Demonstrating a Machine Learning model for redictive Maintenance on Microsoft Azure -new release

An UberCloud Experiment

Written by

Joseph Pareti - Artificial Intelligence consultant
cell +49 1520 1600 209
email: joepareti54@gmail.com

With Support From

UberCloud Case Study

http://www.TheUberCloud.com

This document is the continuation of case study 212, and it's about a new release of the software by Microsoft. The functionality of the model, and predictive precision and accuracy are the same as in the first release, however the current implementation is based on Azure Machine Learning Services and databricks that significantly reduces time-to-solution, while simplifying the end-user's administration task, since the entire application runs in Azure. Therefore, the desk-top front end, "Azure Machine Learning Workbench" which was a required component in the first release, has been removed.

MEET THE TEAM
End-User: TBD
Software Provider: Open Source
Resource Provider: Microsoft Azure
AI Consultant: Joseph Pareti

UberCloud: Wolfgang Gentsch
Microsoft Support: *Yassine Khelifi*

USE CASE

The Predictive Maintenance model described in this report is open source and can be applied to different equipment types for which telemetry data and maintenance data records are available. **The implementation described herein assumes an Azure cloud subscription and some operating knowledge on Azure.**

4 machine types are considered, each machine has 4 components, and there is *telemetry* data available on voltage, vibration, speed, and pressure, as well as maintenance records (indicating when last a component was replaced on what machine), error logs (not necessarily implying failure), machine characteristics, and how long each machine has been in service. The model is built in 4 stages each of which is implemented in a Jupyter notebook running Python version3:

1. Data ingestion
2. Feature engineering
3. ML model
4. Operationalization

Notebook 1, Data ingestion is about accessing the datasets from blob storage, cleaning the data, and storing the data as a SPARK dataframe in cluster for further processing by the next notebooks.

Notebook 2, Feature engineering loads the data sets created in the **Data Ingestion** notebook and combines them to create a single data set of features (variables) that can be used to infer a machine health condition over time.

The goal is to generate a single record for each time unit within each asset. The record includes features and labels to be fed into the machine learning algorithm.

Predictive maintenance takes historical data, marked with a timestamp, to predict current health of a component and the probability of failure within some future window of time. These problems can be characterized as a *classification method* involving *time series* data. Time series, since we want to use historical observations to predict what will happen in the future. Classification, because we classify the future as having a probability of failure.

Notebook 3, The ML model uses the labeled feature data set constructed in notebook 2, it loads the data and splits it into a training and test data set. We then build a machine learning model (a decision tree classifier or a random forest classifier) to predict when different components within our machine population will fail.

Two different classification model approaches are available in this notebook:

- **Decision Tree Classifier**: Decision trees and their ensembles are popular methods for the machine learning tasks of classification and regression. Decision trees are widely used since they are easy to interpret, handle categorical features, extend to the multiclass classification setting, do not require feature scaling, and are able to capture non-linearities and feature interactions.
- **Random Forest Classifier**: A random forest is an ensemble of decision trees. Random forests combine many decision trees in order to reduce the risk of overfitting. Tree ensemble algorithms such as random forests and boosting are among the top performers for classification and regression tasks.

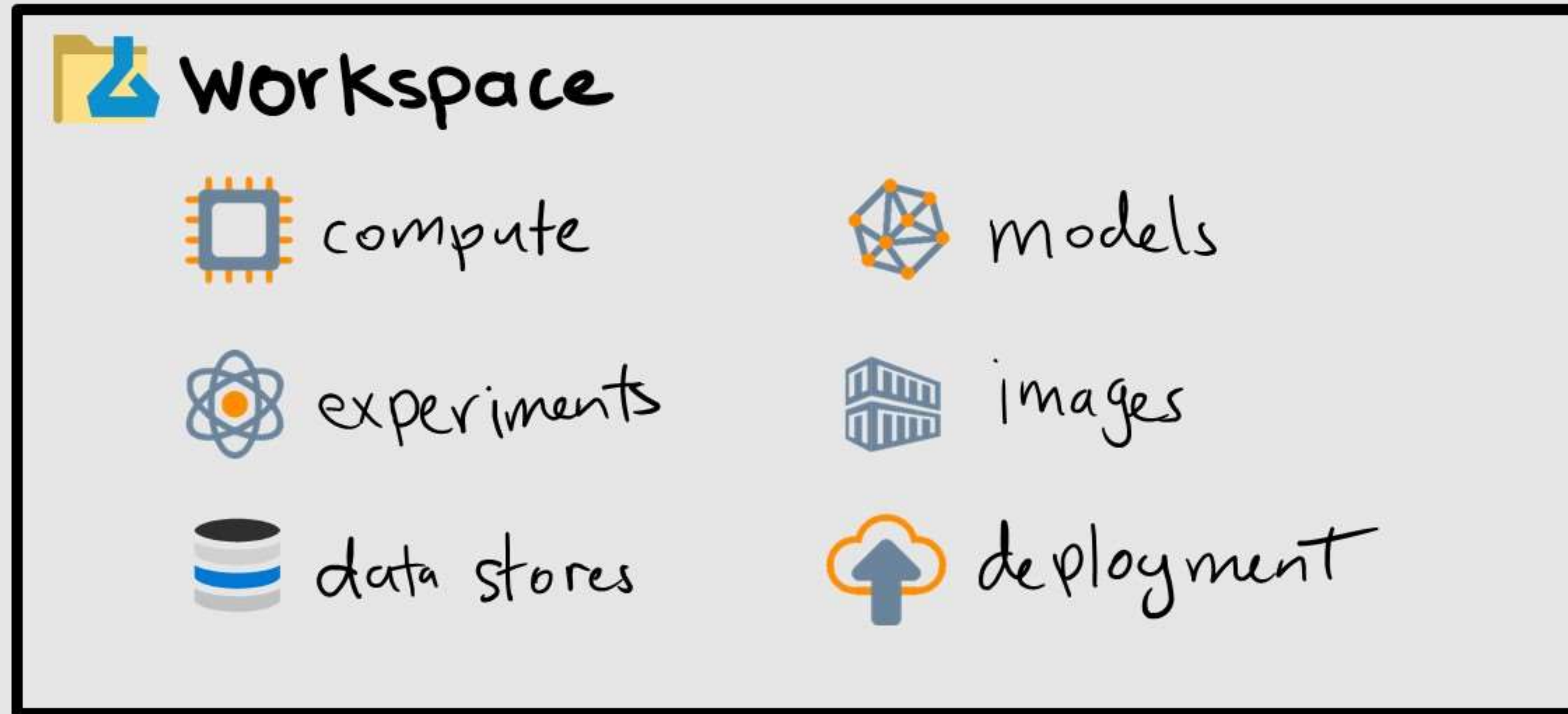
Notebook 4, Operationalization is about loading the model from the `Code/3_model_building.ipynb` Jupyter notebook and the labeled feature data set constructed in the `Code/2_feature_engineering.ipynb` notebook in order to build the model deployment artifacts.

The notebook is used to deploy and operationalize the model and is built on the Azure Machine Learning service SDK.

There are 4 appendices at the end of this report, each of which containing the code and computation output from the 4 notebooks listed above.

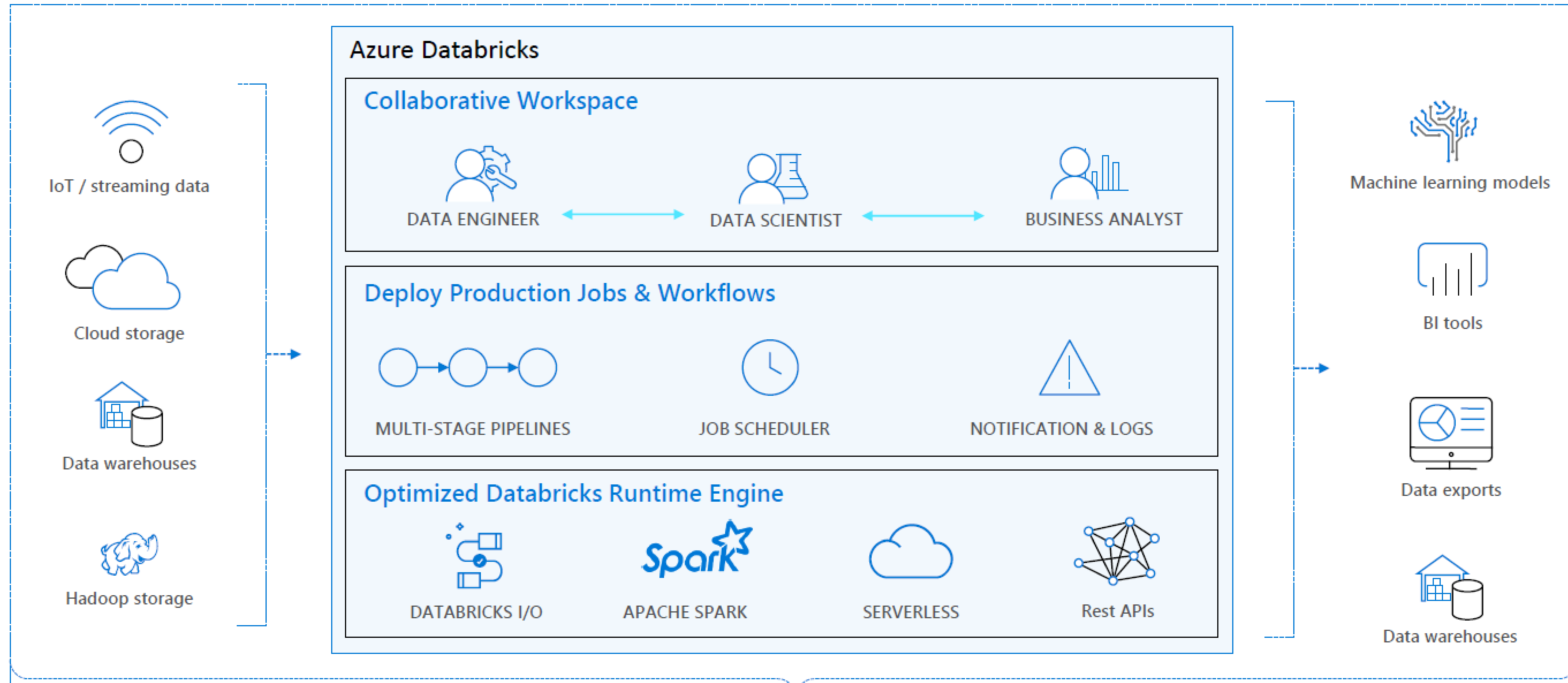
AZURE Machine Learning SDK on Azure Databricks

The ML SDK and databricks are options to implement custom AI which require custom data and model training.
The Azure ML SDK defines a workspace that contains compute and storage resources, as well as models, experiments (i.e. all attempts with different parameters), and deployment services such as docker images



Databricks is an architecture for SPARK environments that provides out-of-the-box support for common interfaces and supports at-scale ML deployment

Azure Databricks



My SYSTEM ARCHITECTURE

The following has been implemented:

- A workspace in Azure
- A development environment for ML

- An Azure Databricks cluster deployed with the following configuration:
  - Databricks Runtime version: (latest stable release)(Scala 2.11)
  - Python version: 3
  - Driver/Worker type: Standard_DS13_v2
  - Python libraries installed:

    `ipython==2.2.0, pyOpenSSL==16.0.0, psutil, azureml-sdk[databricks], cryptography==1.5`

In my implementation the following was set:

- Azure workspace
- Databricks workspace
- Databricks cluster, including 2 to 8 nodes that are DS3_v2 virtual machines
- Libraries on databricks cluster

The screen dumps below provide additional detail to the above.

Home > jp-predictive-maintenance-v2-ws

jp-predictive-maintenance-v2-ws

Machine Learning service workspace

Search (Ctrl+/)

Overview

Activity log

Access control (IAM)

Tags

Diagnose and solve problems

Settings

Locks

Automation script

Properties

Application

Experiments

Pipelines

Compute

Models

Images

Deployments

Delete

jp-resource-group-PAID

Location

West Europe

Subscription

Pay-As-You-Go-JP-ML

Subscription ID

38620b93-e186-4292-b9d1-4159d7be1b28

jppredictivema2861555563

Registry

jppredictivema8201162636

Key Vault

jppredictivema9087192873

Application Insights

jppredictivema6594894108

Getting Started

Explore your Azure Machine Learning service workspace

Explore your Machine Learning service workspace to run and track experiments, compare model performance, and deploy models.

View Documentation

Learn how to use Azure Machine Learning.

View Forum

Join the discussion of Azure Machine Learning.

View more samples at GitHub

Get inspired by a large collection of machine learning examples.

Get Started in Azure Notebooks

Quickly get started with the SDK and run sample experiments in Azure Notebooks.

Figure xxx Azure Machine Learning Workspace; use the forum to ask Microsoft product managers

Dashboard > databricks-pm2-jp

databricks-pm2-jp
Azure Databricks Service

Search (Ctrl+/)

Overview
Activity log
Access control (IAM)
Tags

Settings
Virtual Network Peerings
Locks
Automation script

Support + troubleshooting
New support request

Delete

Resource group (change)
jp-resource-group-PAID

Subscription (change)
Pay-As-You-Go-JP-ML

Subscription ID
38620b93-e186-4292-b9d1-4159d7be1b28

Managed Resource Group
databricks-rg-databricks-pm2-jp-pgji4fyaskbca

URL
https://westeurope.azuredatabricks.net

Pricing Tier
standard

Launch Workspace

Documentation
Getting Started
Import Data from File
Import Data from Azure Storage

Figure xxx databricks Workspace

Clusters / databricks-cluster-jp

Configuration | Notebooks (0) | Libraries (0) | Event Log | Spark UI | Driver Logs | Spark Cluster UI - Master

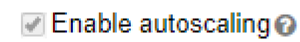
Standard

Databricks Runtime Version

5.2 (includes Apache Spark 2.4.0, Scala 2.11)

3

Autopilot Options



Worker Type		Min Workers	Max Workers
Standard_DS3_v2	14.0 GB Memory, 4 Cores, 0.75 DBU	2	8

Driver Type

Standard_DS3_v2   14.0 GB Memory, 4 Cores, 0.75 DBU

Advanced Options

Figure xxx databricks cluster; set the terminate flag to avoid being charged after the job is done; enable autoscaling to allow databricks to grow or shrink according to job requirement

databricks

View Azure Databricks documentation

Databricks IO Cache
Business Intelligence Tools
Advanced Features
Security
FAQ and Best Practices Guides

Administration Guide
REST API
Release Notes
Databricks Delta Guide
SQL Guide
Spark R Guide
DataFrames and Datasets
Data Sources
Structured Streaming Guide
Machine Learning
Training and FAQ
MLflow Guide
Deep Learning Guide
Graph Analysis Guide
Genomics Guide

Updated Mar 05, 2019

- Create a Workspace library
- View Workspace library details
- Move a Workspace library
- Delete a Workspace library

Create a Workspace library

1. Right-click the Workspace folder where you want to store the library.
2. Select **Create > Library**.

Databricks Guide | 00 Welcome to Databri...
shared
Shared
users
Users
Create | Notebook
Clone | Library
Import | Folder
Export

The Create Library dialog displays.

Create Library
Library Source
Upload | DBFS/S3 | PyPI | Maven | CRAN
Library Type
Jar | Python Egg | Python Whl
Library Name
Optional

Microsoft Azure
PORTAL
joepareti54@gmail.com

Azure Databricks
Home
Workspace
Recents
Data
Clusters
Jobs
Search

Create Library
Library Source
Upload | DBFS | PyPI | Maven | CRAN
Repository
Optional
Package
PyPI package (simplejson or simplejson==3.8.0)
Create | Cancel

Figure xxx Libraries on databricks cluster. Left: User's guide, Right: my configuration

databricks View Azure Databricks documentation

Search

Databricks

Getting Started Guide

User Guide

Supported Browsers

Clusters

Workspace

Notebooks

Visualizations

Accessing Data

Databases and Tables

Libraries

Jobs

Secrets

Developer Tools

Databricks File System - DBFS

Databricks IO Cache

Business Intelligence Tools

Advanced Features

Security

PyPI package

1. In the Library Source button list, select **PyPI**.
2. In the Repository field, optionally enter a PyPI repository URL.
3. Enter a PyPI package name. To install a specific version of a library use this format for the library: `<library>==<version>`. For example, `scikit-learn==0.19.1`.
4. Click **Create**. The library status screen displays.
5. Optionally install the library on a cluster.

Maven or Spark package

1. In the Library Source button list, select **Maven**.
2. In the Repository field, optionally enter a Maven repository URL.

Note

Internal Maven repositories are not supported.

Microsoft Azure PORTAL joepareti54@gmail.com

Azure Databricks

Home

Workspace

Recents

Data

Clusters

Jobs

Search

ipython==2.2.0, pyOpenSSL==16.0.0, psutil, azureml-sdk...

PyPI rules

ipython==2.2.0
pyOpenSSL==16.0.0
psutil
azureml-sdk[databricks]
cryptography==1.5

Status on running clusters

Install automatically on all clusters

Uninstall Install

Status	Cluster Name	Message
Installed	jp-databricks-cluster-4...	

Figure xxx PyPI package on databricks cluster. Left: User's guide, Right: my configuration

RESULTS

The result of this study is an enhanced demo version. If you have an Azure subscription, you can build it yourself starting here.
If you need a step-by-step tutorial on how to deploy Azure ML SDK and databricks, you can use this youtube video.

PERFORMANCE BENCHMARKING

The previous release was tested on a single DSVM with 4 vCPUs and 16 GB RAM;

The current release is on a databricks cluster with 2 to 8 nodes that are Standard_DS13_v2

Release	Notebook 2 Time-to-solution	Notebook 3 Time-to-solution
previous	71 minutes	10 minutes
current	27 minutes	21 minutes

The above figures only give a qualitative indication because I did not use the “run all” feature as recommended in the comments in the code (see appendix).

CONCLUSION & RECOMMENDATIONS

Compared to the previous release, the current one provides:

- Same accuracy, recall, F1
- Shorter time to solution
- Easier administration

If you are interested to implement predictive maintenance using ML, I offer a discovery workshop and Scope of Work service. Please approach me at joepareti54@gmail.com

---

Case Study Authors – Joseph Pareti and Yassine Khelifi

Appendix: data ingestion

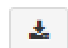
databricks jp (Python)

```
import os
import time
import matplotlib.pyplot as plt

import seaborn as sns
import pandas as pd
plt.style.use('ggplot')

# Time the notebook execution.
# This will only make sense if you "Run all cells"

tic = time.time()

parquet_files_names = {'machines':'machines_files.parquet','maint':'maint_files.parquet',
         'errors': 'errors_files.parquet','telemetry':'telemetry_files.parquet',
        'failures':'failure_files.parquet'}

csv_files_names = {'machines':'machines.csv','maint':'maint.csv',
         'errors': 'errors.csv','telemetry':'telemetry.csv',
        'failures':'failures.csv'}

target_dir = "dbfs:/dataset/"
storage_path = "wasb://predmaintenance@amlgitsamples.blob.core.windows.net/data/"

#dbutils.fs.rm(target_dir, recurse = True)
dbutils.fs.mkdirs(target_dir)
dbutils.fs.cp(storage_path, target_dir, recurse=True)
display(dbutils.fs.ls(target_dir))
```

path	name	size
dbfs:/dataset/errors.csv	errors.csv	405509
dbfs:/dataset/failures.csv	failures.csv	221172
dbfs:/dataset/machines.csv	machines.csv	16435
dbfs:/dataset/maint.csv	maint.csv	1039435
dbfs:/dataset/telemetry.csv	telemetry.csv	809939545

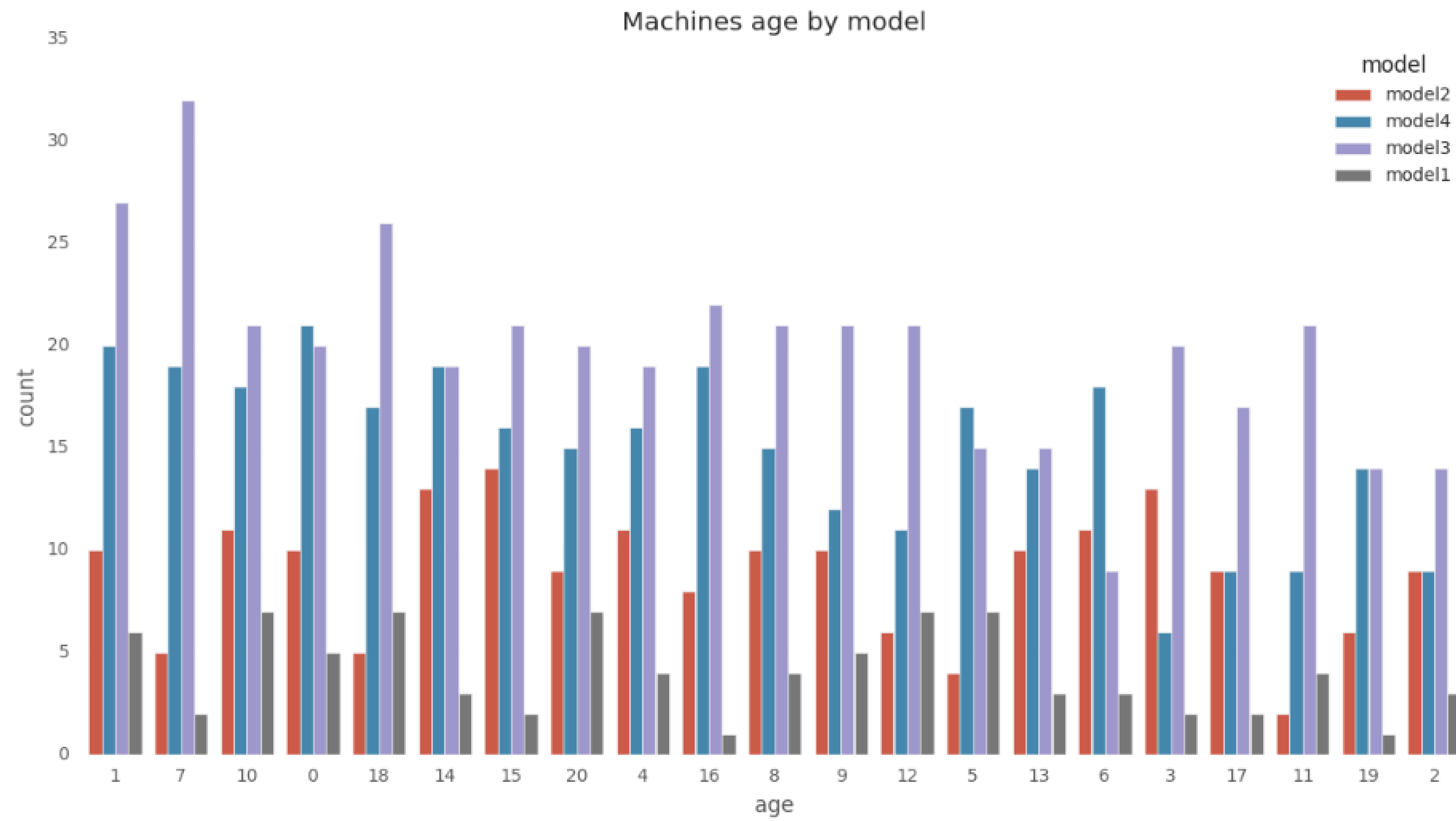
```
machines =  spark.read.format("csv") \
  .option("header", "true") \
  .option("inferSchema", "true") \
  .load(os.path.join(storage_path,csv_files_names['machines']))
display(machines.take(5))
```

machineID	model	age
1	model2	18
2	model4	7
3	model3	8
4	model3	7
5	model2	2



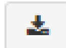
databricks jp (Python)

```
machines_df = machines.toPandas()
plt.figure(figsize=(14, 7))
ax = sns.countplot(x="age", hue="model", data=machines_df,
                   order = machines_df.age.value_counts().index).set_title("Machines age by model")
del machines_df
display(ax.figure)
```



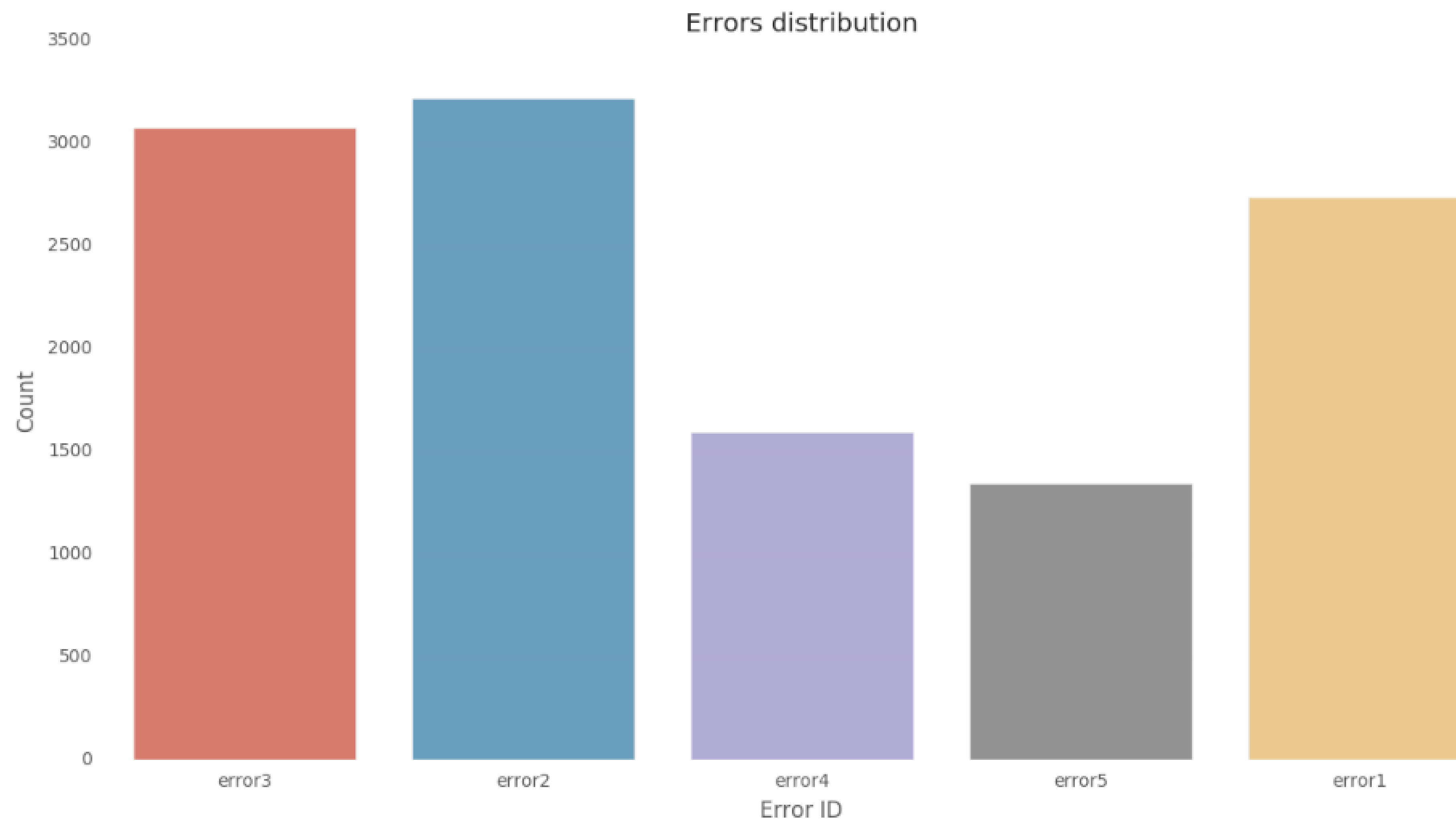
```
errors =  spark.read.format("csv") \
  .option("header", "true") \
  .option("inferSchema", "true") \
  .load(os.path.join(storage_path,csv_files_names['errors']))
display(errors.take(5))
```

datetime	machineID	errorID
2015-01-06T03:00:00.000+0000	1	error3
2015-02-03T06:00:00.000+0000	1	error4
2015-02-21T11:00:00.000+0000	1	error1
2015-02-21T16:00:00.000+0000	1	error2
2015-03-20T06:00:00.000+0000	1	error1



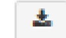
databricks jp (Python)

```
fig, ax = plt.subplots(figsize=(14,7))
errors_count = (spark.createDataFrame(errors.groupBy('errorID').count().collect())
              .toPandas())
sns.barplot(errors_count['errorID'], errors_count['count'], alpha=0.8).set_title('Errors distribution')
ax.set_ylabel("Count")
ax.set_xlabel("Error ID")
display(ax.figure)
```

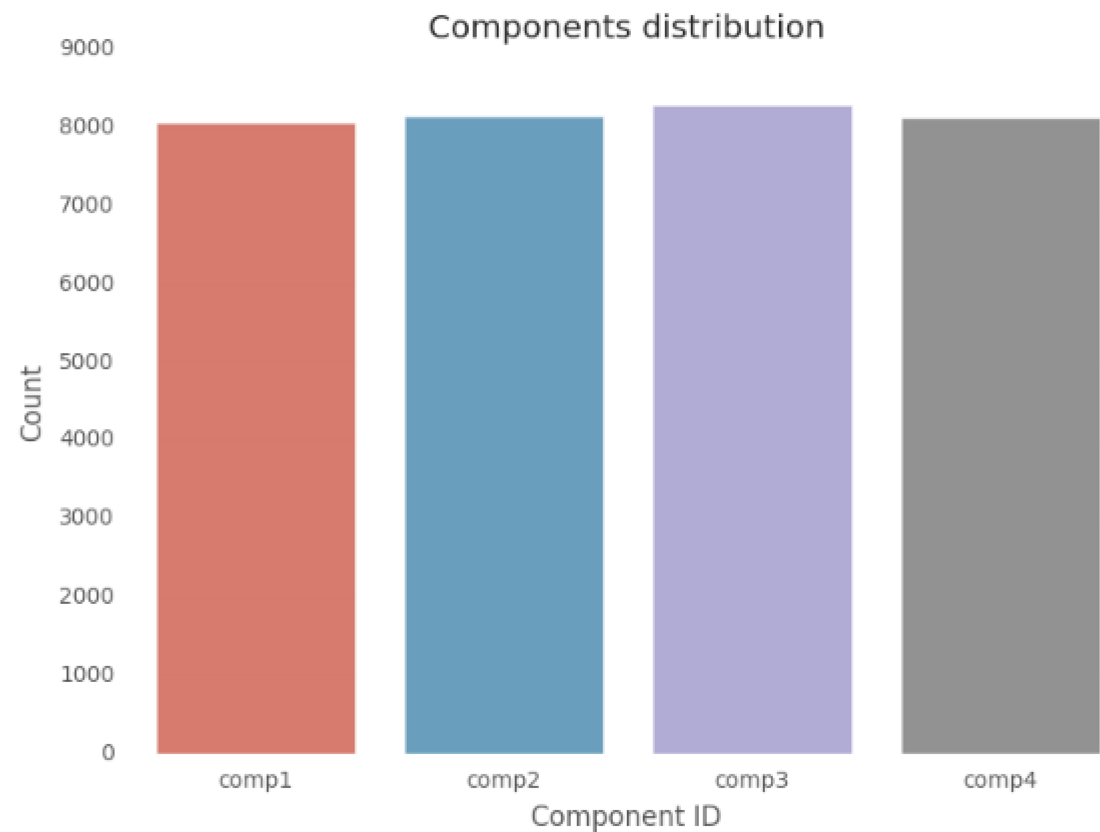


```
maint=  spark.read.format("csv") \
  .option("header", "true") \
  .option("inferSchema", "true") \
  .load(os.path.join(storage_path,csv_files_names['maint']))
display(maint.take(5))
```

datetime	machineID	comp
2014-07-01T06:00:00.000+0000	1	comp4
2014-09-14T06:00:00.000+0000	1	comp1
2014-09-14T06:00:00.000+0000	1	comp2
2014-11-13T06:00:00.000+0000	1	comp3
2015-01-05T06:00:00.000+0000	1	comp1

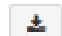


```
fig, ax = plt.subplots()
components_count = (spark.createDataFrame(maint.groupBy('comp').count().collect())
              .toPandas())
sns.barplot(components_count['comp'], components_count['count'], alpha=0.8).set_title('Components distribution')
ax.set_ylabel("Count")
ax.set_xlabel("Component ID")
display(ax.figure)
```

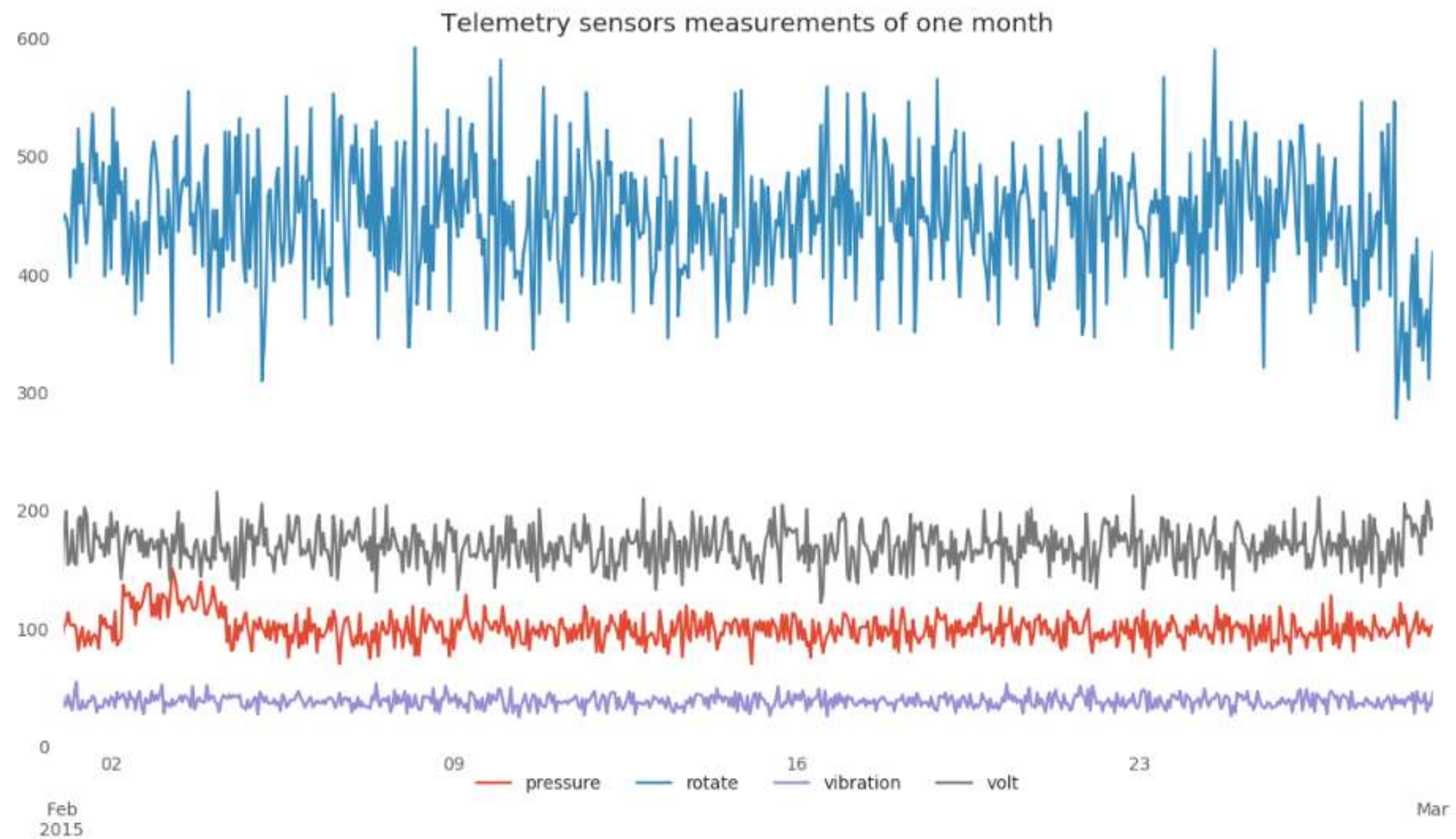


```
telemetry =  spark.read.format("csv") \
  .option("header", "true") \
  .option("inferSchema", "true") \
  .load(os.path.join(storage_path, csv_files_names['telemetry']))

# handle missing values
# define groups of features
telemetry_cols = telemetry.columns

features_datetime = ['datetime']
features_categorical = ['machineID']
features_numeric = list(set(telemetry_cols) - set(features_datetime) - set(features_categorical))

# Replace numeric NA with 0
telemetry = telemetry.fillna(0, subset = features_numeric)

# Replace categorical NA with 'Unknown'
telemetry = telemetry.fillna("Unknown", subset = features_categorical)

# Counts...
print(telemetry.count())

# Examine 10 rows of data.
display(telemetry.take(10))
```

datetime	machineID	volt	rotate	pressure	vibration
2015-01-01T06:00:00.000+0000	1	151.919998705647	530.813577555042	101.788175260076	49.6040134898504
2015-01-01T07:00:00.000+0000	1	174.522001096471	535.523532319384	113.256009499254	41.5159054753218
2015-01-01T08:00:00.000+0000	1	146.912821646066	456.080746005808	107.786964633461	42.0996936545816
2015-01-01T09:00:00.000+0000	1	179.530560852404	503.469990485512	108.283817221771	37.8477274946112
2015-01-01T10:00:00.000+0000	1	180.544276621327	371.600611295334	107.55330679883	41.4678800376109
2015-01-01T11:00:00.000+0000	1	141.41175703074	530.857266087542	87.6140012779218	44.9858461978707
2015-01-01T12:00:00.000+0000	1	184.083821743344	450.2275288129	87.6973797069792	30.8312627133489
2015-01-01T13:00:00.000+0000	1	166.632618417563	486.466837788584	108.067733800301	50.3800539242367
2015-01-01T14:00:00.000+0000	1	159.892748369181	488.968697483274	102.131884360457	43.661296546187

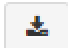


```
plt_data = telemetry.filter(telemetry.machineID == 1).toPandas()

# format datetime field which comes in as string
plt_data['datetime'] = pd.to_datetime(plt_data['datetime'], format="%Y-%m-%d %H:%M:%S")


# Quick plot to show structure
plot_df = plt_data.loc[(plt_data['datetime'] >= pd.to_datetime('2015-02-01')) &
                       (plt_data['datetime'] <= pd.to_datetime('2015-03-01'))]

plt_data = pd.melt(plot_df, id_vars=['datetime', 'machineID'])

fig, ax = plt.subplots(figsize=(14,7))
plt_data.groupby(['datetime','variable']).mean()['value']\
    .unstack().plot(ax=ax, title = 'Telemetry sensors measurements of one month')
ax.legend(loc='center', bbox_to_anchor=(0.5, -0.05),ncol= 4,
         fancybox=True, shadow=True)
display(fig)
```

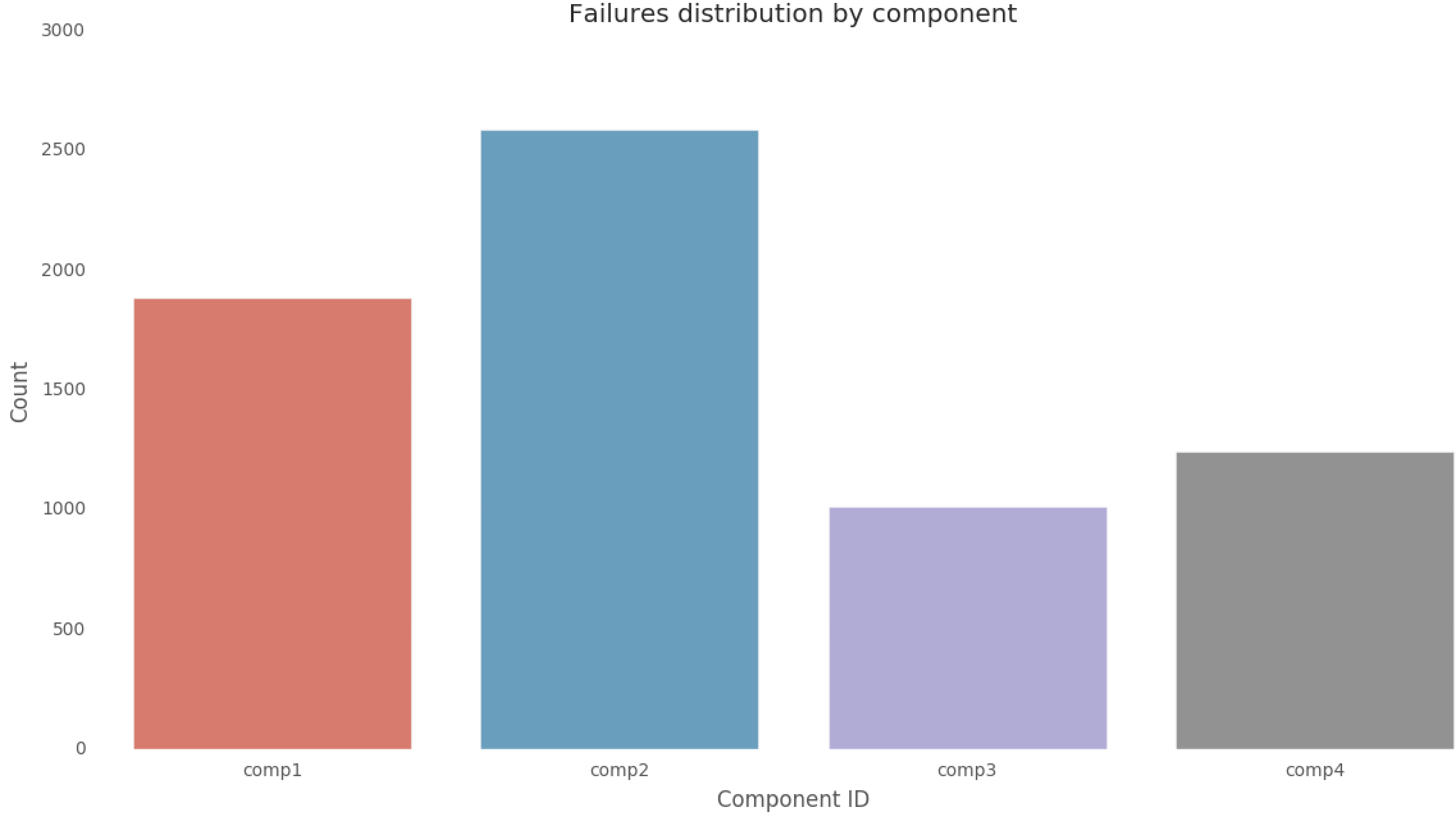


```
failures =  spark.read.format("csv") \
  .option("header", "true") \
  .option("inferSchema", "true") \
  .load(os.path.join(storage_path,csv_files_names['failures']))
display(failures.take(5))
```

datetime	machineID	failure
2015-02-04T06:00:00.000+0000	1	comp3
2015-03-21T06:00:00.000+0000	1	comp1
2015-04-05T06:00:00.000+0000	1	comp4
2015-05-05T06:00:00.000+0000	1	comp3
2015-05-20T06:00:00.000+0000	1	comp2



```
fig, ax = plt.subplots(figsize=(14,7))
failures_count =
(spark.createDataFrame(failures.groupBy('failure').count().collect())
               .toPandas())
sns.barplot(failures_count['failure'], failures_count['count'],
alpha=0.8).set_title('Failures distribution by component')
ax.set_ylabel("Count")
ax.set_xlabel("Component ID")
display(ax.figure)
```


Failures distribution by component
3000
2500
2000
1500
1000
500
0
Count
comp1
comp2
comp3
comp4
Component ID

```
machines.write.mode('overwrite').parquet(os.path.join(target_dir,parquet_files_names['machines']))
errors.write.mode('overwrite').parquet(os.path.join(target_dir,parquet_files_names['errors']))
maint.write.mode('overwrite').parquet(os.path.join(target_dir,parquet_files_names['maint']))
telemetry.write.mode('overwrite').parquet(os.path.join(target_dir,parquet_files_names['telemetry']))
failures.write.mode('overwrite').parquet(os.path.join(target_dir,parquet_files_names['failures']))

for key, val in csv_files_names.items():
  dbutils.fs.rm(os.path.join(target_dir, csv_files_names[key]))

toc = time.time()
print("Full run took %.2f minutes" % ((toc - tic)/60))
```

```
Full run took 13.98 minutes
```

Appendix: feature engineering

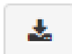
2_features_engineering (Python)

```
## Setup our environment by importing required libraries
import time
import os
import glob


# For creating some preliminary EDA plots.
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
plt.style.use('ggplot')

import datetime

import pyspark.sql.functions as F
from pyspark.sql.functions import (col, unix_timestamp,
                                   round, datediff, to_date)
from pyspark.sql.window import Window
from pyspark.sql.types import DoubleType

from pyspark.ml import Pipeline
from pyspark.ml.feature import OneHotEncoder,StringIndexer
from pyspark.sql import SparkSession

# Time the notebook execution.
# This will only make sense if you "Run all cells"
tic = time.time()
```

```
parquet_files_names = {'machines':'machines_files.parquet','maint':'maint_files.parquet',
         'errors': 'errors_files.parquet','telemetry':'telemetry_files.parquet',
        'failures':'failure_files.parquet', 'features':'featureengineering_files.parquet'}


target_dir = "dbfs:/dataset/"
```

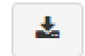
```
# Read in the data
machines = spark.read.parquet(os.path.join(target_dir, parquet_files_names['machines']))

print(machines.count())
display(machines.limit(5))
```

machineID	model	age
1	model2	18
2	model4	7
3	model3	8
4	model3	7
5	model2	2



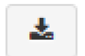
```
errors = spark.read.parquet(os.path.join(target_dir, parquet_files_names['errors']))

print(errors.count())
display(errors.limit(5))
```

datetime	machineID	errorID
2015-01-06T03:00:00.000+0000	1	error3
2015-02-03T06:00:00.000+0000	1	error4
2015-02-21T11:00:00.000+0000	1	error1
2015-02-21T16:00:00.000+0000	1	error2
2015-03-20T06:00:00.000+0000	1	error1



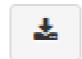
```
maint = spark.read.parquet(os.path.join(target_dir, parquet_files_names['maint']))

print(maint.count())
display(maint.limit(5))
```

datetime	machineID	comp
2014-07-01T06:00:00.000+0000	1	comp4
2014-09-14T06:00:00.000+0000	1	comp1
2014-09-14T06:00:00.000+0000	1	comp2
2014-11-13T06:00:00.000+0000	1	comp3
2015-01-05T06:00:00.000+0000	1	comp1



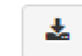
```
telemetry = spark.read.parquet(os.path.join(target_dir, parquet_files_names['telemetry']))

print(telemetry.count())
display(telemetry.limit(5))
```

datetime	machineID	volt	rotate	pressure	vibration
2015-01-01T06:00:00.000+0000	1	151.919998705647	530.813577555042	101.788175260076	49.6040134898504
2015-01-01T07:00:00.000+0000	1	174.522001096471	535.523532319384	113.256009499254	41.5159054753218
2015-01-01T08:00:00.000+0000	1	146.912821646066	456.080746005808	107.786964633461	42.0996936545816
2015-01-01T09:00:00.000+0000	1	179.530560852404	503.469990485512	108.283817221771	37.8477274946112
2015-01-01T10:00:00.000+0000	1	180.544276621327	371.600611295334	107.55330679883	41.4678800376109



```
failures = spark.read.parquet(os.path.join(target_dir, parquet_files_names['failures']))

print(failures.count())
display(failures.limit(5))
```

datetime	machineID	failure
2015-02-04T06:00:00.000+0000	1	comp3
2015-03-21T06:00:00.000+0000	1	comp1
2015-04-05T06:00:00.000+0000	1	comp4
2015-05-05T06:00:00.000+0000	1	comp3
2015-05-20T06:00:00.000+0000	1	comp2



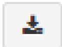
```
# rolling mean and standard deviation
# Temporary storage for rolling means
tel_mean = telemetry

# Which features are we interested in telemetry data set
rolling_features = ['volt','rotate', 'pressure', 'vibration']

# n hours = n * 3600 seconds
time_val = 12 * 3600

# Choose the time_val hour timestamps to align the data
# dt_truncated looks at the column named "datetime" in the current data set.
# remember that Spark is lazy... this doesn't execute until it is in a withColumn statement.
dt_truncated = ((round(unix_timestamp(col("datetime")) / time_val) * time_val).cast("timestamp"))
```

```
# We choose windows for our rolling windows 12hrs, 24 hrs and 36 hrs
lags = [12, 24, 36]

# align the data
for lag_n in lags:
    wSpec = Window.partitionBy('machineID').orderBy('datetime').rowsBetween(1-lag_n, 0)
    for col_name in rolling_features:
        tel_mean = tel_mean.withColumn(col_name+'_rollingmean_'+str(lag_n),
                                       F.avg(col(col_name)).over(wSpec))
        tel_mean = tel_mean.withColumn(col_name+'_rollingstd_'+str(lag_n),
                                       F.stddev(col(col_name)).over(wSpec))

# Calculate lag values...
telemetry_feat = (tel_mean.withColumn("dt_truncated", dt_truncated)
                  .drop('volt', 'rotate', 'pressure', 'vibration')
                  .fillna(0)
                  .groupBy("machineID","dt_truncated")
                  .agg(F.mean('volt_rollingmean_12').alias('volt_rollingmean_12'),
                       F.mean('rotate_rollingmean_12').alias('rotate_rollingmean_12'),
                       F.mean('pressure_rollingmean_12').alias('pressure_rollingmean_12'),
                       F.mean('vibration_rollingmean_12').alias('vibration_rollingmean_12'),
                       F.mean('volt_rollingmean_24').alias('volt_rollingmean_24'),
                       F.mean('rotate_rollingmean_24').alias('rotate_rollingmean_24'),
                       F.mean('pressure_rollingmean_24').alias('pressure_rollingmean_24'),
                       F.mean('vibration_rollingmean_24').alias('vibration_rollingmean_24'),
                       F.mean('volt_rollingmean_36').alias('volt_rollingmean_36'),
                       F.mean('vibration_rollingmean_36').alias('vibration_rollingmean_36'),
                       F.mean('rotate_rollingmean_36').alias('rotate_rollingmean_36'),
                       F.mean('pressure_rollingmean_36').alias('pressure_rollingmean_36'),
                       F.stddev('volt_rollingstd_12').alias('volt_rollingstd_12'),
                       F.stddev('rotate_rollingstd_12').alias('rotate_rollingstd_12'),
                       F.stddev('pressure_rollingstd_12').alias('pressure_rollingstd_12'),
                       F.stddev('vibration_rollingstd_12').alias('vibration_rollingstd_12'),
                       F.stddev('volt_rollingstd_24').alias('volt_rollingstd_24'),
                       F.stddev('rotate_rollingstd_24').alias('rotate_rollingstd_24'),
                       F.stddev('pressure_rollingstd_24').alias('pressure_rollingstd_24'),
                       F.stddev('vibration_rollingstd_24').alias('vibration_rollingstd_24'),
                       F.stddev('volt_rollingstd_36').alias('volt_rollingstd_36'),
                       F.stddev('rotate_rollingstd_36').alias('rotate_rollingstd_36'),
                       F.stddev('pressure_rollingstd_36').alias('pressure_rollingstd_36'),
                       F.stddev('vibration_rollingstd_36').alias('vibration_rollingstd_36'), ))

print(telemetry_feat.count())
telemetry_feat.where((col("machineID") == 1)).limit(5).toPandas()
```

```
731000
Out[9]:
   machineID         dt_truncated  volt_rollingmean_12  rotate_rollingmean_12  \
0          1 2015-05-02 12:00:00           171.712210             451.677582
1          1 2015-06-04 00:00:00           165.451700             468.091340
2          1 2015-06-16 12:00:00           169.018242             451.081749
3          1 2015-10-18 00:00:00           170.253894             457.685553
4          1 2015-10-30 12:00:00           171.302151             458.251722

   pressure_rollingmean_12  vibration_rollingmean_12  volt_rollingmean_24  \
0                95.464429                 39.052661           171.034969
1                95.770983                 50.431712           169.759443
2                99.485598                 42.067891           168.865992
3               102.522720                 38.975971           173.492286
4               100.603054                 47.070461           169.921885

   rotate_rollingmean_24  pressure_rollingmean_24  vibration_rollingmean_24  \
0             455.780008                97.564578                 39.527223
1             461.370378                97.136211                 50.890390
2             442.447741               100.532297                 40.806017
3             458.380635               101.642031                 39.906351
4             457.776886                98.524571                 44.258167
```

```
# create a column for each errorID
error_ind = (errors.groupBy("machineID","datetime","errorID").pivot('errorID')
             .agg(F.count('machineID').alias('dummy')).drop('errorID').fillna(0)
             .groupBy("machineID","datetime")
             .agg(F.sum('error1').alias('error1sum'),
                  F.sum('error2').alias('error2sum'),
                  F.sum('error3').alias('error3sum'),
                  F.sum('error4').alias('error4sum'),
                  F.sum('error5').alias('error5sum')))

# join the telemetry data with errors
error_count = (telemetry.join(error_ind,
                              ((telemetry['machineID'] == error_ind['machineID'])
                               & (telemetry['datetime'] == error_ind['datetime'])), "left")
               .drop('volt', 'rotate', 'pressure', 'vibration')
               .drop(error_ind.machineID).drop(error_ind.datetime)
               .fillna(0))

error_features = ['error1sum','error2sum', 'error3sum', 'error4sum', 'error5sum']

wSpec = Window.partitionBy('machineID').orderBy('datetime').rowsBetween(1-24, 0)
for col_name in error_features:
    # We're only interested in the erros in the previous 24 hours.
    error_count = error_count.withColumn(col_name+'_rollingmean_24',
                                         F.avg(col(col_name)).over(wSpec))

error_feat = (error_count.withColumn("dt_truncated", dt_truncated)
              .drop('error1sum', 'error2sum', 'error3sum', 'error4sum', 'error5sum').fillna(0)
              .groupBy("machineID","dt_truncated")
              .agg(F.mean('error1sum_rollingmean_24').alias('error1sum_rollingmean_24'),
                   F.mean('error2sum_rollingmean_24').alias('error2sum_rollingmean_24'),
                   F.mean('error3sum_rollingmean_24').alias('error3sum_rollingmean_24'),
                   F.mean('error4sum_rollingmean_24').alias('error4sum_rollingmean_24'),
                   F.mean('error5sum_rollingmean_24').alias('error5sum_rollingmean_24')))

print(error_feat.count())
display(error_feat.limit(5))
```

machineID	dt_truncated	error1sum_rollingmean_24	error2sum_rollingmean_24	error3sum_rollingmean_24	error4sum_rollingmean_24	error5sum_rollingmean_24
148	2015-01-31T12:00:00.000+0000	0	0	0	0	0
471	2015-01-03T00:00:00.000+0000	0	0	0	0	0
148	2015-06-13T00:00:00.000+0000	0	0	0	0	0
148	2015-10-06T12:00:00.000+0000	0	0	0	0	0
148	2015-11-26T00:00:00.000+0000	0	0	0	0	0



```
# create a column for each component replacement
maint_replace = (maint.groupBy("machineID","datetime","comp").pivot('comp')
                 .agg(F.count('machineID').alias('dummy')).fillna(0)
                 .groupBy("machineID","datetime")
                 .agg(F.sum('comp1').alias('comp1sum'),
                      F.sum('comp2').alias('comp2sum'),
                      F.sum('comp3').alias('comp3sum'),
                      F.sum('comp4').alias('comp4sum')))

maint_replace = maint_replace.withColumnRenamed('datetime','datetime_maint')

print(maint_replace.count())
maint_replace.limit(5).toPandas()
```

```
25121
Out[11]:
   machineID      datetime_maint  comp1sum  comp2sum  comp3sum  comp4sum
0         25 2015-03-14 06:00:00         0         0         0         1
1        965 2015-09-22 06:00:00         1         0         0         1
2        991 2015-12-28 06:00:00         0         1         0         1
3        880 2015-12-17 06:00:00         0         0         0         1
4        973 2015-10-20 06:00:00         0         1         1         0
```

```
# We want to align the component information on telemetry features timestamps.
telemetry_times = (telemetry_feat.select(telemetry_feat.machineID, telemetry_feat.dt_truncated)
                   .withColumnRenamed('dt_truncated','datetime_tel'))

# Grab component 1 records
maint_comp1 = (maint_replace.where(col("comp1sum") == '1').withColumnRenamed('datetime','datetime_maint')
               .drop('comp2sum', 'comp3sum', 'comp4sum'))

# Within each machine, get the last replacement date for each timepoint
maint_tel_comp1 = (telemetry_times.join(maint_comp1,
                                        ((telemetry_times ['machineID']== maint_comp1['machineID'])
                                         & (telemetry_times ['datetime_tel'] > maint_comp1['datetime_maint'])
                                         & ( maint_comp1['comp1sum'] == '1')))
                   .drop(maint_comp1.machineID))

# Calculate the number of days between replacements
comp1 = (maint_tel_comp1.withColumn("sincelastcomp1",
                                    datediff(maint_tel_comp1.datetime_tel, maint_tel_comp1.datetime_maint))
         .drop(maint_tel_comp1.datetime_maint).drop(maint_tel_comp1.comp1sum))

print(comp1.count())
comp1.filter(comp1.machineID == '625').orderBy(comp1.datetime_tel).limit(5).toPandas()
```

```
3254437
Out[12]:
   machineID        datetime_tel  sincelastcomp1
0        625 2015-01-01 12:00:00              94
1        625 2015-01-02 00:00:00              95
2        625 2015-01-02 12:00:00              95
3        625 2015-01-03 00:00:00              96
4        625 2015-01-03 12:00:00              96
```

```
# Grab component 2 records
maint_comp2 = (maint_replace.where(col("comp2sum") == '1').withColumnRenamed('datetime','datetime_maint')
               .drop('comp1sum', 'comp3sum', 'comp4sum'))

# Within each machine, get the last replacement date for each timepoint
maint_tel_comp2 = (telemetry_times.join(maint_comp2,
                                        ((telemetry_times ['machineID']== maint_comp2['machineID'])
                                         & (telemetry_times ['datetime_tel'] > maint_comp2['datetime_maint'])
                                         & ( maint_comp2['comp2sum'] == '1')))
                   .drop(maint_comp2.machineID))

# Calculate the number of days between replacements
comp2 = (maint_tel_comp2.withColumn("sincelastcomp2",
                                    datediff(maint_tel_comp2.datetime_tel, maint_tel_comp2.datetime_maint))
         .drop(maint_tel_comp2.datetime_maint).drop(maint_tel_comp2.comp2sum))

print(comp2.count())
comp2.filter(comp2.machineID == '625').orderBy(comp2.datetime_tel).limit(5).toPandas()
```

```
3278730
Out[13]:
   machineID        datetime_tel  sincelastcomp2
0        625 2015-01-01 12:00:00              19
1        625 2015-01-02 00:00:00              20
2        625 2015-01-02 12:00:00              20
3        625 2015-01-03 00:00:00              21
4        625 2015-01-03 12:00:00              21
```

```
# Grab component 3 records
maint_comp3 = (maint_replace.where(col("comp3sum") == '1').withColumnRenamed('datetime','datetime_maint')
               .drop('comp1sum', 'comp2sum', 'comp4sum'))

# Within each machine, get the last replacement date for each timepoint
maint_tel_comp3 = (telemetry_times.join(maint_comp3, ((telemetry_times ['machineID']==maint_comp3['machineID'])
                                                       & (telemetry_times ['datetime_tel'] > maint_comp3['datetime_maint'])
                                                       & ( maint_comp3['comp3sum'] == '1')))
                   .drop(maint_comp3.machineID))

# Calculate the number of days between replacements
comp3 = (maint_tel_comp3.withColumn("sincelastcomp3",
                                    datediff(maint_tel_comp3.datetime_tel, maint_tel_comp3.datetime_maint))
         .drop(maint_tel_comp3.datetime_maint).drop(maint_tel_comp3.comp3sum))


print(comp3.count())
comp3.filter(comp3.machineID == '625').orderBy(comp3.datetime_tel).limit(5).toPandas()
```

```
3345413
Out[14]:
   machineID        datetime_tel  sincelastcomp3
0        625 2015-01-01 12:00:00              19
1        625 2015-01-02 00:00:00              20
2        625 2015-01-02 12:00:00              20
3        625 2015-01-03 00:00:00              21
4        625 2015-01-03 12:00:00              21
```

```
# Grab component 4 records
maint_comp4 = (maint_replace.where(col("comp4sum") == '1').withColumnRenamed('datetime','datetime_maint')
               .drop('comp1sum', 'comp2sum', 'comp3sum'))

# Within each machine, get the last replacement date for each timepoint
maint_tel_comp4 = telemetry_times.join(maint_comp4, ((telemetry_times['machineID']==maint_comp4['machineID'])
                                                      & (telemetry_times['datetime_tel'] > maint_comp4['datetime_maint'])
                                                      & (maint_comp4['comp4sum'] == '1'))).drop(maint_comp4.machineID)

# Calculate the number of days between replacements
comp4 = (maint_tel_comp4.withColumn("sincelastcomp4",
                                    datediff(maint_tel_comp4.datetime_tel, maint_tel_comp4.datetime_maint))
         .drop(maint_tel_comp4.datetime_maint).drop(maint_tel_comp4.comp4sum))

print(comp4.count())
comp4.filter(comp4.machineID == '625').orderBy(comp4.datetime_tel).limit(5).toPandas()
```

```
3273666
Out[15]:
   machineID        datetime_tel  sincelastcomp4
0        625 2015-01-01 12:00:00             139
1        625 2015-01-02 00:00:00             140
2        625 2015-01-02 12:00:00             140
3        625 2015-01-03 00:00:00             141
4        625 2015-01-03 12:00:00             141
```

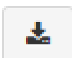
```
# Join component 3 and 4
comp3_4 = (comp3.join(comp4, ((comp3['machineID'] == comp4['machineID'])
                              & (comp3['datetime_tel'] == comp4['datetime_tel'])), "left")
           .drop(comp4.machineID).drop(comp4.datetime_tel))

# Join component 2 to 3 and 4
comp2_3_4 = (comp2.join(comp3_4, ((comp2['machineID'] == comp3_4['machineID'])
                                  & (comp2['datetime_tel'] == comp3_4['datetime_tel'])), "left")
             .drop(comp3_4.machineID).drop(comp3_4.datetime_tel))

# Join component 1 to 2, 3 and 4
comps_feat = (comp1.join(comp2_3_4, ((comp1['machineID'] == comp2_3_4['machineID'])
                                     & (comp1['datetime_tel'] == comp2_3_4['datetime_tel'])), "left")
               .drop(comp2_3_4.machineID).drop(comp2_3_4.datetime_tel)
               .groupBy("machineID", "datetime_tel")
               .agg(F.max('sincelastcomp1').alias('sincelastcomp1'),
                    F.max('sincelastcomp2').alias('sincelastcomp2'),
                    F.max('sincelastcomp3').alias('sincelastcomp3'),
                    F.max('sincelastcomp4').alias('sincelastcomp4'))
               .fillna(0))

# Choose the time_val hour timestamps to align the data
dt_truncated = ((round(unix_timestamp(col("datetime_tel")) / time_val) * time_val).cast("timestamp"))

# Collect data
maint_feat = (comps_feat.withColumn("dt_truncated", dt_truncated)
              .groupBy("machineID","dt_truncated")
              .agg(F.mean('sincelastcomp1').alias('comp1sum'),
                   F.mean('sincelastcomp2').alias('comp2sum'),
                   F.mean('sincelastcomp3').alias('comp3sum'),
                   F.mean('sincelastcomp4').alias('comp4sum')))

print(maint_feat.count())
maint_feat.limit(5).toPandas()
```

```
731000
Out[16]:
```

	machineID	dt_truncated	comp1sum	comp2sum	comp3sum	comp4sum
0	2	2015-05-13 12:00:00	316.0	181.0	211.0	286.0
1	2	2015-05-27 00:00:00	330.0	195.0	225.0	300.0
2	3	2015-01-26 12:00:00	239.0	239.0	179.0	134.0
3	3	2015-02-23 00:00:00	267.0	267.0	207.0	162.0
4	4	2015-02-04 12:00:00	248.0	173.0	218.0	233.0

```
# one hot encoding of the variable model, basically creates a set of dummy boolean variables
catVarNames = ['model']
sIndexers = [StringIndexer(inputCol=x, outputCol=x + '_indexed') for x in catVarNames]
machines_cat = Pipeline(stages=sIndexers).fit(machines).transform(machines)

# one-hot encode
ohEncoders = [OneHotEncoder(inputCol=x + '_indexed', outputCol=x + '_encoded')
              for x in catVarNames]

ohPipelineModel = Pipeline(stages=ohEncoders).fit(machines_cat)
machines_cat = ohPipelineModel.transform(machines_cat)

drop_list = [col_n for col_n in machines_cat.columns if 'indexed' in col_n]

machines_feat = machines_cat.select([column for column in machines_cat.columns if column not in drop_list])

print(machines_feat.count())
display(machines_feat.limit(5))
```

machineID	model	age	model_encoded
1	model2	18	[0,3,[2],[1]]
2	model4	7	[0,3,[1],[1]]
3	model3	8	[0,3,[0],[1]]
4	model3	7	[0,3,[0],[1]]
5	model2	2	[0,3,[2],[1]]

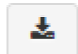


```
# join error features with component maintenance features
error_maint = (error_feat.join(maint_feat,
                               ((error_feat['machineID'] == maint_feat['machineID'])
                                & (error_feat['dt_truncated'] == maint_feat['dt_truncated'])), "left")
               .drop(maint_feat.machineID).drop(maint_feat.dt_truncated))

# now join that with machines features
error_maint_feat = (error_maint.join(machines_feat,
                                     ((error_maint['machineID'] == machines_feat['machineID'])), "left")
                    .drop(machines_feat.machineID))

# Clean up some unecessary columns
error_maint_feat = error_maint_feat.select([c for c in error_maint_feat.columns if c not in
                                            {'error1sum', 'error2sum', 'error3sum', 'error4sum', 'error5sum'}])

# join telemetry with error/maint/machine features to create final feature matrix
final_feat = (telemetry_feat.join(error_maint_feat,
                                  ((telemetry_feat['machineID'] == error_maint_feat['machineID'])
                                   & (telemetry_feat['dt_truncated'] == error_maint_feat['dt_truncated'])), "left")
              .drop(error_maint_feat.machineID).drop(error_maint_feat.dt_truncated))

print(final_feat.count())
final_feat.filter(final_feat.machineID == '625').orderBy(final_feat.dt_truncated).limit(5).toPandas()
```

```
731000
Out[18]:
   machineID        dt_truncated  volt_rollingmean_12  rotate_rollingmean_12  \
0        625 2015-01-01 12:00:00           169.065806             453.899968
1        625 2015-01-02 00:00:00           166.187365             458.219143
2        625 2015-01-02 12:00:00           169.363503             455.143198
3        625 2015-01-03 00:00:00           172.504043             461.494330
4        625 2015-01-03 12:00:00           174.102964             442.074061

   pressure_rollingmean_12  vibration_rollingmean_12  volt_rollingmean_24  \
0                97.857385                 44.903816           169.065806
1                95.377812                 42.361593           166.267437
2                97.519219                 41.000897           167.775434
3               101.483771                 40.299350           170.933773
4                99.900129                 39.624068           173.303503

   rotate_rollingmean_24  pressure_rollingmean_24  vibration_rollingmean_24  \
0             453.899968                97.857385                 44.903816
1             459.462370                96.064038                 42.685859
2             456.681171                96.448515                 41.681245
3             458.318764                99.501495                 40.650123
4             451.784195               100.691950                 39.961709
```

```
dt_truncated = ((round(unix_timestamp(col("datetime")) / time_val) * time_val).cast("timestamp"))

fail_diff = (failures.withColumn("dt_truncated", dt_truncated)
             .drop(failures.datetime))

print(fail_diff.count())
display(fail_diff.limit(5))
```

machineID	failure	dt_truncated
1	comp3	2015-02-04T12:00:00.000+0000
1	comp1	2015-03-21T12:00:00.000+0000
1	comp4	2015-04-05T12:00:00.000+0000
1	comp3	2015-05-05T12:00:00.000+0000
1	comp2	2015-05-20T12:00:00.000+0000



```
# map the failure data to final feature matrix
labeled_features = (final_feat.join(fail_diff,
                                    ((final_feat['machineID'] == fail_diff['machineID'])
                                     & (final_feat['dt_truncated'] == fail_diff['dt_truncated'])), "left")
                    .drop(fail_diff.machineID).drop(fail_diff.dt_truncated)
                    .withColumn('failure', F.when(col('failure') == "comp1", 1.0).otherwise(col('failure')))
                    .withColumn('failure', F.when(col('failure') == "comp2", 2.0).otherwise(col('failure')))
                    .withColumn('failure', F.when(col('failure') == "comp3", 3.0).otherwise(col('failure')))
                    .withColumn('failure', F.when(col('failure') == "comp4", 4.0).otherwise(col('failure'))))

labeled_features = (labeled_features.withColumn("failure",
                                                labeled_features.failure.cast(DoubleType()))
                    .fillna(0))

print(labeled_features.count())
labeled_features.limit(5).toPandas()
```

```
731358
Out[20]:
   machineID        dt_truncated  volt_rollingmean_12  rotate_rollingmean_12  \
0          2 2015-05-13 12:00:00           175.323921             442.849167
1          2 2015-05-27 00:00:00           188.145033             451.339344
2          3 2015-01-26 12:00:00           161.703727             463.396395
3          3 2015-02-23 00:00:00           170.371785             448.846354
4          4 2015-02-04 12:00:00           166.210647             457.021218

   pressure_rollingmean_12  vibration_rollingmean_12  volt_rollingmean_24  \
0                97.735315                 38.853108           171.646680
1                99.481043                 39.607422           190.674288
2               100.549011                 40.526538           165.583780
3               101.029086                 40.852657           170.321834
4                96.402429                 39.004665           168.190004

   rotate_rollingmean_24  pressure_rollingmean_24  vibration_rollingmean_24  \
0             445.629804               101.522361                 40.487794
1             447.989854                98.177035                 39.377754
2             463.213045               100.602100                 40.498309
3             444.113111               100.046844                 41.024862
4             455.525175                97.686885                 40.451908
```

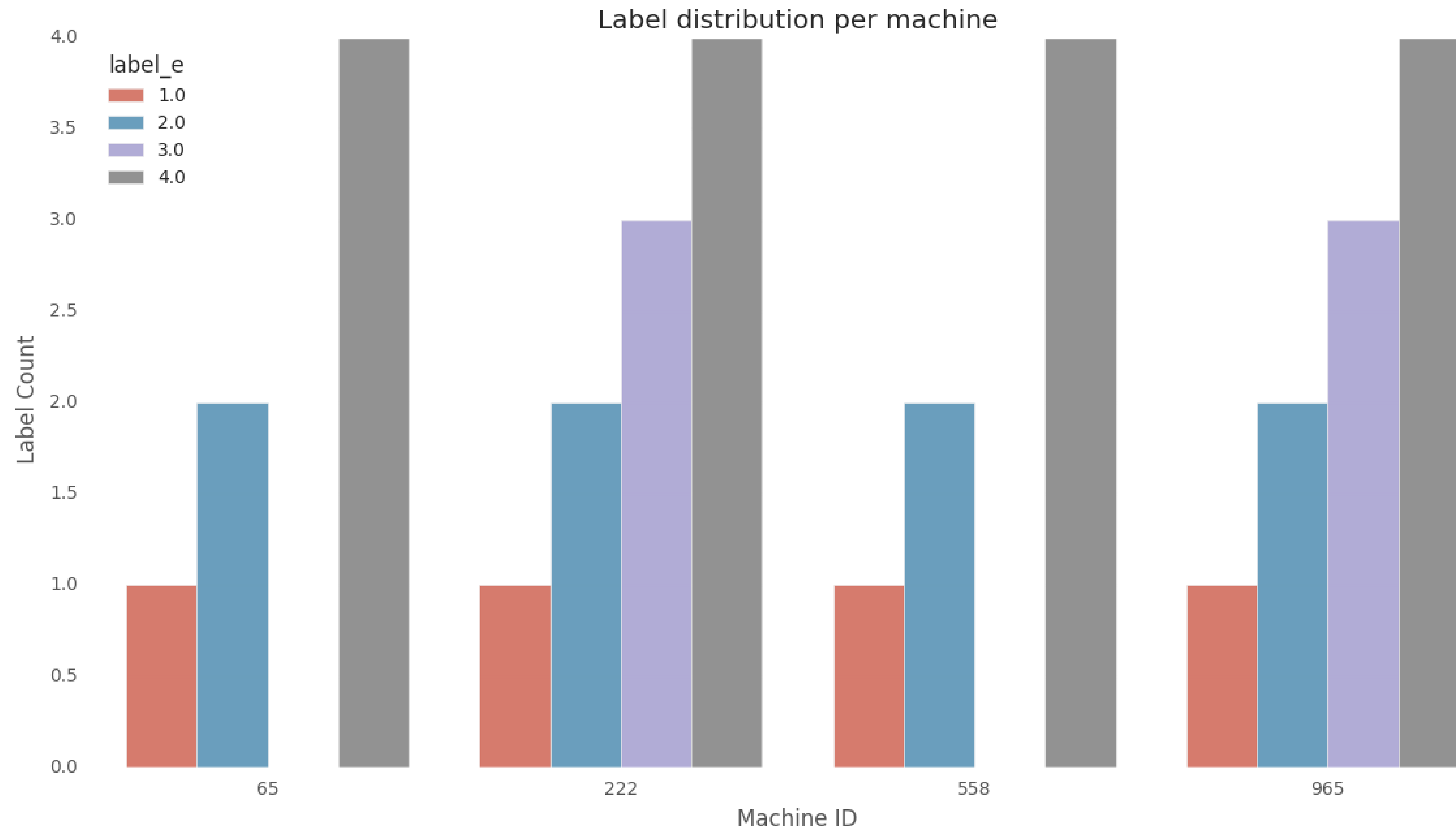
```
# To get the frequency of each component failure
lf_count = labeled_features.groupBy('failure').count().collect()
display(lf_count)
```

failure	count
0	724632
1	1886
4	1241
3	1012
2	2587

```
# lag values to manually backfill label (bfill =7)
my_window = Window.partitionBy('machineID').orderBy(labeled_features.dt_truncated.desc())

# Create the previous 7 days
labeled_features = (labeled_features.withColumn("prev_value1",
                                                F.lag(labeled_features.failure).
                                                over(my_window)).fillna(0))
labeled_features = (labeled_features.withColumn("prev_value2",
                                                F.lag(labeled_features.prev_value1).
                                                over(my_window)).fillna(0))
labeled_features = (labeled_features.withColumn("prev_value3",
                                                F.lag(labeled_features.prev_value2).
                                                over(my_window)).fillna(0))
labeled_features = (labeled_features.withColumn("prev_value4",
                                                F.lag(labeled_features.prev_value3).
                                                over(my_window)).fillna(0))
labeled_features = (labeled_features.withColumn("prev_value5",
                                                F.lag(labeled_features.prev_value4).
                                                over(my_window)).fillna(0))
labeled_features = (labeled_features.withColumn("prev_value6",
                                                F.lag(labeled_features.prev_value5).
                                                over(my_window)).fillna(0))
labeled_features = (labeled_features.withColumn("prev_value7",
                                                F.lag(labeled_features.prev_value6).
                                                over(my_window)).fillna(0))

# Create a label features
labeled_features = (labeled_features.withColumn('label', labeled_features.failure +
                                                labeled_features.prev_value1 +
                                                labeled_features.prev_value2 +
                                                labeled_features.prev_value3 +
                                                labeled_features.prev_value4 +
                                                labeled_features.prev_value5 +
                                                labeled_features.prev_value6 +
                                                labeled_features.prev_value7))

# Restrict the label to be on the range of 0:4, and remove extra columns
labeled_features = (labeled_features.withColumn('label_e', F.when(col('label') > 4, 4.0)
                                                .otherwise(col('label')))
                    .drop(labeled_features.prev_value1).drop(labeled_features.prev_value2)
                    .drop(labeled_features.prev_value3).drop(labeled_features.prev_value4)
                    .drop(labeled_features.prev_value5).drop(labeled_features.prev_value6)
                    .drop(labeled_features.prev_value7).drop(labeled_features.label))

print(labeled_features.count())
labeled_features.limit(5).toPandas()
```

```
731358
Out[22]:
   machineID        dt_truncated  volt_rollingmean_12  rotate_rollingmean_12  \
0        148 2016-01-01 12:00:00           173.270341             439.327705   
1        148 2016-01-01 00:00:00           173.726358             437.405911   
2        148 2015-12-31 12:00:00           172.668539             448.928249   
3        148 2015-12-31 00:00:00           172.766772             456.731624   
4        148 2015-12-30 12:00:00           176.594904             443.839161   

   pressure_rollingmean_12  vibration_rollingmean_12  volt_rollingmean_24  \
0               103.390004                 39.473370           173.250180   
1               102.663146                 40.878636           173.197448   
2                99.298873                 41.416083           172.717655   
3               100.950399                 39.865345           174.680838   
4                99.058761                 40.387021           175.485707   

   rotate_rollingmean_24  pressure_rollingmean_24  vibration_rollingmean_24  \
0             443.242545               102.989579                 40.977328   
1             443.167080               100.981010                 41.147360   
2             452.829937               100.124636                 40.640714   
3             450.285392               100.004580                 40.126183   
```

```
plt_data = (labeled_features.filter(labeled_features.label_e > 0)
            .where(col("machineID").isin({"65", "558", "222", "965"}))
            .select(labeled_features.machineID, labeled_features.label_e)).toPandas()

fig, ax = plt.subplots(figsize = (14,7))
sns.barplot(plt_data['machineID'], plt_data['label_e'], hue = plt_data['label_e'], alpha=0.8).set_title('Label distribution per machine')
ax.set_ylabel(" Label Count")
ax.set_xlabel("Machine ID")
display(ax.figure)
```




```
# Write labeled feature data to storage
labeled_features.write.mode('overwrite').parquet(os.path.join(target_dir, parquet_files_names['features']))

toc = time.time()
print("Full run took %.2f minutes" % ((toc - tic)/60))
```

```
Full run took 27.43 minutes
```

Appendix: model building

model_building1 (Python)

```
import os
import glob
import time

# for creating pipelines and model
from pyspark.ml.feature import (StringIndexer, OneHotEncoder,
                                VectorAssembler, VectorIndexer)
from pyspark.ml import Pipeline, PipelineModel
from pyspark.ml.classification import (RandomForestClassifier,DecisionTreeClassifier)
from pyspark.ml.evaluation import MulticlassClassificationEvaluator

from pyspark.sql.functions import col
from pyspark.sql import SparkSession

# For some data handling & plotting
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
plt.style.use('ggplot')

from azureml.core import (Workspace, VERSION)
from azureml.core.run import Run
from azureml.core.experiment import Experiment

# Time the notebook execution.
# This will only make sense if you "Run all cells"
tic = time.time()

# Check core SDK version number
print("AML-SDK version:", VERSION)
```

```
AML-SDK version: 1.0.17
```

```
# Enter workspace details below

subscription_id = "38620b93-e186-4292-b9d1-4159d7be1b28"
resource_group = "jp-resource-group-PAID"
workspace_name = "jp1-predictive-maintenance-v2-ws"
workspace_location = "West Europe"

ws = Workspace(workspace_name = workspace_name,
               subscription_id = subscription_id,
               resource_group = resource_group)

# persist workspace info in aml_config/config.json which will be needed in notebook 04.
ws.write_config()

myexperiment = Experiment(ws, "Predictive_maintenance_Experiment")
run = myexperiment.start_logging()

Warning: Falling back to use azure cli login credentials.
If you run your code in unattended mode, i.e., where you can't give a user input, then we recommend to use ServicePrincipalAuthentication or MsiAuthentication.
Please refer to aka.ms/aml-notebook-auth for different authentication mechanisms in azureml-sdk.
Performing interactive authentication. Please follow the instructions on the terminal.
To sign in, use a web browser to open the page https://microsoft.com/devicelogin and enter the code CR2CC27H3 to authenticate.
Interactive authentication successfully completed.
Wrote the config file config.json to: /databricks/driver/aml_config/config.json
```

```
features_file = 'featureengineering_files.parquet'
target_dir = "dbfs:/dataset/"
model_dir = "dbfs:/model/"
feat_data = spark.read.parquet(os.path.join(target_dir,features_file))
feat_data.limit(10).toPandas()
```

```
Out[3]:
   machineID        dt_truncated  volt_rollingmean_12  rotate_rollingmean_12  \
0        114 2016-01-01 12:00:00           166.950543             294.433319
1        114 2016-01-01 00:00:00           165.290113             285.328277
2        114 2015-12-31 12:00:00           164.324247             260.243976
3        114 2015-12-31 00:00:00           171.891253             366.555058
4        114 2015-12-30 12:00:00           176.523807             374.260029
5        114 2015-12-30 00:00:00           166.193793             395.526750
6        114 2015-12-29 12:00:00           163.841386             465.801294
7        114 2015-12-29 00:00:00           168.414229             419.006087
8        114 2015-12-28 12:00:00           175.611464             416.986082
9        114 2015-12-28 00:00:00           171.309517             455.897688

   pressure_rollingmean_12  vibration_rollingmean_12  volt_rollingmean_24  \
0                94.473184                 49.062098           165.197336
1                96.147439                 51.315016           164.807180
2               102.238964                 48.625823           168.107750
3               102.072506                 39.475754           174.207530
4               101.621576                 40.830461           171.358800
5                95.027049                 40.731622           165.017589
6                95.226886                 39.178725           166.127807
```

```
# define list of input columns for downstream modeling

# We'll use the known label, and key variables.
label_var = ['label_e']
key_cols =['machineID','dt_truncated']

# Then get the remaing feature names from the data
input_features = feat_data.columns

# We'll use the known label, key variables and
# a few extra columns we won't need.
remove_names = label_var + key_cols + ['failure','model_encoded','model' ]

# Remove the extra names if that are in the input_features list
input_features = [x for x in input_features if x not in set(remove_names)]

input_features
```

```
Out[4]:
['volt_rollingmean_12',
 'rotate_rollingmean_12',
 'pressure_rollingmean_12',
 'vibration_rollingmean_12',
 'volt_rollingmean_24',
 'rotate_rollingmean_24',
 'pressure_rollingmean_24',
 'vibration_rollingmean_24',
 'volt_rollingmean_36',
 'vibration_rollingmean_36',
 'rotate_rollingmean_36',
 'pressure_rollingmean_36',
 'volt_rollingstd_12',
 'rotate_rollingstd_12',
 'pressure_rollingstd_12',
 'vibration_rollingstd_12',
 'volt_rollingstd_24',
 'rotate_rollingstd_24',
 'pressure_rollingstd_24',
 'vibration_rollingstd_24',
```

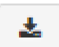
```
# assemble features
va = VectorAssembler(inputCols=(input_features), outputCol='features')
feat_data = va.transform(feat_data).select('machineID','dt_truncated','label_e','features')

# set maxCategories so features with > 10 distinct values are treated as continuous.
featureIndexer = VectorIndexer(inputCol="features",
                               outputCol="indexedFeatures",
                               maxCategories=10).fit(feat_data)

# fit on whole dataset to include all labels in index
labelIndexer = StringIndexer(inputCol="label_e", outputCol="indexedLabel").fit(feat_data)

# split the data into train/test based on date
split_date = "2015-10-30"
training = feat_data.filter(feat_data.dt_truncated < split_date)
testing = feat_data.filter(feat_data.dt_truncated >= split_date)

run.log('training set size',training.count())
run.log('testing set size',testing.count())
```

```
model_type = 'RandomForest' # Use 'DecisionTree', or 'RandomForest'

# train a model.
if model_type == 'DecisionTree':
    model = DecisionTreeClassifier(labelCol="indexedLabel", featuresCol="indexedFeatures",
                                      # Maximum depth of the tree. (>= 0)
                                      # E.g., depth 0 means 1 leaf node; depth 1 means 1 internal node + 2 leaf nodes.'
                                      maxDepth=15,
                                      # Max number of bins for discretizing continuous features.
                                      # Must be >=2 and >= number of categories for any categorical feature.
                                      maxBins=32,
                                      # Minimum number of instances each child must have after split.
                                      # If a split causes the left or right child to have fewer than
                                      # minInstancesPerNode, the split will be discarded as invalid. Should be >= 1.
                                      minInstancesPerNode=1,
                                      # Minimum information gain for a split to be considered at a tree node.
                                      minInfoGain=0.0,
                                      # Criterion used for information gain calculation (case-insensitive).
                                      # Supported options: entropy, gini')
                                      impurity="gini")

    ##==============================================================================================================
    #elif model_type == 'GBTClassifier':
    #    cls_mthd = GBTClassifier(labelCol="indexedLabel", featuresCol="indexedFeatures")
    ##==============================================================================================================
else:
```

```
    model = RandomForestClassifier(labelCol="indexedLabel", featuresCol="indexedFeatures",
                                   # Passed to DecisionTreeClassifier
                                   maxDepth=15,
                                   maxBins=32,
                                   minInstancesPerNode=1,
                                   minInfoGain=0.0,
                                   impurity="gini",
                                   # Number of trees to train (>= 1)
                                   numTrees=50,
                                   # The number of features to consider for splits at each tree node.
                                   # Supported options: auto, all, onethird, sqrt, log2, (0.0-1.0], [1-n].
                                   featureSubsetStrategy="sqrt",
                                   # Fraction of the training data used for learning each
                                   # decision tree, in range (0, 1].'
                                   subsamplingRate = 0.632)

# chain indexers and model in a Pipeline
pipeline_cls_mthd = Pipeline(stages=[labelIndexer, featureIndexer, model])

# train model.  This also runs the indexers.
model_pipeline = pipeline_cls_mthd.fit(training)
```

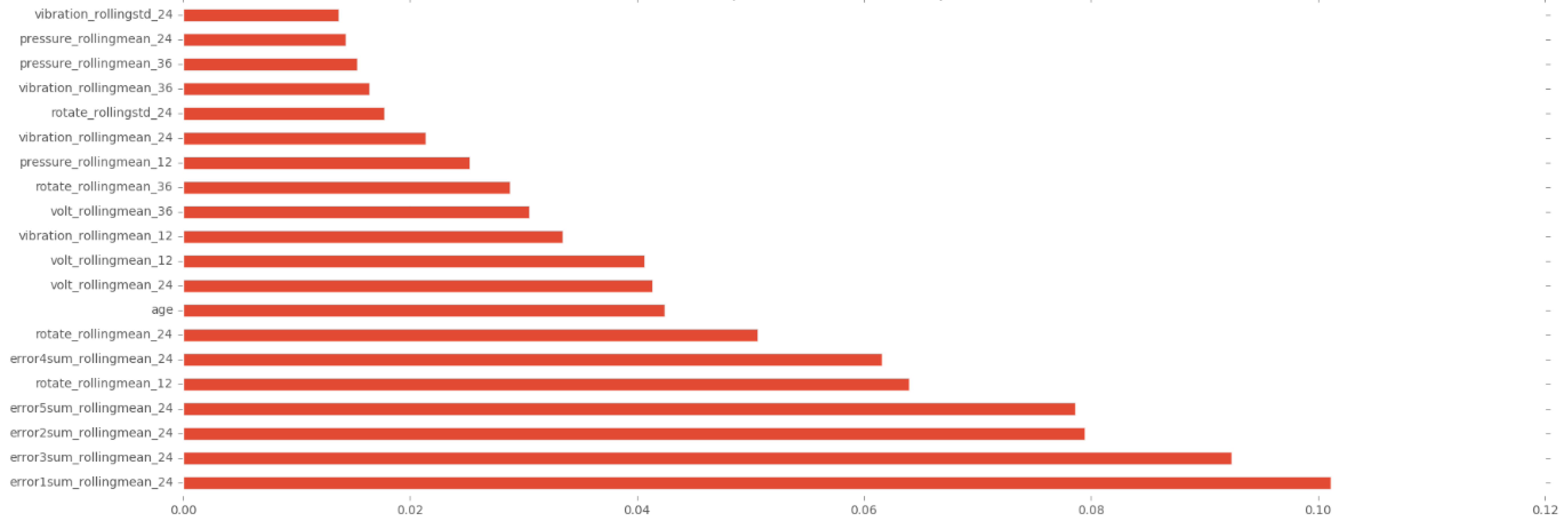
```
# make predictions. The Pipeline does all the same operations on the test data
predictions = model_pipeline.transform(testing)

# Create the confusion matrix for the multiclass prediction results
# This result assumes a decision boundary of p = 0.5
conf_table = predictions.stat.crosstab('indexedLabel', 'prediction')
display(conf_table)
confuse = conf_table.toPandas()
confuse.head()
```

indexedLabel_prediction	0.0	1.0	2.0	3.0	4.0
0.0	119597	9	6	32	4
1.0	2313	787	0	3	1
2.0	1652	0	556	0	0
3.0	1240	4	0	494	0
4.0	1002	8	0	1	342



```
# select (prediction, true label) and compute test error
# select (prediction, true label) and compute test error
# True positives - diagonal failure terms
tp = confuse['1.0'][1]+confuse['2.0'][2]+confuse['3.0'][3]+confuse['4.0'][4]

# False positves - All failure terms - True positives
fp = np.sum(np.sum(confuse[['1.0', '2.0','3.0','4.0']])) - tp

# True negatives
tn = confuse['0.0'][0]

# False negatives total of non-failure column - TN
fn = np.sum(np.sum(confuse[['0.0']])) - tn

# Accuracy is diagonal/total
acc_n = tn + tp
acc_d = np.sum(np.sum(confuse[['0.0','1.0', '2.0','3.0','4.0']]))
acc = acc_n/acc_d

# Calculate precision and recall.
prec = tp/(tp+fp)
rec = tp/(tp+fn)

# Print the evaluation metrics to the notebook
print("Accuracy = %g" % acc)
print("Precision = %g" % prec)
print("Recall = %g" % rec )
print("F1 = %g" % (2.0 * prec * rec/(prec + rec)))
print("")

# track evaluation metrics through AML run.
#
run.log("Model Accuracy", (acc))
run.log("Model Precision", (prec))
run.log("Model Recall", (rec))
run.log("Model F1", (2.0 * prec * rec/(prec + rec)))
run.complete()

Accuracy = 0.950996
Precision = 0.969737
Recall = 0.259838
F1 = 0.409856
```

```
importances = model_pipeline.stages[2].featureImportances

ax = (pd.Series(importances, index=input_features)
       .nlargest(20)
    .plot(kind='barh', title = 'Top 20 model features importance',
          figsize =(20,7)))
run.log_image('Features_importances', plot = ax.figure)
run.complete()
display(ax.figure)
```




```
# save model
model_name = 'pdmrfull.model'
model_pipeline.write().overwrite().save(os.path.join(model_dir,model_name))

# Time the notebook execution.
# This will only make sense if you "Run All" cells
toc = time.time()
print("Full run took %.2f minutes" % ((toc - tic)/60))
```

```
Full run took 21.09 minutes
```

Appendix: operationalization

4_operationalization (Python)

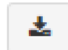
```
## setup our environment by importing required libraries
import json
import os
import shutil
import time

from pyspark.ml import Pipeline
from pyspark.ml.classification import RandomForestClassifier

# for creating pipelines and model
from pyspark.ml.feature import StringIndexer, VectorAssembler, VectorIndexer

# setup the pyspark environment
from pyspark.sql import SparkSession

# AML SDK libraries
from azureml.core import Workspace, Run
from azureml.core.model import Model
from azureml.core.image import ContainerImage
from azureml.core.conda_dependencies import CondaDependencies
from azureml.core.webservice import AciWebservice,Webservice
```

```
features_file = 'featureengineering_files.parquet'
target_dir = "dbfs:/dataset/"

feat_data = spark.read.parquet(os.path.join(target_dir,features_file))
feat_data.limit(5).toPandas().head(5)
```

```
Out[2]:
   machineID         dt_truncated  volt_rollingmean_12  rotate_rollingmean_12  \
0        114 2016-01-01 12:00:00           166.950543             294.433319
1        114 2016-01-01 00:00:00           165.290113             285.328277
2        114 2015-12-31 12:00:00           164.324247             260.243976
3        114 2015-12-31 00:00:00           171.891253             366.555058
4        114 2015-12-30 12:00:00           176.523807             374.260029

   pressure_rollingmean_12  vibration_rollingmean_12  volt_rollingmean_24  \
0                94.473184                 49.062098           165.197336
1                96.147439                 51.315016           164.807180
2               102.238964                 48.625823           168.107750
3               102.072506                 39.475754           174.207530
4               101.621576                 40.830461           171.358800

   rotate_rollingmean_24  pressure_rollingmean_24  vibration_rollingmean_24  \
0             278.987299                97.318305                 50.799015
1             272.786127                99.193202                 49.970419
2             313.399517               102.155735                 44.050788
3             370.407544               101.847041                 40.153107
4             384.893390                98.324312                 40.781041
```

```
model_name = 'pdmrfull.model'
model_local = "file:" + os.getcwd() + "/" + model_name
model_dir = os.path.join("dbfs:/model/", model_name)
dbutils.fs.cp(model_dir, model_local, True)
display(dbutils.fs.ls(model_local))
```

path	name	size
file:/databricks/driver/pdmrfull.model/metadata/	metadata/	4096
file:/databricks/driver/pdmrfull.model/stages/	stages/	4096



```
ws = Workspace.from_config()
model_name = 'pdmrfull.model'
model = Model.register(model_path= model_name, model_name=model_name , workspace=ws)
print("Registered:", model.name)
```

```
Found the config file in: /databricks/driver/aml_config/config.json
Registering model pdmrfull.model
Registered: pdmrfull.model
```

```
conda_env = CondaDependencies.create(conda_packages=['pyspark'])
with open("conda_env.yml","w") as f:
    f.write(conda_env.serialize_to_string())
```

```
%%writefile score.py

from azureml.core.model import Model
from pyspark.ml.feature import StringIndexer, VectorAssembler, VectorIndexer
from pyspark.ml import PipelineModel
import pyspark
import json

def init():

    global pipeline,spark

    spark = pyspark.sql.SparkSession.builder.appName("Predictive maintenance service").getOrCreate()
    model_path = Model.get_model_path('pdmrfull.model')
    pipeline = PipelineModel.load(model_path)
```

```
def run(raw_data):

    try:
        sc = spark.sparkContext
        input_list = json.loads(raw_data)
        input_rdd = sc.parallelize(input_list)
        input_df = spark.read.json(input_rdd)

        key_cols =['label_e','machineID','dt_truncated', 'failure','model_encoded','model']
        input_features = input_df.columns

        # Remove unseen features by the model during training
        input_features = [x for x in input_features if x not in set(key_cols)]


        va = VectorAssembler(inputCols=(input_features), outputCol='features')
        data = va.transform(input_df).select('machineID','features')
        score = pipeline.transform(data)
        predictions = score.collect()

        preds = [str(x['prediction']) for x in predictions]
        result = preds
    except Exception as e:
        result = str(e)

    return json.dumps({"result":result})

Writing score.py
```

```
image_config = ContainerImage.image_configuration(runtime= "spark-py",
                                 execution_script="score.py",
                                 conda_file="conda_env.yml")

aci_config = AciWebservice.deploy_configuration(cpu_cores = 2,
                                               memory_gb = 4,
                                               tags = {'type': "predictive_maintenance"},
                                               description = "Predictive maintenance classifier")



aci_service_name = 'pred-maintenance-service'
print(aci_service_name)

aci_service = Webservice.deploy_from_model(workspace=ws,
                                        name=aci_service_name,
                                        deployment_config = aci_config,
                                        models = [model],
                                        image_config = image_config
                                          )


aci_service.wait_for_deployment(True)
print(aci_service.state)

pred-maintenance-service
Creating image
Image creation operation finished for image pred-maintenance-service:1, operation "Succeeded"
Creating service
Running..........................................................................
SucceededACI service creation operation finished, operation "Succeeded"
Healthy
```

```
test_sample = (feat_data.sample(False, .8).limit(1))
excluded_cols = {'label_e','machineID','dt_truncated','failure','model_encoded','model'}
input_features = set(test_sample.columns)- excluded_cols


raw_input = test_sample.toJSON().collect()
prediction = aci_service.run(json.dumps(raw_input))

print(prediction)

{"result": ["0.0"]}
```

```
aci_service.delete()
```

Thank you for your interest in our free and voluntary UberCloud Experiment

If you, as an end-user, would like to participate in an UberCloud Experiment to explore hands-on the end-to-end process of on-demand Technical Computing as a Service, in the Cloud, for your business then please register at: http://www.theubercloud.com/hpc-experiment/.

If you, as a service provider, are interested in building a SaaS solution and promoting your services on the UberCloud Marketplace then please send us a message at https://www.theubercloud.com/help/.

2013 Compendium of case studies: https://www.theubercloud.com/ubercloud-compendium-2013/
2014 Compendium of case studies: https://www.theubercloud.com/ubercloud-compendium-2014/
2015 Compendium of case studies: https://www.theubercloud.com/ubercloud-compendium-2015/
2016 Compendium of case studies: https://www.theubercloud.com/ubercloud-compendium-2016/
2018 Compendium of case studies: https://www.theubercloud.com/ubercloud-compendium-2018/

The UberCloud Experiments and Teams received several prestigious international Awards, among other:

- HPCwire Readers Choice Award 2013: http://www.hpcwire.com/off-the-wire/ubercloud-receives-top-honors-2013-hpcwire-readers-choice-awards/
- HPCwire Readers Choice Award 2014: https://www.theubercloud.com/ubercloud-receives-top-honors-2014-hpcwire-readers-choice-award/
- Gartner Cool Vendor Award 2015: http://www.digitaleng.news/de/ubercloud-names-cool-vendor-for-oil-gas-industries/
- HPCwire Editors Award 2017: https://www.hpcwire.com/2017-hpcwire-awards-readers-editors-choice/
- IDC/Hyperion Research Innovation Excellence Award 2017: https://www.hpcwire.com/off-the-wire/hyperion-research-announces-hpc-innovation-excellence-award-winners-2/

If you wish to be informed about the latest developments in technical computing in the cloud, then please **register at http://www.theubercloud.com/** and you will get our free monthly newsletter.

With support from

Please contact UberCloud help@theubercloud.com before distributing this material in part or in full. © Copyright 2018 UberCloud™. UberCloud is a trademark of TheUberCloud Inc.