AISE 3010b Final Project: Predictive Maintenance of Machinery

By: Aaron Pedden 251306404, Joseph Toma 251283541

Date: April 4th 2025

Introduction

This project is about using machine learning to help predict when machines might break down in a factory setting. A dataset from Kaggle that was made to look like real machine data was used for this project. It has information like temperatures, torque, rotational speed, and tool wear. The goal was to build two models since the dataset had two prediction targets. One binary classification model that predicts if a machine will fail (yes or no), and another multiclass classification model that predicts the type of failure that occurs (e.g. Overstrain failure). The Data was cleaned and preprocessed using SQL in BigQuery, then both models were trained in Google Colab using PyTorch.

Workflow Overview

Data

For this project, I used a predictive maintenance dataset from Kaggle. This dataset is synthetic but was designed to closely represent real world machine behavior in industrial settings. It includes 10,000 rows of data, each describing the condition of a machine at a certain time using various sensor readings and machine information. The dataset consisted of 8 input features and 2 output targets, one for multiclass classification and one for binary classification.

Key Input Features:

- **UDI**: A unique row ID
- **Product ID**: Includes a serial number with a quality type between low medium and high (L, M, or H)
- **Type**: Product quality type L (low), M (medium), H (high).
- **Air Temperature [K]**: Temperature around the machine.
- **Process Temperature [K]**: Internal machine temperature.
- **Rotational Speed [rpm]**: How fast the machine is spinning.
- **Torque [Nm]**: Force applied by the machine's motor.
- **Tool Wear [min]**: How long the tool has been in use.

Target Variables:

- **Target**: A binary indicator showing whether a failure occurred (1) or not (0).
- **Failure Type**: A multiclass label showing what kind of failure occurred. There are five possible types:
    1. Heat Dissipation Failure
    2. Power Failure
    3. Overstrain Failure
    4. Tool Wear Failure
    5. Random Failures

Since the Dataset had two targets, I thought it would be best to split the data up into two different models shown below:

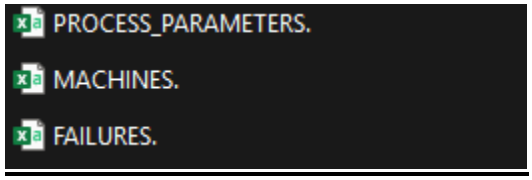
1. A **binary classification** model that predicts whether a machine is likely to fail
2. A **multiclass classification** model that predicts the type of failure present

Screenshot of Dataset opened in excel:

UDI	Target	Failure Type	Air_temperature_K	Process_Temperature_K	Rotational_Speed_rpm	Torque_Nm	Tool _Wear_min	Product ID	Type
10000	0	No Failure	299	308.7	1500	40.2	30	M24859	M
9999	0	No Failure	299	308.7	1408	48.5	25	H39412	H
9998	0	No Failure	299	308.6	1645	33.4	22	M24857	M
9997	0	No Failure	298.9	308.4	1632	31.8	17	H39410	H
9996	0	No Failure	298.8	308.4	1604	29.5	14	M24855	M
9995	0	No Failure	298.8	308.3	1634	27.9	12	L57174	L
9994	0	No Failure	298.8	308.4	1401	47.3	10	L57173	L
9993	0	No Failure	298.8	308.4	1484	39.2	8	L57172	L
9992	0	No Failure	298.9	308.4	1827	26.1	5	M24851	M

Data Preparation:

After downloading the original dataset, it was manually separated into 3 tables based on the type of information each row contained. These were saved as CSV files named PROCESS_PARAMETERS, MACHINES, and FAILURES. Each dataset used the UDI column to keep a consistent link between records.



- The PROCESS_PARAMETERS file included sensor and operating condition data such as air temperature, process temperature, rotational speed, torque, and tool wear.

  PROCESS PARAMETERS Dataset snippet:

UDI	Air_Temperature_K	Process_Temperature_K	Rotational_Speed_rpm	Torque_Nm	Tool _Wear_min
1	298.1	308.6	1551	42.8	0
2	298.2	308.7	1408	46.3	3
3	298.1	308.5	1498	49.4	5
4	298.2	308.6	1433	39.5	7
5	298.2	308.7	1408	40	9
6	298.1	308.6	1425	41.9	11
7	298.1	308.6	1558	42.4	14

- The MACHINES file contained the Product ID and product quality type (L, M, H)

  MACHINES Dataset snippet:

UDI	Product ID	Type
1	M14860	M
2	L47181	L
3	L47182	L
4	L47183	L
5	L47184	L
6	M14865	M
7	L47186	L

- The FAILURES included the target labels: whether a failure occurred and the type of failure that was present.

  FAILURES Dataset snippet:

UDI	Target	Failure Type
1	0	No Failure
2	0	No Failure
3	0	No Failure
4	0	No Failure
5	0	No Failure
6	0	No Failure
7	0	No Failure

Joining the tables:

Once the files were prepared, they were uploaded into a Google Cloud Storage (GCS) bucket named *machine-predict-maintenance-2025*. This setup made it easy to import the files directly into BigQuery as individual tables and maintain an organized workflow. To merge the datasets into one view for later analysis and modeling, the following SQL query was used to join the three tables on the UDI key:

```sql
SELECT f.UDI,
       f.Target,
       f.`Failure Type`,
       pp.Air_temperature_K,
       pp.`Process_Temperature_K `,
       pp.Rotational_Speed_rpm,
       pp.Torque_Nm,
       pp.`Tool _Wear_min`,
       m.`Product ID`,
       m.Type
FROM `final-project-3010b.machine_predictive_maintenance.Process Parameters` AS pp
JOIN `final-project-3010b.machine_predictive_maintenance.Machine` AS m ON pp.UDI = m.UDI
JOIN `final-project-3010b.machine_predictive_maintenance.Failures` AS f ON pp.UDI = f.UDI
ORDER BY UDI DESC
```

Once the joined table was created it was saved to the bucket *machine-predict-maintenance-2025* as *JoinedModelDataset* as a csv file. At this point part 1) Data selection was finished and 2) GCP Data Processing was ready to undergo.

SQL Processing

Once the full dataset was joined and saved as *JoinedModelDataset* in BigQuery, SQL was used to prepare the data for machine learning.

1. Previewing the Data

Before starting any processing, that dataset was previewed using a SELECT * …LIMIT 10 query to make sure all columns and values were present and properly imported.

Query:

```sql
SELECT * FROM `final-project-3010b.machine_predictive_maintenance.JoinedModelDataset` LIMIT 10
```

Result:

Query results

Save results | Open in

Job information | Results | Chart | JSON | Execution details | Execution graph

Row	UDI	Target	Failure Type	Air_temperature_K	Process_Temperatur	Rotational_Speed_rp	Torque_Nm	Tool _Wear_min	Product ID
1	922	0	No Failure	295.5	305.9	1593	37.2	197	H30335
2	928	0	No Failure	295.6	306.0	1396	52.4	0	H30341
3	926	0	No Failure	295.6	306.1	1623	30.9	210	H30339
4	902	0	No Failure	295.6	306.1	1256	62.3	142	H30315
5	957	0	No Failure	295.6	306.2	1632	32.2	76	H30370
6	955	0	No Failure	295.6	306.2	1414	42.5	69	H30368
7	905	0	No Failure	295.7	306.2	1458	49.8	151	H30318
8	900	0	No Failure	295.6	306.2	1329	55.3	135	H30313
9	888	0	No Failure	295.7	306.2	2161	16.5	105	H30301
10	952	0	No Failure	295.6	306.3	1509	35.8	60	H30365

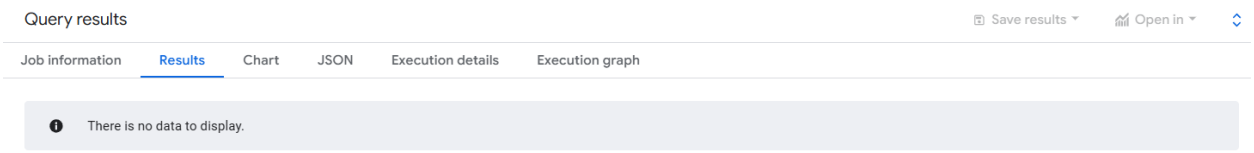
2. Checking for Duplicates

To ensure data integrity, A duplicate check was run on the UDI column, using a GROUP BY and HAVING COUNT(*) > 1 query. No duplicates were found, confirming that every row was unique.

Query:

```
SELECT UDI, COUNT(*) AS count
FROM `final-project-3010b.machine_predictive_maintenance.JoinedModelDataset`
GROUP BY UDI
HAVING COUNT(*) > 1;
```

Result:



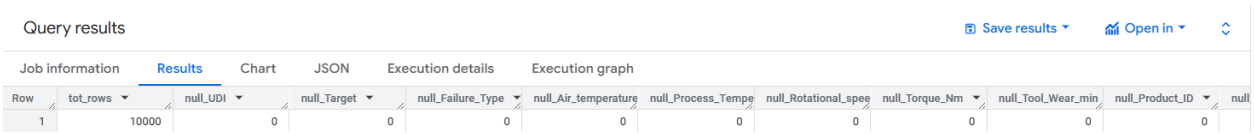
3. Checking for Missing Values

A query was used to count how many NULL values appeared in each column. This confirmed there were no missing values, and that the dataset was clean enough to proceed to handling further issues with the dataset.

Query:

```
SELECT
  COUNT(*) AS tot_rows,
  COUNTIF(UDI IS NULL) AS null_UDI,
  COUNTIF(Target IS NULL) AS null_Target,
  COUNTIF(`Failure Type` IS NULL) AS null_Failure_Type,
  COUNTIF(Air_temperature_K IS NULL) AS null_Air_temperature_K,
  COUNTIF(`Process_Temperature_K ` IS NULL) AS null_Process_Temperature_K,
  COUNTIF(Rotational_speed_rpm IS NULL) AS null_Rotational_speed_rpm,
  COUNTIF(Torque_Nm IS NULL) AS null_Torque_Nm,
  COUNTIF(`Tool _Wear_min` IS NULL) AS null_Tool_Wear_min,
  COUNTIF(`Product ID` IS NULL) AS null_Product_ID,
  COUNTIF(Type IS NULL) AS null_Type
FROM `final-project-3010b.machine_predictive_maintenance.JoinedModelDataset`
```

Result:



4. Handling Labeling Issues

column names, such as “Tool _Wear_min”, “Process_Temperature_K ”, and “ Failure Type” had hidden spaces within the name, which caused problems when querying. These were cleaned up using AS in my SQL queries to avoid errors.

Queries:

```
`Process_Temperature_K ` AS Process_Temperature

`Tool _Wear_min`AS Tool_Wear_min

`Failure Type` AS Failure_Type
```

5. Creating Separate Tables for Binary and Multiclass Classification and Dropping Unimportant Columns

Since the dataset has two target variables, I created two new tables, one for each model:

- *BinaryDataset*: This included all records, with features and the binary Target column (0 = No failure, 1 = Failure). Column “Failure_Type” was dropped due to it only being useful for multiclass classification. “Product_ID” was dropped due to it holding no value, as we already have a UDI that uniquely identifies the row and “Type” that displays the quality type (L, M, H).

  Query:

```
CREATE TABLE `final-project-3010b.machine_predictive_maintenance.BinaryDataset` AS
SELECT
  UDI,
  Type,
  Air_temperature_K,
  `Process_Temperature_K ` AS Process_Temperature,
  Rotational_Speed_rpm,
  Torque_Nm,
  `Tool _Wear_min`AS Tool_Wear_min,
  Target
FROM `final-project-3010b.machine_predictive_maintenance.JoinedModelDataset`
WHERE Target IS NOT NULL;
```

Result:

BinaryDataset Query Open in Share Copy Snapshot Delete Export

Schema Details Preview Table explorer Preview Insights Lineage Data profile Data Quality

Row	UDI	Type	Air_temperature	Process_Tempe	Rotational_Spee	Torque_Nm	Tool_Wear_min	Target
1	952	H	295.6	306.3	1509	35.8	60	0
2	1856	H	297.7	307.3	1249	62.8	47	0
3	329	H	297.6	308.4	1703	31.4	205	0
4	3200	H	300.1	309.1	1560	35.4	14	0
5	9849	H	298.5	309.2	1568	40.6	39	0
6	9317	H	298.5	309.4	1417	41.2	155	0
7	588	H	297.6	309.5	1623	33.1	224	0
8	7144	H	300.3	309.7	1582	30.8	144	0
9	9498	H	299.1	309.9	1443	40.3	0	0
10	1438	H	298.8	309.9	1439	45.2	40	1

- *MultiClassDataset*: This table filtered out rows with “No Failure” and kept only records with actual failure types. It also dropped columns “Target” (Only useful for multiclass classification) and “Product_ID” due to it holding no value.

Query:

```
CREATE OR REPLACE TABLE `final-project-3010b.machine_predictive_maintenance.MultiClassDataset` AS
SELECT
  UDI,
  Type,
  Air_temperature_K,
  `Process_Temperature_K ` AS Process_Temperature_K,
  Rotational_Speed_rpm,
  Torque_Nm,
  `Tool _Wear_min` AS Tool_Wear_min,
  `Failure Type` AS Failure_Type
FROM `final-project-3010b.machine_predictive_maintenance.JoinedModelDataset`
WHERE LOWER(`Failure Type`) != 'no failure' AND `Failure Type` IS NOT NULL;
```

Result:

MultiClassDataset Query Open in Share Copy Snapshot Delete Export

Schema Details Preview Table explorer Preview Insights Lineage Data profile Data Quality

Row	UDI	Type	Air_temperature	Process_Tempe	Rotational_Spee	Torque_Nm	Tool_Wear_min	Failure_Type
1	3237	M	300.8	309.4	1342	62.4	113	Heat Dissipation Failure
2	4308	L	301.4	309.9	1259	63.9	20	Heat Dissipation Failure
3	4327	L	301.6	310.1	1362	55.8	70	Heat Dissipation Failure
4	4329	L	301.6	310.1	1326	48.2	77	Heat Dissipation Failure
5	4141	L	301.7	310.2	1331	61.2	47	Heat Dissipation Failure
6	4286	L	301.7	309.9	1317	49.0	187	Heat Dissipation Failure
7	4139	L	301.7	310.2	1311	57.4	40	Heat Dissipation Failure
8	4384	L	301.7	309.5	1298	65.5	229	Heat Dissipation Failure
9	4324	L	301.8	310.3	1341	50.5	64	Heat Dissipation Failure
10	4284	M	301.8	309.9	1334	61.0	182	Heat Dissipation Failure

6. Splitting into Training and Testing Sets

For each model type (binary and multiclass), the datasets were split into 80% for training and 20% for testing.

- **Binary Dataset**: Used FARM_FINGERPRINT() and MOD() functions on the UDI to randomly split the data. The training data was stored in a table named BinaryTrain, and the testing data was stored in a table named BinaryTest.

  Query:

```
--80%
CREATE OR REPLACE TABLE `final-project-3010b.machine_predictive_maintenance.BinaryTrain` AS
SELECT *
FROM `final-project-3010b.machine_predictive_maintenance.BinaryDataset`
WHERE MOD(ABS(FARM_FINGERPRINT(CAST(UDI AS STRING))), 10) < 8;

--20%
CREATE OR REPLACE TABLE `final-project-3010b.machine_predictive_maintenance.BinaryTest` AS
SELECT *
FROM `final-project-3010b.machine_predictive_maintenance.BinaryDataset`
WHERE MOD(ABS(FARM_FINGERPRINT(CAST(UDI AS STRING))), 10) >= 8;
```

  Result:

- **Multiclass Dataset**: Used NTILE(10) and partitioned by Failure_Type so that each class was evenly distributed between training and testing sets. The training data was stored in a table named MultiClassTrain, and the testing data was stored in a table named MultiClassTest.

Query:

```
--80%
CREATE OR REPLACE TABLE `final-project-3010b.machine_predictive_maintenance.MultiClassTrain`
AS
SELECT *
FROM (
  SELECT *,
         NTILE(10) OVER (PARTITION BY Failure_Type ORDER BY FARM_FINGERPRINT(CAST(UDI AS
STRING))) AS bucket
  FROM `final-project-3010b.machine_predictive_maintenance.MultiClassDataset`
)
WHERE bucket <= 8;

--20%
CREATE OR REPLACE TABLE `final-project-3010b.machine_predictive_maintenance.MultiClassTest`
AS
SELECT *
FROM (
  SELECT *,
         NTILE(10) OVER (PARTITION BY Failure_Type ORDER BY FARM_FINGERPRINT(CAST(UDI AS
STRING))) AS bucket
  FROM `final-project-3010b.machine_predictive_maintenance.MultiClassDataset`
)
WHERE bucket > 8;
```

Result:

Once the multiclass and binary classification datasets were transferred to there individual training and testing sets, it was ready to move on to the next step. All 4 of the datasets were exported to the bucket *machine-predict-maintenance-2025* and downloaded to my computer for the next step in Google Colab.

Model

1. Custom Model

Due to technical issues with Vertex AI, both the binary and multiclass classification models were developed and trained in Google Colab using PyTorch.

Binary Classification Model:

The binary model predicts whether a machine is likely to fail (1) or not (0) based on certain specifications of the machine and its environment. The datasets BinaryTrain and BinaryTest from GCP went through additional preprocessing, using libraries such as pandas and scikit-learn. All categorical values were label encoded, and numeric values were standardized with a StandardScaler().

Model Architecture:

Input layer: 7 neurons (one for each feature)
Hidden layer 1: 16 neurons and ReLU activation
Hidden layer 2: 8 neurons and ReLU activation
Output layer: 1 neuron (binary classification)

The model used the BCEWithLogitsLoss as the loss function and the Adam optimizer with a learning rate of 0.001. The model was trained over 100 epochs and a batch size of 32.

Multiclass Classification Model:

The multiclass model predicts which specific type of failure a machine is going to have based on operation specifications. The datasets MultiClassTrain and MultiClassTest from the GCP also went through additional preprocessing using Python libraries like pandas and scikit-learn. Categorical columns such as “Type” and “Failure_Type” were label encoded, and all numerical values were normalized using StandardScaler().

Model Architecture:

- **Input layer:** 7 neurons (one for each feature)
- **Hidden layer 1:** 32 neurons and ReLU activation
- **Hidden layer 2:** 16 neurons and ReLU activation
- **Output layer:** 5 neurons (one for each failure type class)

The model used CrossEntropyLoss for multiclass classification and the Adam optimizer with a learning rate of 0.001. It was trained over 100 epochs using a batch size of 32 in Google Colab.

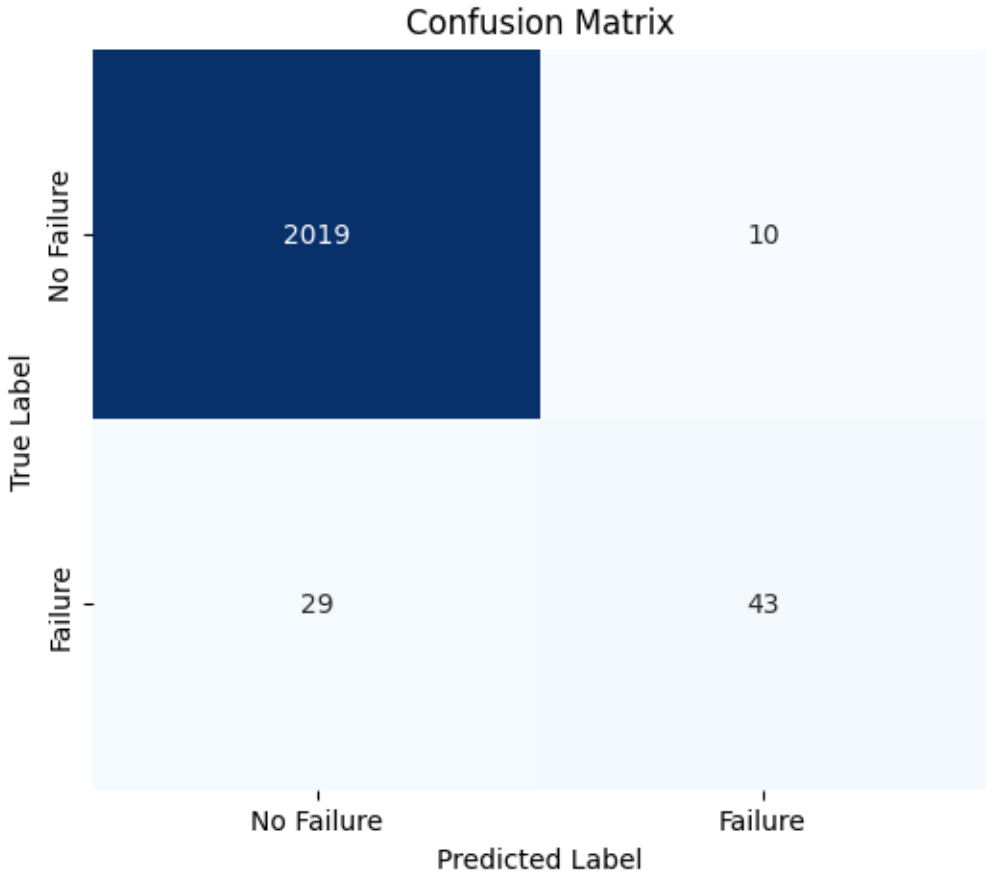
Deployment

In a real-world scenario, the model would be deployed so it can be used to make predictions on new data. While I did not deploy the models using Vertex AI or create a live endpoint, both my binary and multiclass classification models are saved and organized in a way that makes them ready for deployment. If more time were available, the next step would be to upload the models to Vertex AI and serve predictions from an API.

Model Evaluation Results

Binary Classification Model:

Confusion Matrix:



The confusion matrix above displays how well the binary model predicted failures on the test dataset. Out of 2101 samples, the model correctly predicted 2019 machines that did not fail and 43 machines that did fail. It only made 10 false predictions, where it predicted a failure that didn't happen, and 29 misses, where it failed to detect a machine that broke down. This result shows that the model is very good at spotting machines that are working fine. However, it sometimes struggles to catch actual failures due to the dataset being slightly imbalanced. In the real world, missing a failure can be a serious issue, so future improvements should focus on reducing those missed cases.

Loss Over Epochs:

```
Epoch 1, Loss: 73.8208  Epoch 96, Loss: 11.7259
Epoch 2, Loss: 35.5958  Epoch 97, Loss: 11.6945
Epoch 3, Loss: 31.1487  Epoch 98, Loss: 11.7074
Epoch 4, Loss: 27.9282  Epoch 99, Loss: 11.6007
Epoch 5, Loss: 25.6414  Epoch 100, Loss: 11.4829
```

The table shows how the loss value changed during training over 100 epochs. The loss function that was used is called BCEWithLogitsLoss and is very common for binary classification problems. Loss is a measure of how far off the model's predictions are from the actual values, so lower is better. In the beginning, the loss was high at 73.8208 during the first epoch, but this quickly changed as it dropped as training continued. At epoch 5, it was already down to 25.6414. In the last 5 epochs, the loss steadily decreased and ended at 11.4829 by epoch 100. This shows that the model improved over time and learned to make better predictions as it trained.

Classification Report:

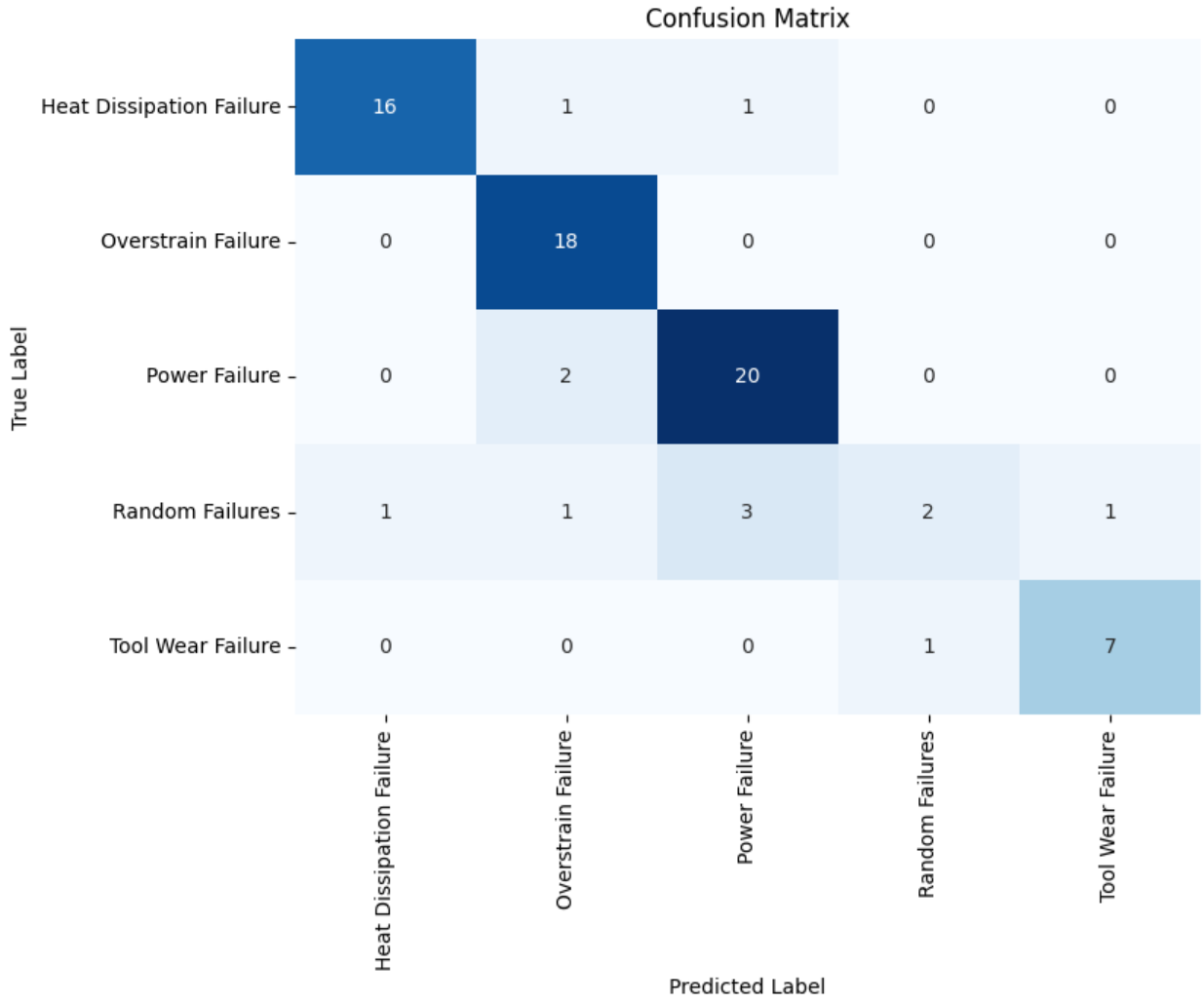
Classification Report:

	precision	recall	f1-score	support
No Failure	0.99	1.00	0.99	2029
Failure	0.81	0.60	0.69	72
accuracy			0.98	2101
macro avg	0.90	0.80	0.84	2101
weighted avg	0.98	0.98	0.98	2101

The classification report shows how well the binary model performed when predicting machine failures. The model had an overall accuracy of 98%, meaning it correctly predicted most of the test samples. For the "No Failure" class, the model was very strong as it had 99% precision and 100% recall, meaning it was almost perfect at identifying machines that did not fail. For the "Failure" class, performance was lower, with 81% precision and 60% recall. This means the model was good at correctly labeling failures when it predicted them, but it still missed several actual failures. The f1-score for failures was 69%, displaying the balance between precision and recall. Despite some room for improvement in detecting failures, the model did very well overall, especially considering the dataset was imbalanced, with many more samples of "No Failure" than "Failure."

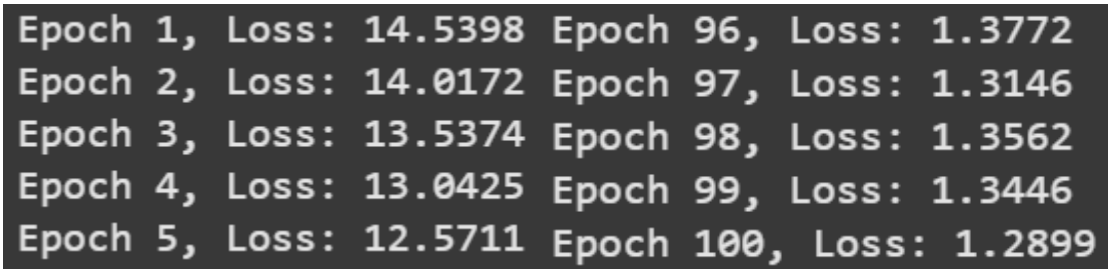
Multi Class Classification Model:

Confusion Matrix:



The confusion matrix for multiclass classification shows how well the model predicted each type of machine failure. Overstrain Failures performed the best, where the model correctly predicted all 18 cases, displaying a perfect result. It also did very well with Power Failures, correctly predicting 20 out of 22, and Heat Dissipation Failures, getting 16 out of 18 right. Some other failure types were a bit more difficult for the model. For example, Random Failures were often confused with Power Failures, and a couple were also mistaken for other types. Tool Wear Failures were mostly predicted correctly, with just one being misclassified. Overall, the model performed strongly, especially on the more common failure types. Only a few mistakes were made, mostly when failure types had similar patterns. Maybe if the dataset were bigger, the low sample size failures like Random Failures would improve.

Loss Over Epochs:



This image shows how the model's training loss changed across 100 epochs. The loss function used for this multiclass classification task was CrossEntropyLoss, which is commonly used when predicting one class out of multiple options. In the beginning, the model had a high loss of about 14.5398, meaning its predictions were not that good. As the training went on, the loss value slowly decreased with each epoch. By the end of the training, the loss had dropped to around 1.2899, showing that the model was learning and getting better at making predictions. The steady drop in loss shows that the model was able to understand the patterns in the data over time.

Classification Report:

Classification Report:

	precision	recall	f1-score	support
Heat Dissipation Failure	0.94	0.89	0.91	18
Overstrain Failure	0.82	1.00	0.90	18
Power Failure	0.83	0.91	0.87	22
Random Failures	0.67	0.25	0.36	8
Tool Wear Failure	0.88	0.88	0.88	8
accuracy			0.85	74
macro avg	0.83	0.78	0.78	74
weighted avg	0.84	0.85	0.83	74

The classification report shows how well the model performed in predicting each type of machine failure based on the 5 types of failure. It did very well on Overstrain Failure, correctly identifying all 18 cases, giving it a perfect recall of 1.00. Power Failure and Tool Wear Failure were also predicted very well, with high precision and recall values between 0.83 and 0.91. The model also performed accurately on Heat Dissipation Failures. However, it struggled the most with Random Failures, which had a low recall of 0.25 and a precision of 0.67, meaning it only correctly predicted 2 out of 8 cases. Overall, the model had an accuracy of 0.85 which is very good. The model is reliable in many cases but has room to improve, especially for failures that are more unpredictable like random failures.

Conclusion

This project successfully demonstrated the application of Google Cloud Platform tools and machine learning techniques to solve a predictive maintenance problem using a relational dataset, beginning with thorough SQL based preprocessing in BigQuery that ensured clean, well-structured input for our machine learning models. The binary classification model achieved 98% accuracy with excellent precision for non failure cases, while the multiclass model performed particularly well in classifying overstrain failures with perfect accuracy and showed robust results for power and heat dissipation failures, though rare failure types like random failures present an opportunity for future improvement with additional training data. Our custom PyTorch implementations proved effective with clear training convergence, validating the practical value of predictive maintenance systems in transforming raw sensor data into actionable insights while maintaining cost efficiency throughout the process. The models developed are preserved and ready for potential deployment, offering a foundation for future enhancements like real-time monitoring integration or expanded failure mode coverage, and the techniques demonstrated from SQL based feature engineering to custom neural network implementation provide a replicable framework for similar classification problems across various domains while successfully meeting all outlined objectives from relational data handling to effective model development.

References

Dataset: https://www.kaggle.com/datasets/shivamb/machine-predictive-maintenance-classification