Business analytics homework assignment

Student ID: 710045253

Note: The data used in this assignment: mailing_test & mailing_train (not 1k & 4k)

1. Fit a Random Forest model

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import statsmodels.api as sm
import pandas as pd
from sklearn import datasets
import csv
import pandas as pd
```

Our outcome variable is class. Provide a quick look at the distribution of class for the training data and test data.

In [2]:
```
#Reading .csv file from mailing_train and mailing_test, removing the first
train_df = pd.read_csv('mailing_train.csv')
del train_df[train_df.columns[0]]
test_df = pd.read_csv('mailing_test.csv')
del test_df[test_df.columns[0]]

#Counting the number of donations(class) in mailing train and mailing test
train_class_counts = train_df['class'].value_counts()
print(train_class_counts)

test_class_counts = test_df['class'].value_counts()
print(test_class_counts)
```

```
1    10000
0    10000
Name: class, dtype: int64
0    4738
1     262
Name: class, dtype: int64
```

Comparing the distribution of the outcome variable in training and test, do they look balanced?

The training set is perfectly balanced(50/50). The testing set looks imbalanced, but it won't impact the model training process.

Fit a random forest model to predict class.

In [4]:
```
#define X_train, X_test, y_train, y_test
X_train = train_df.drop('class', axis=1)
y_train = train_df['class']

X_test = test_df.drop('class', axis=1)
y_test = test_df['class']

#X_train.head()
```

In [5]:
```
from sklearn.ensemble import RandomForestClassifier

#instantiate the model
rf = RandomForestClassifier(random_state=0, oob_score = True, n_estimators

#fit the model with data
rf.fit(X_train, y_train)

#evaluate model
def evaluate_model(model, X_test, y_test):
    from sklearn import metrics
    import seaborn as sns

rf_eval =  evaluate_model(rf,X_train,y_train)
```

Examine the results.

In [6]:
```
print('OOB Score', rf.oob_score_)
print('OOB error', 1 - rf.oob_score_)
```

```
OOB Score 0.756
OOB error 0.244
```

What is the out-of-box error rate?

OOB estimate of error rate is 24.4%, showing an effective prediction.

2. Compute and Compare Predictive Performance

Use the *confusionMatrix* function to compute several metrics of predictive performance.

In [7]:

```
def evaluate_model(model, X_test, y_test):
    from sklearn import metrics
    import seaborn as sns
    #making predictions
    y_pred = model.predict(X_test)

    #Computing evaluation metrics
    acc = metrics.accuracy_score(y_test, y_pred)
    prec = metrics.precision_score(y_test, y_pred)
    rec = metrics.recall_score(y_test, y_pred)
    f1 = metrics.f1_score(y_test, y_pred)
    kappa = metrics.cohen_kappa_score(y_test, y_pred)

    #Calculating Area Under Curve (AUC)
    y_pred_proba = model.predict_proba(X_test)[:,1]
    fpr, tpr, _ = metrics.roc_curve(y_test, y_pred_proba)
    auc = metrics.roc_auc_score(y_test, y_pred_proba)

    #Display Confusion Matrix
    cm = metrics.confusion_matrix(y_test, y_pred)

    #Calculating Specificity (TNR) = TN/(TN+FP)
    FP = cm[0][1]
    TN = cm[0][0]
    TNR = TN/(TN+FP)

    return {'acc': acc, 'prec': prec, 'rec': rec, 'f1': f1, 'kappa': kappa
            'cm': cm, 'TNR':TNR}

rf_eval =  evaluate_model(rf,X_train,y_train)
rf_eval_test = evaluate_model(rf, X_test, y_test)

#Print Metrics
print('Accuracy:', rf_eval['acc'])
print('Precision:', rf_eval['prec'])
print('Recall:', rf_eval['rec'])
print('F1 Score:', rf_eval['f1'])
print('Cohens Kappa Score:', rf_eval['kappa'])
print('Area Under Curve:', rf_eval['auc'])
print('Confusion Matrix:\n', rf_eval['cm'])
print('Specificity:', rf_eval['TNR'])
```

```
Accuracy: 0.98125
Precision: 0.9850821489769177
Recall: 0.9773
F1 Score: 0.9811756437929823
Cohens Kappa Score: 0.9625
Area Under Curve: 0.998277995
Confusion Matrix:
 [[9852  148]
 [ 227 9773]]
Specificity: 0.9852
```

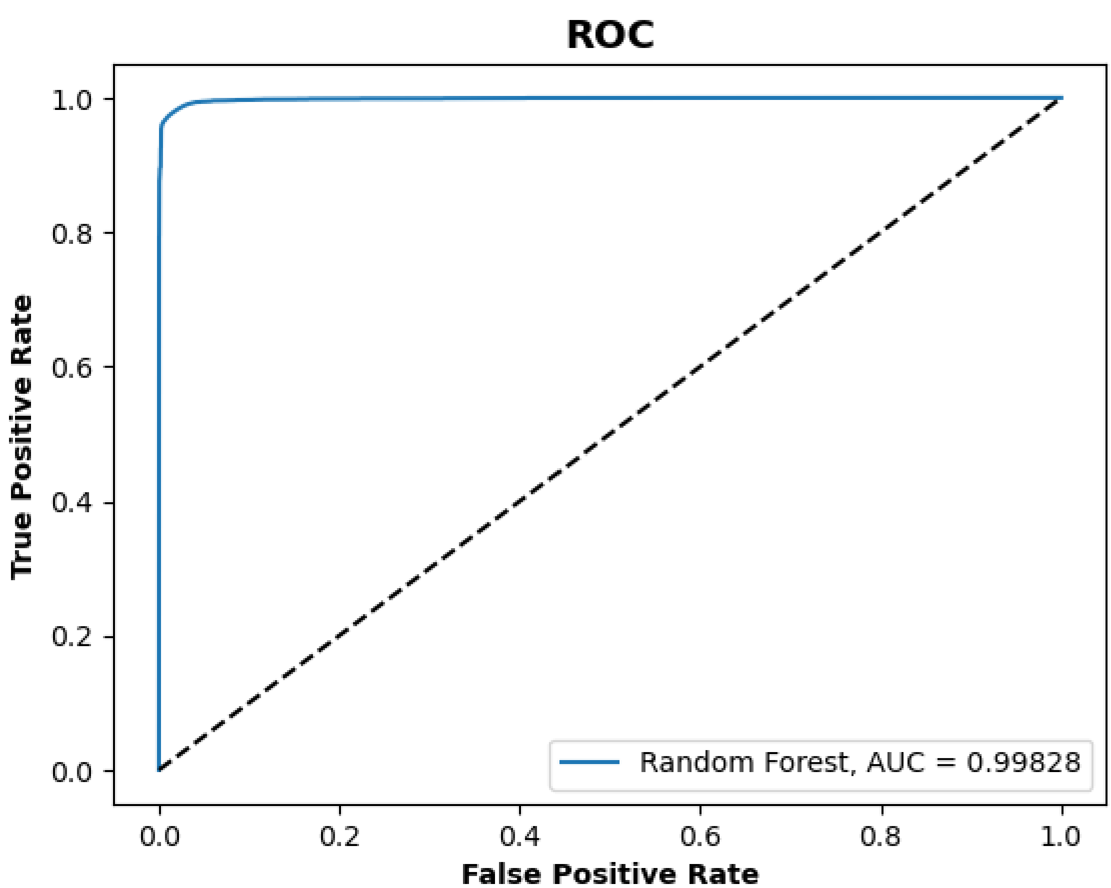
How would you describe the performance of this model?

The accuracy is 98.1%, which is pretty good. The model is more specific(98.52%) than sensitive(97.73%). The model is slightly better at predicting people not giving a donation than people giving donations.

The confusion matrix assumed a 0.50 cutoff for prediction. Now create a ROC plot and compute the AUC for the training set

In [8]:

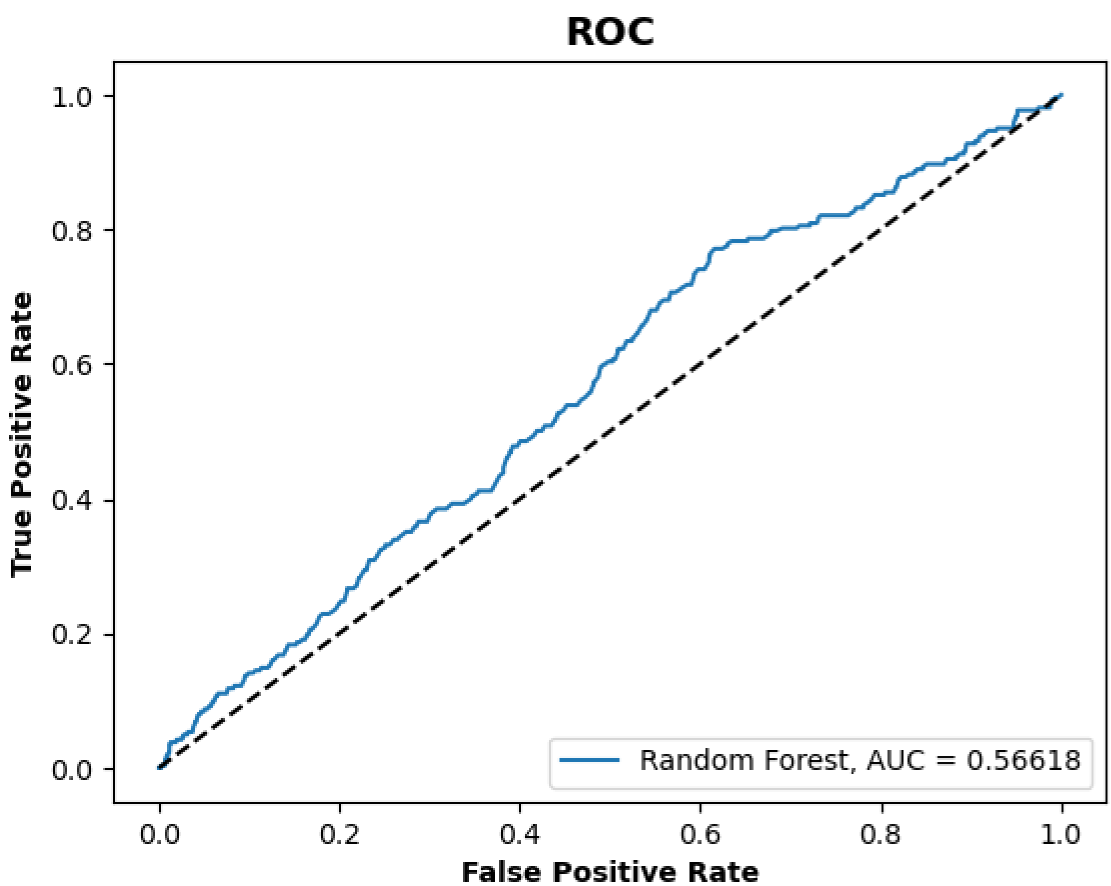
```
#evaluate training set
rf_eval =  evaluate_model(rf,X_train,y_train)
# plot roc curve
plt.plot(rf_eval['fpr'], rf_eval['tpr'],
         label='Random Forest, AUC = {:0.5f}'.format(rf_eval['auc']))

# Configure x and y axis
plt.xlabel('False Positive Rate', fontweight='bold')
plt.ylabel('True Positive Rate', fontweight='bold')
plt.plot([0, 1], [0, 1], 'k--')

# Create legend & title
plt.title('ROC', fontsize=14, fontweight='bold')
plt.legend(loc=4)

plt.show()
```



Now create a ROC chart for the test set and compute the test AUC.

In [9]:

```
# plot roc curve
plt.plot(rf_eval_test['fpr'], rf_eval_test['tpr'],
         label='Random Forest, AUC = {:0.5f}'.format(rf_eval_test['auc']))

# Configure x and y axis
plt.xlabel('False Positive Rate', fontweight='bold')
plt.ylabel('True Positive Rate', fontweight='bold')
plt.plot([0, 1], [0, 1], 'k--')

# Create legend & title
plt.title('ROC', fontsize=14, fontweight='bold')
plt.legend(loc=4)

plt.show()

#Print Metrics
print('Accuracy:', rf_eval_test['acc'])
print('Precision:', rf_eval_test['prec'])
print('Recall:', rf_eval_test['rec'])
print('F1 Score:', rf_eval_test['f1'])
print('Cohens Kappa Score:', rf_eval_test['kappa'])
print('Area Under Curve:', rf_eval_test['auc'])
print('Confusion Matrix:\n', rf_eval_test['cm'])
print('Specificity:', rf_eval_test['TNR'])
```



```
Accuracy: 0.706
Precision: 0.06609195402298851
Recall: 0.3511450381679389
F1 Score: 0.11124546553808949
Cohens Kappa Score: 0.025275469087838065
Area Under Curve: 0.5661760204163834
Confusion Matrix:
 [[3438 1300]
 [ 170   92]]
Specificity: 0.7256226255804137
```

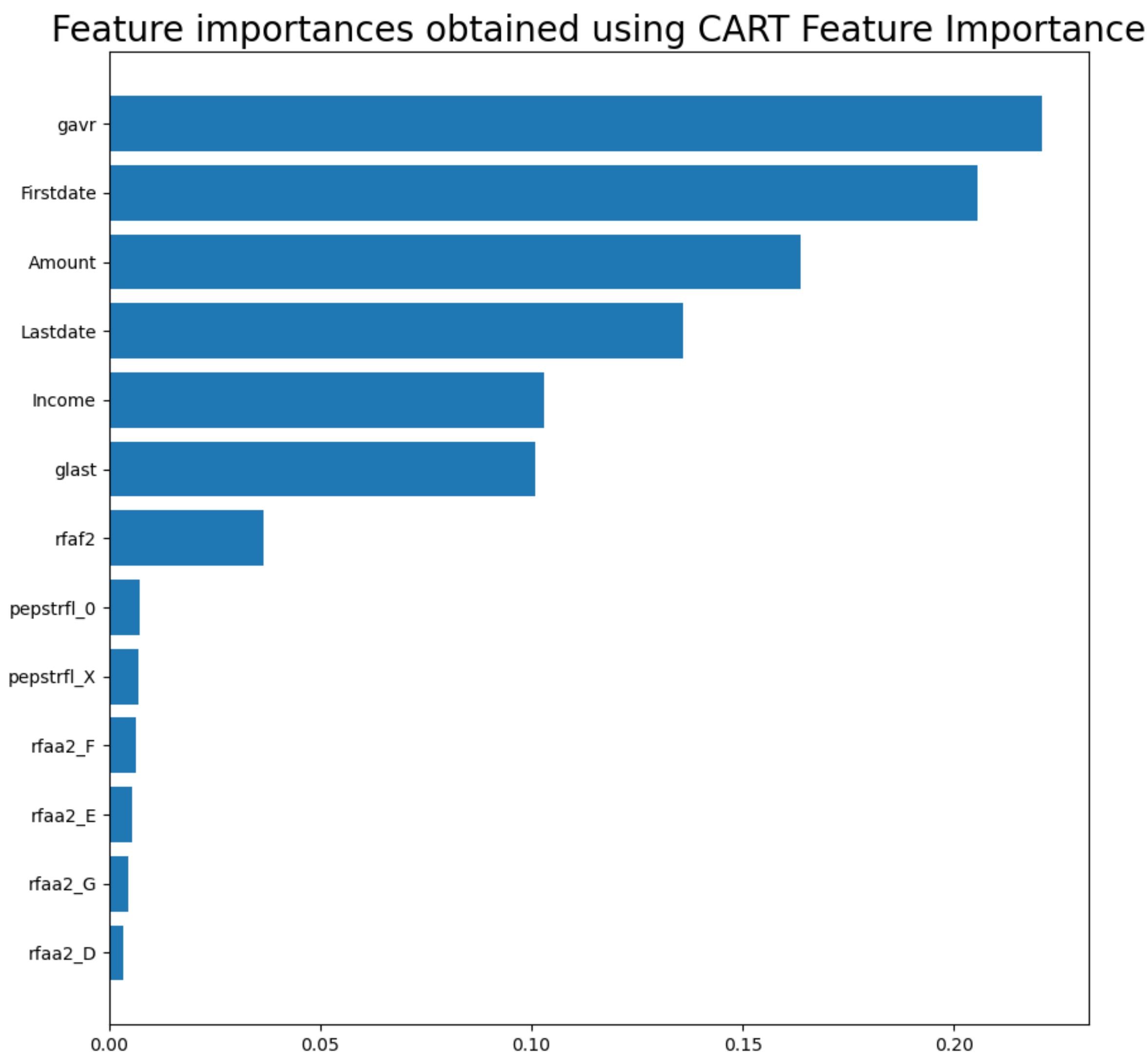
Is the model underfit, overfit, or correctly fit to the data?

The model is overfitto the data. The test-AUC is 0.4321 units lower than the train-AUC, indicating that the model does not correctly fit to the data.

3. Examine the model

Examine the variable importance of the model

In [10]:
```
sorted_idx = rf.feature_importances_.argsort()
plt.figure(figsize=(10,10))
plt.barh(X_train.columns[sorted_idx],rf.feature_importances_[sorted_idx])
plt.title('Feature importances obtained using CART Feature Importance', si
plt.show()
```



Makde some individual predictions of the model. Choose one case from the data and see what the model predicts for that one person.

In [11]:
```
d = X_train.loc[15560] #grabbing the 15559th data
d_reshaped = d.values.reshape(1, -1) #changing to 2-D
rf.predict_proba(d_reshaped)
```

```
D:\Programming\Anaconda\lib\site-packages\sklearn\base.py:450: UserWarning: X
does not have valid feature names, but RandomForestClassifier was fitted with
feature names
  warnings.warn(
```

Out[11]:
```
array([[0.79, 0.21]])
```

Using the most important variable from the plot above, change the value of that variable to something new and make a new prediction for that one case (i.e. set the value to something very small, or very large). How does the prediction change?

In [12]:
```
d['gavr'] = 40
rf.predict_proba(d_reshaped)
```

```
C:\Users\Travis\AppData\Local\Temp\ipykernel_23268\2588936456.py:1: 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 (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  d['gavr'] = 40
D:\Programming\Anaconda\lib\site-packages\sklearn\base.py:450: UserWarning: X does not have valid feature names, but RandomForestClassifier was fitted with feature names
  warnings.warn(
```

Out[12]:
```
array([[0.546, 0.454]])
```

Let's now try and predict the outcome for this case if that important variable was changed from it's minimum value to it's maximum value.

```
1. Create a grid of at least 100 points from the mimimum value to the maximum value
2. Duplicate the case you used above the same number of times.
3. Add the grid of points to the data.
4. Predict the outcome using this new fake data and save the predicted probability in the dataset
```

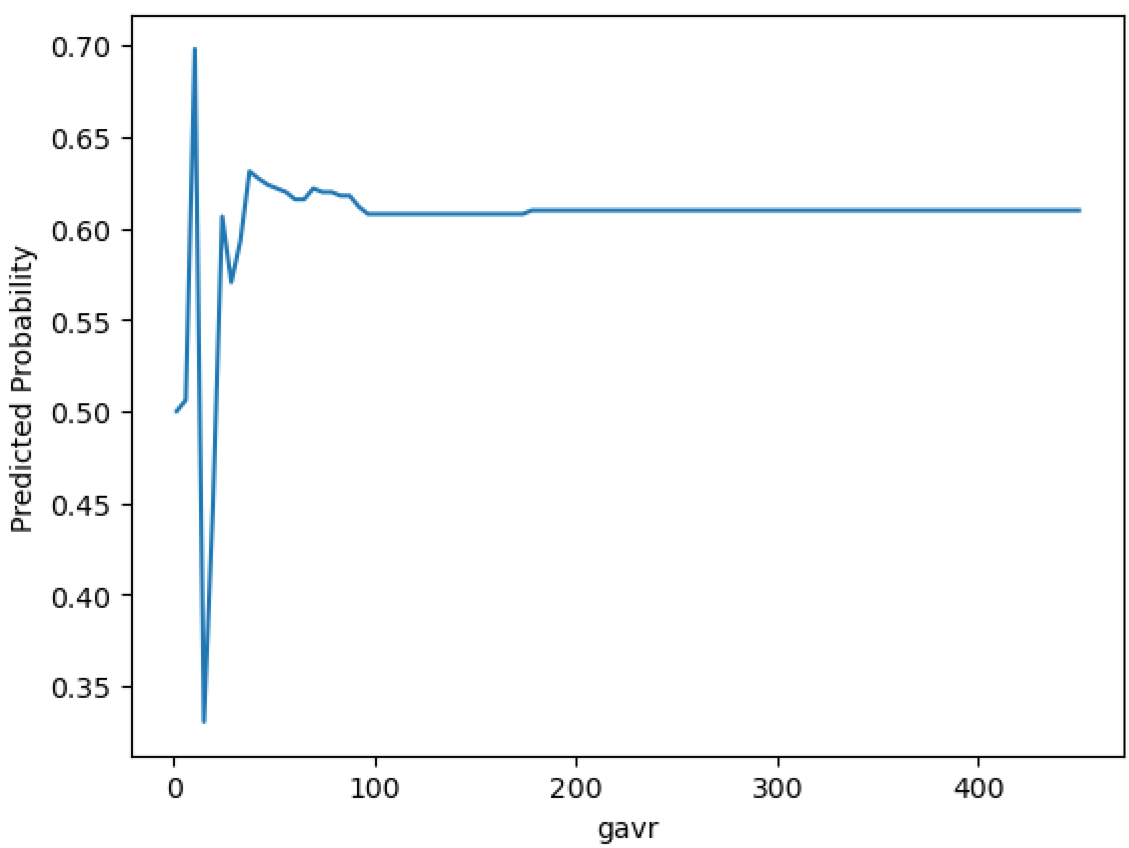
In [19]:

```
# Step 1: Create a grid of at least 100 points from the minimum value to t
N = 100
g = np.linspace(np.min(X_train['gavr']), np.max(X_train['gavr']), num=N)

# Step 2: Duplicate the case you used above the same number of times.
X_train_dup = X_train.loc[[1] * 100, :]

# Step 3: Add the grid of points to the data.
X_train_dup.loc[:, 'gavr'] = g

# Step 4: Predict the outcome using this new fake data and save the predic
probs = rf.predict_proba(X_train_dup)[:, 1]
X_train_dup['predicted_prob'] = probs
X_train_dup = X_train_dup.reset_index(drop=True)

#print(X_train_dup)
#print(probs)
#print(g)
```

Now plot the results of that sequential grid against the predicted probability. How do see the probability of responding the mailer change in response to the variable? Tip: You can use coord_cartesian to change the xlimits to focus on specific areas of detail if you want

In [14]:

```
#Plot the results
plt.plot(X_train_dup['gavr'], X_train_dup['predicted_prob'])
plt.xlabel('gavr')
plt.ylabel('Predicted Probability')
plt.show()
```



Bonus: Use IML to create a PDP+ICE chart

Use the iml package to create an individual conditional expectations combined with partial dependence chart.

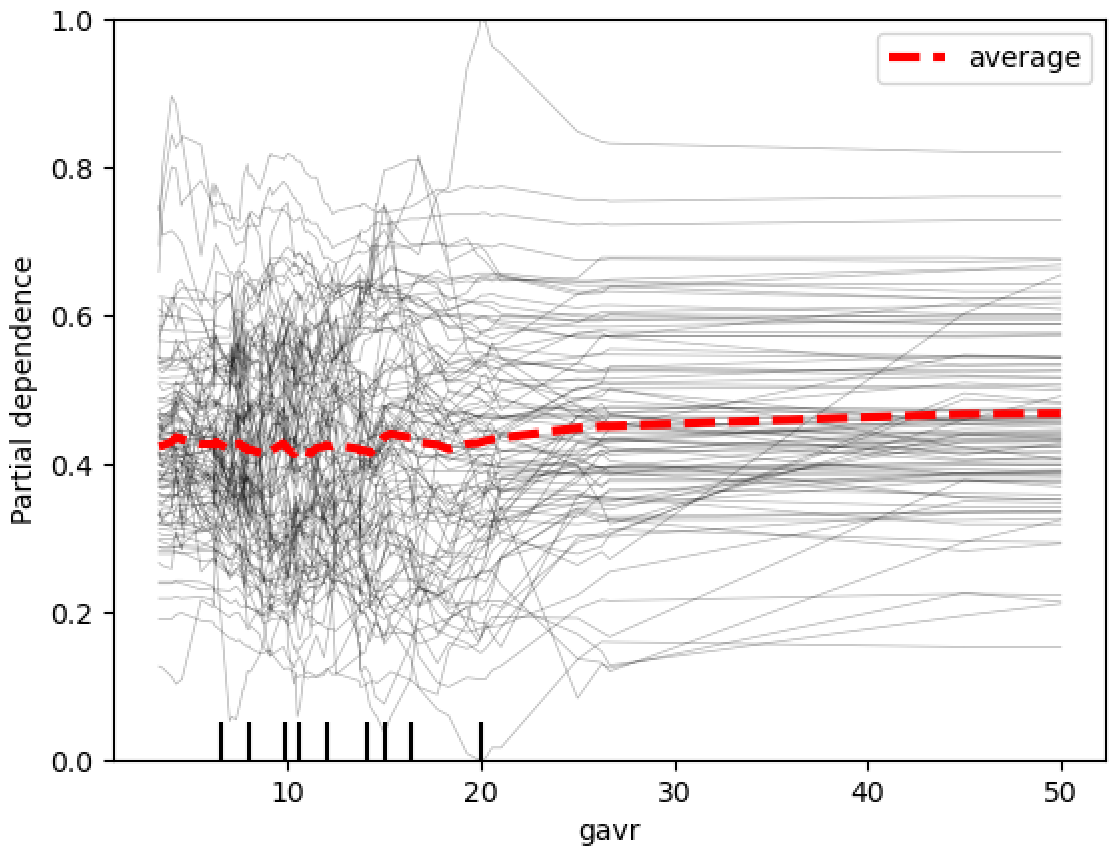
It's highly recommended that you only provide the prediction object a subset of the data. Use *coord_cartesion* to focus the y-limit range to focus the chart to a region where you can observe the effect (it's tiny!).

In [18]:

```
from sklearn.inspection import PartialDependenceDisplay
from sklearn.inspection import partial_dependence

X_test_100 = X_test.sample(n=100)
features = ["gavr"]
PartialDependenceDisplay.from_estimator(rf, X_test_100, features,
                                        kind='both',
                                        ice_lines_kw={"color": "black"},
                                        pd_line_kw={"color": "red","lw":3,
```

Out[18]:

```
<sklearn.inspection._plot.partial_dependence.PartialDependenceDisplay at 0x21
400714310>
```



4. Summarize

What are your thoughts on the impact of these different features on the likelihood for a person to respond to our donation requests? (100-200 words)

Things you could write about:

```
1. What changes would you make to the modeling approach to improve the predictions?
2. What recommendations would you make to the organization?
3. What other kinds of data would be useful in improving these predictions?
```

Summary

According to the code, the model is overfitting to the data. In order to reduce overfitting, it is recommended to regualrize the random forest by finding a good mtry parameter value such that the test and train AUC are more similar, although it could impact training performance, the test performance can be improved.

In order to improve the performacnce of the model, we could consider the model selection. Using a different model other than the random forest model such as the K-nearest neighbours, logistic regression and support vector machines may occur a better result.

People are more likely to continue to donate if the average amount of gift is about 10.6. It is recommended to target people who have given an average amount of 8-12 for a higher chance of receiving a donation. Moreover, people who have an average donation of larger than 37 although has a slightly lower chance than 10.6, still has a moderate chance of giving out a donation. It it also recommended to target people who have an average donation of 37 since it is a relatively high amount and it would be more efficient to target the group as well.

Other kinds of data which would be useful to include are as follows:
Age: certain age groups may be more likely to donate to charities.
Status of the Receiver: Individuals who are married may be more likely to donate to charities.
Education level: Individuals with higher education level may be more likely to donate to charities.
Occupation: Individuals who work in healthcare places such as a hospital may be more likely to donate to charities.
Geographic location: Certain locations may be more charitable than others.