academic performance, behavior, and educational needs. Moreover, the big-data systems can help improve educational services by assisting the teachers to analyze what students know and what techniques are most useful for each pupil. Furthermore, technologies such as data mining and data analytics can provide fast feedback to students and teachers about their academic performance.

However, irrespective of the benefits associated with the big data, several challenges need to be addressed. For instance, the massive amounts of data generated by the big-data systems require much larger storage systems. To overcome such a problem, the data specialists need databases that do not use the traditional SQL based queries. The other challenge is in regards to the analysis as data is generated in different structure and size, and the study of the data may consume a lot of time and resources. Moreover, it is challenging reporting the analyzed information since when a large amount of data are involved, the traditional reports become difficult to interpret by human beings. Therefore, it requires useful tools and techniques that can address such challenges. An example includes the analytics software which can provide an in-depth analysis of some education patterns and extract valuable knowledge from them.

27 ANALYZING BIG DATA SORTING ALGORITHMS

Mark Miller

mgm3@indiana.edu

Indiana University

hid: fa18-523-63

github: [blue icon](#)

code: [blue icon](#)

Keywords: Sorting Algorithms, Python, Big Data Sorting, Computational Efficiency

27.1 ABSTRACT

There are many different sorting algorithms that vary vastly in terms of efficiency, memory usage (depending on the programming language), and time to completion. It is simply not effective to identify a specific data point and find it's place in comparison to the rest of the data. There are many methods in place to help computational efficiency when needing to get data sorted quickly and in a light-weight fashion. The purpose of this project is to provide code for many commonly used sorting algorithms while analyzing their efficiency. This depends highly on the way that the data stored, the initialization of the data, and the computational power inherent to the machine which is performing the operations. We will restrict to identical machine specifics and reset memory usage after each algorithm to ensure balanced and stable results. Not all popular sorting algorithms are used in this project, but many are. A simple timing mechanism, inherent to the programming language in use will be used to time each operation that is performed.

27.2 INTRODUCTION

There are many mathematical principles that contribute to efficient computing. There is a common problem, as defined by **Pointless Programming** [307], which presents the following scenario:

“You have two light bulbs and 100 story building. You must determine the minimal floor such that if you drop the light bulb from that floor it breaks. Once you break a bulb, it can’t be reused. Question: What’s the smallest number of drops required in the WORST case to determine the minimal floor” [307].

Assuming that no damage is done to the light bulb after preliminary drops, one method to this approach is to start at the bottom floor and increment the floor until the light bulb breaks. This way, the answer is obtained with just one lightbulb but it takes a long time to climb the stairs and drop the bulb from each individual floor, especially if the lightbulb can resist 99 floors of drops.

Another method is to begin from the fiftieth floor. This way, half of the floors will be removed at once.

The mathematical answer benefits from the “square root law” [308]. Finding the square root of the length of the data, (ten, in this case), provides the segmentation required to specify that it can take no more than 19 drops in order to determine the number of floors which can be endured for the light bulb.

Similar principles can be used in determining optimal methods for sorting big data algorithms. While not as simple as the problem described above, there are many algorithms that vary in efficiency that will be analyzed in this project. These algorithms include the following: Bubble sort [309], Merge sort [310], Insertion sort [311], Shell sort[309], Selection sort [311], Strand sort [312], Python’s sorting algorithm, [313], and Heap sort [314].

Many companies that utilize large datasets, do this through various Structured Query Language (SQL) databases [315]. These can contain far more rows and columns then other, more user-friendly software, like Microsoft Excel, can, by millions of rows and columns [316] [317]. This project is related to big data because in most algorithms, time complexity and operational complexity is more than most computer resources are able to handle for the tasks that are given to them. Big data often needed to be sorted, which causes problems for many database administrators, programmers, data scientists, and more.

27.3 REQUIREMENTS

Due to the random number generation of a ten-million numbers, it is recommended that these steps be completed on a compute system with at least 16 GB of RAM. Python 3.7 or higher.

Here is a summary of the hardware and software requirements of this project:

- * 16 GB of RAM (Due to the generation of 10 million random numbers)
- * Intel(R) Core(TM) i5-4690K CPU @ 3.50GHz (This level of CPU is sufficient)
- * Python 3.7 [318] or higher (Performed using the Spyder IDE [319])
- * Package Inclusion:

 - * Python Random [320]
 - * Python Time [321]
 - * Python Standard Library [322]
 - * TBD

27.4 DESIGN

Contained in this section is a description of each of the sorting algorithms that are used along with `pseudocode` of the algorithms.

27.4.1 Bubble Sort

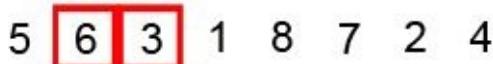


Figure 233: Bubble Sort[323]

The Bubble sort algorithm is known to be one of the slower algorithms [309]. The basic process takes two adjacent data points and if they are in the wrong order it will simply change the order of the two pairs. This is computationally expensive and requires considerable time to compare each individual element to all other elements in the list. At worst case scenario, the amount of operations required are at the order of the number of elements in the list squared [324].

Pseudo-code [323]:

```
procedure bubbleSort( A : list of sortable items )
    n = length(A)
    repeat
        swapped = false
        for i = 1 to n-1 inclusive do
            /* if this pair is out of order */
            if A[i-1] > A[i] then
```

```

        /* swap them and remember something changed */
        swap( A[i-1], A[i] )
        swapped = true
    end if
end for
until not swapped
end procedure

```

27.4.2 Merge Sort

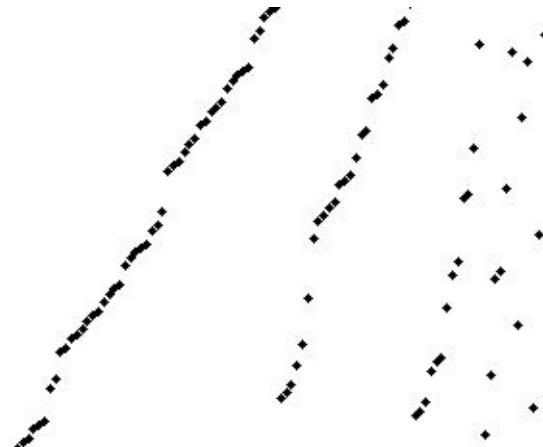


Figure 234: Merge Sort[325]

The Merge sort algorithm is a vast improvement on the bubble sort algorithm. It was originally developed by the popular mathematician and computer scientist John von Neumann [326]. This method segments the data intelligently and sorts in the segmented groups, reducing the computational requirements dramatically [310].

Pseudo-code [325]:

```

function merge_sort(list m)
    // Base case. A list of zero or one elements is sorted, by definition.
    if length of m ≤ 1 then
        return m

    // Recursive case. First, divide the list into equal-sized sublists
    // consisting of the first half and second half of the list.
    // This assumes lists start at index 0.
    var left := empty list
    var right := empty list
    for each x with index i in m do
        if i < (length of m)/2 then
            add x to left
        else
            add x to right

    // Recursively sort both sublists.
    left := merge_sort(left)
    right := merge_sort(right)

```

```

// Then merge the now-sorted sublists.
return merge(left, right)
In this example, the merge function merges the left and right sublists.

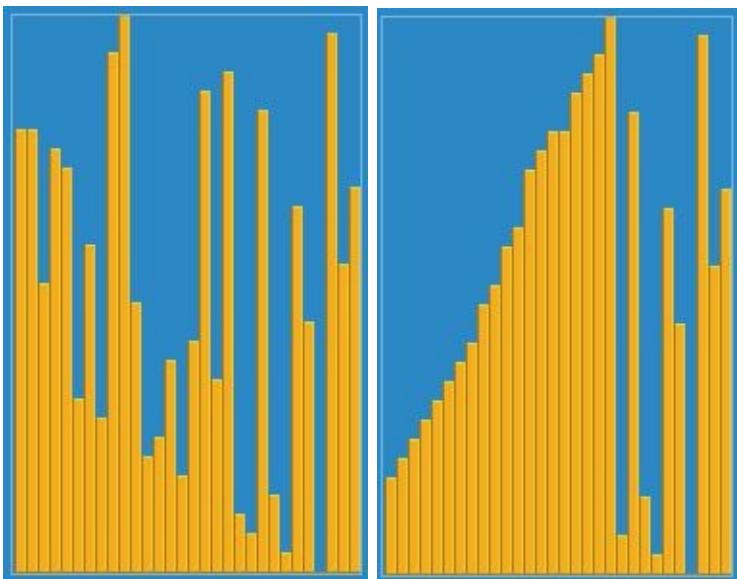
function merge(left, right)
    var result := empty list

    while left is not empty and right is not empty do
        if first(left) ≤ first(right) then
            append first(left) to result
            left := rest(left)
        else
            append first(right) to result
            right := rest(right)

    // Either left or right may have elements left; consume them.
    // (Only one of the following loops will actually be entered.)
    while left is not empty do
        append first(left) to result
        left := rest(left)
    while right is not empty do
        append first(right) to result
        right := rest(right)
    return result

```

27.4.3 Insertion Sort

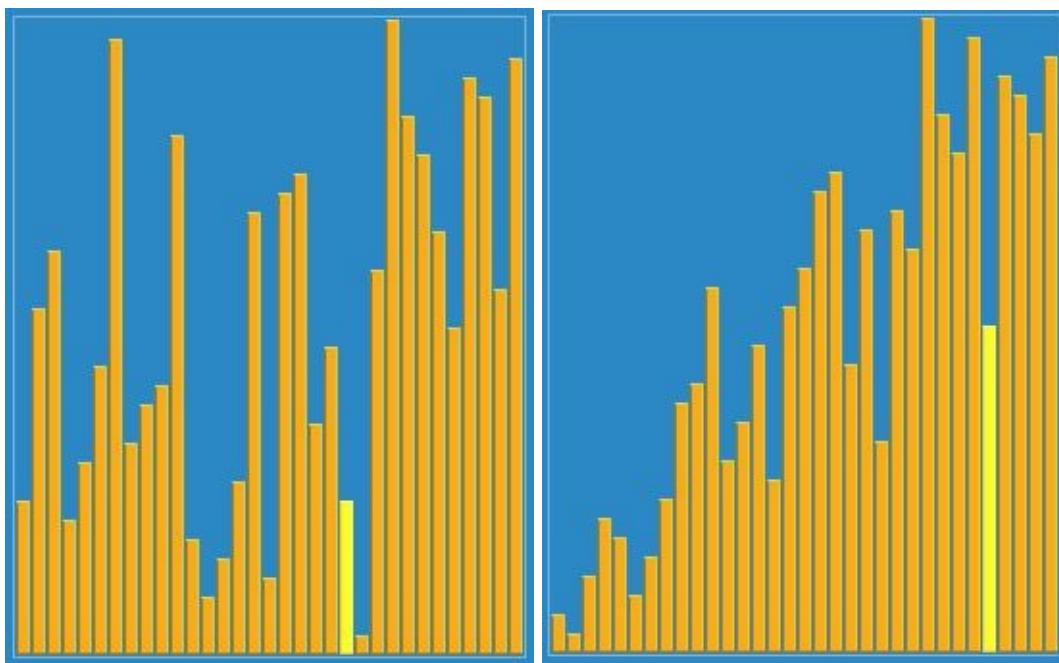


The insertion sort, while far more efficient than the Bubble sort, is inefficient for large lists, because it builds the final sorted list one element at a time [311]. The Insertion sort remains popular for its simple implementation, efficiency for small data sets, its light-weight use of memory, and its stability [328].

Pseudo-code ???:

```
i ← 1
while i < length(A)
    j ← i
    while j > 0 and A[j-1] > A[j]
        swap A[j] and A[j-1]
        j ← j - 1
    end while
    i ← i + 1
end while
```

27.4.4 Shell Sort



The Shell sort is essentially a combination of the Bubble sort and the Insertion sort, in that it uses both exchanges and insertions [328]. It uses these two different methods to provide lighter operations and order of operations. It begins with large values and proceeds to smaller values in order to result in a fully sorted list, after the proper exchanges and computations [309].

Pseudo-code [329]:

```
# Sort an array a[0...n-1].
gaps = [701, 301, 132, 57, 23, 10, 4, 1]

# Start with the largest gap and work down to a gap of 1
foreach (gap in gaps)
{
    # Do a gapped insertion sort for this gap size.
```

```

# The first gap elements a[0..gap-1] are already in gapped order
# keep adding one more element until the entire array is gap sorted
for (i = gap; i < n; i += 1)
{
    # add a[i] to the elements that have been gap sorted
    # save a[i] in temp and make a hole at position i
    temp = a[i]
    # shift earlier gap-sorted elements up until the correct location for a[
    i] is found
    for (j = i; j >= gap and a[j - gap] > temp; j -= gap)
    {
        a[j] = a[j - gap]
    }
    # put temp (the original a[i]) in its correct location
    a[j] = temp
}

```

27.4.5 Selection Sort

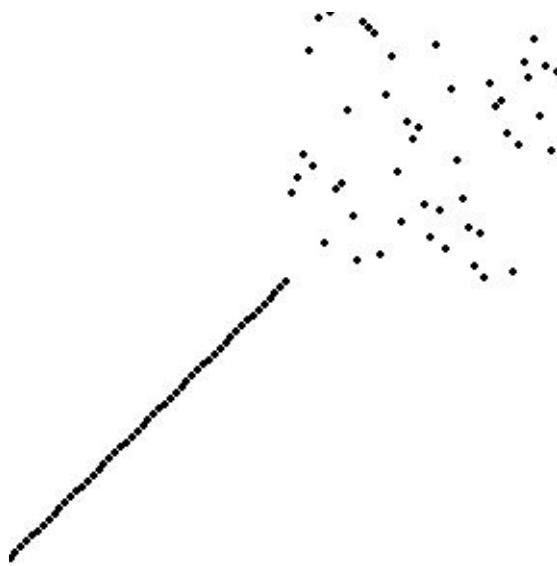


Figure 235: Selection Sort[330]

The Selection sort is not efficient for large data sets because of its time complexity and requirements on computational resources [331]. In this algorithm, two different lists are generated, a list of points that are already sorted and a list of points that are yet to be sorted. The algorithm will search through the entire list of unsorted points and find the smallest in that list and append it to the list that is sorted, as it will be larger than any other element in that list [311].

Pseudocode [330]:

```

bingo(array A)
{ This procedure sorts in ascending order. }

```

```

begin
    max := length(A)-1;

    { The first iteration is written to look very similar to the subsequent
      ones, but
      without swaps. }
    nextValue := A[max];
    for i := max - 1 downto 0 do
        if A[i] > nextValue then
            nextValue := A[i];
    while (max > 0) and (A[max] = nextValue) do
        max := max - 1;

    while max > 0 do begin
        value := nextValue;
        nextValue := A[max];
        for i := max - 1 downto 0 do
            if A[i] = value then begin
                swap(A[i], A[max]);
                max := max - 1;
            end else if A[i] > nextValue then
                nextValue := A[i];
        while (max > 0) and (A[max] = nextValue) do
            max := max - 1;
    end;
end;

```

27.4.6 Strand Sort

The Strand sort algorithm is a method that uses logic to determine if sublists have elements greater than other lists and slowly appends greater elements to other lists as the algorithm iterates through datasets [332].

Pseudocode [333]:

```

procedure strandSort( A : list of sortable items ) defined as:
    while length( A ) > 0
        clear sublist
        sublist[ 0 ] := A[ 0 ]
        remove A[ 0 ]
        for each i in 0 to length( A ) do:
            if A[ i ] > sublist[ last ] then
                append A[ i ] to sublist
                remove A[ i ]
            end if
        end for
        merge sublist into results
    end while
    return results
end procedure

```

27.4.7 Python Sort (Timsort)

The Timsort algorithm is the way that Python sorts lists. It is an extremely

efficient algorithm, utilizing the benefits of many other algorithms. A *min* function is utilized heavily to expedite the processes [334]. It is difficult to identify weaknesses in this algorithm but if there are any it is that it can struggle with memory usage, which tends to be less than other algorithms still, and stability. It utilizes adaptive sorting to find optimal sizes for each step of the process. size sorting,

Pseudocode is intentionally not provided here.

27.4.8 Heap Sort

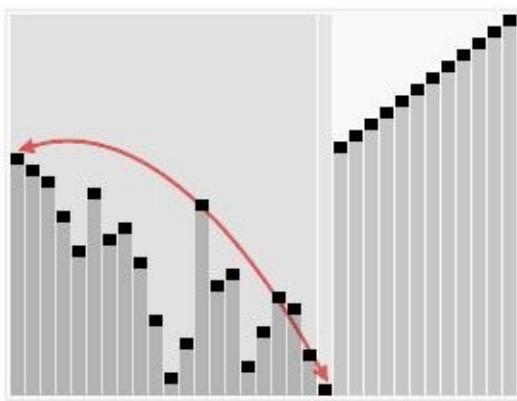


Figure 236: Heap Sort[335]

Heap sort is a method that compares some values to other values by dividing the list of numbers into an unsorted and sorted list (like the Selection sort) with the goal of shrinking the unsorted list based on comparison of a value to the list that is already sorted. It can have high time-complexity, and struggles with larger data sets, do to the number of comparisons that are required at each step of the algorithm.

Pseudocode [www-fa18-523-63-heap-wiki]:

27.4.8.1 Main Routine:

```
procedure heapsort(a, count) is
    input: an unordered array a of length count
    (Build the heap in array a so that largest value is at the root)
    heapify(a, count)
    (The following loop maintains the invariants that a[0:end] is a heap and
     every element
```

```

beyond end is greater than everything before it (so a[end:count] is in
sorted order))
end ← count - 1
while end > 0 do
    (a[0] is the root and largest value. The swap moves it in front of the
sorted elements.)
    swap(a[end], a[0])
    (the heap size is reduced by one)
    end ← end - 1
    (the swap ruined the heap property, so restore it)
    siftDown(a, 0, end)

```

27.4.8.2 Sub-routines:

```

(Put elements of 'a' in heap order, in-place)
procedure heapify(a, count) is
    (start is assigned the index in 'a' of the last parent node)
    (the last element in a 0-based array is at index count-1; find the parent
    of that element)
    start ← iParent(count-1)

    while start ≥ 0 do
        (sift down the node at index 'start' to the proper place such that all
        nodes below
        the start index are in heap order)
        siftDown(a, start, count - 1)
        (go to the next parent node)
        start ← start - 1
    (after sifting down the root all nodes/elements are in heap order)

(Repair the heap whose root element is at index 'start', assuming the heaps
rooted at its children are valid)
procedure siftDown(a, start, end) is
    root ← start

    while iLeftChild(root) ≤ end do      (While the root has at least one child)
        child ← iLeftChild(root)      (Left child of root)
        swap ← root                  (Keeps track of child to swap with)

        if a[swap] < a[child]
            swap ← child
        (If there is a right child and that child is greater)
        if child+1 ≤ end and a[swap] < a[child+1]
            swap ← child + 1
        if swap = root
            (The root holds the largest element. Since we assume the heaps
            rooted at the
            children are valid, this means that we are done.)
            return
        else
            swap(a[root], a[swap])
            root ← swap                (repeat to continue sifting down the child
                                           now)

```

27.5 ARCHITECTURE

The architecture for this project is a main.py file which is the master python file,

calling the other functions from other files, stored in the same directory. The other files will each be a Python 3.7 [318] implementation, generally performed in a rather procedural programming methodology. The following files are called from main.py in the working directory: + bubble.py + merge.py + insertion.py + shell.py + selection.py + squareroot.py + strand.py + pysort.py + heap.py

In the call of the each of the above files, in main.py there is a a timing mechanism, using Python's **Time** library [321].

27.6 DATASET

The only dataset required for this project is a list of 100 randomly generated numbers, using the Python random library [320]. Using this library, ten-million Gaussian Random Numbers [336] will be generated, using the seed of “one”. This data will only be stored for the duration of the program’s run-time and will not be committed to disk, unless memory is usurped , in which case, temporary disk usage will occur.

27.7 IMPLEMENTATION

The implementation is performed using the Spyder terminal and running the Python 3.7 script via Spyder’s command line interface (CLI). Provided that all of the files are stored in a single directory, the files as listed in the code folder for this project should run flawlessly, with the correct Python version and with all of the hardware and software requirements met. I purposely kept the design very simple, for ease of implementation, and ease of programming.

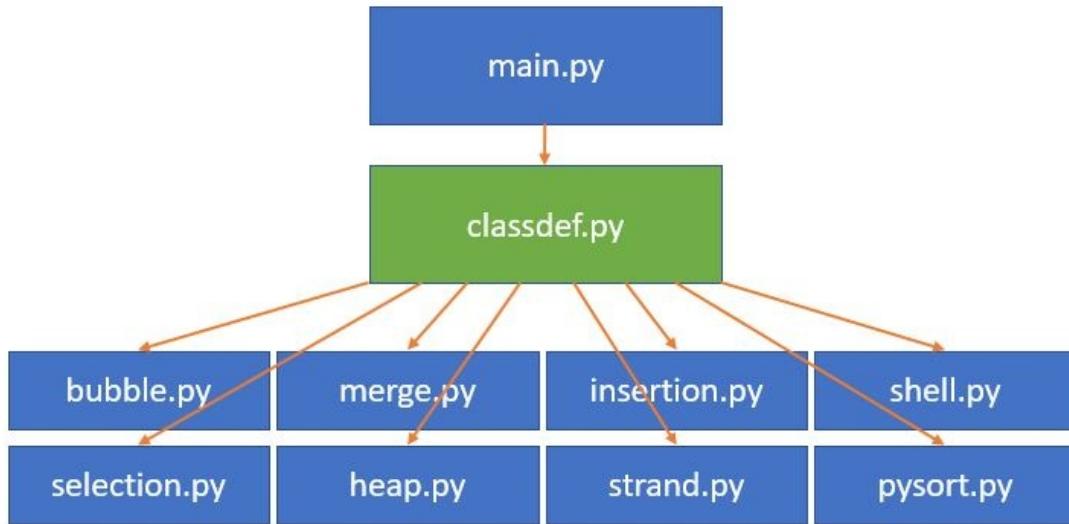


Figure 237: Design Doc

27.8 LIMITATIONS

Admittedly, this project's computing power is significantly less than cloud environments use. The purpose of this project is more about showing the efficiencies gained than it is about the cloud computing advantage. In practice, cloud topologies use sophisticated tuning of many algorithms, depending on the resources available in the computing stack [337]. While this project does not show how the algorithms are used together for ultimate efficiency, it does show how certain efficiencies are gained from very large data sets by implementing more efficient algorithms.

27.9 BENCHMARK

Many different sizes of lists of random numbers were tested. In some cases, smaller lists benefitted from specific algorithms, while others were nearly inoperational for larger data sets. The times for different sizes of lists of random numbers are listed here:

Here is the list of the results, along with some visualizations to help interpret the information that is being presented

```

Python 100 time: 1.114347469410859e-05
Bubble 100 time: 0.0005601062985078897
Select 100 time: 0.0003020468502654694

```

```

Merge 100 time: 0.00021025979731348343
Shell 100 time: 3.313718116260134e-05
Heap 100 time: 0.000275947659247322
Strand 100 time: 3.841566649498418e-05

Python 1,000 time: 0.0001316689667874016
Bubble 1,000 time: 0.07056049045422697
Select 1,000 time: 0.029865982105548028
Merge 1,000 time: 0.0027034659433411434
Shell 1,000 time: 0.0002888506314775441
Heap 1,000 time: 0.0045884728933742736
Strand 1,000 time: 0.00035717773062060587

Python 10,000 time: 0.001863013250840595
Bubble 10,000 time: 7.700457498340256
Select 10,000 time: 3.177057224973396
Merge 10,000 time: 0.036345033688121475
Shell 10,000 time: 0.003031025487871375
Heap 10,000 time: 0.061194692068966106
Strand 10,000 time: 0.01295399768059724

Python 100,000 time: 0.026999469511792995
Bubble 100,000 time: 925.2644678242395
Select 100,000 time: 528.98660055565309
Merge 100,000 time: 0.5586767063505249
Shell 100,000 time: 0.0392740083989338
Heap 100,000 time: 0.8638226169750851
Strand 100,000 time: 1.5463215238887642

```

From this, it is simply determined that for all amounts of randomly generated numerical data, the Selection and Bubble methods are extremely inefficient. These two methods are the bottom scorers for each of the datasets, which means that they do not out perform the other algorithms for either small or large data, at least with these implementations, in this environment.

All of these tests were performed on a single core, to ensure accurate timing for each of the individual algorithms. The memory buffers were never filling, and the core of the CPU which was processing this information was pegged at 100-percent.

To make the visualizations more aesthetic I have excluded the two slowest algorithms (Bubble and Select).

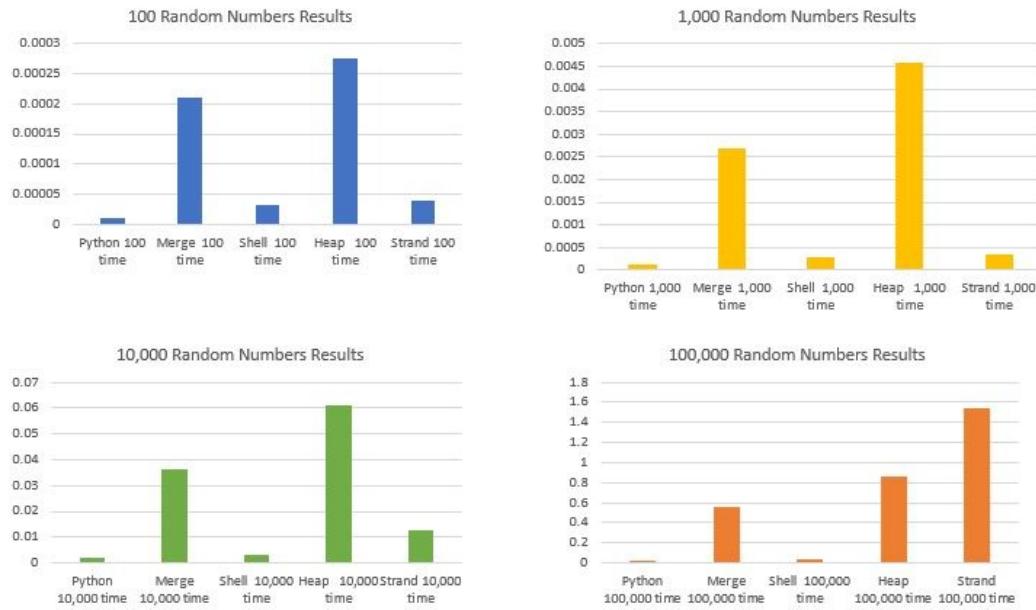
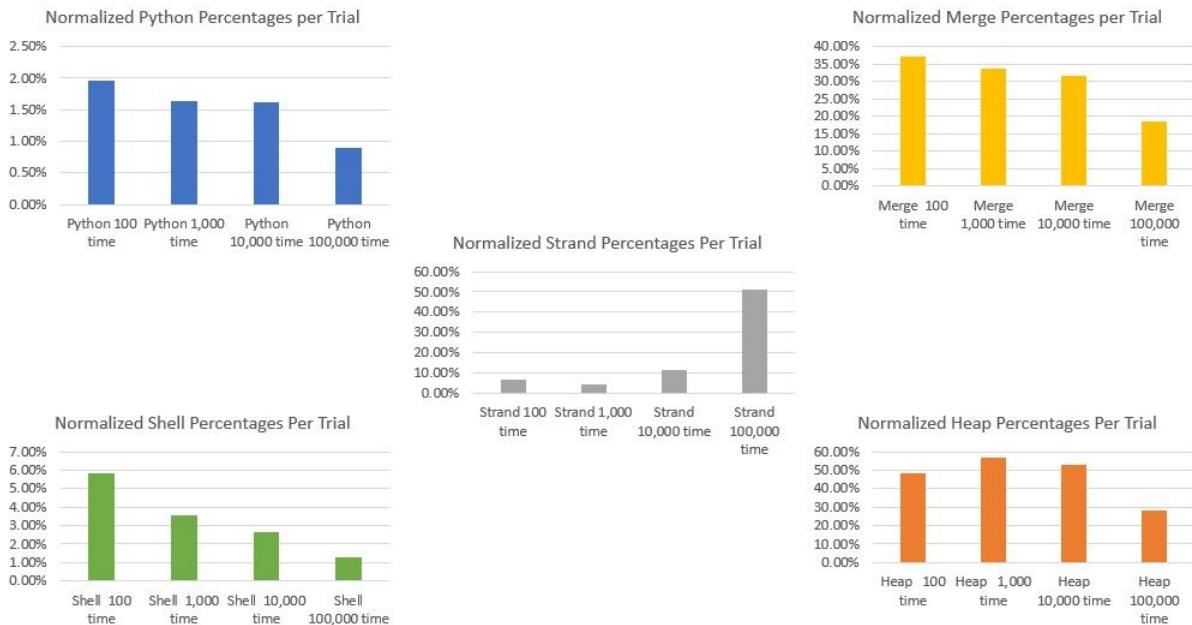


Figure 238: Chart By Size

As seen in the figure above, Among the fize fastest algorithms that were tested, Python's Timsort algorithm out-performs each other algorithm. The Merge sort and Heap sort algorithms are outperformed until that data gets larger, during which the Strand sort algorithm loses its efficiency that it had on the smaller data sets.



{#fig:chartbymethod}

In this set of graphs, a downward trend of the data indicates that the algorithm outperforms the other algorithms as the datasets get larger. As the data sets get larger, Python's Timsort algorithm, and the Shell sort algorithm decrease its time taken, relative to the other algorithms. The Strand sort algorithm appears to perform well for intermediate data sets but as the data gets larger, on either end, it loses its efficiency, at least as it is implemented in this project.

The altogether run-time for this program, on my computer is approximately twenty minutes.

27.10 CONCLUSION

As data gets larger, certain sorting algorithms, such as the Strand sort algorithm, become nearly unfunctional, due to time and operational complexity of the algorithm. In this case, the old saying of “the simpler, the better” does not seem to apply. Python’s inherent sorting algorithm, **Timsort**, outperforms all of the other algorithms that were tested. The Shell sort algorithm is a close second. The Bubble and Select algorithms are very computationally inefficient and should not be implemented unless there are very specific circumstances, which are not addressed in this project. The Select and Bubble sort algorithms tend to be what humans use, but they are not computationally sufficient for computer resources to use to sort big data.

While this project was hampered by the use of a single CPU core the impacts that ineffective sorting algorithms can drastically be felt on cloud systems. One metric that could be used to determine the effectiveness of an algorithm could be to see how that algorithm performs as the datasets get larger. This shows that it can handle the growing needs of big data better than other algorithms, like Python’s Timsort and the Shell sorting algorithms showed.

27.11 ACKNOWLEDGEMENTS

Plenty of research, all which is cited here in references.bib, was used in this analysis of different sorting algorithms. Credit goes to each author of all articles and web pages listed for their brilliance and efforts in furthering the world of data science, computer science, and software development.

###Code Acknowledgements: + [332] + [338] + [339] + [340] + [341] + [342]

- [1] M. Fowler, “Micro Services.” Mar-2014 [Online]. Available: <https://martinfowler.com/articles/microservices.html>
- [2] SnartBear, “Using an API Gateway in Your Microservices Architecture.” 2018 [Online]. Available: <https://smartbear.com/learn/api-design/api-gateways-in-microservices/>
- [3] SnartBear, “Monitoring with Prometheus and Grafana.” Jan-2016 [Online]. Available: <https://github.com/hashicorp/faas-nomad/wiki/Monitoring-with-Prometheus-and-Grafana>
- [4] A. Innovations, “What is Serverless?” [Online]. Available: <https://serverless-stack.com/chapters/what-is-serverless.html>
- [5] OpenFaaS, “OpenFaaS: Introduction.” [Online]. Available: <https://docs.openfaas.com/>
- [6] OpenFaaS, “Become your own Functions as a Service provider using OpenFaaS.”.
- [7] Kaggle, “Dogs vs. Cats.” [Online]. Available: <https://www.kaggle.com/c/dogs-vs-cats>
- [8] F. Chollet, “Building powerful image classification models using very little data.”.
- [9] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016 [Online]. Available: <http://www.deeplearningbook.org>
- [10] Christopher Olah, “Conv Nets: A Modular Perspective.” Jul-2014 [Online]. Available: <http://colah.github.io/posts/2014-07-Conv-Nets-Modular/>
- [11] M. Chang, “Applied Deep Learning 11/03 Convolutional Neural Networks.” Oct-2016 [Online]. Available: <https://www.slideshare.net/ckmarkohchang/applied-deep-learning-1103-convolutional-neural-networks>

- [12] G. von Laszewski, “Raspberry PI 5 Node Cluster Case.” [Online]. Available: <https://github.com/cloudmesh-community/case>
- [13] Gerald Manipon, “HySDS Community Wiki.” Web page [Online]. Available: <https://github.com/hysds/hysds-framework/wiki>
- [14] ARIA Team, “Advanced Rapid Imaging and Analysis.” Web page [Online]. Available: <https://aria.jpl.nasa.gov>
- [15] J. Blumenfeld, “Getting Ready for NISAR—and for Managing Big Data using the Commercial Cloud.” Web page [Online]. Available: <https://earthdata.nasa.gov/getting-ready-for-nisar>
- [16] E. Fetzer, “A Multi-Sensor Water Vapor Climate Data Record Using Cloud Classification.” Web page [Online]. Available: <https://earthdata.nasa.gov/community/community-data-system-programs/measures-projects/multi-sensor-water-vapor>
- [17] Margaret Srinivasan, “NASA SWOT - Mission.” Web page [Online]. Available: <https://swot.jpl.nasa.gov/mission.htm>
- [18] Tony Greicius, “NASA-ISRO Synthetic Aperture Radar.” Web page [Online]. Available: <https://www.jpl.nasa.gov/missions/nasa-isro-synthetic-aperture-radar-nisar/>
- [19] Karen Yuen, “Orbiting Carbon Observatory-2.” Web page [Online]. Available: <https://ocov2.jpl.nasa.gov/>
- [20] European Space Agency, “Sentinel-1 Data Products.” Web page [Online]. Available: <https://sentinel.esa.int/web/sentinel/missions/sentinel-1/data-products>
- [21] US Geological Survey, “Digital Elevation Models.” Web page [Online]. Available: <https://lta.cr.usgs.gov/DEMs>
- [22] IBM, “Hadoop.” website, 2018 [Online]. Available: <https://www.ibm.com/analytics/hadoop>
- [23] V. Naresh Kumar and P. Shindgikar, “Configuring hadoop high availability clusters.” website, 2018 [Online]. Available: <https://www.i->

programmer.info/projects/31-systems/11781-configuring-hadoop-high-availability-clusters.html

[24] C. Wodehouse, “A guide to hadoop.” website, 2018 [Online]. Available: <https://www.upwork.com/hiring/data/a-guide-to-hadoop/>

[25] Hortonworks, “Apache hive.” website, 2018 [Online]. Available: <https://hortonworks.com/apache/hive/>

[26] C. Administrator, “Tutorial - apache hive - apache software foundation.” website, 2018 [Online]. Available: <https://cwiki.apache.org/confluence/display/Hive/Tutorial>

[27] T. A. S. Foundation, “Apache derby.” website, 2018 [Online]. Available: <https://db.apache.org/derby/>

[28] multiple contributors, “Choosing a big data storage technology in azure.” website, 2018 [Online]. Available: <https://docs.microsoft.com/en-us/azure/architecture/data-guide/technology-choices/data-storage>

[29] Python community, “Pyhive project description.” website, 2018 [Online]. Available: <https://pypi.org/project/PyHive/>

[30] “Cloudmesh nist services.” [Online]. Available: <https://github.com/cloudmesh-community/nist/tree/master/services>

[31] “Cloudmesh cm project.” [Online]. Available: <https://github.com/cloudmesh-community/cm/tree/master/cm4>

[32] GitHub, “Electron | build cross platform desktop apps with javascript, html, and css.” website, 2018 [Online]. Available: <https://electronjs.org/>

[33] JeremyAshkenas, “Web framework.” website, 2018 [Online]. Available: <http://backbonejs.org/>

[34] YehudaKatz, “Templating library.” website, 2017 [Online]. Available: <https://handlebarsjs.com/>

[35] JSFoundation, “JavaScript library.” website, 2018 [Online]. Available:

<https://jquery.com/>

[36] Google, “Material components.” website, 2018 [Online]. Available: <https://material.io/>

[37] JSFoundation, “Module bundle.” website, 2018 [Online]. Available: <https://webpack.js.org/>

[38] SyrusAkbary, “GraphQL support for flask.” website, 2015 [Online]. Available: <https://github.com/graphql-python/flask-graphql>

[39] SyrusAkbary, “Graphene mongoengine integration.” website, 2015 [Online]. Available: <https://github.com/graphql-python/graphene-mongo>

[40] DanieleFaraglia, “Fake data.” website, 2012 [Online]. Available: <https://github.com/joke2k/faker>

[41] Open API Initiative, “OpenAPI.” website, 2018 [Online]. Available: <https://www.openapis.org/about>

[42] Amazon Web Services, “AWSEC2.” website, 2018 [Online]. Available: <https://aws.amazon.com/ec2/>

[43] JupyterHub, “JupyterHub.” website, 2018 [Online]. Available: <http://jupyter.org/hub>

[44] AWS, “AWS.” website, 2018 [Online]. Available: <https://console.aws.amazon.com/iam/home?region=us-east-2#/groups>

[45] AWS, “AWS.” website, 2018 [Online]. Available: <https://us-east-2.console.aws.amazon.com/ec2/v2/home?region=us-east-2#SecurityGroups:sort=groupId>

[46] AWS, “AWS.” website, 2018 [Online]. Available: <https://us-east-2.console.aws.amazon.com/ec2/v2/home?region=us-east-2#Instances:sort=instanceId>

[47] E. Schadt, “The role of big data in medicine.” Web page, 2015 [Online]. Available: <https://www.mckinsey.com/industries/pharmaceuticals-and-medical->

[products/our-insights/the-role-of-big-data-in-medicine](#)

[48] Y. Lee CH., “Medical big data: Promise and challenges.” Web page, 2017 [Online]. Available: <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5331970/>

[49] Y. Bhattacharya A., “Precision diagnosis of melanoma and other skin lesions from digital images.” Web page, 2017 [Online]. Available: <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5543387/>

[50] Anaconda, “What is anaconda?” Web page, 2018 [Online]. Available: <https://www.anaconda.com/what-is-anaconda/>

[51] Jupyter, “Jupyter.” Web page, 2018 [Online]. Available: <http://jupyter.org/>

[52] Jupyter, “The jupyter notebook.” Web page, 2018 [Online]. Available: <http://jupyter.org/>

[53] SciPy, “What is numpy?” Web page, 2018 [Online]. Available: <https://docs.scipy.org/doc/numpy-1.13.0/user/whatisnumpy.html>

[54] pandas, “Pandas:Powerfule python data analysis toolkit.” Web page, 2018 [Online]. Available: <https://pandas.pydata.org/pandas-docs/stable/>

[55] matplotlib, “Home.” Web page, 2018 [Online]. Available: <https://matplotlib.org/>

[56] R. Tschandl P., “The ham10000 dataset, a large collection of multi-source dermatoscopic images of common pigmented skin lesions.” Web page, 2018 [Online]. Available: <https://dataverse.harvard.edu/dataset.xhtml?persistentId=doi:10.7910/DVN/DBW86T>

[57] KNIME, “KNIME integrations.” Web page, 2018 [Online]. Available: <https://www.knime.com/knime-software/knime-integrations>

[58] A. Vidhya, “Building your first machine learning model using knime (no coding required!).” Web page, 2017 [Online]. Available: <https://www.analyticsvidhya.com/blog/2017/08/knime-machine-learning/>

[59] KNIME, “JSON processing.” Web page, 2018 [Online]. Available:

<https://www.knime.com/whats-new-in-knime-211#JSON>

[60] U. Sewwandi, “Guided analytics using knime analytics platform.” Web page, 2018 [Online]. Available: <https://towardsdatascience.com/guided-analytics-using-knime-analytics-platform-b6543ebab7e2>

[61] KNIME, “KNIME on amazon web services.” Web page, 2018 [Online]. Available: <https://www.knime.com/knime-software/knime-aws>

[62] KNIME, “KNIME on microsoft azure.” Web page, 2018 [Online]. Available: <https://www.knime.com/knime-software/knime-azure>

[63] The Data Visualisation Catalogue, “Parallel coordinates plot.” Web page, 2018 [Online]. Available: https://datavizcatalogue.com/methods/parallel_coordinates.html

[64] Statistics Solutions, “What is logistic regression?” Web page, 2018 [Online]. Available: <https://www.statisticssolutions.com/what-is-logistic-regression/>

[65] American Cancer Society, “Skin cancer.” Web page, 2018 [Online]. Available: <https://www.cancer.org/cancer/skin-cancer.html>

[66] Team Research Nest, “Real world applications of big data in healthcare.” Web page, 2018 [Online]. Available: <https://medium.com/the-research-nest/real-world-applications-of-big-data-in-healthcare-5c84696fd3d4>

[67] Apache, “Kafka.” Web Page [Online]. Available: <https://kafka.apache.org/>

[68] T. P. Neha Narkhede Gwen Shapira, *Kafka: The definitive guide*, First. O’REILLY, 2017 [Online]. Available: <https://www.confluent.io/wp-content/uploads/confluent-kafka-definitive-guide-complete.pdf>

[69] Apache, “Apache kafka.” Web Page, 2018 [Online]. Available: <https://www.apache.org/dyn/closer.cgi?path=/kafka/2.1.0/kafka-2.1.0-src.tgz>

[70] Apache, “Kafka.” Web Page [Online]. Available: <https://issues.apache.org/jira/browse/KAFKA-6855>

- [71] The Apache Software Foundation, “Apache nifi.” Web page, Oct-2018 [Online]. Available: <https://nifi.apache.org/>
- [72] A. DOKAEVA, “How to make etl simple and intuitive with nifi.” Web page, Mar-2018 [Online]. Available: <https://issart.com/blog/how-to-make-etl-simple-and-intuitive-with-nifi/>
- [73] S. Maarek, “Introduction to apache nifi (hortonworks dataflow - hdf 2.0).” Presentation [Online]. Available: <https://www.udemy.com/apache-nifi/>
- [74] A. Bridgwater, “NSA ’nifi’ big data automation project out in the open.” Web Page, Jul-2015 [Online]. Available: <https://www.forbes.com/sites/adrianbridgwater/2015/07/21/nsa-nifi-big-data-automation-project-out-in-the-open/#68cdd7dc55d6>
- [75] Apache NiFi Team, “Apache nifi overview.” Web page, Oct-2018 [Online]. Available: <https://nifi.apache.org/docs.html>
- [76] hortonworks, “Analyze transit patterns with apache nifi.” Web page, Oct-2018 [Online]. Available: <https://hortonworks.com/tutorial/analyze-transit-patterns-with-apache-nifi/section/1/>
- [77] S. Gupta, “Creating custom processors and controllers in apache nifi.” Web page, May-2018 [Online]. Available: <https://medium.com/hashmapinc/creating-custom-processors-and-controllers-in-apache-nifi-e14148740ea>
- [78] Apache NiFi Team, “Apache nifi downloads.” Web page, Oct-2018 [Online]. Available: <http://nifi.apache.org/download.html>
- [79] Apache Software Foundation, “Apache kafka quickstart.” Web Page, Jan-2017 [Online]. Available: <https://kafka.apache.org/quickstart>
- [80] @humdata, “Asia pacific: Storm tracks 1956 to 2017.” Website [Online]. Available: <https://data.world/ocha-roap/b1068a22-ec52-459d-9541-0fb63906bb39>
- [81] stoltzman consulting llc, “EXPLORATORY data analysis of tropical storms in r.” Website [Online]. Available: <https://www.stoltzmanniac.com/exploratory-data-analysis-of-tropical-storms-in-r/>

- [82] Python, “What is python? Executive summary.” Website [Online]. Available: <https://www.python.org/doc/essays/blurb/>
- [83] Microsoft, “Your vision. Your cloud.” Website [Online]. Available: <https://azure.microsoft.com/en-us/>
- [84] Matplotlib, “Documentation.” Website [Online]. Available: <https://matplotlib.org/>
- [85] Seaborn, “Seaborn: Statistical data visualization.” Website [Online]. Available: <https://seaborn.pydata.org/>
- [86] Folium, “Folium 0.7.0+8.g8adfb77 documentation.” Website [Online]. Available: <https://python-visualization.github.io/folium/>
- [87] Scikit-Learn, “Scikit-learn.” Website [Online]. Available: <https://scikit-learn.org/stable/>
- [88] Wikipedia, “Logistic regression.” Website [Online]. Available: https://en.wikipedia.org/wiki/Logistic_regression
- [89] Wikipedia, Website [Online]. Available: https://en.wikipedia.org/wiki/Random_forest
- [90] Wikipedia, “Support vector machine.” Website [Online]. Available: https://en.wikipedia.org/wiki/Support_vector_machine
- [91] Scikit-Learn, Website [Online]. Available: <https://scikit-learn.org/stable/modules/sgd.html>
- [92] Wikipedia, “XGBoost.” Website [Online]. Available: <https://en.wikipedia.org/wiki/XGBoost>
- [93] T. Voelker and R. McGlashan, “What is crowdfunding? Bringing the power of kickstarter to your entrepreneurship research and teaching activities,” in *Small business institute journal*, 2013, vol. 9.0, pp. 11–22 [Online]. Available: <https://sbij.org/index.php/SBIJ/article/view/175/124>
- [94] R. Walters, “Getting started with python and mongodb.” Web Page, Apr-

2017 [Online]. Available: <https://www.mongodb.com/blog/post/getting-started-with-python-and-mongodb>

[95] R. R. Nishad Tupe Izolda Fetko, “Analyzing france accidents data using mongo db,” Indiana University of Bloomington, technical report, Jul. 2018 [Online]. Available:

<https://iu.instructure.com/courses/1718879/assignments/8129645/submissions/61>

[96] K. Loughead, “Tableau and mongodb analytics: Why it’s a bad marriage.” Web Page, 2018 [Online]. Available: <https://blog.knowi.com/2018/03/tableau-and-mongodb-analytics.html>

[97] T. Matei, “MongoDB performance in the cloud,” Master’s thesis, San Jose State University, San Hose, California, United States, 2013 [Online]. Available: https://scholarworks.sjsu.edu/cgi/viewcontent.cgi?referer=https://www.google.com/&httpsredir=1&article=1300&context=etd_proj

[98] K. Bigbee, J. Chaves, and D. Seaman, “Benchmarking big data cloud-based infrastructures,” Master’s thesis, Worcester Polytechnic Institute, Worcester, MA, United States, 2017 [Online]. Available: https://web.wpi.edu/Pubs/E-project/Available/E-project-031617-185427/unrestricted/MQP_Final.pdf

[99] S. Sverchkov, “Evaluating nosql performance: Which database is right for your data?” Web Page, Feb-2014 [Online]. Available: <https://jaxenter.com/evaluating-nosql-performance-which-database-is-right-for-your-data-107481.html>

[100] Wikipedia, “Application programming interface.” Web Page, 2018 [Online]. Available: https://en.wikipedia.org/wiki/Application_programming_interface

[101] Kaggle, “How to use kaggle: Kaggle api.” Web Page, 2018 [Online]. Available: <https://www.kaggle.com/docs/api>

[102] tikivisik, “Kaggle-api.” Web Page, Oct-2018 [Online]. Available: <https://github.com/Kaggle/kaggle-api>

[103] M. Mouille, “Kicksarter projects.” Web Page, Feb-2018 [Online]. Available: <https://www.kaggle.com/kemical/kickstarter-projects>

[104] D. Research, “Correlation analysis - market research.” Web Page, 2018 [Online]. Available: <https://www.djsresearch.co.uk/glossary/item/correlation-analysis-market-research>

[105] S. Solutions, “Time series analysis.” Web Page, 2018 [Online]. Available: <https://www.statisticssolutions.com/time-series-analysis/>

[106] S. Solutions, “What is logistic regression?” Web Page, 2018 [Online]. Available: <https://www.statisticssolutions.com/what-is-logistic-regression/>

[107] MongoDB, “PyMongo 3.7.2 documentation.” Web Page, 2008 [Online]. Available: <https://api.mongodb.com/python/current/>

[108] MongoDB, ***MongoDB manual 4.0.*** MongoDB Inc, 2018 [Online]. Available: <https://docs.mongodb.com/manual/replication/#edge-cases-2-primaries>

[109] G. James, D. Witten, T. Hastie, and R. Tibshirani, ***An introduction to statistical learning***, vol. 112. Springer, 2013.

[110] “What is twitter and how does it work?” Web Page.

[111] M. Rouse, “Natural language processing - nlp.” Web page [Online]. Available: <https://searchbusinessanalytics.techtarget.com/definition/natural-language-processing-NLP>

[112] “Introduction to tweet json.” Web page [Online]. Available: <https://developer.twitter.com/en/docs/tweets/data-dictionary/overview/intro-to-tweet-json.html>

[113] “Getting started with the twitter developer platform.” Web page [Online]. Available: <https://developer.twitter.com/en/docs/basics/getting-started>

[114] “Data science python.” Web page [Online]. Available: <https://www.coursera.org/specializations/data-science-python>

[115] A. Verma, “5 reasons why you should choose python for big data.” Web page, Jul-2018 [Online]. Available: <https://www.whizlabs.com/blog/python-and->

[big-data/](#)

[116] “Python ide code editors guide.” Web page [Online]. Available: <https://realpython.com/python-ides-code-editors-guide/>

[117] Wikipedia, “PIP package manager.” Web page [Online]. Available: [https://en.wikipedia.org/wiki/Pip_\(package_manager\)](https://en.wikipedia.org/wiki/Pip_(package_manager))

[118] “Authentication tutorial.” Web page [Online]. Available: http://docs.tweepy.org/en/v3.5.0/auth_tutorial.html#introduction

[119] M. Rouse, “OAuth.” Web page [Online]. Available: <https://searchmicroservices.techtarget.com/definition/OAuth>

[120] “Introducing json.” Web page [Online]. Available: <https://www.json.org/>

[121] M. Bonzanini, “Mastering social media mining with python,” PacktPub, 2016 [Online]. Available: https://www.packtpub.com/mapt/book/big_data_and_business_intelligence/9781' media---challenges-and-opportunities

[122] Wikipedia, “Cloud storage.” Web page [Online]. Available: https://en.wikipedia.org/wiki/Cloud_storage

[123] “Amazon aws s3 data lake.” Web page [Online]. Available: <https://aws.amazon.com/big-data/datalakes-and-analytics/>

[124] C. Newton, “How a typo took down s3, the backbone of the internet.” Web page, Mar-2017 [Online]. Available: <https://www.theverge.com/2017/3/2/14792442/amazon-s3-outage-cause-typo-internet-server>

[125] “MongoDB atlas.” Web page [Online]. Available: <https://www.mongodb.com/cloud/atlas>

[126] “Driver connection.” Web page [Online]. Available: <https://docs.atlas.mongodb.com/driver-connection/>

[127] E. Brown, “Collecting/storing tweets with python and mongodb.” Web

page, Jan-2017 [Online]. Available: <https://pythondata.com/collecting-storing-tweets-with-python-and-mongodb/>

[128] “Databases and collections.” Web page [Online]. Available: <https://docs.mongodb.com/manual/core/databases-and-collections/>

[129] “Twitter data analysis from mongodb - part 1 introduction.” Web page, Sep-2014 [Online]. Available: <https://blog.jlevente.com/twitter-data-analysis-from-mongodb-part-1-introduction/>

[130] “Twitter data analysis fro mongodb - part 2, exploring data.” Web page, Sep-2014 [Online]. Available: <https://blog.jlevente.com/twitter-data-analysis-from-mongodb-part-2-exploring-data/>

[131] “Big data explained.” Web page [Online]. Available: <https://www.mongodb.com/big-data-explained>

[132] S. Mei, “Why you should never use mongodb.” Web page, Nov-2013 [Online]. Available: <http://www.sarahmei.com/blog/2013/11/11/why-you-should-never-use-mongodb/>

[133] “Naive bayes explained.” Web page, Sep-2017 [Online]. Available: <https://www.analyticsvidhya.com/blog/2017/09/naive-bayes-explained/>

[134] C. Stergiou and D. Siganos, “What is a neural network?” Web page [Online]. Available: https://www.doc.ic.ac.uk/~nd/surprise_96/journal/vol4/cs11/report.html#What_is_a_Neural_Network

[135] “Neural network definition.” Web page [Online]. Available: <https://skymind.ai/wiki/neural-network#define>

[136] Q. Tul, A. Riaz, M. Ali, and A. U. Rehman, “Sentiment analysis using deep learning techniques: A review,” *International Journal of Advanced Computer Science and Applications*, vol. 8, no. 6, Jan. 2017 [Online]. Available:

https://www.researchgate.net/publication/318096420_Sentiment_Analysis_Using

[137] “Sklearn metrics precision score.” Web page [Online]. Available:

https://scikit-learn.org/stable/modules/generated/sklearn.metrics.precision_score.html

[138] “List of english stopwords.” Web page, Apr-2009 [Online]. Available: <http://xpo6.com/list-of-english-stop-words/>

[139] “Matplotlib.” Web page [Online]. Available: <https://matplotlib.org/>

[140] “Feature extraction.” Web page [Online]. Available: https://scikit-learn.org/stable/modules/feature_extraction.html

[141] A. S. Timmaraju, A. Palnitkar, and V. Khanna, “Game on! Predicting english premier league match outcomes.” Paper [Online]. Available: <http://cs229.stanford.edu/proj2013/TimmarajuPalnitkarKhanna-GameON!PredictionOfEPLMatchOutcomes.pdf>

[142] football-data, “Football-data-results/odds/tipsters.” web page, Nov-2018 [Online]. Available: <http://football-data.co.uk/englandm.php>

[143] Wikipedia, “MongoDB.” web page [Online]. Available: <https://en.wikipedia.org/wiki/MongoDB>

[144] mongoDB, “Install mongodb community edition on macOS.” web page [Online]. Available: <https://docs.mongodb.com/manual/tutorial/install-mongodb-on-os-x/>

[145] C. OKeefe, “Modern web automation with python and selenium.” Blog, Feb-2018 [Online]. Available: <https://realpython.com/modern-web-automation-with-python-and-selenium/>

[146] S. Jain, “Natural language processing for beginners: Using textblob.” Blog, Feb-2018 [Online]. Available: <https://www.analyticsvidhya.com/blog/2018/02/natural-language-processing-for-beginners-using-textblob/>

[147] “Yelp.” Web Page [Online]. Available: <https://www.yelp.com/>

[148] Y. Li, Y. Liu, R. Chiou, and P. Kalipatnapu, “Prediction of useful reviews on yelp dataset.” Paper [Online]. Available:

https://bcourses.berkeley.edu/files/65096735/download?download_frd=1

[149] P. Hajas, L. Gutierrez, and M. S. Krishnamoorthy, “Analysis of yelp reviews.” Web Page, Jul-2014 [Online]. Available: https://www.researchgate.net/publication/263736290_Analysis_of_Yelp_Review

[150] J. Koven, H. Siadati, and C.-Y. Lin, “Finding valuable yelp comments by personality, content, geo, and anomaly analysis.” Paper [Online]. Available: https://isis.poly.edu/~hossein/publications/yelp_finding_valuable_comments.pdf

[151] T. Hale, “How Much Data Does The World Generate Every Minute?” Web page, Jul-2017 [Online]. Available: <https://www.iflscience.com/technology/how-much-data-does-the-world-generate-every-minute/>

[152] R. Kh, “Big Data Analytics and the Financial Services Industry.” Web Page, Mar-2018 [Online]. Available: <https://www.smartdatacollective.com/online-stock-trading-impacted-big-data/>

[153] J. Barrat, ***Our Final Invention : Artificial Intelligence and the End of the Human Era.*** Thomas Dunne Books, 2013.

[154] David Rhodes and Daniel Stelter, ***Accelerating Out of The Great Recession : How to Win in a Slow-Growth Economy.*** McGraw Hill, 2010.

[155] W. Koehrsen, “Stock Prediction in Python.” Web Page, Jan-2018 [Online]. Available: <https://towardsdatascience.com/stock-prediction-in-python-b66555171a2>

[156] Ping-Ning Tan and Michael Steinbach and Vipin Kumar, ***Introduction to Data Mining.*** Pearson, 2014.

[157] CNN, “DOW JONES INDU AVERAGE NDX.” Web Page, Nov-2018 [Online]. Available: <https://money.cnn.com/data/dow30/>

[158] Quandl, “Quandl.” Web Page, Oct-2018 [Online]. Available: <https://www.quandl.com/>

[159] Quandl, “Home Depot Inc. (The) (HD) Stock Prices, Dividends and

Splits.” Web Page, Nov-2018 [Online]. Available: <https://www.quandl.com/data/EOD/HD-Home-Depot-Inc-The-HD-Stock-Prices-Dividends-and-Splits>

[160] SEC, “US Securities and Exchange Commission.” Web Page, Nov-2018 [Online]. Available: <https://www.sec.gov/>

[161] “DJI Performance.” Screenshot, Nov-2018.

[162] “Stock Performance.” Screenshot, Nov-2018.

[163] “Data Correlation 1.” Screenshot, Nov-2018.

[164] “Data Correlation 2.” Screenshot, Nov-2018.

[165] D. Karthick, “TREND FINDING METHOD -This strategy only for intraday.” Web Page, Jun-2011 [Online]. Available: <http://www.mudraa.com/trading/93282/2/trend-finding-method-this-strategy-only-for-intraday--2.html>

[166] “Data Correlation 3.” Screenshot, Nov-2018.

[167] V. Chauhan, “When we use Support Vector machine for Classification?” Web Page, Jan-2016 [Online]. Available: https://www.researchgate.net/post/When_we_use_Support_Vector_machine_for

[168] “SVM Model2 Summary.” Screenshot, Nov-2018.

[169] “SVM Model2 CM.” Screenshot, Nov-2018.

[170] “SVM Model2 Test Pred.” Screenshot, Nov-2018.

[171] “SVM Model3 Summary.” Screenshot, Nov-2018.

[172] “SVM Model3 CM.” Screenshot, Nov-2018.

[173] “SVM Model3 Test Pred.” Screenshot, Nov-2018.

[174] “SVM Model4 Summary.” Screenshot, Nov-2018.

- [175] “SVM Model4 CM.” Screenshot, Nov-2018.
- [176] “SVM Model4 Test Pred.” Screenshot, Nov-2018.
- [177] “SVM Model Performance.” Screenshot, Nov-2018.
- [178] “SVM Model Results.” Screenshot, Nov-2018.
- [179] Qasem Al-Radaideh and Eman Alnagi and Adel Abu Assaf, “Predicting Stock Prices Using Data Mining Techniques,” *Arab Conference on Information Technology*, 2013 [Online]. Available: https://www.researchgate.net/publication/281865047_Predicting_Stock_Prices_U
- [180] Kaggle, “Home credit default risk.” Web Page, Aug-2018 [Online]. Available: <https://www.kaggle.com/c/home-credit-default-risk#description>
- [181] Ö. Özdemir, “AN empirical investigation on consumer credit default risk.” paper, Oct-2004 [Online]. Available: https://www.econstor.eu/bitstream/10419/83237/1/dp_2004-20.pdf
- [182] python, “Python.” web page [Online]. Available: <https://www.python.org/>
- [183] M. Azure, “Provision the windows data science virtual machine on azure.” web page, 2018 [Online]. Available: <https://docs.microsoft.com/en-us/azure/machine-learning/data-science-virtual-machine/provision-vm>
- [184] M. A. Notebook, “Jupyter notebook.” Web page [Online]. Available: <https://notebooks.azure.com/help/jupyter-notebooks>
- [185] M. Mayo, “Notes on feature preprocessing: The what, the why, and the how.” Web Page, Oct-2018 [Online]. Available: <https://www.kdnuggets.com/2018/10/notes-feature-preprocessing-what-why-how.html>
- [186] J. Brownlee, “How to one hot encode sequence data in python.” Web Page, Jul-2017 [Online]. Available: <https://machinelearningmastery.com/how-to-one-hot-encode-sequence-data-in-python/>
- [187] NCSU, “Introduction to principal components and factor analysis.” Web

- Page, Oct-2018 [Online]. Available:
<ftp://statgen.ncsu.edu/pub/thorne/molevoclass/AtchleyOct19.pdf>
- [188] L. Breiman, “RandomForests.” Web Page [Online]. Available:
https://www.stat.berkeley.edu/~breiman/RandomForests/cc_home.htm
- [189] Scikit, “Random forest classifier.” Web Page [Online]. Available:
<https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html>
- [190] A. Bronshtein, “A quick introduction to k-nearest neighbors algorithm.” Web Page, Apr-2017 [Online]. Available:
<https://medium.com/@adi.bronshtein/a-quick-introduction-to-k-nearest-neighbors-algorithm-62214cea29c7>
- [191] D. B, “XGBoost.” Web Page, 2018 [Online]. Available:
<https://www.kaggle.com/dansbecker/xgboost>
- [192] A. C. S. Medical and E. C. Team, “Lifetime Risk of Developing or Dying From Cancer.” Webpage, Jan-2018 [Online]. Available:
<https://www.cancer.org/cancer/cancer-basics/lifetime-probability-of-developing-or-dying-from-cancer.html>
- [193] W. Wolberg, “Computerized Diagnosis of Breast Fine-Needle Aspirates,” *The Breast Journal*, vol. 3, no. 2, p. 77, Nov. 1997 [Online]. Available:
<https://pdfs.semanticscholar.org/34d3/ae4f6c88ae1e276f2aed07888e5a3eaf45fd.pdf>
- [194] N. C. Institute, “NCI Budget and Appropriations.” Web Page, Oct-2018 [Online]. Available: <https://www.cancer.gov/about-nci/budget>
- [195] D. Dheeru and E. Karra Taniskidou, “UCI Machine Learning Repository.” Webpage, 2017 [Online]. Available: <http://archive.ics.uci.edu/ml>
- [196] L. Borges, “Analysis of the Wisconsin Breast Cancer Dataset and Machine Learning for Breast Cancer Detection,” *Proceedings of XI Workshop de Visão Computacional*, Oct. 2015 [Online]. Available:
https://www.researchgate.net/publication/311950799_Analysis_of_the_Wisconsin_Breast_Cancer_Dataset_and_Machine_Learning_for_Breast_Cancer_Detection
- [197] Mangasarian, “Breast Cancer Diagnosis and Prognosis via Linear

Programming,” *Mathematical Programming Technical Report*, no. 10, Dec. 1994 [Online]. Available: <ftp://ftp.cs.wisc.edu/math-prog/tech-reports/94-10.pdf>

[198] MongoDB, “MongoDB.Atlas.” Webpage, Dec-2018 [Online]. Available: <https://www.mongodb.com/cloud/atlas>

[199] N. Street, “Xcyte Remote Breast Cancer Diagnosis System.” Web Page, Nov-2018 [Online]. Available: <https://dollar.biz.uiowa.edu/~street/xcyt/xcyt.html>

[200] Kaggle, “Give me some credit.” Web page [Online]. Available: <https://www.kaggle.com/c/GiveMeSomeCredit>

[201] B. Freshcorn, “Give me some credit :: 2011 competition data.” Web page [Online]. Available: <https://www.kaggle.com/brycecf/give-me-some-credit-dataset/kernels?sortBy=hotness&group=everyone&pageSize=20&datasetId=1889>

[202] Kaggle, “Give me some credit.” Web page [Online]. Available: <https://www.kaggle.com/c/GiveMeSomeCredit/data>

[203] Complete Dissertation by Statistics Solutions, “Correlation (pearson, kendall, spearman).” Web page [Online]. Available: <https://www.statisticssolutions.com/correlation-pearson-kendall-spearman/>

[204] J. Brownlee, “8 tactics to combat imbalanced classes in your machine learning dataset.” Web page, Aug-2015 [Online]. Available: <https://machinelearningmastery.com/tactics-to-combat-imbalanced-classes-in-your-machine-learning-dataset/>

[205] N. Donges, “The random forest algorithm.” Web page, Feb-2018 [Online]. Available: <https://towardsdatascience.com/the-random-forest-algorithm-d457d499ffcd>

[206] S. Swaminathan, “Logistic regression—Detailed overview.” Web page, Mar-2018 [Online]. Available: <https://towardsdatascience.com/logistic-regression-detailed-overview-46c4da4303bc>

[207] I. Reinstein, “XGBoost, a top machine learning method on kaggle,

explained.” Web page, Oct-2017 [Online]. Available: <https://www.kdnuggets.com/2017/10/xgboost-top-machine-learning-method-kaggle-explained.html>

[208] Upwork, “Neural networks demystified.” Web page [Online]. Available: <https://www.upwork.com/hiring/data/neural-networks-demystified/>

[209] A. Mishra, “Metrics to evaluate your machine learning algorithm.” Web page, Feb-24AD [Online]. Available: <https://towardsdatascience.com/metrics-to-evaluate-your-machine-learning-algorithm-f10ba6e38234>

[210] Github, “Official kaggle api.” Github [Online]. Available: <https://github.com/Kaggle/kaggle-api>

[211] Amazon EC2, “Amazon ec2.” Web page [Online]. Available: <https://aws.amazon.com/ec2/>

[212] L. Eikvil, “OCR - optical character recognition.” 1993 [Online]. Available: <https://pdfs.semanticscholar.org/9484/96f9d73cab9c7b4fd5c3b656d1e5b1dc50d3>

[213] S. Viala, “ECM and big data, better together.” 2014 [Online]. Available: <http://www.revasolutions.com/ecm-and-big-data-better-together/>

[214] J. Segarra, “Transform paper documents and pdf files with big data.” May-2017 [Online]. Available: <https://whatsnext.nuance.com/office-productivity/document-management-tools-improve-big-data-processes/>

[215] A. Rosebrock, “Using tesseract ocr with python.” Web page, Nov-2018 [Online]. Available: <https://www.pyimagesearch.com/2017/07/10/using-tesseract-ocr-python/>

[216] M. Lee, “Pytesseract - product description.” Web page, Nov-2018 [Online]. Available: <https://pypi.org/project/pytesseract/>

[217] L. Richardson, “Beautiful soup - product description.” Web page, Nov-2018 [Online]. Available: <https://pypi.org/project/beautifulsoup4/>

[218] J. Gonzalez, “Fuzzywuzzy.” Web page, Nov-2018 [Online]. Available: <https://github.com/seatgeek/fuzzywuzzy>

- [219] N. team, “NumPy.” Web page, Nov-2018 [Online]. Available: <http://www.numpy.org/>
- [220] O. team, “OpenCV library.” Web page, Nov-2018 [Online]. Available: <https://opencv.org/>
- [221] A. Clark, “Overview - python imaging library.” Web page, Nov-2018 [Online]. Available: <https://pillow.readthedocs.io/en/5.3.x/handbook/overview.html>
- [222] J. Wechsler, “Drug pricing and quality are top issues for 2018,” *PharmTech*, vol. 42, no. 1, pp. 20–21, Jan. 2018 [Online]. Available: <http://www.pharmtech.com/drug-pricing-and-quality-are-top-issues-2018-0>
- [223] U. D. of Health and H. Services, “FDA adverse event reporting system (faers) quarterly data extract files.” Online, Sep-2018 [Online]. Available: <https://fis.fda.gov/extensions/FPD-QDE-FAERS/FPD-QDE-FAERS.html>
- [224] Amazon, “Amazon free tier details.” Online [Online]. Available: <https://aws.amazon.com/ec2/?ft=n>
- [225] N. S. Foundation, “Chameleon cloud.” [Online]. Available: <https://www.chameleoncloud.org/>
- [226] Microsoft, “Microsoft azure.” [Online]. Available: <https://azure.microsoft.com/en-us/>
- [227] Cyberduck, “Cyberduck.” [Online]. Available: <https://cyberduck.io/>
- [228] Ubuntu, “Ubuntu 18.04.” [Online]. Available: <http://releases.ubuntu.com/18.04/>
- [229] Apple, “MacOS mojave.” [Online]. Available: <https://www.apple.com/macos/mojave/>
- [230] Microsoft, “Linux virtual machines pricing.” Online [Online]. Available: <https://azure.microsoft.com/en-us/pricing/details/virtual-machines/linux/>
- [231] K. Servick, “Pricey new cholesterol drug scrapes by in heart risk study,”

Science, Mar. 2017 [Online]. Available:
<http://www.sciencemag.org/news/2017/03/pricye-new-cholesterol-drug-scrapes-heart-risk-study>

[232] “Evolocuman - drug summar.” Web [Online]. Available:
<https://www.pdr.net/drug-summary/Repatha-evolocumab-3781.8222>

[233] H. Chaplin, “Accessing an ec2 instance on aws using cyberduck,” *ionic melon*, Dec. 2014.

[234] 12factor, “12-factor.” [Online]. Available: <https://12factor.net/>

[235] yelp, “Yelp.” [Online]. Available: <https://www.yelp.com/fusion>

[236] G. C. AI, “Cloud automl-main.” [Online]. Available:
<https://cloud.google.com/automl/>

[237] Google, “Cloud vision.” [Online]. Available:
<https://cloud.google.com/vision/docs/reference/rest/v1/images/annotate#EntityAr>

[238] Docker, “Docker.” [Online]. Available: <https://docs.docker.com/get-started/>

[239] Google, “Kubernetes.” [Online]. Available: <https://kubernetes.io>

[240] Kubernetes, “Pods.” [Online]. Available:
<https://kubernetes.io/docs/concepts/workloads/pods/>

[241] Google, “Kuberenets services.” [Online]. Available:
<https://kubernetes.io/docs/concepts/services-networking/service/#publishing-services---service-types>

[242] P. Christen, *Data Matching*. Berlin, Germany: Springer, 2012.

[243] C. Doran and C. Martin, “Measuring Success in Outsourced Cataloging: A Data-Driven Investigation,” *Cataloging & Classification Quarterly*, vol. 55, no. 5, pp. 307–317, 2017.

[244] OCLC, “[OCLC homepage].” Web Page, 2018 [Online]. Available:

<http://www.oclc.org>

[245] A. H. Turner, “OCLC WorldCat as a Cooperative Catalog,” *Cataloging & Classification Quarterly*, vol. 48, nos. 2-3, pp. 271–278, 2010.

[246] OCLC, “WorldCat Data Licensing.” Web Page, 2012 [Online]. Available: <https://www.oclc.org/content/dam/oclc/worldcat/documents/worldcat-data-licensing.pdf>

[247] OCLC, “WorldCat Rights and Responsibilities.” Web Page, Jun-2010 [Online]. Available: <https://www.oclc.org/en/worldcat/cooperative-quality/policy.html>

[248] J. McQueen, “Record Matching: Computers Cannot See That Which Is Obvious to ‘Any Idiot’ ... and Vice Versa,” *Information Today*, vol. 9, no. 11, pp. 41–44, Dec. 1992.

[249] Social Networks and Archival Context Project, “Social Networks and Archival Context.” Web Page, 2018 [Online]. Available: <http://snaccooperative.org/>

[250] OCLC, “VIAF: Virtual International Authority File.” Web Page, 2018 [Online]. Available: <http://viaf.org/>

[251] R. R. Larson and K. Janakiraman, “Connecting Archival Collections: The Social Networks and Archival Context Project,” in *Research and advanced technology for digital libraries, tpdl 2011*, 2011, pp. 3–14.

[252] O. F. Reyes-Galaviz, W. Pedrycz, Z. He, and N. J. Pizzi, “A Supervised Gradient-Based Learning Algorithm for Optimized Entity Resolution,” *Data & Knowledge Engineering*, vol. 112, pp. 106–129, 2017.

[253] Yale University Library, “YUL quicksearch search results.” RSS Feed, Nov-2018 [Online]. Available: http://search.library.yale.edu/catalog?commit=Search&format=atom&q=&search_field=all_fields

[254] OCLC, “Inside WorldCat.” 2018 [Online]. Available: <https://www.oclc.org/en/worldcat/inside-worldcat.html>

- [255] H. A. Olson, “The Power to Name: Representation in Library Catalogs,” *Signs*, vol. 26, no. 3, pp. 639–668, 2001 [Online]. Available: <http://www.jstor.org/stable/3175535>
- [256] IPFS, “[IPFS Homepage].” Web Page, 2018 [Online]. Available: <https://ipfs.io>
- [257] Coral Health, “Learn to Securely Share Files on the Blockchain with IPFS!” Blog, Feb-2018 [Online]. Available: <https://medium.com/@mycoralhealth/learn-to-securely-share-files-on-the-blockchain-with-ipfs-219ee47df54c>
- [258] V. Buterin, “On Public and Private Blockchains.” Blog, Aug-2015 [Online]. Available: <https://blog.ethereum.org/2015/08/07/on-public-and-private-blockchains/>
- [259] S. Alman, “Blockchain: Apps and Ideas.” Web Page, Jul-2018 [Online]. Available: <https://ischoolblogs.sjsu.edu/blockchains/blockchain-apps-and-ideas/>
- [260] J. Song, “Why Blockchain Is Hard.” Blog Post, May-2018 [Online]. Available: <https://medium.com/@jimmysong/why-blockchain-is-hard-60416ea4c5c>
- [261] B. A. Scriber, “A Framework for Determining Blockchain Applicability,” *IEEE Software*, vol. 35, no. 4, pp. 70–77, Jul. 2018 [Online]. Available: <https://www.computer.org/csdl/mags/so/2018/04/mso2018040070.html>
- [262] E. Buchman, J. Kwon, and Z. Milosevic, “The Latest Gossip on BFT Consensus.” arXiv.org, Sep-2018 [Online]. Available: <https://arxiv.org/abs/1807.04938v2>
- [263] G. Marin, “Understanding the Value Proposition of Cosmos.” Blog, Apr-2018 [Online]. Available: <https://blog.cosmos.network/understanding-the-value-proposition-of-cosmos-ecaeef63350d>
- [264] T. McConaghy, “[Reply in BigchainDB Gitter Chat].” Public Chat, Jun-2018 [Online]. Available: <https://gitter.im/bigchaindb/bigchaindb?at=5b16ac9599fa7f4c0648cc13>

- [265] T. Hess, “[Reply on Ethereum Stack Exchange].” Ethereum Stack Exchange, Feb-2016 [Online]. Available: <https://ethereum.stackexchange.com/questions/872/what-is-the-cost-to-store-1kb-10kb-100kb-worth-of-data-into-the-ethereum-block>
- [266] BigchainDB Contributors, “Key Concepts of BigchainDB.” Web Page, 2018 [Online]. Available: <https://www.bigchaindb.com/developers/guide/key-concepts-of-bigchaindb/>
- [267] BigchainDB GmbH, “BigchainDB 2.0: The Blockchain Database,” BigchainDB GmbH, Berlin, Germany, May 2018 [Online]. Available: <https://www.bigchaindb.com/whitepaper/bigchaindb-whitepaper.pdf>
- [268] killerstorm, “BigchainDB: A Prime Example of Blockchain Bullshit.” Reddit, May-2016 [Online]. Available: https://www.reddit.com/r/Bitcoin/comments/4j7wjf/bigchaindb_a_prime_exampl
- [269] T. McConaghy *et al.*, “BigchainDB: A scalable blockchain database.” Code Repository, p. 64, Jun-2016 [Online]. Available: <https://github.com/bigchaindb/whitepaper>
- [270] J. R. Douceur, “The Sybil Attack,” in *Revised papers from the first international workshop on peer-to-peer systems*, 2002, pp. 251–260 [Online]. Available: <http://dl.acm.org/citation.cfm?id=646334.687813>
- [271] A. M. Antonopoulos, *Mastering Bitcoin: Programming the Open Blockchain*, 2nd ed. Sebastopol, CA, United States: O’Reilly, 2017.
- [272] T. McConaghy, “BigchainDB 2.0 is Byzantine Fault Tolerant.” Blog, May-2018 [Online]. Available: <https://blog.bigchaindb.com/bigchaindb-2-0-is-byzantine-fault-tolerant-5ffdac96bc44>
- [273] Tendermint Contributors, “Tendermint.” Web Page, 2018 [Online]. Available: <https://tendermint.com/docs/>
- [274] T. McConaghy, “And We’re Off to the Races!” Blog, Sep-2018 [Online]. Available: <https://blog.bigchaindb.com/and-were-off-to-the-races-1aff2b66567c>
- [275] A. Granzotto, T. McConaghy, and M. Khan, “Performance Study:

Analysis of Transaction Throughput in a BigchainDB Network.” Code Repository, Sep-2018 [Online]. Available: <https://github.com/bigchaindb/BEPs/tree/master/23>

[276] T. McConaghy, “[Reply in BigchainDB Gitter Chat].” Public Chat, May-2018 [Online]. Available: <https://gitter.im/bigchaindb/bigchaindb?at=5b055eaf9ed336150ea41180>

[277] BigchainDB Contributors, “Querying BigchainDB.” Web Page, 2018 [Online]. Available: <https://docs.bigchaindb.com/en/latest/query.html>

[278] BigchainDB Contributors, “Drivers & Tools.” Web Page, 2018 [Online]. Available: <http://docs.bigchaindb.com/projects/server/en/latest/drivers-clients/index.html>

[279] S. Thomas, R. Reginelli, and A. Hope-Bailie, “Crypto-Conditions,” IETF, Jan. 2017 [Online]. Available: <https://tools.ietf.org/html/draft-thomas-crypto-conditions-02>

[280] T. McConaghy, “BigchainDB Transactions Spec v2.” Code Repository, Sep-2018 [Online]. Available: <https://github.com/bigchaindb/BEPs/blob/master/13/README.md>

[281] G. Dhameja, “Lifecycle of a BigchainDB Transaction.” Blog, Aug-2018 [Online]. Available: <https://blog.bigchaindb.com/lifecycle-of-a-bigchaindb-transaction-c1e34331cbaa>

[282] MongoDB Contributors, “Introduction to MongoDB.” Web Page, 2018 [Online]. Available: <https://docs.mongodb.com/manual/introduction/>

[283] K. M. Ford, “LC’s Bibliographic Framework Initiative and the Attractiveness of Linked Data,” *ISQ: Information Standards Quarterly*, vol. 24, no. 2/3, pp. 46–50, 2012 [Online]. Available: <https://groups.niso.org/publications/isq/2012/v24no2-3/ford/>

[284] Network Development and MARC Standards Office, Library of Congress, “The Library of Congress Network Development and MARC Standards Office.” Web Page, Oct-2013 [Online]. Available: <https://www.loc.gov/marc/ndmso.html>

- [285] Library of Congress, “Bibliographic Framework Initiative.” Web Page, 2018 [Online]. Available: <https://www.loc.gov/bibframe/>
- [286] R. Cyganiak, D. Wood, and M. Lanthaler, “RDF 1.1 Concepts and Abstract Syntax,” World Wide Web Consortium, Feb. 2014 [Online]. Available: <https://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/>
- [287] P. Hitzler, M. Krötzsch, B. Parsia, P. F. Patel-Schneider, and S. Rudolph, “OWL 2 Web Ontology Language Primer,” World Wide Web Consortium, Dec. 2012 [Online]. Available: <https://www.w3.org/TR/2012/REC-owl2-primer-20121211/>
- [288] Yale University Library, “[IUCAT record 8251671].” Library Catalog, Nov-2018 [Online]. Available: <https://iucat.iu.edu/catalog/8251671>
- [289] E. L. Morgan, “RDF Serializations.” Web Page, 2013 [Online]. Available: <https://sites.tufts.edu/liam/2014/01/31/serializations/>
- [290] Network Development and MARC Standards Office, Library of Congress, “marc2bibframe2.” Code Repository, Oct-2018 [Online]. Available: <https://github.com/lcnetdev/marc2bibframe2>
- [291] T. Thompson, “Invalid json key in metadata causes bigchaindb to crash.” Code Repository, Nov-2018 [Online]. Available: <https://github.com/bigchaindb/bigchaindb/issues/2605>
- [292] G. Dhameja, “Role Based Access Control for BigchainDB Assets.” Blog, Sep-2017 [Online]. Available: <https://blog.bigchaindb.com/role-based-access-control-for-bigchaindb-assets-b7cada491997>
- [293] BigchainDB Contributors, “BigchainDB RBAC Sample.” Code Repository, Sep-2018 [Online]. Available: <https://github.com/bigchaindb/project-jannowitz/tree/master/rbac>
- [294] BigchainDB Contributors, “How to Set Up a BigchainDB Network.” Web Page, 2018 [Online]. Available: <http://docs.bigchaindb.com/projects/server/en/latest/simple-deployment-template/network-setup.html>

- [295] BigchainDB Contributors, “[BigchainDB Testnet].” Web API, 2018 [Online]. Available: <https://test.bigchaindb.com/>
- [296] scikit, “Introduction to machine learning with scikit learn,” [Online]. Available: <http://scikit-learn.org/stable/tutorial/basic/tutorial.html>
- [297] shantanuladhwe, “Kaggle,” 2018 [Online]. Available: <https://www.kaggle.com/shantanuladhwe/bigmart/data>
- [298] H. Njike, “Simple eda on world trade commodities.” Website, 2018 [Online]. Available: <https://www.kaggle.com/hnike25/simple-eda-on-world-trade-commodities>
- [299] P. S. Foundation, “Python.” Website, 2001 [Online]. Available: <https://www.python.org/>
- [300] M. Waskom, “Seaborn.” Website, 2012 [Online]. Available: <https://seaborn.pydata.org/>
- [301] A. Developers, “Altair.” Website, 2016 [Online]. Available: https://altair-viz.github.io/getting_started/overview.html
- [302] P. Pandey, “Visualising geospatial data with python using folium.” Webiste, 2018 [Online]. Available: <https://www.kdnuggets.com/2018/09/visualising-geospatial-data-python-folium.html>
- [303] E. F. John Hunter Darren Dale and the Matplotlib development team, “Matplotlib 3.0.2.” Website, 2012 [Online]. Available: <https://matplotlib.org>
- [304] D. K. John Girard and K. Berg, *Strategic data-based wisdom in the big data era*. IGI Global, 2015 [Online]. Available: https://books.google.com/books/about/Strategic_Data_Based_Wisdom_in_the_B.html?id=yeqfBwAAQBAJ&printsec=frontcover&source=kp_read_button#v=onepage
- [305] J. Vanthienen and K. D. Witte, *Data analytics applications in education*. CRC Press, 2017 [Online]. Available: [https://books.google.com/books?id=c3JQDwAAQBAJ&printsec=frontcover&dq=Vanthienen,+J.,+%26+Witte,+K+\(2018\).+Data+analytics+applications+in+education.+Boca+Raton:+CRC+Press,+](https://books.google.com/books?id=c3JQDwAAQBAJ&printsec=frontcover&dq=Vanthienen,+J.,+%26+Witte,+K+(2018).+Data+analytics+applications+in+education.+Boca+Raton:+CRC+Press,+)

[4z7TeAhUHwYMKHVuYDUkQ6AEIMTAB#v=onepage&q&f=false](#)

- [306] B. Williamson, *Big data in education: The digital future of learning, policy and practice*. SAGE, 2017 [Online]. Available: <https://books.google.com/books?id=yFAsDwAAQBAJ&pg=PP5&dq=Williamson,+B.+%282017%29+Big+Data+in+Education:+The+Digital+Future+of+Learning.+Policy,+and+Practice>
- [307] N. Zarczynski, “2 light bulbs and a 100 story building.” Online, Mar-2011 [Online]. Available: <https://pointlessprogramming.wordpress.com/2011/03/11/2-light-bulbs-and-a-100-story-building/>
- [308] D. Maister, “Centralisation of inventories and the ‘square root law’,” *International Journal of Physical Distribution*, vol. 6, no. 3, pp. 124–134, 1976 [Online]. Available: <https://doi.org/10.1108/eb014366>
- [309] O. Astrachan, “Bubble sort: An archaeological algorithmic analysis,” *SIGCSE Bull.*, vol. 35, no. 1, pp. 1–5, Jan. 2003 [Online]. Available: <http://doi.acm.org/10.1145/792548.611918>
- [310] R. Cole, “Parallel merge sort,” *SIAM Journal on Computing*, vol. 17, no. 4, pp. 770–785, 1988 [Online]. Available: <https://doi.org/10.1137/0217049>
- [311] D. R. MUSSER, “Introspective sorting and selection algorithms,” *Software: Practice and Experience*, vol. 27, no. 8, pp. 983–993 [Online]. Available: <https://onlinelibrary.wiley.com/doi/abs/10.1002/%28SICI%291097-024X%28199708%2927%3A8%3C983%3A%3AAID-SPE117%3E3.0.CO%3B2-%23>
- [312] P. Madhusudan and X. Qiu, “Efficient decision procedures for heaps using strand,” in *Static analysis*, 2011, pp. 43–59.
- [313] W3.CSS, “Python list sort() method.” Online [Online]. Available: https://www.w3schools.com/python/ref_list_sort.asp
- [314] R. D. Dutton, “Weak-heap sort,” *BIT Numerical Mathematics*, vol. 33, no. 3, pp. 372–381, Sep. 1993 [Online]. Available: <https://doi.org/10.1007/BF01990520>

- [315] sqlcourse.com, “What is sql?” Online [Online]. Available: <http://www.sqlcourse.com/intro.html>
- [316] MySQL, *Limits on table size*. MySQL [Online]. Available: <https://dev.mysql.com/doc/mysql-reslimits-excerpt/5.5/en/table-size-limit.html>
- [317] Microsoft, *Excel specifications and limits*. Microsoft [Online]. Available: <https://support.office.com/en-us/article/excel-specifications-and-limits-1672b34d-7043-467e-8e27-269d656771c3#top>
- [318] Python, *Python 3.7.1 documentation*. Python Software Foundation, 2016 [Online]. Available: <https://docs.python.org/3/>
- [319] Spyder, *Spyder: The scientific python development environment — documentation*. [Online]. Available: <https://docs.spyder-ide.org/>
- [320] Python, *Random - generate pseudo-random numbers*. Python Software Foundation [Online]. Available: <https://docs.python.org/3/library/random.html>
- [321] Python, *15.3. Time — time access and conversions*. Python Software Foundation [Online]. Available: <https://docs.python.org/2/library/time.html>
- [322] Python, *The python standard library*. Python Software Foundation [Online]. Available: <https://docs.python.org/3/library/>
- [323] Wikipedia, “Bubble sort.” Web page [Online]. Available: https://en.wikipedia.org/wiki/Bubble_sort
- [324] T. H. Cormen, C. Stein, R. L. Rivest, and C. E. Leiserson, *Introduction to algorithms*, 2nd ed. McGraw-Hill Higher Education, 2001.
- [325] Wikipedia, “Merge sort.” Web page [Online]. Available: https://en.wikipedia.org/wiki/Merge_sort
- [326] D. E. Knuth, *The art of computer programming, volume 1 (3rd ed.): Fundamental algorithms*. Redwood City, CA, USA: Addison Wesley Longman Publishing Co., Inc., 1997.
- [327] Wikipedia, “Insertion sort.” Web page [Online]. Available:

https://en.wikipedia.org/wiki/Insertion_sort

[328] R. Sedgewick and K. Wayne, *Algorithms*, 4th ed. Addison-Wesley Professional, 2011.

[329] Wikipedia, “Shellsort.” Web page [Online]. Available: <https://en.wikipedia.org/wiki/Shellsort>

[330] Wikipedia, “Selection sort.” Web page [Online]. Available: https://en.wikipedia.org/wiki/Selection_sort

[331] A. López-Ortiz, U. C. Meyer, and R. Sedgewick, “Data structures and advanced models of computation on big data (dagstuhl seminar 14091),” *Dagstuhl Reports*, vol. 4, no. 2, pp. 129–149, 2014.

[332] Rosettacode, “Sorting algorithms/strand sort.” Web page, Jul-2018 [Online]. Available: https://rosettacode.org/wiki/Sorting_algorithms/Strand_sort#Python

[333] Wikipedia, “Strand sort.” Web page [Online]. Available: https://cs.wikipedia.org/wiki/Strand_sort

[334] T. Peters, “[Python-Dev] Sorting,” 2002 [Online]. Available: <https://mail.python.org/pipermail/python-dev/2002-July/026837.html>

[335] Wikipedia, “Heap sort.” Web page [Online]. Available: <https://en.wikipedia.org/wiki/Heapsort#Pseudocode>

[336] D. B. Thomas, W. Luk, P. H. Leong, and J. D. Villasenor, “Gaussian random number generators,” *ACM Comput. Surv.*, vol. 39, no. 4, Nov. 2007 [Online]. Available: <http://doi.acm.org/10.1145/1287620.1287622>

[337] M. Dvorský, “History of massive-scale sorting experiments at google,” 2016 [Online]. Available: <https://cloud.google.com/blog/products/gcp/history-of-massive-scale-sorting-experiments-at-google>

[338] Interactive Python, “5.9. The insertion sort.” Web page [Online]. Available:

<http://interactivepython.org/courselib/static/pythonds/SortSearch/TheInsertionSort.html>

[339] W3Resource, “Python data structures and algorithms: Shell sort.” Web page, Aug-2018.

[340] M. K. Shivi Aggarwal Abby Akku, “HeapSort.” Web page [Online]. Available: <https://www.geeksforgeeks.org/heap-sort/>

[341] Interactive Python, “5.11. The merge sort.” Web page [Online]. Available: <http://interactivepython.org/courselib/static/pythonds/SortSearch/TheMergeSort.html>

[342] Interactive Python, “5.8. The selection sort.” Web Page [Online]. Available:

<http://interactivepython.org/runestone/static/pythonds/SortSearch/TheSelectionSort.html>