National College of Ireland

Configuration Manual

MSc Research Project
Data Analytics

Ashutosh Alone
Student ID: x22228381

School of Computing
National College of Ireland

Supervisor: Dr. Ahmed Makki

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name: ……. ……Ashutosh Alone……

Student ID: ……x22228381……

Programme: ……Data Analytics…… **Year:** ……2024……..

Module: ……Research Project ……

Lecturer: ……Dr. Ahmed Makki ……

Submission Due Date: …12/08/2024 ……

Project Title: ……Predictive maintenance in industrial sector using machine learning……

Word Count: …… **Page Count:** ……11……

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: ……Ashutosh Alone ……

Date: ……11/08/2024……

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Ashutosh Alone
Student ID: x22228381

1 Introduction

This configuration manual contains the details of all system requirements which includes both software and hardware requirements for successful implementation of the project. Also, various processes and techniques used for completion of this project is included here.

2 System Requirements

2.1 Hardware requirements:

System Specifications
Processor: AMD Ryzen 5 3550H with Radeon Vega Mobile Gfx, 2.10 GHz
Installed RAM: 16.0 GB (15.7 GB usable)
System type: 64-bit operating system, x64-based processor
Edition: Windows 10 Home Single Language
Version:22H2
Installed on: 17-11-2023
OS build: 19045.4291
Experience: Windows Feature Experience Pack 1000.19056.1000.0

Also, the device storage of the PC used for this project is 1 TB hard disc drive and 500 GB solid state drive.

2.2 Software requirements:

All the code artifacts for this project are written in python language with the version 3.11.4. the IDE used for all the processes like data processing, visualization and model training and evaluation is jupyter notebook with version of 6.5.4 by anaconda navigator.

3 Data Handling

This section covers how the data handling is done from loading the dataset to preprocessing the data for model training.

3.1 Data Loading:

Dataset used in this project is 'ai4i2020.csv' which is loaded into dataframe using 'pandas' library McKinney (2011). And then first 10 lines of the dataset are printed.

```
# Load the dataset
df = pd.read_csv('ai4i2020.csv')
df.head(10)
```

	UDI	Product ID	Type	Air temperature [K]	Process temperature [K]	Rotational speed [rpm]	Torque [Nm]	Tool wear [min]	Machine failure	TWF	HDF	PWF	OSF	RNF
0	1	M14860	M	298.1	308.6	1551	42.8	0	0	0	0	0	0	0
1	2	L47181	L	298.2	308.7	1408	46.3	3	0	0	0	0	0	0
2	3	L47182	L	298.1	308.5	1498	49.4	5	0	0	0	0	0	0
3	4	L47183	L	298.2	308.6	1433	39.5	7	0	0	0	0	0	0
4	5	L47184	L	298.2	308.7	1408	40.0	9	0	0	0	0	0	0
5	6	M14865	M	298.1	308.6	1425	41.9	11	0	0	0	0	0	0
6	7	L47186	L	298.1	308.6	1558	42.4	14	0	0	0	0	0	0
7	8	L47187	L	298.1	308.6	1527	40.2	16	0	0	0	0	0	0
8	9	M14868	M	298.3	308.7	1667	28.6	18	0	0	0	0	0	0
9	10	M14869	M	298.5	309.0	1741	28.0	21	0	0	0	0	0	0

3.2 Data preprocessing:

For data preprocessing first the UDI column is dropped since it was a unique ID column and then the categorical variables are converted to numerical columns. Then for having the same scaling among all variables the features are scaled using 'MinMaxScalar' (Amorim et al., 2022).

DATA PREPROCESSING

```
# Udi column is dropped because its unnecessary for prediction
df = df.drop('UDI', axis=1)
```

```
# covert categorical column to numerical column
label_encoder_product = LabelEncoder()
df['Product ID'] = label_encoder_product.fit_transform(df['Product ID'])

label_encoder_type = LabelEncoder()
df['Type'] = label_encoder_type.fit_transform(df['Type'])
```

```
# Separate features and target variable
X_new = df.drop('Machine failure', axis=1)
y_new = df['Machine failure']

# Scale the features using Min-Max Scaling
scaler_new = MinMaxScaler()
X_scaled_new = scaler_new.fit_transform(X_new)
```

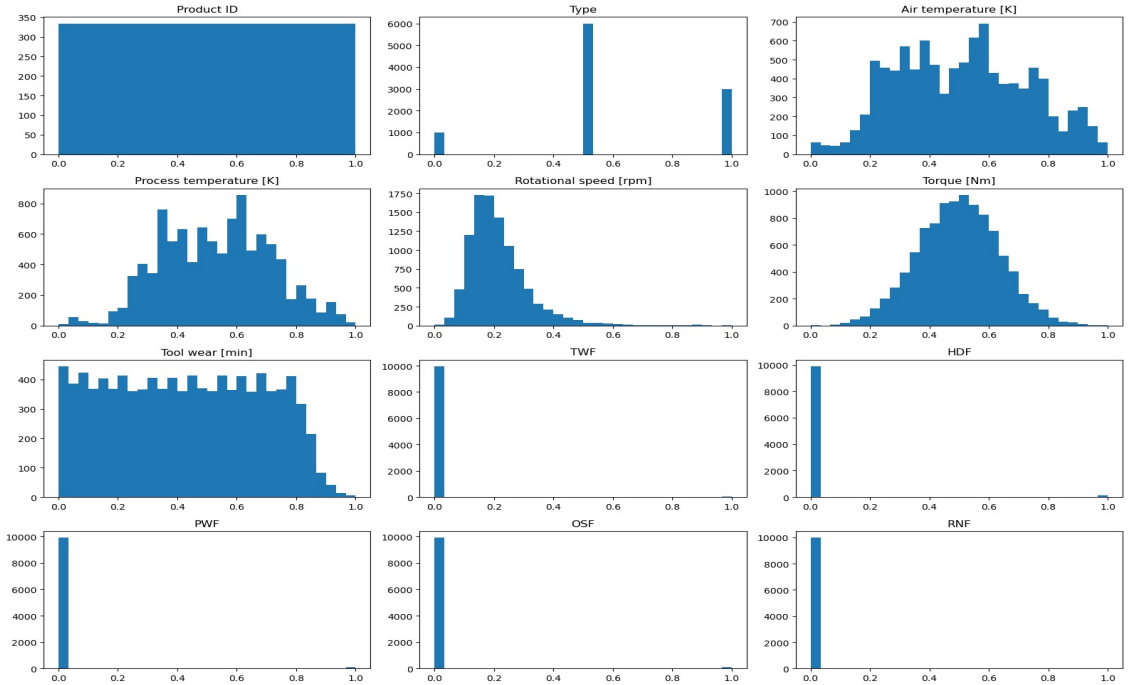
First few rows of pre-processed is data is then printed.

```
: # Display the first few rows of the preprocessed dataset
preprocessed_data_new = pd.DataFrame(X_scaled_new, columns=X_new.columns)
preprocessed_data_new['Machine failure'] = y_new.values

print("\nFirst few rows of the preprocessed dataset:")
print(preprocessed_data_new.head())
```

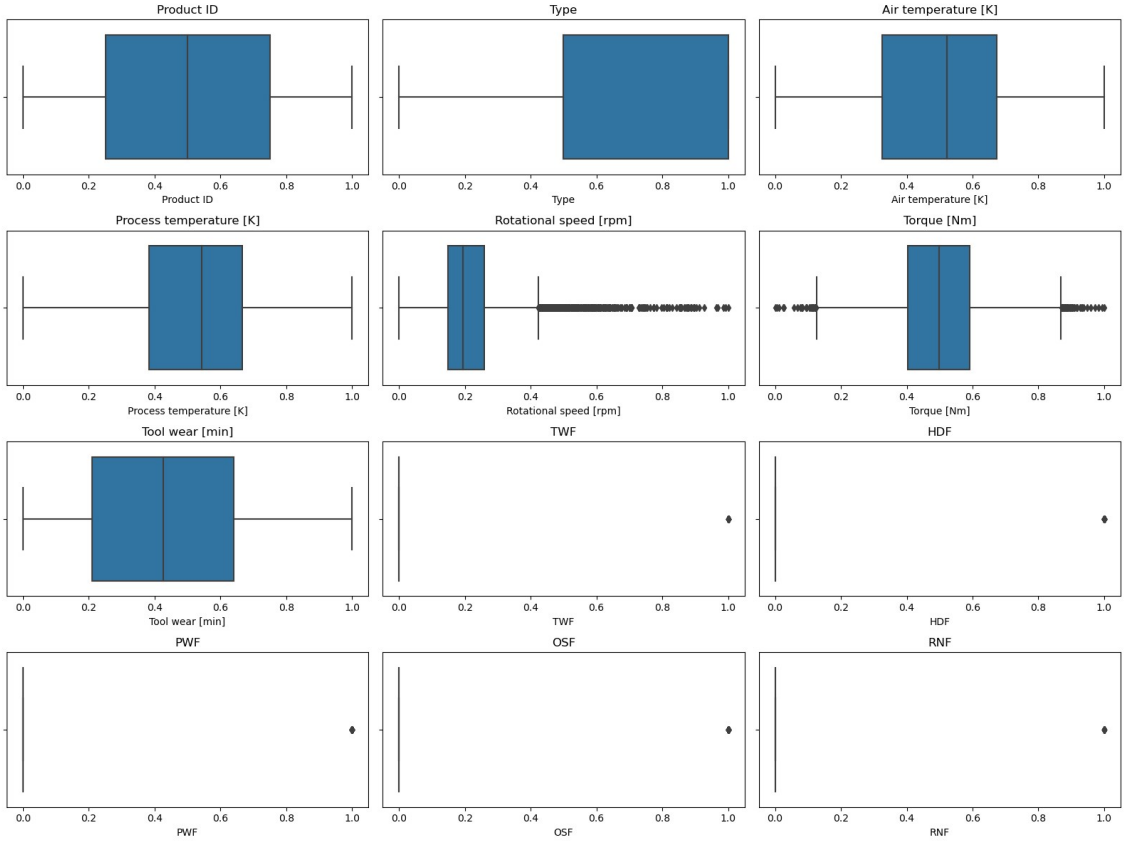
```
First few rows of the preprocessed dataset:
   Product ID  Type  Air temperature [K]  Process temperature [K]  \
0     0.70037   1.0             0.304348                 0.358025
1     0.10031   0.5             0.315217                 0.370370
2     0.10041   0.5             0.304348                 0.345679
3     0.10051   0.5             0.315217                 0.358025
4     0.10061   0.5             0.315217                 0.370370

   Rotational speed [rpm]  Torque [Nm]  Tool wear [min]  TWF  HDF  PWF  OSF  \
0                0.222934     0.535714         0.000000  0.0  0.0  0.0  0.0
1                0.139697     0.583791         0.011858  0.0  0.0  0.0  0.0
2                0.192084     0.626374         0.019763  0.0  0.0  0.0  0.0
3                0.154249     0.490385         0.027668  0.0  0.0  0.0  0.0
4                0.139697     0.497253         0.035573  0.0  0.0  0.0  0.0

   RNF  Machine failure
0  0.0                0
1  0.0                0
2  0.0                0
3  0.0                0
4  0.0                0
```

3.3 Exploratory Data Analysis:

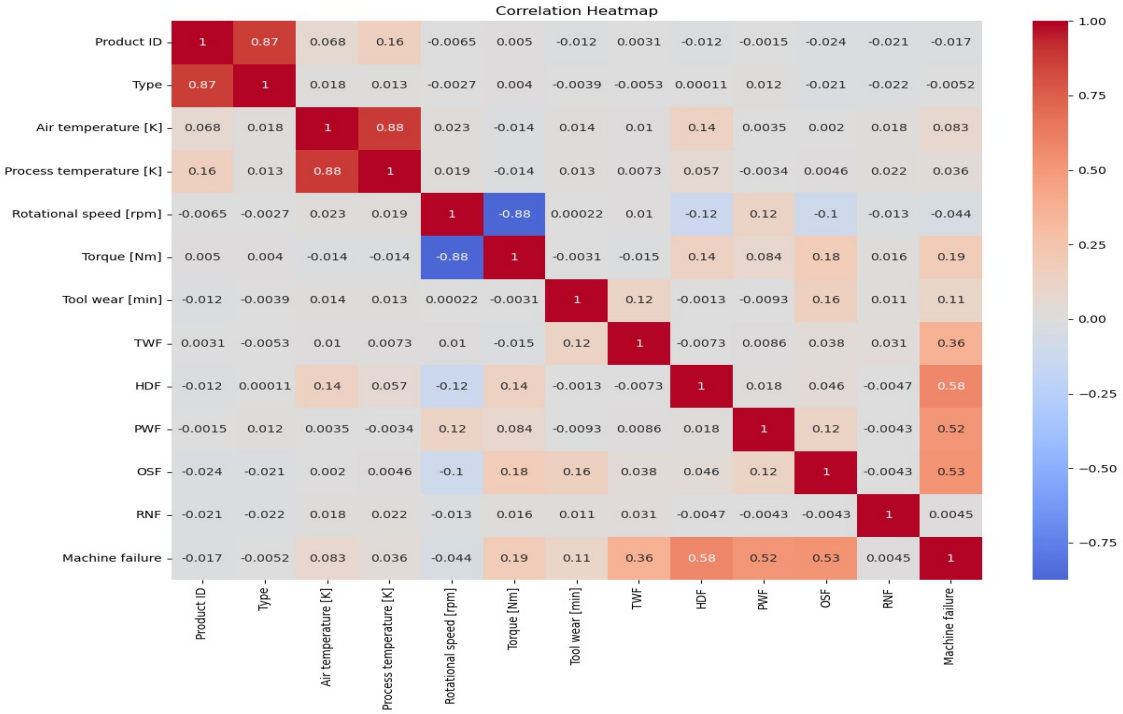
All the features in the pre-processed dataset are shown using various visualizations.
The visualization includes histogram, boxplot and correlation heatmap for the features.



Boxplot:



Heatmap:



4 Model Implementation

Three unsupervised models which are Isolation Forest, One-Class SVM and Local Outlier Factor are implemented and then for benchmark comparison two supervised learning models such as KNN and Random Forest are used.

4.1 Isolation Forest:

Isolation Forest is trained on the entire data with random state of '42' and contamination set to 'auto'.

Isolation Forest

```
# Isolation Forest model
isolation_forest = IsolationForest(contamination='auto', random_state=42)
isolation_forest.fit(X_scaled_new)

y_pred_if = isolation_forest.predict(X_scaled_new)
y_pred_if = np.where(y_pred_if == 1, 0, 1)

# model evaluation
conf_matrix_if = confusion_matrix(y_new, y_pred_if)
class_report_if = classification_report(y_new, y_pred_if)

print("Isolation Forest Confusion Matrix:")
print(conf_matrix_if)
print("\nIsolation Forest Classification Report:")
print(class_report_if)
```

4.2 One-Class SVM:

One-Class SVM is implemented by having kernel 'rbf'. Nu value of '0.01' and gamma set to 'auto'

One-Class SVM

```
from sklearn.svm import OneClassSVM

# One-Class SVM model
one_class_svm = OneClassSVM(nu=0.01, kernel='rbf', gamma='auto')
one_class_svm.fit(X_scaled_new)

# Predict anomalies
y_pred_svm = one_class_svm.predict(X_scaled_new)
y_pred_svm = np.where(y_pred_svm == 1, 0, 1)

# Evaluate the model
conf_matrix_svm = confusion_matrix(y_new, y_pred_svm)
class_report_svm = classification_report(y_new, y_pred_svm)

print("One-Class SVM Confusion Matrix:")
print(conf_matrix_svm)
print("\nOne-Class SVM Classification Report:")
print(class_report_svm)
```

4.3 Local Outlier Factor:

LOF is implemented by setting contamination to ‘auto’ and the number of neighbours to ‘20’.

LOF

```
from sklearn.neighbors import LocalOutlierFactor

# Local Outlier Factor
lof = LocalOutlierFactor(n_neighbors=20, contamination='auto')
y_pred_lof = lof.fit_predict(X_scaled_new)
y_pred_lof = np.where(y_pred_lof == 1, 0, 1)

# Evaluate the model
conf_matrix_lof = confusion_matrix(y_new, y_pred_lof)
class_report_lof = classification_report(y_new, y_pred_lof)

print("Local Outlier Factor Confusion Matrix:")
print(conf_matrix_lof)
print("\nLocal Outlier Factor Classification Report:")
print(class_report_lof)
```

4.4 KNN

The data is split into train and test portioning with 80% of data is allocated to training the model. This data portioning is used for both KNN and random forest.

```
# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled_new, y_new, test_size=0.2, random_state=42)
```

KNN model is trained if the training data with number of k is assigned to ‘5’.

KNN

```
# Fit the KNN model
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)

y_pred_knn = knn.predict(X_test)

# Evaluate the model
conf_matrix_knn = confusion_matrix(y_test, y_pred_knn)
class_report_knn = classification_report(y_test, y_pred_knn)

print("K-Nearest Neighbors Confusion Matrix:")
print(conf_matrix_knn)
print("\nK-Nearest Neighbors Classification Report:")
print(class_report_knn)
```

4.5 Random Forest:

Random forest is trained on the training data by having decision trees number as '100' and the random state is set to '42'.

Random Forest

```
# Random Forest model
random_forest = RandomForestClassifier(n_estimators=100, random_state=42)
random_forest.fit(X_train, y_train)

# Predict on the test set
y_pred_rf = random_forest.predict(X_test)

# Evaluate the model
conf_matrix_rf = confusion_matrix(y_test, y_pred_rf)
class_report_rf = classification_report(y_test, y_pred_rf)

print("Random Forest Confusion Matrix:")
print(conf_matrix_rf)
print("\nRandom Forest Classification Report:")
print(class_report_rf)
```

5 Evaluation

All of the used machine learning models are evaluated using metrics like precision, accuracy, recall and F-1 score. Then the respective graphs of performance of each of these models are plotted using these metrics.

5.1 Isolation Forest:

```
# Evaluation
accuracy_if = accuracy_score(y_new, y_pred_if)
precision_if = precision_score(y_new, y_pred_if, zero_division=1)
recall_if = recall_score(y_new, y_pred_if)
f1_if = f1_score(y_new, y_pred_if)

print("Isolation Forest Performance:")
print(f"Accuracy: {accuracy_if:.2f}")
print(f"Precision: {precision_if:.2f}")
print(f"Recall: {recall_if:.2f}")
print(f"F1 Score: {f1_if:.2f}\n")

# For performance metrics for Isolation Forest
metrics_if = [accuracy_if, precision_if, recall_if, f1_if]
labels = ['Accuracy', 'Precision', 'Recall', 'F1 Score']
x = np.arange(len(labels))

fig, ax = plt.subplots(figsize=(10, 6))
bars = ax.bar(x, metrics_if, width=0.4)
ax.set_xticks(x)
ax.set_xticklabels(labels)
ax.set_ylim(0, 1)
ax.set_title('Isolation Forest Performance Metrics')
ax.set_ylabel('Scores')

# Adding bar labels
for bar in bars:
    height = bar.get_height()
    ax.annotate(f'{height:.2f}', xy=(bar.get_x() + bar.get_width() / 2, height),
                xytext=(0, 3), textcoords='offset points', ha='center', va='bottom')

plt.show()
```

5.2 One- Class SVM

```
# Evaluate One-Class SVM
accuracy_svm = accuracy_score(y_new, y_pred_svm)
precision_svm = precision_score(y_new, y_pred_svm, zero_division=1)
recall_svm = recall_score(y_new, y_pred_svm)
f1_svm = f1_score(y_new, y_pred_svm)

print("One-Class SVM Performance:")
print(f"Accuracy: {accuracy_svm:.2f}")
print(f"Precision: {precision_svm:.2f}")
print(f"Recall: {recall_svm:.2f}")
print(f"F1 Score: {f1_svm:.2f}\n")

# for performance metrics for One-Class SVM
metrics_svm = [accuracy_svm, precision_svm, recall_svm, f1_svm]
labels = ['Accuracy', 'Precision', 'Recall', 'F1 Score']
x = np.arange(len(labels))

fig, ax = plt.subplots(figsize=(10, 6))
bars = ax.bar(x, metrics_svm, width=0.4)
ax.set_xticks(x)
ax.set_xticklabels(labels)
ax.set_ylim(0, 1)
ax.set_title('One-Class SVM Performance Metrics')
ax.set_ylabel('Scores')

# Adding bar labels
for bar in bars:
    height = bar.get_height()
    ax.annotate(f'{height:.2f}', xy=(bar.get_x() + bar.get_width() / 2, height),
                xytext=(0, 3), textcoords='offset points', ha='center', va='bottom')
```

5.3 Local Outlier Factor

```
# Evaluation
accuracy_lof = accuracy_score(y_new, y_pred_lof)
precision_lof = precision_score(y_new, y_pred_lof, zero_division=1)
recall_lof = recall_score(y_new, y_pred_lof)
f1_lof = f1_score(y_new, y_pred_lof)

print("Local Outlier Factor Performance:")
print(f"Accuracy: {accuracy_lof:.2f}")
print(f"Precision: {precision_lof:.2f}")
print(f"Recall: {recall_lof:.2f}")
print(f"F1 Score: {f1_lof:.2f}\n")

# performance metrics for Local Outlier Factor
metrics_lof = [accuracy_lof, precision_lof, recall_lof, f1_lof]
labels = ['Accuracy', 'Precision', 'Recall', 'F1 Score']
x = np.arange(len(labels))

fig, ax = plt.subplots(figsize=(10, 6))
bars = ax.bar(x, metrics_lof, width=0.4)
ax.set_xticks(x)
ax.set_xticklabels(labels)
ax.set_ylim(0, 1)
ax.set_title('Local Outlier Factor Performance Metrics')
ax.set_ylabel('Scores')

# Add bar labels
for bar in bars:
    height = bar.get_height()
    ax.annotate(f'{height:.2f}', xy=(bar.get_x() + bar.get_width() / 2, height),
                xytext=(0, 3), textcoords='offset points', ha='center', va='bottom')

plt.show()
```

5.4 KNN

```
# Evaluate KNN
accuracy_knn = accuracy_score(y_test, y_pred_knn)
precision_knn = precision_score(y_test, y_pred_knn, zero_division=1)
recall_knn = recall_score(y_test, y_pred_knn)
f1_knn = f1_score(y_test, y_pred_knn)

print("K-Nearest Neighbors Performance:")
print(f"Accuracy: {accuracy_knn:.2f}")
print(f"Precision: {precision_knn:.2f}")
print(f"Recall: {recall_knn:.2f}")
print(f"F1 Score: {f1_knn:.2f}\n")

# performance metrics for K-Nearest Neighbors
metrics_knn = [accuracy_knn, precision_knn, recall_knn, f1_knn]
labels = ['Accuracy', 'Precision', 'Recall', 'F1 Score']
x = np.arange(len(labels))

fig, ax = plt.subplots(figsize=(10, 6))
bars = ax.bar(x, metrics_knn, width=0.4)
ax.set_xticks(x)
ax.set_xticklabels(labels)
ax.set_ylim(0, 1)
ax.set_ylabel('Scores')

# Adding bar labels
for bar in bars:
    height = bar.get_height()
    ax.annotate(f'{height:.2f}', xy=(bar.get_x() + bar.get_width() / 2, height),
                xytext=(0, 5), textcoords='offset points', ha='center', va='bottom')


plt.figtext(0.5, -0.1, 'K-Nearest Neighbors Performance Metrics', wrap=True, horizontalalignment='center', fontsize=12)
plt.tight_layout()
plt.show()
```

5.5 Random Forest

```
# Evaluate Random Forest
accuracy_rf = accuracy_score(y_test, y_pred_rf)
precision_rf = precision_score(y_test, y_pred_rf, zero_division=1)
recall_rf = recall_score(y_test, y_pred_rf)
f1_rf = f1_score(y_test, y_pred_rf)

print("Random Forest Performance:")
print(f"Accuracy: {accuracy_rf:.2f}")
print(f"Precision: {precision_rf:.2f}")
print(f"Recall: {recall_rf:.2f}")
print(f"F1 Score: {f1_rf:.2f}\n")

# performance metrics for Random Forest
metrics_rf = [accuracy_rf, precision_rf, recall_rf, f1_rf]
labels = ['Accuracy', 'Precision', 'Recall', 'F1 Score']
x = np.arange(len(labels))

fig, ax = plt.subplots(figsize=(10, 6))
bars = ax.bar(x, metrics_rf, width=0.4)
ax.set_xticks(x)
ax.set_xticklabels(labels)
ax.set_ylim(0, 1)
ax.set_ylabel('Scores')

# Adding bar labels
for bar in bars:
    height = bar.get_height()
    ax.annotate(f'{height:.2f}', xy=(bar.get_x() + bar.get_width() / 2, height),
                xytext=(0, 5), textcoords='offset points', ha='center', va='bottom')


plt.figtext(0.5, -0.1, 'Random Forest Performance Metrics', wrap=True, horizontalalignment='center', fontsize=12)
plt.tight_layout()
plt.show()
```

5.6 Cross validation

For verifying the models like KNN and Random Forest and not overfitting due to their perfect accuracy and precision, cross validation (Berrar, 2018) is performed on both Random Forest and KNN. The cross validation performed is k fold cross validation by using package 'StratifiedKFold'. The value of K is set to 5 for having a 5 fold cross validation.

Cross validation

```
import numpy as np
from sklearn.model_selection import cross_validate, StratifiedKFold
from sklearn.metrics import make_scorer, precision_score, recall_score, f1_score, roc_auc_score
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier

# cross-validation
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

#  scoring metrics
scoring = {
    'precision': make_scorer(precision_score),
    'recall': make_scorer(recall_score),
    'f1': make_scorer(f1_score),
    'roc_auc': make_scorer(roc_auc_score)
}

# performing cross-validation and printing results
def evaluate_model(model, X, y, cv, scoring):
    scores = cross_validate(model, X, y, cv=cv, scoring=scoring)
    print(f"Model: {model.__class__.__name__}")
    print(f"Average Precision: {np.mean(scores['test_precision']):.2f}")
    print(f"Average Recall: {np.mean(scores['test_recall']):.2f}")
    print(f"Average F1-Score: {np.mean(scores['test_f1']):.2f}")
    print(f"Average ROC AUC: {np.mean(scores['test_roc_auc']):.2f}")
    print("\n")

# Initialize both  the models
knn = KNeighborsClassifier(n_neighbors=5)
random_forest = RandomForestClassifier(n_estimators=100, random_state=42)

# Evaluate KNN with cross-validation
evaluate_model(knn, X_scaled_new, y_new, cv, scoring)

# Evaluate Random Forest with cross-validation
evaluate_model(random_forest, X_scaled_new, y_new, cv, scoring)
```

Result of Cross validation is:

```
Model: KNeighborsClassifier
Average Precision: 1.00
Average Recall: 0.97
Average F1-Score: 0.99
Average ROC AUC: 0.99


Model: RandomForestClassifier
Average Precision: 1.00
Average Recall: 0.97
Average F1-Score: 0.99
Average ROC AUC: 0.99
```

References

McKinney, W., 2011. *pandas: a foundational Python library for data analysis and statistics*. Python High Performance Science Computer. Available at: https://www.researchgate.net/publication/265194455_pandas_a_Foundational_Python_Library_for_Data_Analysis_and_Statistics.

Amorim, L.B.V., Cavalcanti, G.D.C., and Cruz, R.M.O., 2022. The choice of scaling technique matters for classification performance. Available at: https://arxiv.org/pdf/2212.12343.

Berrar, D., 2018. Cross-Validation. In: S. Ranganathan, D. Berrar and V. Gribskov, eds. 2018. Encyclopedia of Bioinformatics and Computational Biology. 1st ed. Oxford: Elsevier, pp.542-545. Available at: https://www.researchgate.net/publication/324701535_Cross-Validation.