Libraries

```
# Importing necessary libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report
from sklearn.impute import SimpleImputer
from sklearn.metrics import accuracy_score, precision_score,
recall_score, f1_score, roc_auc_score
from sklearn.ensemble import VotingClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier,
GradientBoostingClassifier
import matplotlib.pyplot as plt
import joblib
```

Loading Data

```
# Reading the dataset
df = pd.read_csv("DataSampled.csv")  # Replace "your_dataset.csv" with
the actual filename

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

df.columns = df.columns.str.strip()
# Drop the 'STAR' variable
df.drop(columns=['STAR'], inplace=True)
```

```
   MMSE  Age  Weight   Height   Waist     Hip  Smoking  Smoking
(packet/year)  \
0   NaN   64     66.0    155.0    89.0  104.0      0.0
0
1   NaN   53     55.0    150.0    77.0   97.0      0.0
0
2  24.0   56     56.0    150.0   112.0  125.0      0.0
0
3   NaN   58     75.0    160.0   113.0    NaN      0.0
0
4  30.0   55     72.0    157.0    98.0  104.0      1.0
4

    Alcohol  DM   ...  Exercise  LowCST   CST  Gait speed  Low grip
strength  \
```

```
0          0    0  ...          0     0.0   8.1        1.28
0.0
1          0    0  ...        NaN     0.0   8.0        1.47
1.0
2          0    0  ...          0     0.0  11.7        0.80
0.0
3          0    1  ...          0     1.0  18.0        1.34
0.0
4          0    0  ...  1-2/week     NaN  10.9        1.09
NaN

  Grip strength  SARCOPENIA  STAR    BMI  Gender
0          28.0         0.0  1.00  27.40       F
1          16.0         0.0  1.84  24.40       F
2          23.0         0.0  1.48  36.00       F
3          23.0         0.0  1.12  29.30       F
4          21.0         0.0  1.45  29.21       F

[5 rows x 39 columns]
```

```
# Printing the information about the dataset
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1303 entries, 0 to 1302
Data columns (total 38 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   MMSE                    812 non-null    float64
 1   Age                     1303 non-null   int64
 2   Weight                  1302 non-null   float64
 3   Height                  1301 non-null   float64
 4   Waist                   1297 non-null   float64
 5   Hip                     1296 non-null   float64
 6   Smoking                 1302 non-null   float64
 7   Smoking (packet/year)   1301 non-null   object
 8   Alcohol                 1303 non-null   object
 9   DM                      1303 non-null   int64
 10  DM duration             1289 non-null   float64
 11  İnsülin                 1297 non-null   float64
 12  DM drug                 379 non-null    object
 13  Hiperlipidemi           1301 non-null   float64
 14  Dyslipidemia duration   1287 non-null   float64
 15  Dyslipidemia drugs      231 non-null    object
 16  KAH                     1261 non-null   float64
 17  KAH duration            1249 non-null   float64
 18  Hipotiroidi             1271 non-null   float64
 19  ASTIM                   1180 non-null   float64
 20  KOAH                    1225 non-null   float64
 21  OP                      355 non-null    float64
```

```
 22  Other(s)                  285 non-null    object
 23  HT                        1303 non-null   int64
 24  Anti-HT drug type         750 non-null    object
 25  HT duration               1261 non-null   float64
 26  Education                 1283 non-null   object
 27  Occupation                1237 non-null   object
 28  Working Status            1223 non-null   object
 29  Exercise                  1283 non-null   object
 30  LowCST                    1113 non-null   float64
 31  CST                       1303 non-null   float64
 32  Gait speed                1300 non-null   float64
 33  Low grip strength         1166 non-null   float64
 34  Grip strength             1303 non-null   float64
 35  SARCOPENIA                1302 non-null   float64
 36  BMI                       1301 non-null   float64
 37  Gender                    1303 non-null   object
dtypes: float64(24), int64(3), object(11)
memory usage: 387.0+ KB
None
```

```
# Printing the shape of the dataset
print("Shape of the dataset:", df.shape)
```

```
Shape of the dataset: (1303, 38)
```

```
# Print data types of all columns
print(df.dtypes)
```

```
MMSE                       float64
Age                          int64
Weight                     float64
Height                     float64
Waist                      float64
Hip                        float64
Smoking                    float64
Smoking (packet/year)       object
Alcohol                     object
DM                           int64
DM duration                float64
İnsülin                    float64
DM drug                     object
Hiperlipidemi              float64
Dyslipidemia duration      float64
Dyslipidemia drugs          object
KAH                        float64
KAH duration               float64
Hipotiroidi                float64
ASTIM                      float64
KOAH                       float64
OP                         float64
```

```
Other(s)                    object
HT                           int64
Anti-HT drug type           object
HT duration                float64
Education                   object
Occupation                  object
Working Status              object
Exercise                    object
LowCST                     float64
CST                        float64
Gait speed                 float64
Low grip strength          float64
Grip strength              float64
SARCOPENIA                 float64
BMI                        float64
Gender                      object
dtype: object
```

Data Cleaning

```
# Convert categorical columns to categorical data type
categorical_cols = ['Smoking (packet/year)', 'Alcohol', 'DM drug',
'Dyslipidemia drugs', 'Other(s)', 'Anti-HT drug type', 'Education',
'Occupation', 'Working Status', 'Exercise', 'Gender']
df[categorical_cols] = df[categorical_cols].astype('category')

# Check for missing values in categorical columns
missing_categorical = df[categorical_cols].isnull().sum()
print("Missing values in categorical columns:")
print(missing_categorical)

# Check unique values in each categorical column
for col in categorical_cols:
    unique_values = df[col].unique()
    print("\nUnique values in", col, ":", unique_values)
```

```
Missing values in categorical columns:
Smoking (packet/year)       2
Alcohol                     0
DM drug                   924
Dyslipidemia drugs       1072
Other(s)                 1018
Anti-HT drug type         553
Education                  20
Occupation                 66
Working Status             80
Exercise                   20
Gender                      0
```

```
dtype: int64

Unique values in Smoking (packet/year) : ['0', '4', '30', '11.5',
'15', ..., '43', '37', '27', '55', 'EX SMOKER 3YEARS/ UNLIMITED']
Length: 71
Categories (70, object): ['0', '0.1', '1', '10', ..., 'Former Smoker:
3', 'ex Smoker: 25',
                          'ex Smoker: 30', 'shisha for 20 years']

Unique values in Alcohol : ['0', 'social', 'Social ', 'Regular',
'regular', 'Social', 'social ']
Categories (7, object): ['0', 'Regular', 'Social', 'Social ',
'regular', 'social', 'social ']

Unique values in DM drug : [NaN, 'Glicazide', 'Metformin',
'metformin+sitagliptin+empagliflozin+insülin a..., 'metformin', ...,
'empagliflozin,metformin,gliklazid,vildaglipti..., 'insulin
glarjin,metformin', 'insülin glarjin,insülin aspart', 'insülin
glargine,sitagliptin,metformin', ' Pioglitazon, Empagliflozin']
Length: 185
Categories (184, object): [' Empagliflozin,Metformin,Linagliptin',
'Cloridrate Metformine + Liraglutide + Pioglit...,
                           'GLICLAZIDE, INSULIN', 'GLICLAZIDE,
SITAGLIPTIN', ...,
                           'vİldagliptin/Metformin/Gliktazid',
'without drug', ' Pioglitazon, Empagliflozin',
                           'İnsülin aspart+aspart protamin']

Unique values in Dyslipidemia drugs : [NaN, 'rosuvastatine',
'Atorvastatin', 'atorvastatine', 'atorvastatin', ..., 'Rosuvastatin,
fenofibrate', 'revostatin', 'Pravastatin', 'Rosuvastatim',
'Simvastatin, Fenofibrate']
Length: 29
Categories (28, object): ['Atorvastatin', 'Atorvastatin ',
'Atorvastatin, fenofibrate',
                          'Fenofibrate', ..., 'rosuvastatin',
'rosuvastatine', 'simvastatin',
                          'simvastatin, fenofibrate']

Unique values in Other(s) : [NaN, 'aritmi', 'LDH', 'gut,epilepsi',
'tromboflebit', ..., 'chd, arritmia', 'CDH', 'bph,hbv', 'kby, hasta
sinüs sendromu, kalp yetmezliği, a..., 'prostat ca']
Length: 179
Categories (178, object): [' İTP', 'AF', 'AS', 'Allergic
rhinitis', ...,
                           'yaygın anksiyete bozukluğu', 'Ürtiker',
'üveit', 'şizofreni']

Unique values in Anti-HT drug type : [NaN, 'ARB+DIURETIC', 'CCB', 'B
BLOCKER', 'ACE+B BLOCKER+CCB+DIURETIC', ..., 'ALDOSTERON+DİÜRETİK', 'B
```

```
BLOCKER+ALDOSTERON', 'ARB+B BLOCKER+CCB+DİÜRETİC', 'ARB+ALFA BLOCKER',
'ARB+CCB+DIURETIC+ALDOSTERON']
Length: 36
Categories (35, object): ['A BLOCKER', 'ACE', 'ACE+B BLOCKER', 'ACE+B
BLOCKER+CCB', ...,
                          'CCB+A BLOCKER', 'CCB+ALDOSTERON', 'CCB+B
BLOCKER ', 'DIURETIC']

Unique values in Education : ['High School', NaN, 'Illiterate',
'illiterate', 'ilkokul', ..., 'ortaokul', 'none', 'ılliterate',
'Lise', 'worker ']
Length: 31
Categories (30, object): ['High School', 'High school', 'Illiterate',
'Lise', ..., 'worker ',
                          'üniversite', 'İlliterate', 'ılliterate']

Unique values in Occupation : ['Officer', NaN, 'house wife',
'housewife', 'ev hanımı', ..., 'satış temsilcisi', 'hairdresser',
'Beautician', 'marketer', 'painter']
Length: 236
Categories (235, object): ['??', 'ADVOCATE', 'Academician',
'Accountant', ..., 'welder',
                           'worker', 'öğretmen', 'İşçi']

Unique values in Working Status : ['Retire', NaN, 'retired',
'unemployed', 'çalışmıyor', ..., 'FULLTIME', 'Emekli veya çalışmıyor',
'tam/kısmi zamnalı çalışıyor', 'ull-time/part-time work', 'actively
working']
Length: 47
Categories (46, object): ['#REF!', 'Abstinence', 'Emekli', 'Emekli
veya çalışmıyor', ...,
                          'ÇALIŞIYOR', 'Çalışmıyor', 'çalışmıyor',
'çalıışıyor']

Unique values in Exercise : ['0', NaN, '1-2/week', '3-4/week']
Categories (3, object): ['0', '1-2/week', '3-4/week']

Unique values in Gender : ['F', 'M', 'F ', 'f']
Categories (4, object): ['F', 'F ', 'M', 'f']

# Convert education levels to lowercase
df['Education'] = df['Education'].str.lower()

# Or if you prefer uppercase:
# df['Education'] = df['Education'].str.upper()

# Check unique values again to confirm standardization
unique_education_levels = df['Education'].unique()
print("Unique Education Levels after standardization:",
unique_education_levels)
```

```
Unique Education Levels after standardization: ['high school' nan
'illiterate' 'ilkokul' 'secondary school' 'university'
 'primary school' 'highschool' 'middle school' 'housewife'
'illeterate'
 'illiterate' 'üniversite' 'lise' 'okur-yzar değil' 'ortaokul' 'none'
 'ılliterate' 'worker ']
```

```python
# Map similar education levels to a standard representation
education_mapping = {
    'highschool': 'high school',
    'illeterate': 'illiterate',
    'illiterate': 'illiterate',
    'ılliterate': 'illiterate',
    'üniversite': 'university',
    'lise': 'secondary school',
    'ilkokul': 'primary school',
    'ortaokul': 'middle school'
}

# Replace mapped values
df['Education'].replace(education_mapping, inplace=True)

# Remove unusual entry 'okur-yzar değil'
df = df[df['Education'] != 'okur-yzar değil']

# Check unique values again
unique_education_levels = df['Education'].unique()
print("Unique Education Levels after mapping and removing unusual
entry:", unique_education_levels)
```

```
Unique Education Levels after mapping and removing unusual entry:
['high school' nan 'illiterate' 'primary school' 'secondary school'
 'university' 'middle school' 'housewife' 'none' 'worker ']
```

```python
# Define mapping of education levels to numerical codes
education_mapping = {
    'illiterate': 0,
    'primary school': 1,
    'middle school': 2,
    'high school': 3,
    'secondary school': 3,  # Mapped to the same code as 'high school'
    'university': 4,
    'housewife': 5,
    'worker': 6,
    'none': 7
}

# Replace education levels with numerical codes
df['Education'] = df['Education'].map(education_mapping)
```

```
# Check unique values again
unique_education_levels = df['Education'].unique()
print("Unique Education Levels after numerical encoding:",
unique_education_levels)
```

```
Unique Education Levels after numerical encoding: [ 3. nan  0.  1.  4.
2.  5.  7.]
```

```
<ipython-input-201-4b3b7c3e9d09>:15: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation:
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#
returning-a-view-versus-a-copy
  df['Education'] = df['Education'].map(education_mapping)
```

```
# Define mapping of exercise levels to numerical codes
exercise_mapping = {
    '0': 0,
    '1-2/week': 1,
    '3-4/week': 2
}

# Replace exercise levels with numerical codes
df['Exercise'] = df['Exercise'].map(exercise_mapping)

# Count occurrences of each exercise level
exercise_counts = df['Exercise'].value_counts()

# Find the most frequent exercise level
most_frequent_exercise = exercise_counts.idxmax()

# Fill NaN values with the most frequent exercise level
df['Exercise'].fillna(most_frequent_exercise, inplace=True)

# Convert the 'Exercise' column to integer type
df['Exercise'] = df['Exercise'].astype(int)

# Check unique values again
unique_exercise_levels = df['Exercise'].unique()
print("Unique Exercise Levels after numerical encoding:",
unique_exercise_levels)
```

```
Unique Exercise Levels after numerical encoding: [0 1 2]

<ipython-input-202-22e563d6550a>:9: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation:
```

```
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#
returning-a-view-versus-a-copy
  df['Exercise'] = df['Exercise'].map(exercise_mapping)
<ipython-input-202-22e563d6550a>:18: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation:
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#
returning-a-view-versus-a-copy
  df['Exercise'].fillna(most_frequent_exercise, inplace=True)
<ipython-input-202-22e563d6550a>:21: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation:
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#
returning-a-view-versus-a-copy
  df['Exercise'] = df['Exercise'].astype(int)
```

```
# Define mapping of gender categories to numerical codes
gender_mapping = {
    'F': 0,
    'F ': 0,  # Consider 'F ' as 'F'
    'f': 0,   # Consider 'f' as 'F'
    'M': 1
}

# Replace gender categories with numerical codes
df['Gender'] = df['Gender'].map(gender_mapping)

# Check unique values again
unique_genders = df['Gender'].unique()
print("Unique Gender Levels after numerical encoding:",
unique_genders)
```

```
Unique Gender Levels after numerical encoding: [0 1]
```

```
# Convert all entries to lowercase and remove leading/trailing spaces
df['Alcohol'] = df['Alcohol'].str.lower().str.strip()

# Define mapping of alcohol categories to numerical codes
alcohol_mapping = {
    '0': 0,
    'regular': 1,
    'social': 2
}

# Replace alcohol categories with numerical codes
df['Alcohol'] = df['Alcohol'].map(alcohol_mapping)

# Check unique values again
```

```
unique_alcohol_levels = df['Alcohol'].unique()
print("Unique Alcohol Levels after numerical encoding:",
unique_alcohol_levels)
```

```
Unique Alcohol Levels after numerical encoding: [0 2 1]
```

```
# Remove words and keep only the numbers
df['Smoking (packet/year)'] = df['Smoking
(packet/year)'].str.replace(r'[^\d\.]', '', regex=True)

# Convert to float
df['Smoking (packet/year)'] = df['Smoking
(packet/year)'].astype(float)


# Display unique values after cleaning
print("Unique Smoking Levels after cleaning:", df['Smoking
(packet/year)'].unique())
```

```
Unique Smoking Levels after cleaning: [0.000e+00 4.000e+00 3.000e+01
1.150e+01 1.500e+01 2.500e+01 3.500e+01
 1.000e+02 7.000e+01 5.000e+00 4.500e+01 1.200e+01 1.000e+01 1.500e+02
 4.000e+01 1.400e+01 1.000e+00 1.000e-01 2.000e+01 7.000e+00 7.200e+01
 2.100e+01 4.900e+01 1.700e+01 5.000e+01 2.000e+00 1.040e+02 7.500e+00
 8.000e+00 3.200e+01 6.000e+01 1.800e+01 3.009e+03 3.000e+00 6.000e+00
 1.300e+01 8.000e+01 4.600e+01 9.000e+00 8.800e+01 6.500e+01 1.250e+01
 2.500e+00 5.100e+01 2.900e+01 9.000e+01 4.400e+01 7.500e+01 3.600e+01
 2.400e+01 6.400e+01 2.800e+01       nan 5.200e+01 3.400e+01 2.300e+01
 6.200e+01 1.600e+01 1.560e+02 4.300e+01 3.700e+01 2.700e+01
5.500e+01]
```

```
# Define the mapping dictionary
working_status_map = {
    'Retire': 0,
    'retired': 0,
    'unemployed': 1,
    'çalışmıyor': 1,  # Assuming this means unemployed
    'FULLTIME': 2,
    'Emekli veya çalışmıyor': 3,  # Retired or unemployed
    'tam/kısmi zamnalı çalışıyor': 4,  # Working full-time or part-
time
    'ull-time/part-time work': 4,  # Typo correction
    'actively working': 5
}

# Replace categorical values with numerical codes
df['Working Status'] = df['Working Status'].map(working_status_map)

# Display unique values after mapping
print("Unique values in Working Status after mapping:", df['Working
Status'].unique())
```

```
Unique values in Working Status after mapping: [ 0. nan  1.  2.  3.
4.  5.]
```

```
# Define the mapping dictionary
occupation_map = {
    'Officer': 0,
    'house wife': 1,
    'housewife': 1,
    'ev hanımı': 1,  # Turkish equivalent of housewife
    'satış temsilcisi': 2,  # Sales representative
    'hairdresser': 3,
    'Beautician': 3,
    'marketer': 4,
    'painter': 5,
}

# Replace categorical values with numerical codes
df['Occupation'] = df['Occupation'].map(occupation_map)

# Display unique values after mapping
print("Unique values in Occupation after mapping:",
df['Occupation'].unique())
```

```
Unique values in Occupation after mapping: [ 0. nan  1.  2.  3.  4.
5.]
```

```
print(df.dtypes)
```

```
MMSE                       float64
Age                          int64
Weight                     float64
Height                     float64
Waist                      float64
Hip                        float64
Smoking                    float64
Smoking (packet/year)      float64
Alcohol                      int64
DM                           int64
DM duration                float64
İnsülin                    float64
DM drug                   category
Hiperlipidemi              float64
Dyslipidemia duration      float64
Dyslipidemia drugs        category
KAH                        float64
KAH duration               float64
Hipotiroidi                float64
ASTIM                      float64
KOAH                       float64
OP                         float64
```

```
Other(s)                  category
HT                           int64
Anti-HT drug type         category
HT duration                float64
Education                  float64
Occupation                 float64
Working Status             float64
Exercise                     int64
LowCST                     float64
CST                        float64
Gait speed                 float64
Low grip strength          float64
Grip strength              float64
SARCOPENIA                 float64
BMI                        float64
Gender                       int64
dtype: object
```

Spliting based on gender

```
# Drop rows with missing target values
df.dropna(subset=['SARCOPENIA'], inplace=True)

# Split the dataset into separate datasets for men and women
df_men = df[df['Gender'] == 1]
df_women = df[df['Gender'] == 0]

print(df.columns)
```

```
Index(['MMSE', 'Age', 'Weight', 'Height', 'Waist', 'Hip', 'Smoking',
       'Smoking (packet/year)', 'Alcohol', 'DM', 'DM duration',
'İnsülin',
       'DM drug', 'Hiperlipidemi', 'Dyslipidemia duration',
       'Dyslipidemia drugs', 'KAH', 'KAH duration', 'Hipotiroidi',
'ASTIM',
       'KOAH', 'OP', 'Other(s)', 'HT', 'Anti-HT drug type', 'HT
duration',
       'Education', 'Occupation', 'Working Status', 'Exercise',
'LowCST',
       'CST', 'Gait speed', 'Low grip strength', 'Grip strength',
'SARCOPENIA',
       'BMI', 'Gender'],
      dtype='object')
```

Models

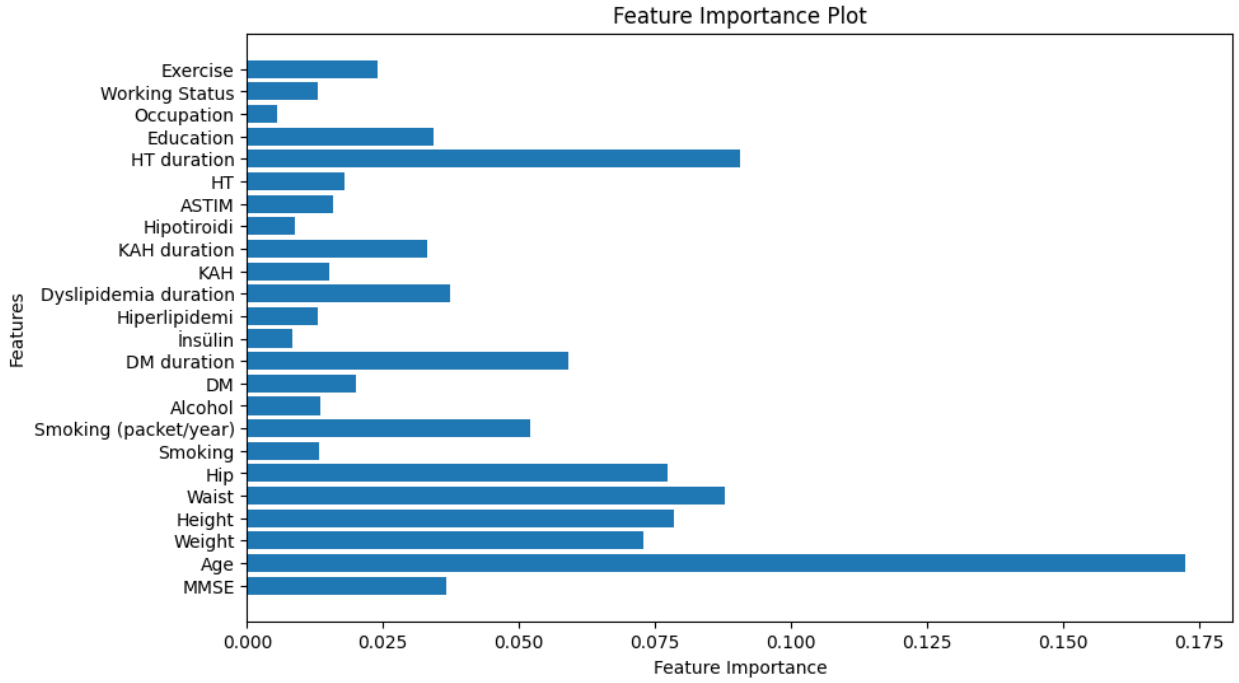
```
# Select features
selected_features = [
    'MMSE', 'Age', 'Weight', 'Height', 'Waist', 'Hip', 'Smoking',
       'Smoking (packet/year)', 'Alcohol', 'DM', 'DM duration',
'İnsülin', 'Hiperlipidemi', 'Dyslipidemia duration','KAH', 'KAH
duration', 'Hipotiroidi', 'ASTIM',
       'HT', 'HT duration','Education', 'Occupation', 'Working
Status', 'Exercise']

# Selecting the features and target variable
X = df_men[selected_features]
y = df_men['SARCOPENIA']

# Create the SimpleImputer object with strategy='mean'
imputer = SimpleImputer(strategy='mean')

# Fit and transform the imputer on the numerical columns with missing
values
X_imputed = imputer.fit_transform(X)

# Splitting the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_imputed, y,
test_size=0.2, random_state=42)

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

# Define models
lr_model = LogisticRegression(random_state=42)
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
gb_model = GradientBoostingClassifier(n_estimators=100,
random_state=42)

# Create ensemble model
ensemble_model1M = VotingClassifier(estimators=[
    ('lr', lr_model),
    ('rf', rf_model),
    ('gb', gb_model)
], voting='soft')

# Train ensemble model
ensemble_model1M.fit(X_train_scaled, y_train)

# Predict using ensemble model
ensemble_pred = ensemble_model1M.predict(X_test_scaled)
```

```
probabilities = ensemble_model1M.predict_proba(X_test_scaled)[:, 1]
# Diagnose patients
threshold_high = 0.8  # Very high probability threshold
threshold_low = 0.2   # Very low probability threshold

predicted_sarcopenia = ['Positive' if prob >= threshold_high else
'Negative' if prob <= threshold_low else 'Further testing required'
for prob in probabilities]

print(predicted_sarcopenia)

# Evaluate ensemble model
print("\nEnsemble Model:")
print(classification_report(y_test, ensemble_pred))
print("Accuracy:", accuracy_score(y_test, ensemble_pred))
print("Precision:", precision_score(y_test, ensemble_pred))
print("Recall:", recall_score(y_test, ensemble_pred))
print("F1-score:", f1_score(y_test, ensemble_pred))
print("ROC AUC:", roc_auc_score(y_test,
ensemble_model1M.predict_proba(X_test_scaled)[:, 1]))

# Save the trained model
joblib.dump(ensemble_model1M, "ensemble_model1M.pkl",  protocol=4)
```

```
['Negative', 'Negative', 'Negative', 'Negative', 'Further testing
required', 'Negative', 'Further testing required', 'Negative',
'Further testing required', 'Negative', 'Further testing required',
'Further testing required', 'Negative', 'Further testing required',
'Negative', 'Negative', 'Negative', 'Further testing required',
'Negative', 'Negative', 'Negative', 'Further testing required',
'Negative', 'Negative', 'Negative', 'Negative', 'Further testing
required', 'Further testing required', 'Negative', 'Negative',
'Negative', 'Negative', 'Further testing required', 'Negative',
'Further testing required', 'Further testing required', 'Negative',
'Further testing required', 'Negative', 'Further testing required',
'Negative', 'Negative', 'Further testing required', 'Negative',
'Negative', 'Negative', 'Further testing required', 'Negative',
'Negative', 'Further testing required', 'Negative', 'Negative',
'Negative', 'Negative', 'Further testing required', 'Negative',
'Negative', 'Negative', 'Negative', 'Negative', 'Negative',
'Negative', 'Further testing required', 'Negative', 'Negative',
'Negative', 'Negative', 'Further testing required', 'Negative',
'Further testing required', 'Negative', 'Negative', 'Negative',
'Negative', 'Further testing required', 'Negative']

Ensemble Model:
              precision    recall  f1-score   support

         0.0       0.83      0.97      0.89        61
```

```
         1.0       0.60      0.20      0.30        15

    accuracy                           0.82        76
   macro avg       0.72      0.58      0.60        76
weighted avg       0.79      0.82      0.78        76

Accuracy: 0.8157894736842105
Precision: 0.6
Recall: 0.2
F1-score: 0.3
ROC AUC: 0.7672131147540984

['ensemble_model1M.pkl']
```

```python
# Get feature importances
feature_importances =
ensemble_model1M.named_estimators_['rf'].feature_importances_
# Plot feature importances
plt.figure(figsize=(10, 6))
plt.barh(selected_features, feature_importances)
plt.xlabel('Feature Importance')
plt.ylabel('Features')
plt.title('Feature Importance Plot')
plt.show()
```



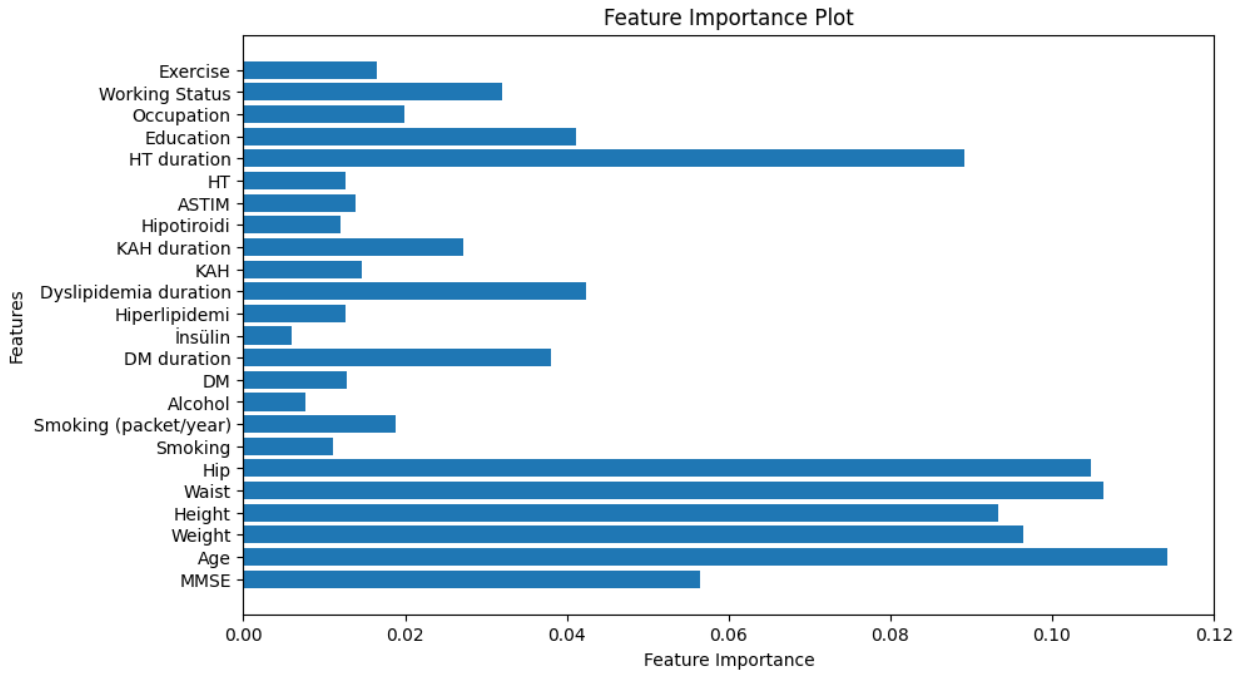

```python
# Select features
selected_features = [
    'MMSE', 'Age', 'Weight', 'Height', 'Waist', 'Hip', 'Smoking',
```

```
       'Smoking (packet/year)', 'Alcohol', 'DM', 'DM duration',
'İnsülin', 'Hiperlipidemi', 'Dyslipidemia duration','KAH', 'KAH
duration', 'Hipotiroidi', 'ASTIM',
       'HT', 'HT duration','Education', 'Occupation', 'Working
Status', 'Exercise']

# Selecting the features and target variable
X = df_women[selected_features]
y = df_women['SARCOPENIA']

# Create the SimpleImputer object with strategy='mean'
imputer = SimpleImputer(strategy='mean')

# Fit and transform the imputer on the numerical columns with missing
values
X_imputed = imputer.fit_transform(X)

# Splitting the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_imputed, y,
test_size=0.2, random_state=42)

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

# Define models
lr_model = LogisticRegression(random_state=42)
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
gb_model = GradientBoostingClassifier(n_estimators=100,
random_state=42)

# Create ensemble model
ensemble_model1F = VotingClassifier(estimators=[
    ('lr', lr_model),
    ('rf', rf_model),
    ('gb', gb_model)
], voting='soft')

# Train ensemble model
ensemble_model1F.fit(X_train_scaled, y_train)

# Predict using ensemble model
ensemble_pred = ensemble_model1F.predict(X_test_scaled)

probabilities = ensemble_model1F.predict_proba(X_test_scaled)[:, 1]
# Diagnose patients
threshold_high = 0.8  # Very high probability threshold
threshold_low = 0.2   # Very low probability threshold
```

```
predicted_sarcopenia = ['Positive' if prob >= threshold_high else
'Negative' if prob <= threshold_low else 'Further testing required'
for prob in probabilities]

print(predicted_sarcopenia)

# Evaluate ensemble model
print("\nEnsemble Model:")
print(classification_report(y_test, ensemble_pred))
print("Accuracy:", accuracy_score(y_test, ensemble_pred))
print("Precision:", precision_score(y_test, ensemble_pred))
print("Recall:", recall_score(y_test, ensemble_pred))
print("F1-score:", f1_score(y_test, ensemble_pred))
print("ROC AUC:", roc_auc_score(y_test,
ensemble_model1F.predict_proba(X_test_scaled)[:, 1]))

# Save the trained model
joblib.dump(ensemble_model1F, "ensemble_model1F.pkl",  protocol=4)
```

```
['Further testing required', 'Negative', 'Negative', 'Negative',
'Negative', 'Negative', 'Further testing required', 'Negative',
'Negative', 'Negative', 'Negative', 'Negative', 'Negative',
'Negative', 'Negative', 'Negative', 'Further testing required',
'Further testing required', 'Further testing required', 'Further
testing required', 'Further testing required', 'Further testing
required', 'Negative', 'Further testing required', 'Negative',
'Negative', 'Negative', 'Negative', 'Further testing required',
'Negative', 'Further testing required', 'Negative', 'Further testing
required', 'Further testing required', 'Negative', 'Negative',
'Negative', 'Negative', 'Negative', 'Further testing required',
'Negative', 'Further testing required', 'Negative', 'Negative',
'Negative', 'Further testing required', 'Further testing required',
'Negative', 'Negative', 'Negative', 'Negative', 'Further testing
required', 'Further testing required', 'Negative', 'Further testing
required', 'Negative', 'Further testing required', 'Negative',
'Negative', 'Negative', 'Negative', 'Further testing required',
'Negative', 'Further testing required', 'Further testing required',
'Negative', 'Further testing required', 'Negative', 'Further testing
required', 'Negative', 'Negative', 'Negative', 'Negative', 'Negative',
'Negative', 'Negative', 'Negative', 'Further testing required',
'Negative', 'Negative', 'Negative', 'Negative', 'Further testing
required', 'Negative', 'Negative', 'Negative', 'Further testing
required', 'Negative', 'Further testing required', 'Negative',
'Further testing required', 'Negative', 'Negative', 'Further testing
required', 'Negative', 'Negative', 'Further testing required',
'Further testing required', 'Further testing required', 'Negative',
'Further testing required', 'Negative', 'Further testing required',
'Further testing required', 'Negative', 'Negative', 'Negative',
```

```
'Negative', 'Further testing required', 'Negative', 'Negative',
'Negative', 'Negative', 'Further testing required', 'Negative',
'Further testing required', 'Negative', 'Negative', 'Negative',
'Negative', 'Negative', 'Negative', 'Negative', 'Negative', 'Further
testing required', 'Further testing required', 'Negative', 'Negative',
'Negative', 'Negative', 'Negative', 'Negative', 'Negative',
'Negative', 'Negative', 'Negative', 'Further testing required',
'Negative', 'Negative', 'Further testing required', 'Negative',
'Further testing required', 'Negative', 'Negative', 'Negative',
'Negative', 'Negative', 'Negative', 'Negative', 'Further testing
required', 'Further testing required', 'Negative', 'Negative',
'Negative', 'Negative', 'Further testing required', 'Further testing
required', 'Further testing required', 'Negative', 'Further testing
required', 'Negative', 'Further testing required', 'Negative',
'Negative', 'Negative', 'Negative', 'Further testing required',
'Further testing required', 'Further testing required', 'Negative',
'Negative', 'Negative', 'Further testing required', 'Negative',
'Negative', 'Negative', 'Further testing required', 'Negative',
'Negative', 'Negative', 'Negative', 'Negative', 'Negative',
'Negative', 'Further testing required']

Ensemble Model:
              precision    recall  f1-score   support

         0.0       0.87      0.97      0.91       155
         1.0       0.58      0.23      0.33        30

    accuracy                           0.85       185
   macro avg       0.73      0.60      0.62       185
weighted avg       0.82      0.85      0.82       185

Accuracy: 0.8486486486486486
Precision: 0.5833333333333334
Recall: 0.23333333333333334
F1-score: 0.33333333333333337
ROC AUC: 0.749247311827957

['ensemble_model1F.pkl']


# Get feature importances
feature_importances =
ensemble_model1F.named_estimators_['rf'].feature_importances_
# Plot feature importances
plt.figure(figsize=(10, 6))
plt.barh(selected_features, feature_importances)
plt.xlabel('Feature Importance')
plt.ylabel('Features')
plt.title('Feature Importance Plot')
plt.show()
```

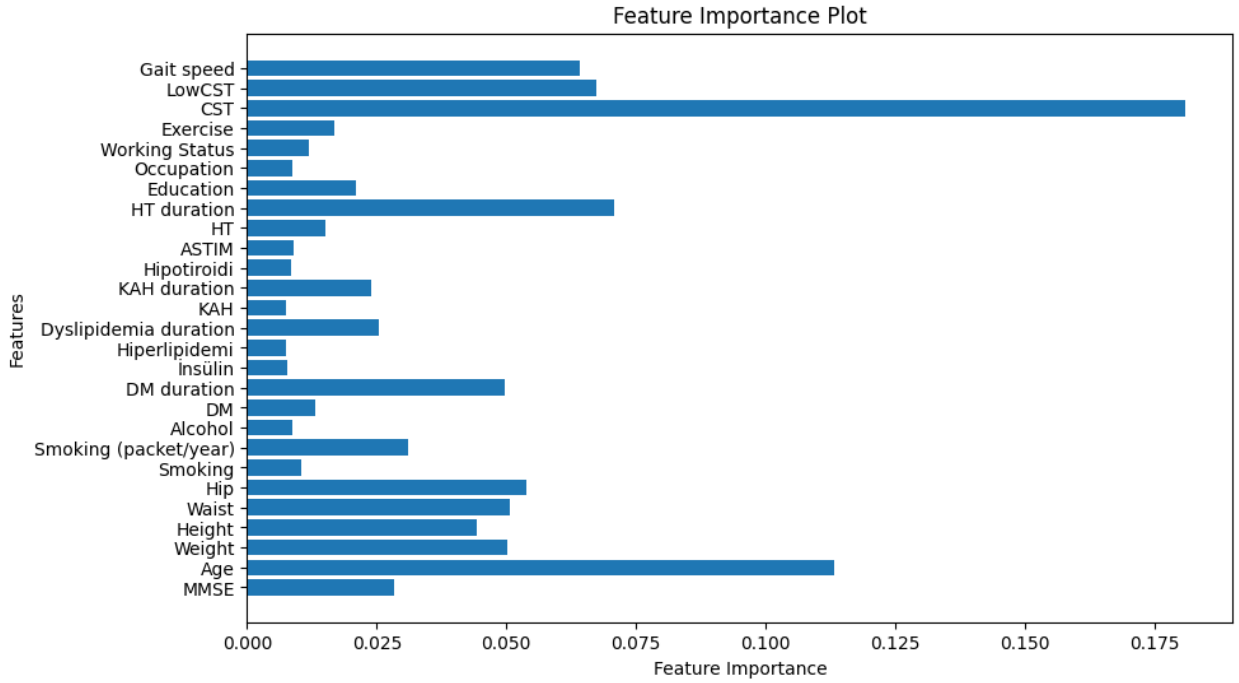



```
# Select features
selected_features = [
    'MMSE', 'Age', 'Weight', 'Height', 'Waist', 'Hip', 'Smoking',
       'Smoking (packet/year)', 'Alcohol', 'DM', 'DM duration',
'İnsülin', 'Hiperlipidemi', 'Dyslipidemia duration','KAH', 'KAH
duration', 'Hipotiroidi', 'ASTIM',
       'HT', 'HT duration','Education', 'Occupation', 'Working
Status', 'Exercise','CST', 'LowCST', 'Gait speed']

# Selecting the features and target variable
X = df_men[selected_features]
y = df_men['SARCOPENIA']

# Create the SimpleImputer object with strategy='mean'
imputer = SimpleImputer(strategy='mean')

# Fit and transform the imputer on the numerical columns with missing
values
X_imputed = imputer.fit_transform(X)

# Splitting the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_imputed, y,
test_size=0.2, random_state=42)

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

```
# Define models
lr_model = LogisticRegression(random_state=42)
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
gb_model = GradientBoostingClassifier(n_estimators=100,
random_state=42)

# Create ensemble model
ensemble_model2M = VotingClassifier(estimators=[
    ('lr', lr_model),
    ('rf', rf_model),
    ('gb', gb_model)
], voting='soft')

# Train ensemble model
ensemble_model2M.fit(X_train_scaled, y_train)

# Predict using ensemble model
ensemble_pred = ensemble_model2M.predict(X_test_scaled)
probabilities = ensemble_model2M.predict_proba(X_test_scaled)[:, 1]
# Diagnose patients
threshold_high = 0.8  # Very high probability threshold
threshold_low = 0.2   # Very low probability threshold

predicted_sarcopenia = ['Positive' if prob >= threshold_high else
'Negative' if prob <= threshold_low else 'Further testing required'
for prob in probabilities]

print(predicted_sarcopenia)

# Evaluate ensemble model
print("\nEnsemble Model:")
print(classification_report(y_test, ensemble_pred))
print("Accuracy:", accuracy_score(y_test, ensemble_pred))
print("Precision:", precision_score(y_test, ensemble_pred))
print("Recall:", recall_score(y_test, ensemble_pred))
print("F1-score:", f1_score(y_test, ensemble_pred))
print("ROC AUC:", roc_auc_score(y_test,
ensemble_model2M.predict_proba(X_test_scaled)[:, 1]))

# Save the trained model
joblib.dump(ensemble_model2M, "ensemble_model2M.pkl",  protocol=4)
```

```
['Negative', 'Negative', 'Further testing required', 'Negative',
'Further testing required', 'Further testing required', 'Further
testing required', 'Negative', 'Further testing required', 'Further
testing required', 'Negative', 'Further testing required', 'Further
testing required', 'Further testing required', 'Negative', 'Negative',
'Negative', 'Negative', 'Negative', 'Negative', 'Negative',
'Negative', 'Negative', 'Negative', 'Negative', 'Negative',
'Negative', 'Negative', 'Negative', 'Negative', 'Negative',
```

```
'Negative', 'Negative', 'Negative', 'Further testing required',
'Further testing required', 'Negative', 'Further testing required',
'Negative', 'Negative', 'Negative', 'Negative', 'Negative', 'Further
testing required', 'Negative', 'Further testing required', 'Further
testing required', 'Negative', 'Further testing required', 'Positive',
'Negative', 'Negative', 'Negative', 'Negative', 'Negative',
'Negative', 'Negative', 'Negative', 'Negative', 'Negative',
'Negative', 'Negative', 'Further testing required', 'Negative',
'Negative', 'Negative', 'Negative', 'Further testing required',
'Further testing required', 'Further testing required', 'Negative',
'Negative', 'Further testing required', 'Negative', 'Negative',
'Negative']

Ensemble Model:
              precision    recall  f1-score   support

         0.0       0.86      0.98      0.92        61
         1.0       0.83      0.33      0.48        15

    accuracy                           0.86        76
   macro avg       0.85      0.66      0.70        76
weighted avg       0.85      0.86      0.83        76

Accuracy: 0.8552631578947368
Precision: 0.8333333333333334
Recall: 0.3333333333333333
F1-score: 0.47619047619047616
ROC AUC: 0.8557377049180328

['ensemble_model2M.pkl']

# Get feature importances
feature_importances =
ensemble_model2M.named_estimators_['rf'].feature_importances_
# Plot feature importances
plt.figure(figsize=(10, 6))
plt.barh(selected_features, feature_importances)
plt.xlabel('Feature Importance')
plt.ylabel('Features')
plt.title('Feature Importance Plot')
plt.show()
```

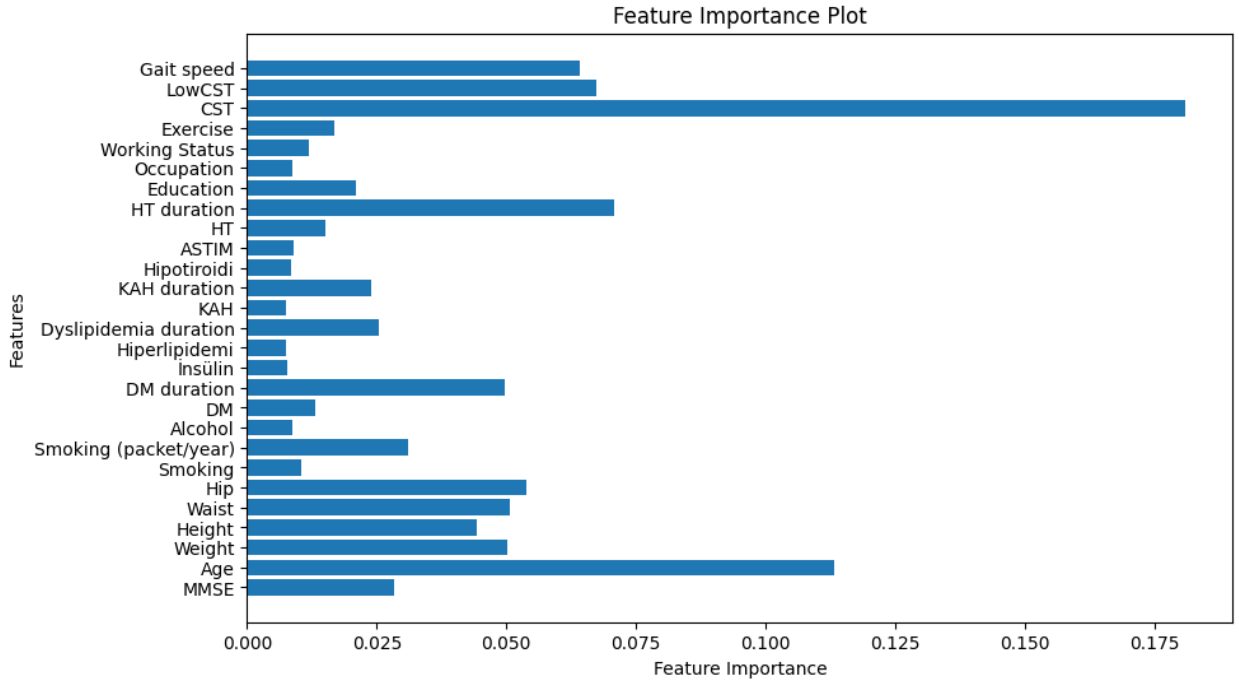



```
# Select features
selected_features = [
    'MMSE', 'Age', 'Weight', 'Height', 'Waist', 'Hip', 'Smoking',
       'Smoking (packet/year)', 'Alcohol', 'DM', 'DM duration',
'İnsülin', 'Hiperlipidemi', 'Dyslipidemia duration','KAH', 'KAH
duration', 'Hipotiroidi', 'ASTIM',
       'HT', 'HT duration','Education', 'Occupation', 'Working
Status', 'Exercise','CST', 'LowCST', 'Gait speed']

# Selecting the features and target variable
X = df_men[selected_features]
y = df_men['SARCOPENIA']

# Create the SimpleImputer object with strategy='mean'
imputer = SimpleImputer(strategy='mean')

# Fit and transform the imputer on the numerical columns with missing
values
X_imputed = imputer.fit_transform(X)

# Splitting the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_imputed, y,
test_size=0.2, random_state=42)

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

```
# Define models
lr_model = LogisticRegression(random_state=42)
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
gb_model = GradientBoostingClassifier(n_estimators=100,
random_state=42)

# Create ensemble model
ensemble_model2F = VotingClassifier(estimators=[
    ('lr', lr_model),
    ('rf', rf_model),
    ('gb', gb_model)
], voting='soft')

# Train ensemble model
ensemble_model2F.fit(X_train_scaled, y_train)

# Predict using ensemble model
ensemble_pred = ensemble_model2F.predict(X_test_scaled)

probabilities = ensemble_model2F.predict_proba(X_test_scaled)[:, 1]
# Diagnose patients
threshold_high = 0.8  # Very high probability threshold
threshold_low = 0.2   # Very low probability threshold

predicted_sarcopenia = ['Positive' if prob >= threshold_high else
'Negative' if prob <= threshold_low else 'Further testing required'
for prob in probabilities]

print(predicted_sarcopenia)

# Evaluate ensemble model
print("\nEnsemble Model:")
print(classification_report(y_test, ensemble_pred))
print("Accuracy:", accuracy_score(y_test, ensemble_pred))
print("Precision:", precision_score(y_test, ensemble_pred))
print("Recall:", recall_score(y_test, ensemble_pred))
print("F1-score:", f1_score(y_test, ensemble_pred))
print("ROC AUC:", roc_auc_score(y_test,
ensemble_model2F.predict_proba(X_test_scaled)[:, 1]))

# Save the trained model
joblib.dump(ensemble_model2F, "ensemble_model2F.pkl",  protocol=4)
```

```
['Negative', 'Negative', 'Further testing required', 'Negative',
'Further testing required', 'Further testing required', 'Further
testing required', 'Negative', 'Further testing required', 'Further
testing required', 'Negative', 'Further testing required', 'Further
testing required', 'Further testing required', 'Negative', 'Negative',
'Negative', 'Negative', 'Negative', 'Negative', 'Negative',
'Negative', 'Negative', 'Negative', 'Negative', 'Negative',
```

```
'Negative', 'Negative', 'Negative', 'Negative', 'Negative',
'Negative', 'Negative', 'Negative', 'Further testing required',
'Further testing required', 'Negative', 'Further testing required',
'Negative', 'Negative', 'Negative', 'Negative', 'Negative', 'Further
testing required', 'Negative', 'Further testing required', 'Further
testing required', 'Negative', 'Further testing required', 'Positive',
'Negative', 'Negative', 'Negative', 'Negative', 'Negative',
'Negative', 'Negative', 'Negative', 'Negative', 'Negative',
'Negative', 'Negative', 'Further testing required', 'Negative',
'Negative', 'Negative', 'Negative', 'Further testing required',
'Further testing required', 'Further testing required', 'Negative',
'Negative', 'Further testing required', 'Negative', 'Negative',
'Negative']

Ensemble Model:
              precision    recall  f1-score   support

         0.0       0.86      0.98      0.92        61
         1.0       0.83      0.33      0.48        15

    accuracy                           0.86        76
   macro avg       0.85      0.66      0.70        76
weighted avg       0.85      0.86      0.83        76

Accuracy: 0.8552631578947368
Precision: 0.8333333333333334
Recall: 0.3333333333333333
F1-score: 0.47619047619047616
ROC AUC: 0.8557377049180328

['ensemble_model2F.pkl']

# Get feature importances
feature_importances =
ensemble_model2F.named_estimators_['rf'].feature_importances_
# Plot feature importances
plt.figure(figsize=(10, 6))
plt.barh(selected_features, feature_importances)
plt.xlabel('Feature Importance')
plt.ylabel('Features')
plt.title('Feature Importance Plot')
plt.show()
```




```
# Select features
selected_features = [
    'MMSE', 'Age', 'Weight', 'Height', 'Waist', 'Hip', 'Smoking',
       'Smoking (packet/year)', 'Alcohol', 'DM', 'DM duration',
'İnsülin', 'Hiperlipidemi', 'Dyslipidemia duration','KAH', 'KAH
duration', 'Hipotiroidi', 'ASTIM',
       'HT', 'HT duration','Education', 'Occupation', 'Working
Status', 'Exercise','CST', 'LowCST', 'Gait speed',
    'Low grip strength', 'Grip strength']

# Selecting the features and target variable
X = df_women[selected_features]
y = df_women['SARCOPENIA']

# Create the SimpleImputer object with strategy='mean'
imputer = SimpleImputer(strategy='mean')

# Fit and transform the imputer on the numerical columns with missing
values
X_imputed = imputer.fit_transform(X)

# Splitting the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_imputed, y,
test_size=0.2, random_state=42)

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

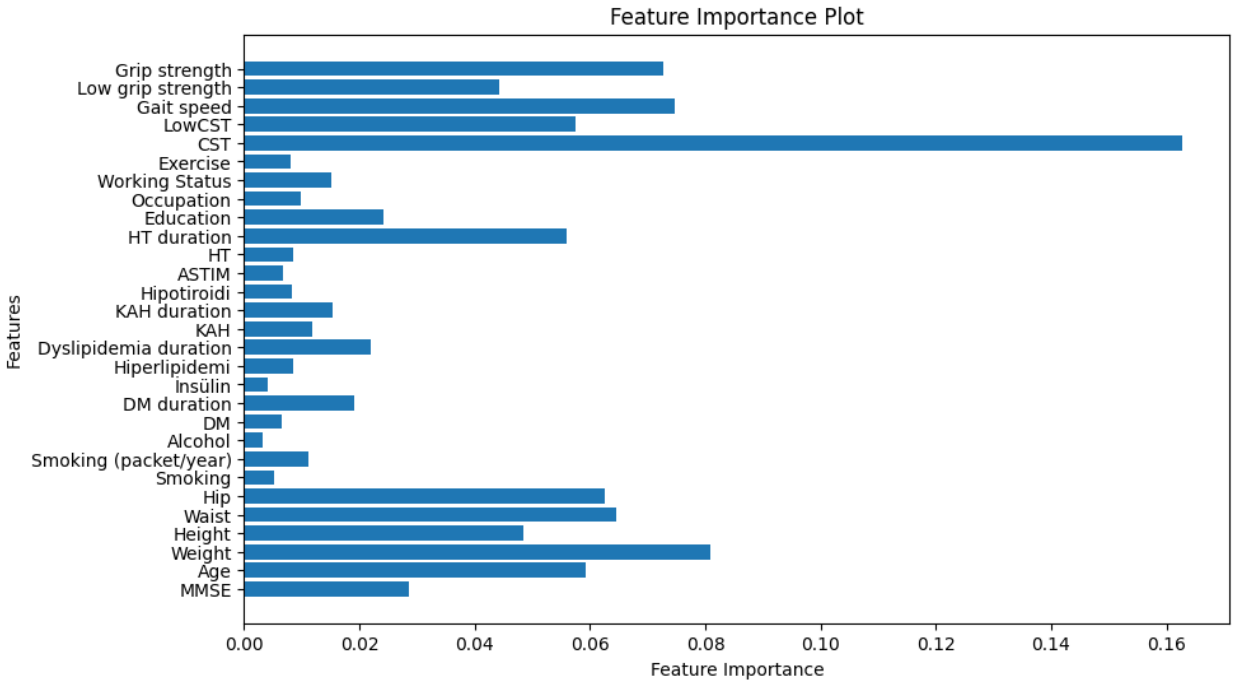
```python
# Define models
lr_model = LogisticRegression(random_state=42)
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
gb_model = GradientBoostingClassifier(n_estimators=100,
random_state=42)

# Create ensemble model
ensemble_model3M = VotingClassifier(estimators=[
    ('lr', lr_model),
    ('rf', rf_model),
    ('gb', gb_model)
], voting='soft')

# Train ensemble model
ensemble_model3M.fit(X_train_scaled, y_train)

# Predict using ensemble model
ensemble_pred = ensemble_model3M.predict(X_test_scaled)

probabilities = ensemble_model3M.predict_proba(X_test_scaled)[:, 1]
# Diagnose patients
threshold_high = 0.8  # Very high probability threshold
threshold_low = 0.2   # Very low probability threshold

predicted_sarcopenia = ['Positive' if prob >= threshold_high else
'Negative' if prob <= threshold_low else 'Further testing required'
for prob in probabilities]

print(predicted_sarcopenia)

# Evaluate ensemble model
print("\nEnsemble Model:")
print(classification_report(y_test, ensemble_pred))
print("Accuracy:", accuracy_score(y_test, ensemble_pred))
print("Precision:", precision_score(y_test, ensemble_pred))
print("Recall:", recall_score(y_test, ensemble_pred))
print("F1-score:", f1_score(y_test, ensemble_pred))
print("ROC AUC:", roc_auc_score(y_test,
ensemble_model3M.predict_proba(X_test_scaled)[:, 1]))

# Save the trained model
joblib.dump(ensemble_model3M, "ensemble_model3M.pkl",  protocol=4)
```

```
['Negative', 'Negative', 'Further testing required', 'Negative',
'Negative', 'Negative', 'Positive', 'Further testing required',
'Negative', 'Negative', 'Negative', 'Negative', 'Negative',
'Negative', 'Negative', 'Negative', 'Negative', 'Further testing
required', 'Positive', 'Further testing required', 'Further testing
required', 'Further testing required', 'Negative', 'Negative',
```

```
'Further testing required', 'Negative', 'Further testing required',
'Negative', 'Positive', 'Negative', 'Further testing required',
'Negative', 'Further testing required', 'Negative', 'Negative',
'Negative', 'Negative', 'Negative', 'Negative', 'Negative',
'Negative', 'Negative', 'Negative', 'Further testing required',
'Negative', 'Further testing required', 'Further testing required',
'Further testing required', 'Negative', 'Negative', 'Negative',
'Negative', 'Further testing required', 'Negative', 'Negative',
'Negative', 'Further testing required', 'Negative', 'Negative',
'Further testing required', 'Negative', 'Negative', 'Negative',
'Further testing required', 'Further testing required', 'Negative',
'Further testing required', 'Negative', 'Negative', 'Negative',
'Negative', 'Negative', 'Negative', 'Negative', 'Negative',
'Negative', 'Negative', 'Further testing required', 'Negative',
'Negative', 'Negative', 'Negative', 'Negative', 'Further testing
required', 'Negative', 'Negative', 'Further testing required',
'Negative', 'Positive', 'Negative', 'Negative', 'Negative', 'Further
testing required', 'Further testing required', 'Negative', 'Negative',
'Negative', 'Negative', 'Positive', 'Negative', 'Further testing
required', 'Negative', 'Further testing required', 'Negative',
'Negative', 'Further testing required', 'Negative', 'Negative',
'Negative', 'Negative', 'Negative', 'Negative', 'Negative', 'Further
testing required', 'Negative', 'Negative', 'Further testing required',
'Negative', 'Negative', 'Negative', 'Negative', 'Negative',
'Negative', 'Negative', 'Negative', 'Negative', 'Negative',
'Negative', 'Negative', 'Negative', 'Negative', 'Further testing
required', 'Negative', 'Negative', 'Negative', 'Negative', 'Negative',
'Negative', 'Negative', 'Further testing required', 'Negative',
'Further testing required', 'Negative', 'Negative', 'Negative',
'Negative', 'Negative', 'Negative', 'Further testing required',
'Negative', 'Further testing required', 'Negative', 'Negative',
'Negative', 'Further testing required', 'Negative', 'Further testing
required', 'Negative', 'Negative', 'Further testing required',
'Negative', 'Further testing required', 'Negative', 'Negative',
'Negative', 'Further testing required', 'Positive', 'Negative',
'Further testing required', 'Negative', 'Negative', 'Negative',
'Further testing required', 'Negative', 'Negative', 'Negative',
'Negative', 'Negative', 'Negative', 'Negative', 'Negative', 'Further
testing required', 'Negative', 'Further testing required', 'Negative']

Ensemble Model:
              precision    recall  f1-score   support

         0.0       0.90      0.97      0.93       155
         1.0       0.72      0.43      0.54        30

    accuracy                           0.88       185
   macro avg       0.81      0.70      0.74       185
weighted avg       0.87      0.88      0.87       185
```

```
Accuracy: 0.8810810810810811
Precision: 0.7222222222222222
Recall: 0.43333333333333335
F1-score: 0.5416666666666666
ROC AUC: 0.924516129032258

['ensemble_model3M.pkl']
```

```
# Get feature importances
feature_importances =
ensemble_model3M.named_estimators_['rf'].feature_importances_
# Plot feature importances
plt.figure(figsize=(10, 6))
plt.barh(selected_features, feature_importances)
plt.xlabel('Feature Importance')
plt.ylabel('Features')
plt.title('Feature Importance Plot')
plt.show()
```



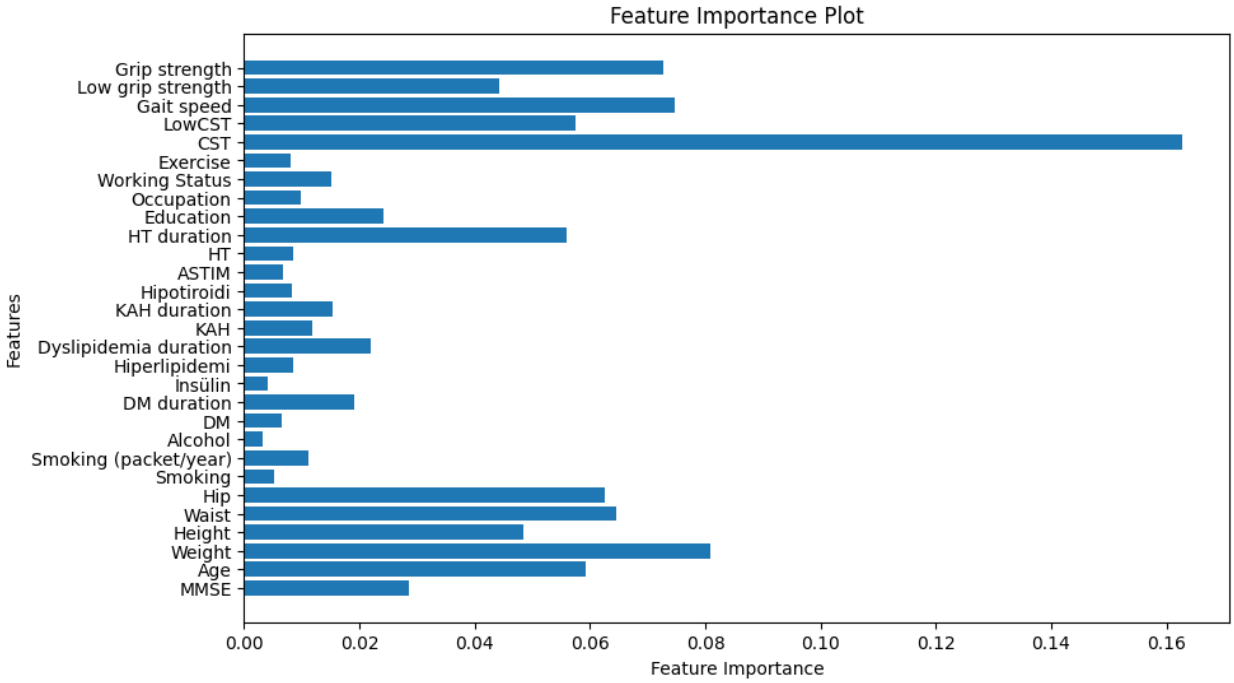

```
# from google.colab import drive
# drive.mount('/content/drive')

# Select features
selected_features = [
    'MMSE', 'Age', 'Weight', 'Height', 'Waist', 'Hip', 'Smoking',
       'Smoking (packet/year)', 'Alcohol', 'DM', 'DM duration',
'İnsülin', 'Hiperlipidemi', 'Dyslipidemia duration','KAH', 'KAH
duration', 'Hipotiroidi', 'ASTIM',
```

```
       'HT', 'HT duration','Education', 'Occupation', 'Working
Status', 'Exercise','CST', 'LowCST', 'Gait speed',
    'Low grip strength', 'Grip strength']

# Selecting the features and target variable
X = df_women[selected_features]
y = df_women['SARCOPENIA']

# Create the SimpleImputer object with strategy='mean'
imputer = SimpleImputer(strategy='mean')

# Fit and transform the imputer on the numerical columns with missing
values
X_imputed = imputer.fit_transform(X)

# Splitting the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_imputed, y,
test_size=0.2, random_state=42)

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

# Define models
lr_model = LogisticRegression(random_state=42)
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
gb_model = GradientBoostingClassifier(n_estimators=100,
random_state=42)

# Create ensemble model
ensemble_model3F = VotingClassifier(estimators=[
    ('lr', lr_model),
    ('rf', rf_model),
    ('gb', gb_model)
], voting='soft')

# Train ensemble model
ensemble_model3F.fit(X_train_scaled, y_train)

# Predict using ensemble model
ensemble_pred = ensemble_model3F.predict(X_test_scaled)
probabilities = ensemble_model3F.predict_proba(X_test_scaled)[:, 1]
# Diagnose patients
threshold_high = 0.8  # Very high probability threshold
threshold_low = 0.2   # Very low probability threshold

predicted_sarcopenia = ['Positive' if prob >= threshold_high else
'Negative' if prob <= threshold_low else 'Further testing required'
for prob in probabilities]
```

```
print(predicted_sarcopenia)

# Evaluate ensemble model
print("\nEnsemble Model:")
print(classification_report(y_test, ensemble_pred))
print("Accuracy:", accuracy_score(y_test, ensemble_pred))
print("Precision:", precision_score(y_test, ensemble_pred))
print("Recall:", recall_score(y_test, ensemble_pred))
print("F1-score:", f1_score(y_test, ensemble_pred))
print("ROC AUC:", roc_auc_score(y_test,
ensemble_model3F.predict_proba(X_test_scaled)[:, 1]))

# Save the trained model
joblib.dump(ensemble_model3F, "ensemble_model3F.pkl",  protocol=4)
```

```
['Negative', 'Negative', 'Further testing required', 'Negative',
'Negative', 'Negative', 'Positive', 'Further testing required',
'Negative', 'Negative', 'Negative', 'Negative', 'Negative',
'Negative', 'Negative', 'Negative', 'Negative', 'Further testing
required', 'Positive', 'Further testing required', 'Further testing
required', 'Further testing required', 'Negative', 'Negative',
'Further testing required', 'Negative', 'Further testing required',
'Negative', 'Positive', 'Negative', 'Further testing required',
'Negative', 'Further testing required', 'Negative', 'Negative',
'Negative', 'Negative', 'Negative', 'Negative', 'Negative',
'Negative', 'Negative', 'Negative', 'Further testing required',
'Negative', 'Further testing required', 'Further testing required',
'Further testing required', 'Negative', 'Negative', 'Negative',
'Negative', 'Further testing required', 'Negative', 'Negative',
'Negative', 'Further testing required', 'Negative', 'Negative',
'Further testing required', 'Negative', 'Negative', 'Negative',
'Further testing required', 'Further testing required', 'Negative',
'Further testing required', 'Negative', 'Negative', 'Negative',
'Negative', 'Negative', 'Negative', 'Negative', 'Negative',
'Negative', 'Negative', 'Further testing required', 'Negative',
'Negative', 'Negative', 'Negative', 'Negative', 'Further testing
required', 'Negative', 'Negative', 'Further testing required',
'Negative', 'Positive', 'Negative', 'Negative', 'Negative', 'Further
testing required', 'Further testing required', 'Negative', 'Negative',
'Negative', 'Negative', 'Positive', 'Negative', 'Further testing
required', 'Negative', 'Further testing required', 'Negative',
'Negative', 'Further testing required', 'Negative', 'Negative',
'Negative', 'Negative', 'Negative', 'Negative', 'Negative', 'Further
testing required', 'Negative', 'Negative', 'Further testing required',
'Negative', 'Negative', 'Negative', 'Negative', 'Negative',
'Negative', 'Negative', 'Negative', 'Negative', 'Negative',
'Negative', 'Negative', 'Negative', 'Negative', 'Further testing
required', 'Negative', 'Negative', 'Negative', 'Negative', 'Negative',
'Negative', 'Negative', 'Further testing required', 'Negative',
```

```
'Further testing required', 'Negative', 'Negative', 'Negative',
'Negative', 'Negative', 'Negative', 'Further testing required',
'Negative', 'Further testing required', 'Negative', 'Negative',
'Negative', 'Further testing required', 'Negative', 'Further testing
required', 'Negative', 'Negative', 'Further testing required',
'Negative', 'Further testing required', 'Negative', 'Negative',
'Negative', 'Further testing required', 'Positive', 'Negative',
'Further testing required', 'Negative', 'Negative', 'Negative',
'Further testing required', 'Negative', 'Negative', 'Negative',
'Negative', 'Negative', 'Negative', 'Negative', 'Further
testing required', 'Negative', 'Further testing required', 'Negative']

Ensemble Model:
              precision    recall  f1-score   support

         0.0       0.90      0.97      0.93       155
         1.0       0.72      0.43      0.54        30

    accuracy                           0.88       185
   macro avg       0.81      0.70      0.74       185
weighted avg       0.87      0.88      0.87       185

Accuracy: 0.8810810810810811
Precision: 0.7222222222222222
Recall: 0.43333333333333335
F1-score: 0.5416666666666666
ROC AUC: 0.924516129032258

['ensemble_model3F.pkl']

# Get feature importances
feature_importances =
ensemble_model3F.named_estimators_['rf'].feature_importances_
# Plot feature importances
plt.figure(figsize=(10, 6))
plt.barh(selected_features, feature_importances)
plt.xlabel('Feature Importance')
plt.ylabel('Features')
plt.title('Feature Importance Plot')
plt.show()
```


Feature Importance Plot
Grip strength
Low grip strength
Gait speed
LowCST
CST
Exercise
Working Status
Occupation
Education
HT duration
HT
ASTIM
Hipotiroidi
KAH duration
KAH
Dyslipidemia duration
Hiperlipidemi
Insülin
DM duration
DM
Alcohol
Smoking (packet/year)
Smoking
Hip
Waist
Height
Weight
Age
MMSE
Features
0.00
0.02
0.04
0.06
0.08
0.10
0.12
0.14
0.16
Feature Importance