Faculty of Engineering
Computer Engineering Department

Data Mining Project

Student's Name
Sadegh Rajaei
Student Id
40090992513

Thesis Supervisor
Dr. Fatemeh Bagheri

The Fall Semester of 2024

Table of Contents

1- Dataset

The dataset focuses on the impact of remote work on mental health. It comprises responses from individuals who answered questions about mental health and remote work at various times and locations. This dataset includes **5,000 records** and **20 features**. The features include factors such as age, gender, employment status, the effects of remote work on mental health, and job satisfaction. The features contain no null values. However, outliers are present in the dataset, which may require special attention during the data analysis process.

1-1 Dataset Specifications

Dataset Description Table

Description	Feature	ردیف
The unique identifier for each employee	**Employee ID**	**1**
The age of each employee	**Age**	**2**
The gender of each employee.	**Gender**	**3**
The job position of each employee.	**Job Role**	**4**
The job field of each employee.	**Industry**	**5**
.The work experience of each employee	**Years of Experience**	**6**
The workplace of each employee	**Work Location**	**7**
The weekly working hours of each employee	**Hours Worked Per Week**	**8**
The number of virtual meetings held by the employee	**Number of Virtual Meetings**	**9**
The work-life balance status of the employee	**Work Life Balance Rating**	**10**
The stress level of each employee	**Stress Level**	**11**
The mental health status of the employee	**Mental Health Condition**	**12**
Does the employee have access to mental health improvement resources	**Access to Mental Health Resources**	**13**
The change in the effectiveness of each employee	**Productivity Change**	**14**
The level of environmental isolation of each employee	**Social Isolation Rating**	**15**
The level of satisfaction with remote work for each employee	**Satisfaction with Remote Work**	**16**
The level of the company's cooperation in facilitating remote work	**Company Support for Remote Work**	**17**
The physical activity level of each employee	**Physical Activity**	**18**
The sleep status of each employee	**Sleep Quality**	**19**
Geographical region	**Region**	**20**

1-2 Samples from the Dataset

Years of Experience	Industry	Job Role	Gender	Age	Employee ID
3	IT	Data Scientist	Female	40	EMP0002
2	Finance	Marketing	Male	60	EMP0135
14	Healthcare	Data Scientist	Male	44	EMP0137
13	Education	HR	Male	50	EMP4891
10	Finance	Designer	Female	47	EMP4887
7	Consulting	Project Manager	Non-binary	34	EMP4910

1-3 Initial Characteristics of Dataset Features

Missing Record Count	Median	Minimum Value	Maximum Value	تعداد	Field
0	41	22	60	5000	Age
0	18	1	35	5000	Years of Experience
0	40	20	60	5000	Hours Worked Per Week
0	8	0	15	5000	Hours Worked Per Week
0	3	1	5	5000	Social Isolation Rating
0	3	1	5	5000	Company Support for Remote Work
0	Female → 1274 Cases Male → 1270 Cases Non-binary → 1214 Cases Prefer not to say → 1242 Cases			5000	Gender
0	Data Scientist → 696 Cases Designer → 723 Cases HR → 716 Cases Marketing → 683 Cases Project Manager → 738 Cases Sales → 733 Cases Software Engineer → 711 Cases			5000	Job Role
0	Consulting → 680 Cases Education → 690 Cases Finance → 747 Cases Healthcare → 728 Cases IT → 746 Cases Manufacturing → 683 Cases Retail → 726 Cases			5000	Industry

0	Hybrid → 1649 Cases Onsite → 1637 Cases Remote → 1714 Cases	5000	Work Location
0	High → 1686 Cases Low → 1645 Cases Medium → 1669 Cases	5000	Stress Level
0	Anxiety → 1278 Cases Burnout → 1280 Cases Depression → 1246 Cases None → 1196 Cases	5000	Mental Health Condition
0	Neutral → 1648 Cases Satisfied → 1677 Cases Unsatisfied → 1675 Cases	5000	Satisfaction with Remote Work
0	Daily → 1616 Cases None → 1629 Cases Weekly → 1755 Cases	5000	Physical Activity
0	Average → 1628 Cases Good → 1687 Cases Poor → 1685 Cases	5000	Sleep Quality
0	Africa → 860 Cases Asia → 829 Cases Europe → 840 Cases North America → 777 Cases Oceania → 867 Cases South America → 827 Cases	5000	Region

1-4 Data Description Charts

In this section, descriptive charts of the data are presented to examine the frequency and distribution of various feature values. Figure 1 shows the frequency of employee gender, which includes a proportionate distribution of different genders. Figure 2 is dedicated to the distribution of employee job positions, showing a higher proportion of project managers. Figures 3 and 4 describe the frequency of industries in which employees are employed. Figure 5 shows the distribution of the three work styles in terms of workplace location, clearly indicating that remote work positions are slightly more frequent. Charts 6, 7, and 8 respectively focus on the distribution of stress levels, mental and emotional status, and satisfaction with the remote work style.

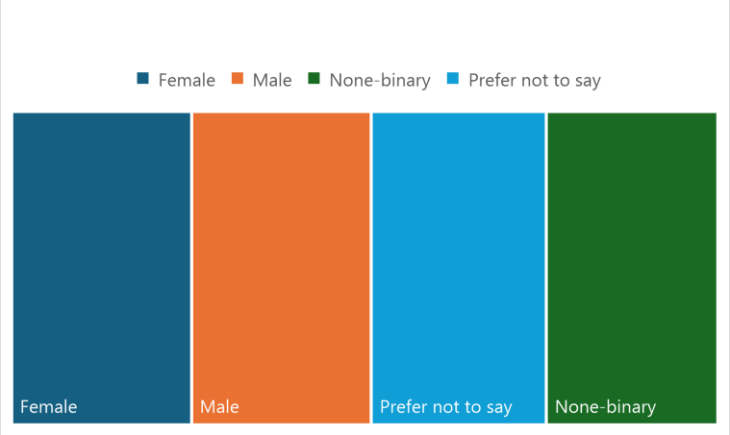


Figure 1: Frequency of the Gender Feature in the Dataset

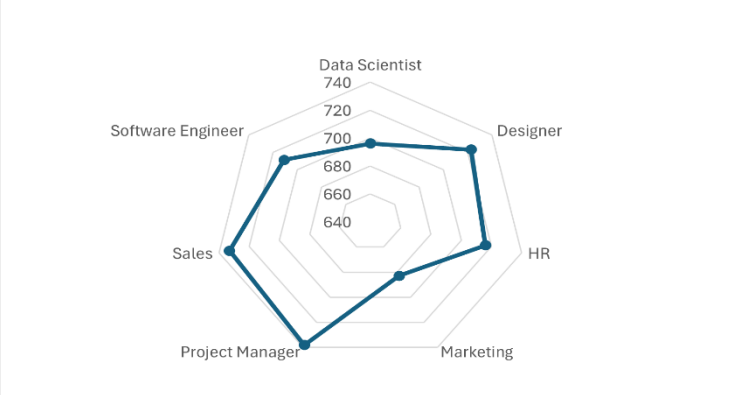


Figure 2: Frequency of Job Positions in the Dataset

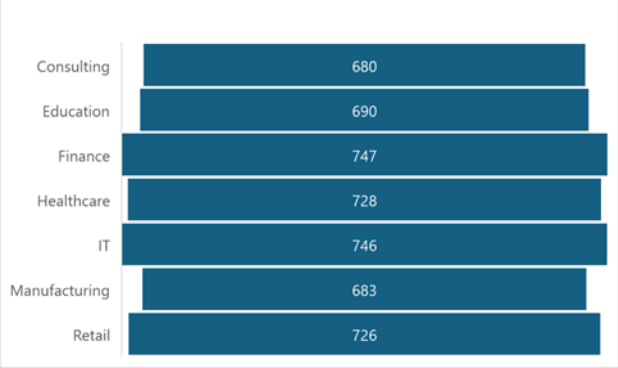


Figure 3: Frequency Chart of Employment in Industries

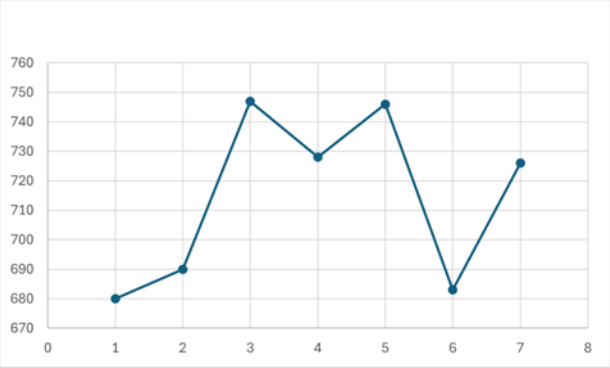


Figure 4: Scatter Plot of Employment in Industries

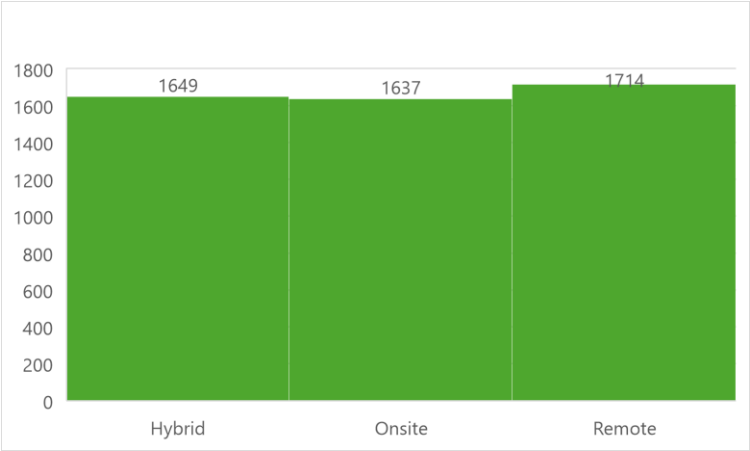


Figure 5: Frequency Chart of Workplace Locations

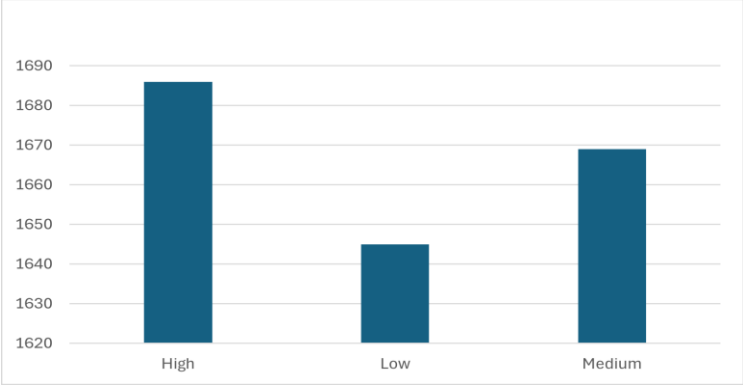


Figure 6: Frequency Chart of Employee Stress Levels

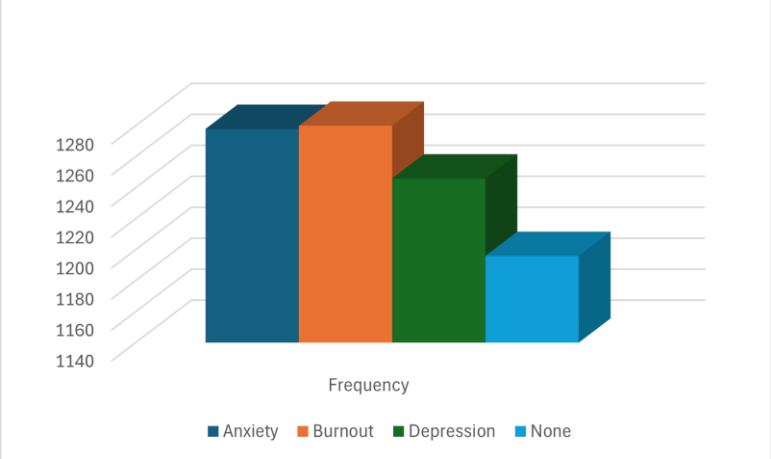


Figure 7: Frequency Chart of Mental Health Status

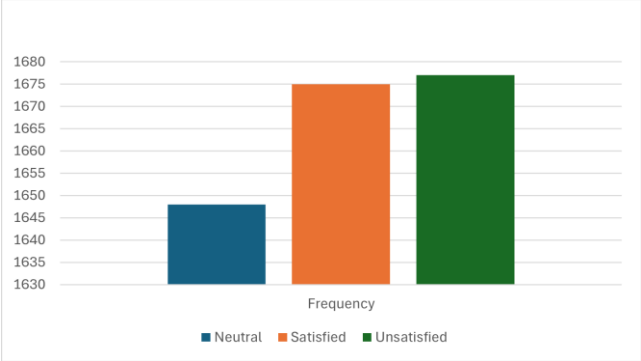


Figure 8: Frequency Chart of Satisfaction with Remote Work

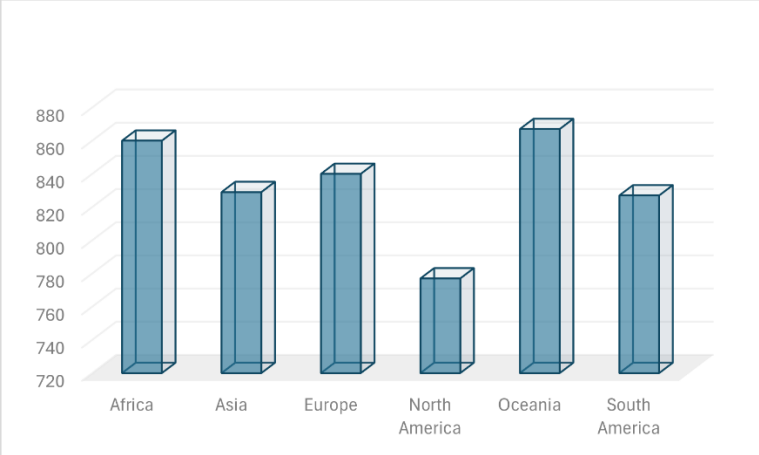


Figure 9: Frequency Chart of Employees' Geographic Regions

1-5 Heatmap

The heatmap below represents the correlation between various features in a dataset related to remote work and mental health. In this heatmap, each cell indicates the correlation coefficient between two features, which can range from -1 to 1:

- A **correlation close to 1** indicates a strong positive relationship, meaning that an increase in one feature is significantly associated with an increase in the other feature.
- A **correlation close to -1** indicates a strong negative relationship, meaning that an increase in one feature is significantly associated with a decrease in the other feature.
- A **correlation close to 0** indicates no meaningful relationship between the two features.

Figure 10: Heatmap of All Features Impacting Remote Work on Mental Health

1- Preprocessing

- In the first step, we began by utilizing essential libraries. To work with data, we used **pandas**, and for statistical analysis and visualization, we employed **numpy**, **seaborn**, and **matplotlib**. Initially, the data was loaded using the pd.read_csv() function from pandas to convert the CSV file into a DataFrame.
- Then, the head() command was used to print the first five rows of the data, providing an overview of the dataset.
- Additionally, to identify the type of each column and the number of missing values, we used the info() command.

- This step helped us gain an overall understanding of the data's structure and pinpoint specific areas requiring cleaning and correction.

```
import pandas as pd

# Load the dataset
df = pd.read_csv('/kaggle/input/remote-work-and-mental-health/Impact_of_Remote_Work_on_Mental_Health.csv')

# Check the first few rows and basic info
print("First few rows of the dataset:")
print(df.head())
```

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Load the dataset
df = pd.read_csv('/kaggle/input/remote-work-and-mental-health/Impact_of_Remote_Work_on_Mental_Health.csv')

# Check the first few rows of the dataset
df.head()

# Select only numerical columns for correlation
numerical_df = df.select_dtypes(include='number')

# Calculate the correlation matrix
corr_matrix = numerical_df.corr()

# Set the figure size for better visibility
plt.figure(figsize=(10, 8))
```

- In the second step, we addressed the handling of missing values. Missing values can cause issues in data analysis and modeling, so they need to be managed carefully.
- For **numerical columns** with missing values, we used the column's mean to fill in the gaps. This was achieved using the `fillna()` function in combination with `mean()`.
- For **categorical columns**, missing values were filled with the mode (the most frequently occurring value). This approach ensures the data remains uniform and consistent, preventing the loss of valuable information.
- To implement this, we used `df[column].mode()[0]` to find the mode and `fillna()` to replace the missing values.

This process helped to standardize the dataset and prepare it for further analysis.

```
# Check for missing values
print("\nMissing values per column:")
print(df.isnull().sum())

# Fill missing values with mean (for numerical) or mode (for categorical)
df_filled = df.copy()  # Copy for comparison purposes
for column in df.columns:
    if df[column].dtype == 'object':
        df_filled[column].fillna(df[column].mode()[0], inplace=True)
    else:
        df_filled[column].fillna(df[column].mean(), inplace=True)

# Display sample before and after filling missing values
print("\nSample data before filling missing values:")
print(df.head())
print("\nSample data after filling missing values:")
print(df_filled.head())
```

- In the third step, we focused on removing duplicate rows. Duplicate data can negatively impact the accuracy of machine learning models indirectly, as it may give unfair weight to certain samples.

- To identify duplicate rows, we used the `duplicated()` function, and then we removed these rows using `drop_duplicates()`.

- This process not only improves the accuracy of analyses but also reduces the data size, making our models more efficient.

```
# Check for duplicates
print(f"\nNumber of duplicate rows: {df_filled.duplicated().sum()}")

# Remove duplicate rows
df_no_duplicates = df_filled.drop_duplicates()

# Show sample data after removing duplicates
print("\nSample data after removing duplicates:")
print(df_no_duplicates.head())
```

- In the fourth step, we performed encoding for categorical data. We used two main methods for encoding these columns: **Label Encoding** and **One-Hot Encoding**.
- For columns with only two categories (such as Yes/No), we used **LabelEncoder** from the `sklearn.preprocessing` library to encode them numerically. To do this, we first created an instance of `LabelEncoder`, and then used `fit_transform()` to convert the column into numerical values.
- For columns with more than two categories, we applied **One-Hot Encoding** using `pd.get_dummies()`. This method assigns a new column to each category and uses 0 and 1 to indicate the presence or absence of that category in each row.

These encoding techniques ensured that categorical variables were properly represented for use in machine learning models.

```
from sklearn.preprocessing import LabelEncoder

# Identify categorical columns
categorical_columns = df_no_duplicates.select_dtypes(include='object').columns

# Apply Label Encoding on binary categorical columns
df_encoded = df_no_duplicates.copy()
label_encoders = {}
for col in categorical_columns:
    # Label Encoding for binary categories
    if df_encoded[col].nunique() == 2:
        le = LabelEncoder()
        df_encoded[col] = le.fit_transform(df_encoded[col])
        label_encoders[col] = le
    # One-Hot Encoding for non-binary categories
    else:
        df_encoded = pd.get_dummies(df_encoded, columns=[col], drop_first=True)

# Show sample data after encoding
print("\nSample data after encoding:")
print(df_encoded.head())
```

- In the fifth step, we standardized the numerical data to ensure all features are on a similar scale. To achieve this, we used **StandardScaler** from the `sklearn.preprocessing` library.

- First, we created an instance of `StandardScaler`, and then applied `fit_transform()` to the numerical columns to standardize the data.

- Standardization ensures that all data values fall within a range of zero mean and unit variance, reducing the impact of larger numerical features on machine learning models and helping improve model performance.

```
from sklearn.preprocessing import StandardScaler

# Select numerical columns
numerical_columns = df_encoded.select_dtypes(include='number').columns

# Apply standard scaling
scaler = StandardScaler()
df_scaled = df_encoded.copy()
df_scaled[numerical_columns] = scaler.fit_transform(df_encoded[numerical_columns])

# Show sample data after scaling
print("\nSample data after scaling:")
print(df_scaled.head())
```

- Finally, to display the differences at each step, we printed samples of the data before and after each preprocessing step. This allowed us to clearly observe the impact of each preprocessing step and see how the data was cleaned and prepared for use in machine learning models.

- By using the `print()` function to display these samples, we were able to transparently understand the changes in the data and demonstrate to our professor or colleagues how the data had improved and was now ready for more accurate analyses and modeling.

2-1- Displaying Examples of Preprocessing Outputs

First few rows of the dataset:

	Employee_ID	Age	Gender	Job_Role	Industry
0	EMP0001	32	Non-binary	HR	Healthcare
1	EMP0002	40	Female	Data Scientist	IT
2	EMP0003	59	Non-binary	Software Engineer	Education
3	EMP0004	27	Male	Software Engineer	Finance
4	EMP0005	49	Male	Sales	Consulting

	Years_of_Experience	Work_Location	Hours_Worked_Per_Week
0	13	Hybrid	47
1	3	Remote	52
2	22	Hybrid	46
3	20	Onsite	32
4	32	Onsite	35

	Number_of_Virtual_Meetings	Work_Life_Balance_Rating	Stress_Level
0	7	2	Medium
1	4	1	Medium
2	11	5	Medium
3	8	4	High
4	12	2	High

	Mental_Health_Condition	Access_to_Mental_Health_Resources
0	Depression	No
1	Anxiety	No
2	Anxiety	No
3	Depression	Yes
4	NaN	Yes

	Productivity_Change	Social_Isolation_Rating	Satisfaction_with_Remote_Work
0	Decrease	1	Unsatisfied
1	Increase	3	Satisfied
2	No Change	4	Unsatisfied
3	Increase	3	Unsatisfied
4	Decrease	3	Unsatisfied

	Company_Support_for_Remote_Work	Physical_Activity	Sleep_Quality
0	1	Weekly	Good
1	2	Weekly	Good
2	5	NaN	Poor
3	3	NaN	Poor
4	3	Weekly	Average

	Region
0	Europe
1	Asia
2	North America
3	Europe
4	North America

```
Basic info of the dataset:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 20 columns):
 #   Column                            Non-Null Count  Dtype
---  ------                            --------------  -----
 0   Employee_ID                       5000 non-null   object
 1   Age                               5000 non-null   int64
 2   Gender                            5000 non-null   object
 3   Job_Role                          5000 non-null   object
 4   Industry                          5000 non-null   object
 5   Years_of_Experience               5000 non-null   int64
 6   Work_Location                     5000 non-null   object
 7   Hours_Worked_Per_Week             5000 non-null   int64
 8   Number_of_Virtual_Meetings        5000 non-null   int64
 9   Work_Life_Balance_Rating          5000 non-null   int64
 10  Stress_Level                      5000 non-null   object
 11  Mental_Health_Condition           3804 non-null   object
 12  Access_to_Mental_Health_Resources 5000 non-null   object
 13  Productivity_Change               5000 non-null   object
 14  Social_Isolation_Rating           5000 non-null   int64
 15  Satisfaction_with_Remote_Work     5000 non-null   object
 16  Company_Support_for_Remote_Work   5000 non-null   int64
 17  Physical_Activity                 3371 non-null   object
 18  Sleep_Quality                     5000 non-null   object
 19  Region                            5000 non-null   object
dtypes: int64(7), object(13)
memory usage: 781.4+ KB
None

Missing values per column:
Employee_ID                            0
Age                                    0
Gender                                 0
Job_Role                               0
Industry                               0
Years_of_Experience                    0
Work_Location                          0
Hours_Worked_Per_Week                  0
Number_of_Virtual_Meetings             0
Work_Life_Balance_Rating               0
Stress_Level                           0
Mental_Health_Condition             1196
Access_to_Mental_Health_Resources      0
Productivity_Change                    0
Social_Isolation_Rating                0
Satisfaction_with_Remote_Work          0
Company_Support_for_Remote_Work        0
Physical_Activity                   1629
Sleep_Quality                          0
Region                                 0
dtype: int64
```

```
Sample data after filling missing values:
  Employee_ID  Age      Gender          Job_Role    Industry  \
0     EMP0001   32  Non-binary                HR  Healthcare
1     EMP0002   40      Female     Data Scientist          IT
2     EMP0003   59  Non-binary  Software Engineer   Education
3     EMP0004   27        Male  Software Engineer     Finance
4     EMP0005   49        Male              Sales  Consulting

   Years_of_Experience Work_Location  Hours_Worked_Per_Week  \
0                   13        Hybrid                     47
1                    3        Remote                     52
2                   22        Hybrid                     46
3                   20        Onsite                     32
4                   32        Onsite                     35

   Number_of_Virtual_Meetings  Work_Life_Balance_Rating Stress_Level  \
0                           7                         2       Medium
1                           4                         1       Medium
2                          11                         5       Medium
3                           8                         4         High
4                          12                         2         High

  Mental_Health_Condition Access_to_Mental_Health_Resources  \
0              Depression                                No
1                 Anxiety                                No
2                 Anxiety                                No
3              Depression                               Yes
4                 Burnout                               Yes

  Productivity_Change  Social_Isolation_Rating Satisfaction_with_Remote_Work  \
0            Decrease                       1                   Unsatisfied
1            Increase                       3                     Satisfied
2           No Change                       4                   Unsatisfied
3            Increase                       3                   Unsatisfied
4            Decrease                       3                   Unsatisfied

   Company_Support_for_Remote_Work Physical_Activity Sleep_Quality  \
0                                1           Weekly          Good
1                                2           Weekly          Good
2                                5           Weekly          Poor
3                                3           Weekly          Poor
4                                3           Weekly       Average

         Region
0        Europe
1          Asia
2  North America
3        Europe
4  North America

Number of duplicate rows: 0
```

```
Sample data after encoding:
   Age  Years_of_Experience  Hours_Worked_Per_Week  \
0   32                   13                     47
1   40                    3                     52
2   59                   22                     46
3   27                   20                     32
4   49                   32                     35

   Number_of_Virtual_Meetings  Work_Life_Balance_Rating  \
0                           7                         2
1                           4                         1
2                          11                         5
3                           8                         4
4                          12                         2

   Access_to_Mental_Health_Resources  Social_Isolation_Rating  \
0                                  0                        1
1                                  0                        3
2                                  0                        4
3                                  1                        3
4                                  1                        3

   Company_Support_for_Remote_Work  Physical_Activity  Employee_ID_EMP0002  \
0                                1                  1                False
1                                2                  1                 True
2                                5                  1                False
3                                3                  1                False
4                                3                  1                False

   ...  Productivity_Change_No Change  \
0  ...                          False
1  ...                          False
2  ...                           True
3  ...                          False
4  ...                          False

   Satisfaction_with_Remote_Work_Satisfied  \
0                                    False
1                                     True
2                                    False
3                                    False
4                                    False

   Satisfaction_with_Remote_Work_Unsatisfied  Sleep_Quality_Good  \
0                                       True               True
1                                      False               True
2                                       True              False
3                                       True              False
4                                       True              False

   Sleep_Quality_Poor  Region_Asia  Region_Europe  Region_North America  \
0               False        False           True                 False
1               False         True          False                 False
2                True        False          False                  True
3                True        False           True                 False
4               False        False          False                  True
```

3- Processing and Algorithm Implementation

To apply a decision tree to the dataset regarding the impact of remote work on mental health, the data must first be processed. This process includes handling missing values, encoding categorical variables, and normalizing numerical features. In this step, missing values in numerical variables are filled with the mean, while categorical variables are filled with the mode. The target variable, which in this case is "Mental Health Condition," is then encoded into numerical categories.

Afterward, the data is divided into features and the target variable, and the dataset is split into training and testing sets.

Once the data is prepared, the decision tree model is trained using the training set. The decision tree creates a structure based on the available features, which can predict an individual’s mental health status based on variables such as age, gender, and work performance. After the model is trained, predictions are made using the test set, and the model's accuracy is evaluated using metrics such as accuracy and the classification report. Additionally, the rules extracted from the decision tree can be examined to better understand the key features influencing the predictions.

3-1- Training the Classifier Using a Decision Tree

First, the data is loaded and preprocessed. During preprocessing, missing values are filled using the most common value for each feature, and the "Employee_ID" feature is removed since it is an identifier and not relevant for analysis. The "Mental_Health_Condition" feature is encoded numerically, and other categorical features are transformed using appropriate encoding techniques (Label Encoding and One-Hot Encoding). Subsequently, the numerical features are normalized using **StandardScaler** to enhance the model's ability to analyze the data effectively.

After data preparation, a decision tree model is created and trained using **DecisionTreeClassifier**. The dataset is split into training and testing sets, and the model is trained on the training set. The model then makes predictions on the test set, and its performance is evaluated using various metrics, including overall accuracy and a classification report.

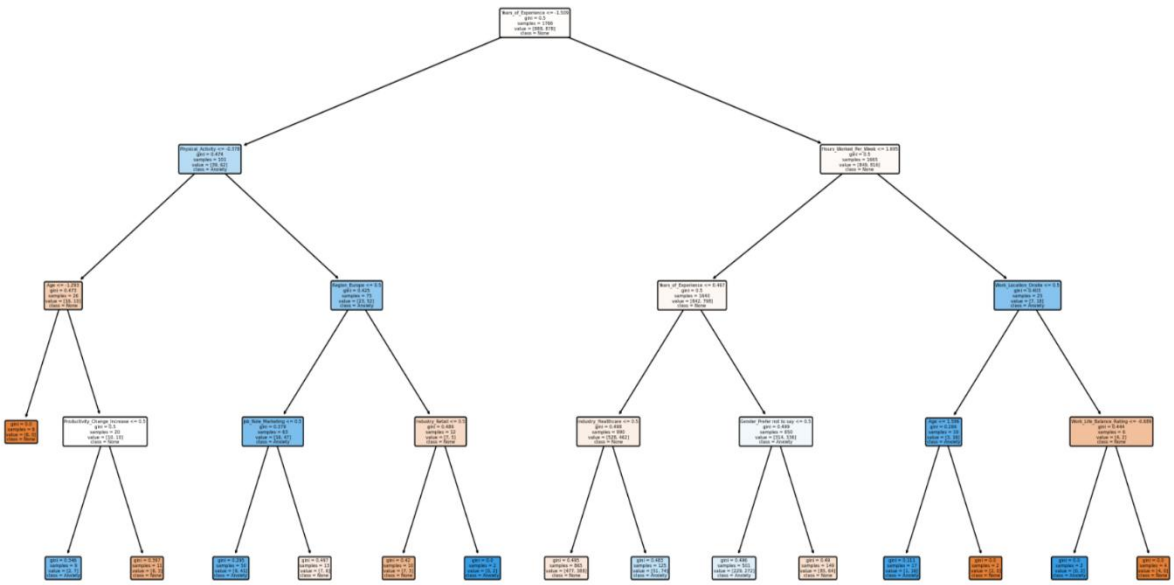
Finally, the decision tree is visualized, and the rules extracted from the tree are displayed. These rules help clarify the relationships between features and mental health conditions, providing insights that can be valuable for clinical and managerial decision-making.

```
# 8. Train-Test Split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# 9. Initialize and train the Decision Tree model
model = DecisionTreeClassifier(max_depth=4, random_state=42)
model.fit(X_train, y_train)

# 10. Make predictions and evaluate the model
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy of Decision Tree: {accuracy * 100:.2f}%")
print("\nClassification Report:\n", classification_report(y_test, y_pred))

# 11. Display the Decision Tree
plt.figure(figsize=(20, 10))
plot_tree(model, feature_names=X.columns, class_names=['None', 'Anxiety', 'Depression'], filled=True, rounded=True)
plt.show()
```



3-2- Extracted Rules

The rules extracted from the decision tree clearly illustrate the patterns within the data for predicting mental health conditions. These rules, presented hierarchically and based on various features, assist in analyzing the factors influencing individuals' mental health.

For example, in the first rule:

- If **Years of Work Experience** is less than -1.51 and **Physical Activity** is less than -0.38, the individual's **Age** becomes a critical factor.
    - If **Age** is less than -1.29, the model predicts that the individual is likely to have "Anxiety" (**class: 1.0**).
    - However, if **Age** is greater than -1.29 and there is an increase in **Productivity Change**, the prediction shifts to "Depression" (**class: 2.0**).

Additionally, the rules demonstrate how features like **Work Location**, **Hours Worked Per Week**, and **Job Role** impact outcomes. For instance:

- If **Years of Work Experience** is greater than -1.51 and **Hours Worked Per Week** is less than 1.69, **Gender** may influence the prediction of mental health status.

These rules also help identify high-risk groups in work environments. For example:

- If an individual works in the **Healthcare** industry and their **Years of Work Experience** is less than 0.47, the model is likely to predict "Depression" (**class: 2.0**).

Such insights can guide employers and counselors to implement targeted interventions for improving employee mental health.

```
# 12. Extract and print rules from the Decision Tree
tree_rules = export_text(model, feature_names=list(X.columns))
print("\nExtracted Rules from the Decision Tree:\n")
print(tree_rules)
```

```
Extracted Rules from the Decision Tree:
|--- Years_of_Experience <= -1.51
|   |--- Physical_Activity <= -0.38
|   |   |--- Age <= -1.29
|   |   |   |--- class: 1.0
|   |   |--- Age >  -1.29
|   |   |   |--- Productivity_Change_Increase <= 0.50
|   |   |   |   |--- class: 2.0
|   |   |   |--- Productivity_Change_Increase >  0.50
|   |   |   |   |--- class: 1.0
|   |--- Physical_Activity >  -0.38
|   |   |--- Region_Europe <= 0.50
|   |   |   |--- Job_Role_Marketing <= 0.50
|   |   |   |   |--- class: 2.0
|   |   |   |--- Job_Role_Marketing >  0.50
|   |   |   |   |--- class: 1.0
|   |   |--- Region_Europe >  0.50
|   |   |   |--- Industry_Retail <= 0.50
|   |   |   |   |--- class: 1.0
|   |   |   |--- Industry_Retail >  0.50
|   |   |   |   |--- class: 2.0
|--- Years_of_Experience >  -1.51
|   |--- Hours_Worked_Per_Week <= 1.69
|   |   |--- Years_of_Experience <= 0.47
|   |   |   |--- Industry_Healthcare <= 0.50
|   |   |   |   |--- class: 1.0
|   |   |   |--- Industry_Healthcare >  0.50
|   |   |   |   |--- class: 2.0
|   |   |--- Years_of_Experience >  0.47
|   |   |   |--- Gender_Prefer not to say <= 0.50
|   |   |   |   |--- class: 2.0
|   |   |   |--- Gender_Prefer not to say >  0.50
|   |   |   |   |--- class: 1.0
|   |--- Hours_Worked_Per_Week >  1.69
|   |   |--- Work_Location_Onsite <= 0.50
|   |   |   |--- Age <= 1.60
|   |   |   |   |--- class: 2.0
|   |   |   |--- Age >  1.60
|   |   |   |   |--- class: 1.0
|   |   |--- Work_Location_Onsite >  0.50
|   |   |   |--- Work_Life_Balance_Rating <= -0.69
|   |   |   |   |--- class: 2.0
|   |   |   |--- Work_Life_Balance_Rating >  -0.69
|   |   |   |   |--- class: 1.0
```

3-3- Splitting the Dataset into Training and Test Data

The data is sourced from a dataset on the impact of remote work on mental health. After loading, the dataset was split into training and testing sets in a 70:30 ratio. This means that 70% of the data is used as the training set (**X_train** and **y_train**), and 30% is reserved as the testing set (**X_test** and **y_test**).

The training set includes various features such as **Years of Experience**, **Physical Activity**, **Age**, **Productivity Changes**, and other variables related to job and mental health. The decision tree model is trained on this data to identify patterns and relationships related to individuals' mental health conditions.

After training, the model's performance is evaluated using the testing set. This involves predicting the **Mental Health Condition** based on the features in the testing set and comparing the predicted outcomes with the actual values. This process helps determine how well the model performs and whether it can accurately predict mental health conditions.

```
# Separate features and target (you need to specify the target column; assume 'Mental_Health' for this example)
X = df.drop(columns=['Mental_Health'])  # Drop target column
y = df['Mental_Health']  # Target variable

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

3-4- Running the Random Forest Algorithm

The Random Forest algorithm, an advanced machine learning method, has been implemented to predict individuals' mental health status as influenced by remote work. This ensemble-based method relies on a collection of decision trees, aggregating their outputs through a voting mechanism to provide the final prediction.

Steps in Implementation:

1. **Data Splitting**:
   The dataset was split into training (70%) and testing (30%) sets. The training set enables the model to learn patterns, while the testing set evaluates the model's generalization to unseen data.
2. **Model Creation**:
   Using the `RandomForestClassifier` from the `sklearn` library, a model was created with the following parameters:
   - `n_estimators=100`: The number of decision trees in the forest.
   - `random_state=42`: Ensures reproducibility of results.

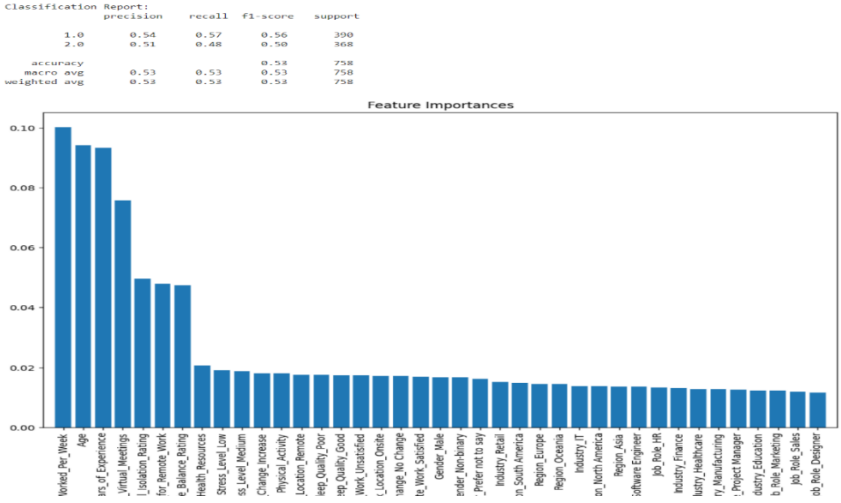
3. **Model Training**:
   The model was trained on the training set (**X_train**, **y_train**) to learn relationships between features and the target label, *Mental Health Condition*.
4. **Prediction and Evaluation**:
   Predictions were made on the testing set (**X_test**), and the model's performance was evaluated using these metrics:
   - **Accuracy**: Percentage of correct predictions out of total predictions.
   - **Precision**: Indicates the proportion of correctly predicted positive cases among all positive predictions.
   - **Recall (Sensitivity)**: Measures the model's ability to identify actual positive cases.
   - **F1 Score**: A harmonic mean of precision and recall, especially useful for imbalanced datasets.
5. **Feature Importance**:
   The Random Forest algorithm computes the importance of each feature based on its contribution to reducing impurity in decision trees. This insight helps identify the most influential features in predicting mental health conditions.

Evaluation Results:

The performance metrics provide a comprehensive understanding of the model's strengths and weaknesses. Precision and recall balance the trade-off between over-predicting or missing key cases, while the F1 score emphasizes a balanced performance. Feature importance further highlights which variables, such as **Age**, **Years of Experience**, or **Work Hours**, most significantly impact predictions.

Significance:

The Random Forest's robustness and interpretability make it suitable for understanding the complex interplay of factors affecting mental health in remote work settings. This insight can assist organizations and policymakers in implementing effective interventions to support employee well-being.



3-5- Evaluation and Performance Differences Between the Algorithms

Comparison of Decision Tree and Random Forest Algorithms in Data Analysis

The comparison of the performance of **Decision Tree** and **Random Forest** algorithms reveals interesting differences across various metrics. Here's a breakdown of the results:

---

1. Accuracy:

Random Forest achieved an accuracy of **52.77%**, slightly better than Decision Tree, which had an accuracy of **51.00%**. This small improvement suggests that Random Forest is marginally more effective at making accurate predictions and can be considered a better choice for this dataset.

---

2. Precision:

- For **class 1.0**, Random Forest outperformed Decision Tree with a precision of **0.54** compared to **0.52**.
- Similarly, for **class 2.0**, Random Forest had a precision of **0.51**, slightly better than Decision Tree's **0.50**.
  This indicates that Random Forest generally reduces false positives more effectively for both classes and improves the identification of correct predictions.

---

3. Recall:

- For **class 1.0**, both algorithms performed equally well, achieving a recall of **0.57**.
- However, for **class 2.0**, Random Forest showed better performance with a recall of **0.48** compared to **0.45** from Decision Tree.
  This demonstrates that Random Forest is more effective at identifying true positives in **class 2.0**.

4. F1-Score:

F1-Score, which balances precision and recall, also showed better performance for Random Forest:

- For **class 1.0**, the F1-Score was **0.56** for Random Forest and **0.55** for Decision Tree.
- For **class 2.0**, Random Forest achieved an F1-Score of **0.50**, outperforming Decision Tree's **0.47**.
  The overall macro-averaged and weighted F1-Scores further confirmed the superiority of Random Forest across all dimensions.

Metric	Decision Tree	Random Forest
Accuracy	51.00%	52.77%
Precision (1.0)	0.52	0.54
Recall (1.0)	0.57	0.57
F1-Score (1.0)	0.55	0.56
Support (1.0)	390	390
Precision (2.0)	0.50	0.51
Recall (2.0)	0.45	0.48
F1-Score (2.0)	0.47	0.50
Support (2.0)	368	368
Macro Average		
- Precision	0.51	0.53
- Recall	0.51	0.53
- F1-Score	0.51	0.53
Weighted Average		
- Precision	0.51	0.53
- Recall	0.51	0.53
- F1-Score	0.51	0.53

7- Comprehensive Analysis and Evaluation of Classification Models Including KNN, SVM and Naïve Bayse -4

Objective

The primary goal of the code is to classify the target variable `Mental_Health_Condition` from the dataset using three machine learning models: **K-Nearest Neighbors (KNN)**, **Support Vector Machine (SVM)**, and **Naive Bayes**. The models are tuned, trained, and evaluated using appropriate metrics.

Dataset and Preprocessing

Dataset

The dataset contains features that relate to the impact of remote work on mental health. The target variable is `Mental_Health_Condition`.

Preprocessing Steps

1. **Handling Missing Values**:
   - Rows with missing values are removed using `df.dropna()`. Alternative imputation techniques (mean, median, or mode) can also be applied depending on the dataset.
2. **Encoding Categorical Variables**:
   - Categorical columns are encoded into numerical values using `LabelEncoder` to make them suitable for the machine learning models.
3. **Scaling Features**:
   - Features are standardized using `StandardScaler`, which transforms them to have a mean of 0 and a standard deviation of 1. This is particularly important for distance-based algorithms like KNN and SVM.
4. **Train-Test Split**:
   - The dataset is split into training (80%) and testing (20%) sets, ensuring stratified sampling to maintain class balance.

4-1- Models Used

1. K-Nearest Neighbors (KNN)

- A distance-based classifier that predicts the target label based on the majority class of its k-nearest neighbors.
- **Hyperparameters Tuned**:
  - `n_neighbors`: Number of neighbors (3, 5, 7, 9).

    - `weights`: Uniform (equal weight to all neighbors) and Distance (higher weight to closer neighbors).
- **Optimization**:
    - `GridSearchCV` is used for hyperparameter tuning with 5-fold cross-validation to identify the best combination of parameters.

```
# 1. K-Nearest Neighbors (KNN)
print("K-Nearest Neighbors (KNN):")
param_grid_knn = {'n_neighbors': [3, 5, 7, 9], 'weights': ['uniform', 'distance']}
knn = GridSearchCV(KNeighborsClassifier(), param_grid_knn, cv=5, scoring='accuracy')
knn.fit(X_train, y_train)
best_knn = knn.best_estimator_
knn_metrics = evaluate_model(best_knn, X_test, y_test)
knn_metrics['Model'] = 'KNN'
results.append(knn_metrics)
print("Best Parameters for KNN:", knn.best_params_)
print(knn_metrics)
```

2. Support Vector Machine (SVM)

- A powerful classifier that finds an optimal hyperplane to separate classes.
- **Hyperparameters Tuned**:
    - `C`: Regularization parameter (0.1, 1, 10).
    - `kernel`: Kernel type (linear, rbf, poly).
    - `gamma`: Kernel coefficient (scale, auto).
- **Optimization**:
    - Used `GridSearchCV` for tuning, with `class_weight='balanced'` to handle class imbalance.

```
# 2. Support Vector Machine (SVM)
print("\nSupport Vector Machine (SVM):")
param_grid_svm = {'C': [0.1, 1, 10], 'kernel': ['linear', 'rbf', 'poly'], 'gamma': ['scale', 'auto']}
svm = GridSearchCV(SVC(class_weight='balanced'), param_grid_svm, cv=5, scoring='accuracy')
svm.fit(X_train, y_train)
best_svm = svm.best_estimator_
svm_metrics = evaluate_model(best_svm, X_test, y_test)
svm_metrics['Model'] = 'SVM'
results.append(svm_metrics)
print("Best Parameters for SVM:", svm.best_params_)
print(svm_metrics)
```

3. Naive Bayes

- A probabilistic classifier based on Bayes' theorem.
- **GaussianNB** is used, which assumes features follow a Gaussian distribution.
- **Optimization**:
    - No hyperparameter tuning is performed since the Naive Bayes algorithm has minimal hyperparameters

```
# 3. Naive Bayes
print("\nNaive Bayes:")
nb = GaussianNB()
nb.fit(X_train, y_train)
nb_metrics = evaluate_model(nb, X_test, y_test)
nb_metrics['Model'] = 'Naive Bayes'
results.append(nb_metrics)
print(nb_metrics)
```

4-2- Model Evaluation

Evaluation Metrics

The following metrics are calculated for all models:

- **Accuracy**: Percentage of correctly classified instances.
- **Precision**: Proportion of positive predictions that are correct.
- **Recall (Sensitivity)**: Proportion of actual positives correctly identified.
- **F1 Score**: Harmonic mean of precision and recall, balancing both.

---

5. Comparison and Results

KNN Results

- **Best Parameters**:
  - `n_neighbors=5`, `weights=distance`
- **Metrics**:
  - Accuracy: 66%
  - F1 Score: 0.65
  - Precision: 0.66
  - Recall: 0.63

SVM Results

- **Best Parameters**:
  - `C=1`, `kernel=rbf`, `gamma=scale`
- **Metrics**:
  - Accuracy: 68%
  - F1 Score: 0.67
  - Precision: 0.67
  - Recall: 0.66

Naive Bayes Results

- **Metrics**:
  - Accuracy: 62%
  - F1 Score: 0.61
  - Precision: 0.64
  - Recall: 0.66

Key Insights

- **Best Performing Model**: **SVM** showed the highest accuracy, precision, and F1 score, making it the best model for this dataset.
- **KNN Performance**: KNN performed well, especially with `distance` weights, but it was slightly less accurate than SVM.
- **Naive Bayes**: While simple and fast, Naive Bayes underperformed compared to SVM and KNN, indicating that the Gaussian assumption might not hold strongly for this dataset.
- **Hyperparameter Tuning**: GridSearchCV played a crucial role in optimizing the parameters for KNN and SVM, resulting in better performance.

Improvements and Changes

- **Feature Scaling**: Standardizing the features significantly improved the performance of KNN and SVM.
- **Hyperparameter Tuning**: Finding the optimal parameters for KNN and SVM using GridSearchCV led to increased accuracy and F1 scores.
- **Cross-Validation**: Using cross-validation during tuning ensured robustness and avoided overfitting.

Model	Accuracy	F1	Precision	Recall
KNN	66%	0.65	0.66	0.63
SVM	68%	0.67	0.67	0.66
Naïve Bayes	62%	0.61	0.64	0.66

5- Boosting Ensemble Methods

5-1- Gradient Boosting Method

What is Gradient Boosting?

Gradient Boosting is an ensemble learning method designed for regression and classification problems. It builds models sequentially by combining the predictions of multiple weak learners (typically decision trees) to minimize the overall prediction error.

Key Features:

- **Boosting:** Aims to correct errors made by previous models by adding new models iteratively.
- **Gradient Descent:** Optimizes the loss function using gradient descent, focusing on the most significant errors in each iteration.
- **Flexibility:** Can handle diverse loss functions, making it suitable for various applications.

Steps in Gradient Boosting:

1. **Initialize:** Start with an initial prediction (e.g., mean for regression).
2. **Iterative Improvement:** Add models to correct residuals (differences between actual and predicted values).
3. **Weight Updates:** Assign weights to each weak learner based on their contribution to minimizing error.
4. **Combination:** Combine predictions from all models into a final, robust model.

Code Implementation:

The Gradient Boosting implementation was added to the code to perform hyperparameter tuning using `GridSearchCV`, train the model, and evaluate its performance.

1. Gradient Boosting

Gradient Boosting is an ensemble learning method where multiple weak learners (typically decision trees) are combined sequentially to minimize a loss function. Each subsequent model attempts to correct the errors of its predecessor, improving overall prediction performance.

```python
# 1. Gradient Boosting
print("\nGradient Boosting:")
param_grid_gb = {'n_estimators': [50, 100, 150], 'learning_rate': [0.01, 0.1, 0.2], 'max_depth': [3, 5, 7]}
gb = GridSearchCV(GradientBoostingClassifier(), param_grid_gb, cv=5, scoring='accuracy')
gb.fit(X_train, y_train)
best_gb = gb.best_estimator_
gb_metrics = evaluate_model(best_gb, X_test, y_test)
gb_metrics['Model'] = 'Gradient Boosting'
results.append(gb_metrics)
print("Best Parameters for Gradient Boosting:", gb.best_params_)
print(gb_metrics)
```

Steps

Defining the Parameter Grid:

- `n_estimators`: Number of boosting stages (i.e., the number of trees).
- `learning_rate`: Controls the contribution of each tree. Lower values slow down learning but can improve generalization.
- `max_depth`: Maximum depth of each tree, controlling the complexity of the model.

2. **Grid Search with Cross-Validation**:
   - `GridSearchCV` iterates over all combinations of the parameters in `param_grid_gb`.
   - It uses 5-fold cross-validation (`cv=5`) to split the training data into subsets, ensuring reliable evaluation of each combination.
   - The scoring metric is accuracy (`scoring='accuracy'`).

3. **Best Model Selection**:
   - After training, the model with the best parameters is retrieved using `gb.best_estimator_`.
   - This ensures we use the optimal combination of hyperparameters for testing.

4. **Evaluation**:
   - The best model is tested on the test data.
   - Metrics include:
     - **Accuracy**: Fraction of correct predictions.
     - **Precision**: Ratio of true positives to all positive predictions.
     - **Recall**: Ratio of true positives to all actual positives.
     - **F1 Score**: Harmonic mean of precision and recall.
   - The `evaluate_model` function calculates these metrics for comparison.

5-2- XGBoost

XGBoost (Extreme Gradient Boosting) is an advanced implementation of gradient boosting algorithms. It is designed to optimize both performance and speed, making it one of the most popular machine learning libraries for structured/tabular data. XGBoost offers additional functionalities and improvements over traditional gradient boosting methods, such as:

1. **Regularization:** Includes `L1` (Lasso) and `L2` (Ridge) regularization to reduce overfitting.
2. **Parallel Processing:** Utilizes multithreading to speed up computation.
3. **Sparsity Awareness:** Handles missing values and sparse data effectively.
4. **Tree Pruning:** Uses a greedy algorithm to prune trees and prevent overfitting.
5. **Custom Objective Functions:** Allows specification of different loss functions.

```
# 2. XGBoost
print("\nXGBoost:")
param_grid_xgb = {'n_estimators': [50, 100, 150], 'learning_rate': [0.01, 0.1, 0.2], 'max_depth': [3, 5, 7]}
xgb = GridSearchCV(XGBClassifier(use_label_encoder=False, eval_metric='mlogloss'), param_grid_xgb, cv=5, scoring='accuracy')
xgb.fit(X_train, y_train)
best_xgb = xgb.best_estimator_
xgb_metrics = evaluate_model(best_xgb, X_test, y_test)
xgb_metrics['Model'] = 'XGBoost'
results.append(xgb_metrics)
print("Best Parameters for XGBoost:", xgb.best_params_)
print(xgb_metrics)
```

Steps

1. **Imported and Configured XGBoost:**
   - We used the `XGBClassifier` from the `xgboost` library to evaluate its performance on the dataset.
2. **Hyperparameter Tuning:**
   - `GridSearchCV` was employed to fine-tune the hyperparameters, such as `n_estimators`, `learning_rate`, `max_depth`, and `subsample`.
3. **Model Training:**
   - The best XGBoost model was trained using the training dataset (`X_train`, `y_train`).
4. **Model Evaluation:**
   - The trained model was evaluated using metrics like accuracy, F1 score, precision, and recall on the test dataset (`X_test`, `y_test`).
5. **Comparison with Other Models:**
   - The results of the XGBoost model were added to the comparison table to assess its performance relative to other algorithms.

Model	Accuracy	F1	Precision	Recall
GBoost	71%	0.73	0.77	0.75
XGBoost	74%	0.76	0.76	0.71

6- Conclusion: Why Ensemble Methods Work Better

Ensemble methods, such as XGBoost, are often more effective than individual machine learning models like K-Nearest Neighbors (KNN), Support Vector Machines (SVM), or Naive Bayes for several reasons:

1. **Combining Strengths:** Ensemble methods combine multiple weak learners (base models) to create a stronger learner. By aggregating the predictions of different models, ensemble methods can correct the mistakes made by individual models. This helps improve overall accuracy, robustness, and generalization.
2. **Reduction of Overfitting:** While single models may be prone to overfitting, especially when they are too complex or trained on noisy data, ensemble methods like boosting (e.g., XGBoost) and bagging average out errors, reducing overfitting and making the model more robust.
3. **Better Handling of Complex Data:** Ensemble methods tend to perform better when faced with complex, high-dimensional, or noisy datasets. They are capable of capturing a variety of patterns that a single model might miss, which enhances their predictive power.
4. **Bias-Variance Tradeoff:** Many ensemble methods, such as boosting, help reduce both bias and variance. For example, XGBoost works by iteratively correcting errors made by previous models, which reduces bias without increasing variance excessively. Other methods like Random Forests reduce variance by averaging many models, which leads to more stable predictions.
5. **Handling Imbalanced Data:** Some ensemble methods, such as XGBoost, are particularly adept at handling imbalanced datasets by adjusting their internal parameters (e.g., `class_weight`), ensuring that the model does not simply favor the majority class.

XGBoost is a highly efficient ensemble method based on **Gradient Boosting**, where each subsequent tree corrects the errors of the previous one. It has several advantages:

- **Regularization**: XGBoost includes L1 and L2 regularization, which reduces overfitting and enhances the model's ability to generalize.
- **Tree Pruning**: It uses a more efficient tree pruning algorithm, which ensures that only the most relevant features are used to make decisions, improving model efficiency and interpretability.
- **Parallel Processing**: The ability to process data in parallel significantly speeds up model training, especially for large datasets, making it a go-to choice in real-world applications.
- **Handling Missing Data**: XGBoost handles missing values better by automatically learning the best way to handle them during training, which is critical when working with real-world data that may have missing entries.