In [1]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report, confusion_matrix
```

In [13]:
```
df=pd.read_csv("covid_impact_on_work.csv")
```

In [15]:
```
#I m Droping unimportant columns
columns_to_drop = ['Stress_Level', 'Childcare_Responsibilities', 'Salary_Changes', 'Affected_by_Covid']
df.drop(columns=columns_to_drop, inplace=True)
```

In [16]:
```
df.head()
```

Out[16]:

	Sector	Increased_Work_Hours	Work_From_Home	Hours_Worked_Per_Day	Meetings_Per_Day	Productivity_Change	Healt
0	Retail	1	1	6.392.393.639.805.820	26.845.944.014.488.700	1	
1	IT	1	1	9.171.983.537.957.560	33.392.245.834.602.800	1	
2	Retail	1	0	10.612.560.951.456.400	2.218.332.712.302.110	0	
3	Education	1	1	5.546.168.647.409.510	5.150.566.193.312.910	0	
4	Education	0	1	11.424.615.456.733.800	31.211.255.258.841.200	1	

In [17]:
```
def extract_first_number(value):
    first_number = value.split('.')[0]
    return int(first_number)


# I m tranforming meet/day and work_done/day to string to int for correct computation

df['Meetings_Per_Day'] = df['Meetings_Per_Day'].apply(extract_first_number)
df['Hours_Worked_Per_Day'] = df['Hours_Worked_Per_Day'].apply(extract_first_number)
```

In [18]:
```
df.head()
```

Out[18]:

	Sector	Increased_Work_Hours	Work_From_Home	Hours_Worked_Per_Day	Meetings_Per_Day	Productivity_Change	Health_Is
0	Retail	1	1	6	26	1	
1	IT	1	1	9	33	1	
2	Retail	1	0	10	2	0	
3	Education	1	1	5	5	0	
4	Education	0	1	11	31	1	

In [19]:
```
df.columns
```

Out[19]:
```
Index(['Sector', 'Increased_Work_Hours', 'Work_From_Home',
       'Hours_Worked_Per_Day', 'Meetings_Per_Day', 'Productivity_Change',
       'Health_Issue', 'Job_Security', 'Commuting_Changes',
       'Technology_Adaptation', 'Team_Collaboration_Challenges'],
      dtype='object')
```

In [20]:
```
# Encoding is being done here
sector_encoder = LabelEncoder()

# I m tranforming sector column to integer representation
df['Sector_encoded'] = sector_encoder.fit_transform(df['Sector'])
```

In [21]:
```
df.head()
```

Out[21]:

	Sector	Increased_Work_Hours	Work_From_Home	Hours_Worked_Per_Day	Meetings_Per_Day	Productivity_Change	Health_Is
0	Retail	1	1	6	26	1	
1	IT	1	1	9	33	1	
2	Retail	1	0	10	2	0	
3	Education	1	1	5	5	0	
4	Education	0	1	11	31	1	

In [25]:
```
df.drop(columns=['Sector'], inplace=True)
```

In [26]:
```
df.head()
```

Out[26]:

	Increased_Work_Hours	Work_From_Home	Hours_Worked_Per_Day	Meetings_Per_Day	Productivity_Change	Health_Issue	Job_S
0	1	1	6	26	1	0	
1	1	1	9	33	1	0	
2	1	0	10	2	0	0	
3	1	1	5	5	0	0	
4	0	1	11	31	1	0	

In [28]:
```
# Define features and target
X = df.drop('Job_Security', axis=1)
y = df['Job_Security']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)
```

In [29]:
```
# Create a Decision Tree Classifier
dt_classifier = DecisionTreeClassifier(random_state=42)

# Fit the model
dt_classifier.fit(X_train, y_train)
```

Out[29]:

DecisionTreeClassifier

DecisionTreeClassifier(random_state=42)

In [30]:
```
# Make predictions
y_pred = dt_classifier.predict(X_test)
```

In [31]:
```
# Evaluate the model
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
[[764 426]
 [488 322]]
              precision    recall  f1-score   support

           0       0.61      0.64      0.63      1190
           1       0.43      0.40      0.41       810

    accuracy                           0.54      2000
   macro avg       0.52      0.52      0.52      2000
weighted avg       0.54      0.54      0.54      2000
```

In [63]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.tree import DecisionTreeClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

# Scale features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Initialize Decision Tree Classifier with balanced class weights
dt_classifier = DecisionTreeClassifier(class_weight='balanced')


# param_grid = {
#     'max_depth': [None, 5, 10, 15, 20, 25, 30],      # Maximum depth of the tree
#     'min_samples_split': [2, 5, 10, 15],             # Minimum samples required to split an internal node
#     'min_samples_leaf': [1, 2, 4, 6, 8],             # Minimum samples required to be at a leaf node
#     'max_features': ['sqrt', 'log2', None],          # Valid options for max_features
#     'criterion': ['gini', 'entropy']                  # Function to measure the quality of a split
# }
# Define expanded parameter grid
param_grid = {
    'max_depth': [None, 5, 10, 15, 20, 25, 30, 35, 40],
    'min_samples_split': [2, 5, 10, 15, 20],
    'min_samples_leaf': [1, 2, 4, 6, 8, 10, 12],
    'max_features': ['sqrt', 'log2', None],
    'criterion': ['gini', 'entropy']
}

# Initialize GridSearchCV
grid_search = GridSearchCV(estimator=dt_classifier, param_grid=param_grid,
```

```
                           scoring='f1_weighted', cv=10, n_jobs=-1, error_score='raise')

try:
    # Fit GridSearchCV
    grid_search.fit(X_train_scaled, y_train)

    # Get the best estimator
    best_dt_classifier = grid_search.best_estimator_

    # Make predictions on the test set
    y_pred = best_dt_classifier.predict(X_test_scaled)

    # Evaluation
    print("Best Parameters:", grid_search.best_params_)
    print("Accuracy:", accuracy_score(y_test, y_pred))
    print("\nClassification Report:\n", classification_report(y_test, y_pred))
    print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))

except Exception as e:
    print(f"An error occurred during grid search: {e}")
```

```
Best Parameters: {'criterion': 'entropy', 'max_depth': 20, 'max_features': 'log2', 'min_samples_leaf': 6, 'min_s
amples_split': 10}
Accuracy: 0.5055

Classification Report:
              precision    recall  f1-score   support

           0       0.59      0.53      0.56      1190
           1       0.40      0.47      0.43       810

    accuracy                           0.51      2000
   macro avg       0.50      0.50      0.50      2000
weighted avg       0.52      0.51      0.51      2000

Confusion Matrix:
 [[634 556]
 [433 377]]
```

In [61]:

```
from sklearn.ensemble import RandomForestClassifier

# Initialize Random Forest Classifier
rf_classifier = RandomForestClassifier()

# Define parameter grid for Random Forest
rf_param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 5, 10, 15],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

# Initialize GridSearchCV
rf_grid_search = GridSearchCV(estimator=rf_classifier, param_grid=rf_param_grid,
                              scoring='f1_weighted', cv=5, n_jobs=-1)

# Fit GridSearchCV
rf_grid_search.fit(X_train_scaled, y_train)

# Get the best estimator and evaluate
best_rf_classifier = rf_grid_search.best_estimator_
y_rf_pred = best_rf_classifier.predict(X_test_scaled)

# Evaluation
print("Best Random Forest Parameters:", rf_grid_search.best_params_)
print("Random Forest Accuracy:", accuracy_score(y_test, y_rf_pred))
print("\nRandom Forest Classification Report:\n", classification_report(y_test, y_rf_pred))
```

```
Best Random Forest Parameters: {'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators'
: 200}
Random Forest Accuracy: 0.5565

Random Forest Classification Report:
              precision    recall  f1-score   support

           0       0.61      0.72      0.66      1190
           1       0.44      0.32      0.37       810

    accuracy                           0.56      2000
   macro avg       0.52      0.52      0.51      2000
weighted avg       0.54      0.56      0.54      2000
```

In [ ]: