Leveraging Distributed Computing for Predictive Maintenance: A PySpark Approach to Industrial Equipment Monitoring

GitHub:**https://github.com/MuneendraMagani/-Intro-to-Big-Data-Analytics**

Problem Statement:

The project aimed to leverage PySpark for distributed data processing and predictive modeling on the AI4I 2020 Predictive Maintenance Dataset. The goals were:

1. To determine whether a machine is likely to fail.
2. To identify the type of defect if a failure is predicted.
3. To compare the performance and interpretability of multiple predictive models, evaluating them on metrics such as accuracy, AUC, precision, recall, sensitivity, and specificity.

Dataset Description: The dataset is synthetic yet reflects real-world industrial scenarios, essential for developing predictive maintenance solutions.

Introduction:

This project outlines methods used to preprocess, analyze, and model the predictive maintenance dataset with PySpark. The process involves dimensionality reduction, statistical analysis, feature engineering, and data cleaning.

Configuring the Environment:

A PySpark session was configured to manage large-scale data processing efficiently.

Data Loading and Description:

The ai4i2020.csv dataset was loaded as a DataFrame. Initial data exploration involved assessing data types, checking for missing values, and examining the data structure.

Handling Missing Values (Muneendra Magani):

To address missing data in numeric columns, each column's mean was calculated and used to impute missing values, ensuring data completeness while maintaining distribution integrity.

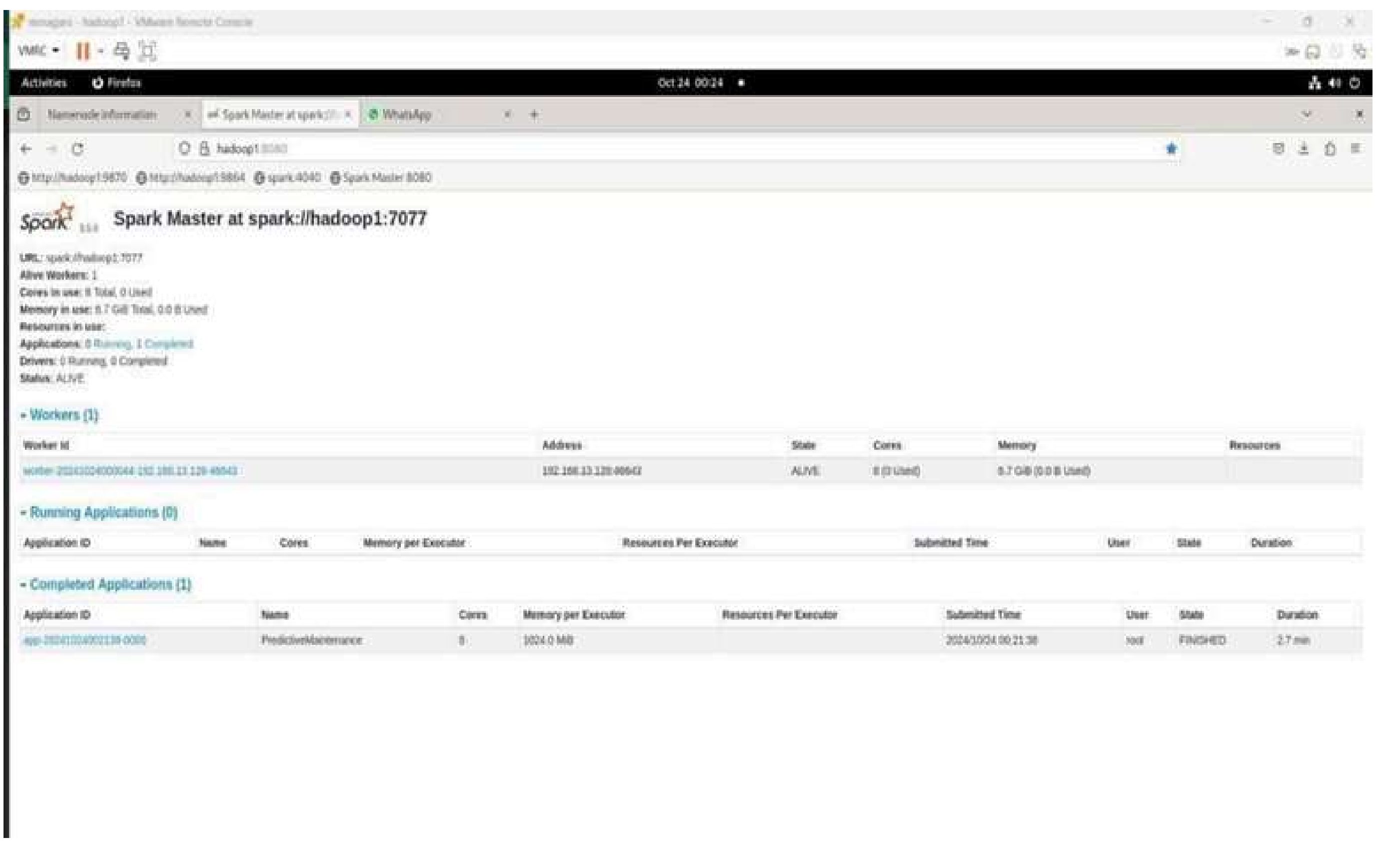
Checking for missing values:

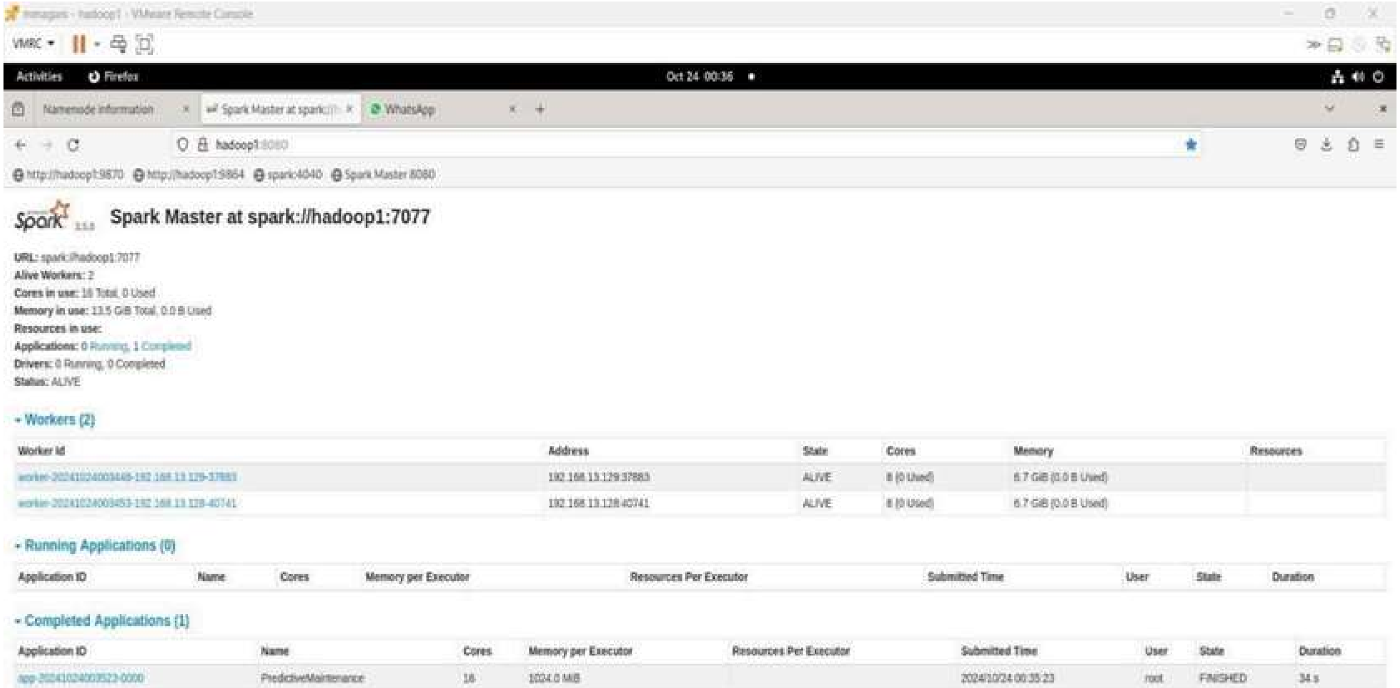
```
root
 |-- UDI: integer (nullable = true)
 |-- Product ID: string (nullable = true)
 |-- Type: string (nullable = true)
 |-- Air temperature [K]: double (nullable = true)
 |-- Process temperature [K]: double (nullable = true)
 |-- Rotational speed [rpm]: integer (nullable = true)
 |-- Torque [Nm]: double (nullable = true)
 |-- Tool wear [min]: integer (nullable = true)
 |-- Machine failure: integer (nullable = true)
 |-- TWF: integer (nullable = true)
 |-- HDF: integer (nullable = true)
 |-- PWF: integer (nullable = true)
 |-- OSF: integer (nullable = true)
 |-- RNF: integer (nullable = true)
```

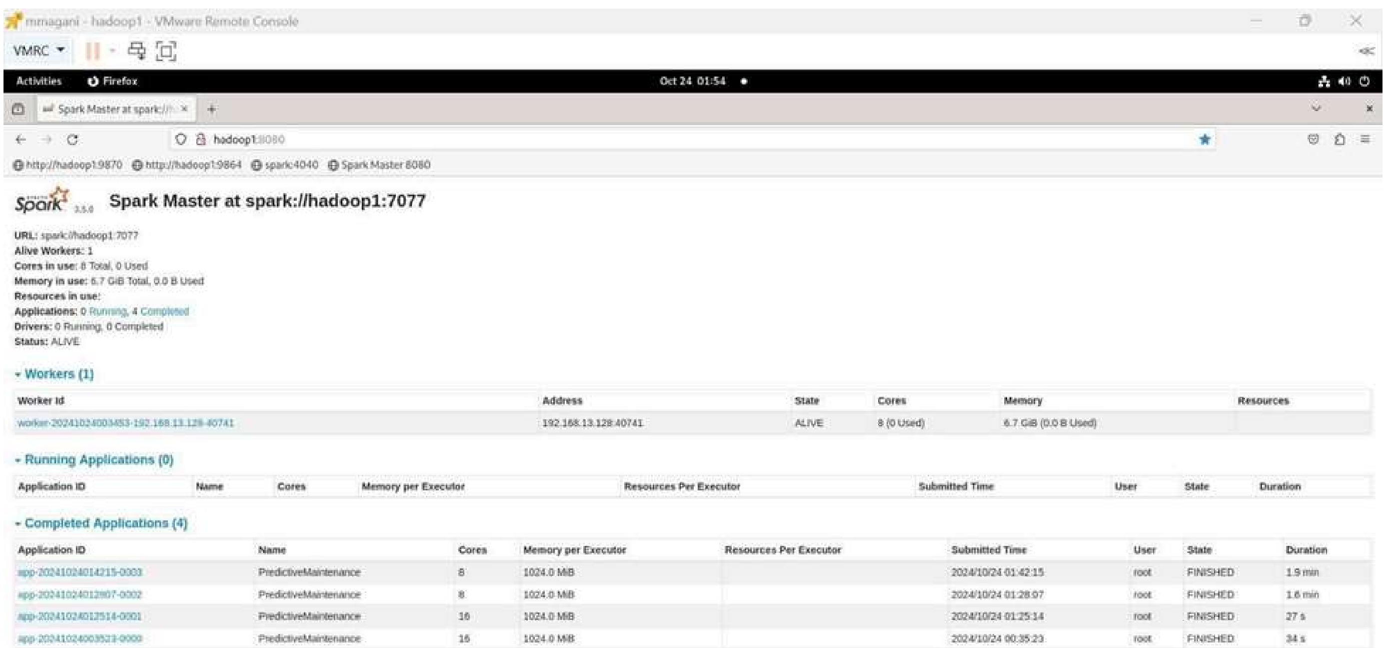
After imputation:

```
+---+----------+----+-------------------+-----------------------+----------------------+-----------+---------------+---------------+---+---+---+---+---+
|UDI|Product ID|Type|Air temperature [K]|Process temperature [K]|Rotational speed [rpm]|Torque [Nm]|Tool wear [min]|Machine failure|TWF|HDF|PWF|OSF|RNF|
+---+----------+----+-------------------+-----------------------+----------------------+-----------+---------------+---------------+---+---+---+---+---+
|  0|         0|   0|                  0|                      0|                     0|          0|              0|              0|  0|  0|  0|  0|  0|
+---+----------+----+-------------------+-----------------------+----------------------+-----------+---------------+---------------+---+---+---+---+---+
```

Computational Time Comparison: One VM vs. Two VM







Using two virtual machines significantly reduced processing time, highlighting the benefit of distributed computing for large datasets.

Data Integrity Check:

Duplicate entries were identified and removed to maintain data integrity.

Feature Engineering (Madhumitha Mandayam):

Binning and dummy variable addition were applied to continuous variables, transforming them for better pattern recognition.

Post-Binning:

UDI	Product ID	Type	Air temperature [K]	Process temperature [K]	Rotational speed [rpm]	Torque [Nm]	Tool wear [min]	Machine failure	TWF	HDF	PWF	OSF	RNF	temp_binned
1	M14860	M	298.1	308.6	1551	42.8	0	0	0	0	0	0	0	0.0
2	L47181	L	298.2	308.7	1408	46.3	3	0	0	0	0	0	0	0.0
3	L47182	L	298.1	308.5	1498	49.4	5	0	0	0	0	0	0	0.0
4	L47183	L	298.2	308.6	1433	39.5	7	0	0	0	0	0	0	0.0
5	L47184	L	298.2	308.7	1408	40.0	9	0	0	0	0	0	0	0.0
6	M14865	M	298.1	308.6	1425	41.9	11	0	0	0	0	0	0	0.0
7	L47186	L	298.1	308.6	1558	42.4	14	0	0	0	0	0	0	0.0
8	L47187	L	298.1	308.6	1527	40.2	16	0	0	0	0	0	0	0.0
9	M14868	M	298.3	308.7	1667	28.6	18	0	0	0	0	0	0	0.0
10	M14869	M	298.5	309.0	1741	28.0	21	0	0	0	0	0	0	0.0
11	H29424	H	298.4	308.9	1782	23.9	24	0	0	0	0	0	0	0.0
12	H29425	H	298.6	309.1	1423	44.3	29	0	0	0	0	0	0	0.0
13	M14872	M	298.6	309.1	1339	51.1	34	0	0	0	0	0	0	0.0
14	M14873	M	298.6	309.2	1742	30.0	37	0	0	0	0	0	0	0.0
15	L47194	L	298.6	309.2	2035	19.6	40	0	0	0	0	0	0	0.0
16	L47195	L	298.6	309.2	1542	48.4	42	0	0	0	0	0	0	0.0
17	M14876	M	298.6	309.2	1311	46.6	44	0	0	0	0	0	0	0.0
18	M14877	M	298.7	309.2	1410	45.6	47	0	0	0	0	0	0	0.0
19	H29432	H	298.8	309.2	1306	54.5	50	0	0	0	0	0	0	0.0
20	M14879	M	298.9	309.3	1632	32.5	55	0	0	0	0	0	0	0.0

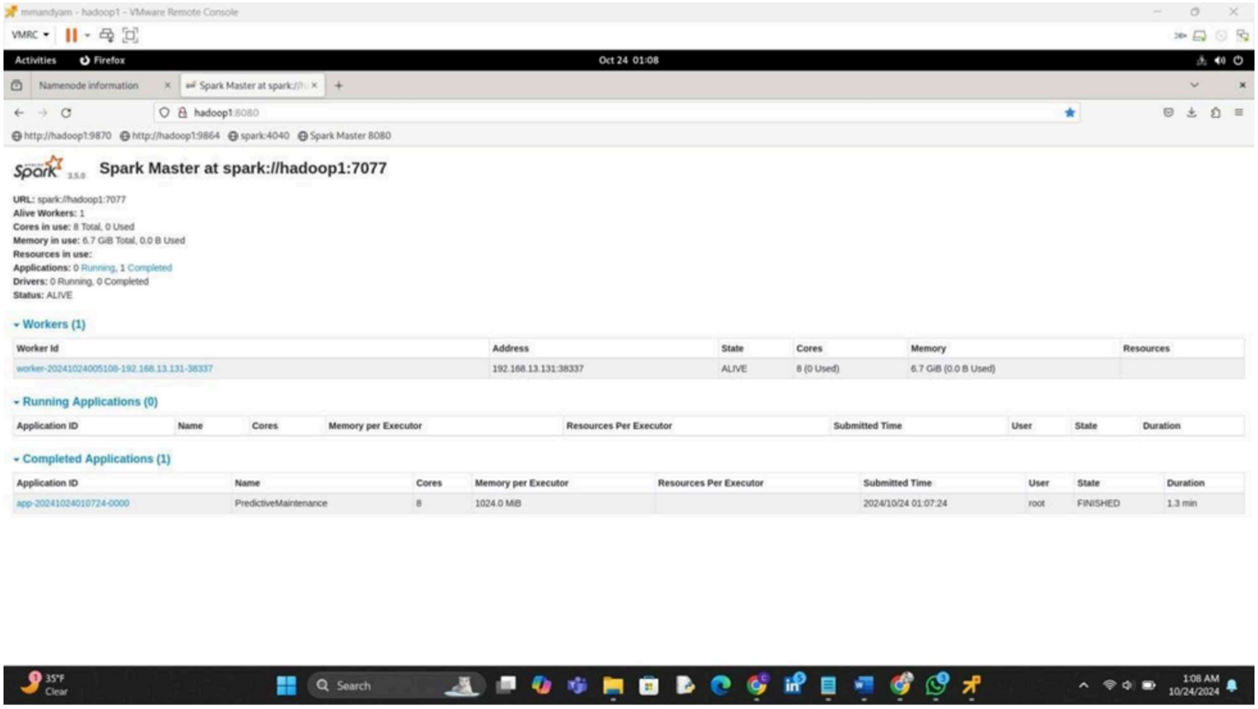
only showing top 20 rows

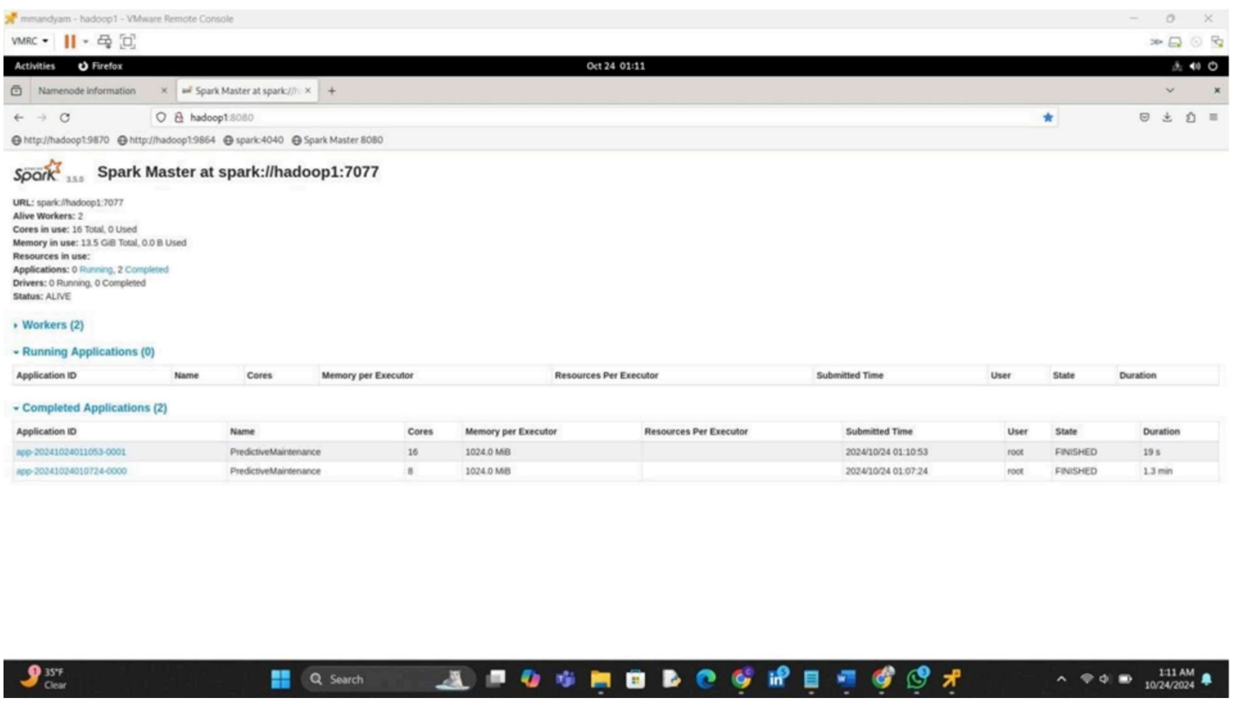
Post-Addition of Dummy Variables:

String indexing and one-hot encoding were applied to categorical variables, ensuring compatibility with machine learning algorithms.

UDI	Product ID	Type	Air temperature [K]	Process temperature [K]	Rotational speed [rpm]	Torque [Nm]	Tool wear [min]	Machine failure	TWF	HDF	PWF	OSF	RNF	temp_binned	machine_failure_index	machine_failure_ohe	count

Computational Time Comparison: One VM vs. Two VMs:





Statistical Analysis (Ganesh Vannam) :

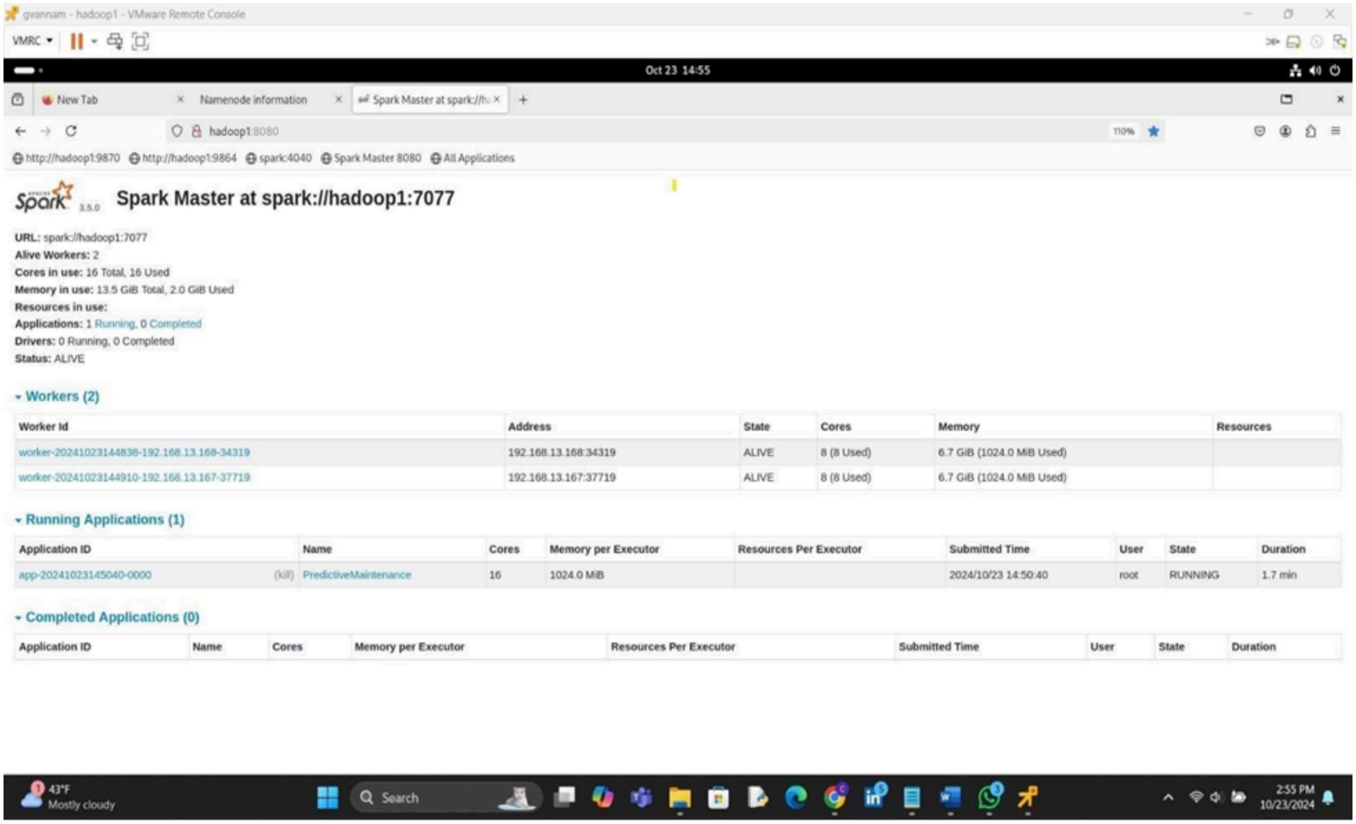
1. **Correlation Analysis**: Pearson's correlation was used to assess relationships among numerical features.

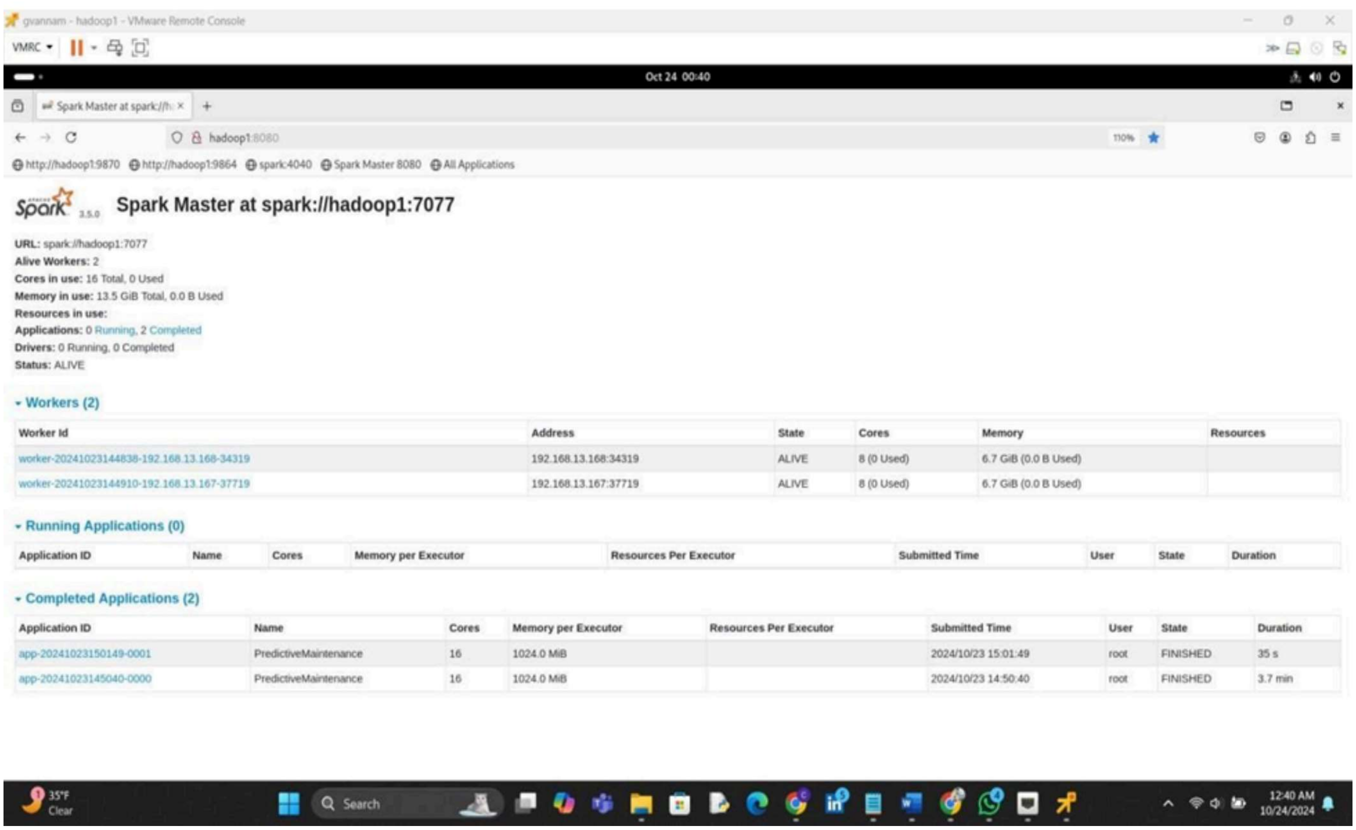
```
Pearson correlation matrix:
DenseMatrix([[ 1.00000000e+00,  1.17427946e-01,  3.24428145e-01,
              -6.61486827e-03,  3.20658647e-03, -1.07020066e-02,
              -2.28918055e-02,  9.15438096e-03, -2.22147372e-02,
              -2.35565593e-02, -9.90258951e-04, -5.95373622e-03],
             [ 1.17427946e-01,  1.00000000e+00,  8.76107158e-01,
               2.26704588e-02, -1.37778231e-02,  1.38528277e-02,
               8.25556898e-02,  9.95472389e-03,  1.37830939e-01,
               3.46950914e-03,  1.98792091e-03,  1.76876819e-02],
             [ 3.24428145e-01,  8.76107158e-01,  1.00000000e+00,
               1.92767139e-02, -1.40606131e-02,  1.34875171e-02,
               3.59459733e-02,  7.31534360e-03,  5.69328837e-02,
              -3.35476275e-03,  4.55351734e-03,  2.22789200e-02],
             [-6.61486827e-03,  2.26704588e-02,  1.92767139e-02,
               1.00000000e+00, -8.75027086e-01,  2.23084840e-04,
              -4.41875597e-02,  1.03890526e-02, -1.21240693e-01,
               1.23017838e-01, -1.04574712e-01, -1.30875702e-02],
             [ 3.20658647e-03, -1.37778231e-02, -1.40606131e-02,
              -8.75027086e-01,  1.00000000e+00, -3.09278144e-03,
               1.91320775e-01, -1.46616270e-02,  1.42610182e-01,
               8.37810778e-02,  1.83464795e-01,  1.61364992e-02],
             [-1.07020066e-02,  1.38528277e-02,  1.34875171e-02,
               2.23084840e-04, -3.09278144e-03,  1.00000000e+00,
               1.05448219e-01,  1.15792057e-01, -1.28734586e-03,
              -9.33444504e-03,  1.55893672e-01,  1.13257088e-02],
             [-2.28918055e-02,  8.25556898e-02,  3.59459733e-02,
              -4.41875597e-02,  1.91320775e-01,  1.05448219e-01,
               1.00000000e+00,  3.62903611e-01,  5.75800152e-01,
               5.22812250e-01,  5.31083451e-01,  4.51599310e-03],
             [ 9.15438096e-03,  9.95472389e-03,  7.31534360e-03,
               1.03890526e-02, -1.46616270e-02,  1.15792057e-01,
               3.62903611e-01,  1.00000000e+00, -7.33230769e-03,
               8.57712261e-03,  3.82429756e-02,  3.09698358e-02],
             [-2.22147372e-02,  1.37830939e-01,  5.69328837e-02,
              -1.21240693e-01,  1.42610182e-01, -1.28734586e-03,
               5.75800152e-01, -7.33230769e-03,  1.00000000e+00,
               1.84432837e-02,  4.63964397e-02, -4.70598302e-03],
             [-2.35565593e-02,  3.46950914e-03, -3.35476275e-03,
               1.23017838e-01,  8.37810778e-02, -9.33444504e-03,
               5.22812250e-01,  8.57712261e-03,  1.84432837e-02,
               1.00000000e+00,  1.15836345e-01, -4.27291580e-03],
             [-9.90258951e-04,  1.98792091e-03,  4.55351734e-03,
              -1.04574712e-01,  1.83464795e-01,  1.55893672e-01,
               5.31083451e-01,  3.82429756e-02,  4.63964397e-02,
               1.15836345e-01,  1.00000000e+00, -4.34051588e-03],
             [-5.95373622e-03,  1.76876819e-02,  2.22789200e-02,
              -1.30875702e-02,  1.61364992e-02,  1.13257088e-02,
               4.51599310e-03,  3.09698358e-02, -4.70598302e-03,
              -4.27291580e-03, -4.34051588e-03,  1.00000000e+00]])
```

2. **Chi-Square Test**: Applied to categorical variables, this test identified significant predictors of the target variable.

```
+-------+----------------+----------+
|pValues|degreesOfFreedom|statistics|
+-------+----------------+----------+
|  [0.0]|             [1]| [10000.0]|
+-------+----------------+----------+
```

Computational Time Comparison: One VM vs. Two VMs:





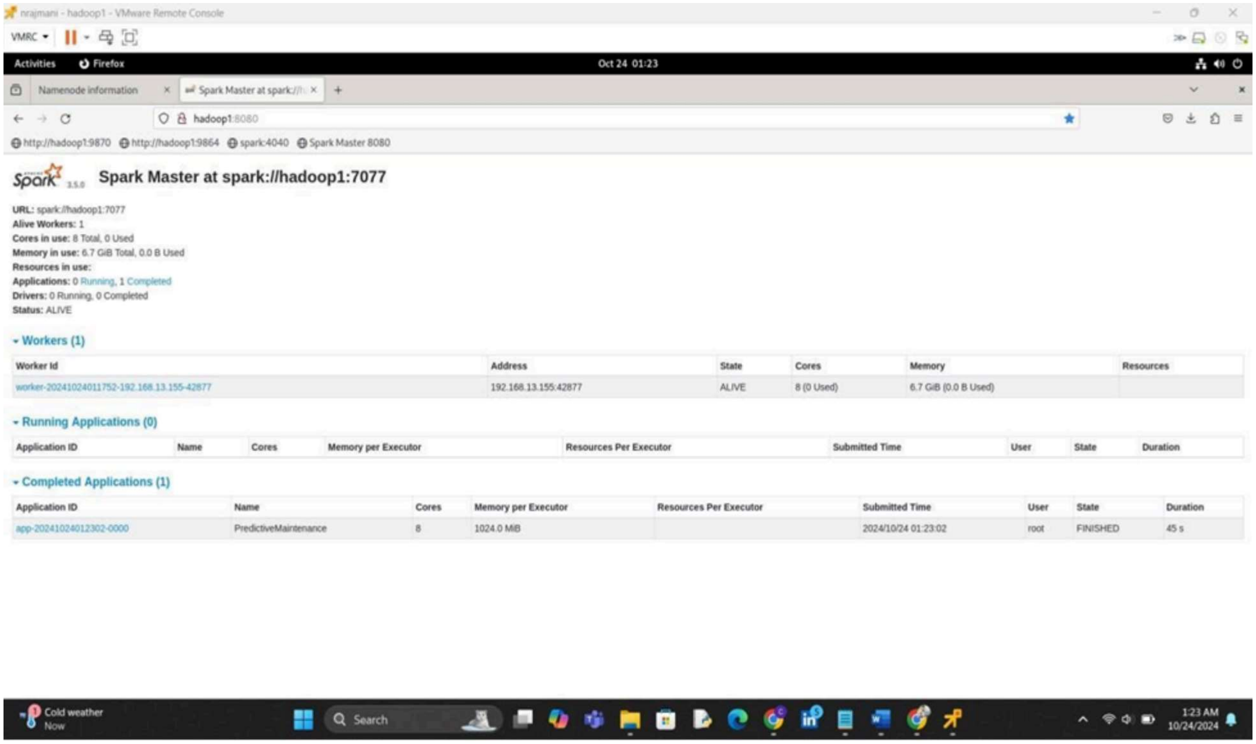
Dimensionality Reduction(Nandhika Rajmanikandan) :

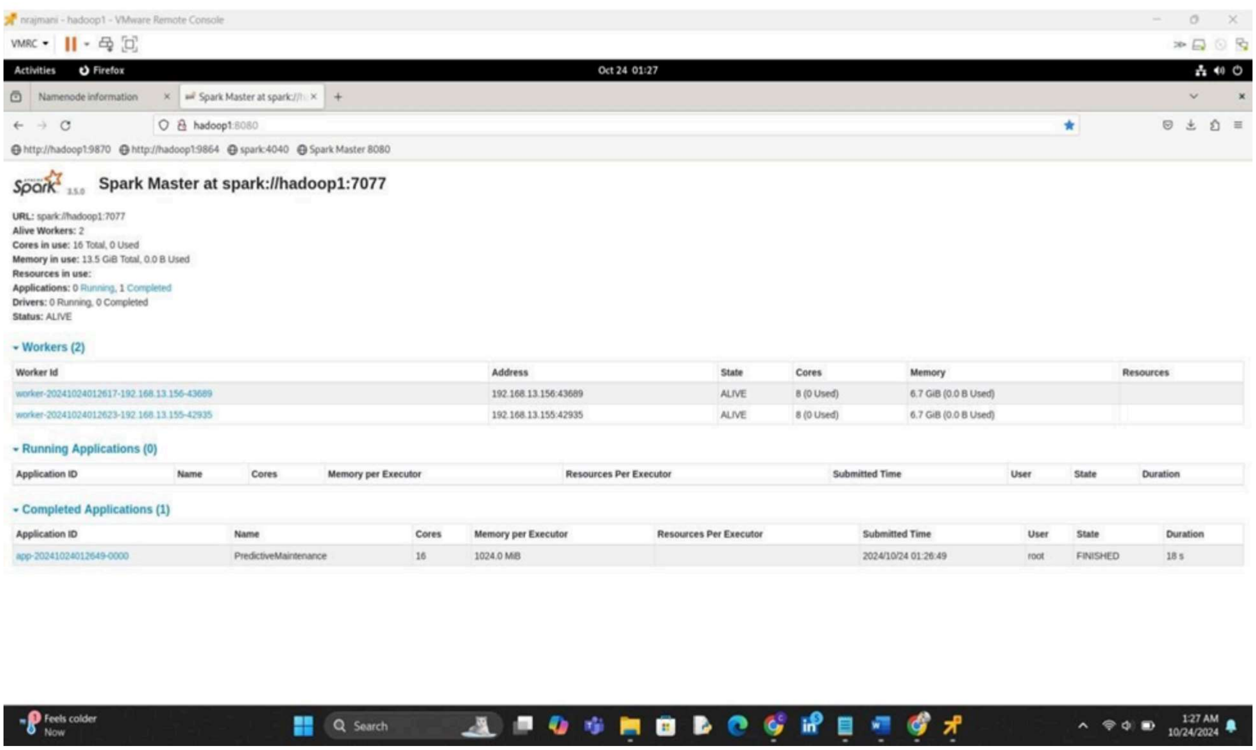
To reduce the dataset's complexity, we used PCA. This technique reduces the complexity of the dataset by maintaining the most important information, making it easier to view and analyze. PCA simplifies complex datasets while maintaining the features that contribute the most to the data's variability.

After PCA:

```
+-----------------------------------------------------------+
|pca_features                                               |
+-----------------------------------------------------------+
|[-0.4365649968819622,-1547.215864740205,0.10890732698675912]|
|[-1.4948927393474452,-1404.2155327130044,3.12013879023679]  |
|[-2.457298085693965,-1493.9587377230232,5.11096032735258]   |
|[-3.4835457919455197,-1429.5177216207737,7.121487050984603] |
|[-4.493405579242876,-1404.5235512153988,9.123720115736777]  |
|[-5.4859189580057794,-1421.4115268670769,11.121513898080838]|
|[-6.430367782368014,-1554.230416987365,14.109887664171229]  |
|[-7.442655151450532,-1523.374640440843,16.113838472753823] |
|[-8.384351333262488,-1663.7737183623321,18.10725866341309]  |
|[-9.353185414265013,-1737.7160349836047,21.10150877296311]  |
|[-10.33549835502489,-1778.8674742956066,24.099941985209984] |
|[-11.482644211861809,-1419.301034813324,29.122437197284725] |
|[-12.516180539217277,-1335.0704736239982,34.126913909390694]|
|[-13.349058893821521,-1738.6203337772301,37.10157228399983] |
|[-14.22740318913697,-2031.7799769251228,40.08092761470856]  |
|[-15.43056210331986,-1537.9633161951444,42.11113542032235]  |
|[-16.5253326691062,-1307.3247781681216,44.13239642956749]   |
|[-17.483794289266832,-1406.2568824934344,47.12447625829836] |
|[-18.526082216384797,-1301.9477561989427,50.12979196994947] |
|[-19.390240811479035,-1628.6326689424309,55.1116426439496]  |
+-----------------------------------------------------------+
only showing top 20 rows
```

Computational Time Comparison: One VM vs. Two VMs:





Examining the distribution of the target variable and identifying the appropriate resampling technique.

```
+-----------------+-----+
|machine_fail_index|count|
+-----------------+-----+
|              0.0| 9661|
|              1.0|  339|
+-----------------+-----+
```

If we can see, the imbalance in the target variable prompted the use of stratified random sampling.

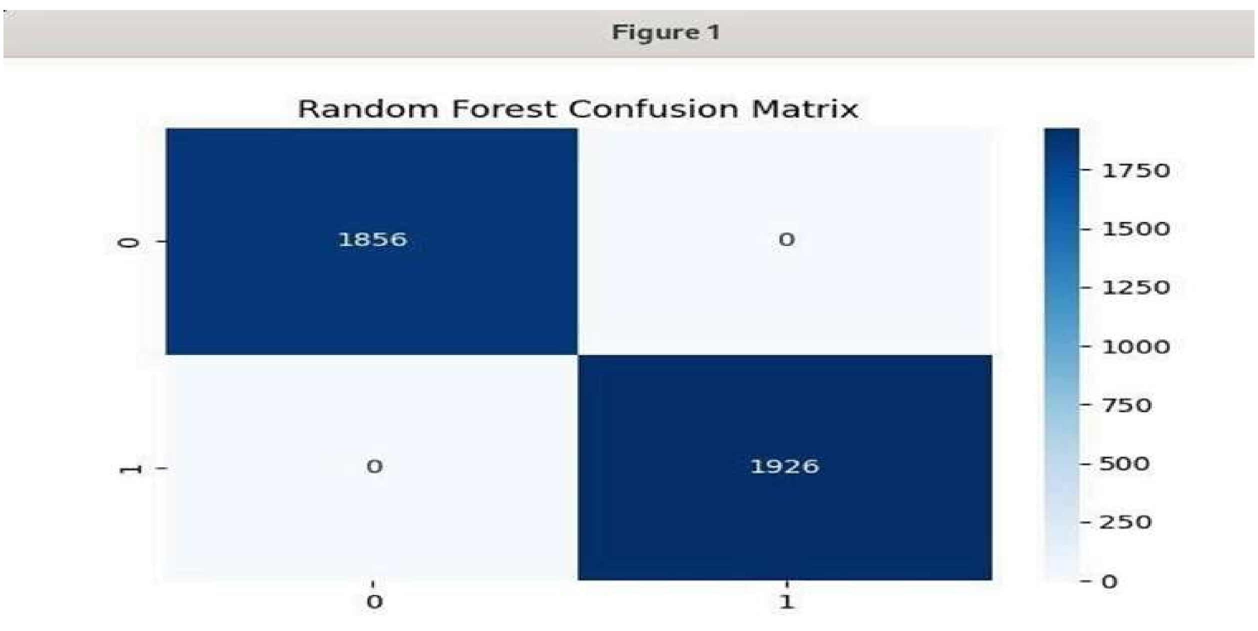
Data Splitting:

The data was split into 80% training and 20% testing sets. A 10-fold cross-validation strategy and a parameter grid builder function were used for hyperparameter tuning.

Model Selection and Performance:

Random Forest Classifier (Muneendra Magani):

To improve accuracy and reduce overfitting, we used PySpark's RandomForestClassifier, which creates multiple decision trees during training and combines their predictions. We adjusted settings like the number of trees and their depth through 10-fold cross-validation to find the best results. We then measured the model's performance using accuracy, AUC, precision, recall, sensitivity, and specificity.

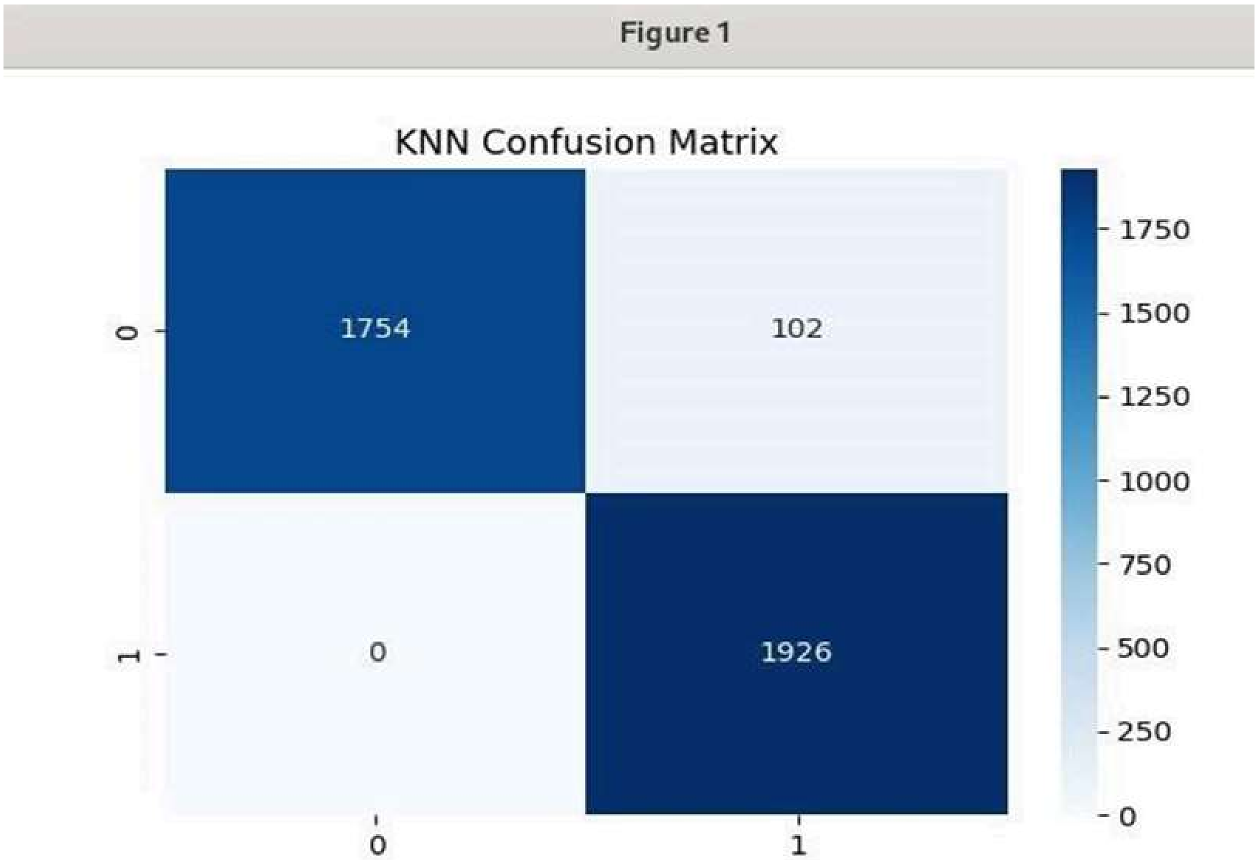


Interpretation:

Accuracy, AUC, Precision, Recall, Sensitivity, Specificity: All metrics at 1.0, indicating perfect classification performance with no misclassifications.

K-Nearest Neighbors (Madhumitha Mandayam):

The KNN algorithm determines the class of each data point by looking at the most common class among its nearest neighbors. To improve the model's performance, we tuned the key hyperparameter,the number of neighbors using grid search combined with cross-validation, helping to find the optimal setting for accurate classification.



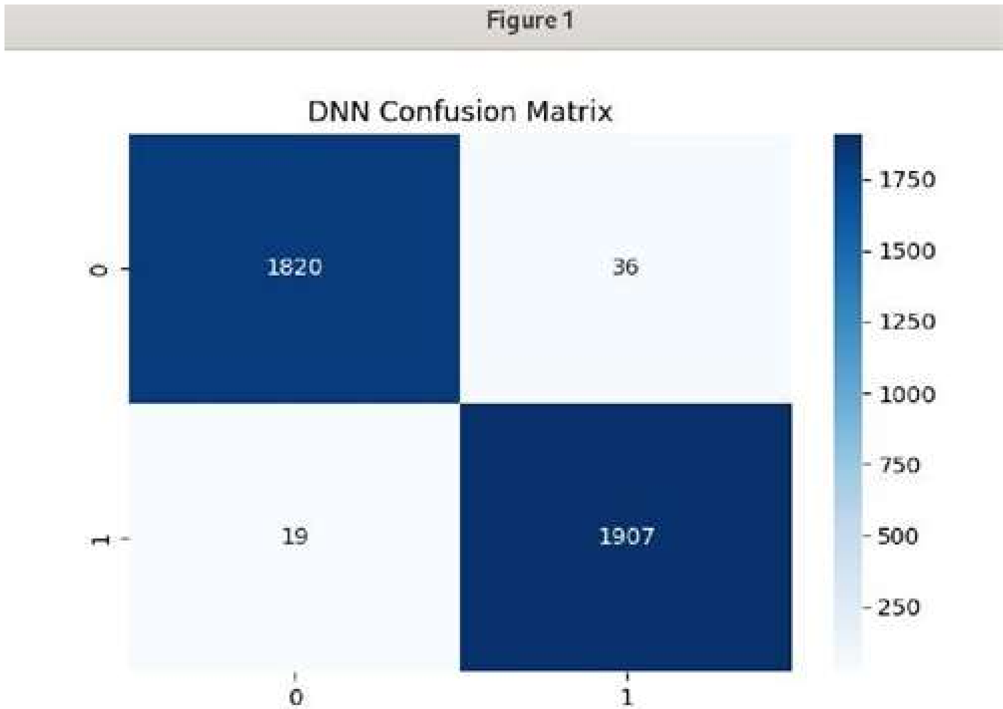
Interpretation:

Using KNeighborsClassifier, KNN showed strong performance but slightly lower specificity:

- Accuracy: 0.973.
- AUC: 0.973.
- Precision: 0.95, with a few false positives.
- Recall and Sensitivity: 1.0, indicating perfect true positive detection.
- Specificity: 0.945, suggesting more false positives compared to other models.

Deep neural network(Nandhika Rajamanikandan):

Using TensorFlow's Keras API, we constructed a multi-layer DNN with dropout layers to reduce overfitting. We applied a stratified 10-fold cross-validation approach to evaluate the model's structure and fine-tune its hyperparameters. The DNN's classification performance was then evaluated on the test set using consistent metrics.



Interpretation:

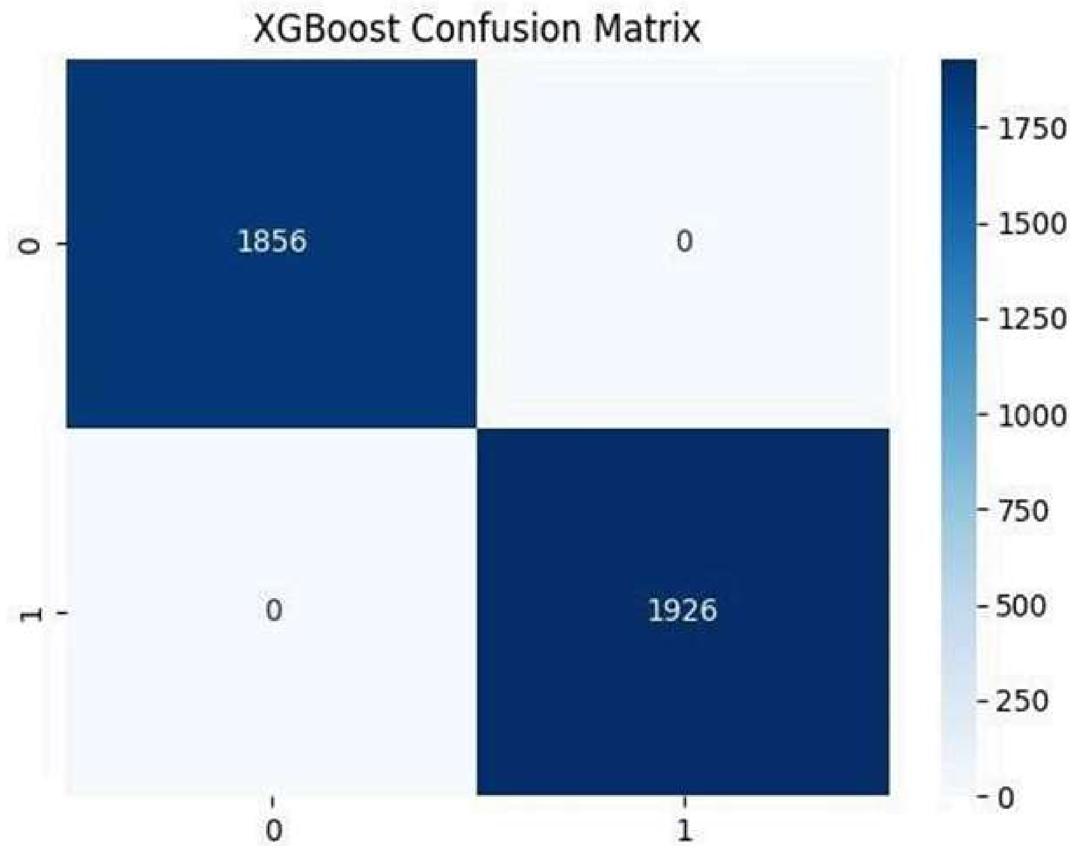
A DNN using TensorFlow's Keras API showed promising results, though with slightly reduced recall and sensitivity:

- **Accuracy**: 0.941.
- **AUC**: 0.941.
- **Precision**: 0.979.
- **Recall and Sensitivity**: 0.903, indicating some missed failures.
- **Specificity**: 0.98.

XGBoost:(Ganesh Vannam):

The approach utilizes a gradient boosting method with the XGBClassifier, which builds a series of models where each is weighted to enhance prediction accuracy. To optimize the model, we employed grid search for hyperparameter tuning, focusing on parameters such as the number of estimators, maximum tree depth, and learning rate. We further improved performance through cross-validation with the XGBClassifier. Key metrics and a confusion matrix were used to rigorously assess model performance, ensuring it met project requirements. The XGBoost technique's gradient boosting mechanism contributed significantly to boosting predictive accuracy.

Figure 1



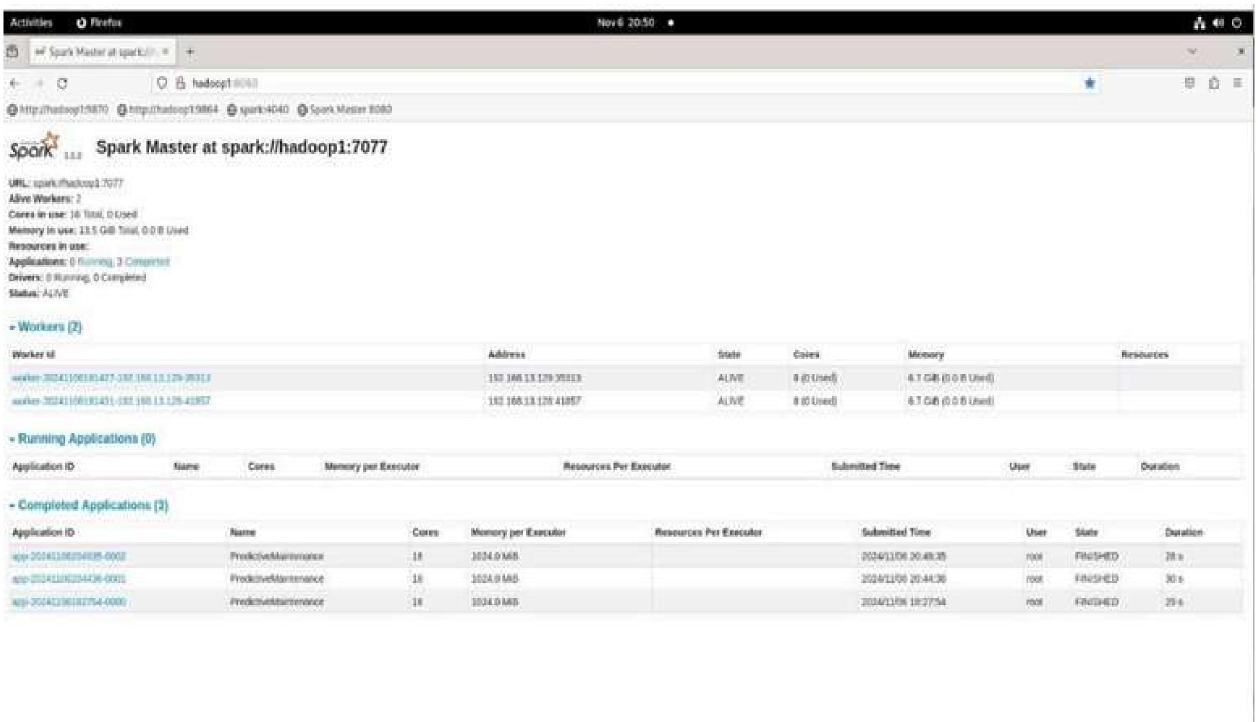
Interpretation:

Utilizing XGBClassifier with grid search for parameter tuning, XGBoost showed optimal performance:

- **Accuracy, AUC, Precision, Recall, Sensitivity, Specificity**: All metrics at 1.0, indicating flawless classification capability**.**

Computational Time Comparison:

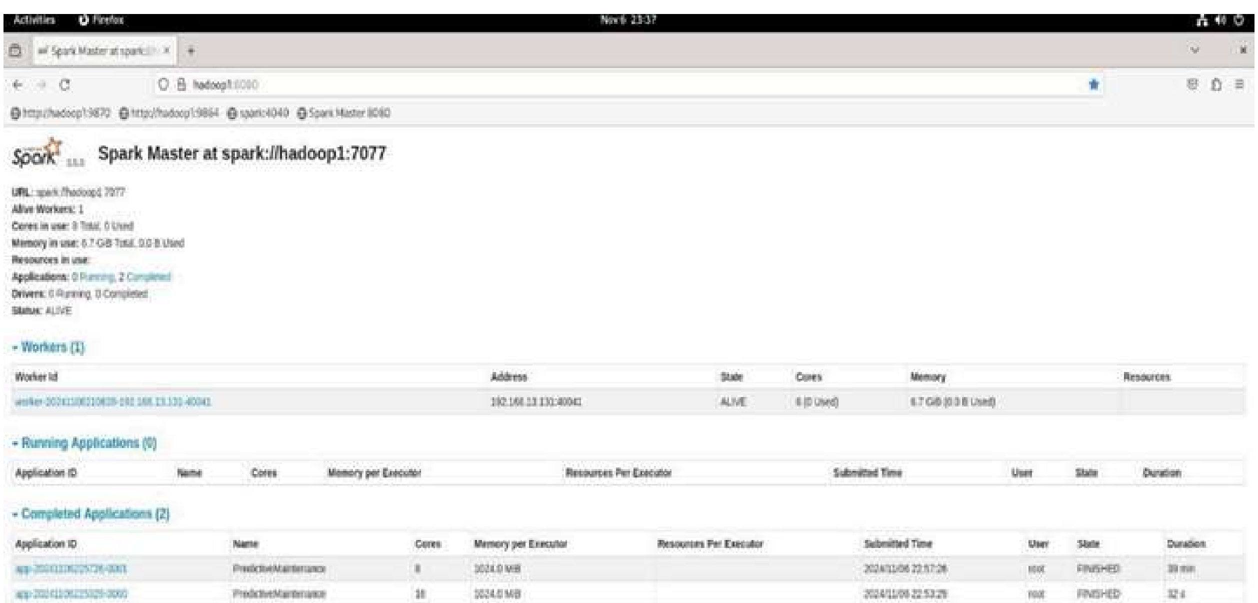
Muneendra Magani:



Time consumed when 1 virtual machine was deployed – 25 mins

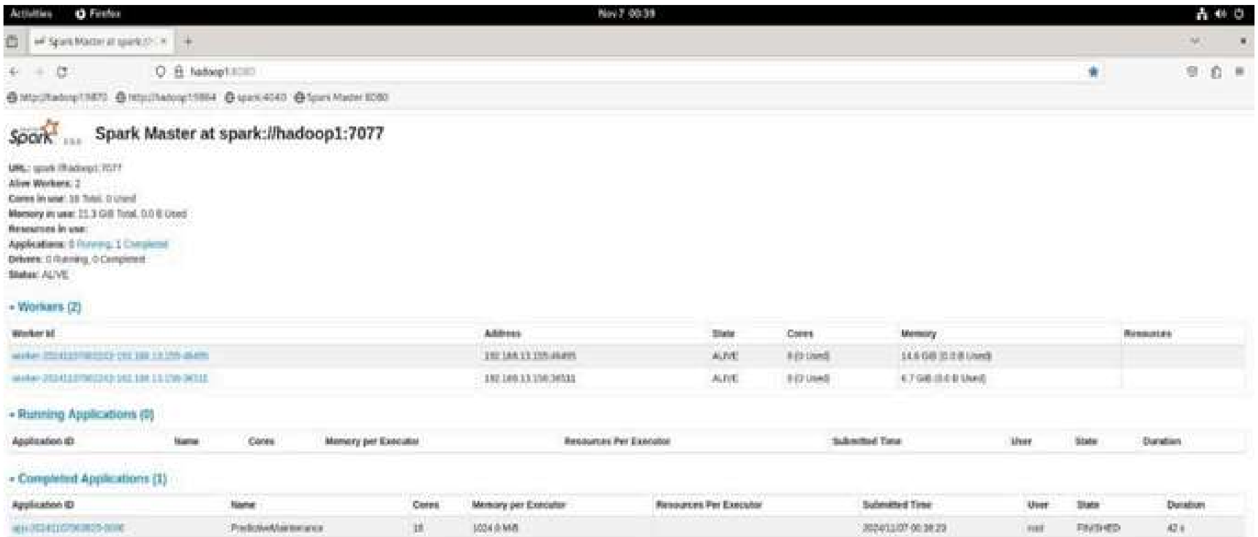
Time consumed when 2 virtual machine was deployed – 30 seconds

Madhumitha Mandyam:



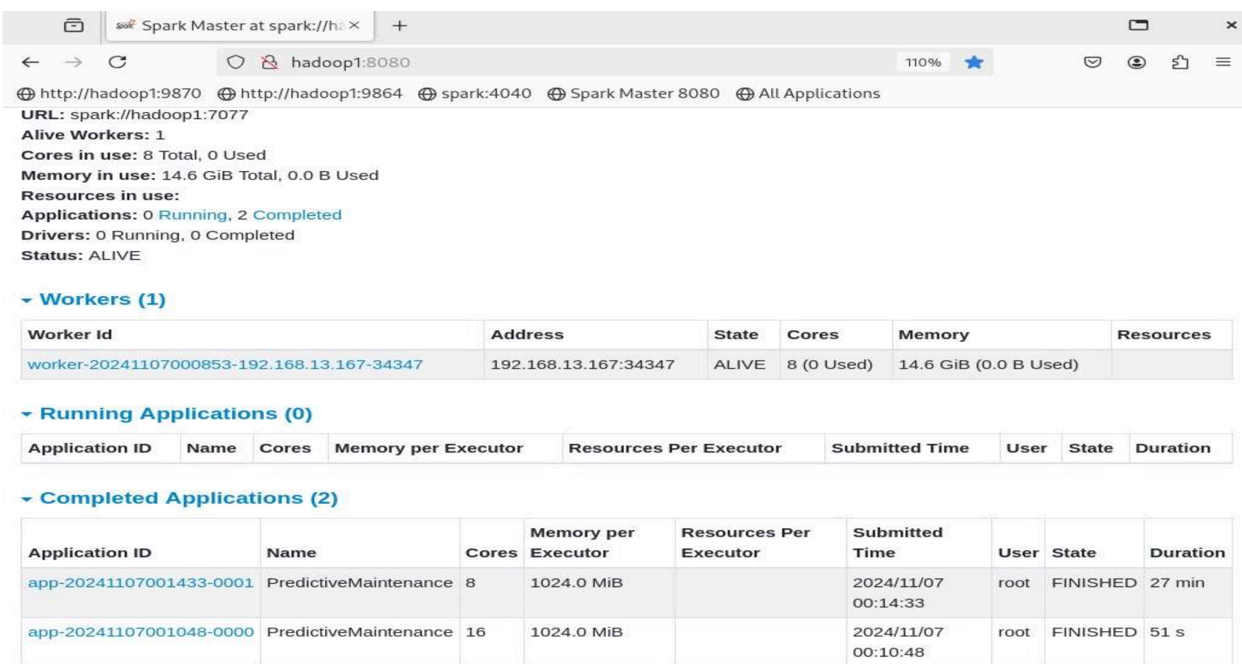
Time consumed when 1 virtual machine was deployed – 39 minutes
Time consumed when 2 virtual machine was deployed – 32 seconds

Nandhika Manikandan:



Time consumed when 1 virtual machine was deployed – 35 minutes
Time consumed when 2 virtual machine was deployed – 42 seconds

Ganesh Vannam:



Time consumed when 1 virtual machine was deployed – 27 minutes

Time consumed when 2 virtual machine was deployed – 51 seconds

Conclusion:

Random Forest and XGBoost outperformed other models in terms of classification metrics, making them ideal for predictive maintenance. Although KNN achieved good results, its slightly lower specificity may affect its suitability. DNN demonstrated high precision and specificity but lower recall, which may reduce its effectiveness for this application.