This is an MSc Data science project.

Datasetlink is :   
[https://archive.ics.uci.edu/dataset/601/ai4i+2020+predictive+maintenance+dataset](https://archive.ics.uci.edu/dataset/601/ai4i+2020+predictive+maintenance+dataset" \t "_blank)

Research Topic :Does oversampling imbalanced data improve the performance of Random Forest, MLP Classifier, and KNN in predicting machine failure from sensor data?

Research Question:To evaluate the impact of oversampling techniques(SMOTE, ADASYN, RandomOverSampler, SMOTETomek) on the performance of Random Forest, MLP Classifier, and KNN in predicting machine failures from imbalanced sensor data, measured by F1-score, recall, and ROC AUC.

I need you to write a 4000(excluding appedix, references, first page.)words Final Project report for the MSC.Data Science course.

I need all the standard format needed for an academic report. The language should not have AI cliches.

These are the references for literature review:  
"Literature Review" section only for these 4 papers

a). Wah, Y.B., Ismail, A., Azid, N.N.N., Jaafar, J., Aziz, I.A., Hasan, M.H. and Zain, J.M. (2023), 'Machine Learning and Synthetic Minority Oversampling Techniques for Imbalanced Data: Improving Machine Failure Prediction', Computers, Materials & Continua, 75(3), pp. 4821-4841. doi: 10.32604/cmc.2023.034470. Available at: <https://www.researchgate.net/publication/370529878_Machine_Learning_and_Synthetic_Minority_Oversampling_Techniques_for_Imbalanced_Data_Improving_Machine_Failure_Prediction> (Accessed: 11 July 2025).

b). Hakami, A. (2024) 'Strategies for overcoming data scarcity, imbalance, and feature selection challenges in machine learning models for predictive maintenance', Scientific Reports, 14, Article 9645. doi: 10.1038/s41598-024-59958-9. Available at: <https://www.nature.com/articles/s41598-024-59958-9> (Accessed: 11 July 2025).

c). Sridhar, S. and Sanagavarapu, S. (2021) 'Handling Data Imbalance in Predictive Maintenance for Machines using SMOTE-based Oversampling', in 2021 13th International Conference on Computational Intelligence and Communication Networks (CICN). Lima, Peru, 22-23 September 2021. IEEE, pp. 44-49. doi: 10.1109/CICN51697.2021.9574668. Available at: <https://ieeexplore.ieee.org/abstract/document/9574668> (Accessed: 11 July 2025).

d). Mahale, Y., Kolhar, S. and More, A.S. (2025) ‘Enhancing predictive maintenance in automotive industry: addressing class imbalance using advanced machine learning techniques’, Discover Applied Sciences, 7, 340. <https://doi.org/10.1007/s42452-025-06827-3>. Available at: <https://link.springer.com/article/10.1007/s42452-025-06827-3(Accessed> 5 August 2025).

project report guidelines:  
 1.⁠ ⁠Managing your project and practical work

This is your research project and you set the agenda for it. You are responsible for the management of your work and you need to manage the time you spend on your project including the time you spend with your supervisor. To produce a good project report you should be working on your project daily and average 40 hours of work a week (600 hours for the whole module).

You will produce a project plan as part of your Project and Data Management (PDM) Plan and you should use it to plan what to do each week. Update it throughout the project and be clear what you are aiming to do each week to progress your project. This will help to keep you focused on your work. You will have to work on tasks in parallel to get all the work done in time, for example, you will need to work on your literature review and do your practical work at the same time. You should expect to learn new things while working on your MSc Project so allow enough time to learn what you need to. Make sure you allocate plenty of time to work on each of the assessments before the due dates.

To pass your project you must produce some practical work which means coding. You need to show some results and demonstrate your code in your viva to pass your project. So, start doing your practical work early and finish it early so that you have time towards the end of the project to spend on writing your report. Not completing your practical work in time could result in you failing the module. When you write your results, analysis and conclusions sections you may realise you need some extra results so leave yourself time to do extra coding to add more results.

You may reach a point when you are not sure what to do next. If this is the case talk to your supervisor and agree a plan to go forward. You may also find there are times when it feels like there is too much to do. Again, you can talk to your supervisor and agree what are the most important things to work on and what can wait.

All your work and results must be included in your final submission of the Final Project Report. Make sure that anything in your code that you want to be marked is included in the report.

Your code is reviewed during the viva.

Make sure any work you have done that explores different parameters, hyperparameters, methods or optimisation of your results, are included in the report. If you have done the work but it is not in the report, you will not get any marks for it.

[25/08/2025, 01:58:15] Sherin UH UK: 2.⁠ ⁠Literature Comparison and Referencing

Literature review and comparison

The literature review is an important part of your project and requires time and effort to find good papers and to summarise the contents of the paper. Looking at the literature gives you background information about the topic you have chosen and the methods you could use. It also informs you about the research that has already been done that is relevant to your project and you should use this information to direct your work. For example, if a model has been shown in the literature to work best for your type of application, then that is a justified reason for you to choose that model as your starting point in the project.

You should provide a comparison between the methods used in the literature with the methods you implement in your project. Provide a justification for your choice of models using this comparison. Furthermore, compare the results in the literature with the results you get from implementing your models.

It is essential that you use peer reviewed published papers in your literature review. These are papers that are in published journals or in published conference proceedings. If your reference list is mostly websites and student theses then you will lose marks. It is common in the computing industry for papers to be published within peer reviewed conference proceedings to give a fast turn-around time for the publication. Computing is a fast-moving industry and research changes quickly.

Searching for relevant papers

To find out more about how to search online and about the format for referencing and citation read the library SkillUp modules on ‘Searching’ and ‘Referencing’.

Library SkillUp and Referencing

Use the UH online library. You can search for published papers using any search engine, but you may not be able to read some of the publications without paying. If you search for the publications using the UH online library, then you will be able to read them since many of the publications that you will be interested in have been subscribed to by the library so that you have free access to the whole paper.

Referencing

Referencing and citations must be in Harvard format alphabetically. Read the Guidance on referencing .

•⁠ ⁠The reference list at the end of your report must be in alphabetical order based on the first author surname.

•⁠ ⁠At the end of each reference, you must include a linked web-address (see format below).

•⁠ ⁠All citations in the report must be in the reference list.

•⁠ ⁠The reference list must only contain the publications that are cited in the report.

•⁠ ⁠All papers referenced should include the author’s list, year of publication, the journal name, the journal volume and the page numbers.

•⁠ ⁠If the paper is from arXiv include the authors, the year, the publication number and the web address.

•⁠ ⁠Any websites quoted should have an author name (or company name) of the publisher and the year it was published (both are often at the bottom of the page in very small print, and you use both for your citations) and then the web address. You must also state when you accessed the website.

•⁠ ⁠If the publication is a book, then you include the author name(s), the year, the book title and the publisher.

•⁠ ⁠If you want to include books, websites or other information that you read as background that you think is relevant but is not a specific reference then you must include these in a ‘Bibliography’.

All references must be real references that can be accessed by the marker. A fake or incorrect reference is an academic offence.

In-text Citations

A citation are the words put in the text of the report that show where the reference relates to. Citations should be put at the first place the author or publication is mentioned. Citations should not be put at the end of the sentence of paragraph if you first mention it at the beginning or middle. Citations come in two forms;

 1.⁠ ⁠‘First author surname (date)’ - for those where the author’s name is part of the sentence

Example: “Smith et al, (2020) said that …”.

 2.⁠ ⁠‘(first author surname, date)’ - for those where the author’s name is not mentioned in the text

Example: “It was shown (Smith et al, 2020) that…”.

Examples of referencing

Book

Author, initials (year) ‘Title of book’, Publisher. (Available at URL)

Example

•⁠ ⁠Frank, R.H., (1997), Microeconomics and behaviour, London: McGraw-Hill (Available at: https://www.mheducation.co.uk/microeconomics-and-behaviour-3e-9781526847843-emea-group )

Journal publications

Author(s), Initials, (Year), ‘Title of the article’, Name of the journal, Volume number, (Part number), First and last pages. (Available at URL)

Example

•⁠ ⁠Watson, M., (2006), ‘Management accounting and budgetary control’, Public Finance Quarterly, 3 2(2), pp. 234-7. (Available at: http://search.global.epnet.com )

[25/08/2025, 01:59:07] Sherin UH UK: The Final Project Report

Word count and format

The Final Project Report should be a maximum of 5,000 words, this means that the report can be less words but cannot be more. The word count includes the abstract and contents page and the subsequent sections up to and including the conclusions. The word count does not include the reference list, the appendices, the front page, the declaration page, and the acknowledgements.

The report must be written in either Arial, Times or Calibri font with a font size of 12 and single line spacing. The Final Project Report should include sections that are relevant to your project, if you are unsure about the sections to include then talk to your supervisor. The following sections give an outline of what should be included.

Front Page and Declaration Page: You must use the template provided in the Assignments section on canvas for the front page and declaration page. You must sign the declaration page. You must add the word count on the front page. The blue writing on the front page template should be replaced with the information on your project. Make sure you include your linked GitHub address on the front page and that it is shared so that the markers can access it.

Acknowledgements: Include acknowledgements if you wish. This is purely your personal choice and you can choose who you wish to mention if you add an acknowledgement. There may be some situations where you have been asked to include acknowledgements, for example, if you have used company data or certain software packages that ask you to include them in the acknowledgements. Talk to your supervisor about what to include if you think this applies to you.

Abstract: This should be a summary of your whole report: your research question and objectives, your methodology and dataset you are using, your results and analysis, and your conclusions. It should be one paragraph only with no references included. The abstract should be before your content table and it does not have a section number.

Contents page: Include a contents page that is 1 to 2 pages long. Do NOT add a list of figures or list of tables.

Introduction: This should give an overview of the purpose of the project and the application it relates to and, if relevant, say what is currently being done in the industry. The research question, aims and objectives of the project should be clearly stated (you may want to have them as a sub-section).

Background: This includes your literature comparison with a suitable number of references with correct in-text citations. Give a clear overview of the technical background to the project; this should be computing based since this is a data science project. It is important that you demonstrate some in-depth critical analysis of at least four (4) or more relevant published papers in peer reviewed journals or conferences (not websites or from a thesis). A table containing lots of papers is not a good literature review unless you have also included a more detailed critical analysis of individual papers.

Start your literature comparison with an introduction to the literature on the subject and why you chose the papers that you did (what was your selection criteria). Then discuss in detail some of the key papers that are relevant to your project as a critical analysis, and put them in context of your project. It should be clear why you are discussing these papers. A critical analysis includes:

•⁠ ⁠what work was done,

•⁠ ⁠what data was used,

•⁠ ⁠what methods were used,

•⁠ ⁠what were their results and conclusions,

•⁠ ⁠how the paper relates to your project,

•⁠ ⁠your view on what is good and what is limited about their work.

Dataset: Include a section that describes your dataset. State where you got it from including a full reference to the exact website. Describe how the data was collected originally (not how you got it but how was the data made/put together), who collected it, which country and when was it collected, why was it collected, and what the data includes (this is required even if you got the data from a website). State why you chose this dataset to answer your research question and justify the reasons.

You should always look at your data before starting work on it so this section should include your Exploratory Data Analysis (EDA) that shows relevant images/tables/plots of the data. Discuss any data pre-processing you have done. Be specific and detailed about the work you did. For example, if you removed null data then, what format was the null data in before and after, how many records were affected, what was the impact on your results of changing the null data? Only discuss the pre-processing that you have done.

Ethical Issues: You must have a statement about the ethics of using your dataset. Even if there were no ethical issues identified you have to show that you thought about the issues to see if there were any relevant to your project. Include the ethical issues that you have considered regarding the dataset and project along with any evidence. Ethical things to consider (including if you got the data from a website):

•⁠ ⁠Is personal data included and if so is it anonymised?

•⁠ ⁠Does your data come under GDPR?

•⁠ ⁠Does using your data require UH ethical approval? For example, do you collect personal data from the internet (e.g. data from social media)?

•⁠ ⁠Does your project require UH ethical approval? For example, do you collect data from people or do a survey?

•⁠ ⁠Do you have permission to use the data? Is there any evidence that you can use it, for example, a creative commons licence, and if so then include a screenshot of it in your report? Do you have to pay to use the data?

•⁠ ⁠Was the data collected ethically? This can be the most difficult thing to determine. For example, if the data includes personal data then did the participants give their consent for the data to be on the website and be used for general research? If your data is from a website then you must explore the whole site to find out as much as you can about the data. Is there a reference to a published paper on the website that states how the data was collected, if so this should be discussed in your report? Was the data collected (or put on the website) by a reputable organisation so that you can assume it was collected ethically? Did the data come from another site that contains the original data in which case you should use the original data (if you do not use the original data then state why you did not)?

Methodology: This is likely to be a long section. This describes the practical work you have done. It has to be a specific description of the technical work that you have done in your project. What did you do and how did you code it? It is likely that you use the words ‘I’, ‘me’ and ‘my’ a lot in this section to show that you did this work. It needs to be as technical as you can. State the exact models you used, a technical explanation of them and why you chose them. State the exact metrics you used, a technical explanation of them and why you chose them. If you include techniques that you did not use then you will lose marks. Make sure that you improve and optimise each of the models and techniques you use and that you record the trials you made in the final report. If you do not include the results of the optimisation work you did then you will lose marks.

Results: Think carefully about which metrics you use and make sure you know what you are measuring and why for your project. For example, what is accuracy measuring for your project? In general, you should be using more than one metric. Make sure your metrics meet the project objectives. Things to think about when producing your results:

•⁠ ⁠What are the best metrics to use? Do not use just one metric - different models may perform better when measured in a different way. Consider the appropriate mathematics behind a metric and whether that suits your data type, model and research question.

•⁠ ⁠What is the best way to present your results? For example, if you are comparing models then a single plot or table comparing all three models could be a good way to see the comparison. Should you include a confusion matrix? Are your results better presented in a table or multiple plots? If your data is images then show plenty of examples of the images.

•⁠ ⁠Understand and write what each metric/result means for your project and what you are measuring.

•⁠ ⁠What do the results mean for your application or for other applications? Can your project be used in a real world situation?

•⁠ ⁠Do the results address your research question?

Analysis and discussion: The analysis section is what can turn a good report into an excellent report. Consider the following issues:

•⁠ ⁠What do your results mean?

•⁠ ⁠Which model works the best and why? What is it about the way a model or method works that makes it work well or poorly with your data?

•⁠ ⁠How do the results compare to the literature that you have discussed in your background section? Why do you think your results are better or worse than the literature?

•⁠ ⁠What are the limitations of your results?

•⁠ ⁠How do the results relate to the project objectives?

•⁠ ⁠How do the results relate to the project application/topic/research question?

•⁠ ⁠Are any of the models useable in a practical situation and if so why and how?

•⁠ ⁠Discuss whether you have answered your research question.

•⁠ ⁠Can you draw comparison to the literature work?

Conclusion: This should be a short section (between one paragraph and two pages). It should include:

•⁠ ⁠a summary of the key results,

•⁠ ⁠your justified conclusions,

•⁠ ⁠the applications and real world situations that your work can be used for,

•⁠ ⁠the future work you would recommend (what would you do next if you carried on with the project?).

References: This should be a full list of all the references that you cite in your report. All references should have an in-text citation and all in-text citations should be in the references list. The references must include the peer reviewed journals that the papers were published in. If most of your references are websites you will lose marks. The references must be in Harvard format (author name and year). The reference list must be in alphabetical order based on the surname of the first author (not numbering). For the correct formatting of your reference list and in-text citations see Section 8 ‘Literature Review and Referencing’. You can shorten conference proceedings e.g. 4th Int. Conf. Comms. & Comp. Tech. You will lose marks for getting the formatting wrong for your citations or for your reference list. It is an academic offence to include any fake references.

Appendices: The appendices provide supporting evidence of the quality and quantity of the work you have done. Include information that you think is relevant as an appendix. Discuss what to include in your appendices with your supervisor. Do NOT include a timeline into the report (this was a task for your Project Plan). In your appendices include the following:

1.⁠ ⁠Extra plots or images: If you have a lot of plots or images in your results consider putting just the most important ones into the report (a small selection that show the main results) and the rest of them into an appendix.

2.⁠ ⁠Other information that you think is relevant.

You do need to include your code to the report, as an appendix. Additionally, the code should also be included in your GitHub site with a working hyper-link address on the front page of the Final Project Report.

Note 1: give me references section wise for my understanding(eg: reference for introduction, reference for methodology, references for literature review etc)  
  
note 2 : report writing should be accompanied with proper referencing (don’t add references only at the end)

Note 3. All references in the reference section must contain accessed dates, clickable website link, clickable DOI link.   
  
note 5: suggest me where I need to add what plots/tables

Note 6: add clickable links to general concepts/algorithm explaining images from website to add (eg:random forest,MLPClassifier,KNN,SMOTE,ADASYN,RandomOverSampler,SMOTETomek, F1 score,Recall, ROC\_AUC etc)

Note 7: ensure to add “ emphasis on your**project aims and objectives** in the introduction.”  
  
the following is the code of the project and output is either pasted or outputs’ githublinks are provided

Note 8: have 20-25 hardward stayle reference (including the 4 that I have provided”

Note 9: below is the code and output. Ignore the numbering of the code.but consider this as the final code and order(since I have deleted unnecessary sections)

Note 10; don’t write anything else in the report that we have not done in the following code

# import packages

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from matplotlib import gridspec

from sklearn.preprocessing import LabelEncoder

from sklearn.preprocessing import StandardScaler

from scipy.stats import chi2\_contingency, ttest\_ind

from sklearn.model\_selection import train\_test\_split

from imblearn.over\_sampling import SMOTE, ADASYN, RandomOverSampler

from imblearn.combine import SMOTETomek

from sklearn.ensemble import RandomForestClassifier

from sklearn.neural\_network import MLPClassifier

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score, matthews\_corrcoef, roc\_auc\_score, log\_loss

from sklearn.model\_selection import GridSearchCV, StratifiedKFold

import time

import warnings

warnings.filterwarnings("ignore")

from sklearn.exceptions import ConvergenceWarning

from sklearn.metrics import roc\_curve, auc

from sklearn.metrics import confusion\_matrix

from sklearn.model\_selection import validation\_curve, learning\_curve

from collections import defaultdict

from sklearn.utils import shuffle

from sklearn.exceptions import NotFittedError

from sklearn.metrics import log\_loss

from sklearn.model\_selection import train\_test\_split

from sklearn.datasets import make\_classification

from IPython.display import display

from sklearn.metrics import classification\_report

from scipy import stats

from sklearn.base import clone

Data Preprocessing

# Load the dataset

df = pd.read\_excel("/content/sample\_data/ai4i2020.xlsx")

# save a copy

data = df.copy()

# check the structure of the dataset

df.head()

# Check and print the shape of the dataset

print("\nShape of Dataset (rows, columns):")

print(df.shape)

Shape of Dataset (rows, columns):

(10000, 15)

# Check and print column names

print("\nColumn Names:")

print(df.columns.tolist())

Column Names:

['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', 'Failure Type']

# Rename columns

df.rename(columns = {

'UDI' : 'UID',

'Product ID' : 'Product\_ID',

'Air temperature [K]' : 'Air\_temperature',

'Process temperature [K]' : 'Process\_temperature',

'Rotational speed [rpm]' : 'Rotational\_speed',

'Torque [Nm]' : 'Torque',

'Tool wear [min]' : 'Tool\_wear',

'Machine failure' : 'Machine\_failure',

'Failure Type' : 'Failure\_type'

}, inplace = True)

# Display the first 5 rows of the DataFrame

df.head()

github link of the output : https://github.com/JencyFrancis/26th-aug/blob/main/df.head%20after%20renaming.png

# Check datatypes of each column

df.dtypes

output:

|  |  |
| --- | --- |
| **UID** | int64 |
| **Product\_ID** | object |
| **Type** | object |
| **Air\_temperature** | float64 |
| **Process\_temperature** | float64 |
| **Rotational\_speed** | int64 |
| **Torque** | float64 |
| **Tool\_wear** | int64 |
| **Machine\_failure** | int64 |
| **TWF** | int64 |
| **HDF** | int64 |
| **PWF** | int64 |
| **OSF** | int64 |
| **RNF** | int64 |
| **Failure\_type** | object |

# Convert numeric columns to float

df['Rotational\_speed'] = df['Rotational\_speed'].astype('float64')

df['Tool\_wear'] = df['Tool\_wear'].astype('float64')

# Check for null values

print("\nNull Values per Column:")

print(df.isnull().sum())

Null Values per Column:

UID 0

Product\_ID 0

Type 0

Air\_temperature 0

Process\_temperature 0

Rotational\_speed 0

Torque 0

Tool\_wear 0

Machine\_failure 0

TWF 0

HDF 0

PWF 0

OSF 0

RNF 0

Failure\_type 0

# Check for duplicate rows

print("\nNumber of Duplicate Rows:")

print(df.duplicated().sum())

Number of Duplicate Rows:

0

# Check for any columns with string values

print("\nChecking for string columns:")

for col in df.columns:

if df[col].dtype == 'object':

print(f"{col}: {df[col].unique()[:10]}")

hecking for string columns:

Product\_ID: ['M14860' 'L47181' 'L47182' 'L47183' 'L47184' 'M14865' 'L47186' 'L47187'

'M14868' 'M14869']

Type: ['M' 'L' 'H']

Failure\_type: ['No Failure' 'Power Failure' 'Tool Wear Failure' 'Overstrain Failure'

'Random Failures' 'Heat Dissipation Failure']

# Check for unique values in "Product ID" column

if "Product\_ID" in df.columns:

print("\nNumber of Unique Product IDs:", df['Product\_ID'].nunique())

Number of Unique Product IDs: 10000

# Extract Product IDs

df['Product\_ID\_clean'] = [''.join(filter(str.isdigit, pid)) for pid in df['Product\_ID']]

print(df['Product\_ID\_clean'])

0 14860

1 47181

2 47182

3 47183

4 47184

...

9995 24855

9996 39410

9997 24857

9998 39412

9999 24859

Name: Product\_ID\_clean, Length: 10000

# Check summary statistics

print("\nStatistical Summary:")

df.describe()

Statistical Summary:

|  | **UID** | **Air\_temperature** | **Process\_temperature** | **Rotational\_speed** | **Torque** | **Tool\_wear** | **Machine\_failure** | **TWF** | **HDF** | **PWF** | **OSF** | **RNF** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **count** | 10000.00000 | 10000.000000 | 10000.000000 | 10000.000000 | 10000.000000 | 10000.000000 | 10000.000000 | 10000.000000 | 10000.000000 | 10000.000000 | 10000.000000 | 10000.00000 |
| **mean** | 5000.50000 | 300.004930 | 310.005560 | 1538.776100 | 39.986910 | 107.951000 | 0.033900 | 0.004600 | 0.011500 | 0.009500 | 0.009800 | 0.00190 |
| **std** | 2886.89568 | 2.000259 | 1.483734 | 179.284096 | 9.968934 | 63.654147 | 0.180981 | 0.067671 | 0.106625 | 0.097009 | 0.098514 | 0.04355 |
| **min** | 1.00000 | 295.300000 | 305.700000 | 1168.000000 | 3.800000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.00000 |
| **25%** | 2500.75000 | 298.300000 | 308.800000 | 1423.000000 | 33.200000 | 53.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.00000 |
| **50%** | 5000.50000 | 300.100000 | 310.100000 | 1503.000000 | 40.100000 | 108.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.00000 |
| **75%** | 7500.25000 | 301.500000 | 311.100000 | 1612.000000 | 46.800000 | 162.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.00000 |
| **max** | 10000.00000 | 304.500000 | 313.800000 | 2886.000000 | 76.600000 | 253.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.00000 |

# Filter rows where Failure Type is "Random Failures" AND Machine failure = 0

rf\_df = df[(df['Failure\_type'] == 'Random Failures') & (df['Machine\_failure'] == 0)]

# Display the result

result = rf\_df[['Machine\_failure', 'Failure\_type']]

print(f"Found {len(result)} entries")

print(result)

# Get indices of rows to remove from df

indices\_to\_drop = df[(df['Failure\_type'] == 'Random Failures') & (df['Machine\_failure'] == 0)].index

# Drop these rows from df

df.drop(indices\_to\_drop, inplace=True)

# Verify removal

print(f"Removed {len(indices\_to\_drop)} entries")

print(df['Failure\_type'].value\_counts())

Removed 18 entries

Failure\_type

No Failure 9652

Heat Dissipation Failure 112

Power Failure 95

Overstrain Failure 78

Tool Wear Failure 45

# Check shape after removal

print("DataFrame shape after removal:", df.shape)

DataFrame shape after removal: (9982, 16)

# Filter rows where Failure Type is "No Failure" AND Machine failure = 1

nf\_df = df[(df['Failure\_type'] == 'No Failure') & (df['Machine\_failure'] == 1)]

# Display the result

result = nf\_df[['Machine\_failure', 'Failure\_type']]

print(f"Found {len(result)} inconsistent entries:")

print(result)

Found 9 inconsistent entries:

Machine\_failure Failure\_type

1437 1 No Failure

2749 1 No Failure

4044 1 No Failure

4684 1 No Failure

5536 1 No Failure

5941 1 No Failure

6478 1 No Failure

8506 1 No Failure

9015 1 No Failure

# Get indices of rows to remove from df

indices\_to\_drop\_2 = df[(df['Failure\_type'] == 'No Failure') & (df['Machine\_failure'] == 1)].index

# Drop these rows from df

df.drop(indices\_to\_drop\_2, inplace=True)

# Verify removal

print(f"Removed {len(indices\_to\_drop\_2)} inconsistent entries from df")

print("Final df shape:", df.shape)

Removed 9 inconsistent entries from df

Final df shape: (9973, 16)

# Target variable distribution (Class Imbalance Analysis)

class\_counts = df['Machine\_failure'].value\_counts()

print(f"No Failure (0): {class\_counts[0]} ({class\_counts[0]/len(df)\*100:.2f}%)")

print(f"Failure (1): {class\_counts[1]} ({class\_counts[1]/len(df)\*100:.2f}%)")

print(f"Imbalance Ratio: {class\_counts[0]/class\_counts[1]:.2f}:1")

No Failure (0): 9643 (96.69%)

Failure (1): 330 (3.31%)

Imbalance Ratio: 29.22:1

# Save a copy after preprocessing

df1 = df.copy()

Feature Preparation

# Remove non-predictive columns

columns\_to\_drop = ['UID', 'Product\_ID', 'Failure\_type']

X = df.drop(columns = ['Machine\_failure'] + columns\_to\_drop)

Feature Encoding

# Encode product type in df

le = LabelEncoder()

df1['Type\_encoded'] = le.fit\_transform(df1['Type'])

# Show the mapping

print("Type encoding mapping:")

for i, label in enumerate(le.classes\_):

print(f" '{label}' -> {i}")

Type encoding mapping:

'H' -> 0

'L' -> 1

'M' -> 2

# Features for importance analysis

numeric\_features = ['Air\_temperature', 'Process\_temperature', 'Rotational\_speed',

'Torque', 'Tool\_wear', 'Type\_encoded']

# Calculate correlation-based importance

correlations = []

for feature in numeric\_features:

corr = abs(df1[feature].corr(df1['Machine\_failure']))

correlations.append(corr)

# Create feature importance table

feature\_names = ['Air Temperature', 'Process Temperature', 'Rotational Speed',

'Torque', 'Tool Wear', 'Machine Type']

# Create a DataFrame

importance\_df = pd.DataFrame({

'Feature': feature\_names,

'Importance': correlations

}).sort\_values('Importance', ascending = False)

# Assign Rank (1 = most important, n = least important)

importance\_df['Rank'] = range(1, len(importance\_df) + 1)

# Print heading

print("FEATURE IMPORTANCE RANKING (1 = Most Important):")

# Display the table

display(importance\_df.style.hide(axis='index').format({'Importance': '{:.4f}'}))

FEATURE IMPORTANCE RANKING (1 = Most Important):

| **Feature** | **Importance** | **Rank** |
| --- | --- | --- |
| Torque | 0.1934 | 1 |
| Tool Wear | 0.1063 | 2 |
| Air Temperature | 0.0831 | 3 |
| Rotational Speed | 0.0440 | 4 |
| Process Temperature | 0.0360 | 5 |
| Machine Type | 0.0065 | 6 |

# Plot feature importance as a horizontal bar chart

plot\_df = importance\_df.sort\_values(by = 'Importance', ascending = False)

# Create figure

plt.figure(figsize = (10, 6))

# Horizontal bar plot

bars = plt.barh(range(len(plot\_df)), plot\_df['Importance'], color = 'lime')

# Reverse the y-axis order to put highest at top

plt.gca().invert\_yaxis()

# Y-ticks

plt.yticks(range(len(plot\_df)), plot\_df['Feature'])

# Choose axes labels

plt.xlabel('Importance', fontsize = 12)

plt.ylabel('Feature', fontsize = 12)

# Choose title

plt.title('Feature Importance', fontsize = 14)

# Set X-axis limit

plt.xlim(0, 0.25)

# Add feature importance values onto each bars

for i, bar in enumerate(bars):

width = bar.get\_width()

plt.text(width + 0.005, bar.get\_y() + bar.get\_height()/2,

f'{width:.4f}', ha = 'left', va = 'center', fontsize = 10)

# Adjust layout

plt.tight\_layout()

# Display the plot

plt.show()

github link of the output: <https://github.com/JencyFrancis/26th-aug>

#distribution plots  
# Failure Class Distribution Plot

def plot\_class\_distribution(data, fig, gs):

"""Plot class distribution pie chart"""

# Create a subplot in the given figure and gridspec position

ax1 = fig.add\_subplot(gs[0, 0])

# Count occurrences of each class in 'Machine failure' column

machine\_failure\_counts = data['Machine failure'].value\_counts()

# Choose colors and labels

colors = ['lime', 'gold']

labels = ['No Failure', 'Failure']

# Generate the pie chart

wedges, texts, autotexts = ax1.pie(machine\_failure\_counts.values,

labels = None,

autopct = '%.2f%%',

colors = colors,

startangle = 90,

wedgeprops = {'linewidth': 1, 'edgecolor': 'white', 'alpha': 0.8})

# Add legend

ax1.legend(wedges, labels,

title = "Failure Status",

loc = "upper left",

bbox\_to\_anchor = (1, 0, 0.5, 1))

# Set the title

ax1.set\_title('Failure Class Distribution', fontsize = 14, fontweight = 'bold')

# Create figure

fig = plt.figure(figsize = (12, 6))

gs = gridspec.GridSpec(1, 2)

# Call the function

plot\_class\_distribution(data, fig, gs)

# Adjust layout

plt.tight\_layout()

# Display plot

plt.show()

github link of the output:

<https://github.com/JencyFrancis/26th-aug/blob/main/failure%20class%20distribution.png>

# Machine Type Distribution Plot

def plot\_type\_distribution(data, fig, gs):

"""Plot type distribution pie chart"""

# Create a subplot in the given figure and gridspec position

ax1 = fig.add\_subplot(gs[0, 0])

# Count occurrences of each class in 'Type' column

machine\_type\_counts = data['Type'].value\_counts()

# Choose colors and labels

colors = ['lime', 'violet', 'gold']

labels = ['L', 'M', 'H']

# Generate the pie chart

wedges, texts, autotexts = ax1.pie(machine\_type\_counts.values,

labels = None,

autopct = '%.2f%%',

colors = colors,

startangle = 90,

wedgeprops = {'linewidth': 1, 'edgecolor': 'white', 'alpha': 0.8})

# Add legend

ax1.legend(wedges, labels,

title = "Machine Type",

loc = "upper left",

bbox\_to\_anchor = (1, 0, 0.5, 1))

# Set the title

ax1.set\_title('Machine Type Distribution', fontsize = 14, fontweight = 'bold')

# Create figure

fig = plt.figure(figsize = (12, 6))

gs = gridspec.GridSpec(1, 2)

# Call the function

plot\_type\_distribution(data, fig, gs)

# # Adjust layout

plt.tight\_layout()

# Display plot

plt.show()

github link of the output:  
<https://github.com/JencyFrancis/26th-aug/blob/main/machine%20type%20distribution.png>

# Machine Failure Type distribution after pre-processing

# Calculate failure counts for machine failures

failure\_counts = df1[df1['Machine\_failure'] == 1]['Failure\_type'].value\_counts()

# Ensure consistent category order

categories = ['Power Failure', 'Overstrain Failure', 'Tool Wear Failure', 'Heat Dissipation Failure']

# Choose consistent color mapping

category\_colors = {

'Power Failure': 'gold',

'Overstrain Failure': 'violet',

'Tool Wear Failure': 'lime',

'Heat Dissipation Failure': 'orange'

}

# Reorder counts according to predefined categories

failure\_counts = [failure\_counts.get(cat, 0) for cat in categories]

# Create pie chart with consistent category-color mapping

plt.pie(

failure\_counts,

labels = categories,

autopct = '%.2f%%',

colors=[category\_colors[cat] for cat in categories],

wedgeprops = {'linewidth': 1, 'edgecolor': 'white'}

)

# Set title

plt.title('Original Machine Failure Type Distribution', fontweight = 'bold')

# Set layout

plt.tight\_layout()

# Display plot

plt.show()

github link of the output:

<https://github.com/JencyFrancis/26th-aug/blob/main/original%20machine%20failure%20type%20distribution.png>

# Machine Failure Analysis

# Create figure

fig = plt.figure(figsize = (16, 9))

# Set title

fig.suptitle('Machine Failure Analysis', fontsize = 18, fontweight = 'bold', y = 0.94)

# Create gridspec layout for two plots side by side

gs = fig.add\_gridspec(1, 2, hspace = 0.3, wspace = 0.4)

# Create subplot axes

ax1 = fig.add\_subplot(gs[0, 0])

ax2 = fig.add\_subplot(gs[0, 1])

# 1. Distribution of Failure Types

# Count occurrences of each class in 'Failure Type' column

failure\_counts = data['Failure Type'].value\_counts()

# Create a bar plot

bars = ax1.bar(failure\_counts.index, failure\_counts.values, color = 'crimson', alpha = 0.8)

# Set title

ax1.set\_title('Distribution of Machine Failure Types', fontsize = 14, pad = 10)

# Set labels

ax1.set\_xlabel('Failure Type', fontsize = 12)

ax1.set\_ylabel('Count', fontsize = 12)

# Set axes ticks

ax1.tick\_params(axis = 'x', rotation = 75, labelsize = 10)

ax1.tick\_params(axis = 'y', labelsize = 10)

# Add value labels on top of each bar

for bar in bars:

height = bar.get\_height()

ax1.annotate(f'{int(height)}', xy = (bar.get\_x() + bar.get\_width() / 2, height),

xytext = (0, 5), textcoords = "offset points", ha = 'center', va = 'bottom',

fontsize = 11, fontweight = 'bold')

# Set y-axis range and ticks for first plot

ax1.set\_ylim(0, 10000)

ax1.set\_yticks(range(0, 11001, 1000))

# 2. Machine Failures by Product Type

# Create a cross-tabulation contingency table to analyze the relationship between Type and Machine Failure

failure\_by\_type = pd.crosstab(data['Type'], data['Machine failure'])

# Create bar plot

bars2 = failure\_by\_type.plot(kind='bar', ax = ax2, color = ['limegreen', 'crimson'], alpha = 0.8)

# Set title

ax2.set\_title('Machine Failures by Product Type', fontsize = 14, pad = 10)

# Set label

ax2.set\_xlabel('Product Type', fontsize = 12)

ax2.set\_ylabel('Count', fontsize = 12)

# Create legend

ax2.legend(['No Failure', 'Machine Failure'], fontsize = 11, frameon = True, fancybox = True, shadow = False)

# Set axes ticks

ax2.tick\_params(axis = 'x', rotation = 0, labelsize = 12)

ax2.tick\_params(axis = 'y', labelsize = 10)

# Add value labels on top of each bar

for container in ax2.containers:

ax2.bar\_label(container, fmt = '%d', label\_type = 'edge', fontsize = 11, fontweight = 'bold')

# Set y-axis range and ticks

ax2.set\_ylim(0, 6500)

ax2.set\_yticks(range(0, 6501, 500))

# Set layout

plt.subplots\_adjust(top = 0.88, bottom = 0.15, left = 0.08, right = 0.95, hspace = 0.2, wspace = 0.3)

# Display plot

plt.show()

github link of the output:  
<https://github.com/JencyFrancis/26th-aug/blob/main/machine%20failure%20analysis.png>

# Specific Failure Types by Product Type after pre-processing

# Create figure

fig, ax = plt.subplots(figsize = (8, 6))

# Filter out non-relevant failure types

specific\_failures = df1[(df1['Failure\_type'] != 'No Failure') &

(df1['Failure\_type'] != 'Random Failures')]

# Create cross-tabulation of failures by product type

failure\_product = pd.crosstab(specific\_failures['Type'],

specific\_failures['Failure\_type'])

# Create stacked bar plot

failure\_product.plot(kind = 'bar', stacked = True, ax = ax,

color = ['cornflowerblue', 'lightcoral', 'mediumseagreen', 'plum'],

alpha = 0.8)

# Set title

ax.set\_title('Specific Failure Types by Product Type',

fontsize = 13, fontweight = 'normal', pad = 10)

# Name labels

ax.set\_xlabel('Product Type', fontsize = 10)

ax.set\_ylabel('Number of Failures', fontsize = 10)

# Set axes limits and ticks

ax.set\_ylim(0, 250)

ax.set\_yticks(range(0, 251, 50))

ax.tick\_params(axis = 'x', rotation = 0)

# Add legend

ax.legend(loc = 'upper right', fontsize = 8)

# Add total count labels on bars

for i, product\_type in enumerate(failure\_product.index):

total = failure\_product.loc[product\_type].sum()

ax.text(i, total + 5, f'{int(total)}',

ha = 'center', va = 'bottom', fontsize = 10, fontweight = 'bold')

# Set layout

plt.tight\_layout()

# Display plot

plt.show()

github link of the output:

<https://github.com/JencyFrancis/26th-aug/blob/main/specific%20failure%20types%20by%20product%20type.png>

# Distribution of Original Variables

# Create figure

plt.figure(figsize = (16, 14))

# Set main title

plt.suptitle('Distribution of Original Variables', fontsize = 18, y = 1.02, fontweight = 'bold')

# Choose colors

hist\_color = 'steelblue'

kde\_color = 'red'

# Define the numerical features to plot

features = ['Air\_temperature', 'Process\_temperature', 'Rotational\_speed', 'Torque', 'Tool\_wear']

# Loop through each feature to create distribution plots

for i, col in enumerate(features):

# Create subplot for each feature

ax = plt.subplot(2, 3, i + 1)

# Plot histogram

sns.histplot(df1[col], bins = 30, color = hist\_color, edgecolor = 'white',

alpha = 0.8, kde = True, line\_kws = {'color': kde\_color, 'lw': 2})

# Set title

ax.set\_title(col, fontsize = 13, pad = 10, fontweight = 'semibold')

# Set axes labels

ax.set\_xlabel('Value', fontsize = 10, fontweight = 'semibold')

ax.set\_ylabel('Count', fontsize = 10, fontweight = 'semibold')

# Set custom x-axis ranges and tick spacing

if col == 'Torque':

ax.set\_xlim(left = 0)

ax.set\_xticks(range(0, 81, 10))

ax.set\_yticks(range(0, 1001, 100))

elif col == 'Tool\_wear':

ax.set\_xlim(left = -50)

ax.set\_xticks(range(-50, 301, 25))

ax.set\_yticks(range(0, 451, 50))

# Remove empty subplot

plt.delaxes(plt.subplot(2, 3, 6))

# Adjust layout

plt.tight\_layout(h\_pad = 4.0, w\_pad = 3.0)

# Display plot

plt.show()

github link of the output:

<https://github.com/JencyFrancis/26th-aug/blob/main/distribution%20of%20original%20variables.png>

# Original Feature Distribution

# Define numerical features

numerical\_features = ['Air\_temperature', 'Process\_temperature', 'Rotational\_speed', 'Torque', 'Tool\_wear']

# Create subplot grid layout

figure, axes = plt.subplots(nrows = 2, ncols = 3, figsize = (18, 7))

# Create main title

figure.suptitle('Original Feature distribution', fontsize = 16, fontweight = 'bold')

# Generate KDE plots for each numerical feature

for index, feature\_name in enumerate(numerical\_features):

# Calculate row and column positions for subplot grid

row\_position = index // 3

col\_position = index % 3

# Create KDE plot for machine failure classes

sns.kdeplot(ax = axes[row\_position, col\_position],

data = df1,

x = feature\_name,

hue = 'Machine\_failure',

fill = True,

palette = ['steelblue', 'darkorange'],

alpha = 0.7)

# Map legend labels for machine failure status

current\_legend = axes[row\_position, col\_position].get\_legend()

if current\_legend:

current\_legend.set\_title(' Failure Status')

legend\_labels = current\_legend.get\_texts()

if len(legend\_labels) >= 2:

legend\_labels[0].set\_text('No Failure')

legend\_labels[1].set\_text('Failure')

# Set axes labels

axes[row\_position, col\_position].set\_xlabel('Value', fontsize = 12, fontweight = 'semibold')

axes[row\_position, col\_position].set\_ylabel('Density', fontsize = 12, fontweight = 'semibold')

axes[row\_position, col\_position].set\_title(f'{feature\_name}', fontsize = 12, fontweight = 'semibold', pad = 10)

# Set custom y-tick spacing

if feature\_name == 'Tool\_wear':

y\_max = 0.005

axes[row\_position, col\_position].set\_ylim(0, y\_max)

axes[row\_position, col\_position].set\_yticks(np.linspace(0, y\_max, 6))

axes[row\_position, col\_position].yaxis.set\_major\_formatter(plt.FuncFormatter(lambda x, p: f'{x:.3f}'))

# Remove empty subplot

figure.delaxes(axes[1, 2])

# Adjust layout

plt.tight\_layout(h\_pad = 3.0, w\_pad = 2.5)

# Display plot

plt.show()

github link of the output :  
<https://github.com/JencyFrancis/26th-aug/blob/main/original%20feature%20distribution.png>

#correlation analysis

# Correlation Analysis

# Convert categorical Failure\_type to numeric codes

df1['Failure\_type'] = pd.factorize(df1['Failure\_type'])[0]

# Analysis with numerical features

numerical\_features = ['Type\_encoded', 'Air\_temperature', 'Process\_temperature',

'Rotational\_speed', 'Torque', 'Tool\_wear', 'Failure\_type']

# Compute Pearson correlation matrix

corr\_matrix = df1[numerical\_features + ['Machine\_failure']].corr()

# Create figure

plt.figure(figsize = (8, 6))

# Mask upper triangle

mask = np.triu(np.ones\_like(corr\_matrix, dtype = bool))

# Heatmap

sns.heatmap(corr\_matrix, mask = mask, annot = True, fmt = ".3f", cmap = 'coolwarm',

vmin = -1, vmax = 1, linewidths = 0.5, linecolor = 'white')

# Set title

plt.title('Feature Correlation with Machine Failure', fontsize = 14, fontweight = 'bold', pad = 10)

# Set layout

plt.tight\_layout()

# Correlation results output

print("\nFeature correlations with Machine failure:")

print(corr\_matrix['Machine\_failure'].sort\_values(ascending = False).drop('Machine\_failure').to\_string(float\_format = "%.3f"))

# Display plot

plt.show()

github link of the output:

<https://github.com/JencyFrancis/26th-aug/blob/main/feature%20correlation%20with%20machine%20learning.png>

#outlier inspection

# Define numerical features for analysis

features = ['Air\_temperature', 'Process\_temperature', 'Rotational\_speed', 'Torque', 'Tool\_wear']

# Define custom x-axis ranges for each feature

x\_ranges = {

'Air\_temperature': (294, 306),

'Process\_temperature': (304, 316),

'Rotational\_speed': (974, 3250),

'Torque': (-10, 100),

'Tool\_wear': (-100, 350)

}

# Create box plots for outlier inspection

plt.figure(figsize = (16, 8))

# Set main title

plt.suptitle('Box Plots for Outlier Detection', fontsize = 16, fontweight = 'bold')

# Generate box plots for each feature

for i, feature in enumerate(features):

# Create subplot grid (2 rows, 3 columns)

plt.subplot(2, 3, i + 1)

# Create horizontal box plot

sns.boxplot(data = df1, y = 'Machine\_failure', x = feature, hue = 'Machine\_failure',

palette = ['orange', 'lightgreen'], legend = False, orient = 'h')

# Set subplot titles

plt.title(f'{feature}', fontsize = 12, fontweight = 'semibold')

# Set axes labels

plt.xlabel('Value', fontsize = 10, fontweight = 'semibold')

plt.ylabel('Machine Failure Status', fontsize = 10, fontweight = 'semibold')

# Map Y-axis labels

plt.yticks([0, 1], ['No Failure', 'Failure'], fontweight = 'semibold')

# Set custom x-axis range for each feature

plt.xlim(x\_ranges[feature])

# Choose axis ticks to include range boundaries for Torque

if feature == 'Torque':

plt.xticks(range(-10, 101, 10))

# Remove empty subplot

plt.delaxes(plt.subplot(2, 3, 6))

# Adjust layout

plt.tight\_layout(h\_pad = 3.0, w\_pad = 2.5)

# Display the plot

plt.show()

github link of the output:

<https://github.com/JencyFrancis/26th-aug/blob/main/box%20plots%20for%20outlier%20inspection.png>

# feature scaling

# Scale numerical features

# Initialize StandardScaler to normalize numerical features (mean=0, std=1)

scaler = StandardScaler()

# List of numerical columns to be scaled

numerical\_cols = ['Air\_temperature', 'Process\_temperature', 'Rotational\_speed','Torque', 'Tool\_wear']

# Create a copy of the original DataFrame

df1\_scaled = df1.copy()

# Apply scaling to the selected numerical columns

df1\_scaled[numerical\_cols] = scaler.fit\_transform(df1[numerical\_cols])

# Display first 5 rows of scaled DataFrame

df1\_scaled.head()

output :

| **UID** | **Product\_ID** | **Type** | **Air\_temperature** | **Process\_temperature** | **Rotational\_speed** | **Torque** | **Tool\_wear** | **Machine\_failure** | **TWF** | **HDF** | **PWF** | **OSF** | **RNF** | **Failure\_type** | **Product\_ID\_clean** | **Type\_encoded** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 1 | M14860 | M | -0.951417 | -0.946356 | 0.067484 | 0.283054 | -1.695647 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 14860 | 2 |
| **1** | 2 | L47181 | L | -0.901428 | -0.878954 | -0.729604 | 0.634238 | -1.648511 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 47181 | 1 |
| **2** | 3 | L47182 | L | -0.951417 | -1.013759 | -0.227940 | 0.945286 | -1.617087 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 47182 | 1 |
| **3** | 4 | L47183 | L | -0.901428 | -0.946356 | -0.590253 | -0.048061 | -1.585664 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 47183 | 1 |
| **4** | 5 | L47184 | L | -0.901428 | -0.878954 | -0.729604 | 0.002108 | -1.554240 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 47184 | 1 |

# Print all column names

print("\nAll columns in data:")

print(df.columns.tolist())

print("\nAll columns in df1:")

print(df1\_scaled.columns.tolist())

print("\nAll columns in df4\_scaled:")

print(df1\_scaled.columns.tolist())

# Print shape

print(f"\nOriginal shape: {data.shape}")

print(f"Feature Engineered shape: {df1\_scaled.shape}")

print(f"Scaled df1 shape: {df1\_scaled.shape}")

All columns in data:

['UID', 'Product\_ID', 'Type', 'Air\_temperature', 'Process\_temperature', 'Rotational\_speed', 'Torque', 'Tool\_wear', 'Machine\_failure', 'TWF', 'HDF', 'PWF', 'OSF', 'RNF', 'Failure\_type', 'Product\_ID\_clean']

All columns in df1:

['UID', 'Product\_ID', 'Type', 'Air\_temperature', 'Process\_temperature', 'Rotational\_speed', 'Torque', 'Tool\_wear', 'Machine\_failure', 'TWF', 'HDF', 'PWF', 'OSF', 'RNF', 'Failure\_type', 'Product\_ID\_clean', 'Type\_encoded']

All columns in df4\_scaled:

['UID', 'Product\_ID', 'Type', 'Air\_temperature', 'Process\_temperature', 'Rotational\_speed', 'Torque', 'Tool\_wear', 'Machine\_failure', 'TWF', 'HDF', 'PWF', 'OSF', 'RNF', 'Failure\_type', 'Product\_ID\_clean', 'Type\_encoded']

Original shape: (10000, 15)

Feature Engineered shape: (9973, 17)

Scaled df1 shape: (9973, 17)

# statistical significance test

# Initialize results list

results = []

# Chi-square tests (Type vs flags)

for flag in ['TWF', 'HDF', 'PWF', 'OSF']:

chi2, p = chi2\_contingency(pd.crosstab(df1\_scaled['Type\_encoded'], df1\_scaled[flag]))[:2]

results.append({

'Important Feature': flag,

'Result': 'Significant association' if p < 0.05 else 'No association',

'Test Type': 'Chi-square',

'Test Statistic': f"{chi2:.3f}",

'p-value': f"{p:.4f}"

})

# T-tests (Numerical features vs failure)

for num\_var in ['Air\_temperature', 'Process\_temperature', 'Rotational\_speed', 'Torque', 'Tool\_wear']:

t\_stat, p = ttest\_ind(

df1\_scaled[df1\_scaled['Machine\_failure'] == 1][num\_var],

df1\_scaled[df1\_scaled['Machine\_failure'] == 0][num\_var],

equal\_var = False

)

results.append({

'Important Feature': num\_var,

'Result': 'Significant difference' if p < 0.05 else 'No difference',

'Test Type': 'T-test',

'Test Statistic': f"{t\_stat:.3f}",

'p-value': f"{p:.4f}"

})

# Create styled DataFrame

styled\_df = (

pd.DataFrame(results)

.style

.hide(axis="index")

.set\_properties(\*\*{'text-align': 'center'})

.set\_table\_styles([

{'selector': 'th, td', 'props': 'border: 1px solid black;'},

{'selector': 'th', 'props': 'background-color: lightgray;'}

])

)

# Display table

display(styled\_df)

github link of the output:

<https://github.com/JencyFrancis/26th-aug/blob/main/statistical%20significance%20test%20result.png>

# data preparation

# DATA PREPARATION FOR MODELING

# =============================

# Select features for modeling

feature\_columns = ['Air\_temperature', 'Process\_temperature', 'Rotational\_speed',

'Torque', 'Tool\_wear', 'Type\_encoded']

# IMPORTANT: Create UNSCALED data first (from df1, NOT df1\_scaled)

X\_unscaled = df1[feature\_columns] # From original df1

y = df1['Machine\_failure']

# Split the UNSCALED data

X\_train\_unscaled, X\_test\_unscaled, y\_train, y\_test = train\_test\_split(

X\_unscaled, y, test\_size=0.2, random\_state=17, stratify=y

)

# Now create SCALED versions for MLP and KNN

scaler = StandardScaler()

X\_train\_scaled = scaler.fit\_transform(X\_train\_unscaled) # Returns NumPy array

X\_test\_scaled = scaler.transform(X\_test\_unscaled) # Returns NumPy array

# Print the sizes and distribution

print("Data Split Information:")

print(f"Training set size: {X\_train\_unscaled.shape}")

print(f"Test set size: {X\_test\_unscaled.shape}")

print(f"Training set failure rate: {y\_train.mean():.4f}")

print(f"Test set failure rate: {y\_test.mean():.4f}")

print(f"\nData types:")

print(f"Unscaled data type: {type(X\_train\_unscaled)}") # DataFrame

print(f"Scaled data type: {type(X\_train\_scaled)}") # NumPy array

output :

Data Split Information:

Training set size: (7978, 6)

Test set size: (1995, 6)

Training set failure rate: 0.0331

Test set failure rate: 0.0331

Data types:

Unscaled data type: <class 'pandas.core.frame.DataFrame'>

Scaled data type: <class 'numpy.ndarray'>

# SECTION 1: BASELINE MODEL TRAINING WITH LOSS TRACKING

# =====================================================

# Initialize models

models = {

'Random Forest': RandomForestClassifier(random\_state=17),

'MLP Classifier': MLPClassifier(

hidden\_layer\_sizes=(128, 64, 32),

max\_iter=200,

learning\_rate\_init=0.001,

random\_state=17,

early\_stopping=True,

validation\_fraction=0.1,

n\_iter\_no\_change=5,

verbose=False # We'll handle output manually

),

'KNN': KNeighborsClassifier()

}

# Initialize storage for baseline results

baseline\_results = {}

baseline\_probabilities = {}

baseline\_predictions = {}

baseline\_training\_history = {}

mlp\_epoch\_details = []

# Train each model with appropriate data

print("\n" + "="\*80)

print("TRAINING BASELINE MODELS (Without Oversampling)")

print("="\*80)

for name, model in models.items():

print(f"\nTraining {name}...")

# Select appropriate data based on model type

if name == 'Random Forest':

X\_train\_use = X\_train\_unscaled

X\_test\_use = X\_test\_unscaled

print(" Using: Unscaled data (DataFrame)")

else:

X\_train\_use = X\_train\_scaled

X\_test\_use = X\_test\_scaled

print(" Using: Scaled data (NumPy array)")

# Train model on ACTUAL data

start\_time = time.time()

if name == 'MLP Classifier':

# Custom training loop to capture per-epoch metrics

from sklearn.model\_selection import train\_test\_split as tts

# Split for validation

X\_train\_mlp, X\_val\_mlp, y\_train\_mlp, y\_val\_mlp = tts(

X\_train\_use, y\_train, test\_size=0.1, random\_state=17, stratify=y\_train

)

# Train epoch by epoch

model.set\_params(warm\_start=True, max\_iter=1)

best\_val\_score = -np.inf

no\_improvement\_count = 0

for epoch in range(1, 201):

model.fit(X\_train\_mlp, y\_train\_mlp)

# Calculate metrics

train\_pred = model.predict(X\_train\_mlp)

train\_proba = model.predict\_proba(X\_train\_mlp)[:, 1]

val\_pred = model.predict(X\_val\_mlp)

val\_proba = model.predict\_proba(X\_val\_mlp)[:, 1]

train\_loss = log\_loss(y\_train\_mlp, train\_proba)

val\_loss = log\_loss(y\_val\_mlp, val\_proba)

train\_f1 = f1\_score(y\_train\_mlp, train\_pred, zero\_division=0)

train\_recall = recall\_score(y\_train\_mlp, train\_pred, zero\_division=0)

val\_f1 = f1\_score(y\_val\_mlp, val\_pred, zero\_division=0)

val\_recall = recall\_score(y\_val\_mlp, val\_pred, zero\_division=0)

# Print epoch results

print(f" Epoch {epoch} - loss: {train\_loss:.4f} - f1\_score: {train\_f1:.4f} - "

f"recall: {train\_recall:.4f} - val\_loss: {val\_loss:.4f} - "

f"val\_f1\_score: {val\_f1:.4f} - val\_recall: {val\_recall:.4f} - "

f"learning\_rate: {model.learning\_rate\_init:.4f}")

# Store epoch data

mlp\_epoch\_details.append({

'Epoch': epoch,

'Learning\_Rate': model.learning\_rate\_init,

'Loss': train\_loss,

'F1\_Score': train\_f1,

'Recall': train\_recall,

'Val\_Loss': val\_loss,

'Val\_F1\_Score': val\_f1,

'Val\_Recall': val\_recall

})

# Early stopping check

val\_score = val\_f1

if val\_score > best\_val\_score + 0.0001:

best\_val\_score = val\_score

no\_improvement\_count = 0

else:

no\_improvement\_count += 1

if no\_improvement\_count >= 5:

print(f" Validation score did not improve for 5 consecutive epochs. Stopping.")

break

# Store final iteration count

baseline\_training\_history[name] = {

'n\_iter': epoch,

'epoch\_details': mlp\_epoch\_details.copy()

}

else:

# For Random Forest and KNN

model.fit(X\_train\_use, y\_train)

training\_time = time.time() - start\_time

# Make predictions

y\_pred = model.predict(X\_test\_use)

y\_proba = model.predict\_proba(X\_test\_use)[:, 1]

# Calculate loss

loss = log\_loss(y\_test, y\_proba)

# Store results

baseline\_predictions[name] = y\_pred

baseline\_probabilities[name] = y\_proba

# Calculate performance metrics

metrics = {

'Model': name,

'Accuracy': accuracy\_score(y\_test, y\_pred),

'Precision': precision\_score(y\_test, y\_pred, zero\_division=0),

'Recall': recall\_score(y\_test, y\_pred, zero\_division=0),

'F1-Score': f1\_score(y\_test, y\_pred, zero\_division=0),

'ROC-AUC': roc\_auc\_score(y\_test, y\_proba),

'Loss': loss

}

baseline\_results[name] = metrics

# Print results for RF and KNN

if name in ['Random Forest', 'KNN']:

print(f" Result - 0s 0ms/step - "

f"loss: {loss:.4f} - f1\_score: {metrics['F1-Score']:.4f} - recall: {metrics['Recall']:.4f} - "

f"val\_loss: {loss:.4f} - val\_f1\_score: {metrics['F1-Score']:.4f} - val\_recall: {metrics['Recall']:.4f} - "

f"learning\_rate: 0.0000e+00")

output:

================================================================================

TRAINING BASELINE MODELS (Without Oversampling)

================================================================================

Training Random Forest...

Using: Unscaled data (DataFrame)

Result - 0s 0ms/step - loss: 0.0777 - f1\_score: 0.7143 - recall: 0.6061 - val\_loss: 0.0777 - val\_f1\_score: 0.7143 - val\_recall: 0.6061 - learning\_rate: 0.0000e+00

Training MLP Classifier...

Using: Scaled data (NumPy array)

Epoch 1 - loss: 0.1597 - f1\_score: 0.0000 - recall: 0.0000 - val\_loss: 0.1672 - val\_f1\_score: 0.0000 - val\_recall: 0.0000 - learning\_rate: 0.0010

Epoch 2 - loss: 0.1186 - f1\_score: 0.0000 - recall: 0.0000 - val\_loss: 0.1179 - val\_f1\_score: 0.0000 - val\_recall: 0.0000 - learning\_rate: 0.0010

Epoch 3 - loss: 0.0963 - f1\_score: 0.0726 - recall: 0.0378 - val\_loss: 0.0859 - val\_f1\_score: 0.2667 - val\_recall: 0.1538 - learning\_rate: 0.0010

Epoch 4 - loss: 0.0886 - f1\_score: 0.4098 - recall: 0.2815 - val\_loss: 0.0762 - val\_f1\_score: 0.5405 - val\_recall: 0.3846 - learning\_rate: 0.0010

Epoch 5 - loss: 0.0843 - f1\_score: 0.4689 - recall: 0.3487 - val\_loss: 0.0723 - val\_f1\_score: 0.6000 - val\_recall: 0.4615 - learning\_rate: 0.0010

Epoch 6 - loss: 0.0807 - f1\_score: 0.4780 - recall: 0.3655 - val\_loss: 0.0690 - val\_f1\_score: 0.5854 - val\_recall: 0.4615 - learning\_rate: 0.0010

Epoch 7 - loss: 0.0776 - f1\_score: 0.5213 - recall: 0.4118 - val\_loss: 0.0662 - val\_f1\_score: 0.6047 - val\_recall: 0.5000 - learning\_rate: 0.0010

Epoch 8 - loss: 0.0746 - f1\_score: 0.5185 - recall: 0.4118 - val\_loss: 0.0641 - val\_f1\_score: 0.6667 - val\_recall: 0.5769 - learning\_rate: 0.0010

Epoch 9 - loss: 0.0714 - f1\_score: 0.5445 - recall: 0.4370 - val\_loss: 0.0609 - val\_f1\_score: 0.6667 - val\_recall: 0.5769 - learning\_rate: 0.0010

Epoch 10 - loss: 0.0685 - f1\_score: 0.5617 - recall: 0.4496 - val\_loss: 0.0579 - val\_f1\_score: 0.6957 - val\_recall: 0.6154 - learning\_rate: 0.0010

Epoch 11 - loss: 0.0658 - f1\_score: 0.5864 - recall: 0.4706 - val\_loss: 0.0557 - val\_f1\_score: 0.6957 - val\_recall: 0.6154 - learning\_rate: 0.0010

Epoch 12 - loss: 0.0633 - f1\_score: 0.6158 - recall: 0.5084 - val\_loss: 0.0534 - val\_f1\_score: 0.7234 - val\_recall: 0.6538 - learning\_rate: 0.0010

Epoch 13 - loss: 0.0610 - f1\_score: 0.6466 - recall: 0.5420 - val\_loss: 0.0514 - val\_f1\_score: 0.7234 - val\_recall: 0.6538 - learning\_rate: 0.0010

Epoch 14 - loss: 0.0589 - f1\_score: 0.6650 - recall: 0.5630 - val\_loss: 0.0501 - val\_f1\_score: 0.7347 - val\_recall: 0.6923 - learning\_rate: 0.0010

Epoch 15 - loss: 0.0568 - f1\_score: 0.6832 - recall: 0.5798 - val\_loss: 0.0476 - val\_f1\_score: 0.7500 - val\_recall: 0.6923 - learning\_rate: 0.0010

Epoch 16 - loss: 0.0550 - f1\_score: 0.6880 - recall: 0.5882 - val\_loss: 0.0457 - val\_f1\_score: 0.7500 - val\_recall: 0.6923 - learning\_rate: 0.0010

Epoch 17 - loss: 0.0536 - f1\_score: 0.6923 - recall: 0.6050 - val\_loss: 0.0444 - val\_f1\_score: 0.7600 - val\_recall: 0.7308 - learning\_rate: 0.0010

Epoch 18 - loss: 0.0521 - f1\_score: 0.7062 - recall: 0.6261 - val\_loss: 0.0426 - val\_f1\_score: 0.7843 - val\_recall: 0.7692 - learning\_rate: 0.0010

Epoch 19 - loss: 0.0508 - f1\_score: 0.7294 - recall: 0.6681 - val\_loss: 0.0416 - val\_f1\_score: 0.7843 - val\_recall: 0.7692 - learning\_rate: 0.0010

Epoch 20 - loss: 0.0494 - f1\_score: 0.7471 - recall: 0.6765 - val\_loss: 0.0402 - val\_f1\_score: 0.8077 - val\_recall: 0.8077 - learning\_rate: 0.0010

Epoch 21 - loss: 0.0488 - f1\_score: 0.7376 - recall: 0.6849 - val\_loss: 0.0401 - val\_f1\_score: 0.8148 - val\_recall: 0.8462 - learning\_rate: 0.0010

Epoch 22 - loss: 0.0473 - f1\_score: 0.7488 - recall: 0.6765 - val\_loss: 0.0383 - val\_f1\_score: 0.8462 - val\_recall: 0.8462 - learning\_rate: 0.0010

Epoch 23 - loss: 0.0470 - f1\_score: 0.7636 - recall: 0.7059 - val\_loss: 0.0390 - val\_f1\_score: 0.8364 - val\_recall: 0.8846 - learning\_rate: 0.0010

Epoch 24 - loss: 0.0463 - f1\_score: 0.7613 - recall: 0.7101 - val\_loss: 0.0387 - val\_f1\_score: 0.8519 - val\_recall: 0.8846 - learning\_rate: 0.0010

Epoch 25 - loss: 0.0456 - f1\_score: 0.7658 - recall: 0.7143 - val\_loss: 0.0378 - val\_f1\_score: 0.8519 - val\_recall: 0.8846 - learning\_rate: 0.0010

Epoch 26 - loss: 0.0454 - f1\_score: 0.7726 - recall: 0.7353 - val\_loss: 0.0391 - val\_f1\_score: 0.8364 - val\_recall: 0.8846 - learning\_rate: 0.0010

Epoch 27 - loss: 0.0449 - f1\_score: 0.7726 - recall: 0.7353 - val\_loss: 0.0384 - val\_f1\_score: 0.8519 - val\_recall: 0.8846 - learning\_rate: 0.0010

Epoch 28 - loss: 0.0443 - f1\_score: 0.7726 - recall: 0.7353 - val\_loss: 0.0381 - val\_f1\_score: 0.8519 - val\_recall: 0.8846 - learning\_rate: 0.0010

Epoch 29 - loss: 0.0436 - f1\_score: 0.7736 - recall: 0.7395 - val\_loss: 0.0376 - val\_f1\_score: 0.8519 - val\_recall: 0.8846 - learning\_rate: 0.0010

Validation score did not improve for 5 consecutive epochs. Stopping.

Training KNN...

Using: Scaled data (NumPy array)

Result - 0s 0ms/step - loss: 0.3240 - f1\_score: 0.4444 - recall: 0.3030 - val\_loss: 0.3240 - val\_f1\_score: 0.4444 - val\_recall: 0.3030 - learning\_rate: 0.0000e+00

# SECTION 2: COMPREHENSIVE BASELINE ANALYSIS

# ==========================================

# 2.1 Performance Summary Table

print("\n" + "="\*80)

print("BASELINE MODEL PERFORMANCE SUMMARY")

print("="\*80)

# Create DataFrame for baseline results

baseline\_df = pd.DataFrame(baseline\_results).T.round(4)

# Style the table

styled\_baseline\_table = (

baseline\_df.style

.hide(axis="index")

.set\_properties(\*\*{'text-align': 'center'})

.set\_table\_styles([

{'selector': 'th, td', 'props': 'border: 1px solid black;'},

{'selector': 'th', 'props': 'background-color: lightgray;'}

])

.format({

'Accuracy': '{:.4f}',

'Precision': '{:.4f}',

'Recall': '{:.4f}',

'F1-Score': '{:.4f}',

'ROC-AUC': '{:.4f}',

'Loss': '{:.4f}'

})

.set\_caption("Baseline Model Performance Metrics")

)

display(styled\_baseline\_table)

github link of the output:

<https://github.com/JencyFrancis/26th-aug/commit/fb20e4769706d4c9fab86fcf63efc5910044709c>

# 2.2 Epochwise Performance Matrix - MLP Classifier (ACTUAL DATA)

if mlp\_epoch\_details:

print("\n" + "="\*80)

print("EPOCHWISE PERFORMANCE MATRIX - MLP CLASSIFIER")

print("="\*80)

# Select key epochs to display

total\_epochs = len(mlp\_epoch\_details)

if total\_epochs > 10:

# Show first 5, some middle, and last 5 epochs

indices = list(range(5)) + list(range(total\_epochs//2 - 2, total\_epochs//2 + 3)) + list(range(total\_epochs - 5, total\_epochs))

indices = sorted(set([i for i in indices if 0 <= i < total\_epochs]))

else:

indices = range(total\_epochs)

epoch\_display\_data = [mlp\_epoch\_details[i] for i in indices]

epoch\_df = pd.DataFrame(epoch\_display\_data)

styled\_epoch\_table = (

epoch\_df.style

.hide(axis="index")

.set\_properties(\*\*{'text-align': 'center'})

.set\_table\_styles([

{'selector': 'th, td', 'props': 'border: 1px solid black;'},

{'selector': 'th', 'props': 'background-color: lightgray;'}

])

.format({

'Epoch': '{:d}',

'Learning\_Rate': '{:.4f}',

'Loss': '{:.4f}',

'F1\_Score': '{:.4f}',

'Recall': '{:.4f}',

'Val\_Loss': '{:.4f}',

'Val\_F1\_Score': '{:.4f}',

'Val\_Recall': '{:.4f}'

})

.set\_caption("MLP Classifier Training Progress (Selected Epochs)")

)

display(styled\_epoch\_table)

hithub link of the output:

<https://github.com/JencyFrancis/26th-aug/commit/02174901985fe5fc6fc23b4bd40813be127bb165>

# 2.4 ROC Curves for Baseline

plt.figure(figsize=(10, 8))

plt.title('Baseline Model ROC Curves', fontsize=16, fontweight='bold')

colors = ['blue', 'red', 'green']

for idx, (name, y\_proba) in enumerate(baseline\_probabilities.items()):

fpr, tpr, \_ = roc\_curve(y\_test, y\_proba)

roc\_auc = auc(fpr, tpr)

plt.plot(fpr, tpr, color=colors[idx], lw=2,

label=f'{name} (AUC = {roc\_auc:.4f})')

plt.plot([0, 1], [0, 1], 'k--', lw=2, label='Random Classifier (AUC = 0.5000)')

plt.xlabel('False Positive Rate', fontsize=12)

plt.ylabel('True Positive Rate', fontsize=12)

plt.legend(loc='lower right')

plt.grid(True, alpha=0.3)

plt.show()

github link of the output:

<https://github.com/JencyFrancis/26th-aug/commit/83f4602c6db5b91a5e290b39e62b63072d4be101>

# 2.5 Training/Validation Plot for MLP Classifier (ACTUAL DATA)

if mlp\_epoch\_details:

fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2, 2, figsize=(15, 12))

fig.suptitle('Baseline MLP Classifier Training Progress', fontsize=16, fontweight='bold')

# Extract data from epoch details

epochs = [d['Epoch'] for d in mlp\_epoch\_details]

train\_loss = [d['Loss'] for d in mlp\_epoch\_details]

val\_loss = [d['Val\_Loss'] for d in mlp\_epoch\_details]

train\_f1 = [d['F1\_Score'] for d in mlp\_epoch\_details]

val\_f1 = [d['Val\_F1\_Score'] for d in mlp\_epoch\_details]

train\_recall = [d['Recall'] for d in mlp\_epoch\_details]

val\_recall = [d['Val\_Recall'] for d in mlp\_epoch\_details]

# Calculate ROC-AUC progression (approximate from F1 and Recall)

train\_roc = [0.5 + 0.5 \* f1 for f1 in train\_f1] # Approximation

val\_roc = [0.5 + 0.5 \* f1 for f1 in val\_f1] # Approximation

# Plot Loss

ax1.plot(epochs, train\_loss, 'b-', label='Training Loss', linewidth=2)

ax1.plot(epochs, val\_loss, 'r-', label='Validation Loss', linewidth=2)

ax1.set\_title('Loss', fontsize=14, fontweight='bold')

ax1.set\_xlabel('Epoch')

ax1.set\_ylabel('Loss')

ax1.legend()

ax1.grid(True, alpha=0.3)

# Plot F1-Score

ax2.plot(epochs, train\_f1, 'b-', label='Training F1', linewidth=2)

ax2.plot(epochs, val\_f1, 'r-', label='Validation F1', linewidth=2)

ax2.set\_title('F1-Score', fontsize=14, fontweight='bold')

ax2.set\_xlabel('Epoch')

ax2.set\_ylabel('F1-Score')

ax2.set\_ylim(0, 1)

ax2.legend()

ax2.grid(True, alpha=0.3)

# Plot Recall

ax3.plot(epochs, train\_recall, 'b-', label='Training Recall', linewidth=2)

ax3.plot(epochs, val\_recall, 'r-', label='Validation Recall', linewidth=2)

ax3.set\_title('Recall', fontsize=14, fontweight='bold')

ax3.set\_xlabel('Epoch')

ax3.set\_ylabel('Recall')

ax3.set\_ylim(0, 1)

ax3.legend()

ax3.grid(True, alpha=0.3)

# Plot ROC-AUC

ax4.plot(epochs, train\_roc, 'b-', label='Training ROC-AUC', linewidth=2)

ax4.plot(epochs, val\_roc, 'r-', label='Validation ROC-AUC', linewidth=2)

ax4.set\_title('ROC-AUC', fontsize=14, fontweight='bold')

ax4.set\_xlabel('Epoch')

ax4.set\_ylabel('ROC-AUC')

ax4.set\_ylim(0, 1)

ax4.legend()

ax4.grid(True, alpha=0.3)

plt.tight\_layout()

plt.show()

github link of the output:

<https://github.com/JencyFrancis/26th-aug/commit/7c9d04908f358935ef16eaac1890e3cbf15de02c>

# SECTION 3: OVERSAMPLING WITH CONSISTENT ANALYSIS

# ================================================

# Apply oversampling techniques

print("\n" + "="\*80)

print("APPLYING OVERSAMPLING TECHNIQUES")

print("="\*80)

# Initialize oversampling techniques (including SMOTETomek)

techniques = {

'SMOTE': SMOTE(random\_state=17),

'ADASYN': ADASYN(random\_state=17),

'RandomOverSampler': RandomOverSampler(random\_state=17),

'SMOTETomek': SMOTETomek(random\_state=17)

}

# Apply each technique and store results

resampled\_datasets = {}

oversampling\_summary = []

# Store original class distribution

original\_counts = y\_train.value\_counts().sort\_index()

original\_total = len(y\_train)

original\_failure\_count = original\_counts[1]

original\_failure\_rate = original\_failure\_count / original\_total

# Add original to summary

oversampling\_summary.append({

'Technique': 'Original',

'Original\_Samples': original\_total,

'Resampled\_Samples': original\_total,

'Failure\_Count': original\_failure\_count,

'Failure\_Rate': original\_failure\_rate

})

for name, technique in techniques.items():

# 1. Resample SCALED data (for MLP and KNN)

X\_resampled\_scaled, y\_resampled\_scaled = technique.fit\_resample(X\_train\_scaled, y\_train)

# 2. Resample UNSCALED data (for Random Forest)

if name == 'SMOTE':

technique\_unscaled = SMOTE(random\_state=17)

elif name == 'ADASYN':

technique\_unscaled = ADASYN(random\_state=17)

elif name == 'RandomOverSampler':

technique\_unscaled = RandomOverSampler(random\_state=17)

else:

technique\_unscaled = SMOTETomek(random\_state=17)

X\_resampled\_unscaled, y\_resampled\_unscaled = technique\_unscaled.fit\_resample(X\_train\_unscaled, y\_train)

# Store both versions

resampled\_datasets[name] = {

'scaled': (X\_resampled\_scaled, y\_resampled\_scaled),

'unscaled': (X\_resampled\_unscaled, y\_resampled\_unscaled)

}

# Print class distribution

counts = pd.Series(y\_resampled\_scaled).value\_counts().sort\_index()

print(f"{name} - No Failure: {counts[0]}, Failure: {counts[1]}")

# Add to summary

resampled\_total = len(y\_resampled\_scaled)

failure\_count = counts[1]

failure\_rate = failure\_count / resampled\_total

oversampling\_summary.append({

'Technique': name,

'Original\_Samples': original\_total,

'Resampled\_Samples': resampled\_total,

'Failure\_Count': failure\_count,

'Failure\_Rate': failure\_rate

})

# Display oversampling summary table

print("\n" + "="\*80)

print("OVERSAMPLING SUMMARY")

print("="\*80)

summary\_df = pd.DataFrame(oversampling\_summary)

styled\_summary\_table = (

summary\_df.style

.hide(axis="index")

.set\_properties(\*\*{'text-align': 'center'})

.set\_table\_styles([

{'selector': 'th, td', 'props': 'border: 1px solid black;'},

{'selector': 'th', 'props': 'background-color: lightgray;'}

])

.format({

'Original\_Samples': '{:,}',

'Resampled\_Samples': '{:,}',

'Failure\_Count': '{:,}',

'Failure\_Rate': '{:.4f}'

})

.set\_caption("Oversampling Techniques Summary")

)

display(styled\_summary\_table)

github link of the output:

<https://github.com/JencyFrancis/26th-aug/commit/18fbb9cb40cb45bdb89aef474ca9a28741c786b5>

# Visualization before and after oversampling

fig, ax = plt.subplots(figsize=(14, 8))

# Prepare data for grouped bar chart

techniques\_list = ['Original'] + list(techniques.keys())

no\_failure\_counts = []

failure\_counts = []

# Original data

no\_failure\_counts.append(original\_counts[0])

failure\_counts.append(original\_counts[1])

# Oversampled data

for technique in techniques.keys():

counts = pd.Series(resampled\_datasets[technique]['scaled'][1]).value\_counts().sort\_index()

no\_failure\_counts.append(counts[0])

failure\_counts.append(counts[1])

x = np.arange(len(techniques\_list))

width = 0.35

# Create bars

bars1 = ax.bar(x - width/2, no\_failure\_counts, width, label='No Failure', color='skyblue')

bars2 = ax.bar(x + width/2, failure\_counts, width, label='Failure', color='lightcoral')

ax.set\_xlabel('Technique', fontsize=12, fontweight='bold')

ax.set\_ylabel('Sample Count', fontsize=12, fontweight='bold')

ax.set\_title('Class Distribution: Before and After Oversampling', fontsize=14, fontweight='bold')

ax.set\_xticks(x)

ax.set\_xticklabels(techniques\_list, rotation=45, ha='right')

ax.legend()

ax.grid(True, alpha=0.3, axis='y')

plt.tight\_layout()

plt.show()

github link of the output:

<https://github.com/JencyFrancis/26th-aug/commit/94ebbfee4bae547552d57a524f8327d7843e78b3>

# Train models with oversampled data

oversampled\_results = {}

oversampled\_predictions = {}

oversampled\_probabilities = {}

oversampled\_training\_history = {}

for technique\_name, data\_dict in resampled\_datasets.items():

print(f"\n{'-'\*50}")

print(f"TRAINING WITH {technique\_name.upper()}")

print(f"{'-'\*50}")

for model\_name, model in models.items():

# Clone model to ensure fresh start

model\_clone = clone(model)

combination\_key = f"{model\_name}\_{technique\_name}"

print(f"\nTraining {model\_name} with {technique\_name}:")

# Select appropriate data based on model type

if model\_name == 'Random Forest':

X\_train\_use = data\_dict['unscaled'][0]

y\_train\_use = data\_dict['unscaled'][1]

X\_test\_use = X\_test\_unscaled

print(" Using: Unscaled data (DataFrame)")

else:

X\_train\_use = data\_dict['scaled'][0]

y\_train\_use = data\_dict['scaled'][1]

X\_test\_use = X\_test\_scaled

print(" Using: Scaled data (NumPy array)")

# Train model

start\_time = time.time()

if model\_name == 'MLP Classifier':

# Custom training loop for MLP

from sklearn.model\_selection import train\_test\_split as tts

# Split for validation

X\_train\_mlp, X\_val\_mlp, y\_train\_mlp, y\_val\_mlp = tts(

X\_train\_use, y\_train\_use, test\_size=0.1, random\_state=17, stratify=y\_train\_use

)

# Train epoch by epoch

model\_clone.set\_params(warm\_start=True, max\_iter=1)

best\_val\_score = -np.inf

no\_improvement\_count = 0

epoch\_data = []

for epoch in range(1, 201):

model\_clone.fit(X\_train\_mlp, y\_train\_mlp)

# Calculate metrics

train\_pred = model\_clone.predict(X\_train\_mlp)

train\_proba = model\_clone.predict\_proba(X\_train\_mlp)[:, 1]

val\_pred = model\_clone.predict(X\_val\_mlp)

val\_proba = model\_clone.predict\_proba(X\_val\_mlp)[:, 1]

train\_loss = log\_loss(y\_train\_mlp, train\_proba)

val\_loss = log\_loss(y\_val\_mlp, val\_proba)

train\_f1 = f1\_score(y\_train\_mlp, train\_pred, zero\_division=0)

train\_recall = recall\_score(y\_train\_mlp, train\_pred, zero\_division=0)

val\_f1 = f1\_score(y\_val\_mlp, val\_pred, zero\_division=0)

val\_recall = recall\_score(y\_val\_mlp, val\_pred, zero\_division=0)

# Print epoch results

print(f" Epoch {epoch} - loss: {train\_loss:.4f} - f1\_score: {train\_f1:.4f} - "

f"recall: {train\_recall:.4f} - val\_loss: {val\_loss:.4f} - "

f"val\_f1\_score: {val\_f1:.4f} - val\_recall: {val\_recall:.4f} - "

f"learning\_rate: {model\_clone.learning\_rate\_init:.4f}")

# Store epoch data

epoch\_data.append({

'epoch': epoch,

'train\_loss': train\_loss,

'val\_loss': val\_loss,

'train\_f1': train\_f1,

'val\_f1': val\_f1,

'train\_recall': train\_recall,

'val\_recall': val\_recall

})

# Early stopping check

val\_score = val\_f1

if val\_score > best\_val\_score + 0.0001:

best\_val\_score = val\_score

no\_improvement\_count = 0

else:

no\_improvement\_count += 1

if no\_improvement\_count >= 5:

print(f" Validation score did not improve for 5 consecutive epochs. Stopping.")

break

# Store training history

oversampled\_training\_history[combination\_key] = {

'n\_iter': epoch,

'epoch\_data': epoch\_data

}

else:

# For Random Forest and KNN

model\_clone.fit(X\_train\_use, y\_train\_use)

training\_time = time.time() - start\_time

# Make predictions

y\_pred = model\_clone.predict(X\_test\_use)

y\_proba = model\_clone.predict\_proba(X\_test\_use)[:, 1]

# Calculate metrics

loss = log\_loss(y\_test, y\_proba)

# Store predictions and probabilities

oversampled\_predictions[combination\_key] = y\_pred

oversampled\_probabilities[combination\_key] = y\_proba

# Calculate all metrics

metrics = {

'Model': model\_name,

'Oversampling': technique\_name,

'Accuracy': accuracy\_score(y\_test, y\_pred),

'Precision': precision\_score(y\_test, y\_pred, zero\_division=0),

'Recall': recall\_score(y\_test, y\_pred, zero\_division=0),

'F1-Score': f1\_score(y\_test, y\_pred, zero\_division=0),

'ROC-AUC': roc\_auc\_score(y\_test, y\_proba),

'Loss': loss

}

oversampled\_results[combination\_key] = metrics

# Print results for RF and KNN

if model\_name in ['Random Forest', 'KNN']:

print(f" Result - 0s 0ms/step - "

f"loss: {loss:.4f} - f1\_score: {metrics['F1-Score']:.4f} - recall: {metrics['Recall']:.4f} - "

f"val\_loss: {loss:.4f} - val\_f1\_score: {metrics['F1-Score']:.4f} - val\_recall: {metrics['Recall']:.4f} - "

f"learning\_rate: 0.0000e+00")

# Display comprehensive results

print("\n" + "="\*80)

print("OVERSAMPLED MODEL PERFORMANCE SUMMARY")

print("="\*80)

# Create results DataFrame

oversampled\_results\_df = pd.DataFrame.from\_dict(oversampled\_results, orient='index')

# Style the results table

styled\_results\_table = (

oversampled\_results\_df.style

.hide(axis="index")

.set\_properties(\*\*{'text-align': 'center'})

.set\_table\_styles([

{'selector': 'th, td', 'props': 'border: 1px solid black;'},

{'selector': 'th', 'props': 'background-color: lightgray;'}

])

.format({

'Accuracy': '{:.4f}',

'Precision': '{:.4f}',

'Recall': '{:.4f}',

'F1-Score': '{:.4f}',

'ROC-AUC': '{:.4f}',

'Loss': '{:.4f}'

})

.set\_caption("Oversampled Models Performance Matrix")

)

display(styled\_results\_table)

output:

--------------------------------------------------

TRAINING WITH SMOTE

--------------------------------------------------

Training Random Forest with SMOTE:

Using: Unscaled data (DataFrame)

Result - 0s 0ms/step - loss: 0.0796 - f1\_score: 0.5934 - recall: 0.8182 - val\_loss: 0.0796 - val\_f1\_score: 0.5934 - val\_recall: 0.8182 - learning\_rate: 0.0000e+00

Training MLP Classifier with SMOTE:

Using: Scaled data (NumPy array)

Epoch 1 - loss: 0.2741 - f1\_score: 0.8775 - recall: 0.8652 - val\_loss: 0.2685 - val\_f1\_score: 0.8831 - val\_recall: 0.8756 - learning\_rate: 0.0010

Epoch 2 - loss: 0.1894 - f1\_score: 0.9264 - recall: 0.9453 - val\_loss: 0.1955 - val\_f1\_score: 0.9185 - val\_recall: 0.9339 - learning\_rate: 0.0010

Epoch 3 - loss: 0.1536 - f1\_score: 0.9440 - recall: 0.9644 - val\_loss: 0.1639 - val\_f1\_score: 0.9344 - val\_recall: 0.9495 - learning\_rate: 0.0010

Epoch 4 - loss: 0.1310 - f1\_score: 0.9568 - recall: 0.9785 - val\_loss: 0.1457 - val\_f1\_score: 0.9544 - val\_recall: 0.9767 - learning\_rate: 0.0010

Epoch 5 - loss: 0.1157 - f1\_score: 0.9618 - recall: 0.9840 - val\_loss: 0.1341 - val\_f1\_score: 0.9578 - val\_recall: 0.9845 - learning\_rate: 0.0010

Epoch 6 - loss: 0.1047 - f1\_score: 0.9661 - recall: 0.9879 - val\_loss: 0.1262 - val\_f1\_score: 0.9578 - val\_recall: 0.9845 - learning\_rate: 0.0010

Epoch 7 - loss: 0.0966 - f1\_score: 0.9688 - recall: 0.9902 - val\_loss: 0.1203 - val\_f1\_score: 0.9603 - val\_recall: 0.9858 - learning\_rate: 0.0010

Epoch 8 - loss: 0.0902 - f1\_score: 0.9708 - recall: 0.9912 - val\_loss: 0.1154 - val\_f1\_score: 0.9634 - val\_recall: 0.9883 - learning\_rate: 0.0010

Epoch 9 - loss: 0.0848 - f1\_score: 0.9726 - recall: 0.9924 - val\_loss: 0.1112 - val\_f1\_score: 0.9639 - val\_recall: 0.9870 - learning\_rate: 0.0010

Epoch 10 - loss: 0.0800 - f1\_score: 0.9739 - recall: 0.9924 - val\_loss: 0.1077 - val\_f1\_score: 0.9639 - val\_recall: 0.9870 - learning\_rate: 0.0010

Epoch 11 - loss: 0.0760 - f1\_score: 0.9748 - recall: 0.9929 - val\_loss: 0.1045 - val\_f1\_score: 0.9658 - val\_recall: 0.9870 - learning\_rate: 0.0010

Epoch 12 - loss: 0.0725 - f1\_score: 0.9762 - recall: 0.9934 - val\_loss: 0.1017 - val\_f1\_score: 0.9670 - val\_recall: 0.9870 - learning\_rate: 0.0010

Epoch 13 - loss: 0.0694 - f1\_score: 0.9775 - recall: 0.9935 - val\_loss: 0.0992 - val\_f1\_score: 0.9676 - val\_recall: 0.9870 - learning\_rate: 0.0010

Epoch 14 - loss: 0.0671 - f1\_score: 0.9794 - recall: 0.9945 - val\_loss: 0.0974 - val\_f1\_score: 0.9682 - val\_recall: 0.9870 - learning\_rate: 0.0010

Epoch 15 - loss: 0.0645 - f1\_score: 0.9810 - recall: 0.9948 - val\_loss: 0.0950 - val\_f1\_score: 0.9682 - val\_recall: 0.9858 - learning\_rate: 0.0010

Epoch 16 - loss: 0.0625 - f1\_score: 0.9818 - recall: 0.9950 - val\_loss: 0.0932 - val\_f1\_score: 0.9701 - val\_recall: 0.9870 - learning\_rate: 0.0010

Epoch 17 - loss: 0.0605 - f1\_score: 0.9819 - recall: 0.9944 - val\_loss: 0.0920 - val\_f1\_score: 0.9701 - val\_recall: 0.9870 - learning\_rate: 0.0010

Epoch 18 - loss: 0.0589 - f1\_score: 0.9825 - recall: 0.9947 - val\_loss: 0.0906 - val\_f1\_score: 0.9714 - val\_recall: 0.9883 - learning\_rate: 0.0010

Epoch 19 - loss: 0.0572 - f1\_score: 0.9832 - recall: 0.9952 - val\_loss: 0.0895 - val\_f1\_score: 0.9714 - val\_recall: 0.9883 - learning\_rate: 0.0010

Epoch 20 - loss: 0.0557 - f1\_score: 0.9835 - recall: 0.9952 - val\_loss: 0.0886 - val\_f1\_score: 0.9726 - val\_recall: 0.9883 - learning\_rate: 0.0010

Epoch 21 - loss: 0.0541 - f1\_score: 0.9838 - recall: 0.9951 - val\_loss: 0.0875 - val\_f1\_score: 0.9726 - val\_recall: 0.9883 - learning\_rate: 0.0010

Epoch 22 - loss: 0.0530 - f1\_score: 0.9838 - recall: 0.9955 - val\_loss: 0.0867 - val\_f1\_score: 0.9739 - val\_recall: 0.9896 - learning\_rate: 0.0010

Epoch 23 - loss: 0.0513 - f1\_score: 0.9844 - recall: 0.9952 - val\_loss: 0.0850 - val\_f1\_score: 0.9732 - val\_recall: 0.9883 - learning\_rate: 0.0010

Epoch 24 - loss: 0.0503 - f1\_score: 0.9845 - recall: 0.9957 - val\_loss: 0.0852 - val\_f1\_score: 0.9745 - val\_recall: 0.9909 - learning\_rate: 0.0010

Epoch 25 - loss: 0.0489 - f1\_score: 0.9849 - recall: 0.9955 - val\_loss: 0.0838 - val\_f1\_score: 0.9745 - val\_recall: 0.9909 - learning\_rate: 0.0010

Epoch 26 - loss: 0.0479 - f1\_score: 0.9850 - recall: 0.9955 - val\_loss: 0.0835 - val\_f1\_score: 0.9745 - val\_recall: 0.9896 - learning\_rate: 0.0010

Epoch 27 - loss: 0.0468 - f1\_score: 0.9852 - recall: 0.9958 - val\_loss: 0.0829 - val\_f1\_score: 0.9751 - val\_recall: 0.9896 - learning\_rate: 0.0010

Epoch 28 - loss: 0.0459 - f1\_score: 0.9855 - recall: 0.9952 - val\_loss: 0.0828 - val\_f1\_score: 0.9745 - val\_recall: 0.9883 - learning\_rate: 0.0010

Epoch 29 - loss: 0.0449 - f1\_score: 0.9856 - recall: 0.9948 - val\_loss: 0.0822 - val\_f1\_score: 0.9725 - val\_recall: 0.9845 - learning\_rate: 0.0010

Epoch 30 - loss: 0.0440 - f1\_score: 0.9859 - recall: 0.9942 - val\_loss: 0.0808 - val\_f1\_score: 0.9731 - val\_recall: 0.9845 - learning\_rate: 0.0010

Epoch 31 - loss: 0.0434 - f1\_score: 0.9864 - recall: 0.9939 - val\_loss: 0.0807 - val\_f1\_score: 0.9731 - val\_recall: 0.9832 - learning\_rate: 0.0010

Epoch 32 - loss: 0.0426 - f1\_score: 0.9869 - recall: 0.9945 - val\_loss: 0.0807 - val\_f1\_score: 0.9731 - val\_recall: 0.9832 - learning\_rate: 0.0010

Validation score did not improve for 5 consecutive epochs. Stopping.

Training KNN with SMOTE:

Using: Scaled data (NumPy array)

Result - 0s 0ms/step - loss: 0.9947 - f1\_score: 0.4601 - recall: 0.7424 - val\_loss: 0.9947 - val\_f1\_score: 0.4601 - val\_recall: 0.7424 - learning\_rate: 0.0000e+00

--------------------------------------------------

TRAINING WITH ADASYN

--------------------------------------------------

Training Random Forest with ADASYN:

Using: Unscaled data (DataFrame)

Result - 0s 0ms/step - loss: 0.0867 - f1\_score: 0.5902 - recall: 0.8182 - val\_loss: 0.0867 - val\_f1\_score: 0.5902 - val\_recall: 0.8182 - learning\_rate: 0.0000e+00

Training MLP Classifier with ADASYN:

Using: Scaled data (NumPy array)

Epoch 1 - loss: 0.2838 - f1\_score: 0.8797 - recall: 0.8935 - val\_loss: 0.2858 - val\_f1\_score: 0.8878 - val\_recall: 0.9039 - learning\_rate: 0.0010

Epoch 2 - loss: 0.2111 - f1\_score: 0.9230 - recall: 0.9707 - val\_loss: 0.2132 - val\_f1\_score: 0.9230 - val\_recall: 0.9727 - learning\_rate: 0.0010

Epoch 3 - loss: 0.1727 - f1\_score: 0.9391 - recall: 0.9838 - val\_loss: 0.1766 - val\_f1\_score: 0.9392 - val\_recall: 0.9935 - learning\_rate: 0.0010

Epoch 4 - loss: 0.1481 - f1\_score: 0.9480 - recall: 0.9885 - val\_loss: 0.1558 - val\_f1\_score: 0.9433 - val\_recall: 0.9948 - learning\_rate: 0.0010

Epoch 5 - loss: 0.1308 - f1\_score: 0.9547 - recall: 0.9899 - val\_loss: 0.1419 - val\_f1\_score: 0.9498 - val\_recall: 0.9948 - learning\_rate: 0.0010

Epoch 6 - loss: 0.1186 - f1\_score: 0.9596 - recall: 0.9913 - val\_loss: 0.1321 - val\_f1\_score: 0.9528 - val\_recall: 0.9961 - learning\_rate: 0.0010

Epoch 7 - loss: 0.1104 - f1\_score: 0.9625 - recall: 0.9929 - val\_loss: 0.1261 - val\_f1\_score: 0.9582 - val\_recall: 0.9974 - learning\_rate: 0.0010

Epoch 8 - loss: 0.1041 - f1\_score: 0.9650 - recall: 0.9931 - val\_loss: 0.1214 - val\_f1\_score: 0.9588 - val\_recall: 0.9974 - learning\_rate: 0.0010

Epoch 9 - loss: 0.0985 - f1\_score: 0.9672 - recall: 0.9944 - val\_loss: 0.1174 - val\_f1\_score: 0.9606 - val\_recall: 0.9974 - learning\_rate: 0.0010

Epoch 10 - loss: 0.0955 - f1\_score: 0.9674 - recall: 0.9948 - val\_loss: 0.1149 - val\_f1\_score: 0.9613 - val\_recall: 0.9987 - learning\_rate: 0.0010

Epoch 11 - loss: 0.0909 - f1\_score: 0.9698 - recall: 0.9947 - val\_loss: 0.1106 - val\_f1\_score: 0.9618 - val\_recall: 0.9974 - learning\_rate: 0.0010

Epoch 12 - loss: 0.0882 - f1\_score: 0.9706 - recall: 0.9951 - val\_loss: 0.1085 - val\_f1\_score: 0.9630 - val\_recall: 0.9974 - learning\_rate: 0.0010

Epoch 13 - loss: 0.0851 - f1\_score: 0.9722 - recall: 0.9954 - val\_loss: 0.1061 - val\_f1\_score: 0.9637 - val\_recall: 0.9987 - learning\_rate: 0.0010

Epoch 14 - loss: 0.0828 - f1\_score: 0.9731 - recall: 0.9960 - val\_loss: 0.1039 - val\_f1\_score: 0.9667 - val\_recall: 0.9987 - learning\_rate: 0.0010

Epoch 15 - loss: 0.0807 - f1\_score: 0.9743 - recall: 0.9965 - val\_loss: 0.1023 - val\_f1\_score: 0.9673 - val\_recall: 0.9987 - learning\_rate: 0.0010

Epoch 16 - loss: 0.0791 - f1\_score: 0.9748 - recall: 0.9965 - val\_loss: 0.1015 - val\_f1\_score: 0.9679 - val\_recall: 0.9987 - learning\_rate: 0.0010

Epoch 17 - loss: 0.0772 - f1\_score: 0.9756 - recall: 0.9970 - val\_loss: 0.1001 - val\_f1\_score: 0.9685 - val\_recall: 0.9987 - learning\_rate: 0.0010

Epoch 18 - loss: 0.0747 - f1\_score: 0.9766 - recall: 0.9975 - val\_loss: 0.0979 - val\_f1\_score: 0.9703 - val\_recall: 0.9987 - learning\_rate: 0.0010

Epoch 19 - loss: 0.0742 - f1\_score: 0.9764 - recall: 0.9978 - val\_loss: 0.0984 - val\_f1\_score: 0.9710 - val\_recall: 0.9987 - learning\_rate: 0.0010

Epoch 20 - loss: 0.0721 - f1\_score: 0.9775 - recall: 0.9977 - val\_loss: 0.0960 - val\_f1\_score: 0.9710 - val\_recall: 0.9987 - learning\_rate: 0.0010

Epoch 21 - loss: 0.0696 - f1\_score: 0.9782 - recall: 0.9980 - val\_loss: 0.0939 - val\_f1\_score: 0.9710 - val\_recall: 0.9987 - learning\_rate: 0.0010

Epoch 22 - loss: 0.0686 - f1\_score: 0.9783 - recall: 0.9978 - val\_loss: 0.0939 - val\_f1\_score: 0.9716 - val\_recall: 1.0000 - learning\_rate: 0.0010

Epoch 23 - loss: 0.0666 - f1\_score: 0.9786 - recall: 0.9978 - val\_loss: 0.0916 - val\_f1\_score: 0.9710 - val\_recall: 0.9987 - learning\_rate: 0.0010

Epoch 24 - loss: 0.0664 - f1\_score: 0.9790 - recall: 0.9984 - val\_loss: 0.0926 - val\_f1\_score: 0.9722 - val\_recall: 1.0000 - learning\_rate: 0.0010

Epoch 25 - loss: 0.0646 - f1\_score: 0.9793 - recall: 0.9981 - val\_loss: 0.0913 - val\_f1\_score: 0.9728 - val\_recall: 0.9987 - learning\_rate: 0.0010

Epoch 26 - loss: 0.0634 - f1\_score: 0.9801 - recall: 0.9984 - val\_loss: 0.0908 - val\_f1\_score: 0.9728 - val\_recall: 0.9987 - learning\_rate: 0.0010

Epoch 27 - loss: 0.0617 - f1\_score: 0.9810 - recall: 0.9984 - val\_loss: 0.0888 - val\_f1\_score: 0.9734 - val\_recall: 0.9987 - learning\_rate: 0.0010

Epoch 28 - loss: 0.0604 - f1\_score: 0.9816 - recall: 0.9987 - val\_loss: 0.0884 - val\_f1\_score: 0.9741 - val\_recall: 1.0000 - learning\_rate: 0.0010

Epoch 29 - loss: 0.0587 - f1\_score: 0.9821 - recall: 0.9981 - val\_loss: 0.0864 - val\_f1\_score: 0.9734 - val\_recall: 0.9987 - learning\_rate: 0.0010

Epoch 30 - loss: 0.0571 - f1\_score: 0.9825 - recall: 0.9986 - val\_loss: 0.0854 - val\_f1\_score: 0.9741 - val\_recall: 1.0000 - learning\_rate: 0.0010

Epoch 31 - loss: 0.0558 - f1\_score: 0.9830 - recall: 0.9983 - val\_loss: 0.0839 - val\_f1\_score: 0.9741 - val\_recall: 1.0000 - learning\_rate: 0.0010

Epoch 32 - loss: 0.0555 - f1\_score: 0.9834 - recall: 0.9986 - val\_loss: 0.0844 - val\_f1\_score: 0.9735 - val\_recall: 1.0000 - learning\_rate: 0.0010

Epoch 33 - loss: 0.0541 - f1\_score: 0.9841 - recall: 0.9986 - val\_loss: 0.0828 - val\_f1\_score: 0.9741 - val\_recall: 1.0000 - learning\_rate: 0.0010

Validation score did not improve for 5 consecutive epochs. Stopping.

Training KNN with ADASYN:

Using: Scaled data (NumPy array)

Result - 0s 0ms/step - loss: 1.1198 - f1\_score: 0.4505 - recall: 0.7576 - val\_loss: 1.1198 - val\_f1\_score: 0.4505 - val\_recall: 0.7576 - learning\_rate: 0.0000e+00

--------------------------------------------------

TRAINING WITH RANDOMOVERSAMPLER

--------------------------------------------------

Training Random Forest with RandomOverSampler:

Using: Unscaled data (DataFrame)

Result - 0s 0ms/step - loss: 0.0471 - f1\_score: 0.7667 - recall: 0.6970 - val\_loss: 0.0471 - val\_f1\_score: 0.7667 - val\_recall: 0.6970 - learning\_rate: 0.0000e+00

Training MLP Classifier with RandomOverSampler:

Using: Scaled data (NumPy array)

Epoch 1 - loss: 0.2793 - f1\_score: 0.8793 - recall: 0.8748 - val\_loss: 0.2814 - val\_f1\_score: 0.8790 - val\_recall: 0.8795 - learning\_rate: 0.0010

Epoch 2 - loss: 0.2018 - f1\_score: 0.9202 - recall: 0.9381 - val\_loss: 0.2059 - val\_f1\_score: 0.9217 - val\_recall: 0.9456 - learning\_rate: 0.0010

Epoch 3 - loss: 0.1618 - f1\_score: 0.9456 - recall: 0.9659 - val\_loss: 0.1685 - val\_f1\_score: 0.9402 - val\_recall: 0.9676 - learning\_rate: 0.0010

Epoch 4 - loss: 0.1355 - f1\_score: 0.9551 - recall: 0.9731 - val\_loss: 0.1464 - val\_f1\_score: 0.9514 - val\_recall: 0.9754 - learning\_rate: 0.0010

Epoch 5 - loss: 0.1181 - f1\_score: 0.9655 - recall: 0.9865 - val\_loss: 0.1321 - val\_f1\_score: 0.9591 - val\_recall: 0.9883 - learning\_rate: 0.0010

Epoch 6 - loss: 0.1064 - f1\_score: 0.9690 - recall: 0.9905 - val\_loss: 0.1225 - val\_f1\_score: 0.9618 - val\_recall: 0.9935 - learning\_rate: 0.0010

Epoch 7 - loss: 0.0976 - f1\_score: 0.9704 - recall: 0.9905 - val\_loss: 0.1151 - val\_f1\_score: 0.9630 - val\_recall: 0.9935 - learning\_rate: 0.0010

Epoch 8 - loss: 0.0909 - f1\_score: 0.9710 - recall: 0.9905 - val\_loss: 0.1094 - val\_f1\_score: 0.9642 - val\_recall: 0.9935 - learning\_rate: 0.0010

Epoch 9 - loss: 0.0853 - f1\_score: 0.9754 - recall: 0.9967 - val\_loss: 0.1048 - val\_f1\_score: 0.9667 - val\_recall: 0.9974 - learning\_rate: 0.0010

Epoch 10 - loss: 0.0807 - f1\_score: 0.9766 - recall: 0.9967 - val\_loss: 0.1010 - val\_f1\_score: 0.9692 - val\_recall: 0.9974 - learning\_rate: 0.0010

Epoch 11 - loss: 0.0765 - f1\_score: 0.9772 - recall: 0.9967 - val\_loss: 0.0973 - val\_f1\_score: 0.9692 - val\_recall: 0.9974 - learning\_rate: 0.0010

Epoch 12 - loss: 0.0730 - f1\_score: 0.9786 - recall: 0.9967 - val\_loss: 0.0940 - val\_f1\_score: 0.9710 - val\_recall: 0.9974 - learning\_rate: 0.0010

Epoch 13 - loss: 0.0697 - f1\_score: 0.9789 - recall: 0.9967 - val\_loss: 0.0913 - val\_f1\_score: 0.9716 - val\_recall: 0.9974 - learning\_rate: 0.0010

Epoch 14 - loss: 0.0667 - f1\_score: 0.9798 - recall: 0.9967 - val\_loss: 0.0891 - val\_f1\_score: 0.9716 - val\_recall: 0.9974 - learning\_rate: 0.0010

Epoch 15 - loss: 0.0642 - f1\_score: 0.9807 - recall: 0.9967 - val\_loss: 0.0868 - val\_f1\_score: 0.9735 - val\_recall: 0.9974 - learning\_rate: 0.0010

Epoch 16 - loss: 0.0618 - f1\_score: 0.9813 - recall: 0.9967 - val\_loss: 0.0845 - val\_f1\_score: 0.9735 - val\_recall: 0.9974 - learning\_rate: 0.0010

Epoch 17 - loss: 0.0597 - f1\_score: 0.9822 - recall: 0.9967 - val\_loss: 0.0831 - val\_f1\_score: 0.9735 - val\_recall: 0.9974 - learning\_rate: 0.0010

Epoch 18 - loss: 0.0576 - f1\_score: 0.9843 - recall: 1.0000 - val\_loss: 0.0807 - val\_f1\_score: 0.9760 - val\_recall: 1.0000 - learning\_rate: 0.0010

Epoch 19 - loss: 0.0556 - f1\_score: 0.9848 - recall: 1.0000 - val\_loss: 0.0789 - val\_f1\_score: 0.9791 - val\_recall: 1.0000 - learning\_rate: 0.0010

Epoch 20 - loss: 0.0539 - f1\_score: 0.9854 - recall: 1.0000 - val\_loss: 0.0774 - val\_f1\_score: 0.9791 - val\_recall: 1.0000 - learning\_rate: 0.0010

Epoch 21 - loss: 0.0522 - f1\_score: 0.9861 - recall: 1.0000 - val\_loss: 0.0759 - val\_f1\_score: 0.9791 - val\_recall: 1.0000 - learning\_rate: 0.0010

Epoch 22 - loss: 0.0507 - f1\_score: 0.9861 - recall: 1.0000 - val\_loss: 0.0747 - val\_f1\_score: 0.9785 - val\_recall: 1.0000 - learning\_rate: 0.0010

Epoch 23 - loss: 0.0491 - f1\_score: 0.9866 - recall: 1.0000 - val\_loss: 0.0731 - val\_f1\_score: 0.9791 - val\_recall: 1.0000 - learning\_rate: 0.0010

Epoch 24 - loss: 0.0478 - f1\_score: 0.9869 - recall: 1.0000 - val\_loss: 0.0721 - val\_f1\_score: 0.9797 - val\_recall: 1.0000 - learning\_rate: 0.0010

Epoch 25 - loss: 0.0465 - f1\_score: 0.9872 - recall: 1.0000 - val\_loss: 0.0712 - val\_f1\_score: 0.9803 - val\_recall: 1.0000 - learning\_rate: 0.0010

Epoch 26 - loss: 0.0452 - f1\_score: 0.9872 - recall: 1.0000 - val\_loss: 0.0703 - val\_f1\_score: 0.9797 - val\_recall: 1.0000 - learning\_rate: 0.0010

Epoch 27 - loss: 0.0440 - f1\_score: 0.9874 - recall: 1.0000 - val\_loss: 0.0695 - val\_f1\_score: 0.9797 - val\_recall: 1.0000 - learning\_rate: 0.0010

Epoch 28 - loss: 0.0432 - f1\_score: 0.9876 - recall: 1.0000 - val\_loss: 0.0696 - val\_f1\_score: 0.9803 - val\_recall: 1.0000 - learning\_rate: 0.0010

Epoch 29 - loss: 0.0422 - f1\_score: 0.9878 - recall: 1.0000 - val\_loss: 0.0691 - val\_f1\_score: 0.9803 - val\_recall: 1.0000 - learning\_rate: 0.0010

Epoch 30 - loss: 0.0413 - f1\_score: 0.9881 - recall: 1.0000 - val\_loss: 0.0684 - val\_f1\_score: 0.9803 - val\_recall: 1.0000 - learning\_rate: 0.0010

Validation score did not improve for 5 consecutive epochs. Stopping.

Training KNN with RandomOverSampler:

Using: Scaled data (NumPy array)

Result - 0s 0ms/step - loss: 0.8094 - f1\_score: 0.5238 - recall: 0.6667 - val\_loss: 0.8094 - val\_f1\_score: 0.5238 - val\_recall: 0.6667 - learning\_rate: 0.0000e+00

--------------------------------------------------

TRAINING WITH SMOTETOMEK

--------------------------------------------------

Training Random Forest with SMOTETomek:

Using: Unscaled data (DataFrame)

Result - 0s 0ms/step - loss: 0.0842 - f1\_score: 0.5795 - recall: 0.7727 - val\_loss: 0.0842 - val\_f1\_score: 0.5795 - val\_recall: 0.7727 - learning\_rate: 0.0000e+00

Training MLP Classifier with SMOTETomek:

Using: Scaled data (NumPy array)

Epoch 1 - loss: 0.2658 - f1\_score: 0.8878 - recall: 0.9022 - val\_loss: 0.2666 - val\_f1\_score: 0.8846 - val\_recall: 0.9001 - learning\_rate: 0.0010

Epoch 2 - loss: 0.1906 - f1\_score: 0.9258 - recall: 0.9505 - val\_loss: 0.1878 - val\_f1\_score: 0.9308 - val\_recall: 0.9507 - learning\_rate: 0.0010

Epoch 3 - loss: 0.1550 - f1\_score: 0.9441 - recall: 0.9710 - val\_loss: 0.1535 - val\_f1\_score: 0.9488 - val\_recall: 0.9728 - learning\_rate: 0.0010

Epoch 4 - loss: 0.1317 - f1\_score: 0.9563 - recall: 0.9820 - val\_loss: 0.1326 - val\_f1\_score: 0.9584 - val\_recall: 0.9870 - learning\_rate: 0.0010

Epoch 5 - loss: 0.1160 - f1\_score: 0.9624 - recall: 0.9899 - val\_loss: 0.1198 - val\_f1\_score: 0.9624 - val\_recall: 0.9948 - learning\_rate: 0.0010

Epoch 6 - loss: 0.1045 - f1\_score: 0.9656 - recall: 0.9903 - val\_loss: 0.1098 - val\_f1\_score: 0.9660 - val\_recall: 0.9935 - learning\_rate: 0.0010

Epoch 7 - loss: 0.0957 - f1\_score: 0.9677 - recall: 0.9903 - val\_loss: 0.1020 - val\_f1\_score: 0.9672 - val\_recall: 0.9935 - learning\_rate: 0.0010

Epoch 8 - loss: 0.0900 - f1\_score: 0.9688 - recall: 0.9905 - val\_loss: 0.0974 - val\_f1\_score: 0.9677 - val\_recall: 0.9922 - learning\_rate: 0.0010

Epoch 9 - loss: 0.0850 - f1\_score: 0.9708 - recall: 0.9908 - val\_loss: 0.0936 - val\_f1\_score: 0.9677 - val\_recall: 0.9909 - learning\_rate: 0.0010

Epoch 10 - loss: 0.0808 - f1\_score: 0.9722 - recall: 0.9909 - val\_loss: 0.0907 - val\_f1\_score: 0.9683 - val\_recall: 0.9909 - learning\_rate: 0.0010

Epoch 11 - loss: 0.0774 - f1\_score: 0.9740 - recall: 0.9918 - val\_loss: 0.0883 - val\_f1\_score: 0.9695 - val\_recall: 0.9909 - learning\_rate: 0.0010

Epoch 12 - loss: 0.0740 - f1\_score: 0.9753 - recall: 0.9918 - val\_loss: 0.0856 - val\_f1\_score: 0.9708 - val\_recall: 0.9909 - learning\_rate: 0.0010

Epoch 13 - loss: 0.0714 - f1\_score: 0.9765 - recall: 0.9908 - val\_loss: 0.0837 - val\_f1\_score: 0.9720 - val\_recall: 0.9909 - learning\_rate: 0.0010

Epoch 14 - loss: 0.0690 - f1\_score: 0.9772 - recall: 0.9915 - val\_loss: 0.0824 - val\_f1\_score: 0.9739 - val\_recall: 0.9935 - learning\_rate: 0.0010

Epoch 15 - loss: 0.0665 - f1\_score: 0.9779 - recall: 0.9916 - val\_loss: 0.0806 - val\_f1\_score: 0.9739 - val\_recall: 0.9935 - learning\_rate: 0.0010

Epoch 16 - loss: 0.0646 - f1\_score: 0.9783 - recall: 0.9916 - val\_loss: 0.0796 - val\_f1\_score: 0.9739 - val\_recall: 0.9935 - learning\_rate: 0.0010

Epoch 17 - loss: 0.0629 - f1\_score: 0.9789 - recall: 0.9926 - val\_loss: 0.0781 - val\_f1\_score: 0.9752 - val\_recall: 0.9948 - learning\_rate: 0.0010

Epoch 18 - loss: 0.0620 - f1\_score: 0.9794 - recall: 0.9931 - val\_loss: 0.0779 - val\_f1\_score: 0.9746 - val\_recall: 0.9935 - learning\_rate: 0.0010

Epoch 19 - loss: 0.0603 - f1\_score: 0.9800 - recall: 0.9929 - val\_loss: 0.0769 - val\_f1\_score: 0.9758 - val\_recall: 0.9935 - learning\_rate: 0.0010

Epoch 20 - loss: 0.0589 - f1\_score: 0.9809 - recall: 0.9935 - val\_loss: 0.0760 - val\_f1\_score: 0.9758 - val\_recall: 0.9948 - learning\_rate: 0.0010

Epoch 21 - loss: 0.0580 - f1\_score: 0.9810 - recall: 0.9935 - val\_loss: 0.0755 - val\_f1\_score: 0.9771 - val\_recall: 0.9961 - learning\_rate: 0.0010

Epoch 22 - loss: 0.0565 - f1\_score: 0.9813 - recall: 0.9939 - val\_loss: 0.0744 - val\_f1\_score: 0.9777 - val\_recall: 0.9961 - learning\_rate: 0.0010

Epoch 23 - loss: 0.0550 - f1\_score: 0.9822 - recall: 0.9938 - val\_loss: 0.0724 - val\_f1\_score: 0.9790 - val\_recall: 0.9961 - learning\_rate: 0.0010

Epoch 24 - loss: 0.0539 - f1\_score: 0.9825 - recall: 0.9934 - val\_loss: 0.0713 - val\_f1\_score: 0.9796 - val\_recall: 0.9961 - learning\_rate: 0.0010

Epoch 25 - loss: 0.0525 - f1\_score: 0.9832 - recall: 0.9944 - val\_loss: 0.0700 - val\_f1\_score: 0.9821 - val\_recall: 0.9974 - learning\_rate: 0.0010

Epoch 26 - loss: 0.0519 - f1\_score: 0.9833 - recall: 0.9947 - val\_loss: 0.0701 - val\_f1\_score: 0.9809 - val\_recall: 0.9974 - learning\_rate: 0.0010

Epoch 27 - loss: 0.0511 - f1\_score: 0.9837 - recall: 0.9947 - val\_loss: 0.0703 - val\_f1\_score: 0.9809 - val\_recall: 0.9974 - learning\_rate: 0.0010

Epoch 28 - loss: 0.0500 - f1\_score: 0.9840 - recall: 0.9945 - val\_loss: 0.0697 - val\_f1\_score: 0.9809 - val\_recall: 0.9974 - learning\_rate: 0.0010

Epoch 29 - loss: 0.0491 - f1\_score: 0.9845 - recall: 0.9945 - val\_loss: 0.0690 - val\_f1\_score: 0.9815 - val\_recall: 0.9974 - learning\_rate: 0.0010

Epoch 30 - loss: 0.0481 - f1\_score: 0.9844 - recall: 0.9945 - val\_loss: 0.0676 - val\_f1\_score: 0.9821 - val\_recall: 0.9974 - learning\_rate: 0.0010

Validation score did not improve for 5 consecutive epochs. Stopping.

Training KNN with SMOTETomek:

Using: Scaled data (NumPy array)

Result - 0s 0ms/step - loss: 1.0117 - f1\_score: 0.4601 - recall: 0.7424 - val\_loss: 1.0117 - val\_f1\_score: 0.4601 - val\_recall: 0.7424 - learning\_rate: 0.0000e+00

Github link of the “Oversampled Models Performance Matrix”<https://github.com/JencyFrancis/26th-aug/commit/b13fc6f351517b8e55180e602649a5f134760c34>

# SECTION 4: UNIFIED PERFORMANCE COMPARISON

# =========================================

# 4.1 Combined Results DataFrame

baseline\_data = []

for model\_name, metrics in baseline\_results.items():

baseline\_data.append({

'Model': model\_name,

'Technique': 'Baseline',

'F1-Score': metrics['F1-Score'],

'Recall': metrics['Recall'],

'ROC-AUC': metrics['ROC-AUC'],

'Loss': metrics['Loss']

})

baseline\_df = pd.DataFrame(baseline\_data)

oversampled\_df = pd.DataFrame.from\_dict(oversampled\_results, orient='index')[

['Model', 'Oversampling', 'F1-Score', 'Recall', 'ROC-AUC', 'Loss']

].rename(columns={'Oversampling': 'Technique'})

all\_results = pd.concat([baseline\_df, oversampled\_df], ignore\_index=True)

# 4.2 Performance Comparison Visualizations (without Loss)

fig, axes = plt.subplots(1, 3, figsize=(20, 8))

fig.suptitle('Performance Comparison: Baseline vs Oversampling Techniques',

fontsize=20, fontweight='bold')

metrics\_to\_plot = ['F1-Score', 'Recall', 'ROC-AUC']

colors = ['red', 'skyblue', 'lightgreen', 'lightcoral', 'gold']

for i, metric in enumerate(metrics\_to\_plot):

ax = axes[i]

pivot\_data = all\_results.pivot(index='Model', columns='Technique', values=metric)

column\_order = ['Baseline', 'ADASYN', 'RandomOverSampler', 'SMOTE', 'SMOTETomek']

pivot\_data = pivot\_data.reindex(columns=[col for col in column\_order if col in pivot\_data.columns])

bars = pivot\_data.plot(kind='bar', ax=ax, width=0.8, color=colors[:len(pivot\_data.columns)])

ax.set\_title(f'{metric} Comparison', fontsize=16, fontweight='bold')

ax.set\_xlabel('Models', fontweight='bold', fontsize=14)

ax.set\_ylabel(metric, fontweight='bold', fontsize=14)

ax.legend(title='Technique', loc='best', fontsize=10)

ax.tick\_params(axis='x', rotation=45, labelsize=12)

ax.grid(True, alpha=0.3)

ax.set\_ylim(0, 1.2)

ax.set\_yticks(np.arange(0, 1.1, 0.1))

# Add value labels on bars

for container in ax.containers:

ax.bar\_label(container, fmt='%.3f', fontsize=10, rotation=90, padding=3)

plt.tight\_layout(pad=3.0)

plt.subplots\_adjust(wspace=0.4)

plt.show()

github link of the output:

<https://github.com/JencyFrancis/26th-aug/commit/e20c992b911ecb84dddefdb04cd33a0d525b36d3>

# CONFUSION MATRICES - BEST PERFORMING COMBINATIONS ONLY

# ======================================================

# Find best performing oversampling technique for each model

best\_combinations = {}

for model\_name in model\_names:

model\_results = {k: v for k, v in oversampled\_results.items() if v['Model'] == model\_name}

best\_key = max(model\_results.keys(), key=lambda k: model\_results[k]['F1-Score'])

best\_combinations[model\_name] = {

'key': best\_key,

'technique': model\_results[best\_key]['Oversampling'],

'f1\_score': model\_results[best\_key]['F1-Score']

}

# Create confusion matrices for baseline and best oversampling only

fig, axes = plt.subplots(2, 3, figsize=(18, 12))

fig.suptitle('Confusion Matrices: Baseline vs Best Oversampling Technique', fontsize=16, fontweight='bold')

# Row 1: Baseline confusion matrices

for idx, model\_name in enumerate(model\_names):

ax = axes[0, idx]

# Baseline confusion matrix

y\_pred = baseline\_predictions[model\_name]

cm = confusion\_matrix(y\_test, y\_pred)

tn, fp, fn, tp = cm.ravel()

annotations = [[f'{tn}\n(TN)', f'{fp}\n(FP)'],

[f'{fn}\n(FN)', f'{tp}\n(TP)']]

sns.heatmap(cm, annot=annotations, fmt='', cmap='Blues',

ax=ax, cbar=False,

xticklabels=['No Failure', 'Failure'],

yticklabels=['No Failure', 'Failure'])

# Calculate metrics

precision = tp / (tp + fp) if (tp + fp) > 0 else 0

recall = tp / (tp + fn) if (tp + fn) > 0 else 0

f1 = baseline\_results[model\_name]['F1-Score']

ax.set\_title(f'{model\_name}\nBaseline (F1: {f1:.3f})', fontsize=12, fontweight='bold')

ax.set\_xlabel('Predicted', fontweight='semibold')

ax.set\_ylabel('Actual', fontweight='semibold')

# Row 2: Best oversampling technique confusion matrices

for idx, model\_name in enumerate(model\_names):

ax = axes[1, idx]

# Best oversampling confusion matrix

best\_key = best\_combinations[model\_name]['key']

best\_technique = best\_combinations[model\_name]['technique']

best\_f1 = best\_combinations[model\_name]['f1\_score']

y\_pred = oversampled\_predictions[best\_key]

cm = confusion\_matrix(y\_test, y\_pred)

tn, fp, fn, tp = cm.ravel()

annotations = [[f'{tn}\n(TN)', f'{fp}\n(FP)'],

[f'{fn}\n(FN)', f'{tp}\n(TP)']]

sns.heatmap(cm, annot=annotations, fmt='', cmap='Greens',

ax=ax, cbar=False,

xticklabels=['No Failure', 'Failure'],

yticklabels=['No Failure', 'Failure'])

ax.set\_title(f'{model\_name}\n{best\_technique} (F1: {best\_f1:.3f})',

fontsize=12, fontweight='bold')

ax.set\_xlabel('Predicted', fontweight='semibold')

ax.set\_ylabel('Actual', fontweight='semibold')

plt.tight\_layout()

plt.show()

# Print summary of improvements

print("\n" + "="\*80)

print("CONFUSION MATRIX ANALYSIS - BEST PERFORMING COMBINATIONS")

print("="\*80)

for model\_name in model\_names:

print(f"\n{model\_name}:")

# Baseline metrics

baseline\_cm = confusion\_matrix(y\_test, baseline\_predictions[model\_name])

tn\_base, fp\_base, fn\_base, tp\_base = baseline\_cm.ravel()

# Best oversampling metrics

best\_key = best\_combinations[model\_name]['key']

best\_technique = best\_combinations[model\_name]['technique']

best\_cm = confusion\_matrix(y\_test, oversampled\_predictions[best\_key])

tn\_best, fp\_best, fn\_best, tp\_best = best\_cm.ravel()

print(f" Best Technique: {best\_technique}")

print(f" Baseline → Best Oversampling:")

print(f" True Positives: {tp\_base} → {tp\_best} ({tp\_best - tp\_base:+d})")

print(f" False Negatives: {fn\_base} → {fn\_best} ({fn\_best - fn\_base:+d})")

print(f" False Positives: {fp\_base} → {fp\_best} ({fp\_best - fp\_base:+d})")

print(f" True Negatives: {tn\_base} → {tn\_best} ({tn\_best - tn\_base:+d})")

# Calculate improvement in failure detection

baseline\_recall = tp\_base / (tp\_base + fn\_base) if (tp\_base + fn\_base) > 0 else 0

best\_recall = tp\_best / (tp\_best + fn\_best) if (tp\_best + fn\_best) > 0 else 0

recall\_improvement = (best\_recall - baseline\_recall) / baseline\_recall \* 100 if baseline\_recall > 0 else 0

print(f" Failure Detection Rate: {baseline\_recall:.2%} → {best\_recall:.2%} "

f"({recall\_improvement:+.1f}% improvement)")

# Create a summary table

summary\_data = []

for model\_name in model\_names:

# Baseline

baseline\_cm = confusion\_matrix(y\_test, baseline\_predictions[model\_name])

tn\_base, fp\_base, fn\_base, tp\_base = baseline\_cm.ravel()

# Best oversampling

best\_key = best\_combinations[model\_name]['key']

best\_technique = best\_combinations[model\_name]['technique']

best\_cm = confusion\_matrix(y\_test, oversampled\_predictions[best\_key])

tn\_best, fp\_best, fn\_best, tp\_best = best\_cm.ravel()

summary\_data.append({

'Model': model\_name,

'Best\_Technique': best\_technique,

'Baseline\_TP': tp\_base,

'Baseline\_FN': fn\_base,

'Baseline\_Recall': tp\_base / (tp\_base + fn\_base) if (tp\_base + fn\_base) > 0 else 0,

'Best\_TP': tp\_best,

'Best\_FN': fn\_best,

'Best\_Recall': tp\_best / (tp\_best + fn\_best) if (tp\_best + fn\_best) > 0 else 0,

'TP\_Improvement': tp\_best - tp\_base,

'FN\_Reduction': fn\_base - fn\_best

})

summary\_df = pd.DataFrame(summary\_data)

styled\_summary = (

summary\_df.style

.hide(axis="index")

.set\_properties(\*\*{'text-align': 'center'})

.set\_table\_styles([

{'selector': 'th, td', 'props': 'border: 1px solid black;'},

{'selector': 'th', 'props': 'background-color: lightgray;'}

])

.format({

'Baseline\_Recall': '{:.2%}',

'Best\_Recall': '{:.2%}'

})

.set\_caption("Failure Detection Improvement Summary")

)

display(styled\_summary)

github link of the output  
<https://github.com/JencyFrancis/26th-aug/commit/bb6867fb79528cf225dcc524d5055b2343ea64fd>

<https://github.com/JencyFrancis/26th-aug/commit/81be136cf81eb7b102a95415aef762dcec48b684>

# 4.4 ROC Curves Comparison - All Models and Techniques (2x2 layout)

fig, axes = plt.subplots(2, 2, figsize=(16, 14))

fig.suptitle('ROC Curves Comparison: Baseline vs Oversampling Techniques',

fontsize=16, fontweight='bold')

# Flatten axes for easier indexing

axes\_flat = axes.flatten()

for i, model\_name in enumerate(model\_names):

ax = axes\_flat[i]

# Plot baseline ROC curve

y\_proba\_baseline = baseline\_probabilities[model\_name]

fpr\_baseline, tpr\_baseline, \_ = roc\_curve(y\_test, y\_proba\_baseline)

roc\_auc\_baseline = auc(fpr\_baseline, tpr\_baseline)

ax.plot(fpr\_baseline, tpr\_baseline, label=f'Baseline (AUC = {roc\_auc\_baseline:.4f})',

linewidth=3, alpha=0.9, color='black', linestyle='--')

# Plot ROC curves for each oversampling technique

colors\_roc = ['red', 'blue', 'green', 'purple']

for j, technique\_name in enumerate(techniques.keys()):

combination\_key = f"{model\_name}\_{technique\_name}"

y\_proba = oversampled\_probabilities[combination\_key]

fpr, tpr, \_ = roc\_curve(y\_test, y\_proba)

roc\_auc\_value = auc(fpr, tpr)

ax.plot(fpr, tpr, label=f'{technique\_name} (AUC = {roc\_auc\_value:.4f})',

linewidth=2, alpha=0.8, color=colors\_roc[j])

# Plot diagonal line

ax.plot([0, 1], [0, 1], 'k--', alpha=0.5, label='Random Classifier (AUC = 0.5000)')

ax.set\_title(f'{model\_name} ROC Curves', fontsize=14, fontweight='bold')

ax.set\_xlabel('False Positive Rate', fontsize=12)

ax.set\_ylabel('True Positive Rate', fontsize=12)

ax.legend(loc='lower right', fontsize=9)

ax.grid(True, alpha=0.3)

# Remove empty subplot

axes\_flat[3].axis('off')

plt.tight\_layout(pad=3.0)

plt.subplots\_adjust(hspace=0.3, wspace=0.3)

plt.show()

github link of the output:

<https://github.com/JencyFrancis/26th-aug/commit/5cf50e5db81232129311bfe8f5f4cd7423576e7a>

# SECTION 5: HYPERPARAMETER TUNING

# =================================

# Identify best performing combinations for tuning

results\_df = pd.DataFrame.from\_dict(oversampled\_results, orient='index')

best\_performers = []

for model\_name in model\_names:

model\_results = results\_df[results\_df['Model'] == model\_name]

best\_idx = model\_results['F1-Score'].idxmax()

best\_technique = model\_results.loc[best\_idx, 'Oversampling']

best\_performers.append((model\_name, best\_technique))

print("\nBEST COMBINATIONS SELECTED FOR TUNING:")

for model, technique in best\_performers:

print(f" {model} + {technique}")

# Define hyperparameter grids

hyperparameter\_grids = {

'KNN': {

'n\_neighbors': [3, 5, 7, 9],

'weights': ['uniform', 'distance'],

'metric': ['euclidean', 'manhattan']

},

'MLP Classifier': {

'hidden\_layer\_sizes': [(50,), (100,), (128, 64), (128, 64, 32)],

'alpha': [0.001, 0.01, 0.1],

'learning\_rate\_init': [0.001, 0.01]

},

'Random Forest': {

'n\_estimators': [50, 100, 200],

'max\_depth': [5, 10, 15, None],

'min\_samples\_split': [2, 5, 10]

}

}

# Perform hyperparameter tuning

tuned\_results = {}

tuned\_predictions = {}

tuned\_probabilities = {}

tuning\_summary = []

for model\_name, best\_technique in best\_performers:

print(f"\nTuning {model\_name} with {best\_technique}...")

# Get resampled data

if model\_name == 'Random Forest':

X\_resampled = resampled\_datasets[best\_technique]['unscaled'][0]

X\_test\_use = X\_test\_unscaled

else:

X\_resampled = resampled\_datasets[best\_technique]['scaled'][0]

X\_test\_use = X\_test\_scaled

y\_resampled = resampled\_datasets[best\_technique]['scaled'][1]

# Initialize model

if model\_name == 'KNN':

base\_model = KNeighborsClassifier()

elif model\_name == 'MLP Classifier':

base\_model = MLPClassifier(random\_state=17, max\_iter=200)

else:

base\_model = RandomForestClassifier(random\_state=17)

# Grid search

grid\_search = GridSearchCV(

base\_model,

hyperparameter\_grids[model\_name],

cv=3,

scoring='f1',

n\_jobs=-1

)

start\_time = time.time()

grid\_search.fit(X\_resampled, y\_resampled)

tuning\_time = time.time() - start\_time

# Get best model and predictions

best\_model = grid\_search.best\_estimator\_

y\_pred = best\_model.predict(X\_test\_use)

y\_proba = best\_model.predict\_proba(X\_test\_use)[:, 1]

# Calculate metrics

test\_f1 = f1\_score(y\_test, y\_pred, zero\_division=0)

combination\_key = f"{model\_name}\_{best\_technique}\_tuned"

tuned\_results[combination\_key] = {

'Model': model\_name,

'Technique': f"{best\_technique} (Tuned)",

'Best\_Params': grid\_search.best\_params\_,

'CV\_Score': grid\_search.best\_score\_,

'Accuracy': accuracy\_score(y\_test, y\_pred),

'Precision': precision\_score(y\_test, y\_pred, zero\_division=0),

'Recall': recall\_score(y\_test, y\_pred, zero\_division=0),

'F1-Score': test\_f1,

'ROC-AUC': roc\_auc\_score(y\_test, y\_proba),

'Loss': log\_loss(y\_test, y\_proba),

'Tuning\_Time': tuning\_time

}

tuned\_predictions[combination\_key] = y\_pred

tuned\_probabilities[combination\_key] = y\_proba

# Add to tuning summary

tuning\_summary.append({

'Model': model\_name,

'Technique': best\_technique,

'CV\_F1\_Score': grid\_search.best\_score\_,

'Test\_F1\_Score': test\_f1,

'Tuning\_Time\_s': tuning\_time,

'Best\_Parameters': str(grid\_search.best\_params\_)

})

print(f" Best params: {grid\_search.best\_params\_}")

print(f" CV F1-Score: {grid\_search.best\_score\_:.4f}")

print(f" Test F1-Score: {test\_f1:.4f}")

# Display tuning summary table

print("\n" + "="\*80)

print("HYPERPARAMETER TUNING SUMMARY")

print("="\*80)

tuning\_df = pd.DataFrame(tuning\_summary)

styled\_tuning\_table = (

tuning\_df.style

.hide(axis="index")

.set\_properties(\*\*{'text-align': 'center'})

.set\_table\_styles([

{'selector': 'th, td', 'props': 'border: 1px solid black;'},

{'selector': 'th', 'props': 'background-color: lightgray;'}

])

.format({

'CV\_F1\_Score': '{:.4f}',

'Test\_F1\_Score': '{:.4f}',

'Tuning\_Time\_s': '{:.1f}'

})

.set\_caption("Hyperparameter Optimization Results")

)

display(styled\_tuning\_table)

github link of the output:

<https://github.com/JencyFrancis/26th-aug/commit/1ff713347f72f0a209d56f61bfdba8b2ddc51e29>

# SECTION 6: COMPREHENSIVE COMPARISON - ALL STAGES

# ================================================

# 6.1 Create complete results DataFrame

all\_stage\_results = []

# Add baseline results

for model\_name, metrics in baseline\_results.items():

all\_stage\_results.append({

'Model': model\_name,

'Stage': 'Baseline',

'Technique': 'None',

'F1-Score': metrics['F1-Score'],

'Recall': metrics['Recall'],

'ROC-AUC': metrics['ROC-AUC'],

'Loss': metrics['Loss']

})

# Add oversampled results

for key, metrics in oversampled\_results.items():

all\_stage\_results.append({

'Model': metrics['Model'],

'Stage': 'Oversampled',

'Technique': metrics['Oversampling'],

'F1-Score': metrics['F1-Score'],

'Recall': metrics['Recall'],

'ROC-AUC': metrics['ROC-AUC'],

'Loss': metrics['Loss']

})

# Add tuned results

for key, metrics in tuned\_results.items():

all\_stage\_results.append({

'Model': metrics['Model'],

'Stage': 'Tuned',

'Technique': metrics['Technique'],

'F1-Score': metrics['F1-Score'],

'Recall': metrics['Recall'],

'ROC-AUC': metrics['ROC-AUC'],

'Loss': metrics['Loss']

})

all\_stage\_df = pd.DataFrame(all\_stage\_results)

# Show count of experiments

print(f"\nTotal experiments conducted: {len(all\_stage\_df)}")

print(f"- Baseline experiments: {len(all\_stage\_df[all\_stage\_df['Stage']=='Baseline'])}")

print(f"- Oversampling experiments: {len(all\_stage\_df[all\_stage\_df['Stage']=='Oversampled'])}")

print(f"- Tuning experiments: {len(all\_stage\_df[all\_stage\_df['Stage']=='Tuned'])}")

output:

Total experiments conducted: 18

- Baseline experiments: 3

- Oversampling experiments: 12

- Tuning experiments: 3

# 6.2 Stage-wise Performance Comparison (without Loss)

fig, axes = plt.subplots(1, 3, figsize=(20, 8))

fig.suptitle('Performance Evolution: Baseline → Oversampling → Tuning',

fontsize=20, fontweight='bold')

metrics\_to\_compare = ['F1-Score', 'Recall', 'ROC-AUC']

for i, metric in enumerate(metrics\_to\_compare):

ax = axes[i]

# Create grouped bar plot by model

model\_groups = []

for model in model\_names:

model\_data = all\_stage\_df[all\_stage\_df['Model'] == model]

baseline\_val = model\_data[model\_data['Stage'] == 'Baseline'][metric].values[0]

oversampled\_vals = model\_data[model\_data['Stage'] == 'Oversampled'][metric].values

tuned\_val = model\_data[model\_data['Stage'] == 'Tuned'][metric].values[0] if len(model\_data[model\_data['Stage'] == 'Tuned']) > 0 else 0

model\_groups.append({

'Model': model,

'Baseline': baseline\_val,

'Best Oversampled': oversampled\_vals.max() if len(oversampled\_vals) > 0 else 0,

'Tuned': tuned\_val

})

comparison\_df = pd.DataFrame(model\_groups)

bars = comparison\_df.set\_index('Model').plot(kind='bar', ax=ax, width=0.8)

ax.set\_title(f'{metric} Evolution', fontsize=16, fontweight='bold')

ax.set\_xlabel('Models', fontsize=14, fontweight='bold')

ax.set\_ylabel(metric, fontsize=14, fontweight='bold')

ax.legend(title='Stage', fontsize=10)

ax.tick\_params(axis='x', rotation=45, labelsize=12)

ax.grid(True, alpha=0.3)

ax.set\_ylim(0, 1.2)

ax.set\_yticks(np.arange(0, 1.1, 0.1))

# Add value labels on bars

for container in ax.containers:

if i == 2: # ROC-AUC (3rd plot)

ax.bar\_label(container, fmt='%.3f', fontsize=10, rotation=90, padding=3)

else:

ax.bar\_label(container, fmt='%.3f', fontsize=10, rotation=0, padding=3)

plt.tight\_layout(pad=3.0)

plt.subplots\_adjust(wspace=0.4)

plt.show()

github libnk of the output:

<https://github.com/JencyFrancis/26th-aug/commit/e4f26e29c25798967a08657e5e0278bc87f97552>

# 6.4 Final Performance Summary Table

print("\n" + "="\*80)

print("FINAL PERFORMANCE SUMMARY - ALL STAGES")

print("="\*80)

# Create summary for each model

summary\_data = []

for model\_name in model\_names:

# Baseline

baseline\_metrics = baseline\_results[model\_name]

# Best oversampled

model\_oversampled = {k: v for k, v in oversampled\_results.items() if v['Model'] == model\_name}

best\_oversampled\_key = max(model\_oversampled.keys(), key=lambda k: model\_oversampled[k]['F1-Score'])

best\_oversampled = model\_oversampled[best\_oversampled\_key]

# Tuned

tuned\_key = [k for k in tuned\_results.keys() if tuned\_results[k]['Model'] == model\_name][0]

tuned = tuned\_results[tuned\_key]

summary\_data.append({

'Model': model\_name,

'Baseline F1': baseline\_metrics['F1-Score'],

'Baseline Recall': baseline\_metrics['Recall'],

'Baseline ROC-AUC': baseline\_metrics['ROC-AUC'],

'Best Oversampling': best\_oversampled['Oversampling'],

'Oversampled F1': best\_oversampled['F1-Score'],

'Oversampled Recall': best\_oversampled['Recall'],

'Oversampled ROC-AUC': best\_oversampled['ROC-AUC'],

'Tuned F1': tuned['F1-Score'],

'Tuned Recall': tuned['Recall'],

'Tuned ROC-AUC': tuned['ROC-AUC'],

'F1 Improvement (%)': ((tuned['F1-Score'] - baseline\_metrics['F1-Score']) / baseline\_metrics['F1-Score'] \* 100) if baseline\_metrics['F1-Score'] > 0 else 0,

'Recall Improvement (%)': ((tuned['Recall'] - baseline\_metrics['Recall']) / baseline\_metrics['Recall'] \* 100) if baseline\_metrics['Recall'] > 0 else 0,

'ROC-AUC Improvement (%)': ((tuned['ROC-AUC'] - baseline\_metrics['ROC-AUC']) / baseline\_metrics['ROC-AUC'] \* 100) if baseline\_metrics['ROC-AUC'] > 0 else 0

})

summary\_df = pd.DataFrame(summary\_data)

# Display styled summary table

styled\_summary = (

summary\_df.style

.hide(axis="index")

.set\_properties(\*\*{'text-align': 'center'})

.set\_table\_styles([

{'selector': 'th, td', 'props': 'border: 1px solid black;'},

{'selector': 'th', 'props': 'background-color: lightgray;'}

])

.format({

'Baseline F1': '{:.4f}',

'Baseline Recall': '{:.4f}',

'Baseline ROC-AUC': '{:.4f}',

'Oversampled F1': '{:.4f}',

'Oversampled Recall': '{:.4f}',

'Oversampled ROC-AUC': '{:.4f}',

'Tuned F1': '{:.4f}',

'Tuned Recall': '{:.4f}',

'Tuned ROC-AUC': '{:.4f}',

'F1 Improvement (%)': '{:+.2f}%',

'Recall Improvement (%)': '{:+.2f}%',

'ROC-AUC Improvement (%)': '{:+.2f}%'

})

.set\_caption("Complete Performance Evolution Summary")

)

display(styled\_summary)

print("\nNote: Improvements are calculated as (Tuned - Baseline) / Baseline × 100%")

github link of the output:

<https://github.com/JencyFrancis/26th-aug/commit/5a4cd6d68de01f2c4d7e5c06c05022f488142ee9>

# 6.5 Performance Improvement Validation (with ROC-AUC)

print("\n" + "="\*80)

print("PERFORMANCE IMPROVEMENT VALIDATION")

print("="\*80)

# Create validation summary

validation\_data = []

for model\_name in model\_names:

baseline\_f1 = baseline\_results[model\_name]['F1-Score']

baseline\_recall = baseline\_results[model\_name]['Recall']

baseline\_roc = baseline\_results[model\_name]['ROC-AUC']

# Find best tuned result

tuned\_key = [k for k in tuned\_results.keys() if model\_name in k][0]

tuned\_f1 = tuned\_results[tuned\_key]['F1-Score']

tuned\_recall = tuned\_results[tuned\_key]['Recall']

tuned\_roc = tuned\_results[tuned\_key]['ROC-AUC']

# Calculate improvements - handle all cases including negative improvements

if baseline\_f1 > 0:

f1\_improvement = ((tuned\_f1 - baseline\_f1) / baseline\_f1) \* 100

else:

f1\_improvement = 100 if tuned\_f1 > 0 else 0

if baseline\_recall > 0:

recall\_improvement = ((tuned\_recall - baseline\_recall) / baseline\_recall) \* 100

else:

recall\_improvement = 100 if tuned\_recall > 0 else 0

if baseline\_roc > 0:

roc\_improvement = ((tuned\_roc - baseline\_roc) / baseline\_roc) \* 100

else:

roc\_improvement = 100 if tuned\_roc > 0 else 0

# Determine meaningful improvement based on F1-score improvement

meaningful = '✓' if f1\_improvement > 5 else '✗'

validation\_data.append({

'Model': model\_name,

'Baseline\_F1': baseline\_f1,

'Tuned\_F1': tuned\_f1,

'F1\_Change': f'{f1\_improvement:+.1f}%',

'Baseline\_Recall': baseline\_recall,

'Tuned\_Recall': tuned\_recall,

'Recall\_Change': f'{recall\_improvement:+.1f}%',

'Baseline\_ROC\_AUC': baseline\_roc,

'Tuned\_ROC\_AUC': tuned\_roc,

'ROC\_AUC\_Change': f'{roc\_improvement:+.1f}%',

'Meaningful\_Improvement': meaningful

})

validation\_df = pd.DataFrame(validation\_data)

styled\_validation\_table = (

validation\_df.style

.hide(axis="index")

.set\_properties(\*\*{'text-align': 'center'})

.set\_table\_styles([

{'selector': 'th, td', 'props': 'border: 1px solid black;'},

{'selector': 'th', 'props': 'background-color: lightgray;'}

])

.format({

'Baseline\_F1': '{:.4f}',

'Tuned\_F1': '{:.4f}',

'Baseline\_Recall': '{:.4f}',

'Tuned\_Recall': '{:.4f}',

'Baseline\_ROC\_AUC': '{:.4f}',

'Tuned\_ROC\_AUC': '{:.4f}'

})

.set\_caption("Performance Improvement Validation")

)

display(styled\_validation\_table)

print("\nNote: Improvements are calculated as (Tuned - Baseline) / Baseline × 100%")

github link of the output:  
  
<https://github.com/JencyFrancis/26th-aug/commit/fd1e1dccf9e64b7ce4424f296d3e696e949ad0a9>

# SECTION 7: OVERFITTING/UNDERFITTING ANALYSIS

# ============================================

print("\n" + "="\*80)

print("OVERFITTING/UNDERFITTING ANALYSIS FOR TOP PERFORMERS")

print("="\*80)

# Identify 3 best performing model+oversampling combinations

all\_combinations = []

for key, result in oversampled\_results.items():

all\_combinations.append({

'Combination': key,

'Model': result['Model'],

'Technique': result['Oversampling'],

'F1-Score': result['F1-Score']

})

# Sort and get top 3

sorted\_combinations = sorted(all\_combinations, key=lambda x: x['F1-Score'], reverse=True)

# Ensure KNN + RandomOverSampler is included

knn\_ros = next((x for x in sorted\_combinations if x['Model'] == 'KNN' and x['Technique'] == 'RandomOverSampler'), None)

if knn\_ros:

# If KNN+ROS is not in top 3, replace the 3rd one

if knn\_ros not in sorted\_combinations[:3]:

top\_3\_combinations = sorted\_combinations[:2] + [knn\_ros]

else:

top\_3\_combinations = sorted\_combinations[:3]

else:

top\_3\_combinations = sorted\_combinations[:3]

print("\nTop 3 Performing Combinations:")

for i, combo in enumerate(top\_3\_combinations):

print(f"{i+1}. {combo['Model']} + {combo['Technique']} (F1: {combo['F1-Score']:.4f})")

# Analyze overfitting/underfitting for top 3

overfitting\_analysis = []

for combo in top\_3\_combinations:

model\_name = combo['Model']

technique = combo['Technique']

# Get appropriate model and data

if model\_name == 'Random Forest':

X\_use = resampled\_datasets[technique]['unscaled'][0]

model = RandomForestClassifier(random\_state=17)

else:

X\_use = resampled\_datasets[technique]['scaled'][0]

if model\_name == 'KNN':

model = KNeighborsClassifier()

else:

model = MLPClassifier(hidden\_layer\_sizes=(128, 64, 32),

max\_iter=200, random\_state=17)

y\_use = resampled\_datasets[technique]['scaled'][1]

# Generate learning curve for final analysis

train\_sizes\_abs, train\_scores, val\_scores = learning\_curve(

model, X\_use, y\_use, train\_sizes=[0.3, 0.5, 0.7, 0.9, 1.0],

cv=3, scoring='f1', n\_jobs=-1, random\_state=17

)

final\_train = np.mean(train\_scores[-1])

final\_val = np.mean(val\_scores[-1])

gap = final\_train - final\_val

# Determine status

if gap > 0.05:

if final\_val < 0.7:

status = "OVERFITTING - High bias, high variance"

recommendation = "Reduce complexity, add regularization"

else:

status = "MILD OVERFITTING - Acceptable"

recommendation = "Consider slight regularization"

elif gap < -0.02:

status = "GOOD GENERALIZATION"

recommendation = "Model is performing optimally"

elif final\_val < 0.6:

status = "UNDERFITTING - High bias"

recommendation = "Increase model complexity"

else:

status = "WELL-FITTED"

recommendation = "Model is well-tuned"

overfitting\_analysis.append({

'Model\_Technique': f"{model\_name} + {technique}",

'Training\_Score': final\_train,

'Validation\_Score': final\_val,

'Gap': gap,

'Status': status,

'Recommendation': recommendation

})

# Display overfitting analysis table

overfitting\_df = pd.DataFrame(overfitting\_analysis)

styled\_overfitting\_table = (

overfitting\_df.style

.hide(axis="index")

.set\_properties(\*\*{'text-align': 'center'})

.set\_table\_styles([

{'selector': 'th, td', 'props': 'border: 1px solid black;'},

{'selector': 'th', 'props': 'background-color: lightgray;'}

])

.format({

'Training\_Score': '{:.4f}',

'Validation\_Score': '{:.4f}',

'Gap': '{:.4f}'

})

.set\_caption("Overfitting/Underfitting Analysis for Top Performers")

)

display(styled\_overfitting\_table)

github link of the output:

<https://github.com/JencyFrancis/26th-aug/commit/c0c4f680ea92da3320772d0a82923173cc4c66b4>

# SECTION 8: RESEARCH FINDINGS AND CONCLUSIONS

# ============================================

print("\n" + "="\*80)

print("RESEARCH FINDINGS - ANSWERING THE RESEARCH QUESTION")

print("="\*80)

print("\nResearch Question: Does oversampling imbalanced data improve the performance of")

print("Random Forest, MLP Classifier, and KNN in predicting machine failure from sensor data?")

print("\nANSWER: YES, with varying degrees of improvement across models and metrics.")

# Find absolute best model + oversampling combination

best\_overall = max([(k, v) for k, v in oversampled\_results.items()],

key=lambda x: x[1]['F1-Score'])

best\_model = best\_overall[1]['Model']

best\_technique = best\_overall[1]['Oversampling']

best\_f1\_score = best\_overall[1]['F1-Score']

print(f"\n★ BEST OVERALL COMBINATION: {best\_model} + {best\_technique}")

print(f" Achieves F1-Score: {best\_f1\_score:.4f}")

# Calculate average improvements

avg\_improvements = []

for model\_name in model\_names:

baseline\_f1 = baseline\_results[model\_name]['F1-Score']

baseline\_recall = baseline\_results[model\_name]['Recall']

baseline\_roc = baseline\_results[model\_name]['ROC-AUC']

# Get best oversampled results

model\_oversampled = {k: v for k, v in oversampled\_results.items() if v['Model'] == model\_name}

best\_oversampled = max(model\_oversampled.items(), key=lambda x: x[1]['F1-Score'])

best\_technique\_model = best\_oversampled[1]['Oversampling']

best\_f1 = best\_oversampled[1]['F1-Score']

best\_recall = max([v['Recall'] for v in model\_oversampled.values()])

best\_roc = max([v['ROC-AUC'] for v in model\_oversampled.values()])

# Calculate improvements

f1\_imp = ((best\_f1 - baseline\_f1) / baseline\_f1 \* 100) if baseline\_f1 > 0 else 0

recall\_imp = ((best\_recall - baseline\_recall) / baseline\_recall \* 100) if baseline\_recall > 0 else 0

roc\_imp = ((best\_roc - baseline\_roc) / baseline\_roc \* 100) if baseline\_roc > 0 else 0

avg\_improvements.append({

'Model': model\_name,

'Best\_Technique': best\_technique\_model,

'F1 Improvement': f1\_imp,

'Recall Improvement': recall\_imp,

'ROC-AUC Improvement': roc\_imp

})

improvements\_df = pd.DataFrame(avg\_improvements)

print("\nKEY FINDINGS:")

print("\n1. F1-Score Improvements:")

for \_, row in improvements\_df.iterrows():

improvement = row['F1 Improvement']

print(f" {row['Model']} + {row['Best\_Technique']}: {improvement:+.2f}%")

print("\n2. Recall Improvements (Critical for Failure Detection):")

for \_, row in improvements\_df.iterrows():

improvement = row['Recall Improvement']

print(f" {row['Model']} + {row['Best\_Technique']}: {improvement:+.2f}%")

print("\n3. ROC-AUC Improvements:")

for \_, row in improvements\_df.iterrows():

improvement = row['ROC-AUC Improvement']

print(f" {row['Model']} + {row['Best\_Technique']}: {improvement:+.2f}%")

# Best techniques analysis

print("\n4. Most Effective Oversampling Techniques:")

technique\_effectiveness = {}

for key, result in oversampled\_results.items():

technique = result['Oversampling']

if technique not in technique\_effectiveness:

technique\_effectiveness[technique] = []

technique\_effectiveness[technique].append(result['F1-Score'])

for technique, scores in technique\_effectiveness.items():

avg\_score = np.mean(scores)

print(f" {technique}: Average F1-Score = {avg\_score:.4f}")

print("\n5. Model-Specific Recommendations:")

for model\_name in model\_names:

model\_results = results\_df[results\_df['Model'] == model\_name]

best\_technique = model\_results.loc[model\_results['F1-Score'].idxmax(), 'Oversampling']

best\_f1 = model\_results['F1-Score'].max()

print(f" {model\_name}: Use {best\_technique} (F1-Score: {best\_f1:.4f})")

print("\nCONCLUSION:")

print("Oversampling techniques significantly improve the performance of all three models")

print("in predicting machine failures from imbalanced sensor data. The improvements are")

print("most pronounced in recall scores, which is critical for failure detection systems")

print("where missing a failure (false negative) is more costly than false alarms.")

print("\nSPECIFIC INSIGHTS:")

print("1. KNN showed the highest relative improvement, suggesting it benefits most from balanced data")

print("2. Random Forest maintained strong performance even with imbalanced data")

print("3. MLP Classifier showed consistent improvements across all metrics")

print("4. SMOTE and RandomOverSampler were generally the most effective techniques")

print("5. The combination of oversampling and hyperparameter tuning yielded the best results")

# Save all results to CSV for documentation

all\_results\_export = pd.concat([

pd.DataFrame.from\_dict(baseline\_results, orient='index').reset\_index().rename(columns={'index': 'Configuration'}),

pd.DataFrame.from\_dict(oversampled\_results, orient='index').reset\_index().rename(columns={'index': 'Configuration'}),

pd.DataFrame.from\_dict(tuned\_results, orient='index').reset\_index().rename(columns={'index': 'Configuration'})

], ignore\_index=True)

all\_results\_export.to\_csv('machine\_failure\_prediction\_results.csv', index=False)

print("\nResults saved to 'machine\_failure\_prediction\_results.csv'")

# Final recommendations

print("\n" + "="\*80)

print("FINAL RECOMMENDATIONS FOR IMPLEMENTATION")

print("="\*80)

print("\n1. For Maximum Recall (Catching Most Failures):")

best\_recall\_combo = max([(k, v) for k, v in oversampled\_results.items()],

key=lambda x: x[1]['Recall'])

print(f" Use {best\_recall\_combo[1]['Model']} + {best\_recall\_combo[1]['Oversampling']}")

print(f" Achieves {best\_recall\_combo[1]['Recall']:.2%} recall")

print("\n2. For Balanced Performance (F1-Score):")

best\_f1\_combo = max([(k, v) for k, v in oversampled\_results.items()],

key=lambda x: x[1]['F1-Score'])

print(f" Use {best\_f1\_combo[1]['Model']} + {best\_f1\_combo[1]['Oversampling']}")

print(f" Achieves {best\_f1\_combo[1]['F1-Score']:.4f} F1-Score")

print("\n3. For Production Deployment:")

print(" - Implement real-time monitoring with the chosen model")

print(" - Set up alerts for predicted failures")

print(" - Continuously collect data to retrain and improve the model")

print(" - Consider ensemble methods combining multiple models for robustness")

print("\n" + "="\*80)

print("END OF ANALYSIS")

print("="\*80)

output;

================================================================================

RESEARCH FINDINGS - ANSWERING THE RESEARCH QUESTION

================================================================================

Research Question: Does oversampling imbalanced data improve the performance of

Random Forest, MLP Classifier, and KNN in predicting machine failure from sensor data?

ANSWER: YES, with varying degrees of improvement across models and metrics.

★ BEST OVERALL COMBINATION: Random Forest + RandomOverSampler

Achieves F1-Score: 0.7667

KEY FINDINGS:

1. F1-Score Improvements:

Random Forest + RandomOverSampler: +7.33%

MLP Classifier + RandomOverSampler: -6.57%

KNN + RandomOverSampler: +17.86%

2. Recall Improvements (Critical for Failure Detection):

Random Forest + RandomOverSampler: +35.00%

MLP Classifier + RandomOverSampler: +43.59%

KNN + RandomOverSampler: +150.00%

3. ROC-AUC Improvements:

Random Forest + RandomOverSampler: +1.39%

MLP Classifier + RandomOverSampler: +0.22%

KNN + RandomOverSampler: +3.50%

4. Most Effective Oversampling Techniques:

SMOTE: Average F1-Score = 0.5459

ADASYN: Average F1-Score = 0.5393

RandomOverSampler: Average F1-Score = 0.6309

SMOTETomek: Average F1-Score = 0.5380

5. Model-Specific Recommendations:

Random Forest: Use RandomOverSampler (F1-Score: 0.7667)

MLP Classifier: Use RandomOverSampler (F1-Score: 0.6023)

KNN: Use RandomOverSampler (F1-Score: 0.5238)

CONCLUSION:

Oversampling techniques significantly improve the performance of all three models

in predicting machine failures from imbalanced sensor data. The improvements are

most pronounced in recall scores, which is critical for failure detection systems

where missing a failure (false negative) is more costly than false alarms.

SPECIFIC INSIGHTS:

1. KNN showed the highest relative improvement, suggesting it benefits most from balanced data

2. Random Forest maintained strong performance even with imbalanced data

3. MLP Classifier showed consistent improvements across all metrics

4. SMOTE and RandomOverSampler were generally the most effective techniques

5. The combination of oversampling and hyperparameter tuning yielded the best results

Results saved to 'machine\_failure\_prediction\_results.csv'

================================================================================

FINAL RECOMMENDATIONS FOR IMPLEMENTATION

================================================================================

1. For Maximum Recall (Catching Most Failures):

Use MLP Classifier + ADASYN

Achieves 84.85% recall

2. For Balanced Performance (F1-Score):

Use Random Forest + RandomOverSampler

Achieves 0.7667 F1-Score

3. For Production Deployment:

- Implement real-time monitoring with the chosen model

- Set up alerts for predicted failures

- Continuously collect data to retrain and improve the model

- Consider ensemble methods combining multiple models for robustness

================================================================================

END OF ANALYSIS