This is an MSc Data science project.

Datasetlink is :
https://archive.ics.uci.edu/dataset/601/ai4i+2020+predictive+maintenance+dataset

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()
```

```python
# Check and print the shape of the dataset
print("\nShape of Dataset (rows, columns):")
print(df.shape)
```

```
Shape of Dataset (rows, columns):
(10000, 15)
```

```python
# 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']
```

```python
# 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']
```

```python
# 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
```

```python
# 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
```

```python
# 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	2886.8	2.000259	1.483734	179.284096	9.968934	63.654147	0.180981	0.067671	0.106625	0.097009	0.098514	0.04355

	UID	Air_temperature	Process_temperature	Rotational_speed	Torque	Tool_wear	Machine_failure	TWF	HDF	PWF	OSF	RNF
	9568											
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

```python
# 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	-0.729296	0.634238	-1.648511	0	0	0	0	0	0	0	47181	1

	UDI	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
						04											
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

```python
# 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
```

```python
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'>
```

```python
# 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().so
rt_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')
```

```python
# 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
```

```python
    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

```python
# SECTION 6: COMPREHENSIVE COMPARISON - ALL STAGES
# ==================================================

# 6.1 Create complete results DataFrame
all_stage_results = []
```

```python
# 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
```

```python
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
```