

Politechnika Warszawska

WYDZIAŁ ELEKTRONIKI
I TECHNIK INFORMACYJNYCH



Instytut Informatyki

Studia Podyplomowe
Big Data - przetwarzanie i analiza dużych zbiorów danych

PRACA KOŃCOWA

Marcin Rydelski

Procesy ETL w środowisku Microsoft Azure

Opiekun pracy
Dr inż. Jakub Nowacki

Warszawa, 2020

Politechnika Warszawska

WYDZIAŁ ELEKTRONIKI
I TECHNIK INFORMACYJNYCH



Institute of Computer Science

Postgraduate studies

Big Data - processing and analysis of large data sets

FINAL THESIS

Marcin Rydelski

ETL processes in Microsoft Azure environment

Thesis supervisor
Jakub Nowacki BEng, PhD

Warsaw, 2020

STRESZCZENIE

Procesy ETL w środowisku Microsoft Azure

Celem pracy jest połączenie nowoczesnych narzędzi do przetwarzania danych w celu stworzenia strumienia integracji danych w usłudze chmurowej Microsoft Azure.

Cały proces składa się z generowania, transformacji i ewaluacji danych przy użyciu technologii Big Data oraz algorytmów uczenia maszynowego. Dane wyjściowe strumienia ETL są następnie eksportowane do bazy danych Azure SQL i interaktywnego pulpitu nawigacyjnego Power BI.

Słowa kluczowe: Big Data, MS Azure, ETL, Uczenie Maszynowe, Python, Apache Spark

SUMMARY

ETL processes in Microsoft Azure environment

The aim of the thesis is to combine modern data processing tools to create data integration pipelines in the Microsoft Azure cloud service.

The overall process consists of data generation, transformation and evaluation using Big Data technologies and Machine Learning algorithms. The output of the ETL pipeline is next exported to Azure SQL database and interactive Power BI dashboard.

Keywords: Big Data, MS Azure, ETL, Machine Learning, Python, Apache Spark

TABLE OF CONTENTS

INTRODUCTION.....	2
1. DATA OVERVIEW	3
2. DATA GENERATOR.....	4
3. MACHINE LEARNING MODELING	5
4. DATA PIPELINE WITH AZURE DATA FACTORY	8
5. POWER BI DATA DASHBOARD	12
6. CONCLUSIONS	14
7. LITERATURE	15
8. LIST OF FIGURES	16
9. LIST OF TABLES	16
10. APPENDIX	17

INTRODUCTION

Over 2.5 quintillion bytes of data are created every single day and by 2020, it's estimated that 1.7MB of data will be created every second for every person on earth [1].

This phenomenon, called **Big Data**, makes it possible to gain more complete answers because of more information and more complete answers mean more confidence in the data - which means a completely different approach to tackling problems. This tactic must manage three key features of Big Data:

- **Volume** - high volumes of low-density, unstructured data,
- **Velocity** – the fast rate at which data is received,
- **Variety** - different types of data that are available (also unstructured) [2].

In 2020 there are many technologies being used to take on this task and new ones are created each month. Behind every approach and every solution in the business, there is a tool that enables specialists to drive a result. We can divide big data-related technologies into two groups:

- **Data Engineering** – data collection with infrastructure setting, proper formatting and generally processing it,
- **Data Analysis** – the act on the received information, getting insights into how the business works and make further decisions rely upon [3].

Both are essential in the process of creating value from received data. The goal of the thesis is to build a modern pipeline which shows how to implement ETL (extract, transform and load) data integration process with analytics workload in the Microsoft Azure cloud environment.

For the purpose of the thesis a case study involving “Wine Quality Data Set” from Machine Learning Center [4] was chosen to create a data generator to send instances of data which next can be transformed and evaluated using tools available in the Microsoft Azure cloud service:

- Azure Data Factory,
- Azure Databricks with Apache Spark and Machine Learning algorithms,
- Azure Storage – Data Lake Gen2,
- Azure SQL Server,
- Microsoft Power BI.

1. DATA OVERVIEW

There are two datasets: red and white variants of the Portuguese "Vinho Verde" wine. For the purposes of analysis and data transformation red wine dataset was chosen for further processing.

Dataset consists of 1599 instances and 12 variables, as shown in below example:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
1.	7,4	0,70	0,00	1,9	0,076	11	34	0,9978	3,51	0,56	9,4	5
2.	7,8	0,88	0,00	2,6	0,098	25	67	0,9968	3,2	0,68	9,8	5
3.	7,8	0,76	0,04	2,3	0,092	15	54	0,997	3,26	0,65	9,8	5
4.	11,2	0,28	0,56	1,9	0,075	17	60	0,998	3,16	0,58	9,8	6
5.	7,4	0,70	0,00	1,9	0,076	11	34	0,9978	3,51	0,56	9,4	5

Table 1.1 Wine dataset sample

First eleven variables are a continuous data and the quality variable is a categorical column with bell-shape distribution. Most values are in 5th and 6th class:

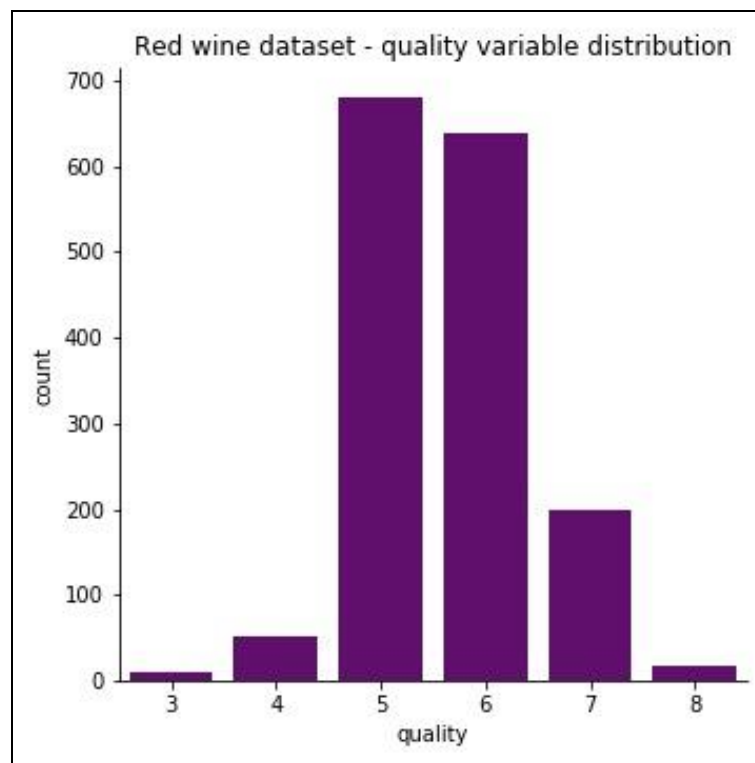


Figure 1.1 Quality variable distribution

2. DATA GENERATOR

In order to store more varied instances of data on Azure Data Lake Gen2 Blob Storage a data generator, in Python programming language, was created. At first it deletes quality variable, next chooses randomly (with repetition) 5000 rows of original red wine dataset, and creates additional variables for each instance:

- datetime (20190101_000000 to 20191231_235959 range),
- an arbitrary location (from 0 to 9),

which are then stored as a file name, for example: “2019011_160512_7.json”.

Next each of the original eleven continuous variables is randomly modified, stored in JSON format and sent to “rawsink” storage container.

```
# Load data
df = pd.read_csv(filepath_or_buffer="../../data/raw/winequality-red.csv", sep=";")
df = df.drop("quality", axis=1)
# Loop
for i in range(5000):
    x = df.sample(1)
    location = np.random.randint(0,9)
    sample_time = random_date("20190101_000000", "20191231_235959", np.random.random())
    # sample_time = time.strftime("%Y%m%d_%H%M%S") # for current time
    path = os.path.join("../../data//processed/", sample_time + "_" +
                        str(location) + ".json")
    # Randomly change instance values
    for j in range(11):
        x.iloc[0,j] = x.iloc[0,j] + np.random.normal(0, 0.05) * x.iloc[0,j]
    # Save to *.json
    x.to_json(
        path_or_buf=path,
        orient="records",
    )
    # Push to Blob Storage
    block_blob_service.create_blob_from_path(
        container_name="rawsink",
        blob_name=os.path.basename(path),
        file_path=path,
        content_settings=ContentSettings(content_type="application/JSON")
    )
```

Figure 2.1 Red wine metrics data generator

3. MACHINE LEARNING MODELING

A following step in the process of constructing a modern data pipeline is creating a machine learning model which can later be used to automatically label the wine quality from incoming JSON files.

A Random forest classifier was chosen as a method able to perform multiclass classification and originally implemented in Apache Spark MLlib - scalable machine learning library. Random forests combine many decision trees to reduce the risk of overfitting [5].

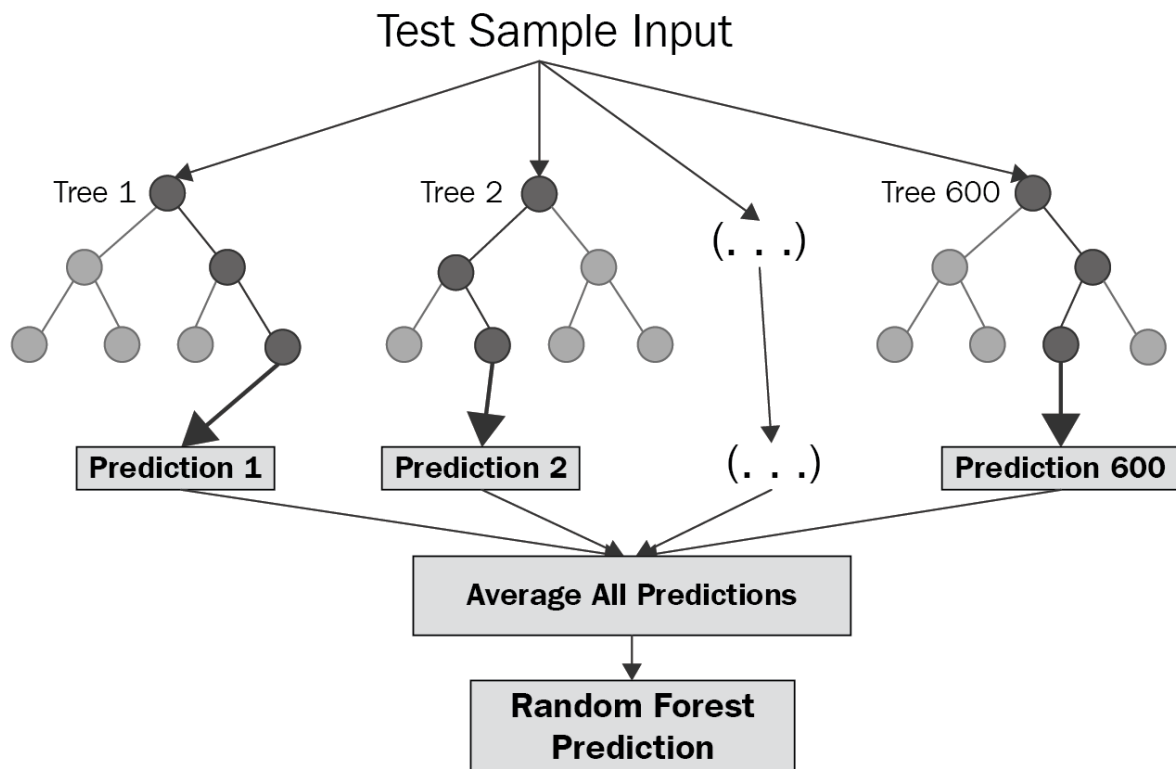


Figure 3.1 Random forest classifier schema [6]

Because fine tuning machine learning predictive model is a crucial step to improve accuracy of the forecasted results, original wine dataset was first split into train / test data (80/20 ratio) and with the usage of grid search on four hyperparameters (impurity, number of trees, maximum tree depth and maximum bins parameters) was next was fit to create the best possible output. All machine learning processes were created with Apache PySpark - the Python API for Spark.

```

) # Split the input data into training and test DataFrames with 80% to 20% weights
train_data, test_data = data.randomSplit([0.80, 0.20], seed=12345)

# Load Random Forest Classifier
rfc = RandomForestClassifier(featuresCol="features", labelCol="label", )

# Build ParamGrid
paramGrid = ParamGridBuilder()\
    .addGrid(rfc.impurity, ["entropy", "gini"]) \
    .addGrid(rfc.numTrees, [10, 25, 50, 75, 100, 150, 200, 250]) \
    .addGrid(rfc.maxDepth, [2, 3, 5, 10, 15, 20, 25, 30]) \
    .addGrid(rfc.maxBins, [5, 10, 20, 25, 30, 50, 75]) \
    .build()

tv = TrainValidationSplit(estimator=rfc,
                          estimatorParamMaps=paramGrid,
                          evaluator=MulticlassClassificationEvaluator(),
                          # 80% of the data will be used for training, 20% for validation.
                          trainRatio=0.80)

rfc_model = tv.fit(train_data)

```

Figure 3.2 Random forest classifier with hyperparameters tuning

The final Random forest classifier has an accuracy: 0.6405 and RMSE: 0.6803, which is only calculated for multiclassification problem to illustrate that the model, when wrong, is usually one class below / above the original quality label, as shown in below chart:

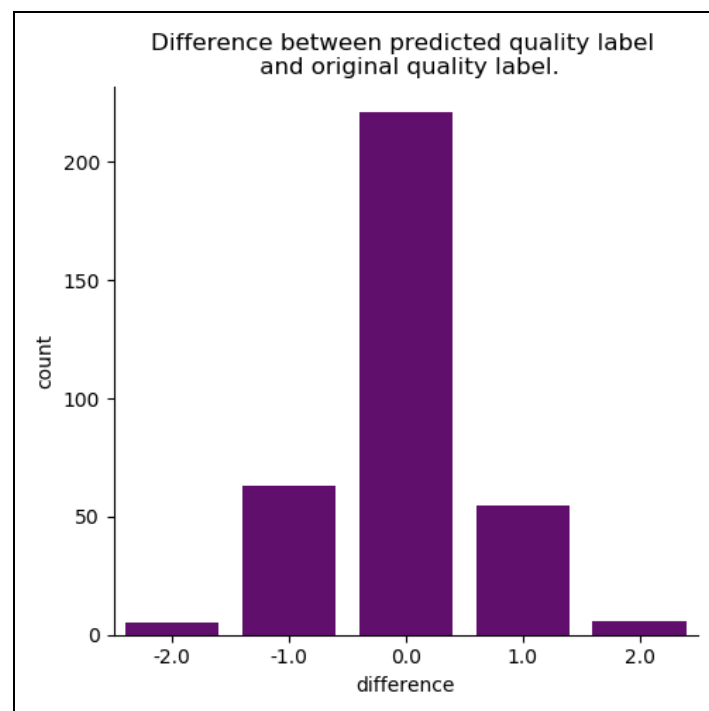


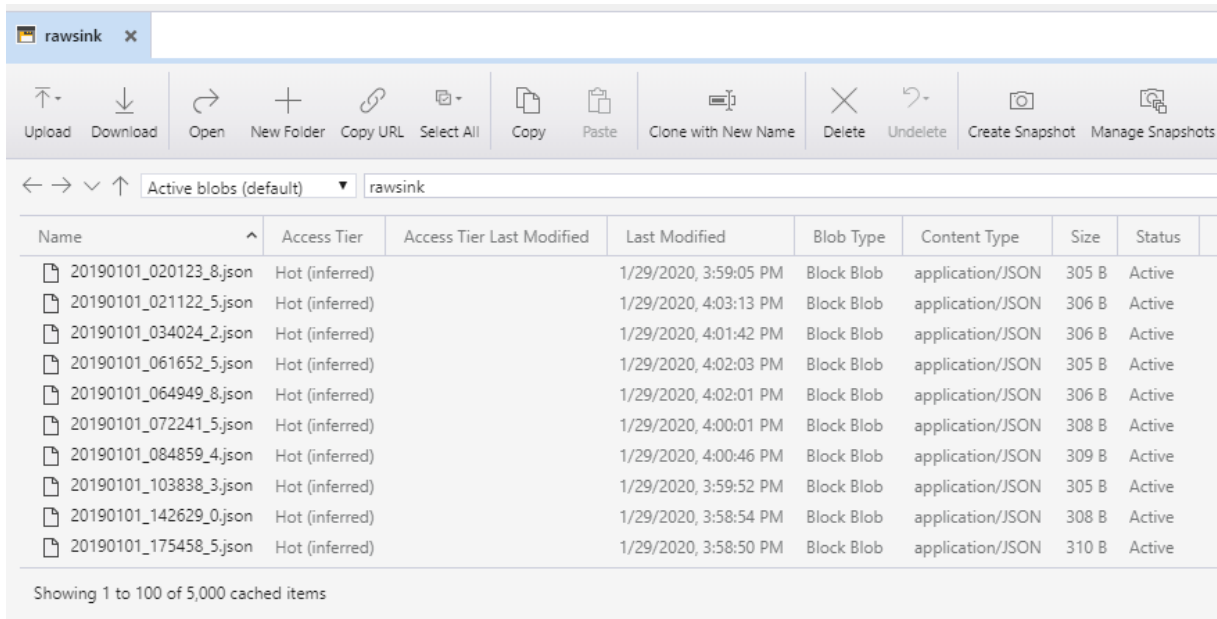
Figure 3.3. Difference between predicted quality label and original quality label

The best model was constructed with below hyperparameters:

- impurity: gini,
- number of trees: 75,
- maximum tree depth: 15,
- maximum bins parameters: 30.

4. DATA PIPELINE WITH AZURE DATA FACTORY

With raw JSON files stored by now on Azure Storage Account and ready to use machine learning model for data evaluation, the next step is to build data pipeline for automatization of the whole process. As a case study we can image that raw readings of data parameters come each day to our rawsink Blob Storage from which they should be next transformed, evaluated and pushed to more readable format, available for SQL queries, for more complex data exploration.



The screenshot shows the Azure Storage Explorer interface for a container named 'rawsink'. The interface includes a toolbar with actions like Upload, Download, Open, New Folder, Copy URL, Select All, Copy, Paste, Clone with New Name, Delete, Undelete, Create Snapshot, and Manage Snapshots. Below the toolbar, there's a breadcrumb navigation showing 'Active blobs (default)' and 'rawsink'. The main area displays a table of blobs with the following columns: Name, Access Tier, Access Tier Last Modified, Last Modified, Blob Type, Content Type, Size, and Status. The table lists 10 JSON files, all of which are 'Hot (inferred)' and 'Active'.

Name	Access Tier	Access Tier Last Modified	Last Modified	Blob Type	Content Type	Size	Status
20190101_020123_8.json	Hot (inferred)		1/29/2020, 3:59:05 PM	Block Blob	application/JSON	305 B	Active
20190101_021122_5.json	Hot (inferred)		1/29/2020, 4:03:13 PM	Block Blob	application/JSON	306 B	Active
20190101_034024_2.json	Hot (inferred)		1/29/2020, 4:01:42 PM	Block Blob	application/JSON	306 B	Active
20190101_061652_5.json	Hot (inferred)		1/29/2020, 4:02:03 PM	Block Blob	application/JSON	305 B	Active
20190101_064949_8.json	Hot (inferred)		1/29/2020, 4:02:01 PM	Block Blob	application/JSON	306 B	Active
20190101_072241_5.json	Hot (inferred)		1/29/2020, 4:00:01 PM	Block Blob	application/JSON	308 B	Active
20190101_084859_4.json	Hot (inferred)		1/29/2020, 4:00:46 PM	Block Blob	application/JSON	309 B	Active
20190101_103838_3.json	Hot (inferred)		1/29/2020, 3:59:52 PM	Block Blob	application/JSON	305 B	Active
20190101_142629_0.json	Hot (inferred)		1/29/2020, 3:58:54 PM	Block Blob	application/JSON	308 B	Active
20190101_175458_5.json	Hot (inferred)		1/29/2020, 3:58:50 PM	Block Blob	application/JSON	310 B	Active

Showing 1 to 100 of 5,000 cached items

Figure 4.1 Raw JSON files stored on Azure Data Lake Gen2 Blob Storage

For this purpose, we will use **Azure Data Factory** – a cloud-based ETL and data integration service that allows you to create data-driven workflows for orchestrating data movement and transforming data at scale [7]. The core schema of the data pipeline was adapted from “Integrating Data in Microsoft Azure” course by Marcelo Pastorino [8].

First, we create **Get Metadata** activity which connects to our rawsink storage and lists all the items (files and folders) in our storage. Next, just in case any folder schema is sent to the rawsink, we use **Filter** activity to filter out folders from the items returned by the Get Metadata activity. As we cannot delete all files at once from our rawsink container, because it can receive new files while we are running the process, we will move files between two storage containers rawsink and sinkstage to guarantee that each file is processed only once. A **ForEach** activity is used for this process.

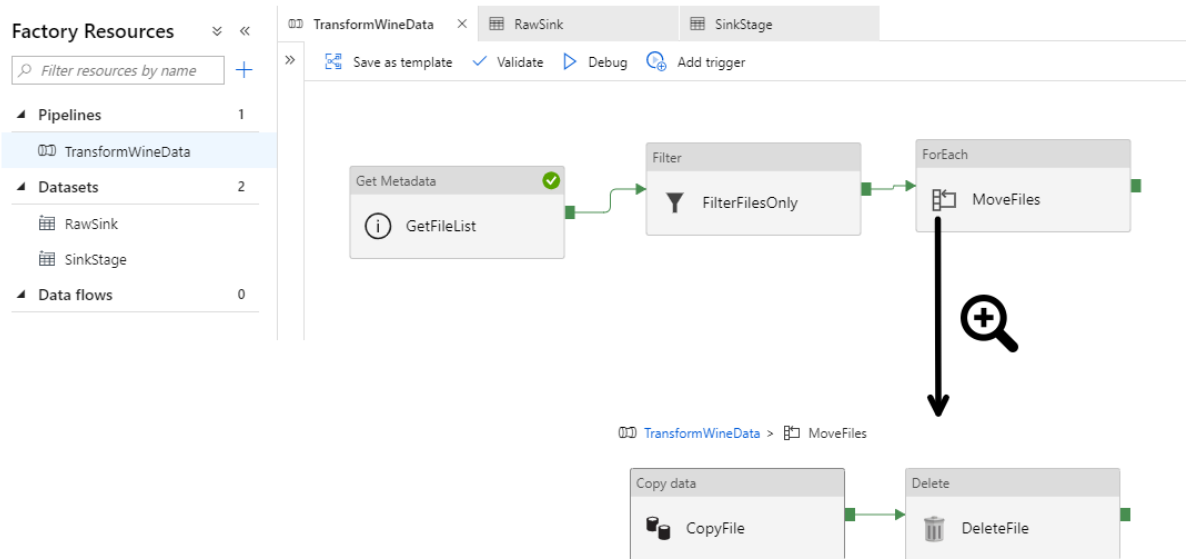


Figure 4.2. Moving JSON files between containers in Azure Data Factory pipeline

After the filtered files were move to proper storage container, advanced data transformations and wine data quality evaluation will be done inside **Azure Databricks** notebook, which will be connected to previously created pipeline. Azure Databricks is an Apache Spark-based analytics platform optimized for the Microsoft Azure cloud services platform to provide one-click setup, streamlined workflows, and an interactive workspace [9].

Initially, a **Databricks cluster** - a set of computation resources and configurations needs to be built in order to run data engineering, data science, and data analytics workloads. For the purpose of the thesis a simple cluster with one worker was created.

Cluster Mode ⓘ

Standard ▼

Databricks Runtime Version

6.2 (includes Apache Spark 2.4.4, Scala 2.11)

New This Runtime version supports only Python 3.

Autopilot Options

☐ Enable autoscaling ⓘ

☒ Terminate after 15 minutes of inactivity ⓘ

Worker Type

Standard_F4s 8.0 GB Memory, 4 Cores, 0.5 DBU

Workers

1

Driver Type

Standard_F4s 8.0 GB Memory, 4 Cores, 0.5 DBU

Figure 4.3 Databricks cluster configuration

Next, a new notebook with the code to load, transform and store data back into output database is created. First step is to load all the JSON files to single Data Frame object and enrich the dataset with date, time and location columns. Following, locations are being mapped with corresponding values from provided dictionary.

```

1 data = spark.read.load(path=file_path, format="json", inferSchema="true")
2 data = data.withColumn("date", input_file_name().substr(-22,8))
3 data = data.withColumn("date", F.to_date(data["date"], "yyyyMMdd"))
4 data = data.withColumn("time", input_file_name().substr(-13,6))
5 data = data.withColumn("time", F.unix_timestamp("time", "HHmmss")) \
6     .withColumn("time", F.from_unixtime("time", "HH:mm:ss"))
7 data = data.withColumn("location", input_file_name().substr(-6,1))
8
9 # Map locations column to proper names
10 location_dict = {
11     "0": "Okanagan Valley",
12     "1": "Bordeaux",
13     "2": "New York",
14     "3": "Mendoza",
15     "4": "Willamette Valley",
16     "5": "Tuscany",
17     "6": "Cape Town",
18     "7": "Napa & Sonoma",
19     "8": "Barcelona",
20     "9": "Yarra Valley",
21 }
22 data = data.replace(to_replace=location_dict, subset=["location"])

```

Figure 4.4. Raw JSON files loaded and transformed in Databricks notebook

Next a Vector Assembler of wine parameters and earlier created Random forest classifier are prepared to predict wine quality.

```

1 #Create and configure the assembler
2 assembler = VectorAssembler(inputCols=featureColumns, outputCol="features")
3
4 # Transform the original data
5 data = assembler.transform(data)
6
7 # Load model and predict wine quality
8 loaded_model = TrainValidationSplitModel.load(model_path)
9 loaded_preds = loaded_model.transform(data).select("features", "prediction")
10 loaded_preds.show()

```

► (15) Spark Jobs
 ► loaded_preds: pyspark.sql.dataframe.DataFrame = [features: udt, prediction: double]

features	prediction
[5.5510557234,0.4...]	6.0
[8.6048608452,0.7...]	5.0
[7.1085461888,0.6...]	6.0
[8.560441981,0.64...]	6.0
[7.4957589191,0.5...]	5.0
[7.499880307,0.56...]	6.0

Figure 4.5 Wine quality predictions

Lastly new Data Frame is being created to store sorted and evaluated data.

```

1 # Create and sort final DataFrame
2 data = data.join(loaded_preds, on=["features"]).drop("features")
3 data = data.withColumnRenamed("prediction", "quality")
4 data = data.select("date", "time", "location", "quality", "fixed acidity",
5                   "volatile acidity", "citric acid", "residual sugar", "chlorides",
6                   "free sulfur dioxide", "total sulfur dioxide", "density",
7                   "pH", "sulphates", "alcohol")
8 data = data.orderBy(data["date"], data["time"])
9 data.show()

```

▶ (1) Spark Jobs

▶ data: pyspark.sql.dataframe.DataFrame = [date: date, time: string ... 13 more fields]

date	time	location	quality	fixed acidity	volatile acidity	citric acid
2019-02-07	11:15:14	Napa & Sonoma	5.0	5.6869727905	0.3891719446	0.1345689512
2019-03-17	11:33:39	Barcelona	6.0	8.560441981	0.6484688708	0.1505259157
2019-04-18	08:57:09	Mendoza	6.0	5.5510557234	0.4572196417	0.0200125918
2019-04-25	11:16:58	Napa & Sonoma	6.0	8.4153018572	0.3719475971	0.522524967
2019-05-03	21:32:06	Bordeaux	5.0	9.4623597641	0.8330322293	0.0
2019-05-29	11:23:59	Yarra Valley	5.0	7.4957589191	0.5554430641	0.2862950472
2019-07-02	11:24:11	Willamette Valley	5.0	8.6048608452	0.7068944006	0.3082012032
2019-07-17	18:51:45	Bordeaux	6.0	7.2164677305	0.5223865769	0.0865971765
2019-07-28	06:36:01	Willamette Valley	6.0	7.1085461888	0.6808827466	0.0095572524
2019-12-06	10:47:57	Yarra Valley	6.0	7.499880307	0.5665005634	0.020010508

Figure 4.6 Final Data Frame output

And next it is sent to **Azure SQL Database** - a general-purpose relational database, provided as a managed service, which enables to process both relational data and non-relational structures, such as graphs, JSON, spatial, and XML [10].

The last step in the Azure Data Factory pipeline, which ensures that the JSON files are processed only once, is adding **Delete** activity to delete process files from the stagesink container. Whole process (presented in appendix number 1 to the thesis) can be tracked in real time using **Monitor** panel.

Activity runs

Pipeline run ID af1f7362-a525-4ba1-8e31-f04a42720a6d

All status ▾					
Showing 1 - 31 of 31 items					
ACTIVITY NAME	ACTIVITY TYPE	RUN START ↑↓	DURATION	STATUS	INTEGRATION RUNTIME
DeleteStageSinkData	Delete	1/30/20, 11:07:13 AM	00:00:12	✔ Succeeded	DefaultIntegrationRuntime (West Europe)
TransformEvaluateData	DatabricksNotebc	1/30/20, 11:06:17 AM	00:00:42	✔ Succeeded	DefaultIntegrationRuntime (West Europe)
DeleteFile	Delete	1/30/20, 11:05:56 AM	00:00:07	✔ Succeeded	DefaultIntegrationRuntime (West Europe)
DeleteFile	Delete	1/30/20, 11:05:55 AM	00:00:09	✔ Succeeded	DefaultIntegrationRuntime (West Europe)
DeleteFile	Delete	1/30/20, 11:05:51 AM	00:00:10	✔ Succeeded	DefaultIntegrationRuntime (West Europe)

Figure 4.7 Azure Data Factory Monitor panel

5. POWER BI DATA DASHBOARD

Since the input data has been transformed, evaluated and stored on Azure SQL Server it is possible to prepare additional SQL queries and create interactive dashboards.

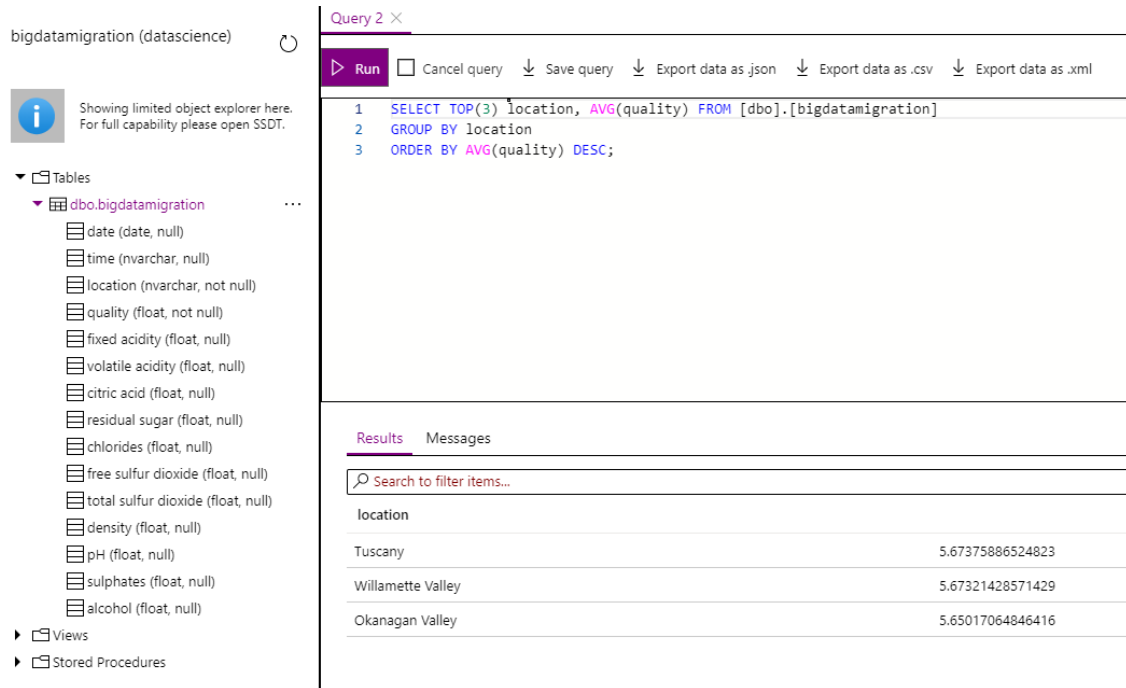


Figure 5.1 Azure SQL Database example query

For graphical exploration it is possible to use **Microsoft Power BI**, connect directly to Azure SQL Database and create reports that use live data. Microsoft Power BI is a business analytics solution that allows good quality data visualizations and sharing insights on website. For the work objective a simple dashboard with direct access to **bigdatamigration** database.

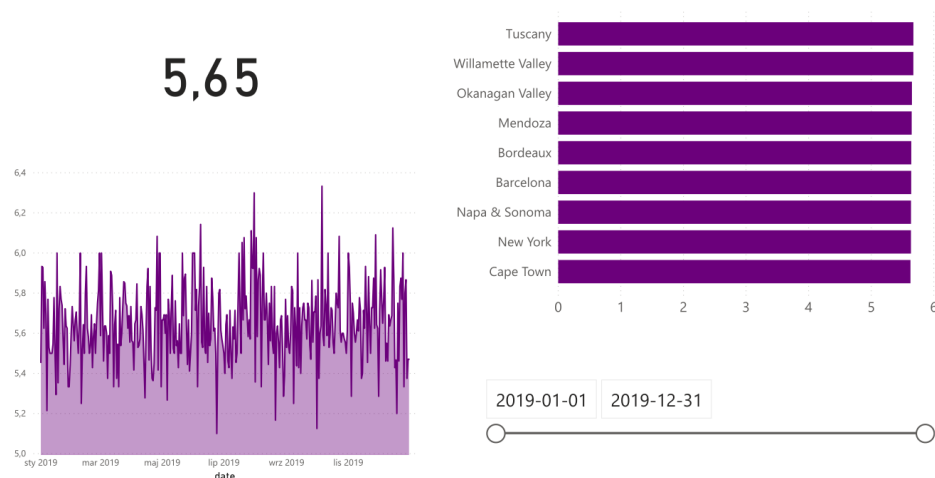


Figure 5.2 Power BI example dashboard

Moving around the interactive panel it is also possible to select a specific date range and filter only the location we want to analyze.

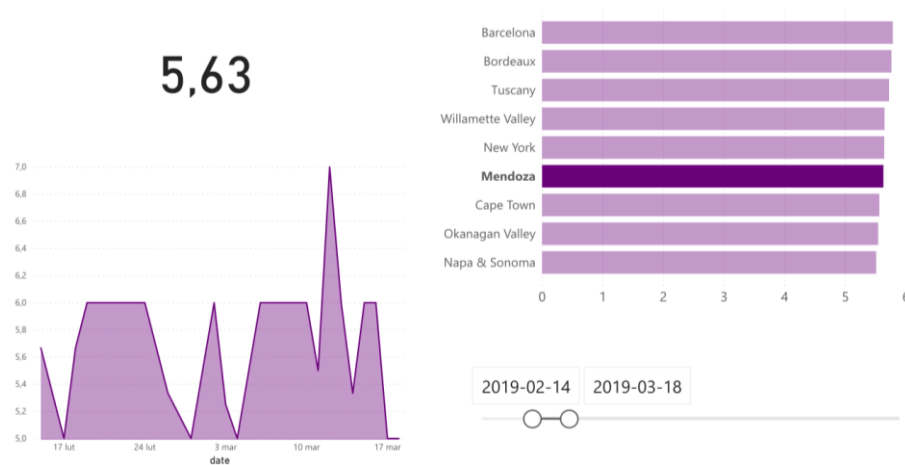


Figure 5.3 Power BI selection example dashboard

Other worth mentioning option is an integration of Power BI with ArcGIS, which offers enhanced mapping and analysis capabilities, demographic data, and compelling visualizations. An example of spatial based dashboard is shown below:

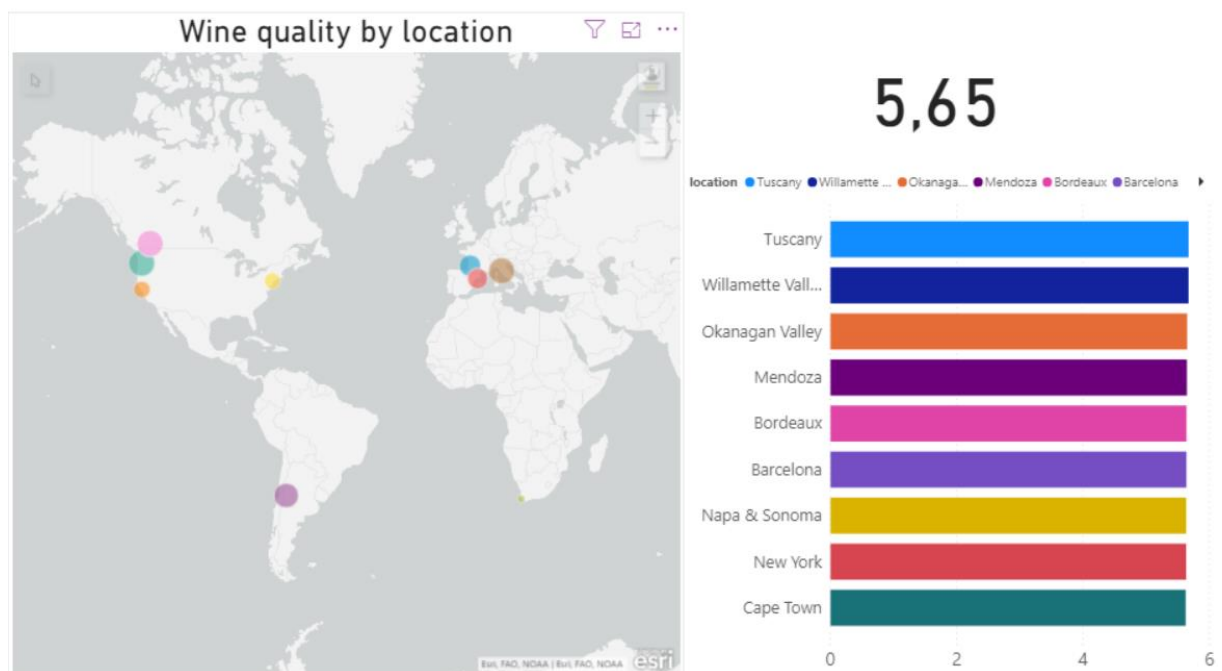


Figure 5.4 Power BI with ArcGIS example dashboard

6. CONCLUSIONS

The phrase "data is the new oil" has become a common refrain and Big data is not just a futuristic terminology anymore. Consequently, we need effective and reliable techniques to get the most value out of our data. Combining technologies like artificial intelligence, machine learning, blockchain, augmented reality, IOT and many more to our classical databases is necessary for a modern enterprise in order to create new products and build an advantage on rapidly evolving market.

In the thesis an approach to combine tools available in Microsoft Azure cloud service was presented, as Microsoft enables simple, scalable, and reliable data pipelines in Azure Data Factory using a serverless, hybrid, parallel data movement service to create new modern ETL workflows in an intuitive working environment. As the final project a fusion of ETL, machine learning and visualization dashboard were shown as output in automating work with data topics.

In summary, combining data engineering and data analysis with AI algorithms in automatized, scaled cloud environment is assuredly one of the most important trends for the near future.

7. LITERATURE

Internet pages:

1. <https://www.socialmediatoday.com/news/how-much-data-is-generated-every-minute-infographic-1/525692/>
2. <https://www.oracle.com/big-data/guide/what-is-big-data.html>
3. <https://mangosoft.tech/blog/top-big-data-technologies-trends-in-2019/>
4. <https://archive.ics.uci.edu/ml/datasets/Wine+Quality>
5. <https://spark.apache.org/docs/2.2.0/ml-classification-regression.html#random-forests>
6. <https://towardsdatascience.com/random-forest-and-its-implementation-71824ced454f>
7. <https://docs.microsoft.com/en-us/azure/data-factory/introduction>
8. <https://app.pluralsight.com/library/courses/microsoft-azure-data-integrating/table-of-contents>
9. <https://docs.microsoft.com/en-us/azure/azure-databricks/what-is-azure-databricks>
10. <https://docs.microsoft.com/en-us/azure/sql-database/sql-database-technical-overview>

8. LIST OF FIGURES

Figure 1.1 Quality variable distribution	3
Figure 2.1 Red wine metrics data generator	4
Figure 3.1 Random forest classifier schema [6]	5
Figure 3.2 Random forest classifier with hyperparameters tuning	6
Figure 3.3. Difference between predicted quality label and original quality label	6
Figure 4.1 Raw JSON files stored on Azure Data Lake Gen2 Blob Storage	8
Figure 4.2. Moving JSON files between containers in Azure Data Factory pipeline	9
Figure 4.3 Databricks cluster configuration	9
Figure 4.4. Raw JSON files loaded and transformed in Databricks notebook	10
Figure 4.5 Wine quality predictions	10
Figure 4.6 Final Data Frame output	11
Figure 4.7 Azure Data Factory Monitor panel	11
Figure 5.1 Azure SQL Database example query	12
Figure 5.2 Power BI example dashboard	12
Figure 5.3 Power BI selection example dashboard	13
Figure 5.4 Power BI with ArcGIS example dashboard	13

9. LIST OF TABLES

Table 1.1 Wine dataset sample	3
-------------------------------------	---

10. APPENDIX

1. WINE QUALITY DATASET DATA PIPELINE DIAGRAM

