PREDICTIVE
MAINTENANCE

DEPI GRADUATION PROJECT

"Unlocking the future of efficiency: Discover how predictive maintenance transforms operational challenges into proactive solutions, saving time, money, and resources."

MEET OUR
TEAM

Menna El Sayed

checkout my linkedin

Nourhan Ebrahim

checkout my linkedin

Hanan Elhosary

checkout my linkedin

Mariam Ahmed

checkout my linkedin

AGENDA

1. problem
2. EDA
3. Modeling
4. Key insights
5. Deployment

PROJECT OVERVIEW

Predictive maintenance is a data-driven approach that helps industries predict equipment failures before they occur. The dataset includes sensor readings from machinery under various operating conditions.

The goal is to:

1. Classify machine status as either No Failure or Failure.
2. Predict failures before they occur to enable preventive maintenance.
3. Use sensor data to understand operational conditions leading to failures, improving overall system reliability.

REAL-WORLD IMPACT

- **Reduced Costs:** Predictive maintenance can lower maintenance costs by up to 20%.
- **Increased Equipment Lifespan:** It extends the lifespan of machines by 15-20%, maximizing the value of capital investments.
- **Minimized Downtime:** Reducing unplanned downtimes improves operational efficiency and increases overall productivity.
- **Efficient Resources:** Optimizes repairs and reduces waste by predicting failures.

EDA

Step 1

Data Cleaning

Step 2

Data Transformation

Step 3

Data Exploration

DATA CLEANING

Outliers

- **Issue:** We identified significant outliers in two columns, which could distort the model's accuracy.
- **Action:** Removed these outliers using the Z-score technique, which filters out extreme values that lie beyond the standard deviation, ensuring our model isn't biased by abnormal data points.

```
In [118]: RationalSpeed_bounds=IQR('Rotational speed')
          RationalSpeed_bounds
          outliers=data[(data['Rotational speed'] < RationalSpeed_bounds[0]) | (data['Rotational speed'] > RationalSpeed_bounds[1])]
          print("Number of outliers: ",outliers.shape[0],'\n')
          outliers.head()

          Number of outliers:  418
```

Out[118]:

	Type	Air temperature	Process temperature	Rotational speed	Torque	Tool wear	Target	Index
14	L	298.6	309.2	2035	19.6	49	0	14
50	L	298.9	309.1	2861	4.6	143	1	50
70	M	298.9	309.0	1924	22.6	193	0	70
101	L	298.8	308.8	1991	20.7	59	0	101
155	H	298.4	308.2	1997	19.8	198	0	155

```
In [119]: Torque_bounds=IQR('Torque')
          Torque_bounds
          outliers=data[(data['Torque'] < Torque_bounds[0]) | (data['Torque'] > Torque_bounds[1])]
          print("Number of outliers: ",outliers.shape[0],'\n')
          outliers.head()

          Number of outliers:  69
```

Out[119]:

	Type	Air temperature	Process temperature	Rotational speed	Torque	Tool wear	Target	Index
50	L	298.9	309.1	2861	4.6	143	1	50
194	M	298.2	308.5	2678	10.7	86	1	194
380	L	297.5	308.3	2564	12.8	127	1	380
463	L	297.4	308.7	2874	4.2	118	1	463
663	L	297.9	309.8	1336	71.6	31	1	663

```
In [ ]:
```

```
In [203]: data = data[(stats.zscore(data['Rotational speed']) < 1.9)]
          data  = data[(stats.zscore(data['Torque']) < 2.8)]
          data.reset_index(drop=True,inplace=True)
```

```
In [ ]:
```

DATA CLEANING

Columns Removal:

“UDI” and “Product ID” Columns:

- These columns are identifiers and don't carry any meaningful information for predictive modeling.

Renaming Columns

- To enhance clarity and efficiency during analysis, we renamed columns to more intuitive and readable names.

DATA TRANSFORMATION

we used Label Encoder to convert Categorical features to numerical values .
our categorical feature is “Type”
this transformation allows machine learning model to effictively interpret categorical data .

"We standardized our data to ensure that all features contribute equally to the analysis, allowing us to better identify patterns and relationships in the dataset, which is crucial for accurate predictive maintenance modeling."

#categorical conversion using "Label Encoder" for "Type" column

```
In [198]: le = LabelEncoder()
          data['Type'] = le.fit_transform(data['Type'])

In [199]: data['Type']

Out[199]: 0       2
          1       1
          2       1
          3       1
          4       1
                 ..
          9995    2
          9996    0
          9997    2
          9998    0
          9999    2
          Name: Type, Length: 10000, dtype: int32
```

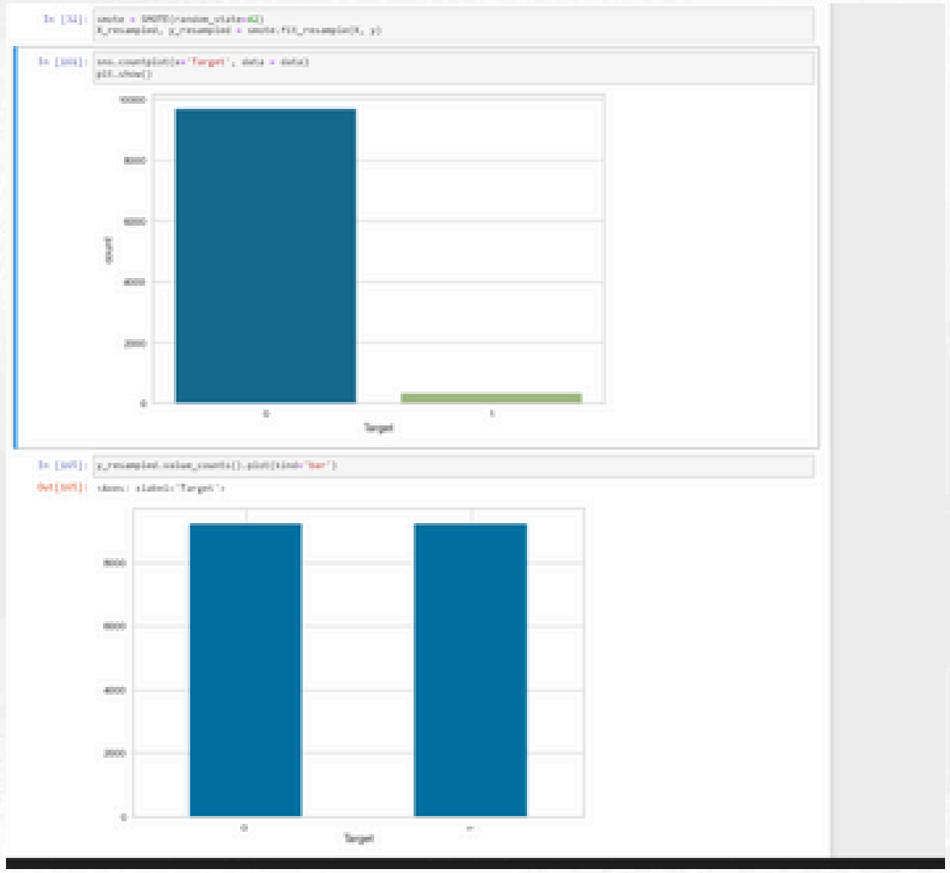
#--standardization for data without (target and Type)

```
In [200]: sc = StandardScaler()
          to_scale = ["Air temperature", "Process temperature", "Rotational speed", "Torque", "Tool wear"]
          data[to_scale] = sc.fit_transform(data[to_scale])
```

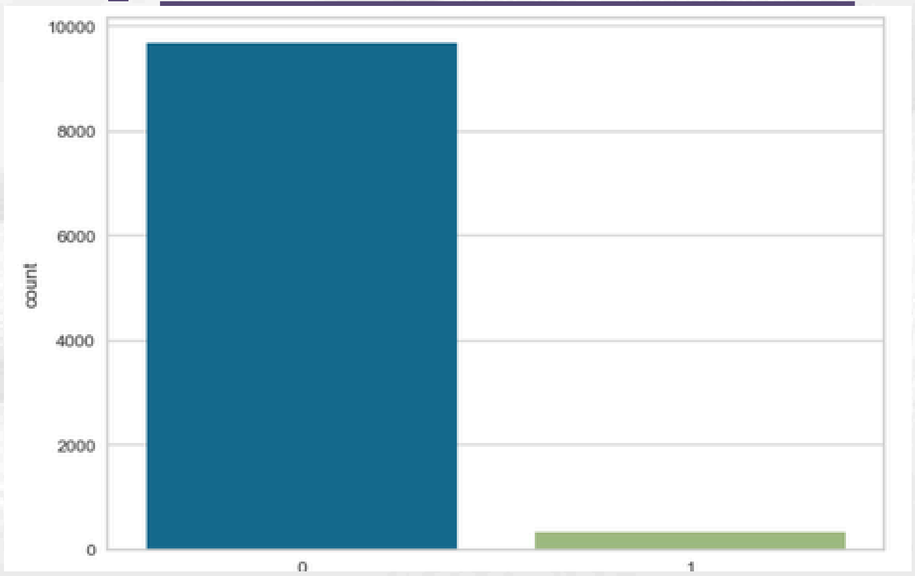
DATA TRANSFORMATION

"We applied PCA to reduce the dimensionality of our dataset, enabling us to extract the most significant features while retaining the essential information, which facilitates improved visualization and enhances the performance of our predictive maintenance models."

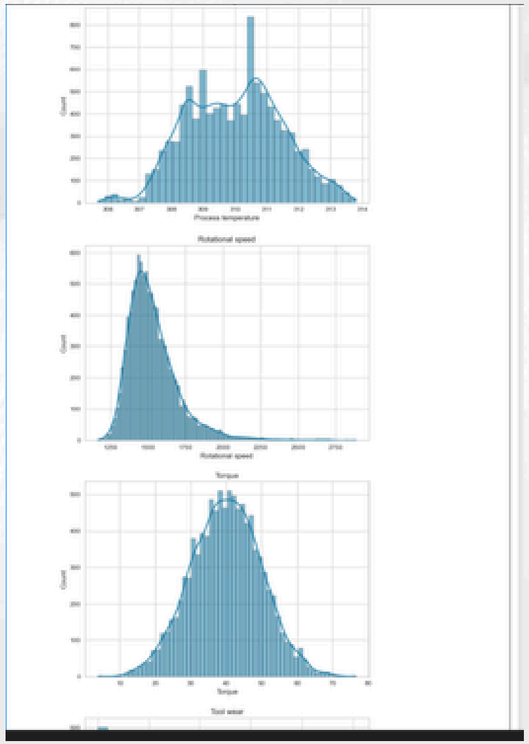
"We applied SMOTE to address the class imbalance in our dataset, generating synthetic examples for the minority class to enhance the model's ability to learn from all classes effectively, thereby improving the accuracy and robustness of our predictive maintenance outcomes."



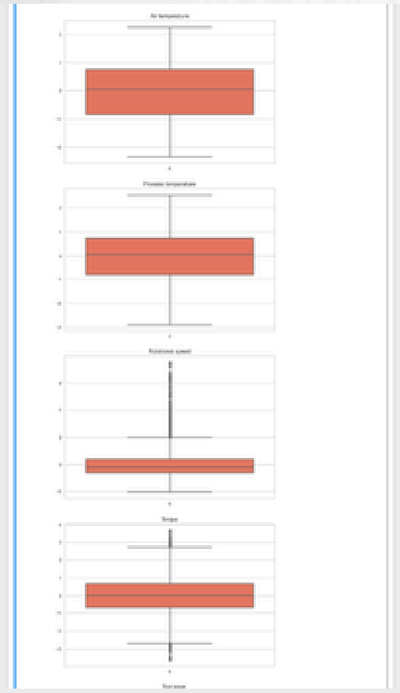
DATA EXPLORATION



we find that our target have imbalnce as the 0 class way higher than the class 1.

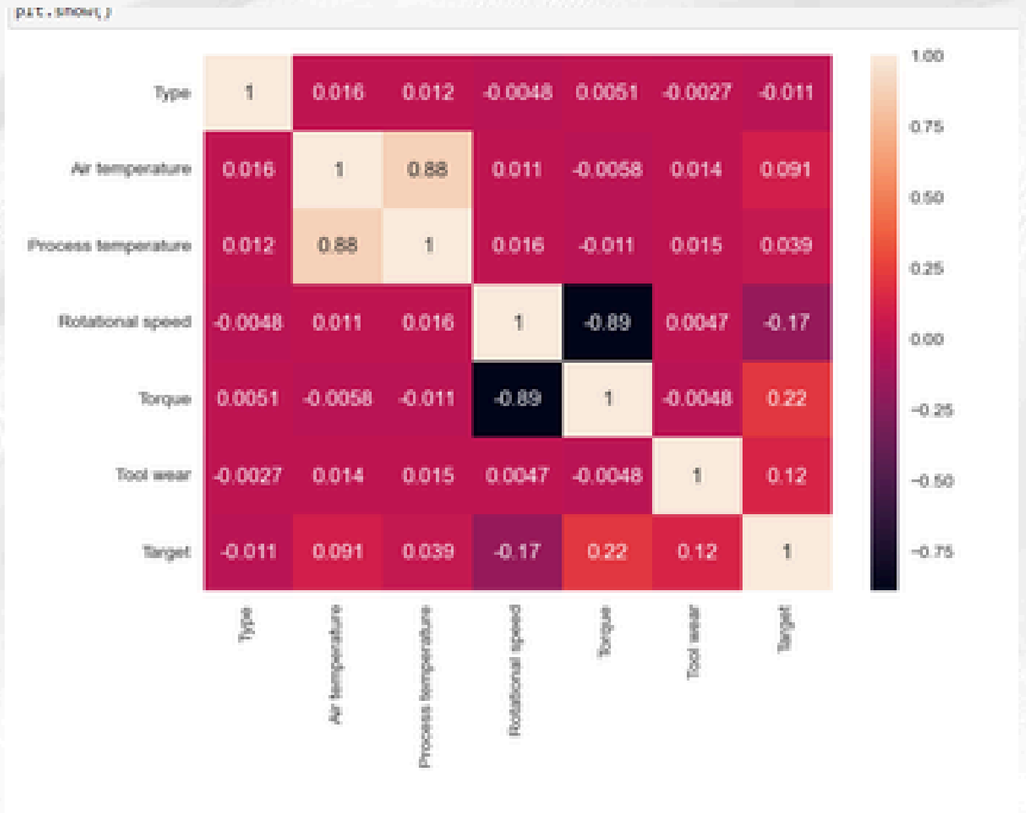


We explored the distribution and density of our data through histplotting our data .



we found out that we have outliers i 2 columns through this boxplot .

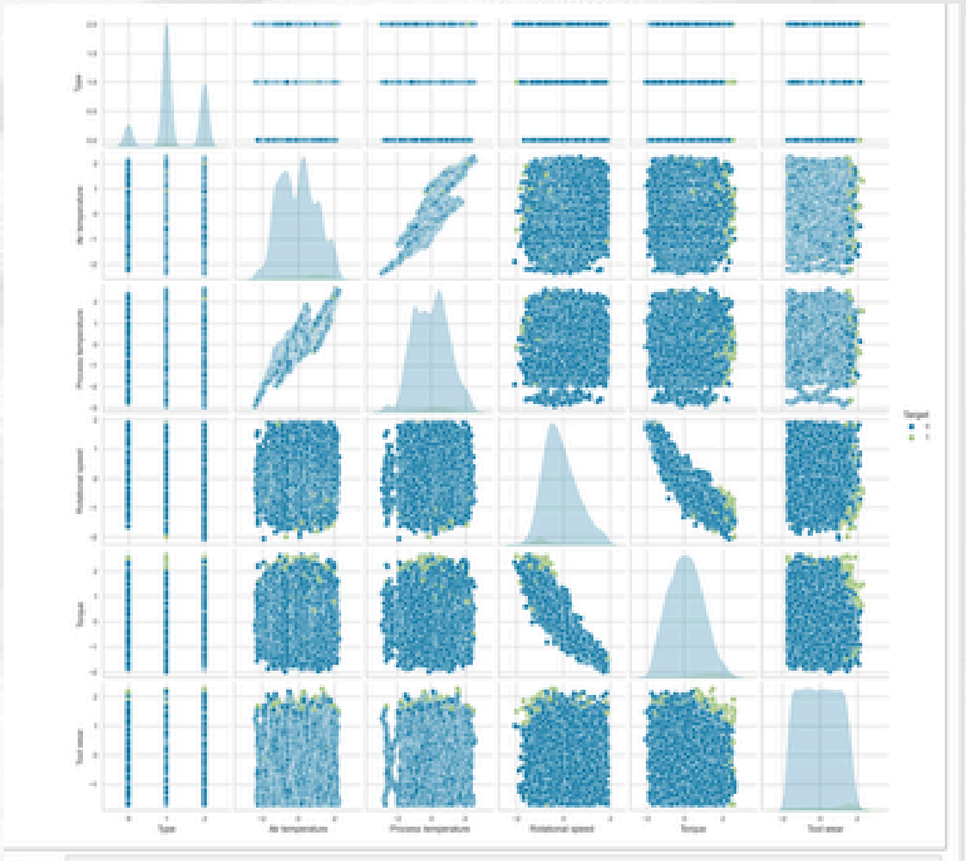
DATA EXPLORATION



This heatmap illustrates the correlation coefficients between the key features in our dataset. The values range from -1 to 1, where:

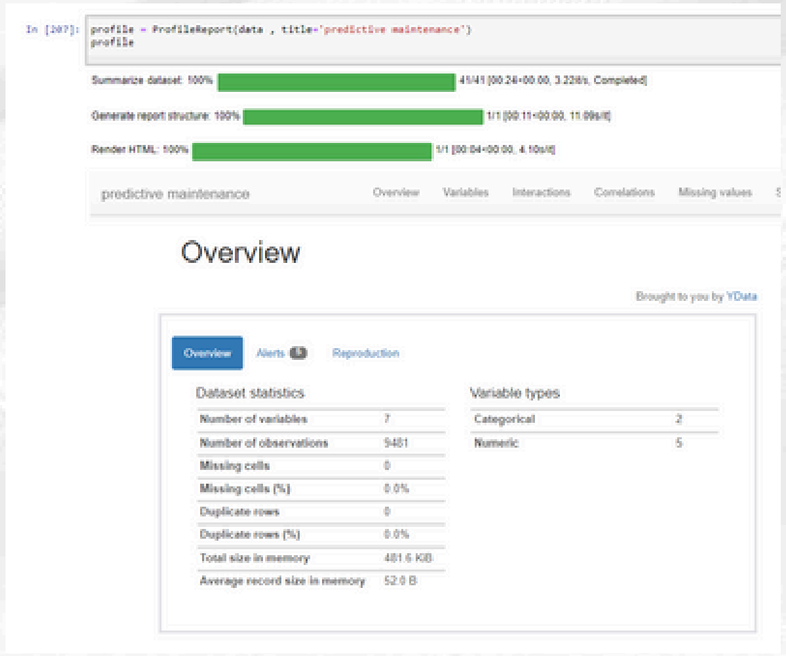
- Air temperature and Process temperature show a strong positive correlation (0.88). This indicates that these two variables tend to increase or decrease together.
- Rotational speed and Torque are strongly negatively correlated (-0.89). As the speed increases, the torque decreases, and vice-versa.
- The Target (indicating machine failure or the prediction target) has moderate positive correlations with:
- Torque (0.22): Higher torque values slightly relate to a higher chance of failure.
- Tool wear (0.12): More tool wear has a mild positive relation with the target.
- Air temperature also shows a small but positive correlation with the target (0.09).

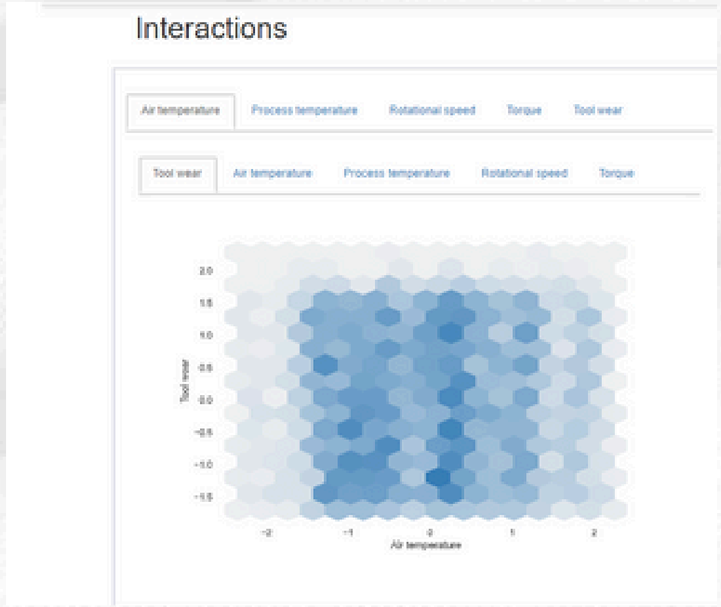
DATA EXPLORATION

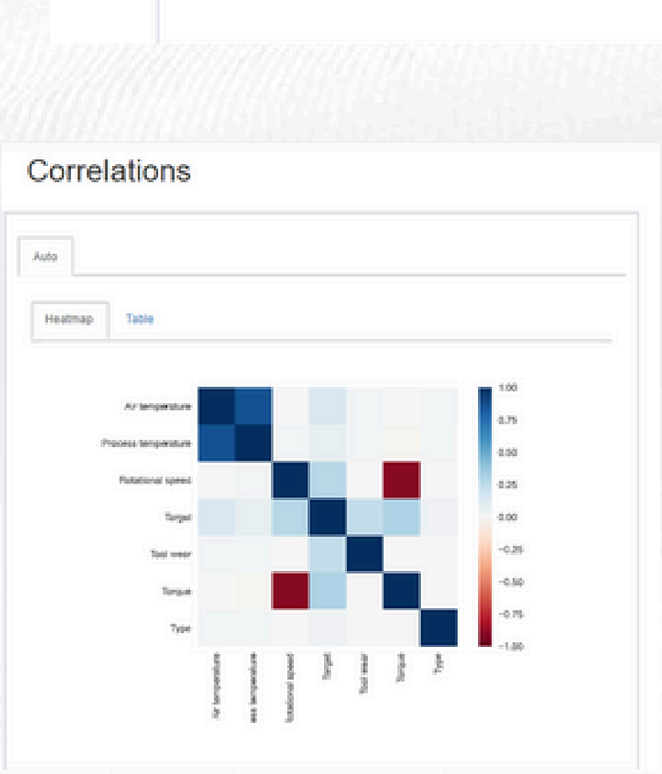


"To gain a deeper understanding of the relationships between our variables and the target variable, we created a pair plot.
This visualization allowed us to simultaneously explore the distribution of each variable and the pairwise relationships between them.
By examining the scatter plots and distributions, we were able to identify potential correlations, patterns, and trends that might be relevant to our predictive modeling efforts."

DATA EXPLORATION







We used ProfileReport to generate a comprehensive exploratory data analysis (EDA) report, which provided key insights into the dataset, including
summary statistics, data types, missing values, correlations, distributions, and potential outliers.

MODELING

Support Vector Classifier

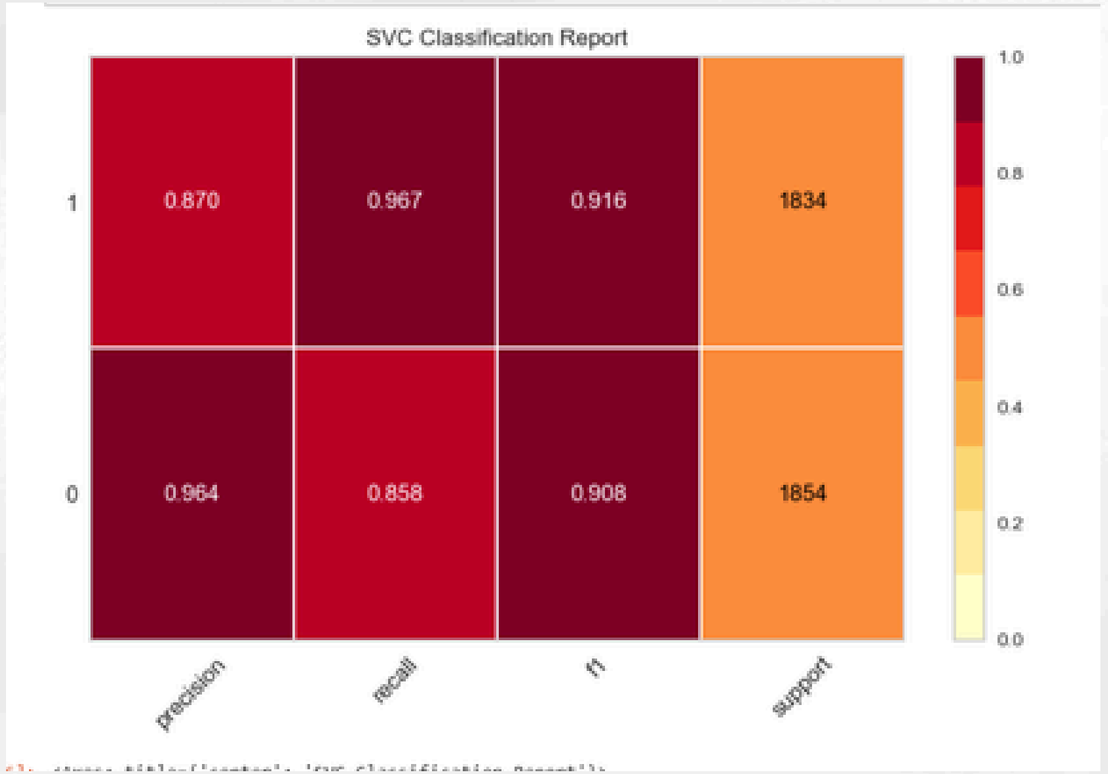
```
Training Accuracy    : 91.31 %
Model Accuracy Score : 91.21 %
-------------------------------------------------------
Classification_Report:
              precision    recall  f1-score   support

           0       0.96      0.86      0.91      1854
           1       0.87      0.97      0.92      1834

    accuracy                           0.91      3688
   macro avg       0.92      0.91      0.91      3688
weighted avg       0.92      0.91      0.91      3688

-------------------------------------------------------
```

91.2%



MODELING

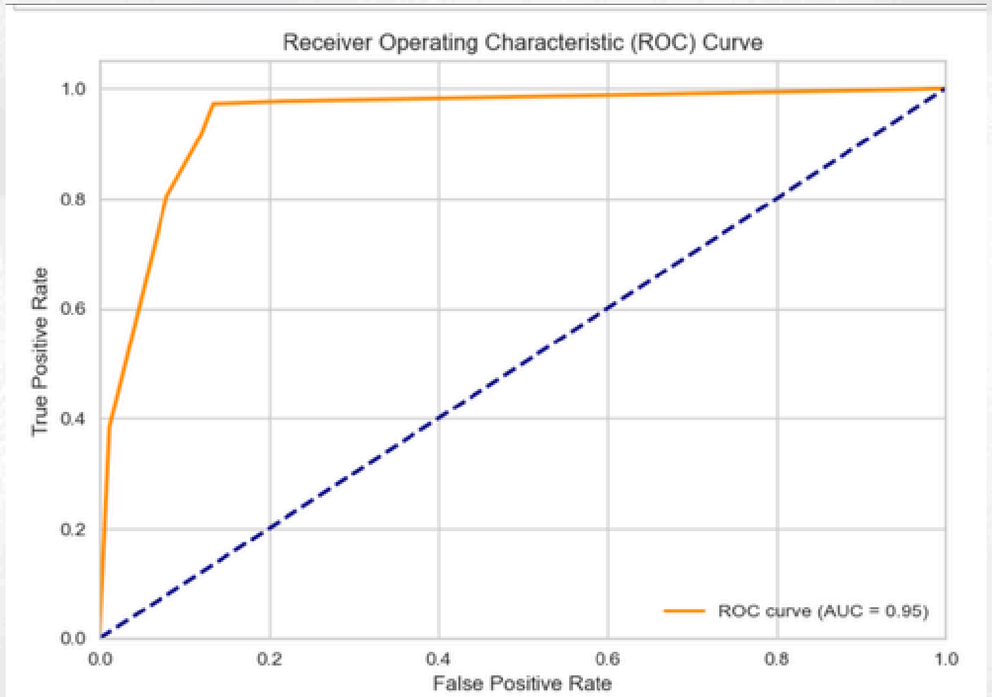
Decision Tree Classifier

```
Training Accuracy    : 91.72 %
Model Accuracy Score : 91.89 %
-----------------------------------------------------
Classification_Report:
              precision    recall  f1-score   support

           0       0.97      0.87      0.91      1854
           1       0.88      0.97      0.92      1834

    accuracy                           0.92      3688
   macro avg       0.92      0.92      0.92      3688
weighted avg       0.92      0.92      0.92      3688

-----------------------------------------------------
```



92%

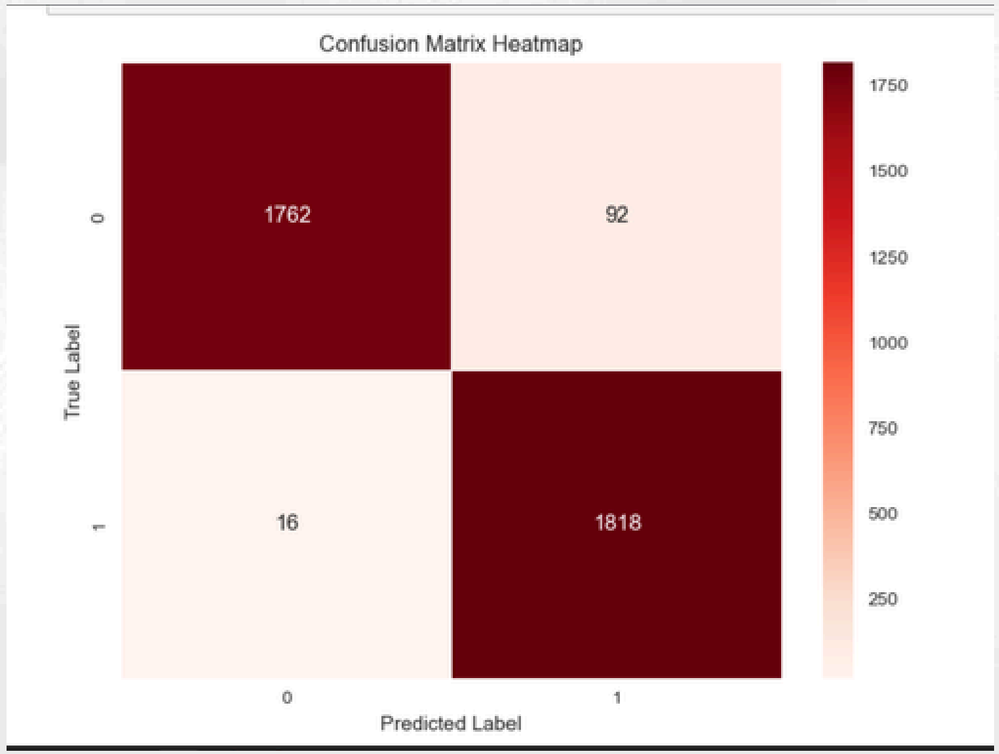
MODELING

Random Forest Classifier

```
Training Accuracy    : 91.72 %
Model Accuracy Score : 91.89 %
-------------------------------------------------------
Classification_Report:
              precision    recall  f1-score   support

           0       0.97      0.87      0.91      1854
           1       0.88      0.97      0.92      1834

    accuracy                           0.92      3688
   macro avg       0.92      0.92      0.92      3688
weighted avg       0.92      0.92      0.92      3688
```



91.89%

MODELING

Hyperparameter tunning

we applied BayesSearchCV with random forest to improve its effitiency and robustness

also applied 5 fold crossvalidation to avoid the overfitting .

then we tried to get the best hyper parameters to train the model using baysian optimization .

finally, we trained the model at those best parameters .

```
Fitting 5 folds for each of 1 candidates, totalling 5 fits
Fitting 5 folds for each of 1 candidates, totalling 5 fits
Fitting 5 folds for each of 1 candidates, totalling 5 fits
Fitting 5 folds for each of 1 candidates, totalling 5 fits
Fitting 5 folds for each of 1 candidates, totalling 5 fits
Fitting 5 folds for each of 1 candidates, totalling 5 fits
Fitting 5 folds for each of 1 candidates, totalling 5 fits
Fitting 5 folds for each of 1 candidates, totalling 5 fits
Fitting 5 folds for each of 1 candidates, totalling 5 fits
Fitting 5 folds for each of 1 candidates, totalling 5 fits
Fitting 5 folds for each of 1 candidates, totalling 5 fits
Best parameters found:  OrderedDict([('max_depth', 24), ('max_features', 'sqrt'), ('min_samples_leaf', 1), ('min_samples_spli
t', 2), ('n_estimators', 500)])
```

```
In [70]: rf_best = RandomForestClassifier(**best_params, random_state=42)
         rf_best.fit(X_train, y_train)
```

```
Out[70]: RandomForestClassifier(max_depth=24, n_estimators=500, random_state=42)
```

MODELING

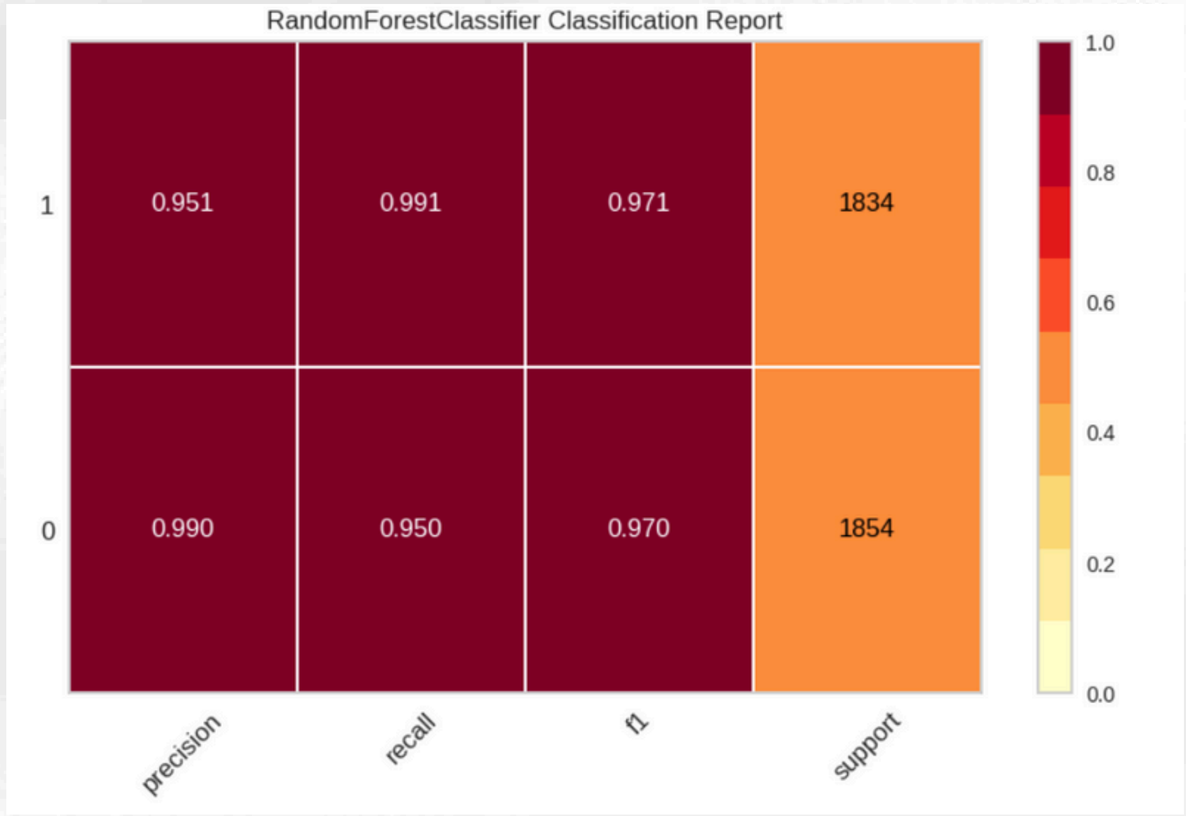
Random Forest Classifier after hyperparameter tuning

```
Training Accuracy    : 100.0 %
Model Accuracy Score : 97.02 %
-----------------------------------------------------
Classification_Report:
              precision    recall  f1-score   support

           0       0.99      0.95      0.97      1854
           1       0.95      0.99      0.97      1834

    accuracy                           0.97      3688
   macro avg       0.97      0.97      0.97      3688
weighted avg       0.97      0.97      0.97      3688

-----------------------------------------------------
```

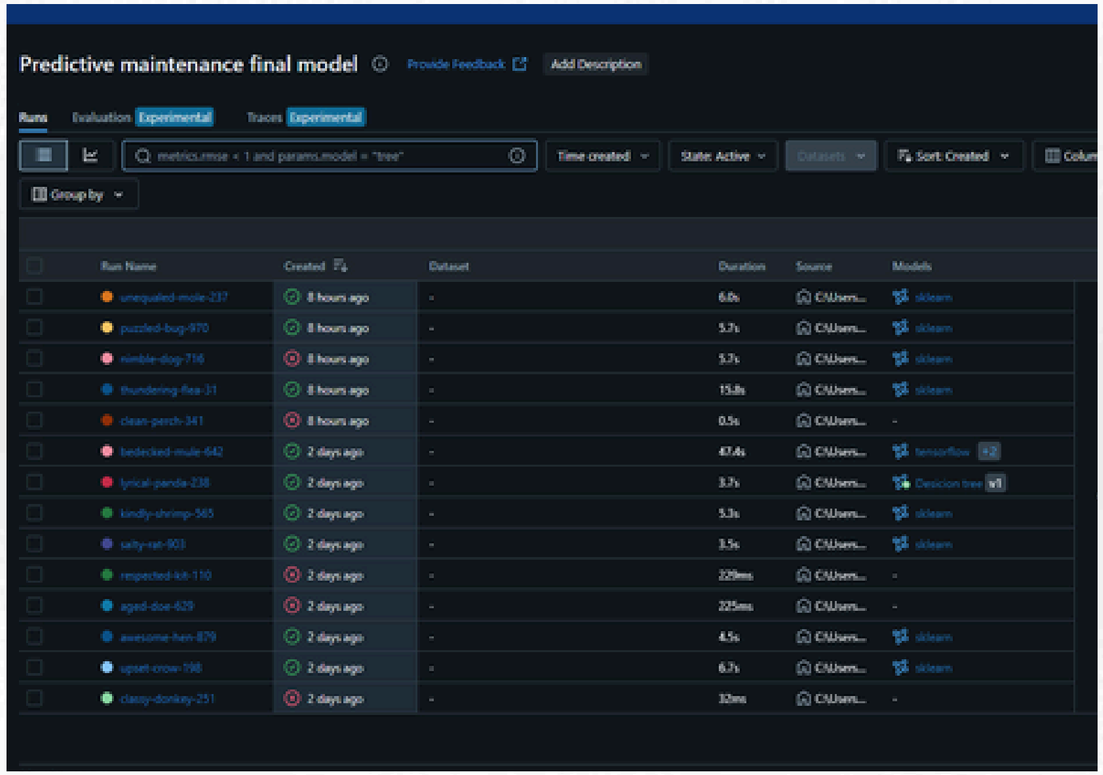


97.02%

mlflow™

WHY??

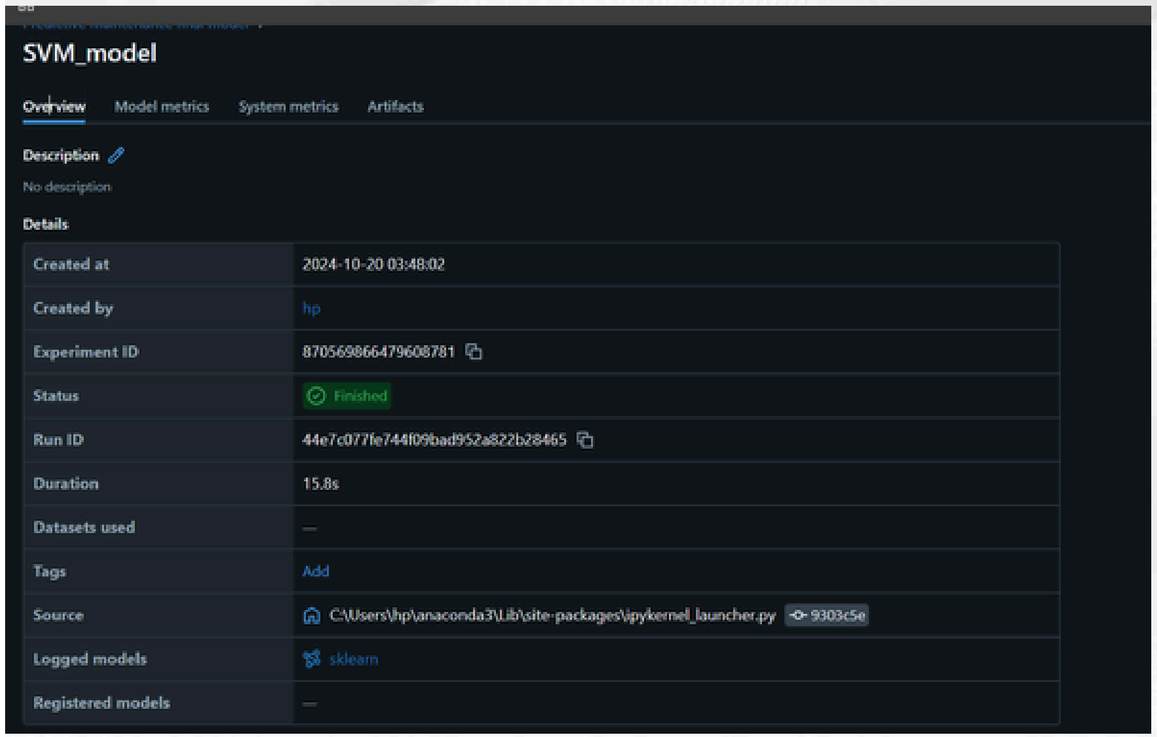
we used it to monitor the perfomance of machine learning models during training , tracking metrics like accuracy , AUC-ROC curve , confusion matrix for models such as SVM , DT and Random forest .
It helped at visualizing improvement and optimizing hyperparameters effictively .



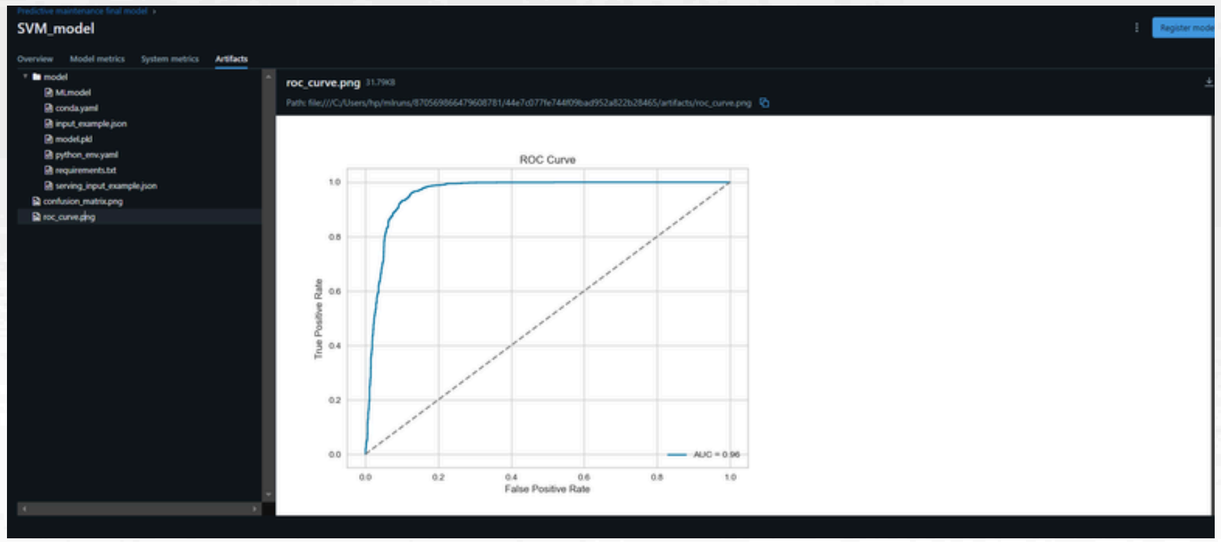
HOW??

- Logged accuracy ,AUC-ROC curve and confusion metrics for models .
- Tracked and visualized the best hyperparameter during model tuning.

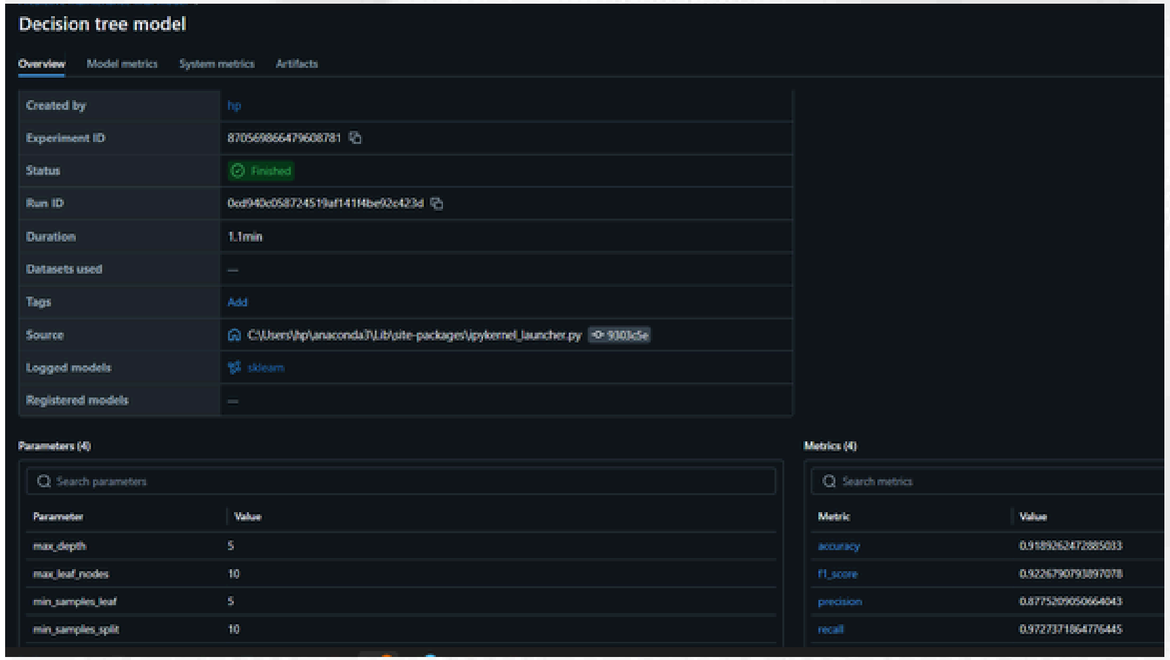
MLFLOW & SVM

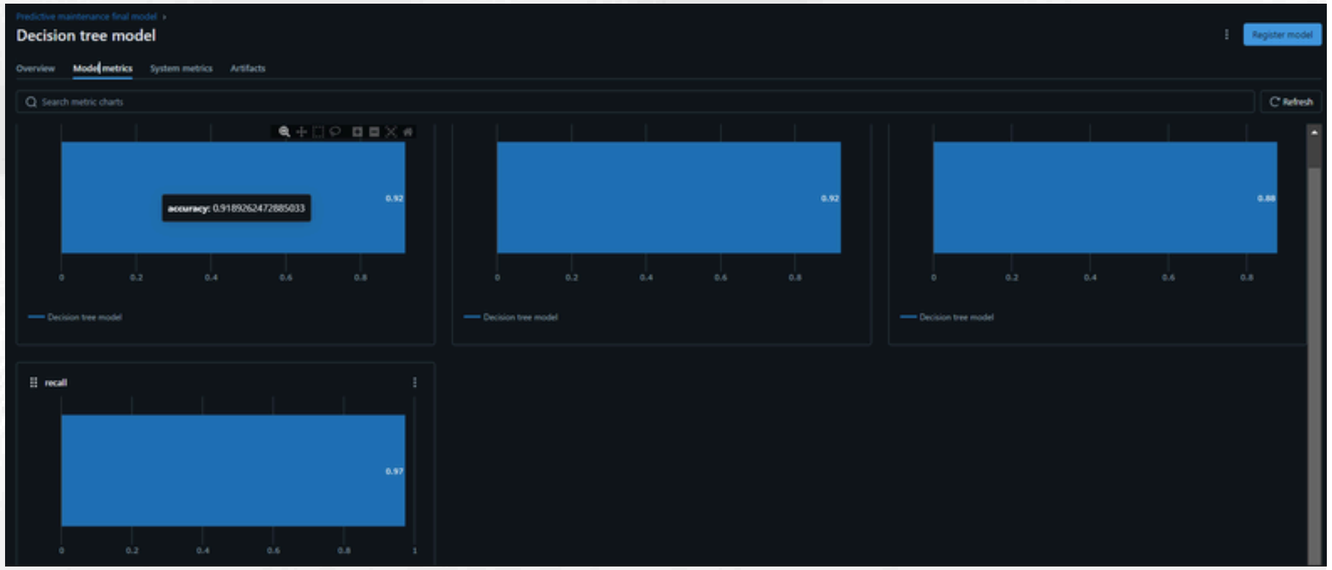

SVM_model
Overview
Model metrics
System metrics
Artifacts
Description
No description
Details
Created at
2024-10-20 03:48:02
Created by
hp
Experiment ID
870569866479608781
Status
Finished
Run ID
44e7c077fe744f09bad952a822b28465
Duration
15.8s
Datasets used
—
Tags
Add
Source
C:\Users\hp\anaconda3\Lib\site-packages\ipykernel_launcher.py
9303c5e
Logged models
sklearn
Registered models
—

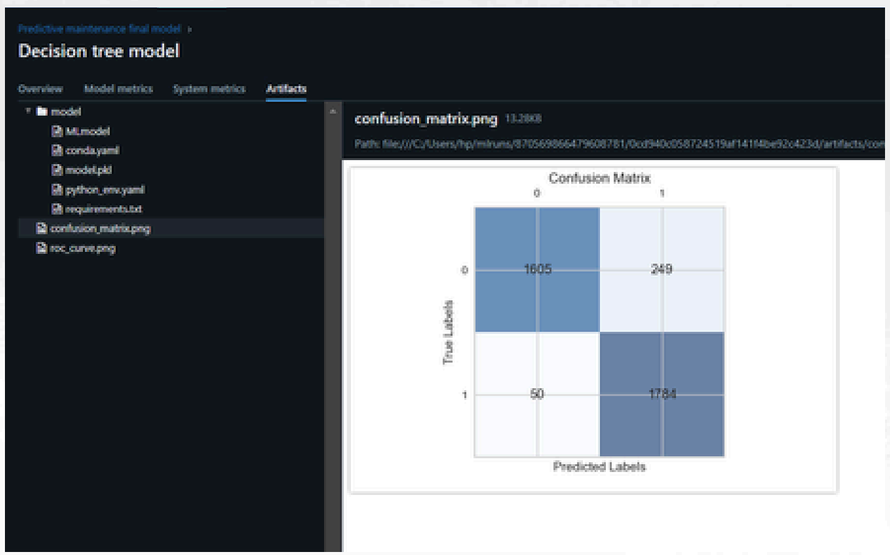

SVM_model
Overview
Model metrics
System metrics
Artifacts
accuracy
f1_score
precision
recall
roc_auc
f1_score: 0.9163223140495869


SVM_model
Overview
Model metrics
System metrics
Artifacts
model
MLmodel
conda.yaml
input_example.json
model.pkl
python_env.yaml
requirements.txt
serving_input_example.json
confusion_matrix.png
roc_curve.png
roc_curve.png
ROC Curve
True Positive Rate
False Positive Rate
AUC = 0.96

MLFLOW & DT

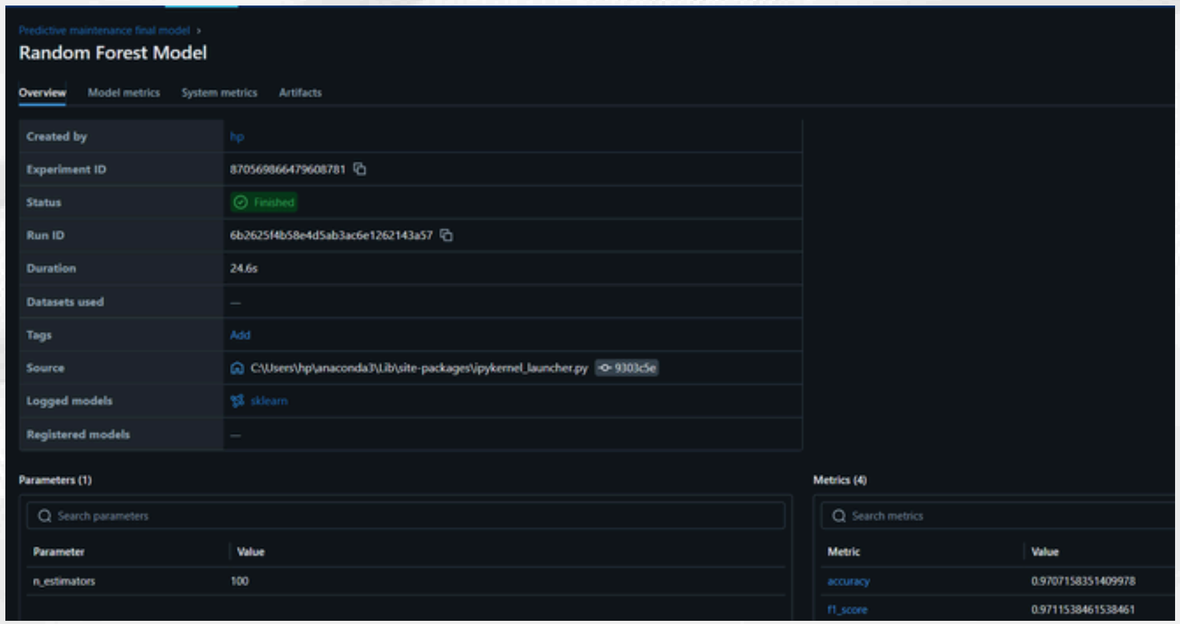

Decision tree model
Overview
Model metrics
System metrics
Artifacts
Created by
hp
Experiment ID
870569866479608781
Status
Finished
Run ID
0cd940c058724519af141f4be92c423d
Duration
1.1min
Datasets used
—
Tags
Add
Source
C:\Users\hp\anaconda3\Lib\site-packages\ipykernel_launcher.py
Logged models
sklearn
Registered models
—
Parameters (4)
Search parameters
Parameter
Value
max_depth
5
max_leaf_nodes
10
min_samples_leaf
5
min_samples_split
10
Metrics (4)
Search metrics
Metric
Value
accuracy
0.9189262472885033
F1_score
0.9226790793897078
precision
0.8775269050664843
recall
0.9727371864776445

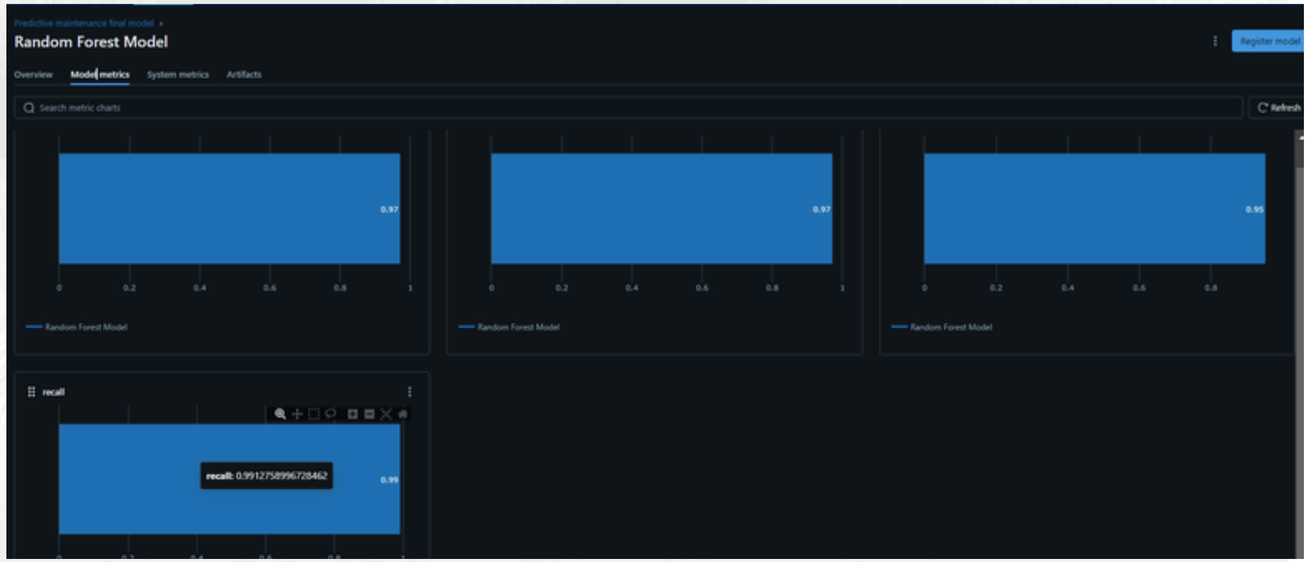

Predictive maintenance final model
Decision tree model
Register model
Overview
Model metrics
System metrics
Artifacts
Search metric charts
Refresh
accuracy: 0.9189262472885033
0.92
0.92
0.88
Decision tree model
recall
0.97

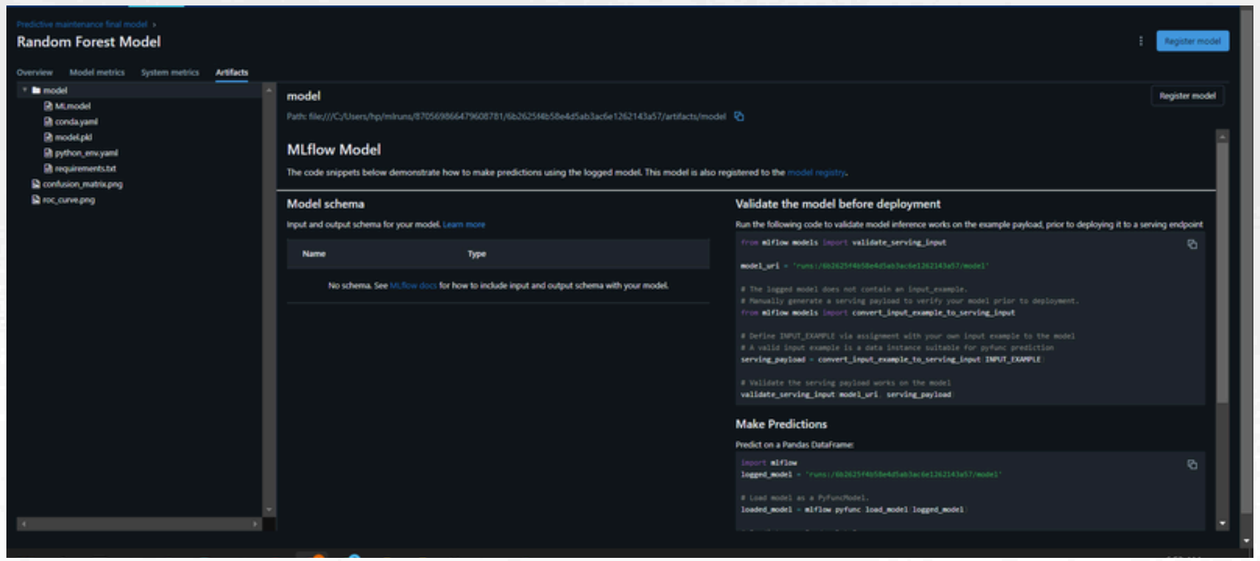

Predictive maintenance final model
Decision tree model
Overview
Model metrics
System metrics
Artifacts
model
MLmodel
conda.yaml
model.pkl
python_env.yaml
requirements.txt
confusion_matrix.png
roc_curve.png
confusion_matrix.png
13.28KB
Confusion Matrix
True Labels
Predicted Labels
1605
249
50
1784

mlflow™

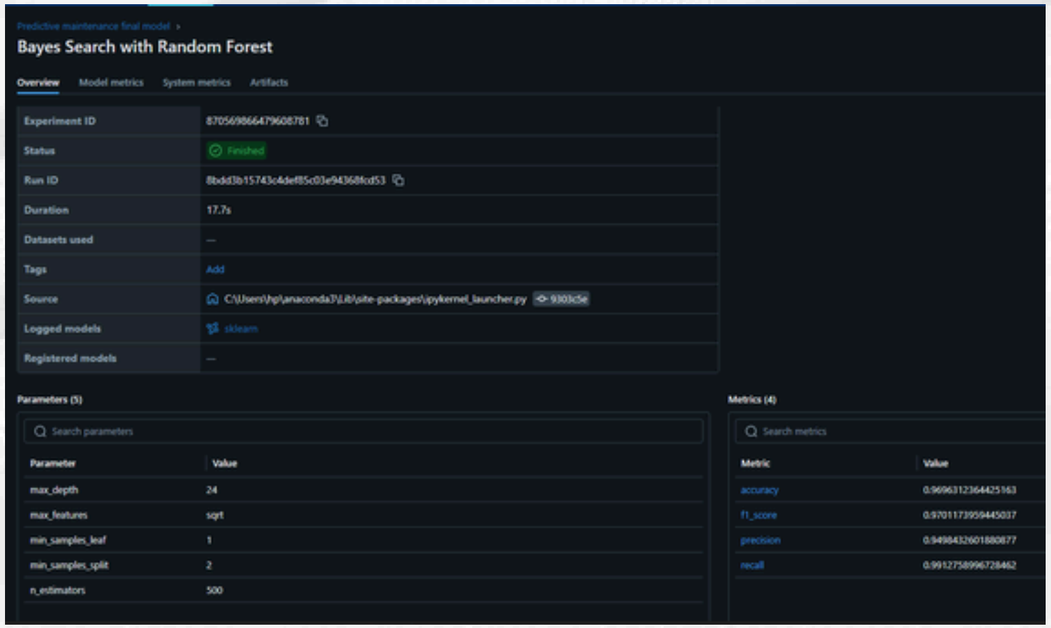
MLFLOW & RF

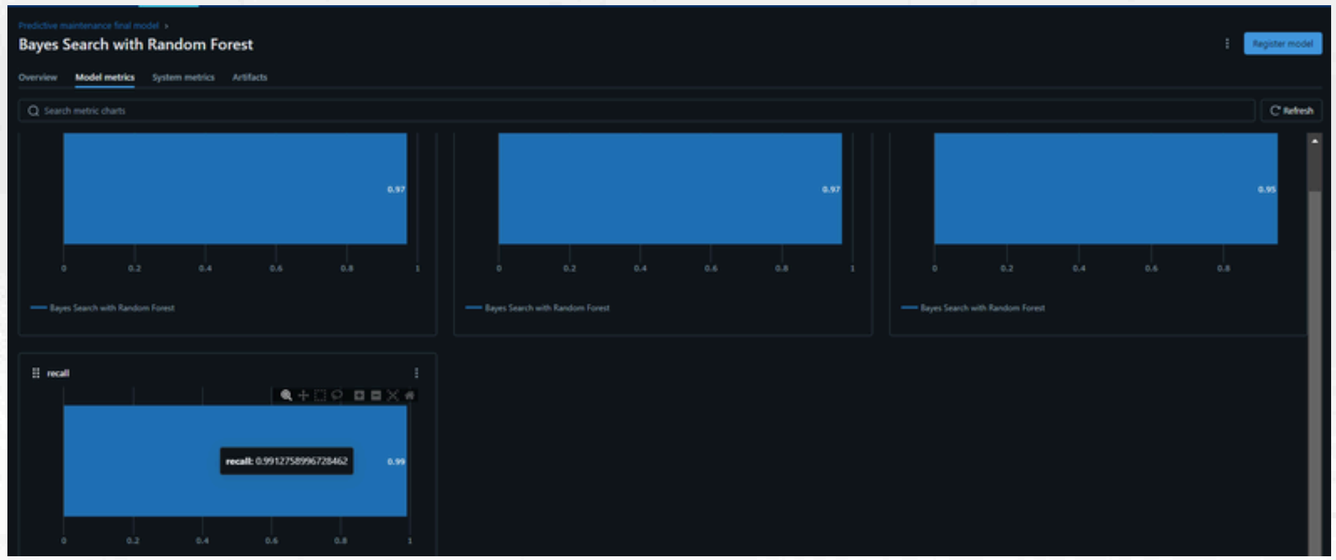






MLFLOW & RF ENHANCED BY BAYES SEARCH





GANS

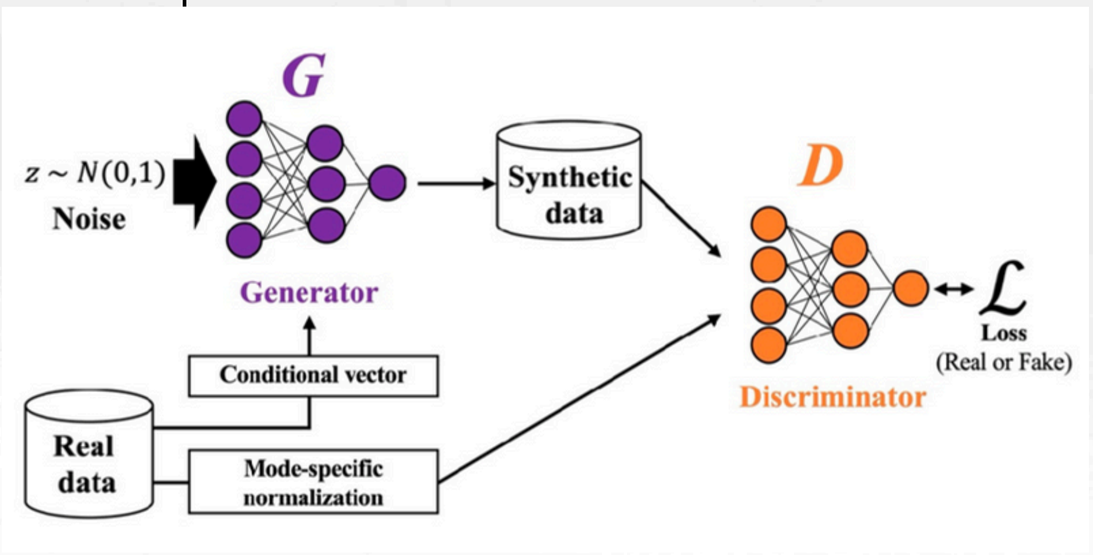
USING GANS FOR TABULAR DATA GENERATION AND MODEL IMPLEMENTATION

THE SYNTHETIC DATA VAULT

CTGAN

Generative Adversarial Networks (GANs) have traditionally been applied to tasks like image synthesis, but recent advancements have extended their use to tabular data generation. GANs offer a powerful way to generate synthetic tabular data that retains statistical properties of the original dataset while avoiding privacy risks and data limitations. In this process, the generator learns to produce realistic, synthetic samples that resemble the original data, while the discriminator attempts to distinguish between real and synthetic samples.

In our project, we used CTGAN's built-in library to generate synthetic tabular data that preserves the statistical properties of the original dataset. The CTGAN model leverages a tailored architecture to deal with the diversity and sparsity found in real-world tabular data, allowing us to produce high-quality synthetic samples.



GANS

USING GANS FOR TABULAR DATA GENERATION AND MODEL IMPLEMENTATION

THE SYNTHETIC DATA VAULT

CTGAN

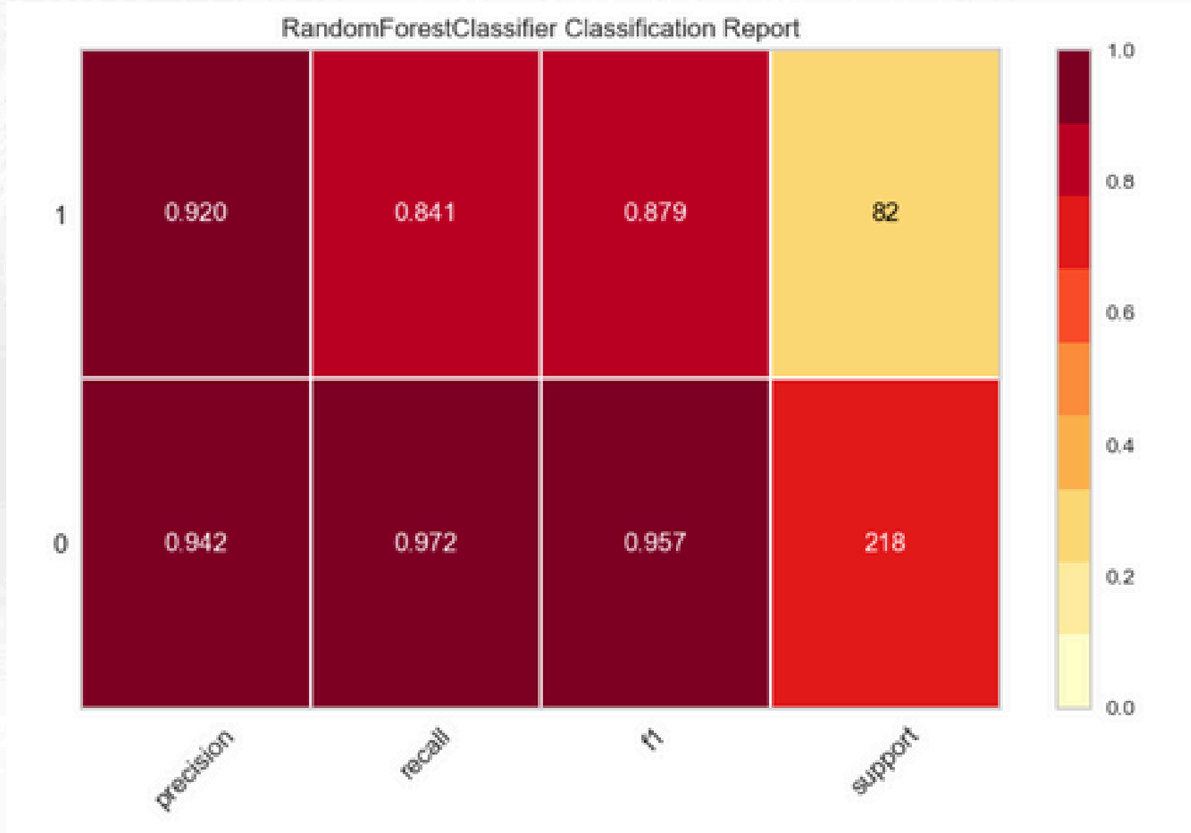
After generating the data, we tested its usability by training a Random Forest classifier to predict logical labels derived from the generated features.

```
Training Accuracy    : 1.0 %
Model Accuracy Score :0.94 %
-----------------------------------------------------
Classification_Report:
                precision    recall  f1-score   support

           0       0.94      0.97      0.96       218
           1       0.92      0.84      0.88        82

    accuracy                           0.94       300
   macro avg       0.93      0.91      0.92       300
weighted avg       0.94      0.94      0.94       300

-----------------------------------------------------
```



AZURE SIMULATION

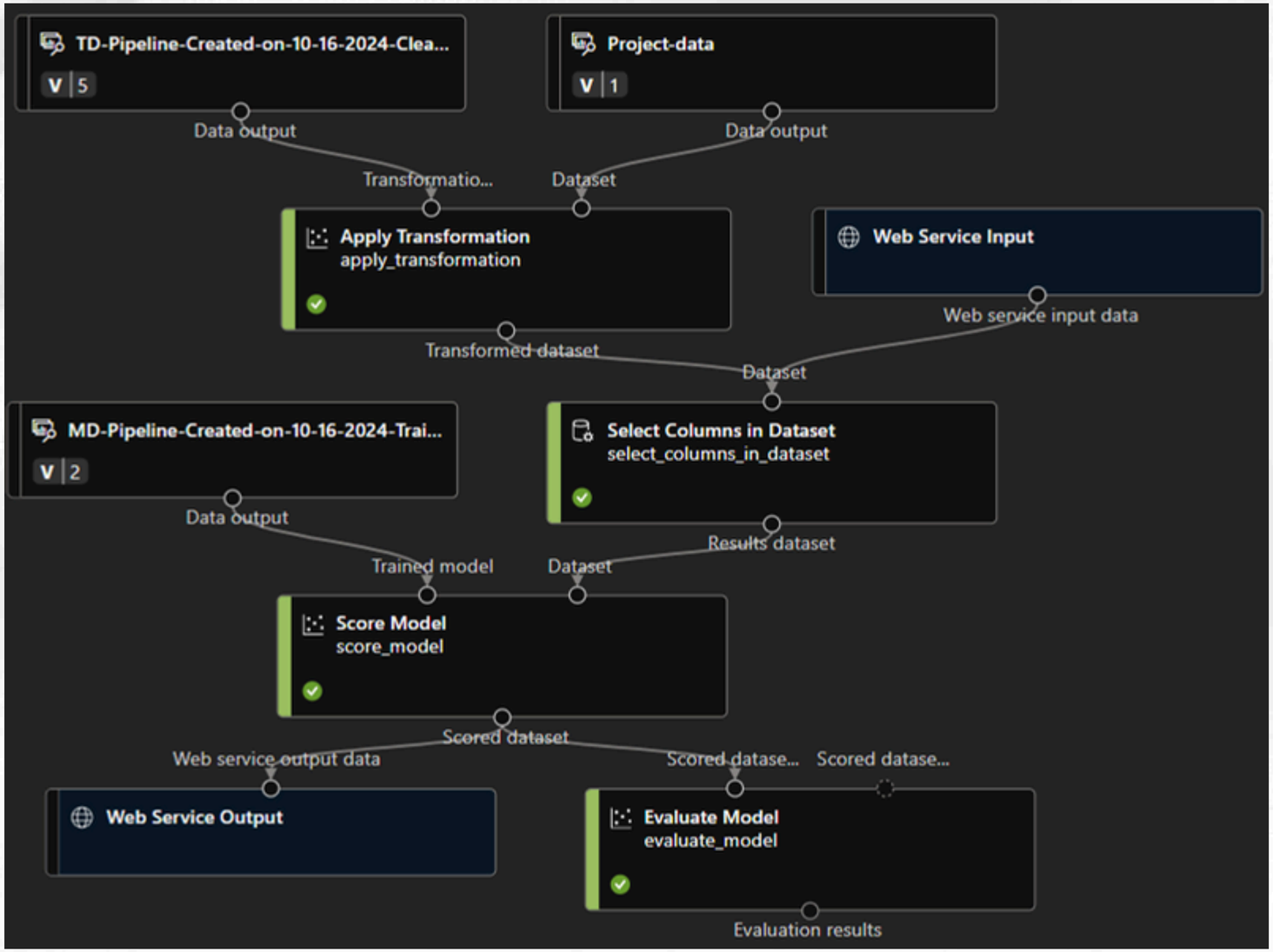
DESIGN AND WORKFLOW.

In this phase, we simulate a real project in azure machine learning studio following these key stages:

1. **Data Upload:** Import the dataset into the Azure environment.
2. **Data Preprocessing:** Prepare and clean the data to ensure it's ready for model development as the real model.
   - Data transformation
   - removed “UDI and product id” columns cause we don’t it at out analytics .
3. **Model Training:** Use the processed data to train a machine learning model we use here two-class decision tree model.
4. **Model Scoring & Evaluation:** Assess the model's performance with scoring and evaluation metrics.

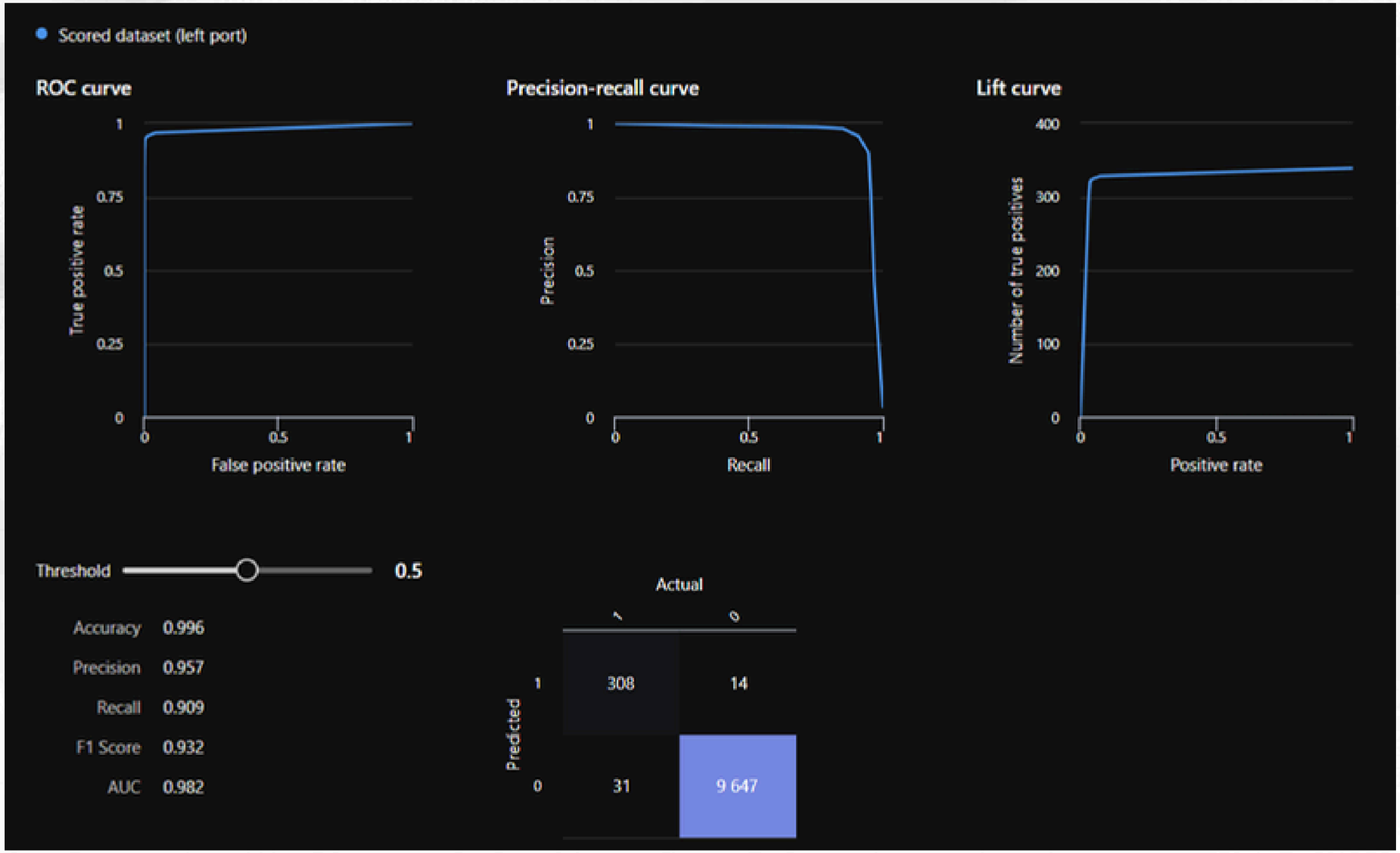
AZURE SIMULATION

DESIGN AND WORKFLOW.



AZURE SIMULATION

SCORING & EVALUATION.

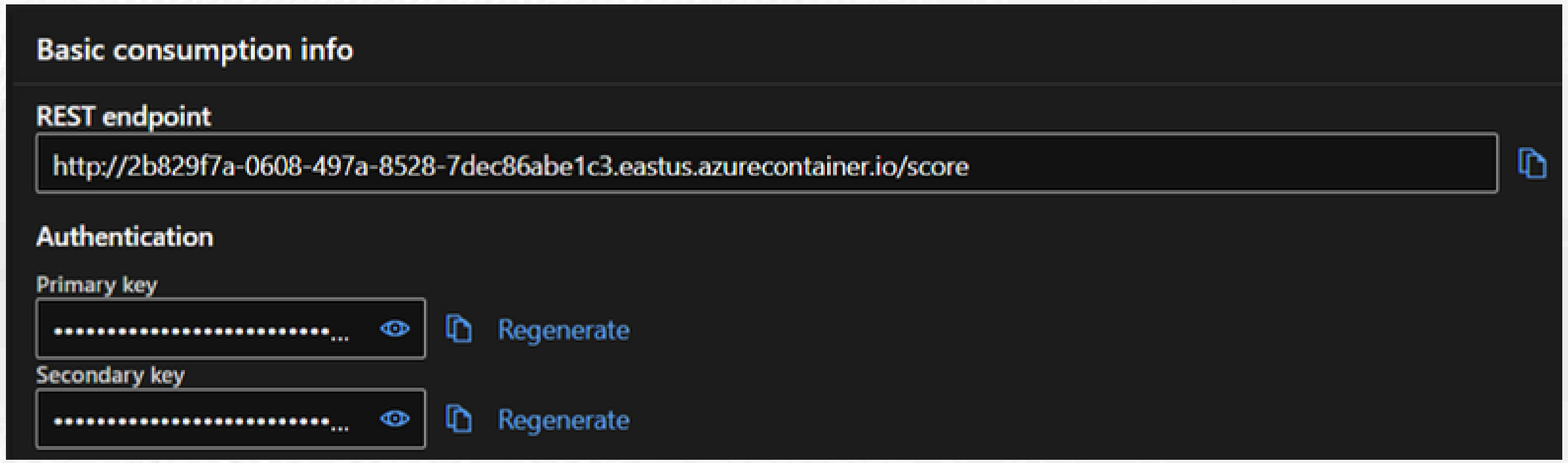


AZURE SIMULATION

DEPLOYMENT.

In this phase, Deploy the fully trained model for live use in real-world applications.

submitting the job in Azure, we deploy the model, and a REST endpoint is automatically created, enabling real-time predictions and interaction with the model.



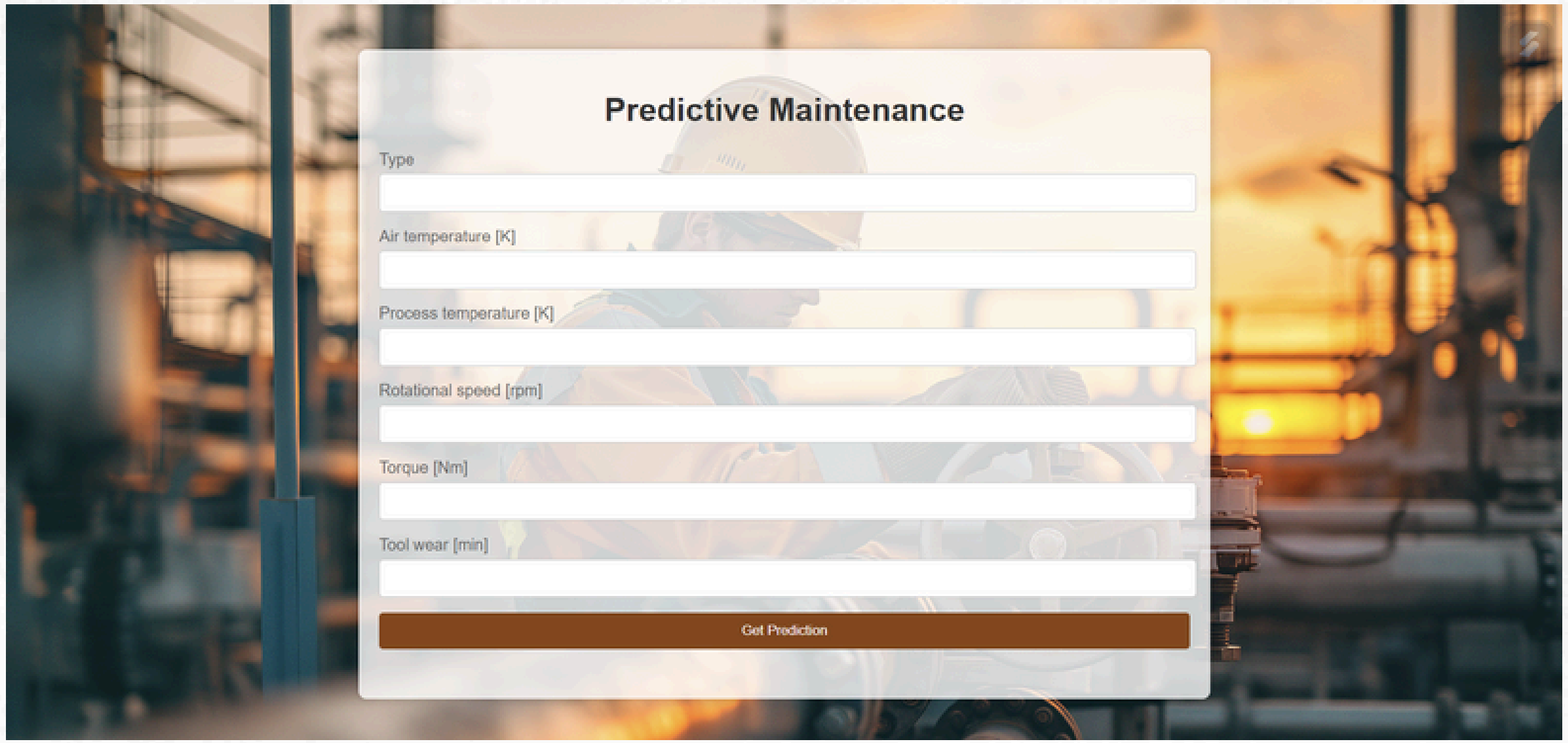
AZURE SIMULATION

DEPLOYMENT.

In this phase, Deploy the fully trained model for live use in real-world applications.

- We use the REST endpoint URL and the Authorization key to set up the API for making predictions.

- The user interface is designed using HTML and CSS, allowing for seamless interaction with the deployed model.

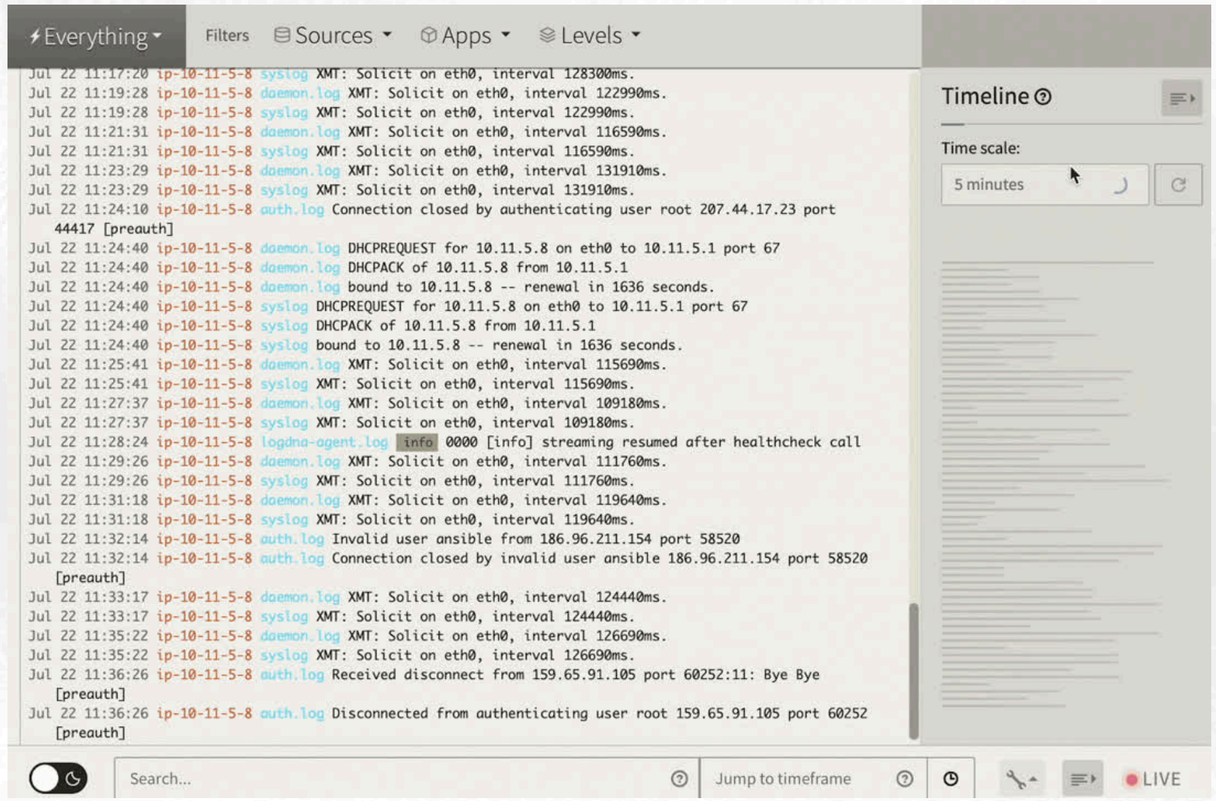


PREDICTIVE LOG ANALYTICS

OVERVIEW.

AS the previous project but using log data.

The Log Analytics Classification project focuses on analyzing log data generated by various systems and applications to classify and categorize events accurately. By leveraging advanced machine learning techniques, this project aims to enhance the understanding of system behaviors, identify anomalies, and improve overall operational efficiency.



PREDICTIVE LOG ANALYTICS

ABOUT THE DATASET.

BGL dataset.

BGL is an open dataset of logs collected from a BlueGene/L supercomputer system at Lawrence Livermore National Labs (LLNL) in Livermore, California, with 131,072 processors and 32,768GB memory. The log contains alert and non-alert messages identified by alert category tags. In the first column of the log, "-" indicates non-alert messages while others are alert messages. The label information is amenable to alert detection and prediction research. It has been used in several studies on log parsing, anomaly detection, and failure prediction.

General Information

- Total Rows: **2000**
- Total Columns: **13**

Column Types

- Integer Columns: **2**
- Object (String) Columns: **11**

Column Name	Description
ID	A unique identifier for each event. Example: `1117838570`.
Date	The date when the event occurred in `YYYY.MM.DD` format. Example: `2005.06.03`.
Node	Indicates the system or unit that logged the event. Example: `R02-M1-N0-C:J12-U11`.
Timestamp	The detailed time when the event occurred in `YYYY-MM-DD-HH.MM.SS.MMMMMM` format. Example: `2005-06-03-15.42.50.675872`.
Node Repeat	Similar to the Node column, it shows a repeat or copy of the node, e.g., `R02-M1-N0-C:J12-U11`.
Type	Specifies the type of event or log, such as `RAS` (Remote Access Service).
Level	Indicates the severity level of the event, such as `INFO` or `FATAL`.
Content	Contains additional information or description of the event, e.g., `instruction cache parity error corrected`.

PREDICTIVE LOG ANALYTICS

EXPLORATORY DATA ANALYSIS.

EDA

Step 1

Data Transformation

Step 2

Data Exploration

Step 3

Data Cleaning

PREDICTIVE LOG ANALYTICS

DATA TRANSFORMATION.

Convert Data Type:

Converting object types to string standardizes text data representation, ensuring consistency for analysis and modeling, while also improving performance by making operations more efficient in terms of memory usage.

Fill Missing Values:

This step is crucial for handling missing values, as it improves data quality, minimizes the negative impact of missing entries on analysis and modeling, and reduces potential bias.

Convert object datatype to string

+ Code + Markdown

[19]:

```
for col in df.columns:
    if df[col].dtype == 'object':
        df[col] = df[col].astype('string')
```

Fill null with mode

[22]:

```
for col in df.columns:
    df[col].fillna(df[col].mode()[0], inplace=True)
```

PREDICTIVE LOG ANALYTICS

DATA EXPLORATION.

Compute Similarity between:

- **node and noderepeat**
- **Content and EventTemplate**

Using:

- **CountVectorizer**
- **SentenceTransformer**

Here, we compute the similarity between text columns to identify any redundancy using two methods: CountVectorizer and SentenceTransformer. The results show that every two columns are highly similar. Therefore, during the cleaning process, we will remove the less important columns to reduce redundancy and improve data quality.

1. using CountVectorizer

[24]:

```
df['Node'] = df['Node'].astype(str)
df['NodeRepeat'] = df['NodeRepeat'].astype(str)

# Text processing
vectorizer = CountVectorizer()
node_vectors = vectorizer.fit_transform(df['Node'])
noderepeat_vectors = vectorizer.transform(df['NodeRepeat'])

# Compute similarity using Cosine
similarity_matrix = cosine_similarity(node_vectors, noderepeat_vectors)

# Create a DataFrame for similarity
similarity_df = pd.DataFrame(similarity_matrix, columns=df['NodeRepeat'], index=df['Node'])

# Display some results
print(similarity_df.head())
```

2. using SentenceTransformer

[27]:

```
# 2. Compute Similarity between Content and EventTemplate using SentenceTransformer

# Assuming df is your DataFrame and 'Content' and 'EventTemplate' are your columns
model = SentenceTransformer('all-MiniLM-L6-v2')

# Encode the texts
content_embeddings = model.encode(df['Content'].tolist())
event_template_embeddings = model.encode(df['EventTemplate'].tolist())

# Compute similarity using Cosine
similarity_1_matrix = cosine_similarity(content_embeddings, event_template_embeddings)

# Create a DataFrame for similarity
similarity_1_df = pd.DataFrame(similarity_1_matrix, columns=df['EventTemplate'], index=df['Content'])

# Display some results
print(similarity_1_df.head())
```

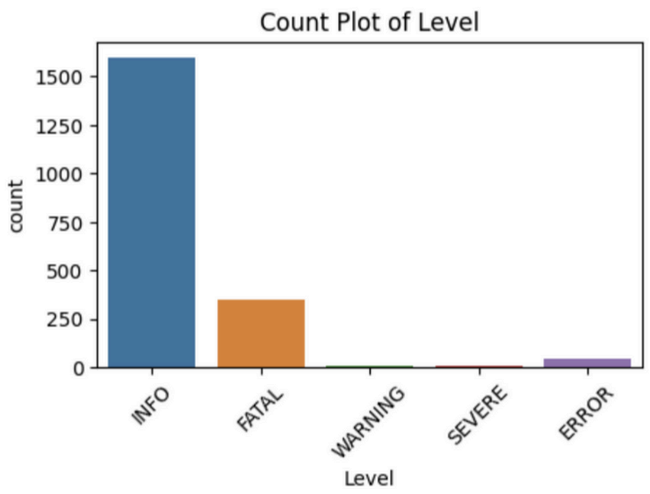
PREDICTIVE LOG ANALYTICS

DATA EXPLORATION.

visualize the frequency distribution of the categories in the 'Component' and 'Level' columns

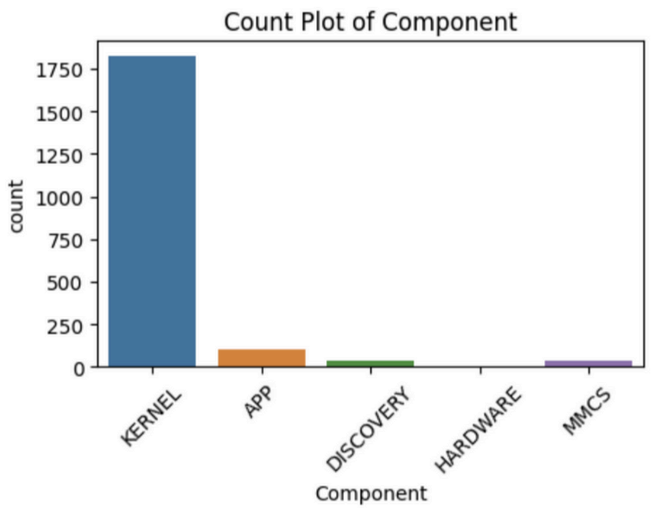
Log Levels Summary

- **INFO:** System activity.
- **FATAL:** Critical failure.
- **ERROR:** Affects functionality.
- **WARNING:** Potential future issue.
- **SEVERE:** Urgent error.



Components Summary

- **KERNEL:** Core system operations and resource management.
- **APP:** Application-related events and errors.
- **DISCOVERY:** Identifies hardware or network resources.
- **MMCS:** Manages and monitors system processes.
- **HARDWARE:** Physical system components and related issue**s.**

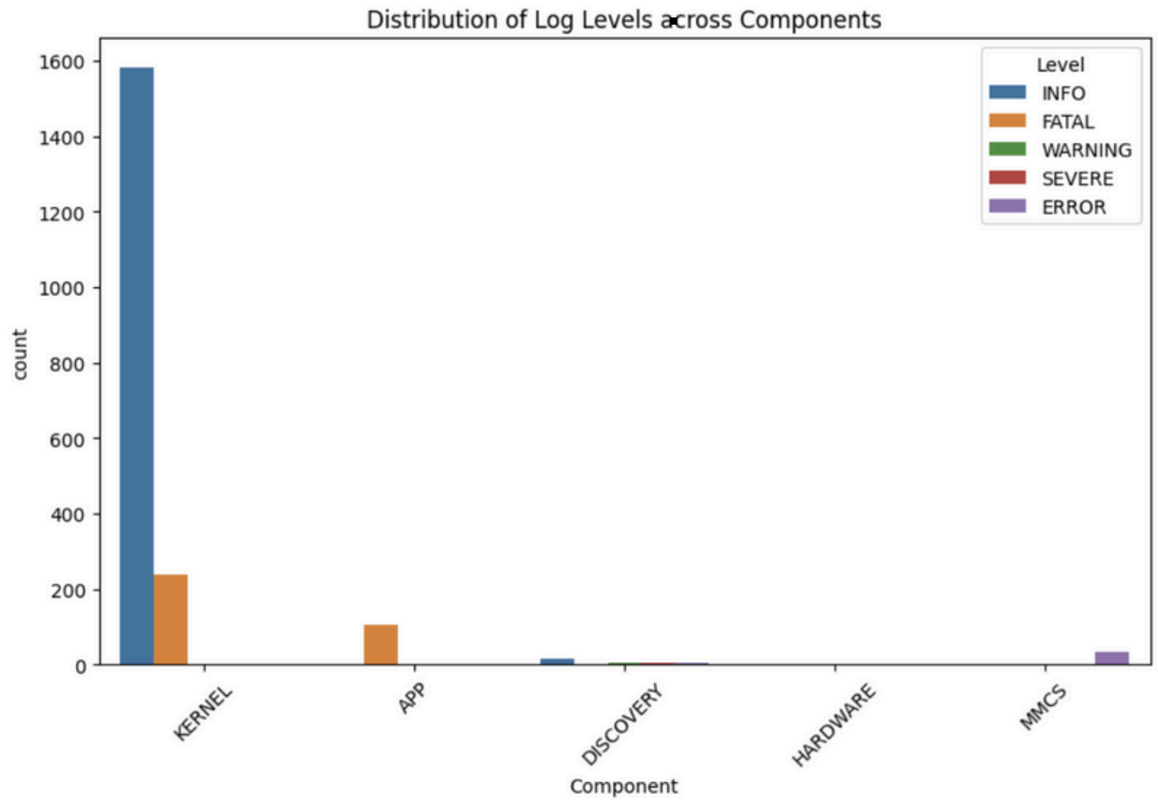


PREDICTIVE LOG ANALYTICS

DATA EXPLORATION.

Distribution of Log Levels across Components

This plot indicates that the **KERNEL** component has the highest ratio of **FATAL** errors, followed by the **APP** component. Additionally, the **MMCS** component exhibits the highest ratio of **ERROR** messages.



PREDICTIVE LOG ANALYTICS

DATA EXPLORATION.

Display the most common words in content column and the result is:

- **gener:** Create or produce.
- **interrupt:** Signal to halt operations.
- **0:** Default or numerical value.
- **iar:** Manages system interrupts.
- **dear** Handles data exceptions.
- **error:** System issue.
- **except:** Error exception.
- **microsecond:** Small time unit.
- **align:** Data formatting.
- **input:** Data entered.

Summary:

The frequent words reflect technical aspects related to system operations, such as:

- **Interrupt** management and controlling the flow of data.
- **Exceptions** and **errors** indicating issues or warnings in the code.
- **Alignment** and **input** referring to data formatting.
- Words like **gener** suggest that processes or data are being generated.

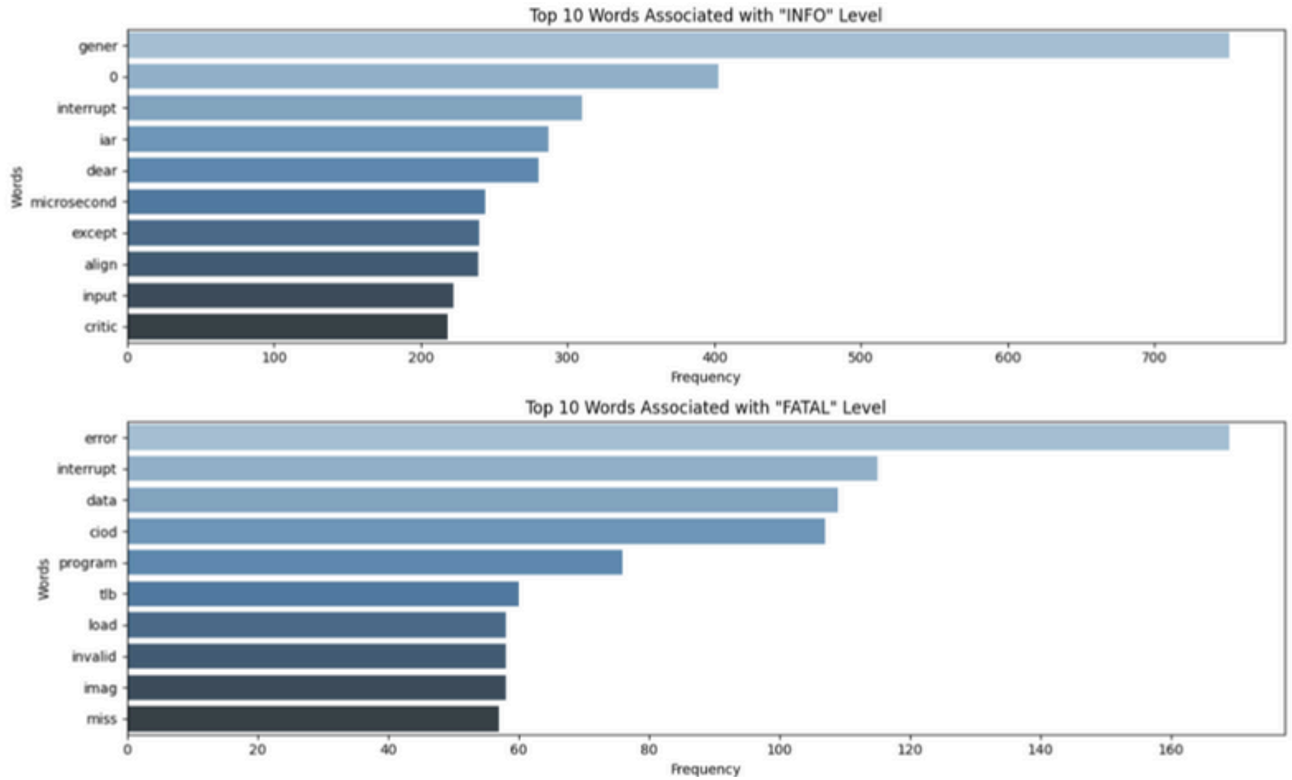
These findings indicate recurring technical processes within the system, focusing on data generation, error handling, and managing operations flow.

PREDICTIVE LOG ANALYTICS

DATA EXPLORATION.

Log Level Word Frequency Distribution

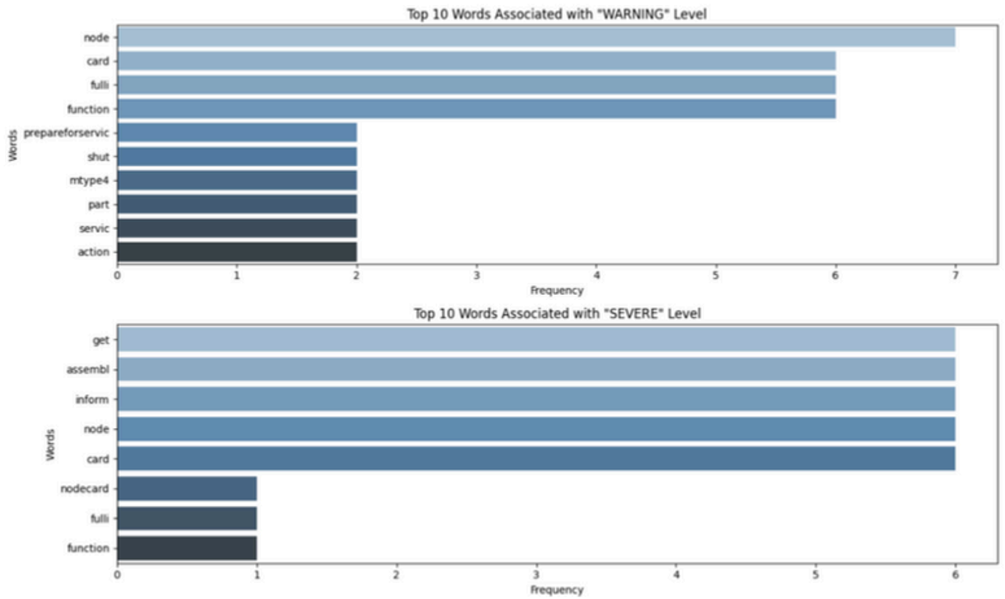
"Log Level Word Frequency Distribution" analyzes how often specific words appear across different log severity levels. This helps identify common issues and enhances our understanding of system performance by focusing on frequently occurring terms in log entries.



PREDICTIVE LOG ANALYTICS

DATA EXPLORATION.

Log Level Word Frequency Distribution



PREDICTIVE LOG ANALYTICS

DATA CLEANING.

After exploring the data, we will proceed with cleaning it accordingly.

Drop unnecessary columns.

- **[LineId ,Label , Date , EventId]:** as they do not contribute to the decision-making process.

```
df.drop(columns=['LineId','Label','Date'], inplace=True)
```

- **Type:** This column contains only one unique value, making it irrelevant to the decision-making process.

- **NodeRepeat , EventTemplate:** These columns are similar to other, more important columns and do not add significant value to the analysis.

```
[29]: df.drop(columns=['Type','NodeRepeat','EventTemplate','EventId'],inplace=True)
```

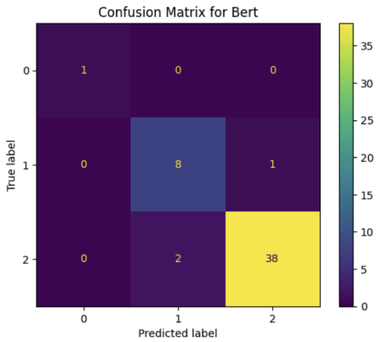
PREDICTIVE LOG ANALYTICS

MODLING.

In this classification task using log data, three transformer models were trained and evaluated: **BERT**, **DistilBERT**, and **RoBERTa**. The following is a performance breakdown for each model over five epochs:

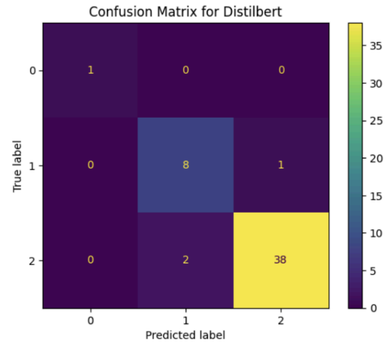
1. BERT:

Precision: 94.35%
Recall: 94.00%
F1 Score: 94.12%
Accuracy: 94.00%



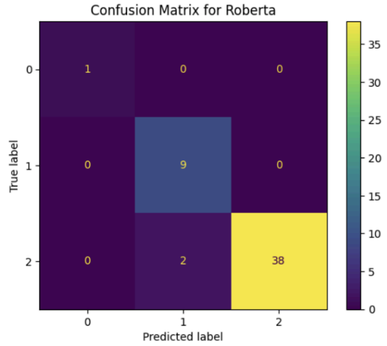
2. DistilBERT:

Precision: 94.35%
Recall: 94.00%
F1 Score: 94.12%
Accuracy: 94.00%



3. RoBERTa:

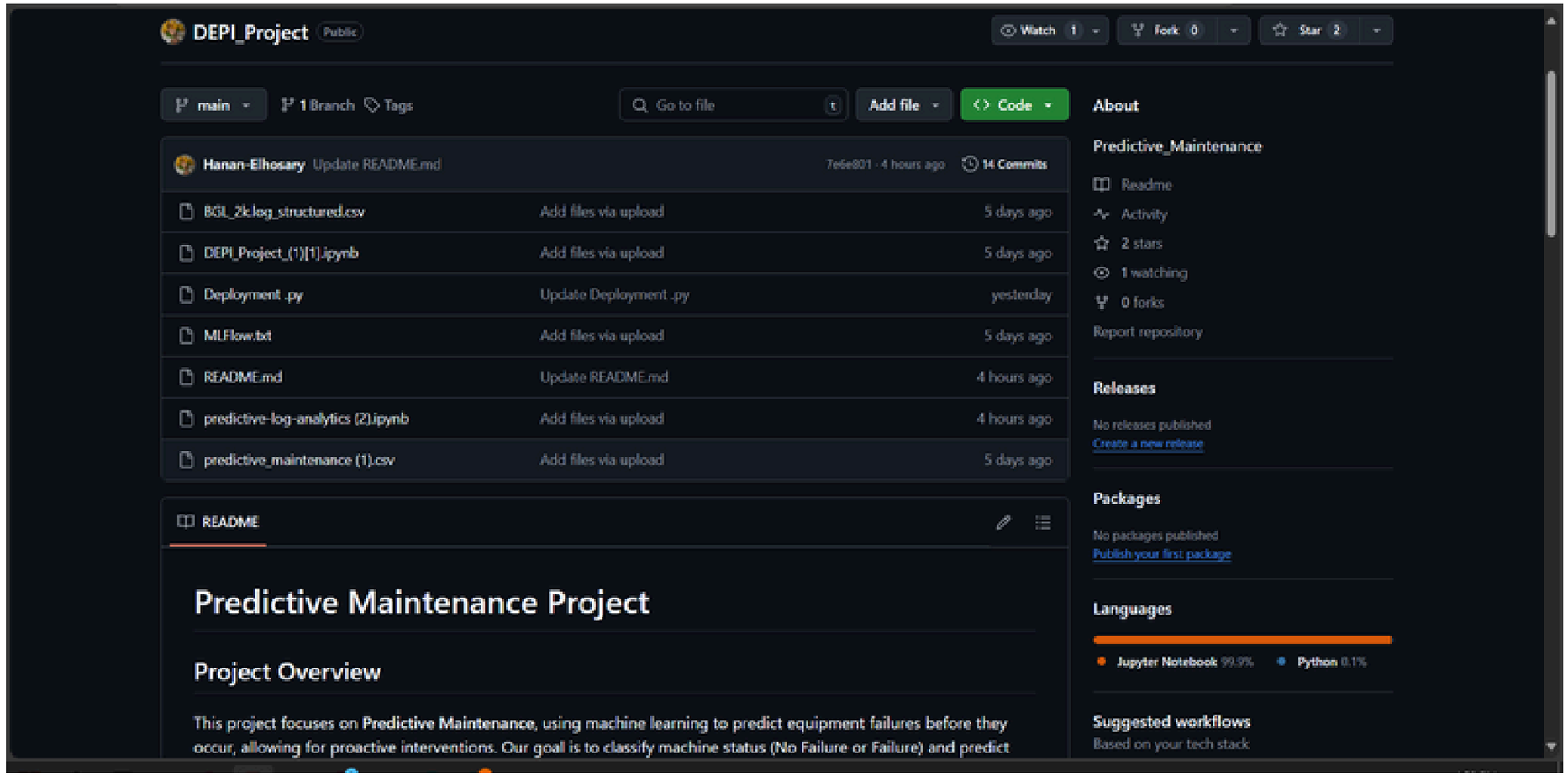
Precision: 96.73%
Recall: 96.00%
F1 Score: 96.15%
Accuracy: 96.00%



Conclusion

Among the three models, **RoBERTa** outperformed both **BERT** and **DistilBERT**, achieving the highest precision, recall, F1 score, and accuracy. This suggests that RoBERTa is particularly effective for the classification task using log data, making it a robust choice for handling complex log data analysis

GITHUB REPO


DEPI_Project
Public
Watch 1
Fork 0
Star 2
main
1 Branch
Tags
Go to file
t
Add file
Code
Hanan-Elhosary Update README.md
7e6e801 · 4 hours ago
14 Commits
BGL_2k.log_structured.csv
Add files via upload
5 days ago
DEPI_Project_(1)[1].ipynb
Add files via upload
5 days ago
Deployment .py
Update Deployment .py
yesterday
MLFlow.txt
Add files via upload
5 days ago
README.md
Update README.md
4 hours ago
predictive-log-analytics (2).ipynb
Add files via upload
4 hours ago
predictive_maintenance (1).csv
Add files via upload
5 days ago
README
Predictive Maintenance Project
Project Overview
This project focuses on Predictive Maintenance, using machine learning to predict equipment failures before they occur, allowing for proactive interventions. Our goal is to classify machine status (No Failure or Failure) and predict
About
Predictive_Maintenance
Readme
Activity
2 stars
1 watching
0 forks
Report repository
Releases
No releases published
Create a new release
Packages
No packages published
Publish your first package
Languages
Jupyter Notebook 99.9%
Python 0.1%
Suggested workflows
Based on your tech stack

THANK YOU