KBS-Assignment3

November 23, 2021

0.1 Exercise 03.01: Scikit Learn, Interpretable Machine Learning

Part 1: Learn about Scikit Learn (Machine Learning in Python, e.g. * https://scikit-learn.org/stable/tutorial/index.html * Video: http://videolectures.net/icml2010_varaquaux_scik/ * Another alternative (*): https://www.youtube.com/watch?v=0Lt9w-BxKFQ

Part 2: Learn about feature importance and SHAP * https://towardsdatascience.com/interpretable-machine-learning-models-aef0c7be3fd9 * https://shap.readthedocs.io/en/latest/index.html * Video: https://www.youtube.com/watch?v=GzEgxFtHH4w

Part 3: Apply Scikit Learn & SHAP * Use the heart disease dataset: https://www.kaggle.com/ronitf/heart-disease-uci * Apply preprocessing on the data as needed, see the *() video, also provide some overview/visualization of the features* Build models using logistic regression, decision trees (C4.5), random forest, neural networks and evaluate them (accuracy, ROC/AUC) * For the evaluation, you can print some tables or use plots * Which are the important features of the model(s)? Use feature importance for the tree-based models. Are there any differences there? * Apply SHAP on all the models, and provide a view on the important features and their impact * Discuss (write some text!) about which model works best, and which is "most interpretable".

[1]:
```
import numpy as np
import pandas as pd

#required packages
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn import svm
from sklearn.neural_network import MLPClassifier
#from sklearn.linear_model import SGDClassifier
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.model_selection import train_test_split
%matplotlib inline

# ... add here
```

```
#Loading dataset
names = ["age","sex","cp",␣
 ↪"trestbps","chol","fbs","restecg","thalach","exang","oldpeak","slope","ca","thal","target"]
heartData = pd.read_csv('./heart.csv')
```

1 Preprocessing

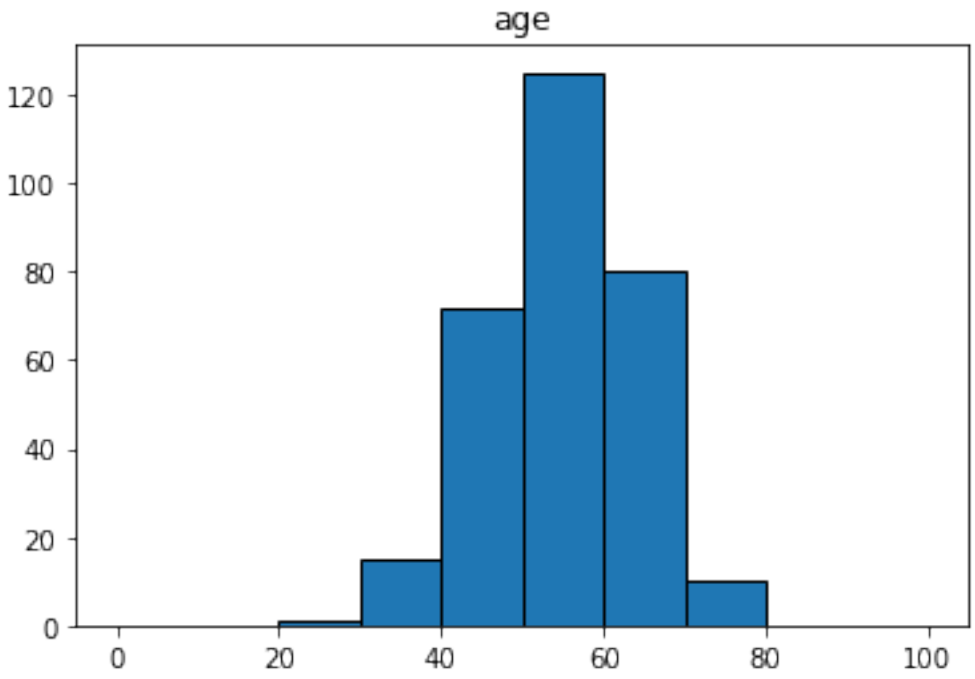
The only Preprocesing shown in (*) was the rearrangement of the wine quality from a value from 0 to 8 to a binary value. This is not needed here.

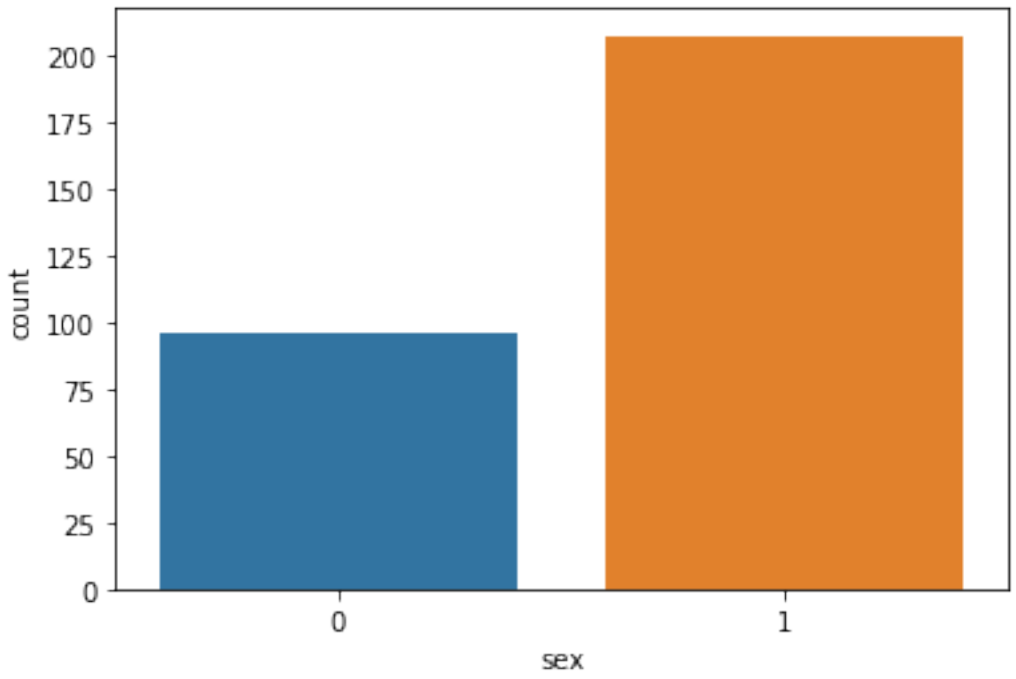
[2]:
```
#preprocessing
#bins = 2
#group_names = ['no_diagnosis','diagnosis']
#heartData['target'] = pd.cut(heartData['target'], bins = bins, labels =␣
 ↪group_names)
#heartData['target'].unique()
#label_sex = LabelEncoder()
#heartData['target'] = label_sex.fit_transform(heartData['target'])
```

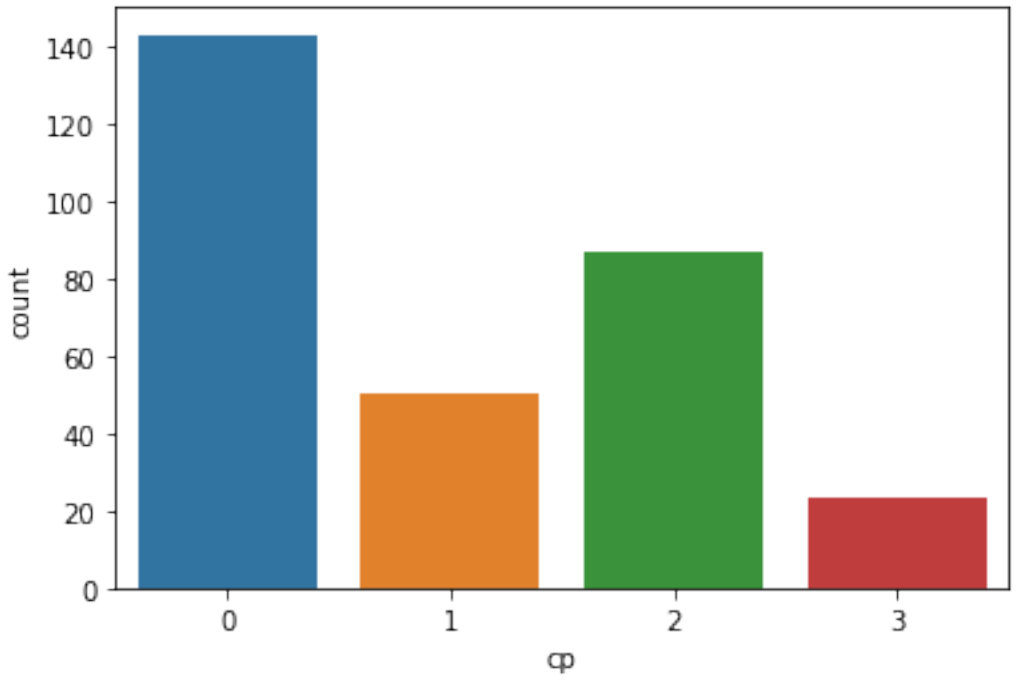
2 Feature Visualization

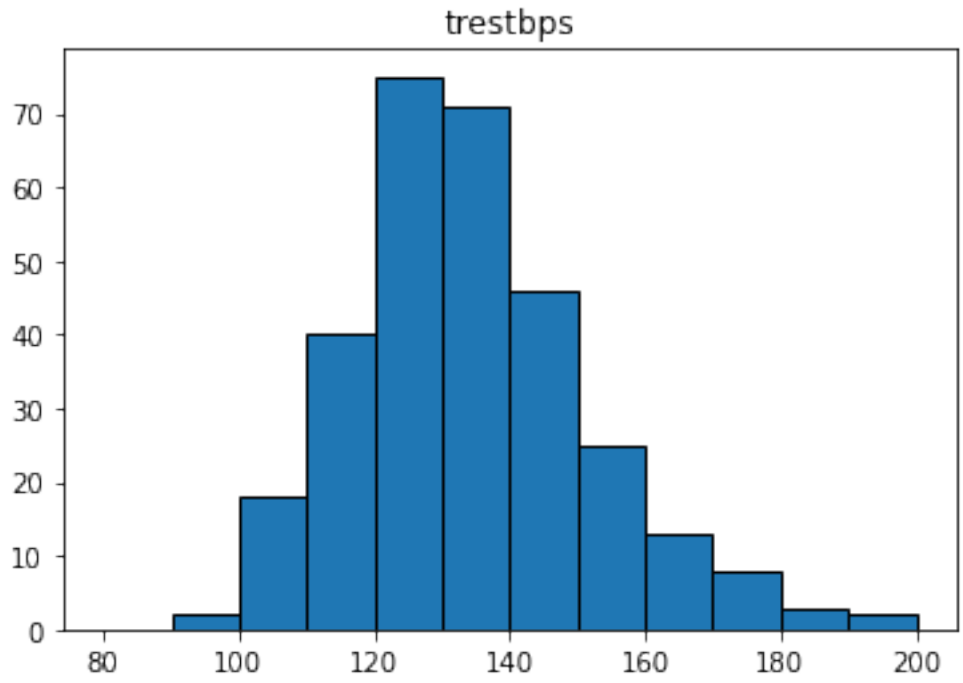
[3]:
```
#Feature Visualization
#plt.show(sns.countplot(x = heartData['age']))
hist, edges = np.histogram(heartData['age'], bins=np.arange(0,101,10))
plt.bar(edges[:-1], hist, align="edge", ec="k", width=np.diff(edges))
plt.title('age')
plt.show()

plt.show(sns.countplot(x = heartData['sex']))
plt.show(sns.countplot(x = heartData['cp']))
#plt.show(sns.countplot(x = heartData['trestbps']))
hist, edges = np.histogram(heartData['trestbps'], bins=np.arange(80,201,10))
plt.bar(edges[:-1], hist, align="edge", ec="k", width=np.diff(edges))
plt.title('trestbps')
plt.show()

#plt.show(sns.countplot(x = heartData['chol']))
hist, edges = np.histogram(heartData['chol'], bins=np.arange(100,400,10))
plt.bar(edges[:-1], hist, align="edge", ec="k", width=np.diff(edges))
plt.title('chol')
plt.show()

plt.show(sns.countplot(x = heartData['fbs']))
plt.show(sns.countplot(x = heartData['restecg']))
#plt.show(sns.countplot(x = heartData['thalach']))
hist, edges = np.histogram(heartData['thalach'], bins=np.arange(50,240,10))
```

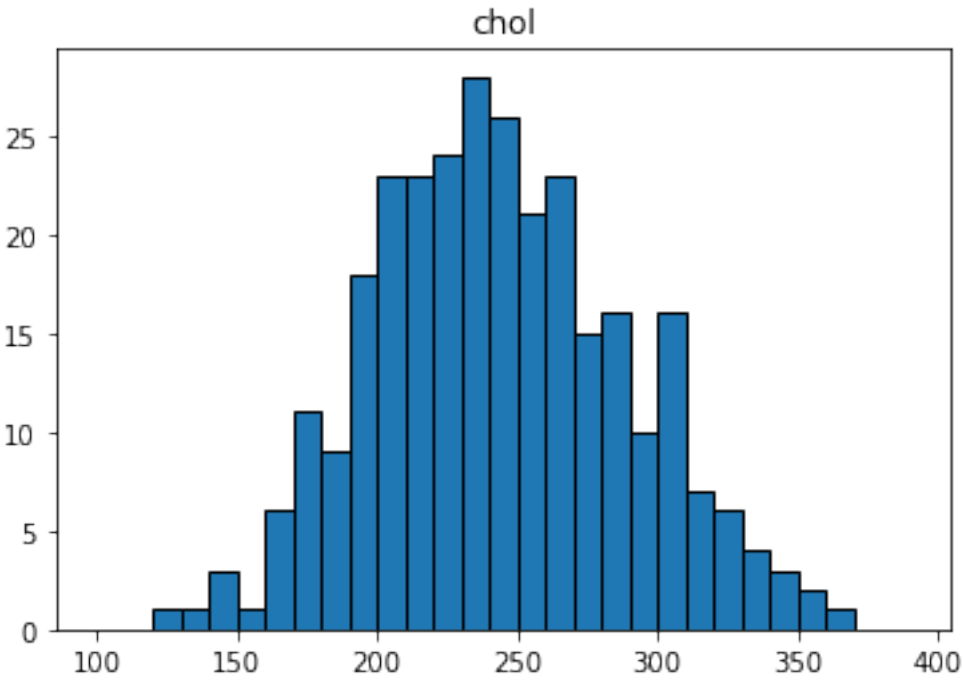
```
plt.bar(edges[:-1], hist, align="edge", ec="k", width=np.diff(edges))
plt.title('thalach')
plt.show()

plt.show(sns.countplot(x = heartData['exang']))
#plt.show(sns.countplot(x = heartData['oldpeak']))
hist, edges = np.histogram(heartData['oldpeak'], bins=np.arange(0,8,0.5))
plt.bar(edges[:-1], hist, align="edge", ec="k", width=np.diff(edges))
plt.title('oldpeak')
plt.show()

plt.show(sns.countplot(x = heartData['slope']))
plt.show(sns.countplot(x = heartData['ca']))
plt.show(sns.countplot(x = heartData['thal']))
plt.show(sns.countplot(x = heartData['target']))
```

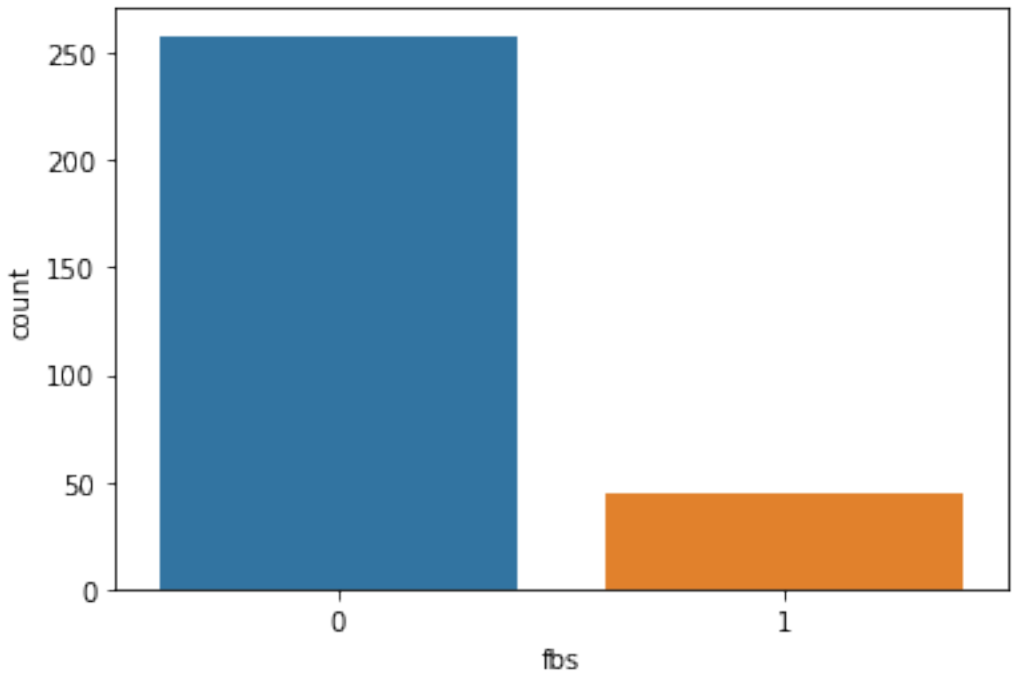


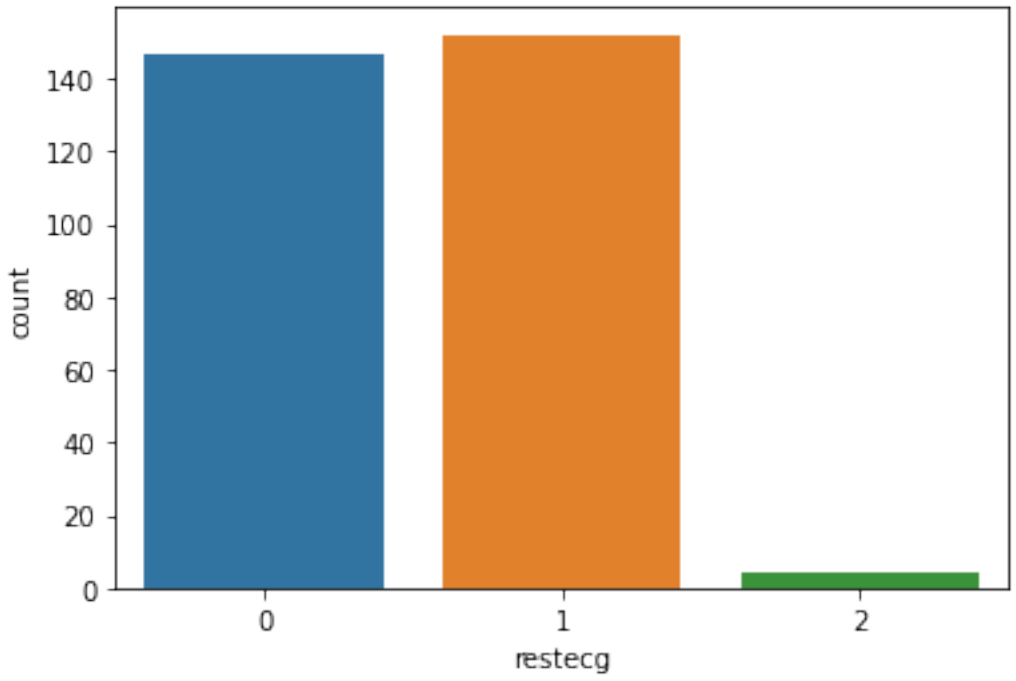

200
175
150
125
100
75
50
25
0
count
0
1
sex

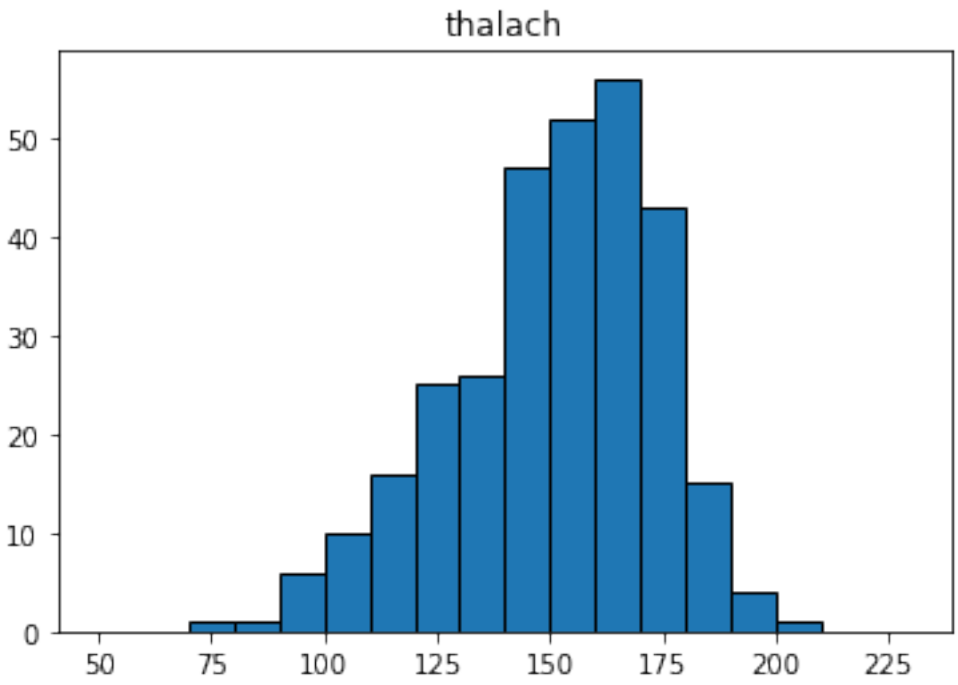

140
120
100
80
60
40
20
0
count
0
1
2
3
cp

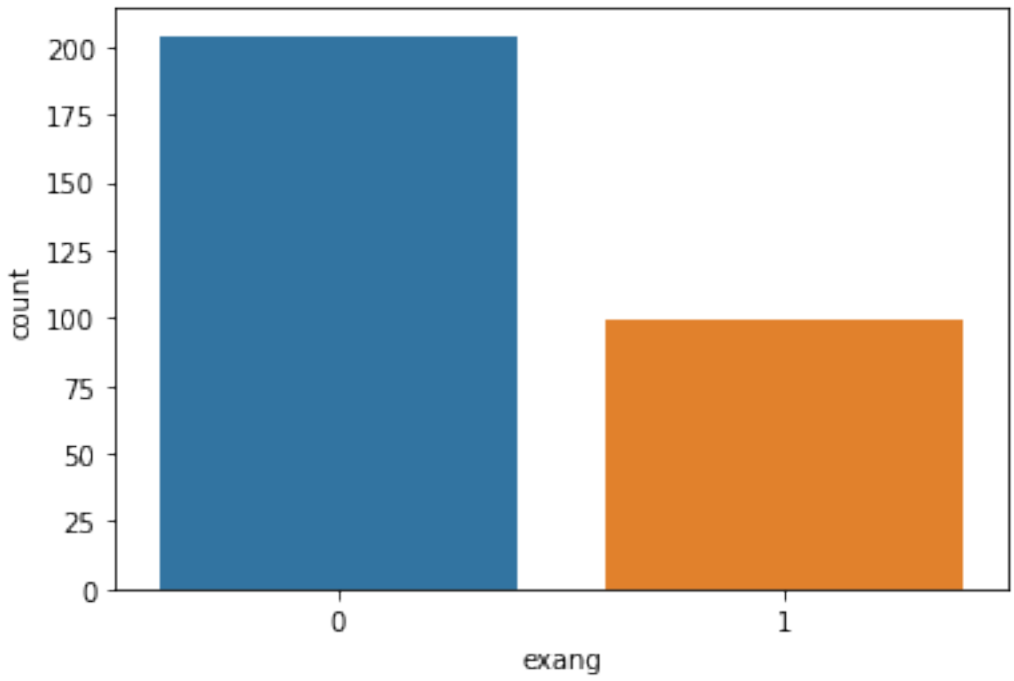

trestbps
70
60
50
40
30
20
10
0
80
100
120
140
160
180
200

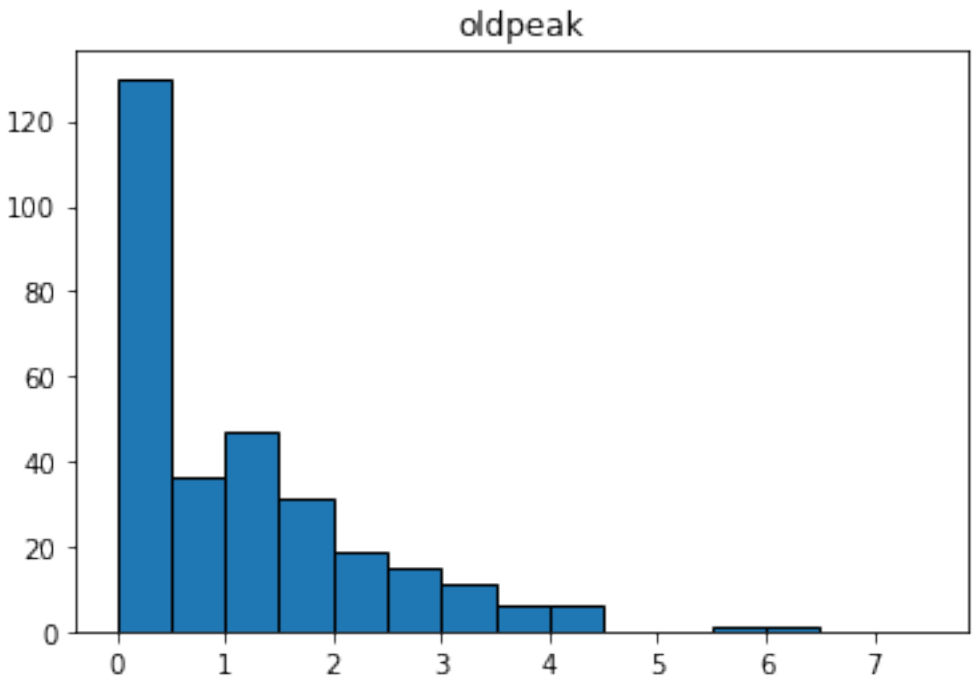

chol
25
20
15
10
5
0
100
150
200
250
300
350
400

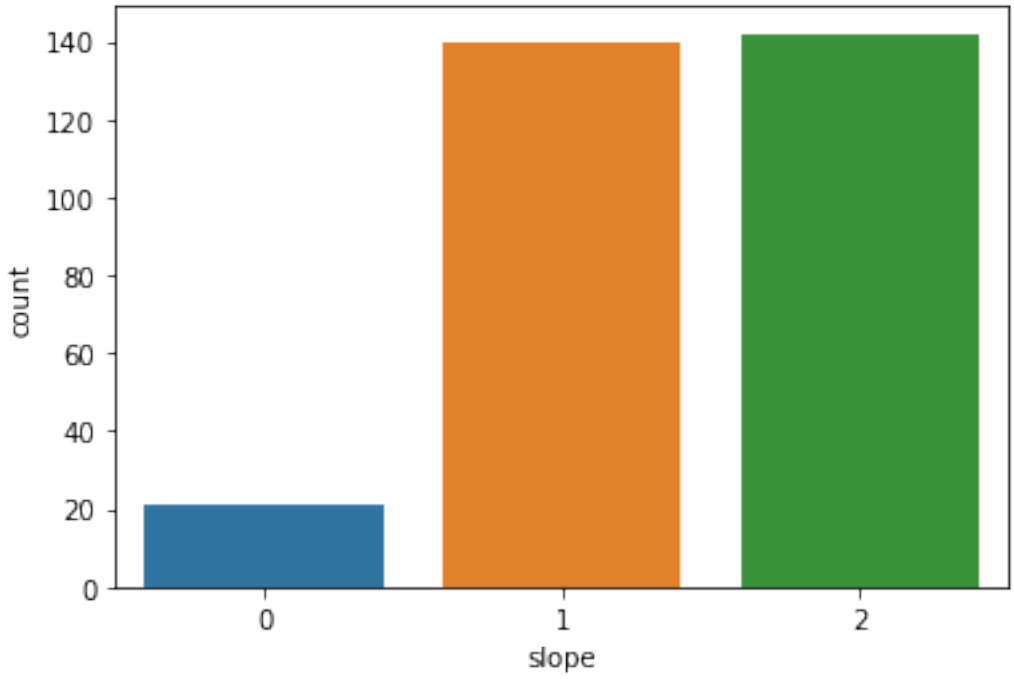

250
200
150
100
50
0
count
0
1
fbs

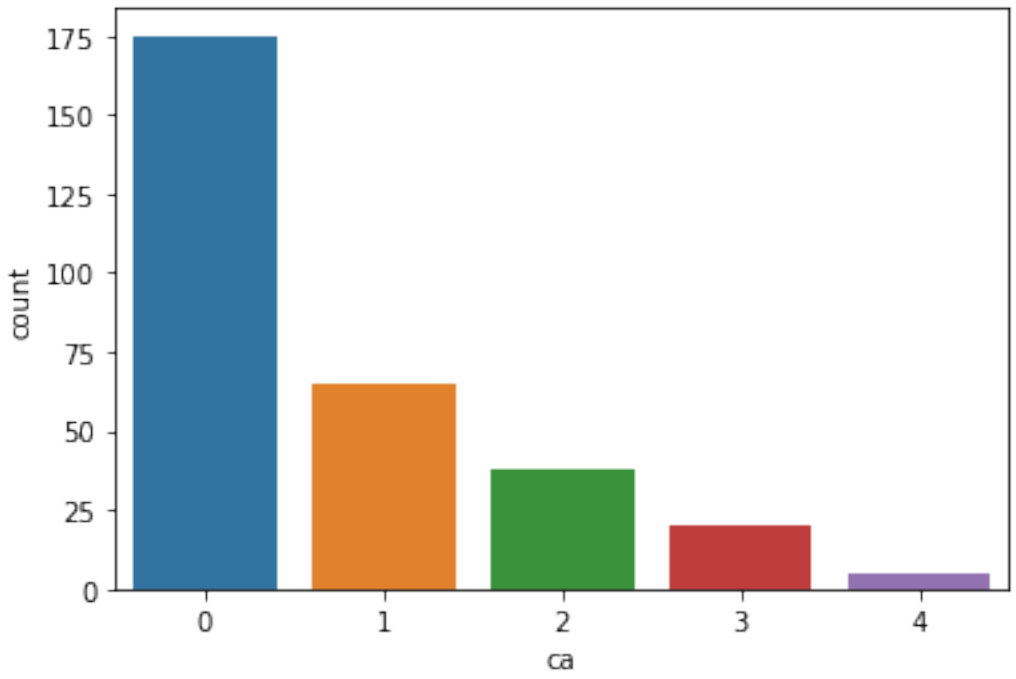

140
120
100
80
60
40
20
0
count
0
1
2
restecg

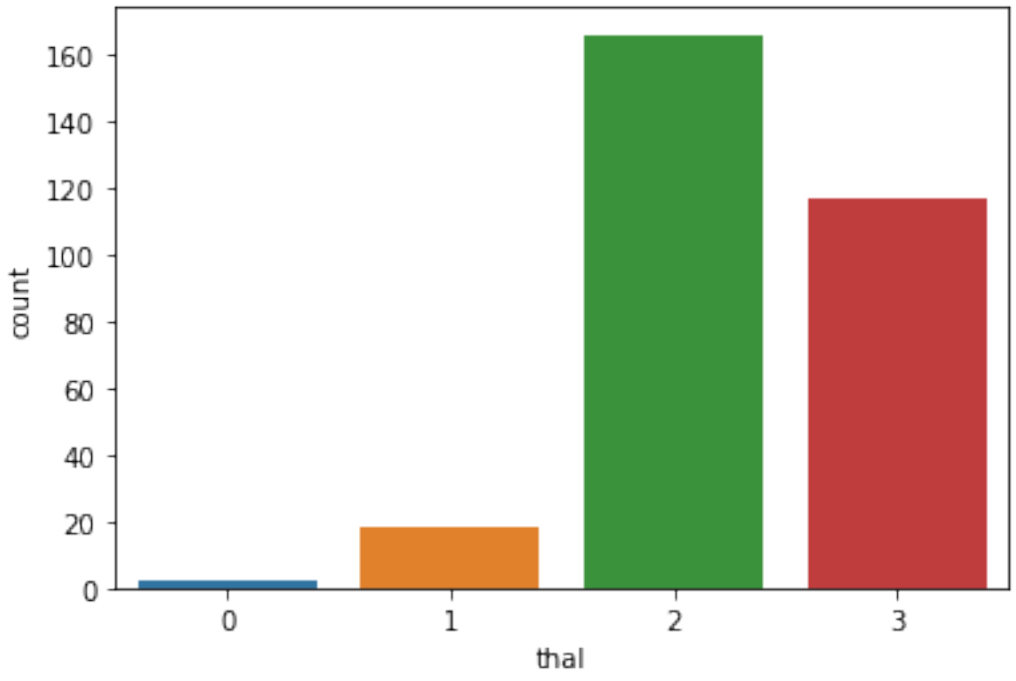

thalach
50
40
30
20
10
0
50
75
100
125
150
175
200
225

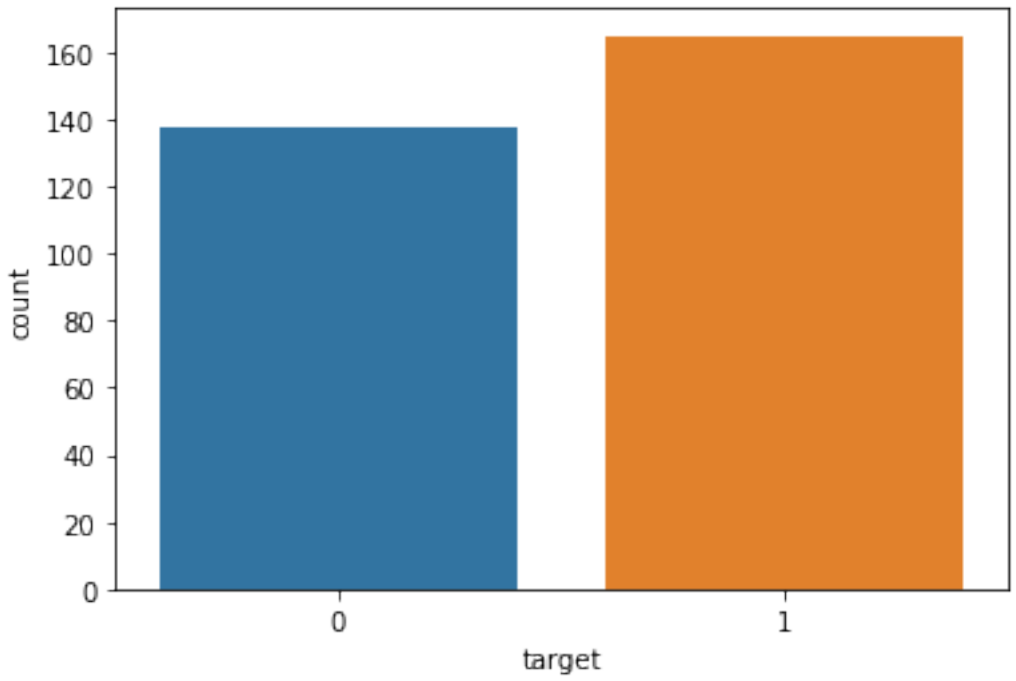

200
175
150
125
100
75
50
25
0
count
0
1
exang


oldpeak
120
100
80
60
40
20
0
0
1
2
3
4
5
6
7


140
120
100
80
60
40
20
0
count
0
1
2
slope


175
150
125
100
75
50
25
0
count
0
1
2
3
4
ca


160
140
120
100
80
60
40
20
0
count
0
1
2
3
thal



3 Random Forest

[4]:
```
heartData.dtypes
heartData.columns
```

[4]:
```
Index(['age', 'sex', 'cp', 'trestbps', 'chol', 'fbs', 'restecg', 'thalach',
       'exang', 'oldpeak', 'slope', 'ca', 'thal', 'target'],
      dtype='object')
```

[5]:
```
#separate features and response variable
X = heartData.drop('target', axis = 1) #all features without target
y = heartData['target']

#standardscaling
sc = StandardScaler()
X = sc.fit_transform(X)
#X_test = sc.fit_transform(X_test)

X = pd.DataFrame(X, columns = names[0:13])
#X_test = pd.DataFrame(X_test, columns = names[0:13])

#split into train and test data
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size =0.2,␣
 ↪random_state=42)
```

[6]:
```
#Apply random forest model
rfc = RandomForestClassifier(n_estimators = 200)
rfc.fit(X_train.values, y_train.values)
pred_rfc = rfc.predict(X_test.values)
```

[7]:
```
#look at performance
from sklearn.metrics import accuracy_score
from sklearn.metrics import roc_auc_score
cm = accuracy_score(y_test, pred_rfc)
print('Accuracy Score Random Forest:  ' + str(cm))
roc = roc_auc_score(y_test, pred_rfc, average=None)
print('ROC/AUC Score Random Forest:  ' + str(roc))
```

```
Accuracy Score Random Forest:  0.8688524590163934
ROC/AUC Score Random Forest:  0.8669181034482758
```

[8]:
```
# save these values for comparison
modelnames = [None] * 4
cml = [None] * 4
rocl = [None] * 4

modelnames[0] = "rf"
cml[0] = cm
rocl[0]= roc
```

4 Neural Network

[9]:
```
mlpc=MLPClassifier(hidden_layer_sizes=(11,11,11),max_iter=1000)
mlpc.fit(X_train.values, y_train.values)
pred_mlpc = mlpc.predict(X_test.values)
```

[10]:
```
#look at performance
cm = accuracy_score(y_test, pred_mlpc)
print('Accuracy Score NN:  ' + str(cm))
roc = roc_auc_score(y_test, pred_mlpc, average=None)
print('ROC/AUC Score NN:  ' + str(roc))
```

```
Accuracy Score NN:  0.8360655737704918
ROC/AUC Score NN:  0.8389008620689655
```

[11]:
```
modelnames[1] = "nn"
cml[1] = cm
rocl[1]= roc
```

5 Decision Trees

[12]:
```
from sklearn.tree import DecisionTreeClassifier
dtc = DecisionTreeClassifier(random_state=0).fit(X_train.values, y_train.values)
pred_dtc = dtc.predict(X_test.values)
```

[13]:
```
#look at performance
cm = accuracy_score(y_test, pred_dtc)
print('Accuracy Score Decision Trees:  ' + str(cm))
roc = roc_auc_score(y_test, pred_dtc, average=None)
print('ROC/AUC Score Decision Trees:  ' + str(roc))
```

```
Accuracy Score Decision Trees:  0.7868852459016393
ROC/AUC Score Decision Trees:  0.7904094827586206
```

[14]:
```
modelnames[2] = "dt"
cml[2] = cm
rocl[2]= roc
```

6 Logistic Regression

[15]:
```
from sklearn.linear_model import LogisticRegression
lgr = LogisticRegression(random_state=0).fit(X_train.values, y_train.values)
pred_lgr = lgr.predict(X_test.values)
```

[16]:
```
#look at performance
cm = accuracy_score(y_test, pred_lgr)
print('Accuracy Score NN:  ' + str(cm))
roc = roc_auc_score(y_test, pred_lgr, average=None)
print('ROC/AUC Score NN:  ' + str(roc))
```

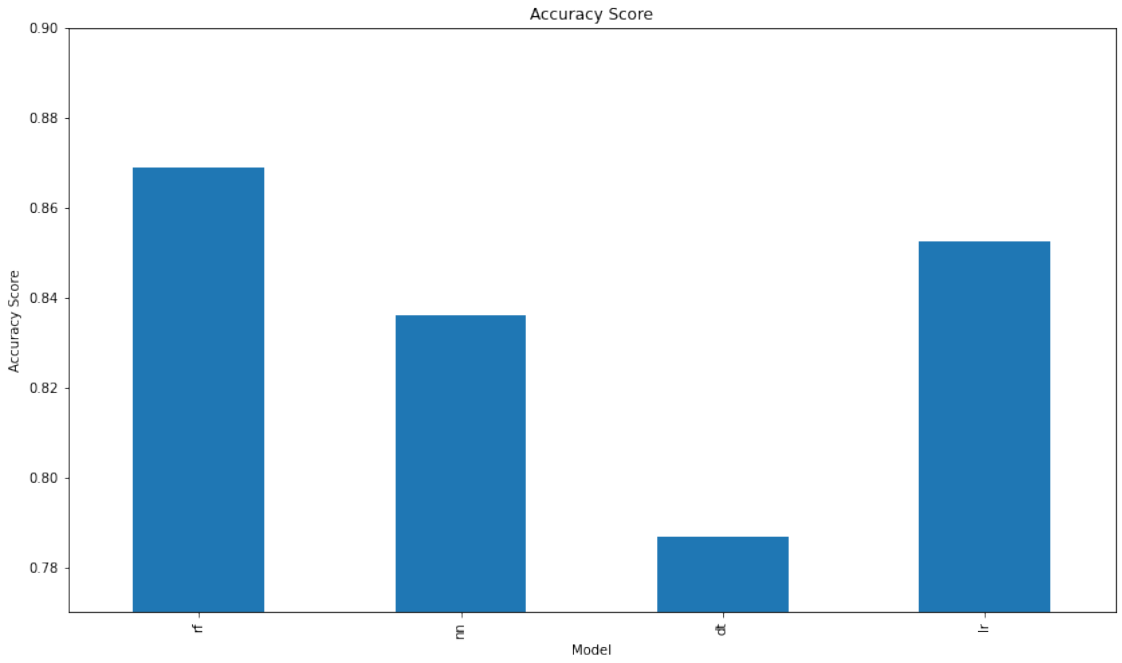
```
Accuracy Score NN:  0.8524590163934426
ROC/AUC Score NN:  0.8529094827586207
```

[17]:
```
modelnames[3] = "lr"
cml[3] = cm
rocl[3]= roc
```

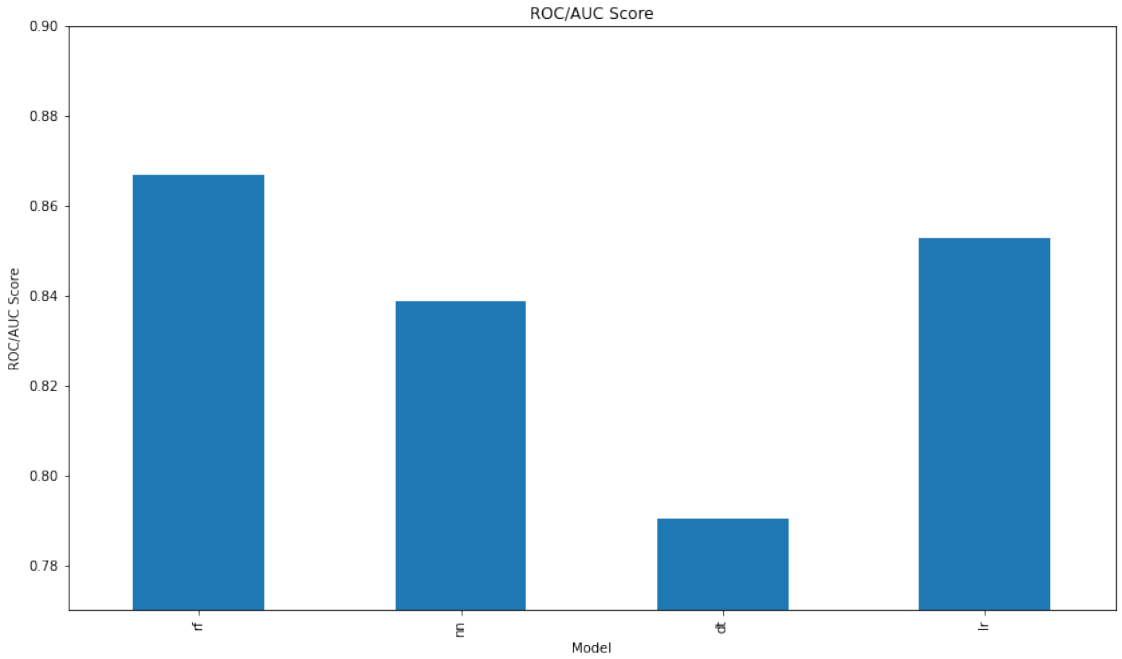
7 Plots For Comparison

[18]:
```
# Plot the Accuracy Score

plt.figure(figsize=(14, 8))
ax = pd.Series(cml).plot(kind="bar")
ax.set_title("Accuracy Score")
ax.set_xlabel("Model")
ax.set_ylabel("Accuracy Score")
```

```
ax.set_xticklabels(modelnames)
plt.ylim(0.77, 0.9)

plt.show()
```



[19]:
```
# Plot the ROCAUC

plt.figure(figsize=(14, 8))
ax = pd.Series(rocl).plot(kind="bar")
ax.set_title("ROC/AUC Score")
ax.set_xlabel("Model")
ax.set_ylabel("ROC/AUC Score")
ax.set_xticklabels(modelnames)
plt.ylim(0.77, 0.9)

plt.show()
```



8 Evaluation

Decision Trees tend to be by far the worst model. Depending on the randomly chosen test data, sometimes neural netrworks and sometimes random forests are better than each other. Logistical regression tends to be either the best or close to the best model(over the course of multiple executions). Generally, Accuracy Scores and ROC/AUC Scores tend to be very similar.

9 Shap

[20]:
```
import shap

X_importance_dtc = X_test
X_importance_rfc = X_test

X_importance_nn = X_test
X_importance_lg = X_test

# Explain model predictions for decision trees using shap library:
#explainer_dtc = shap.TreeExplainer(dtc)
#shap_values_dtc = explainer_dtc.shap_values(X_importance_dtc)
explainer_dtc = shap.KernelExplainer(dtc.predict,X_train)
shap_values_dtc = explainer_dtc.shap_values(X_importance_dtc)

#random forest
```

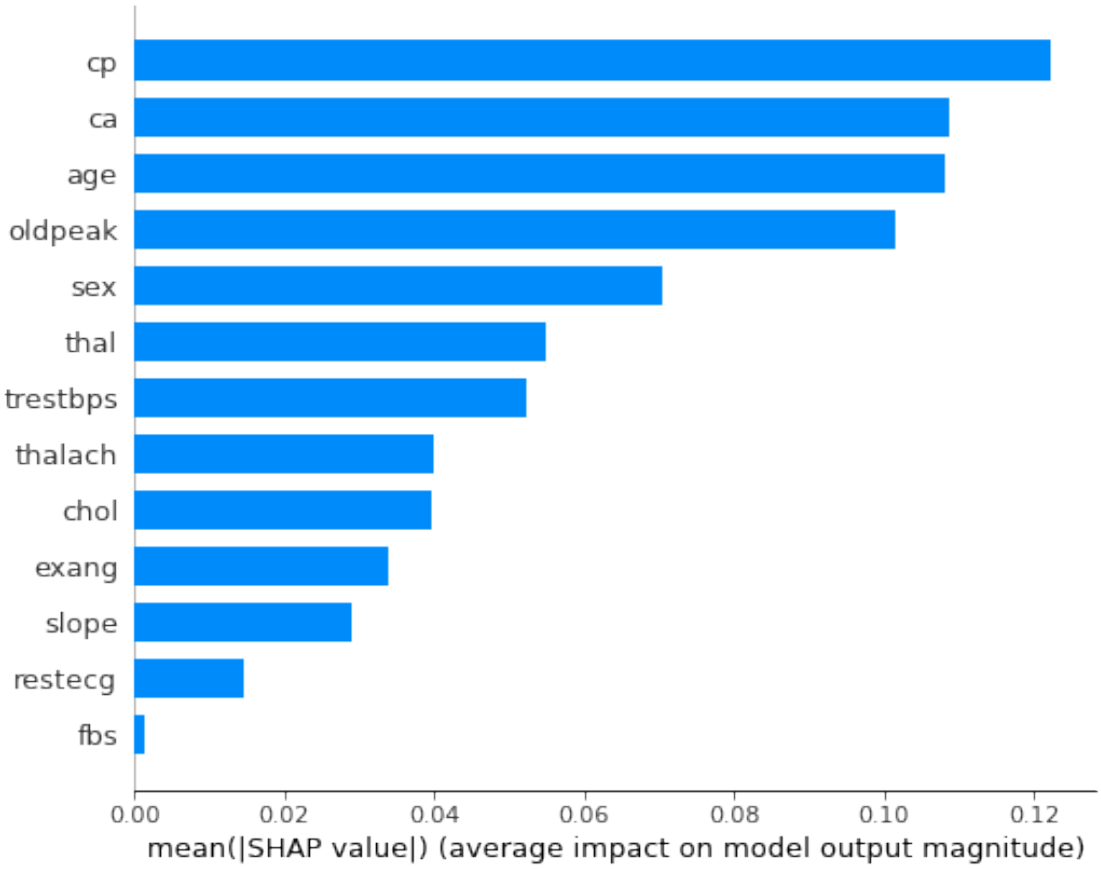
```
#explainer_rfc = shap.TreeExplainer(rfc)
#shap_values_rfc = explainer_rfc.shap_values(X_importance_rfc)
explainer_rfc = shap.KernelExplainer(rfc.predict,X_train)
shap_values_rfc = explainer_rfc.shap_values(X_importance_rfc)

#neural network
explainer_nn = shap.KernelExplainer(mlpc.predict,X_train)
shap_values_nn = explainer_nn.shap_values(X_importance_nn)

#logistic regression
explainer_lg = shap.KernelExplainer(lgr.predict,X_train)
shap_values_lg = explainer_lg.shap_values(X_importance_lg)
```

```
Using 242 background data samples could cause slower run times. Consider using
shap.sample(data, K) or shap.kmeans(data, K) to summarize the background as K
samples.

  0%|          | 0/61 [00:00<?, ?it/s]

Using 242 background data samples could cause slower run times. Consider using
shap.sample(data, K) or shap.kmeans(data, K) to summarize the background as K
samples.

  0%|          | 0/61 [00:00<?, ?it/s]

Using 242 background data samples could cause slower run times. Consider using
shap.sample(data, K) or shap.kmeans(data, K) to summarize the background as K
samples.

  0%|          | 0/61 [00:00<?, ?it/s]

Using 242 background data samples could cause slower run times. Consider using
shap.sample(data, K) or shap.kmeans(data, K) to summarize the background as K
samples.

  0%|          | 0/61 [00:00<?, ?it/s]
```

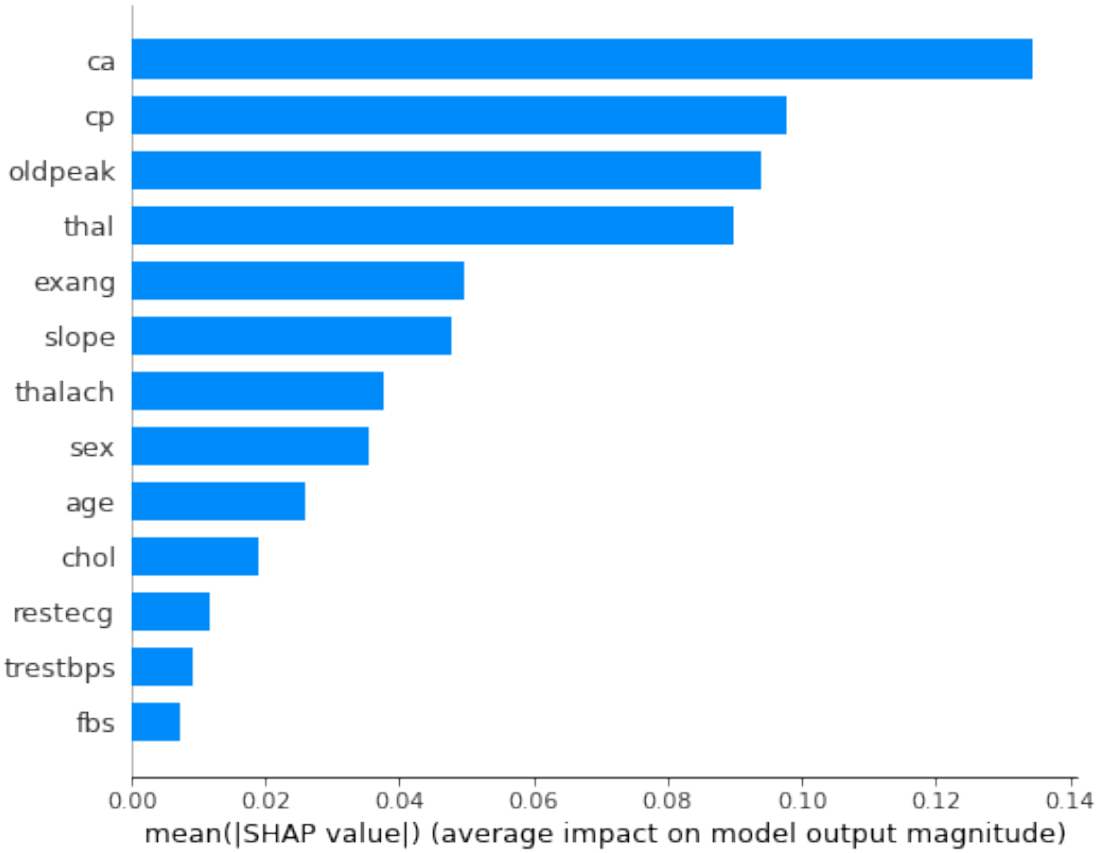
[21]:
```
# Plot summary_plot for dt as barplot:
print("Decision Trees")
shap.summary_plot(shap_values_dtc, X_importance_dtc, plot_type='bar')

# Plot summary_plot for rfc as barplot:
print("Random Forest")
shap.summary_plot(shap_values_rfc, X_importance_rfc, plot_type='bar')

#neural network:
print("Neural Networks")
shap.summary_plot(shap_values_nn, X_importance_nn, plot_type='bar')

#logistic regression:
print("Logistic Regression")
```

```
shap.summary_plot(shap_values_lg, X_importance_lg, plot_type='bar')
```

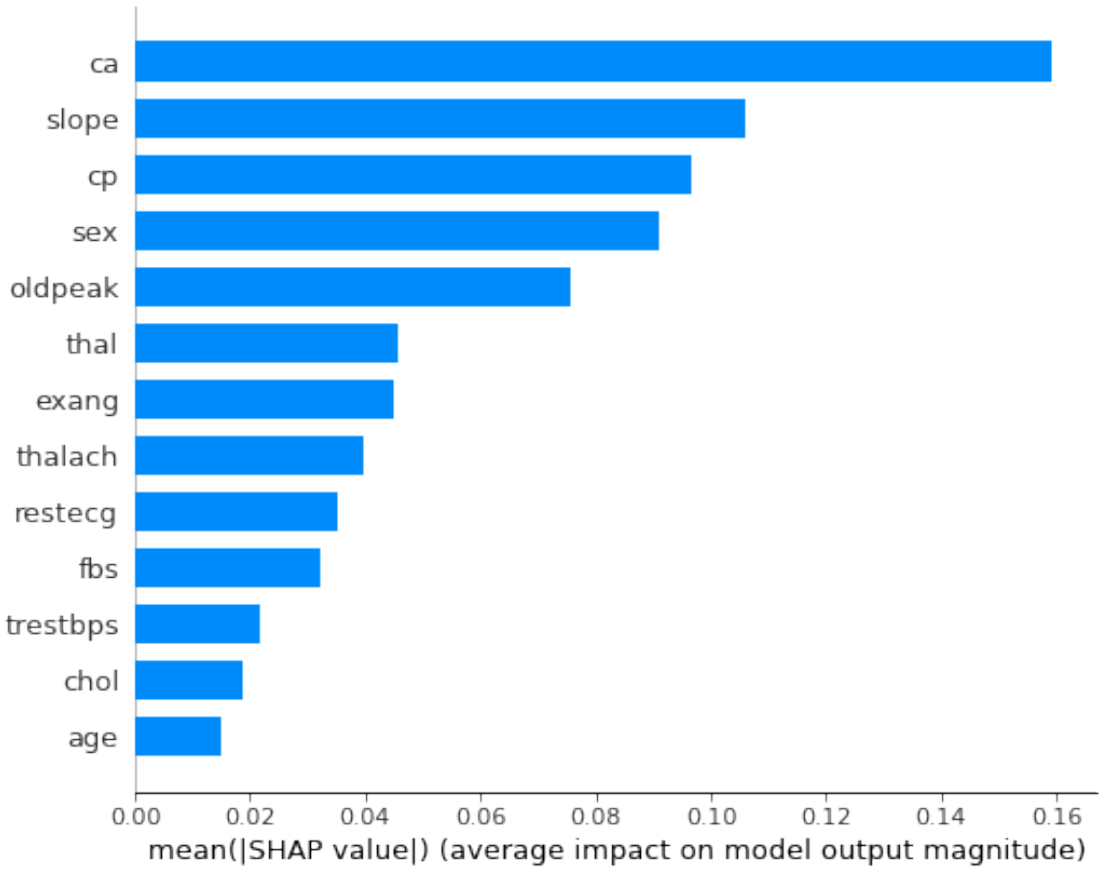
Decision Trees



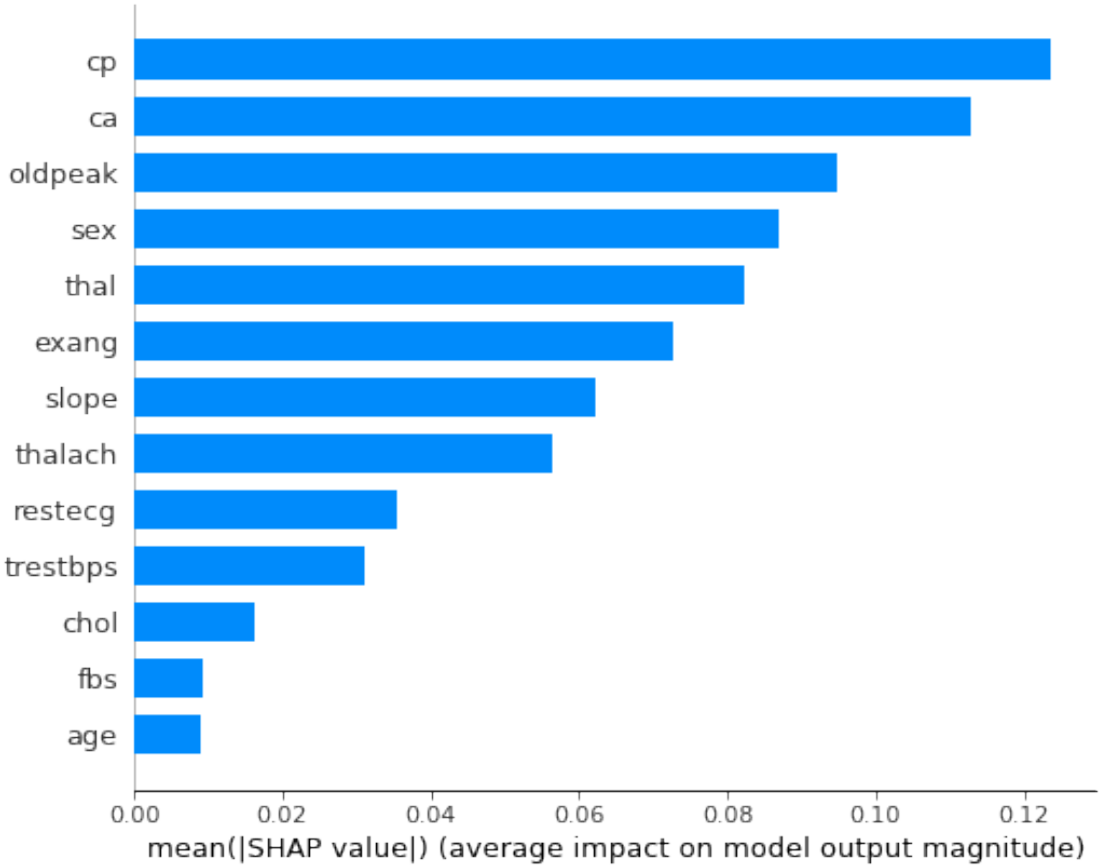
Random Forest



Neural Networks



Logistic Regression



10 Discussion

Difference in importance of features between the tree models decision trees and random forest: The differences are quite strong, ca and cp are the most important feature for each model and whereas age is the third most importand feature for the decision tree model, it is only the 9th most important feature for the random forest model.Yet, there are similiarities, like ca and cp being among the most important features and rbs and restecg being among the least important. In the other diagrams the different relevance of age can also be observed, so some features get very different importances depending on the model.

Most important features: ca and cp are the most important features in all models. Oldpeak is also always amonst the most important features. Age, that, slope or sex are only amongst the most important features in some models.

Discuss (write some text!) about which model works best, and which is "most interpretable": The model that works best and is most robust, based on the accuracy score of multiple runs, is the logistic regression, meaning it is the est classifier out of the four models usually. When looking at the force graphs and importance graphs, it seems that the tree models are most interpretable, because comparably few features have most of the force/impact, and therefore the classification can be more easily attributed to one of the few very influential features.

[22]:
```
shap.initjs()
print("Decision Trees")
shapvalues_extra = explainer_dtc.shap_values(X_test.iloc[0,:])
shap.plots.force(base_value = explainer_dtc.expected_value,shap_values =␣
↪shapvalues_extra,features = X_test.iloc[0,:].to_numpy(), feature_names =␣
↪names[0:13])
```

```
<IPython.core.display.HTML object>
```

```
Decision Trees
```

[22]: <shap.plots._force.AdditiveForceVisualizer at 0x241bc0d9580>

[23]:
```
print("Random Forest")
shapvalues_extra = explainer_rfc.shap_values(X_test.iloc[0,:])
shap.plots.force(base_value = explainer_rfc.expected_value,shap_values =␣
↪shapvalues_extra,features = X_test.iloc[0,:].to_numpy(), feature_names =␣
↪names[0:13])
```

```
Random Forest
```

[23]: <shap.plots._force.AdditiveForceVisualizer at 0x241bbf841f0>

[24]:
```
print("Neural Networks")
shapvalues_extra = explainer_nn.shap_values(X_test.iloc[0,:])
shap.plots.force(base_value = explainer_nn.expected_value,shap_values =␣
↪shapvalues_extra,features = X_test.iloc[0,:].to_numpy(), feature_names =␣
↪names[0:13])
```

```
Neural Networks
```

[24]: <shap.plots._force.AdditiveForceVisualizer at 0x241c12128b0>

[25]:
```
print("Logistic Regression")
shapvalues_extra = explainer_lg.shap_values(X_test.iloc[0,:])
shap.plots.force(base_value = explainer_lg.expected_value,shap_values =␣
↪shapvalues_extra,features = X_test.iloc[0,:].to_numpy(), feature_names =␣
↪names[0:13])
```

```
Logistic Regression
```

[25]: <shap.plots._force.AdditiveForceVisualizer at 0x241c12179a0>

10.1 Exercise 03.02: Reading/Discussion/Summary

Part 1: Reading: * Read the following paper: Rudin (2019) “Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead” * The paper is available here: https://www.nature.com/articles/s42256-019-0048-x

Part 2: Think about the following questions: * What is Explainable Machine Learning? * What are the features of Interpretable Machine Learning? * Why and when do we need both? * Are

there any disadvantages? * What are specific challenges in their application? * Can, and if so, how, explainable and interpretable ML be implemented efficiently? * What are some exemplary techniques to apply?

Part 3: Discussing, Summary * Prepare answers for these questions for the practical session on November 16, 2021. You will first discuss these in groups, and then we will discuss them in the plenary meeting. * After that, summarize your findings (and those of the group discussion) in a small report (max. half a Din A4 page). For example, you could write 2-3 sentences for answering a specific question.

10.2 Report

Explainable machine Learning consists of two diffrent models. The first one is a black box, that ist just to complicated for humans to understand. The second one replicates most of the features of the original model, but represents the output in a way that makes it possible for humans to understand.

Interpretable machine learning on the other hand is inherently explainable. Instead of an translation from the original model it is possible to just follow its reasoning.

If you plan to develop a machine learning model for high-stakes decisions, for the final product an interpretable machine learning model should be used because the explanation of a explainable model may leaves out so much information that its decision makes no sense. A explainable machine Learning model still could be used for knowledge discovery.

Interpretable models require significant effort to construct and are less attractive for companys due to their transparency. Black box models on the other hand can lead to false assumtions on the reasoning of the model. This can have a big impact if it is needed for humans to understand the decision.

A challenge of interpretable models is, that interpretability is not always clearly defined, for example in computer vision. Also the construction of optimal interpretable models can be computational hard, so good optimization methods need to be found.

These computational problems can be solved by leveraging a combination of theoretical and systems-level tech- niques. Logical Models can be accelerated by using special datastructures, search space reduction and fast exploration of search space. Interpretable neural networks are generally harder than normal neural networks tho and more research needs to be conducted.

Some exemplary techniques to solves these computational hard problems is for example the CORELS algorithm it solves the minimization problem for computing a logical model. RiskSLIM is another technique to solve another type of minimization. It solves it using cutting planes for setting coefficients of a scoring system.

10.3 Uploading your solution

For uploading your solution, please upload two files: * The Jupyter-Notebook file (.ipynb) * A PDF (printout/file) of the Jupyter notebook file (.pdf)

[ ]: