In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```
dt = pd.read_excel("Downloads/cardio_data.xlsx")
dt.head()
```

Out[2]:

	id	age	gender	height	weight	ap_hi	ap_lo	cholesterol	gluc	smoke	alco	active	cardio
0	0	18393	2	168	62.0	110	80	1	1	0	0	1	0
1	1	20228	1	156	85.0	140	90	3	1	0	0	1	1
2	2	18857	1	165	64.0	130	70	3	1	0	0	0	1
3	3	17623	2	169	82.0	150	100	1	1	0	0	1	1
4	4	17474	1	156	56.0	100	60	1	1	0	0	0	0

In [3]:
```
dt['gender'] = np.where(dt['gender'] == 1, 0, 1)
dt.head()
```

Out[3]:

	id	age	gender	height	weight	ap_hi	ap_lo	cholesterol	gluc	smoke	alco	active	cardio
0	0	18393	1	168	62.0	110	80	1	1	0	0	1	0
1	1	20228	0	156	85.0	140	90	3	1	0	0	1	1
2	2	18857	0	165	64.0	130	70	3	1	0	0	0	1
3	3	17623	1	169	82.0	150	100	1	1	0	0	1	1
4	4	17474	0	156	56.0	100	60	1	1	0	0	0	0

In [4]:
```
dt.rename(columns={'ap_hi': 'systolic_bp', 'ap_lo': 'diastolic_bp'}, inplace=True)
```

In [4]:
```
dt.rename(columns={'ap_hi': 'systolic_bp', 'ap_lo': 'diastolic_bp'}, inplace=True)
dt.head()
```

Out[4]:

	id	age	gender	height	weight	systolic_bp	diastolic_bp	cholesterol	gluc	smoke	alco	active	cardio
0	0	18393	1	168	62.0	110	80	1	1	0	0	1	0
1	1	20228	0	156	85.0	140	90	3	1	0	0	1	1
2	2	18857	0	165	64.0	130	70	3	1	0	0	0	1
3	3	17623	1	169	82.0	150	100	1	1	0	0	1	1
4	4	17474	0	156	56.0	100	60	1	1	0	0	0	0

In [5]:
```
dt['age_year'] = np.round(dt['age'] / 365)
dt.head()
```

Out[5]:

	id	age	gender	height	weight	systolic_bp	diastolic_bp	cholesterol	gluc	smoke	alco	active	cardio	age_year
0	0	18393	1	168	62.0	110	80	1	1	0	0	1	0	50.0
1	1	20228	0	156	85.0	140	90	3	1	0	0	1	1	55.0
2	2	18857	0	165	64.0	130	70	3	1	0	0	0	1	52.0
3	3	17623	1	169	82.0	150	100	1	1	0	0	1	1	48.0
4	4	17474	0	156	56.0	100	60	1	1	0	0	0	0	48.0

In [6]:
```
dt.info()
```

In [6]:
```
dt.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 70000 entries, 0 to 69999
Data columns (total 14 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   id            70000 non-null  int64  
 1   age           70000 non-null  int64  
 2   gender        70000 non-null  int32  
 3   height        70000 non-null  int64  
 4   weight        70000 non-null  float64
 5   systolic_bp   70000 non-null  int64  
 6   diastolic_bp  70000 non-null  int64  
 7   cholesterol   70000 non-null  int64  
 8   gluc          70000 non-null  int64  
 9   smoke         70000 non-null  int64  
 10  alco          70000 non-null  int64  
 11  active        70000 non-null  int64  
 12  cardio        70000 non-null  int64  
 13  age_year      70000 non-null  float64
dtypes: float64(2), int32(1), int64(11)
memory usage: 7.2 MB
```

In [7]:
```
dt.isna().sum()
```

Out[7]:
```
id              0
age             0
gender          0
height          0
weight          0
systolic_bp     0
diastolic_bp    0
cholesterol     0
gluc            0
smoke           0
alco            0
active          0
cardio          0
age_year        0
dtype: int64
```

In [8]:
```
#checking the variance of variables
```

In [8]:
```
#checking the variance of variables
dt.var()
```

Out[8]:
```
id              8.323976e+08
age             6.087331e+06
gender          2.273745e-01
height          6.740617e+01
weight          2.072378e+02
systolic_bp     2.371952e+04
diastolic_bp    3.552189e+04
cholesterol     4.627405e-01
gluc            3.274933e-01
smoke           8.036307e-02
alco            5.088079e-02
active          1.577512e-01
cardio          2.500035e-01
age_year        4.576920e+01
dtype: float64
```

logistic regression, support vector machines, random forest, and gradient boosting will be used to evaluate on the classification performance. The models will then be combined to develop a weighted ensemble model, capable of leveraging the performance of the disparate models to improve detection accuracy

In [9]:
```
X_dt = dt.drop(columns=['id','age','cardio'], axis=1)
X = X_dt.values
print("Shape X: ", X.shape)

print('\n')

y = dt['cardio'].values
print("Shape y: ", y.shape)
```

```
Shape X:  (70000, 11)


Shape y:  (70000,)
```

In [10]:
```
from sklearn.model_selection import train_test_split
```

In [10]:
```
from sklearn.model_selection import train_test_split

X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.3,random_state=21,stratify=y)
```

Logistic Regression

Some models such as Logistic regression uses some form of distance to inform them so for the above features (explanatory variable) where there are features on larer scales it can disproportionately influence the model. For this reason I want the features to be on same scale using standardizing.

In [11]:
```
from sklearn.preprocessing import StandardScaler
```

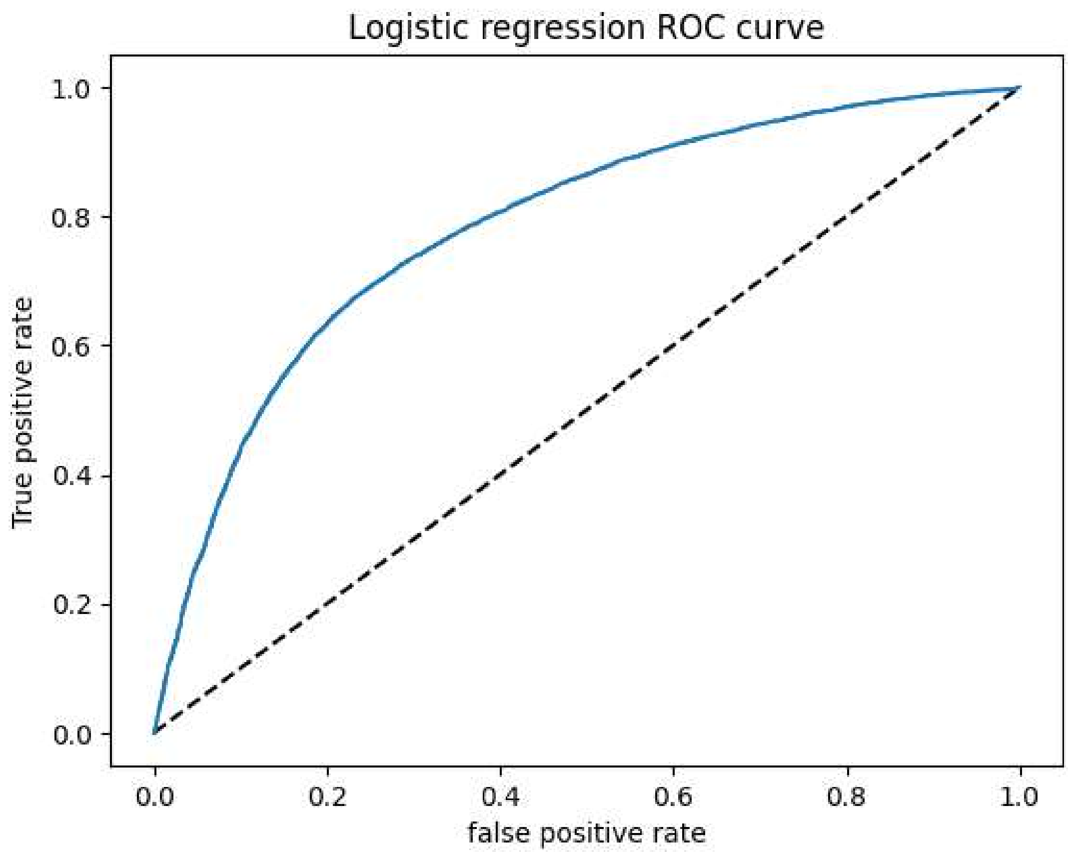
In [11]:
```
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline
from sklearn.metrics import accuracy_score, recall_score, f1_score, precision_score


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

steps = [('scaler', StandardScaler()), ('log', LogisticRegression())]
pipeline = Pipeline(steps)

# Fit the pipeline to the training data
pipeline.fit(X_train_scaled, y_train)

# Make predictions on the test data
y_pred = pipeline.predict(X_test_scaled)

# Calculate and print the accuracy score on the test data
lr_accuracy = pipeline.score(X_test_scaled, y_test)
print("Accuracy on test data:", lr_accuracy)

# Calculate and print the recall score on the test data
lr_recall = recall_score(y_test, y_pred)
print("recall score:", lr_recall)

# Calculate and print the f1 score on the test data
lr_f1 = f1_score(y_test, y_pred)
print("f1 score :", lr_f1)

# Calculate and print the precision score on the test data
lr_precision = precision_score(y_test, y_pred)
print("Precision score:", lr_precision)
```

```
Accuracy on test data: 0.721047619047619
recall score: 0.6778158947970269
f1 score : 0.7083250348536148
Precision score: 0.7417101147028154
```

Logistic regression is used for classifiation problems. If the probability p>0.5, the data is labeled 1, if the probability p < 0.5 the data is labeled 0

In [12]:
```
#predicting probabilities
```

In [12]:
```
#predicting probabilities

y_pred_probs = pipeline.predict_proba(X_test_scaled)[:,1]

#plotting ROC curve
from sklearn.metrics import roc_curve
fpr,tpr,thresholds = roc_curve(y_test, y_pred_probs)
plt.plot([0,1] , [0,1], 'k--')
plt.plot(fpr, tpr)
plt.xlabel("false positive rate")
plt.ylabel("True positive rate")
plt.title('Logistic regression ROC curve')
plt.show()
```



An ROC curve shows the performance of one classification model at all classification thresholds. It can be used to evaluate the strength

An ROC curve shows the performance of one classification model at all classification thresholds. It can be used to evaluate the strength of a model.ROC curves can also be used to compare two models. A classifier that gives curves closer to the top left corner indicates a better performance.

AUC represents the degree or measure of seperability, it tells how much the model is capable of distinguishing between classes. An AUC of 0.5 suggest no discrimmination while 0.7 to 0.8 is consisdered acceptable, 0.8 to 0.9 is considered excellent and more than 0.9 is consisdered outstanding

In [13]:
```
#calculating ROCAUC in scikit learn
from sklearn.metrics import roc_auc_score
log_reg = roc_auc_score(y_test, y_pred_probs)
print("logistic regression auc score:" , log_reg)
```

```
logistic regression auc score: 0.7836148227676519
```

Briefly Note that although we can notice there is a huge difference between the variance of the different columns, from here henceforth, I wont be standardizing or normalizing the variables as I would be using classification tree models because it has ability to describe non-linear dependencies and it does not require preprocessing of varibales before modelling.

RandomForest Classifier

RandomForest Classifier is an ensemble method, it uses a decision tree as a base estimator. the final prediction is made by majority voting.

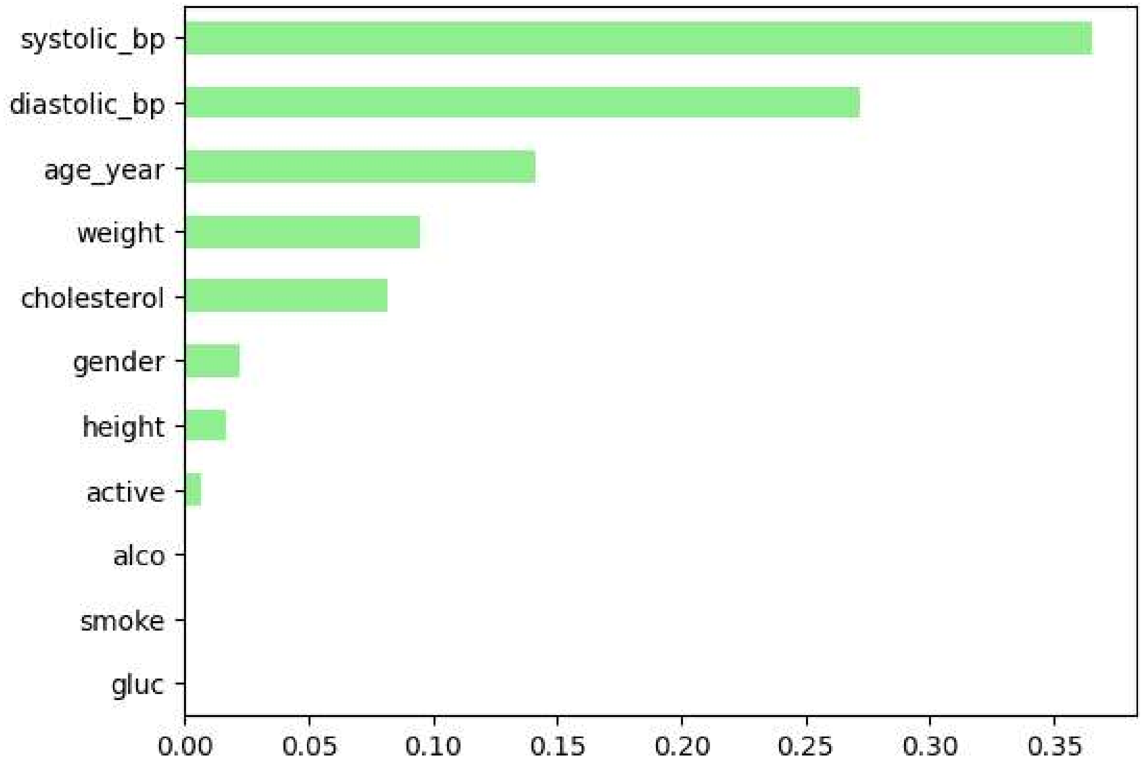
In [14]:
```
# Import necessary libraries
```

In [14]:
```
# Import necessary libraries
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

# Initialize the Random Forest Classifier with desired parameters
rf = RandomForestClassifier(n_estimators=100, min_samples_leaf=0.12, random_state=21)

# Fit the classifier on the training data
rf.fit(X_train, y_train)

# Make predictions on the test data
y_pred = rf.predict(X_test)

# Calculate and print the accuracy score
rf_accuracy = accuracy_score(y_test, y_pred)
print("Random Forest accuracy score:", rf_accuracy)

# Calculate and print the recall score on the test data
rf_recall = recall_score(y_test, y_pred)
print("recall score:", rf_recall)

# Calculate and print the f1 score on the test data
rf_f1 = f1_score(y_test, y_pred)
print("f1 score :", rf_f1)

# Calculate and print the precision score on the test data
rf_precision = precision_score(y_test, y_pred)
print("Precision score:", rf_precision)
```

```
Random Forest accuracy score: 0.7148095238095238
recall score: 0.6407470935772822
f1 score : 0.6918763183618871
Precision score: 0.7518729732751873
```

In [15]:
```
#features importances in sklearn
```

In [15]:
```
#features importances in sklearn

#create a pd.series of features importance
importances_rf = pd.Series(rf.feature_importances_, index=X_dt.columns)
sorted_importances_rf = importances_rf.sort_values()

#barplot
sorted_importances_rf.plot(kind='barh', color='lightgreen')
plt.show()
```



In [16]:
```
#predicting probabilities
```

In [16]:
```
#predicting probabilities

y_pred_probs = rf.predict_proba(X_test)[:,1]

#calculating ROCAUC in scikit learn
from sklearn.metrics import roc_auc_score
rf_auc = roc_auc_score(y_test, y_pred_probs)
print('random forest auc score:', rf_auc)
```

```
random forest auc score: 0.7796844405318809
```

In [17]:
```
from sklearn.metrics import roc_curve
fpr,tpr,thresholds = roc_curve(y_test, y_pred_probs)
rf_fpr = fpr
rf_tpr = tpr

print(rf_fpr, rf_tpr)
```

```
[0.00000000e+00 3.80734818e-04 4.75918523e-04 ... 9.96668570e-01
 9.98667428e-01 1.00000000e+00] [0.         0.00152468 0.00190585 ... 0.99971412 0.99990471 1.        ]
```

Gradient boosting

In gradient boosting each predictor in the ensemblr corrects it predecessor error but the weight of training instances are not changed rather each predecessor is trained using the residual error of its predicessor as labels.

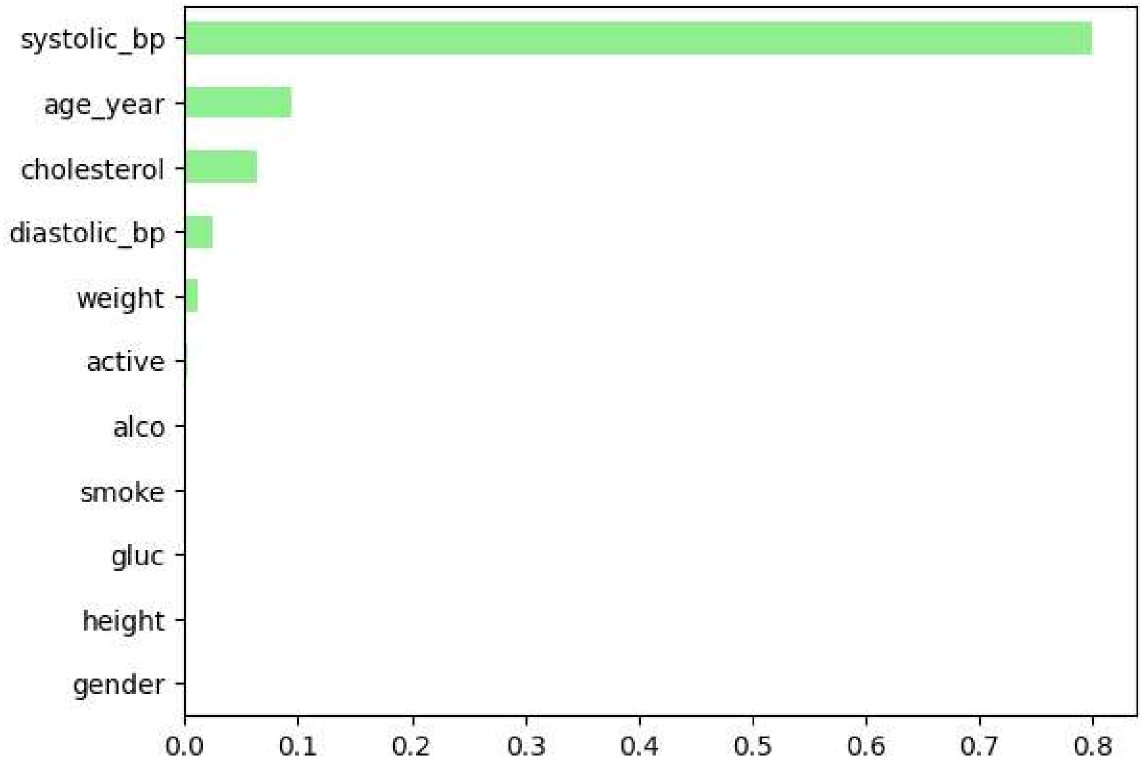
In [18]:
```
from sklearn.ensemble import GradientBoostingClassifier
```

In [18]:
```
from sklearn.ensemble import GradientBoostingClassifier

gbt = GradientBoostingClassifier(n_estimators=100, max_depth=1, random_state=21)
gbt.fit(X_train, y_train)
y_pred = gbt.predict(X_test)

# Calculate and print the accuracy score
gbt_accuracy = accuracy_score(y_test, y_pred)
print("GradientBoosting accuracy score:", gbt_accuracy)

# Calculate and print the recall score on the test data
gbt_recall = recall_score(y_test, y_pred)
print("recall score:", gbt_recall)

# Calculate and print the f1 score on the test data
gbt_f1 = f1_score(y_test, y_pred)
print("f1 score on:", gbt_f1)

# Calculate and print the precision score on the test data
gbt_precision = precision_score(y_test, y_pred)
print("Precision score:", gbt_precision)
```

```
GradientBoosting accuracy score: 0.7272380952380952
recall score: 0.6483704974271012
f1 score on: 0.7037649979313197
Precision score: 0.7695091608233431
```

In [19]:
```
#features importances in sklearn
```

In [19]:
```
#features importances in sklearn

#create a pd.series of features importance
importances_gbt = pd.Series(gbt.feature_importances_, index=X_dt.columns)
sorted_importances_gbt = importances_gbt.sort_values()

#barplot
sorted_importances_gbt.plot(kind='barh', color='lightgreen')
plt.show()
```



In [20]:
```
#predicting probabilities
```

In [20]:
```
#predicting probabilities

y_pred_probs = gbt.predict_proba(X_test)[:,1]

#calculating ROCAUC in scikit learn
from sklearn.metrics import roc_auc_score
gbt_auc = roc_auc_score(y_test, y_pred_probs)
print('gbt_auc_score :', gbt_auc)
```

```
gbt_auc_score : 0.7963984006380265
```

Support Vector Machine

In [21]:
```
from sklearn import svm
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
```

In [22]:
```
# Fit your SVM classifier with probability=False
```

In [22]:
```
# Fit your SVM classifier with probability=False
svm_classifier = SVC(probability=False)
svm_classifier.fit(X_train, y_train)

# Get decision scores instead of probabilities
y_decision_scores = svm_classifier.decision_function(X_test)

# Calculate ROC AUC score
from sklearn.metrics import roc_auc_score
svm_auc = roc_auc_score(y_test, y_decision_scores)
print('svm_auc_score: ', svm_auc)

# Make predictions on the test data
y_pred = svm_classifier.predict(X_test)

# Calculate and print the accuracy score
svm_accuracy = accuracy_score(y_test, y_pred)
print("SVM accuracy score:", svm_accuracy)

# Calculate and print the recall score on the test data
svm_recall = recall_score(y_test, y_pred)
print("recall score:", svm_recall)

# Calculate and print the f1 score on the test data
svm_f1 = f1_score(y_test, y_pred)
print("f1 score on:", svm_f1)

# Calculate and print the precision score on the test data
svm_precision = precision_score(y_test, y_pred)
print("Precision score:", svm_precision)
```

```
svm_auc_score:  0.7858993586610151
SVM accuracy score: 0.7228095238095238
recall score: 0.6234991423670669
f1 score on: 0.6921246099328292
Precision score: 0.777724949482943
```

In [23]:
```
#features importances in sklearn
```

In [23]:
```
#features importances in sklearn
# Create an SVM classifier (you can choose different kernels, such as 'linear' or 'rbf')
svm_classifier = SVC(kernel='linear', C=1)

# Fit the classifier on the training data
svm_classifier.fit(X_train, y_train)

# Make predictions on the test data
y_pred = svm_classifier.predict(X_test)

# Calculate and print the accuracy score
accuracy = accuracy_score(y_test, y_pred)
print("SVM accuracy score:", accuracy)

# Get the coefficients or feature importances from the SVM model
coef = svm_classifier.coef_
# The magnitude of the coefficients can be used as feature importances
feature_importances = np.abs(coef)

# You can sort the feature importances if needed
sorted_indices = np.argsort(feature_importances)

# Print or visualize the feature importances
for idx in sorted_indices:
    print(f"Feature {idx}: Importance {feature_importances[0][idx]}")
```

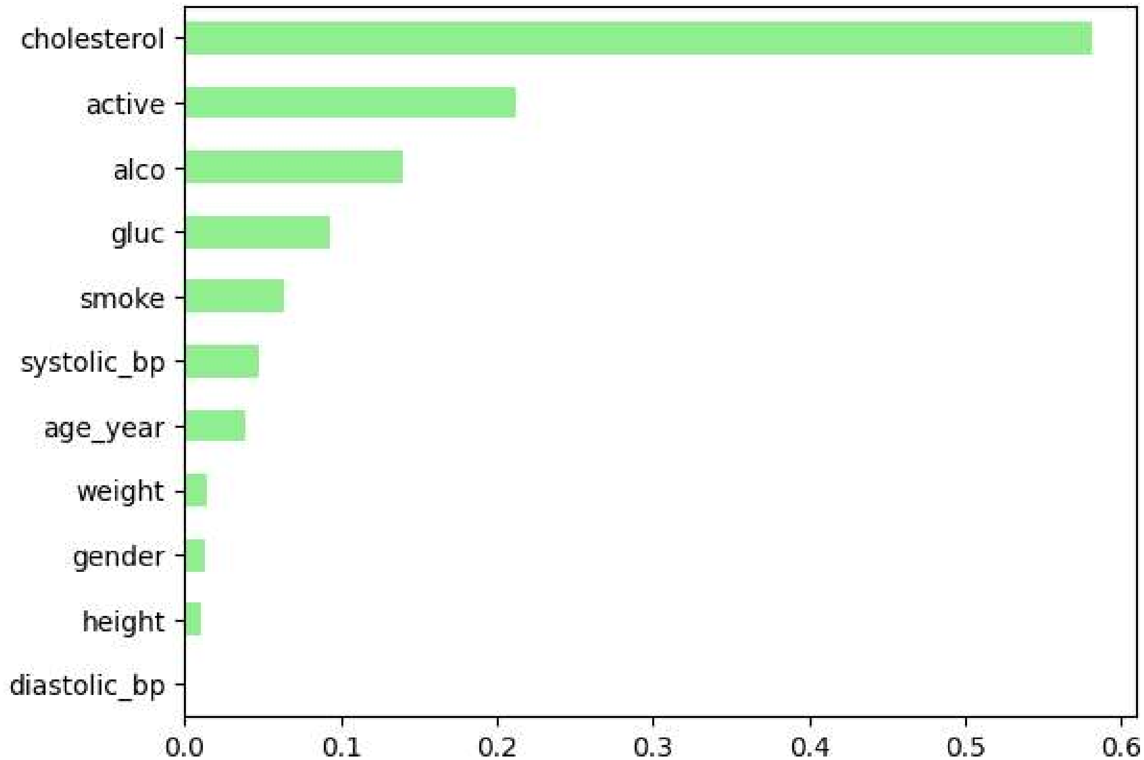
```
SVM accuracy score: 0.7251904761904762
Feature [ 4  1  0  2 10  3  7  6  8  9  5]: Importance [3.29063507e-04 1.04629025e-02 1.33577519e-02 1.38680646
e-02
 3.93293261e-02 4.76891072e-02 6.31112871e-02 9.34503149e-02
 1.39399670e-01 2.12117175e-01 5.80750819e-01]
```

In [24]:
```
# Convert the list element to a NumPy array
result_array = np.array(feature_importances[0])

print(type(result_array))
```

```
<class 'numpy.ndarray'>
```

In [25]:
```
#features importances in sklearn
```

In [25]:
```
#features importances in sklearn

#create a pd.series of features importance
importances_rf = pd.Series(result_array, index=X_dt.columns)
sorted_importances = importances_rf.sort_values()

#barplot
sorted_importances.plot(kind='barh', color='lightgreen')
plt.show()
```



Ensemble Learning

An ensemble learning is a solution that takes the adavantage of CART while reducing their ability to memorize noise.In enesemble learning, different models are trained on thesame dataset and each model are allowed to make its prediction. The final predictions are more robust and less prone to errors.

In [26]:
```
from sklearn import svm
```

In [26]:
```
from sklearn import svm
from sklearn.svm import SVC
from sklearn.ensemble import VotingClassifier

#instantiate the individual classifiers
steps = [('scaler', StandardScaler()), ('log', LogisticRegression())]
pipeline = Pipeline(steps)
#lr = LogisticRegression(random_state=21)
rf = RandomForestClassifier(n_estimators=100, min_samples_leaf=0.12, random_state=21)
gbt = GradientBoostingClassifier(n_estimators=100, max_depth=1, random_state=21)
svm = SVC(probability=False)

#define a list called classifier containing tuples
classifiers = [('Logistic Regression', pipeline), ('RandomForest classifier', rf), ('gradientboosting clf', gbt)

#iterate over the defined list of tuples containing the classifier
for clf_name, clf in classifiers:
  clf.fit(X_train, y_train)
  y_pred = clf.predict(X_test)
  print(clf_name, accuracy_score(y_test, y_pred))
```

```
Logistic Regression 0.721047619047619
RandomForest classifier 0.7148095238095238
gradientboosting clf 0.7272380952380952
Support vector machine 0.7228095238095238
```

Voting Classifier

The voting classifier is an ensemble learning technique where diferent models are used for one training set.

In [27]:
```
#instantiate a voting classifier
```

In [27]:
```
#instantiate a voting classifier
vc = VotingClassifier(estimators=classifiers)

#fit vc to the triaining and test data
vc.fit(X_train, y_train)
y_pred = vc.predict(X_test)

#evaluate the test data accuaracy of vc
vc_accuracy = accuracy_score(y_test, y_pred)
print('Voting classifier accuracy:', vc_accuracy)

# Calculate and print the recall score on the test data
vc_recall = recall_score(y_test, y_pred)
print("recall score:", vc_recall)

# Calculate and print the f1 score on the test data
vc_f1 = f1_score(y_test, y_pred)
print("f1 score on:", vc_f1)

# Calculate and print the precision score on the test data
vc_precision = precision_score(y_test, y_pred)
print("Precision score:", vc_precision)
```

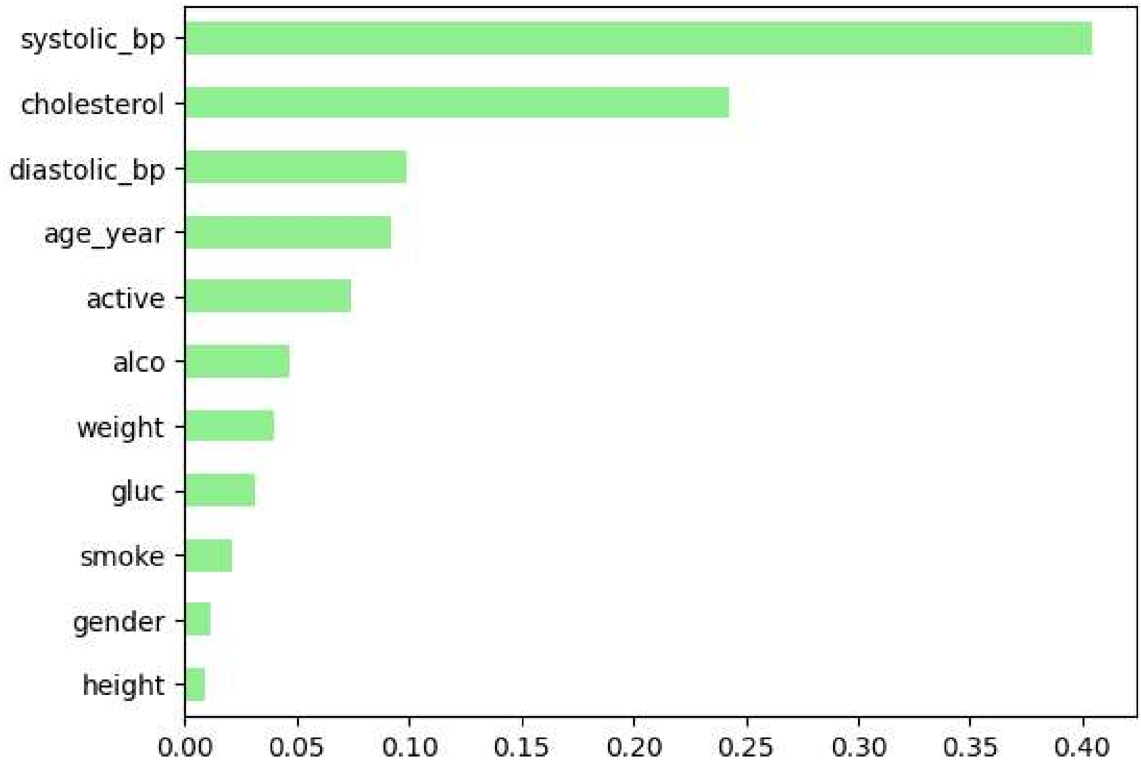
```
Voting classifier accuracy: 0.7218571428571429
recall score: 0.614446350295407
f1 score on: 0.6882638629449752
Precision score: 0.7822394759189616
```

In [28]:
```
# Calculate the combined feature importances
combined_feature_importances = np.average([rf.feature_importances_, gbt.feature_importances_ , result_array], ax

# Print or use the combined feature importances
print("Combined Feature Importances:")
print(combined_feature_importances)
```

```
Combined Feature Importances:
[0.01174477 0.00902243 0.04024718 0.4041874  0.09909567 0.24238381
 0.0311501  0.0210371  0.04659272 0.07406179 0.09176552]
```

In [29]:
```
#features importances in sklearn
```

In [29]:
```
#features importances in sklearn

#create a pd.series of features importance
importances_rf = pd.Series(combined_feature_importances, index=X_dt.columns)
sorted_importances = importances_rf.sort_values()

#barplot
sorted_importances.plot(kind='barh', color='lightgreen')
plt.show()
```



In [30]:
```
#predicting on xtest
```

In [30]:
```
#predicting on xtest

y_pred = vc.predict(X_test)

#calculating ROCAUC in scikit learn
from sklearn.metrics import roc_auc_score
vc_auc = roc_auc_score(y_test, y_pred)
print('vc_auc_score :', vc_auc)
```

```
vc_auc_score : 0.7217958003142749
```

In [31]:
```
test_metrics = pd.DataFrame({
    'model':['Logistic_regression','RandomForest Classifier','GradientBoostin Classifier','SVM Classifier','Ense
    'accuracy':[lr_accuracy,rf_accuracy,gbt_accuracy,svm_accuracy,vc_accuracy],
    'precision':[lr_precision,rf_precision,gbt_precision,svm_precision,vc_precision],
    'recall':[lr_recall,rf_recall,gbt_recall,svm_recall,vc_recall],
    'f1':[lr_f1,rf_f1,gbt_f1,svm_f1,vc_f1],
    'Auc score':[log_reg,rf_auc,gbt_auc,svm_auc,vc_auc]
})

test_metrics.transpose().reset_index().rename(columns={'index':'metrics'})
```

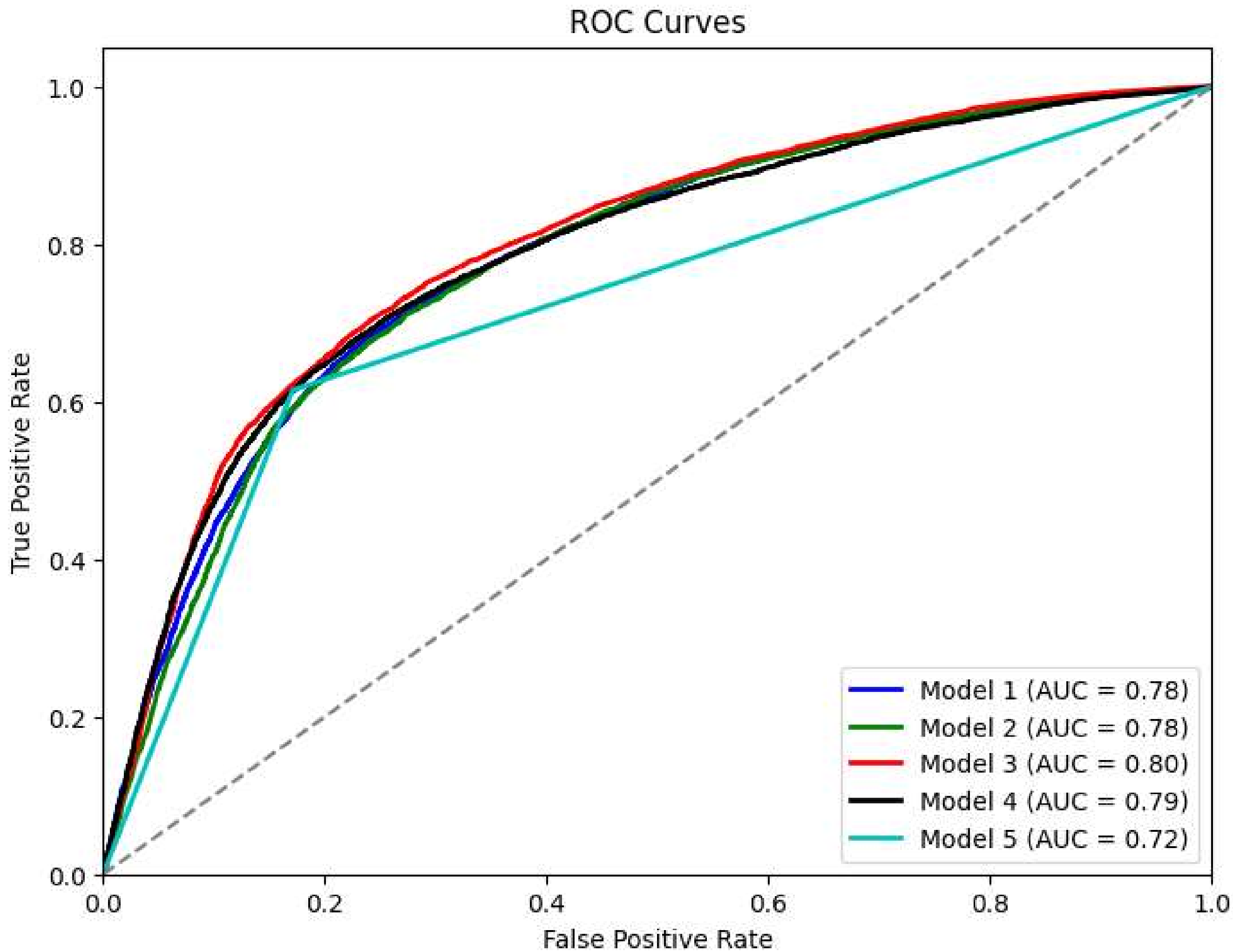
Out[31]:

	metrics	0	1	2	3	4
0	model	Logistic_regression	RandomForest Classifier	GradientBoostin Classifier	SVM Classifier	Ensemble Voting Classifier
1	accuracy	0.721048	0.71481	0.727238	0.72281	0.721857
2	precision	0.74171	0.751873	0.769509	0.777725	0.782239
3	recall	0.677816	0.640747	0.64837	0.623499	0.614446
4	f1	0.708325	0.691876	0.703765	0.692125	0.688264
5	Auc score	0.783615	0.779684	0.796398	0.785899	0.721796

In [32]:
```
svm = SVC(probability=True, random_state=42)
```

In [32]:
```
svm = SVC(probability=True, random_state=42)

# Fit the models
svm.fit(X_train, y_train)

from sklearn.metrics import roc_curve, roc_auc_score

# Calculate ROC curve and ROC AUC for each model
fpr1, tpr1, _ = roc_curve(y_test, pipeline.predict_proba(X_test)[:, 1])
roc_auc1 = roc_auc_score(y_test, pipeline.predict_proba(X_test)[:, 1])

fpr2, tpr2, _ = roc_curve(y_test, rf.predict_proba(X_test)[:, 1])
roc_auc2 = roc_auc_score(y_test, rf.predict_proba(X_test)[:, 1])

fpr3, tpr3, _ = roc_curve(y_test, gbt.predict_proba(X_test)[:, 1])
roc_auc3 = roc_auc_score(y_test, gbt.predict_proba(X_test)[:, 1])

fpr4, tpr4, _ = roc_curve(y_test, svm.predict_proba(X_test)[:, 1])
roc_auc4 = roc_auc_score(y_test, svm.predict_proba(X_test)[:, 1])

fpr5, tpr5, _ = roc_curve(y_test, vc.predict(X_test))
roc_auc5 = roc_auc_score(y_test, vc.predict(X_test))


# Plot ROC curves
plt.figure(figsize=(8, 6))
plt.plot(fpr1, tpr1, color='b', lw=2, label=f'Model 1 (AUC = {roc_auc1:.2f})')
plt.plot(fpr2, tpr2, color='g', lw=2, label=f'Model 2 (AUC = {roc_auc2:.2f})')
plt.plot(fpr3, tpr3, color='r', lw=2, label=f'Model 3 (AUC = {roc_auc3:.2f})')
plt.plot(fpr4, tpr4, color='k', lw=2, label=f'Model 4 (AUC = {roc_auc4:.2f})')
plt.plot(fpr5, tpr5, color='c', lw=2, label=f'Model 5 (AUC = {roc_auc5:.2f})')

plt.plot([0, 1], [0, 1], color='gray', linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curves')
plt.legend(loc='lower right')
plt.show()
```

ROC Curves


ROC Curves
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
Model 1 (AUC = 0.78)
Model 2 (AUC = 0.78)
Model 3 (AUC = 0.80)
Model 4 (AUC = 0.79)
Model 5 (AUC = 0.72)

In [ ]:

In [ ]: