IMPORTING LIBRARIES

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

from sklearn.model_selection import train_test_split
from sklearn import preprocessing
from sklearn.preprocessing import  LabelEncoder,MinMaxScaler


from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB

from sklearn import metrics
from sklearn.metrics import accuracy_score,classification_report,confusion_matrix
```

```
data=pd.read_csv('/content/survey.csv')
```

```
data.shape
```

```
(1259, 27)
```

```
data.describe()
```

	Age
count	1.259000e+03
mean	7.942815e+07
std	2.818299e+09
min	-1.726000e+03
25%	2.700000e+01
50%	3.100000e+01
75%	3.600000e+01
max	1.000000e+11

```
data = data[(data['Age'] >= 18) & (data['Age'] <= 100)]
```

```
data.describe()
```

	Age
count	1251.000000
mean	32.076739
std	7.288272
min	18.000000
25%	27.000000
50%	31.000000
75%	36.000000
max	72.000000

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 1251 entries, 0 to 1258
Data columns (total 27 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   Timestamp                  1251 non-null   object
```

```
 1   Age                        1251 non-null   int64 
 2   Gender                     1251 non-null   object
 3   Country                    1251 non-null   object
 4   state                      738 non-null    object
 5   self_employed              1233 non-null   object
 6   family_history             1251 non-null   object
 7   treatment                  1251 non-null   object
 8   work_interfere             989 non-null    object
 9   no_employees               1251 non-null   object
 10  remote_work                1251 non-null   object
 11  tech_company               1251 non-null   object
 12  benefits                   1251 non-null   object
 13  care_options               1251 non-null   object
 14  wellness_program           1251 non-null   object
 15  seek_help                  1251 non-null   object
 16  anonymity                  1251 non-null   object
 17  leave                      1251 non-null   object
 18  mental_health_consequence  1251 non-null   object
 19  phys_health_consequence    1251 non-null   object
 20  coworkers                  1251 non-null   object
 21  supervisor                 1251 non-null   object
 22  mental_health_interview    1251 non-null   object
 23  phys_health_interview      1251 non-null   object
 24  mental_vs_physical         1251 non-null   object
 25  obs_consequence            1251 non-null   object
 26  comments                   161 non-null    object
dtypes: int64(1), object(26)
memory usage: 273.7+ KB
```

```
data.head()
```

	Timestamp	Age	Gender	Country	state	self_employed	family_history	treatment	work_interfere	no_employees	...	leave	mental_
0	2014-08-27 11:29:31	37	Female	United States	IL	NaN	No	Yes	Often	6-25	...	Somewhat easy	
1	2014-08-27 11:29:37	44	M	United States	IN	NaN	No	No	Rarely	More than 1000	...	Don't know	
2	2014-08-27 11:29:44	32	Male	Canada	NaN	NaN	No	No	Rarely	6-25	...	Somewhat difficult	
3	2014-08-27 11:29:46	31	Male	United Kingdom	NaN	NaN	Yes	Yes	Often	26-100	...	Somewhat difficult	
4	2014-08-27 11:30:22	31	Male	United States	TX	NaN	No	No	Never	100-500	...	Don't know	

5 rows × 27 columns

```
data.isnull().sum()
```

	0
Timestamp	0
Age	0
Gender	0
Country	0
state	513
self_employed	18
family_history	0
treatment	0
work_interfere	262
no_employees	0
remote_work	0
tech_company	0
benefits	0
care_options	0
wellness_program	0
seek_help	0
anonymity	0
leave	0
mental_health_consequence	0
phys_health_consequence	0
coworkers	0
supervisor	0
mental_health_interview	0
phys_health_interview	0
mental_vs_physical	0
obs_consequence	0
comments	1090

dtype: int64

```
data.drop(['comments'], axis= 1, inplace=True)
data.drop(['state'], axis= 1, inplace=True)
data.drop(['Country'], axis= 1, inplace=True)
data.drop(['Timestamp'], axis= 1, inplace=True)
```

```
data.isnull().sum()
```

	0
Age	0
Gender	0
self_employed	18
family_history	0
treatment	0
work_interfere	262
no_employees	0
remote_work	0
tech_company	0
benefits	0
care_options	0
wellness_program	0
seek_help	0
anonymity	0
leave	0
mental_health_consequence	0
phys_health_consequence	0
coworkers	0
supervisor	0
mental_health_interview	0
phys_health_interview	0
mental_vs_physical	0
obs_consequence	0

dtype: int64

```
defaultInt = 0
defaultString = 'NaN'
defaultFloat = 0.0


intFeatures = ['Age']
stringFeatures = ['Gender', 'Country', 'self_employed', 'family_history', 'treatment', 'work_interfere',
                 'no_employees', 'remote_work', 'tech_company', 'anonymity', 'leave', 'mental_health_consequence',
                 'phys_health_consequence', 'coworkers', 'supervisor', 'mental_health_interview', 'phys_health_interview',
                 'mental_vs_physical', 'obs_consequence', 'benefits', 'care_options', 'wellness_program',
                 'seek_help']
floatFeatures = []


for feature in data:
    if feature in intFeatures:
        data[feature] = data[feature].fillna(defaultInt)
    elif feature in stringFeatures:
        data[feature] = data[feature].fillna(defaultString)
    elif feature in floatFeatures:
        data[feature] = data[feature].fillna(defaultFloat)
    else:
        print('Error: Feature %s not recognized.' % feature)


data.isnull().sum()
```

	0
Age	0
Gender	0
self_employed	0
family_history	0
treatment	0
work_interfere	0
no_employees	0
remote_work	0
tech_company	0
benefits	0
care_options	0
wellness_program	0
seek_help	0
anonymity	0
leave	0
mental_health_consequence	0
phys_health_consequence	0
coworkers	0
supervisor	0
mental_health_interview	0
phys_health_interview	0
mental_vs_physical	0
obs_consequence	0

dtype: int64

```
gender = data['Gender'].unique()
print(gender)
```

```
['Female' 'M' 'Male' 'male' 'female' 'm' 'Male-ish' 'maile' 'Trans-female'
 'Cis Female' 'F' 'something kinda male?' 'Cis Male' 'Woman' 'f' 'Mal'
 'Male (CIS)' 'queer/she/they' 'non-binary' 'Femake' 'woman' 'Make' 'Nah'
 'Enby' 'fluid' 'Genderqueer' 'Female ' 'Androgyne' 'Agender'
 'cis-female/femme' 'Guy (-ish) ^_^' 'male leaning androgynous' 'Male '
 'Man' 'Trans woman' 'msle' 'Neuter' 'Female (trans)' 'queer'
 'Female (cis)' 'Mail' 'cis male' 'Malr' 'femail' 'Cis Man'
 'ostensibly male, unsure what that really means']
```

```
male_str = ["male", "m", "male-ish", "maile", "mal", "male (cis)", "make", "male ", "man","msle", "mail", "malr","cis man", "Cis Male", "cis
trans_str = ["trans-female", "something kinda male?", "queer/she/they", "non-binary","nah", "all", "enby", "fluid", "genderqueer", "androgyr
female_str = ["cis female", "f", "female", "woman",  "femake", "female ","cis-female/femme", "female (cis)", "femail"]

for (row, col) in data.iterrows():

    if str.lower(col.Gender) in male_str:
        data['Gender'].replace(to_replace=col.Gender, value='male', inplace=True)

    if str.lower(col.Gender) in female_str:
        data['Gender'].replace(to_replace=col.Gender, value='female', inplace=True)

    if str.lower(col.Gender) in trans_str:
        data['Gender'].replace(to_replace=col.Gender, value='trans', inplace=True)

#Get rid of bullshit
stk_list = ['A little about you', 'p']
data = data[~data['Gender'].isin(stk_list)]

print(data['Gender'].unique())
```

```
/tmp/ipython-input-99-2513340261.py:11: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained as
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting valu

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].me


  data['Gender'].replace(to_replace=col.Gender, value='female', inplace=True)
/tmp/ipython-input-99-2513340261.py:8: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained ass
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting valu

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].me


  data['Gender'].replace(to_replace=col.Gender, value='male', inplace=True)
/tmp/ipython-input-99-2513340261.py:14: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained as
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting valu

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].me


  data['Gender'].replace(to_replace=col.Gender, value='trans', inplace=True)
['female' 'male' 'trans']
```

```
data['self_employed'] =data['self_employed'].replace([defaultString], 'No')
print(data['self_employed'].unique())
```

```
['No' 'Yes']
```

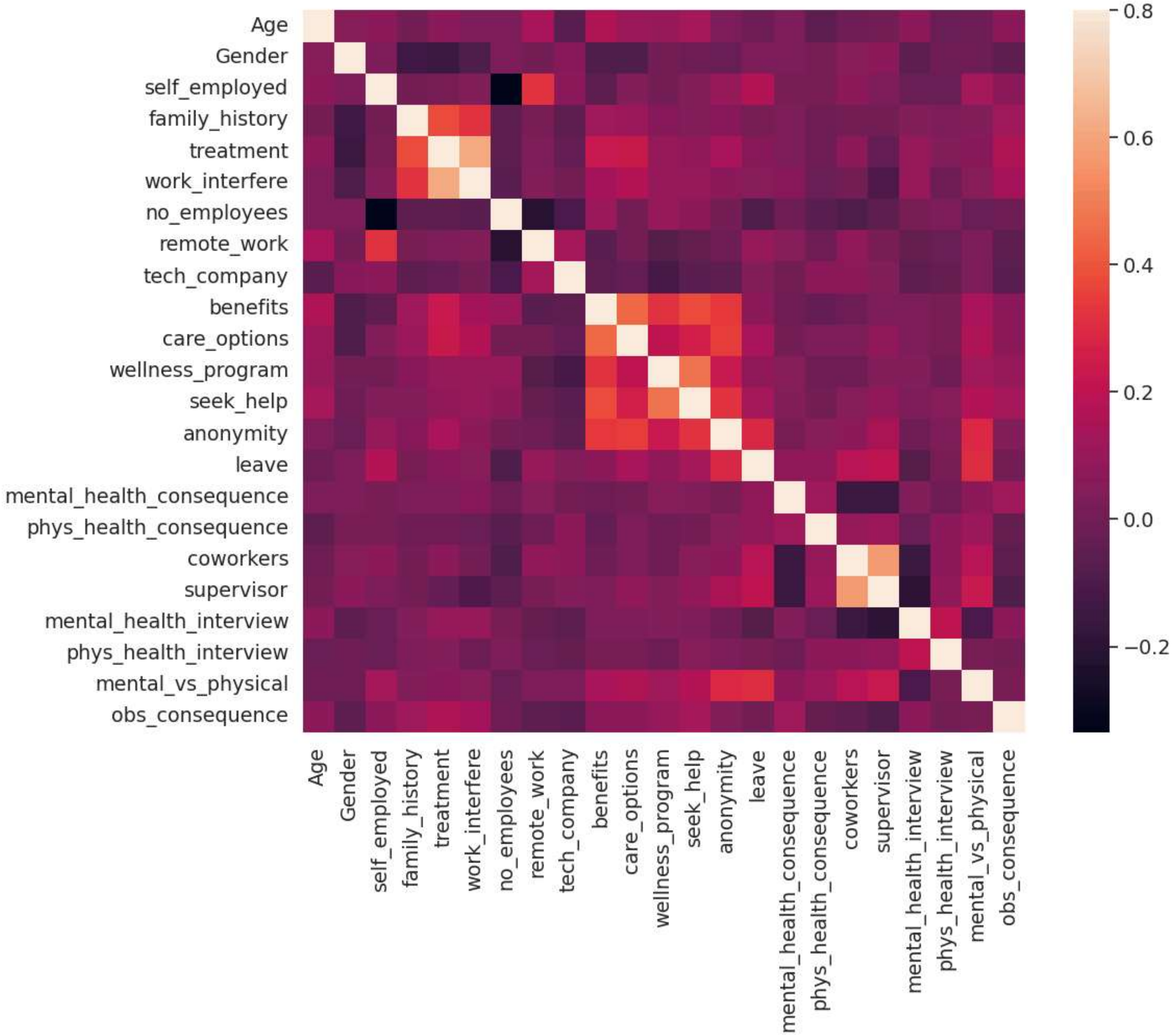
```
data['work_interfere'] = data['work_interfere'].replace([defaultString], 'Don\'t know' )
print(data['work_interfere'].unique())
```

```
['Often' 'Rarely' 'Never' 'Sometimes' "Don't know"]
```

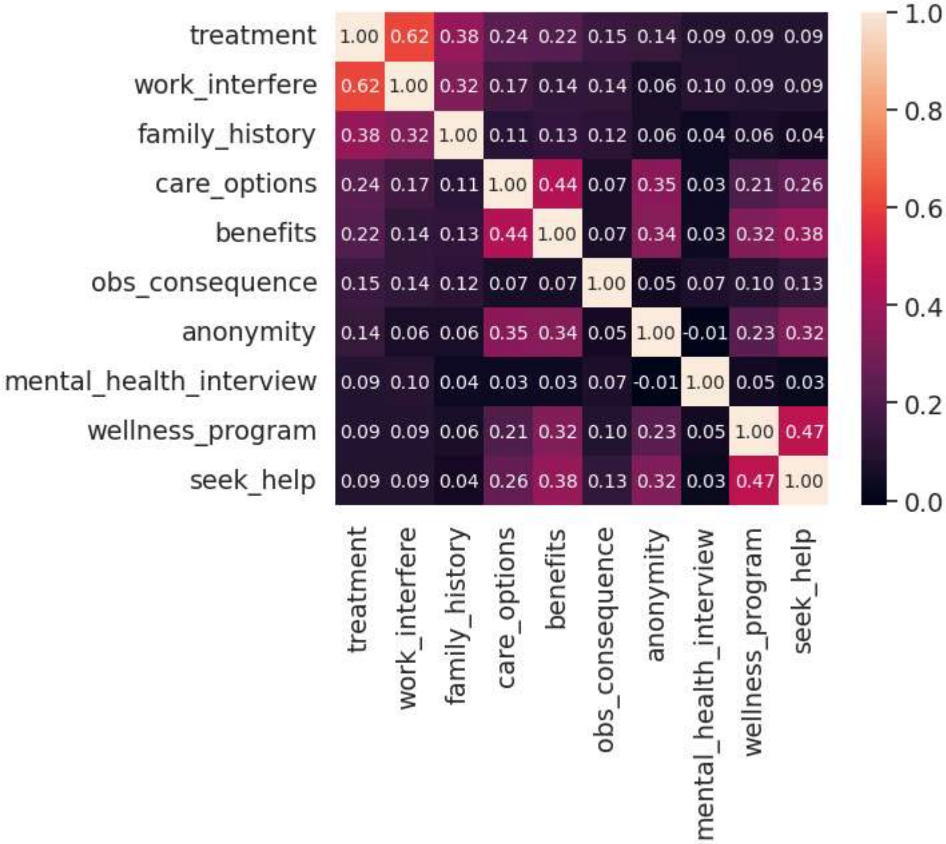
```
from sklearn import preprocessing

labelDict = {}


categorical_features = [col for col in data.columns if data[col].dtype == 'object' and col != 'Age']

for feature in categorical_features:
    le = preprocessing.LabelEncoder()
    le.fit(data[feature])
    data[feature] = le.transform(data[feature])

    le_name_mapping = dict(zip(le.classes_, le.transform(le.classes_)))
    labelKey = 'label_' + feature
    labelValue = list(le_name_mapping.keys())
    labelDict[labelKey] = labelValue

for key, value in labelDict.items():
    print(key, value)
```

```
label_Gender ['female', 'male', 'trans']
label_self_employed ['No', 'Yes']
label_family_history ['No', 'Yes']
label_treatment ['No', 'Yes']
label_work_interfere ["Don't know", 'Never', 'Often', 'Rarely', 'Sometimes']
label_no_employees ['1-5', '100-500', '26-100', '500-1000', '6-25', 'More than 1000']
label_remote_work ['No', 'Yes']
label_tech_company ['No', 'Yes']
label_benefits ["Don't know", 'No', 'Yes']
label_care_options ['No', 'Not sure', 'Yes']
label_wellness_program ["Don't know", 'No', 'Yes']
label_seek_help ["Don't know", 'No', 'Yes']
label_anonymity ["Don't know", 'No', 'Yes']
label_leave ["Don't know", 'Somewhat difficult', 'Somewhat easy', 'Very difficult', 'Very easy']
label_mental_health_consequence ['Maybe', 'No', 'Yes']
label_phys_health_consequence ['Maybe', 'No', 'Yes']
label_coworkers ['No', 'Some of them', 'Yes']
label_supervisor ['No', 'Some of them', 'Yes']
label_mental_health_interview ['Maybe', 'No', 'Yes']
label_phys_health_interview ['Maybe', 'No', 'Yes']
label_mental_vs_physical ["Don't know", 'No', 'Yes']
label_obs_consequence ['No', 'Yes']
```

```
corrmat = data.corr()
f, ax = plt.subplots(figsize=(12, 9))
sns.heatmap(corrmat, vmax=.8, square=True);
plt.show()
```




```
k=10
cols = corrmat.nlargest(k, 'treatment')['treatment'].index
cm = np.corrcoef(data[cols].values.T)
sns.set(font_scale=1.25)
hm = sns.heatmap(cm, cbar=True, annot=True, square=True, fmt='.2f', annot_kws={'size': 10}, yticklabels=cols.values, xticklabels=cols.values
plt.show()
```




```
scaler = MinMaxScaler()
data['Age'] = scaler.fit_transform(data[['Age']])
data.head()
```

	Age	Gender	self_employed	family_history	treatment	work_interfere	no_employees	remote_work	tech_company	benefits	...	an
0	0.351852	0	0	0	1	2	4	0	1	2	...	
1	0.481481	1	0	0	0	3	5	0	0	0	...	
2	0.259259	1	0	0	0	3	4	0	1	1	...	
3	0.240741	1	0	1	1	2	2	0	1	1	...	
4	0.240741	1	0	0	0	1	1	1	1	2	...	

5 rows × 23 columns

```
feature_cols = ['Age', 'Gender', 'family_history', 'work_interfere', 'leave', 'anonymity', 'mental_health_consequence', 'benefits']
X = data[feature_cols]
y = data.treatment
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state=0)
```

```
models = {
    "Logistic Regression":LogisticRegression(),
    "Decision Tree": DecisionTreeClassifier(),
    "Random Forest": RandomForestClassifier(),
    "Naive Bayes": GaussianNB()
}

# Evaluate each model
for name, model in models.items():
    print(f"--- {name} ---")
    # Train the model
    model.fit(X_train, y_train)
    # Make predictions
    y_pred = model.predict(X_test)
    # Calculate accuracy
    accuracy = accuracy_score(y_test, y_pred)
    # Display results
```

```
    print(f"Accuracy: {accuracy:.2f}")
    print("\nConfusion Matrix:")
    print(confusion_matrix(y_test, y_pred))
    print("\nClassification Report:")
    print(classification_report(y_test, y_pred))
    print("\n")
```

```
--- Decision Tree ---
Accuracy: 0.72

Confusion Matrix:
[[126  54]
 [ 50 146]]

Classification Report:
              precision    recall  f1-score   support

           0       0.72      0.70      0.71       180
           1       0.73      0.74      0.74       196

    accuracy                           0.72       376
   macro avg       0.72      0.72      0.72       376
weighted avg       0.72      0.72      0.72       376



--- Random Forest ---
Accuracy: 0.77

Confusion Matrix:
[[128  52]
 [ 33 163]]

Classification Report:
              precision    recall  f1-score   support

           0       0.80      0.71      0.75       180
           1       0.76      0.83      0.79       196

    accuracy                           0.77       376
   macro avg       0.78      0.77      0.77       376
weighted avg       0.78      0.77      0.77       376



--- Naive Bayes ---
Accuracy: 0.81

Confusion Matrix:
[[137  43]
 [ 28 168]]

Classification Report:
              precision    recall  f1-score   support

           0       0.83      0.76      0.79       180
           1       0.80      0.86      0.83       196

    accuracy                           0.81       376
   macro avg       0.81      0.81      0.81       376
weighted avg       0.81      0.81      0.81       376
```

```
from sklearn.model_selection import GridSearchCV

# Fine-tuning DecisionTreeClassifier
dt_param_grid = {
    'max_depth': [3, 5, 10, None],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}
dt_grid_search = GridSearchCV(DecisionTreeClassifier(), dt_param_grid, cv=5, scoring='accuracy')
dt_grid_search.fit(X_train, y_train)

print("Best parameters for DecisionTree:", dt_grid_search.best_params_)
print("Best score for DecisionTree:", dt_grid_search.best_score_)

# Fine-tuning RandomForestClassifier
```

```
rf_param_grid = {
    'n_estimators': [10, 50, 100, 200],
    'max_depth': [3, 5, 10, None],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}
rf_grid_search = GridSearchCV(RandomForestClassifier(), rf_param_grid, cv=5, scoring='accuracy')
rf_grid_search.fit(X_train, y_train)

print("Best parameters for RandomForest:", rf_grid_search.best_params_)
print("Best score for RandomForest:", rf_grid_search.best_score_)
```

```
Best parameters for DecisionTree: {'max_depth': 3, 'min_samples_leaf': 1, 'min_samples_split': 2}
Best score for DecisionTree: 0.8400000000000001
Best parameters for RandomForest: {'max_depth': 10, 'min_samples_leaf': 2, 'min_samples_split': 10, 'n_estimators': 50}
Best score for RandomForest: 0.8514285714285714
```

```
from sklearn.model_selection import RandomizedSearchCV

param_dist = {
    'n_estimators': [10, 50, 100, 200],
    'max_depth': [3, 5, 10, None],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

rf_random_search = RandomizedSearchCV(RandomForestClassifier(), param_distributions=param_dist, n_iter=100, cv=5, random_state=42)
rf_random_search.fit(X_train, y_train)
print("Best Parameters for Random Forest:", rf_random_search.best_params_)
print("Best Score for Random Forest:", rf_random_search.best_score_)
```

```
Best Parameters for Random Forest: {'n_estimators': 200, 'min_samples_split': 10, 'min_samples_leaf': 2, 'max_depth': 10}
Best Score for Random Forest: 0.8480000000000001
```

```
final_model1 = rf_random_search.best_estimator_  # or dt_grid_search.best_estimator_
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

y_pred = final_model1.predict(X_test)

print("Accuracy:", accuracy_score(y_test, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))
```

```
Accuracy: 0.7925531914893617
Confusion Matrix:
 [[128  52]
 [ 26 170]]
Classification Report:
               precision    recall  f1-score   support

           0       0.83      0.71      0.77       180
           1       0.77      0.87      0.81       196

    accuracy                           0.79       376
```