In [283]:
```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt
import seaborn as sns
```

In [182]:
```
from pathlib import Path

input_dir = Path('/Users/pavangedala/Documents/Self Projects /Heart Atta
for file_path in input_dir.rglob('*'):
    if file_path.is_file():
        print(file_path)
```

In [183]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import accuracy_score, roc_curve
```

In [184]:
```
import warnings
warnings.filterwarnings("ignore")
```

In [185]:
```
import os
file_path = "/Users/pavangedala/Documents/Self Projects/Heart Attack Dat

if os.path.exists(file_path):
    df = pd.read_csv(file_path)
else:
    print("File not found! Check the file path.")
```

```
File not found! Check the file path.
```

In [186]:
```
import os
for dirname, _, filenames in os.walk('/Users/pavangedala/Documents/Self
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

In [187]:
```
#Read and Analyse Data
```

In [188]:
```

df = pd.read_csv("/content/heart.csv")
```

In [189]:
```
df.head()
#
```

Out[189]:

	age	sex	cp	trtbps	chol	fbs	restecg	thalachh	exng	oldpeak	slp	caa	thall	output
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	1
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2	1
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2	1
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2	1

In [190]:
```
df.describe()
```

Out[190]:

	age	sex	cp	trtbps	chol	fbs	restecg	tha
count	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.0
mean	54.366337	0.683168	0.966997	131.623762	246.264026	0.148515	0.528053	149.6
std	9.082101	0.466011	1.032052	17.538143	51.830751	0.356198	0.525860	22.9
min	29.000000	0.000000	0.000000	94.000000	126.000000	0.000000	0.000000	71.0
25%	47.500000	0.000000	0.000000	120.000000	211.000000	0.000000	0.000000	133.5
50%	55.000000	1.000000	1.000000	130.000000	240.000000	0.000000	1.000000	153.0
75%	61.000000	1.000000	2.000000	140.000000	274.500000	0.000000	1.000000	166.0
max	77.000000	1.000000	3.000000	200.000000	564.000000	1.000000	2.000000	202.0

In [191]:
```
# information about data frame
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 303 entries, 0 to 302
Data columns (total 14 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   age       303 non-null    int64  
 1   sex       303 non-null    int64  
 2   cp        303 non-null    int64  
 3   trtbps    303 non-null    int64  
 4   chol      303 non-null    int64  
 5   fbs       303 non-null    int64  
 6   restecg   303 non-null    int64  
 7   thalachh  303 non-null    int64  
 8   exng      303 non-null    int64  
 9   oldpeak   303 non-null    float64
 10  slp       303 non-null    int64  
 11  caa       303 non-null    int64  
 12  thall     303 non-null    int64  
 13  output    303 non-null    int64  
dtypes: float64(1), int64(13)
memory usage: 33.3 KB
```

In [192]:
```
#Missing Value Analysis¶
```

In [193]:
```
df.isnull().sum()
```

Out[193]:

	0
age	0
sex	0
cp	0
trtbps	0
chol	0
fbs	0
restecg	0
thalachh	0
exng	0
oldpeak	0
slp	0
caa	0
thall	0
output	0

dtype: int64

In [194]:
```
#Unique Value Analysis¶
```

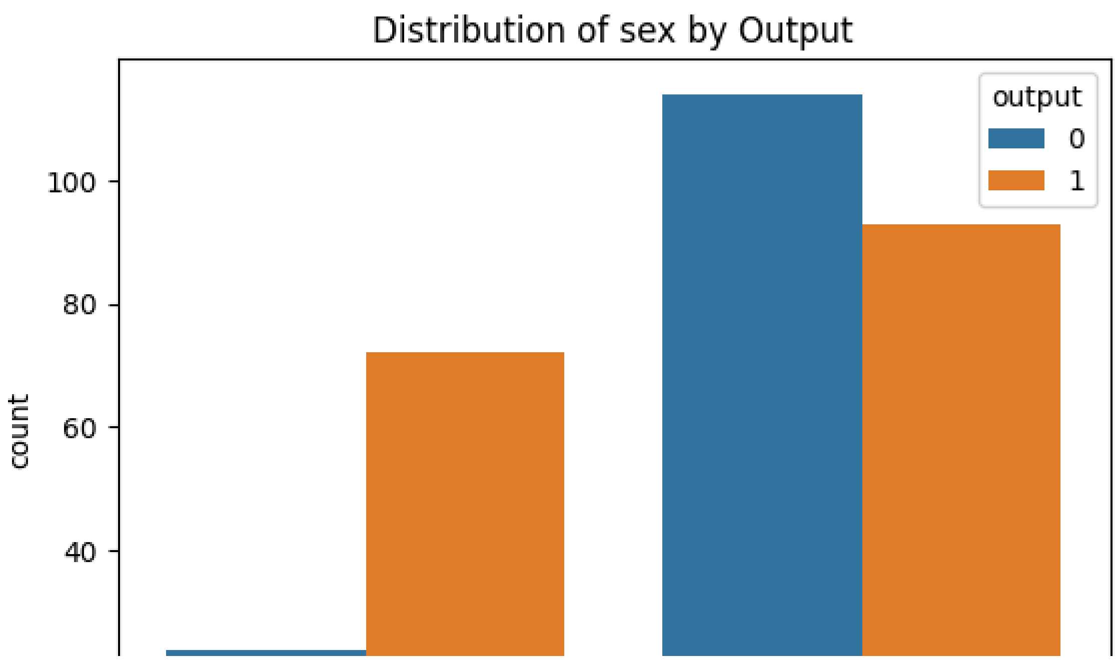
In [195]:
```
for i in list(df.columns):
    print("{} -- {}".format(i, df[i].value_counts().shape[0]))
```

```
age -- 41
sex -- 2
cp -- 4
trtbps -- 49
chol -- 152
fbs -- 2
restecg -- 3
thalachh -- 91
exng -- 2
oldpeak -- 40
slp -- 3
caa -- 5
thall -- 4
output -- 2
```

In [196]:
```
#Categorical Feature Analysis¶
```

In [197]:
```
categorical_list = ["sex", "cp","fbs","restecg","exng","slp","caa","thal
```

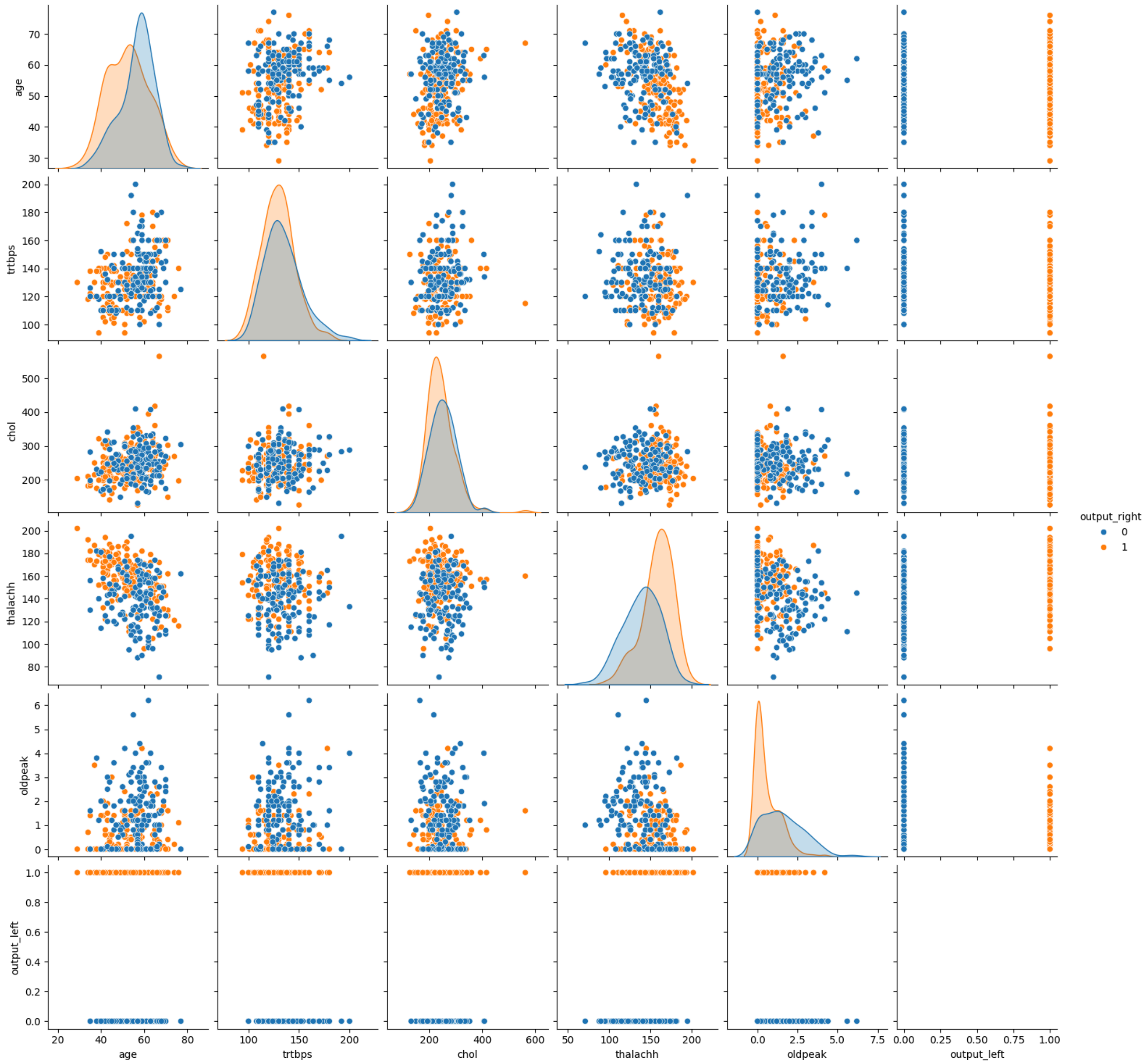
In [198]:
```
df_categoric = df[categorical_list]

for col in categorical_list:
    plt.figure()
    sns.countplot(x=col, data=df_categoric, hue='output')
    plt.title(f'Distribution of {col} by Output')
    plt.show()
```



In [199]:
```
#Numeric Feature Analysis
#Bivariate data analysis with scatter plot
```

In [200]:
```
numeric_list = ["age", "trtbps","chol","thalachh","oldpeak","output"]
```

In [201]:
```
df_numeric = df[numeric_list]
# Use the merge function instead of join and specify suffixes for overla
result = pd.merge(df_numeric, df['output'], left_index=True, right_index
# Plot the result using sns.pairplot
sns.pairplot(result, hue='output_right', diag_kind='kde') # Use the corr
plt.show()
```



In [202]:
```
#Standardization¶
```

In [203]:
```
scaler = StandardScaler()
scaler
```

Out[203]: StandardScaler()

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [204]:
```
scaled_array = scaler.fit_transform(df[numeric_list[:-1]])
```

In [205]:
```
scaled_array
```

Out[205]:
```
array([[ 0.9521966 ,  0.76395577, -0.25633371,  0.01544279,  1.0873380
6],
       [-1.91531289, -0.09273778,  0.07219949,  1.63347147,  2.1225727
3],
       [-1.47415758, -0.09273778, -0.81677269,  0.97751389,  0.3109120
6],
       ...,
       [ 1.50364073,  0.70684287, -1.029353  , -0.37813176,  2.0363031
7],
       [ 0.29046364, -0.09273778, -2.2275329 , -1.51512489,  0.1383729
5],
       [ 0.29046364, -0.09273778, -0.19835726,  1.0649749 , -0.8968617
2]])
```

In [206]:
```
# pd.DataFrame(scaled_array).describe()
```

In [207]:
```
#Box Plot Analysis¶
```

In [208]:
```
df_dummy = pd.DataFrame(scaled_array, columns = numeric_list[:-1])
df_dummy.head()
```

Out[208]:

	age	trtbps	chol	thalachh	oldpeak
0	0.952197	0.763956	-0.256334	0.015443	1.087338
1	-1.915313	-0.092738	0.072199	1.633471	2.122573
2	-1.474158	-0.092738	-0.816773	0.977514	0.310912
3	0.180175	-0.663867	-0.198357	1.239897	-0.206705
4	0.290464	-0.663867	2.082050	0.583939	-0.379244

In [209]:
```
df_dummy = pd.concat([df_dummy, df.loc[:, "output"]], axis = 1)
df_dummy.head()
```

Out[209]:

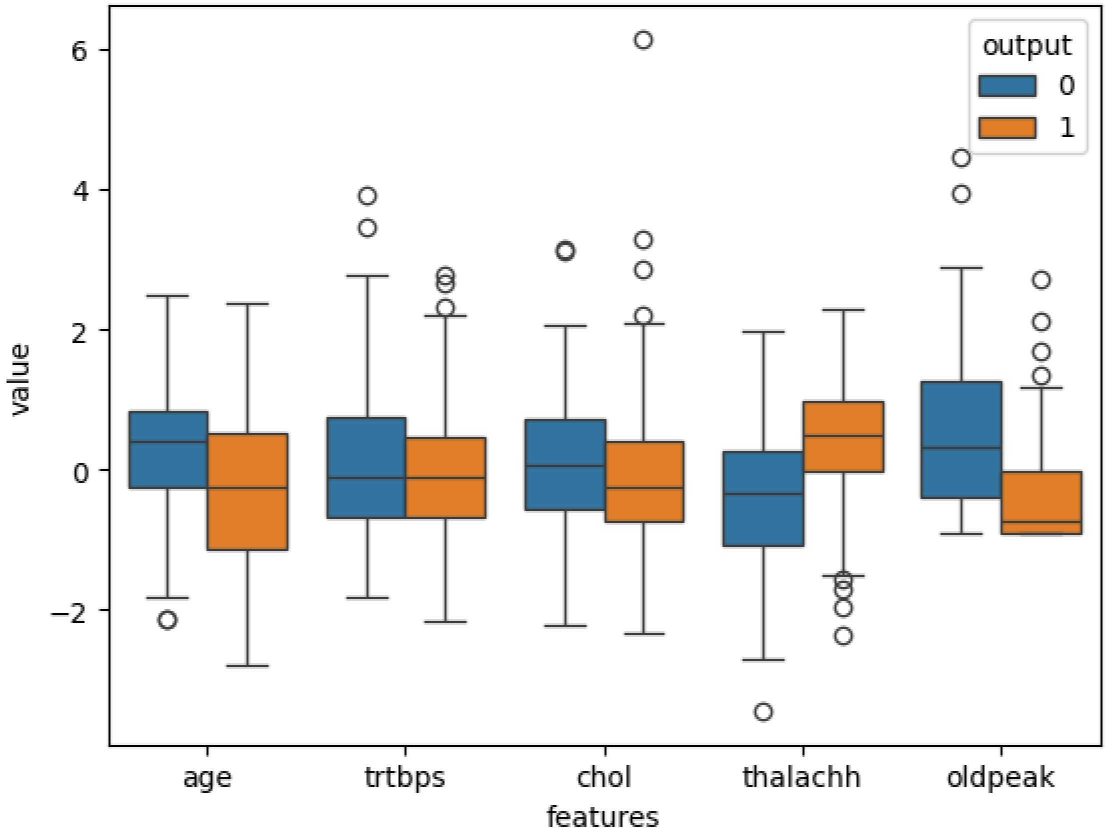
	age	trtbps	chol	thalachh	oldpeak	output
0	0.952197	0.763956	-0.256334	0.015443	1.087338	1
1	-1.915313	-0.092738	0.072199	1.633471	2.122573	1
2	-1.474158	-0.092738	-0.816773	0.977514	0.310912	1
3	0.180175	-0.663867	-0.198357	1.239897	-0.206705	1
4	0.290464	-0.663867	2.082050	0.583939	-0.379244	1

In [210]:
```
data_melted = pd.melt(df_dummy, id_vars = "output", var_name = "features
data_melted.head(20)
```

Out[210]:

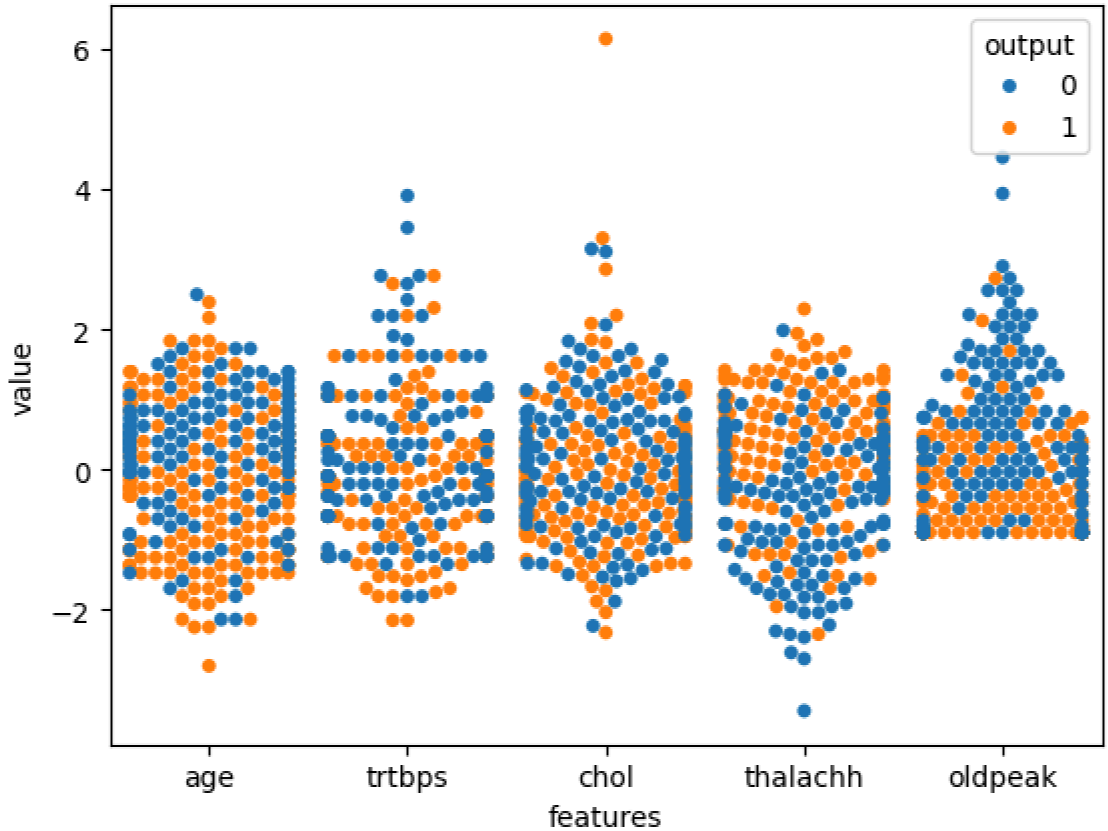
	output	features	value
0	1	age	0.952197
1	1	age	-1.915313
2	1	age	-1.474158
3	1	age	0.180175
4	1	age	0.290464
5	1	age	0.290464
6	1	age	0.180175
7	1	age	-1.143291
8	1	age	-0.260980
9	1	age	0.290464
10	1	age	-0.040403
11	1	age	-0.702136
12	1	age	-0.591847
13	1	age	1.062485
14	1	age	0.400752
15	1	age	-0.481558
16	1	age	0.400752
17	1	age	1.283063
18	1	age	-1.253580
19	1	age	1.613930

In [211]:
```
# box plot
plt.figure()
sns.boxplot(x = "features", y = "value", hue = "output", data= data_melt
plt.show()
```



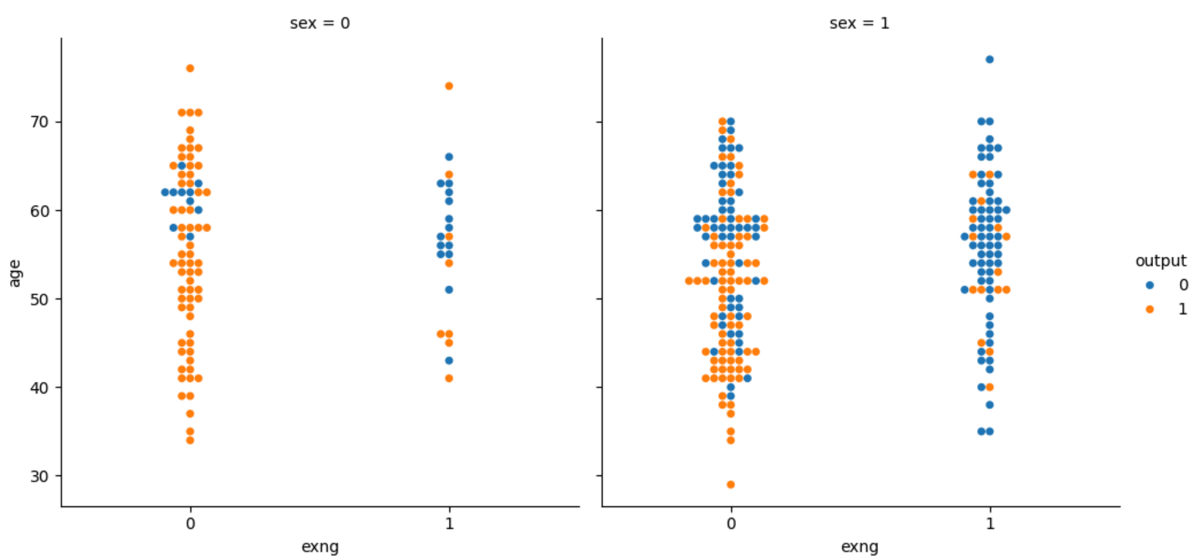
In [212]:
```
#Swarm Plot Analysis¶
```

In [213]:
```
# swarm plot
plt.figure()
sns.swarmplot(x = "features", y = "value", hue = "output", data= data_me
plt.show()
```



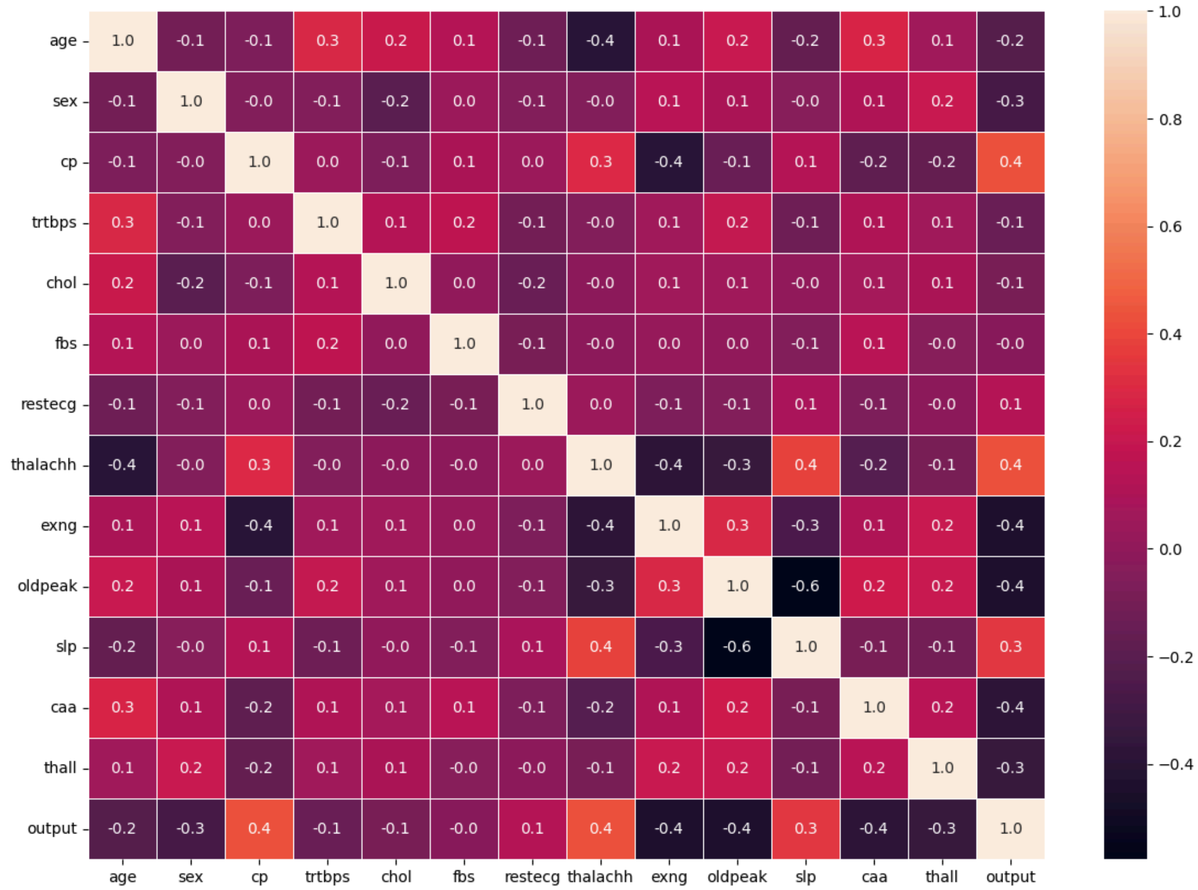
In [214]:
```
#Cat Plot Analysis¶
```

In [215]:
```
sns.catplot(x = "exng", y = "age", hue = "output", col = "sex", kind = "
plt.show()
```



In [216]:
```
#Correlation Analysis¶
```

In [217]:
```
plt.figure(figsize = (14,10))
sns.heatmap(df.corr(), annot = True, fmt = ".1f", linewidths = .7)
plt.show()
```



In [218]:
```
#Outlier Detection¶
```

In [219]:
```
numeric_list = ["age", "trtbps","chol","thalachh","oldpeak"]
df_numeric = df.loc[:, numeric_list]
df_numeric.head()
```

Out[219]:

	age	trtbps	chol	thalachh	oldpeak
0	63	145	233	150	2.3
1	37	130	250	187	3.5
2	41	130	204	172	1.4
3	56	120	236	178	0.8
4	57	120	354	163	0.6

In [220]:
```
df.describe()
```

Out[220]:

	age	sex	cp	trtbps	chol	fbs	restecg	tha
count	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.0
mean	54.366337	0.683168	0.966997	131.623762	246.264026	0.148515	0.528053	149.6
std	9.082101	0.466011	1.032052	17.538143	51.830751	0.356198	0.525860	22.9
min	29.000000	0.000000	0.000000	94.000000	126.000000	0.000000	0.000000	71.0
25%	47.500000	0.000000	0.000000	120.000000	211.000000	0.000000	0.000000	133.5
50%	55.000000	1.000000	1.000000	130.000000	240.000000	0.000000	1.000000	153.0
75%	61.000000	1.000000	2.000000	140.000000	274.500000	0.000000	1.000000	166.0
max	77.000000	1.000000	3.000000	200.000000	564.000000	1.000000	2.000000	202.0

In [221]:
```
# outlier detection
for i in numeric_list:

    # IQR
    Q1 = np.percentile(df.loc[:, i],25)
    Q3 = np.percentile(df.loc[:, i],75)

    IQR = Q3 - Q1

    print("Old shape: ", df.loc[:, i].shape)

    # upper bound
    upper = np.where(df.loc[:, i] >= (Q3 +2.5*IQR))

    # lower bound
    lower = np.where(df.loc[:, i] <= (Q1 - 2.5*IQR))

    print("{} -- {}".format(upper, lower))

    try:
        df.drop(upper[0], inplace = True)
    except: print("KeyError: {} not found in axis".format(upper[0]))

    try:
        df.drop(lower[0], inplace = True)
    except:  print("KeyError: {} not found in axis".format(lower[0]))

    print("New shape: ", df.shape)
```

```
Old shape:  (303,)
(array([], dtype=int64),) -- (array([], dtype=int64),)
New shape:  (303, 14)
Old shape:  (303,)
(array([223, 248]),) -- (array([], dtype=int64),)
New shape:  (301, 14)
Old shape:  (301,)
(array([85]),) -- (array([], dtype=int64),)
New shape:  (300, 14)
Old shape:  (300,)
(array([], dtype=int64),) -- (array([], dtype=int64),)
New shape:  (300, 14)
Old shape:  (300,)
(array([203, 220]),) -- (array([], dtype=int64),)
New shape:  (298, 14)
```

In [222]:
```
#Modelling¶
```

In [223]:
```
df1 = df.copy()
```

In [224]:
```
df1 = pd.get_dummies(df1, columns = categorical_list[:-1], drop_first = 
df1.head()
```

Out[224]:

	age	trtbps	chol	thalachh	oldpeak	output	sex_1	cp_1	cp_2	cp_3	...	exng_1	slp_1	slp_
0	63	145	233	150	2.3	1	True	False	False	True	...	False	False	Fal
1	37	130	250	187	3.5	1	True	False	True	False	...	False	False	Fal
2	41	130	204	172	1.4	1	False	True	False	False	...	False	False	Tr
3	56	120	236	178	0.8	1	True	True	False	False	...	False	False	Tr
4	57	120	354	163	0.6	1	False	False	False	False	...	True	False	Tr

5 rows × 23 columns

In [225]:
```
X = df1.drop(["output"], axis = 1)
y = df1[["output"]]
```

In [226]:
```
#Scaling
```

In [227]:
```
scaler = StandardScaler()
scaler
```

Out[227]:

```
StandardScaler()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [228]:
```
X[numeric_list[:-1]] = scaler.fit_transform(X[numeric_list[:-1]])
X.head()
```

Out[228]:

	age	trtbps	chol	thalachh	oldpeak	sex_1	cp_1	cp_2	cp_3	fbs_1	...	exng_
0	0.965901	0.845093	-0.236684	0.021855	2.3	True	False	False	True	True	...	Fals
1	-1.902555	-0.061886	0.119326	1.639116	3.5	True	False	True	False	False	...	Fals
2	-1.461254	-0.061886	-0.843995	0.983470	1.4	False	True	False	False	False	...	Fals
3	0.193624	-0.666538	-0.173859	1.245729	0.8	True	True	False	False	False	...	Fals
4	0.303950	-0.666538	2.297269	0.590082	0.6	False	False	False	False	False	...	Tru

5 rows × 22 columns

In [229]:
```
#Train/Test Split¶
```

In [230]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.
print("X_train: {}".format(X_train.shape))
print("X_test: {}".format(X_test.shape))
print("y_train: {}".format(y_train.shape))
print("y_test: {}".format(y_test.shape))
```

```
X_train: (268, 22)
X_test: (30, 22)
y_train: (268, 1)
y_test: (30, 1)
```

In [231]:
```
#Logistic Regression¶
```

In [232]:
```
logreg = LogisticRegression()
logreg
```

Out[232]:
```
LogisticRegression()
```
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [233]:
```
# fitting = training
logreg.fit(X_train, y_train)
```

Out[233]:
```
LogisticRegression()
```
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [234]:
```
# calculate probabilities
y_pred_prob = logreg.predict_proba(X_test)
y_pred_prob
```

Out[234]:
```
array([[0.94268667, 0.05731333],
       [0.06997145, 0.93002855],
       [0.11282989, 0.88717011],
       [0.48063308, 0.51936692],
       [0.08770139, 0.91229861],
       [0.01943155, 0.98056845],
       [0.01314682, 0.98685318],
       [0.25649952, 0.74350048],
       [0.93057819, 0.06942181],
       [0.04667535, 0.95332465],
       [0.95754617, 0.04245383],
       [0.01122815, 0.98877185],
       [0.41901335, 0.58098665],
       [0.60411313, 0.39588687],
       [0.02729659, 0.97270341],
       [0.02614833, 0.97385167],
       [0.84042163, 0.15957837],
       [0.03595037, 0.96404963],
       [0.86160266, 0.13839734],
       [0.97611122, 0.02388878],
       [0.61984028, 0.38015972],
       [0.31186627, 0.68813373],
       [0.93464752, 0.06535248],
       [0.00474577, 0.99525423],
       [0.44456547, 0.55543453],
       [0.3387469 , 0.6612531 ],
       [0.03944728, 0.96055272],
       [0.99154986, 0.00845014],
       [0.46893753, 0.53106247],
       [0.69712671, 0.30287329]])
```

In [235]:
```
y_pred = np.argmax(y_pred_prob, axis = 1)
y_pred
```

Out[235]:
```
array([0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0,
1,
       0, 1, 1, 1, 1, 0, 1, 0])
```

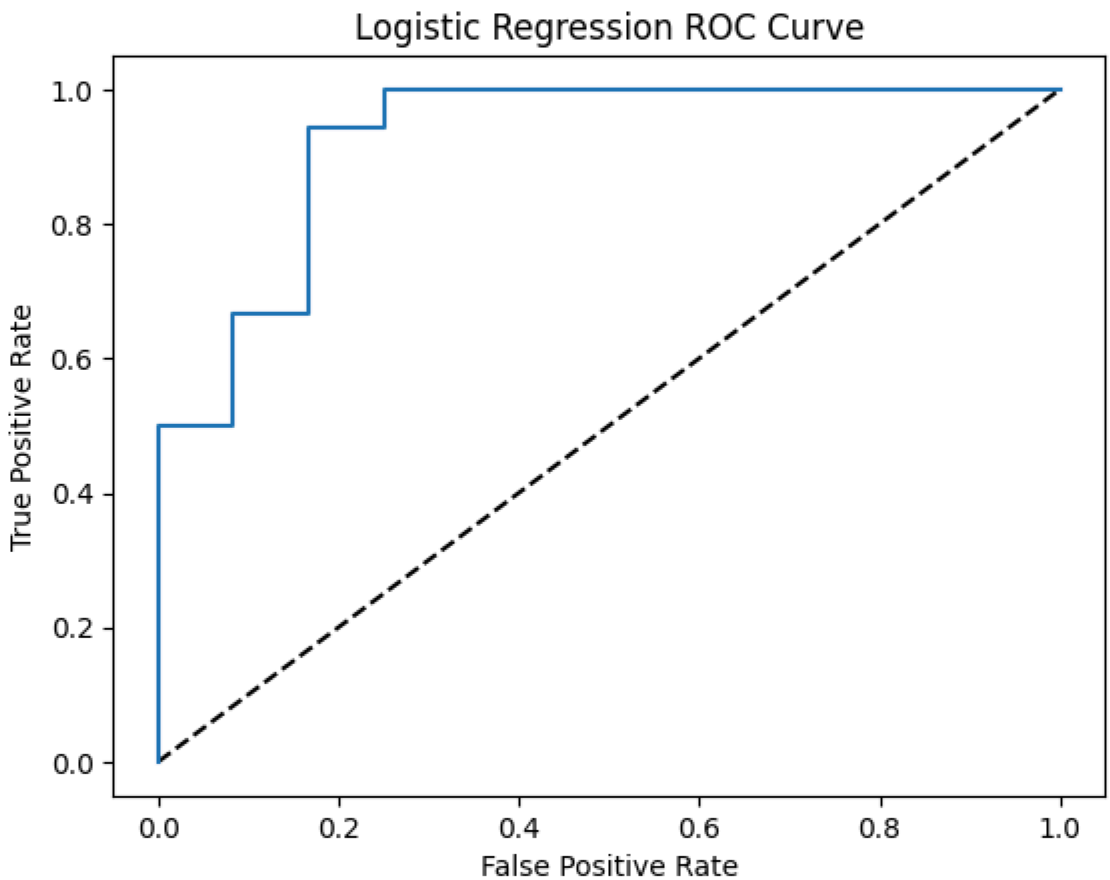
In [236]:
```
#dummy_ = pd.DataFrame(y_pred_prob)
#dummy_["y_pred"] = y_pred
#dummy_.head()
```

In [237]:
```
print("Test accuracy: {}".format(accuracy_score(y_pred, y_test)))
```

```
Test accuracy: 0.9
```

In [238]:
```
# ROC Curve
fpr, tpr, thresholds = roc_curve(y_test, y_pred_prob[:,1])
```

In [239]:
```
# plot curve
plt.plot([0,1],[0,1],"k--")
plt.plot(fpr, tpr, label = "Logistic Regression")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("Logistic Regression ROC Curve")
plt.show()
```



In [240]:
```
#Logistic Regression Hyperparameter Tuning¶
```

In [241]:
```
lr = LogisticRegression()
lr
```

Out[241]:
```
LogisticRegression()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [242]:
```
penalty = ["l1", "l2"]

parameters = {"penalty":penalty}
```

In [243]:
```
lr_searcher = GridSearchCV(lr, parameters)
```

In [244]:
```
lr_searcher.fit(X_train, y_train)
```

Out[244]:
```
GridSearchCV(estimator=LogisticRegression(),
             param_grid={'penalty': ['l1', 'l2']})
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [245]:
```
print("Best parameters: ",lr_searcher.best_params_)
```

```
Best parameters:  {'penalty': 'l2'}
```

In [246]:
```
y_pred = lr_searcher.predict(X_test)
```

In [247]:
```
print("Test accuracy: {}".format(accuracy_score(y_pred, y_test)))
```

```
Test accuracy: 0.9
```

In [248]:
```
#Random Forest Extension
```

In [249]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [250]:
```
# Random Forest Classifier with Hyperparameter Tuning
rf = RandomForestClassifier(random_state=42)
```

In [251]:
```
# Define the hyperparameters grid
param_grid_rf = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}
```

In [252]:
```
# GridSearchCV for Random Forest
rf_grid_search = GridSearchCV(rf, param_grid_rf, cv=5, scoring='accuracy
rf_grid_search.fit(X_train, y_train)
```

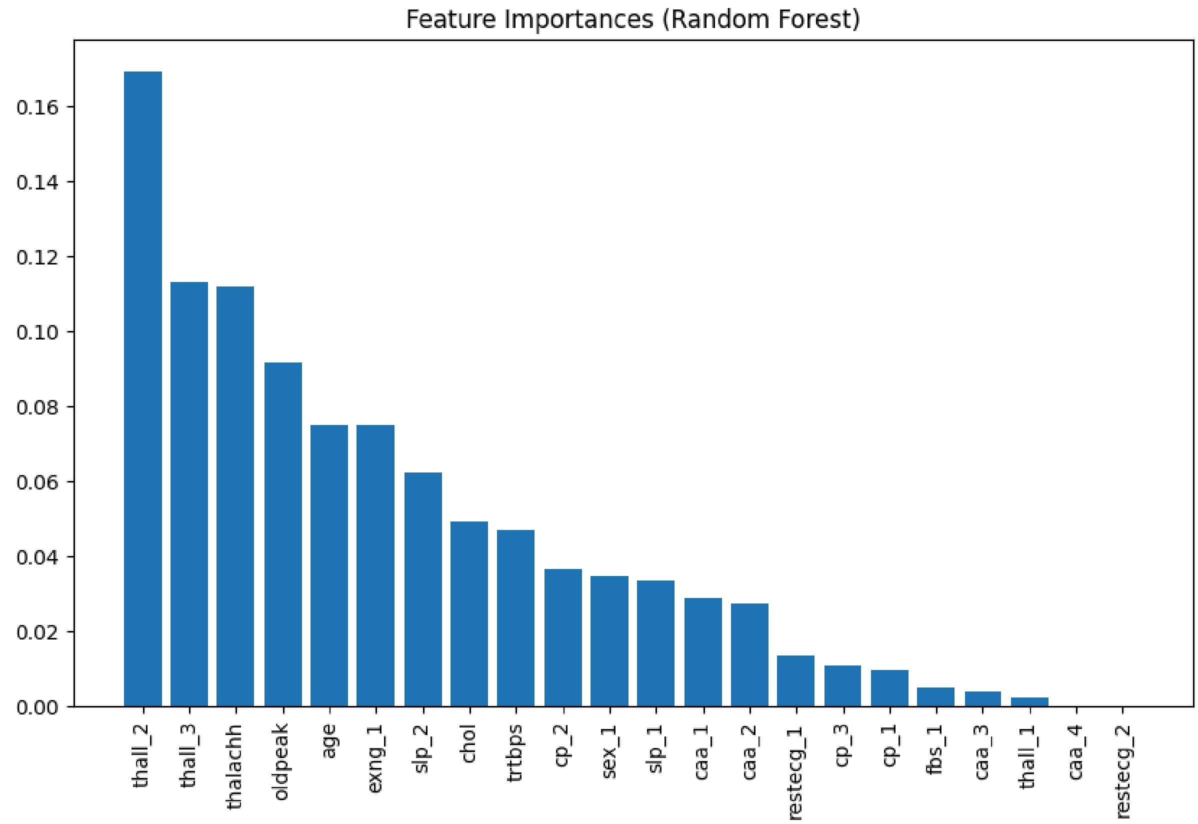
Out[252]:
```
GridSearchCV(cv=5, estimator=RandomForestClassifier(random_state=42),
             param_grid={'max_depth': [None, 10, 20, 30],
                         'min_samples_leaf': [1, 2, 4],
                         'min_samples_split': [2, 5, 10],
                         'n_estimators': [50, 100, 200]},
             scoring='accuracy')
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [253]:
```
# Best Parameters and Evaluation
print("Best Random Forest Parameters:", rf_grid_search.best_params_)
y_pred_rf = rf_grid_search.predict(X_test)
accuracy_rf = accuracy_score(y_test, y_pred_rf)
print(f"Random Forest Accuracy: {accuracy_rf:.4f}")
```

```
Best Random Forest Parameters: {'max_depth': None, 'min_samples_leaf':
4, 'min_samples_split': 10, 'n_estimators': 200}
Random Forest Accuracy: 0.8000
```

In [254]:
```
# Feature Importance Plot
plt.figure(figsize=(10, 6))
importances_rf = rf_grid_search.best_estimator_.feature_importances_
indices_rf = np.argsort(importances_rf)[::-1]
plt.title("Feature Importances (Random Forest)")
plt.bar(range(X.shape[1]), importances_rf[indices_rf], align="center")
plt.xticks(range(X.shape[1]), X.columns[indices_rf], rotation=90)
plt.show()
```



In [255]:
```
#Decision Tree Extension
```

In [256]:
```
from sklearn.tree import DecisionTreeClassifier  # Import DecisionTreeCl
```

In [257]:
```
# Decision Tree Classifier with Hyperparameter Tuning
dt = DecisionTreeClassifier(random_state=42)
```

In [258]:
```
# Define the hyperparameters grid
param_grid_dt = {
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}
```

In [259]:
```
# GridSearchCV for Decision Tree
dt_grid_search = GridSearchCV(dt, param_grid_dt, cv=5, scoring='accuracy
dt_grid_search.fit(X_train, y_train)
```

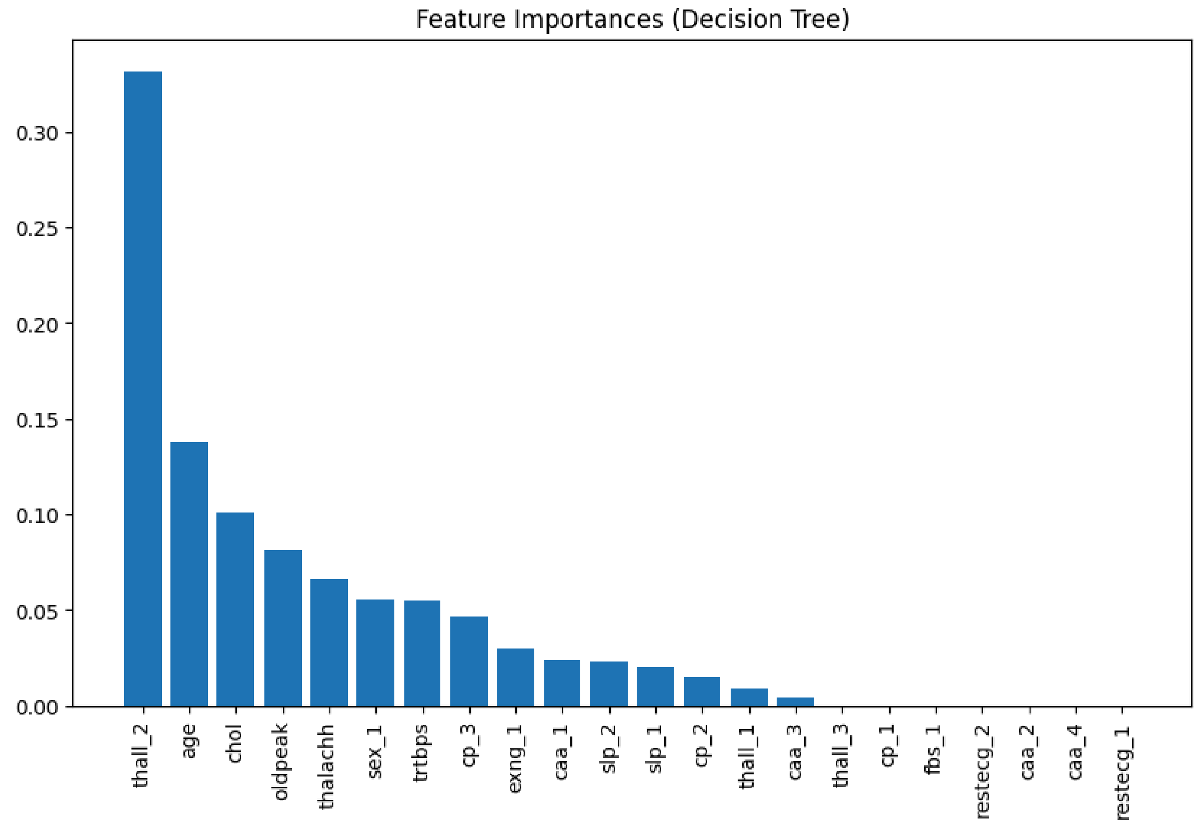
Out[259]:
```
GridSearchCV(cv=5, estimator=DecisionTreeClassifier(random_state=42),
             param_grid={'max_depth': [None, 10, 20, 30],
                         'min_samples_leaf': [1, 2, 4],
                         'min_samples_split': [2, 5, 10]},
             scoring='accuracy')
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [260]:
```
# Best Parameters and Evaluation
print("Best Decision Tree Parameters:", dt_grid_search.best_params_)
y_pred_dt = dt_grid_search.predict(X_test)
accuracy_dt = accuracy_score(y_test, y_pred_dt)
print(f"Decision Tree Accuracy: {accuracy_dt:.4f}")
```

```
Best Decision Tree Parameters: {'max_depth': None, 'min_samples_leaf':
2, 'min_samples_split': 5}
Decision Tree Accuracy: 0.6000
```

In [261]:
```
# Feature Importance Plot
plt.figure(figsize=(10, 6))
importances_dt = dt_grid_search.best_estimator_.feature_importances_
indices_dt = np.argsort(importances_dt)[::-1]
plt.title("Feature Importances (Decision Tree)")
plt.bar(range(X.shape[1]), importances_dt[indices_dt], align="center")
plt.xticks(range(X.shape[1]), X.columns[indices_dt], rotation=90)
plt.show()
```



In [262]:
```
#K-Nearest Neighbors (KNN) Extension
```

In [263]:
```
from sklearn.neighbors import KNeighborsClassifier
```

In [264]:
```
# KNN Classifier with Hyperparameter Tuning
knn = KNeighborsClassifier()
```

In [265]:
```
# Define the hyperparameters grid
param_grid_knn = {
    'n_neighbors': [3, 5, 7, 9],
    'weights': ['uniform', 'distance'],
    'metric': ['euclidean', 'manhattan', 'minkowski']
}
```

In [266]:

```
# GridSearchCV for KNN
knn_grid_search = GridSearchCV(knn, param_grid_knn, cv=5, scoring='accur
knn_grid_search.fit(X_train, y_train)
```

Out[266]:

```
GridSearchCV(cv=5, estimator=KNeighborsClassifier(),
             param_grid={'metric': ['euclidean', 'manhattan', 'minkowsk
i'],
                         'n_neighbors': [3, 5, 7, 9],
                         'weights': ['uniform', 'distance']},
             scoring='accuracy')
```

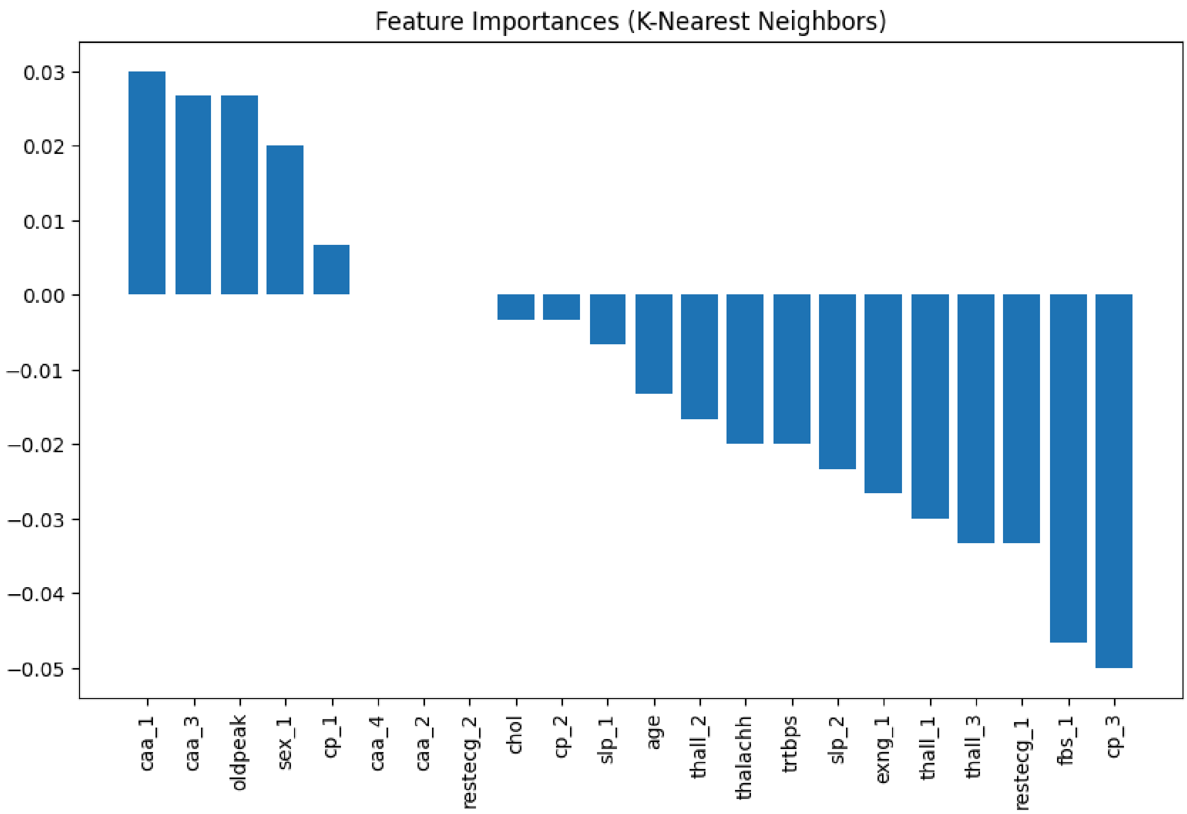
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [267]:

```
# Best Parameters and Evaluation
print("Best KNN Parameters:", knn_grid_search.best_params_)
y_pred_knn = knn_grid_search.predict(X_test)
accuracy_knn = accuracy_score(y_test, y_pred_knn)
print(f"KNN Accuracy: {accuracy_knn:.4f}")
```

```
Best KNN Parameters: {'metric': 'manhattan', 'n_neighbors': 3, 'weight
s': 'uniform'}
KNN Accuracy: 0.7667
```

In [268]:
```
# Feature Importance Plot (using permutation importance instead)
from sklearn.inspection import permutation_importance
plt.figure(figsize=(10, 6))

# Calculate permutation importance
result = permutation_importance(knn_grid_search, X_test, y_test, n_repea
importances_knn = result.importances_mean
indices_knn = np.argsort(importances_knn)[::-1]

plt.title("Feature Importances (K-Nearest Neighbors)")
plt.bar(range(X.shape[1]), importances_knn[indices_knn], align="center")
plt.xticks(range(X.shape[1]), X.columns[indices_knn], rotation=90)
plt.show()
```



In [269]:
```
#Support Vector Machine (SVM) Extension
```

In [270]:
```
# Import the necessary class from the sklearn.svm module
from sklearn.svm import SVC
```

In [271]:
```
# SVM Classifier with Hyperparameter Tuning
svm = SVC(probability=True)
```

In [272]:
```
# Define the hyperparameters grid
param_grid_svm = {
    'C': [0.1, 1, 10, 100],
    'kernel': ['linear', 'poly', 'rbf', 'sigmoid'],
    'gamma': ['scale', 'auto']
}
```

In [273]:
```
# GridSearchCV for SVM
svm_grid_search = GridSearchCV(svm, param_grid_svm, cv=5, scoring='accur
svm_grid_search.fit(X_train, y_train)
```

Out[273]:
```
GridSearchCV(cv=5, estimator=SVC(probability=True),
             param_grid={'C': [0.1, 1, 10, 100], 'gamma': ['scale', 'au
to'],
                         'kernel': ['linear', 'poly', 'rbf', 'sigmoi
d']},
             scoring='accuracy')
```

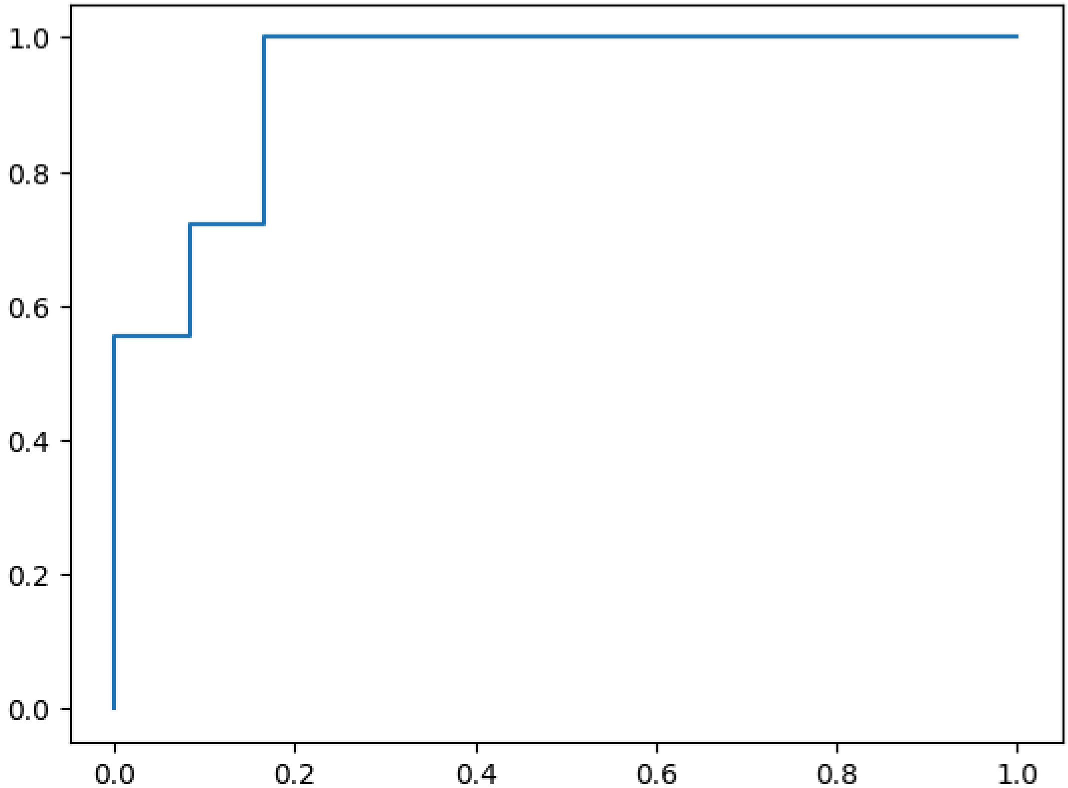
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [274]:
```

# Best Parameters and Evaluation
print("Best SVM Parameters:", svm_grid_search.best_params_)
y_pred_svm = svm_grid_search.predict(X_test)
accuracy_svm = accuracy_score(y_test, y_pred_svm)
print(f"SVM Accuracy: {accuracy_svm:.4f}")
```

```
Best SVM Parameters: {'C': 1, 'gamma': 'auto', 'kernel': 'sigmoid'}
SVM Accuracy: 0.8667
```

In [275]:
```
# Plotting ROC Curve for SVM
y_pred_prob_svm = svm_grid_search.predict_proba(X_test)[:, 1]
fpr_svm, tpr_svm, _ = roc_curve(y_test, y_pred_prob_svm)
plt.plot(fpr_svm, tpr_svm, label=f'SVM (AUC = {accuracy_svm:.4f})')
```

Out[275]:
```
[<matplotlib.lines.Line2D at 0x7f9d1ccb3f40>]
```



In [276]:
```
#Gradient Boosting Extension
```

In [277]:
```
# Gradient Boosting Extension
from sklearn.ensemble import GradientBoostingClassifier  # Import Gradie
```

In [278]:
```

# Gradient Boosting Classifier with Hyperparameter Tuning
gb = GradientBoostingClassifier(random_state=42)
```

In [279]:
```
# Define the hyperparameters grid
param_grid_gb = {
    'n_estimators': [100, 200, 300],
    'learning_rate': [0.01, 0.1, 0.2],
    'max_depth': [3, 4, 5],
    'min_samples_split': [2, 5, 10]
}
```

In [280]:
```
# GridSearchCV for Gradient Boosting
gb_grid_search = GridSearchCV(gb, param_grid_gb, cv=5, scoring='accuracy
gb_grid_search.fit(X_train, y_train)
```

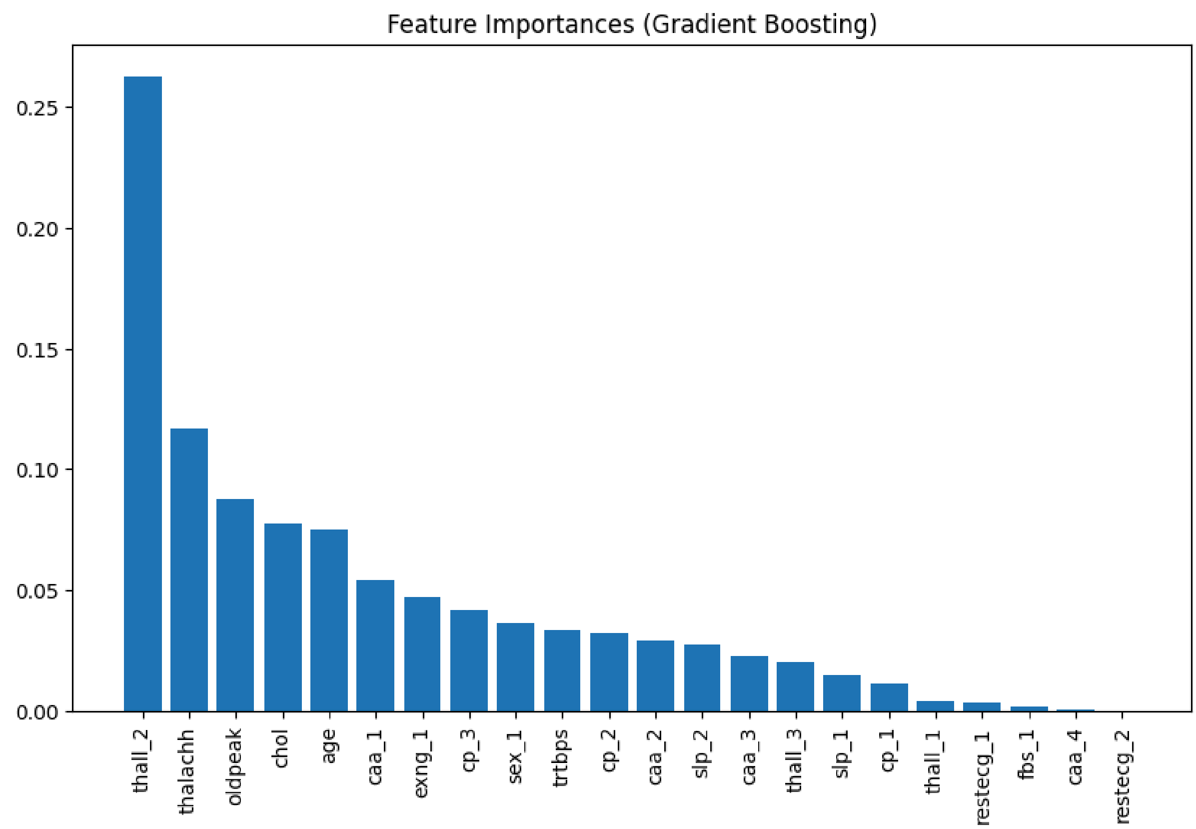
Out[280]:
```
GridSearchCV(cv=5, estimator=GradientBoostingClassifier(random_state=4
2),
             param_grid={'learning_rate': [0.01, 0.1, 0.2],
                         'max_depth': [3, 4, 5],
                         'min_samples_split': [2, 5, 10],
                         'n_estimators': [100, 200, 300]},
             scoring='accuracy')
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [281]:
```
# Best Parameters and Evaluation
print("Best Gradient Boosting Parameters:", gb_grid_search.best_params_)
y_pred_gb = gb_grid_search.predict(X_test)
accuracy_gb = accuracy_score(y_test, y_pred_gb)
print(f"Gradient Boosting Accuracy: {accuracy_gb:.4f}")
```

```
Best Gradient Boosting Parameters: {'learning_rate': 0.2, 'max_depth':
3, 'min_samples_split': 2, 'n_estimators': 100}
Gradient Boosting Accuracy: 0.7667
```

In [284]:
```
# Feature Importance Plot for Gradient Boosting
plt.figure(figsize=(10, 6))
importances_gb = gb_grid_search.best_estimator_.feature_importances_
indices_gb = np.argsort(importances_gb)[::-1]
plt.title("Feature Importances (Gradient Boosting)")
plt.bar(range(X.shape[1]), importances_gb[indices_gb], align="center")
plt.xticks(range(X.shape[1]), X.columns[indices_gb], rotation=90)
plt.show()
```

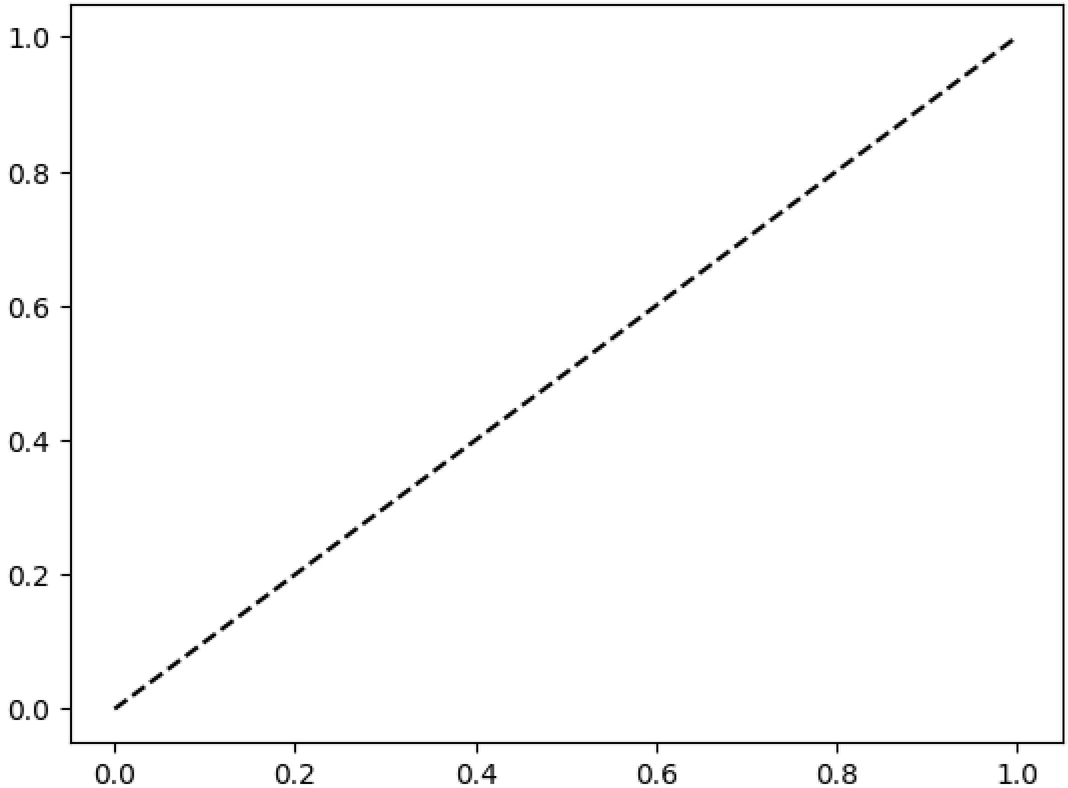


In [286]:
```
#ROC Curves for All Models (Combined)

# Plot ROC Curves for all models
plt.plot([0, 1], [0, 1], "k--")  # Reference line for random guessing

# Plot ROC curve for each model
# Assign a value to y_pred_prob_rf
y_pred_prob_rf = rf_grid_search.predict_proba(X_test)[:, 1]  # Assuming

if y_pred_prob_rf is not None:
    fpr_rf, tpr_rf, _ = roc_curve(y_test, y_pred_prob_rf)
    plt.plot(fpr_rf, tpr_rf, label=f'Random Forest (AUC = {accuracy_rf:.

# Plot ROC curve for each model
# Calculate and assign a value to y_pred_prob_dt
y_pred_prob_dt = dt_grid_search.predict_proba(X_test)[:, 1]  # Assuming

if y_pred_prob_dt is not None:
    fpr_dt, tpr_dt, _ = roc_curve(y_test, y_pred_prob_dt)
    plt.plot(fpr_dt, tpr_dt, label=f'Decision Tree (AUC = {accuracy_dt:.

if y_pred_prob_svm is not None:
    plt.plot(fpr_svm, tpr_svm, label=f'SVM (AUC = {accuracy_svm:.4f})')

# Plot ROC curve for each model
# Calculate and assign a value to y_pred_prob_gb
y_pred_prob_gb = gb_grid_search.predict_proba(X_test)[:, 1]  # Assuming

if y_pred_prob_gb is not None:
    fpr_gb, tpr_gb, _ = roc_curve(y_test, y_pred_prob_gb)
    plt.plot(fpr_gb, tpr_gb, label=f'Gradient Boosting (AUC = {accuracy_

plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curves for All Models")
plt.legend(loc="lower r
```

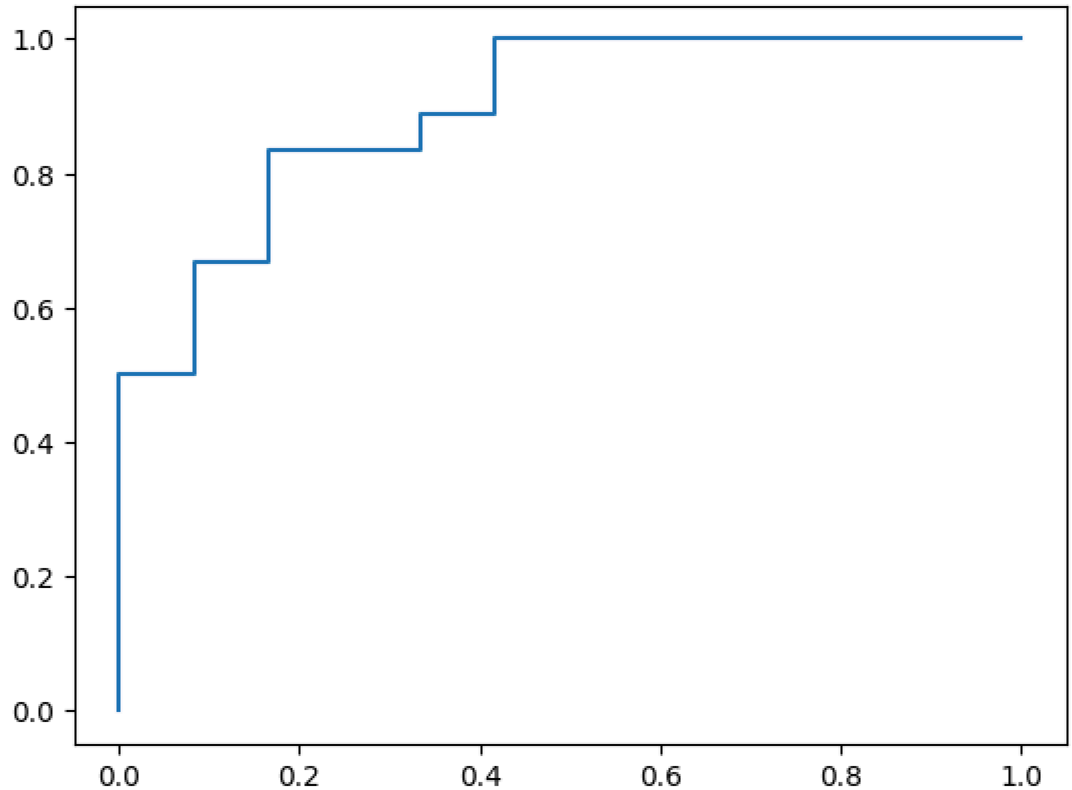
```
  File "<ipython-input-286-caac7b6e1a2a>", line 36
    plt.legend(loc="lower r
                   ^
SyntaxError: unterminated string literal (detected at line 36)
```

In [287]:
```
# Plot ROC Curves for all models
plt.plot([0, 1], [0, 1], "k--")  # Reference line for random guessing
```

Out[287]: [<matplotlib.lines.Line2D at 0x7f9d258d6410>]

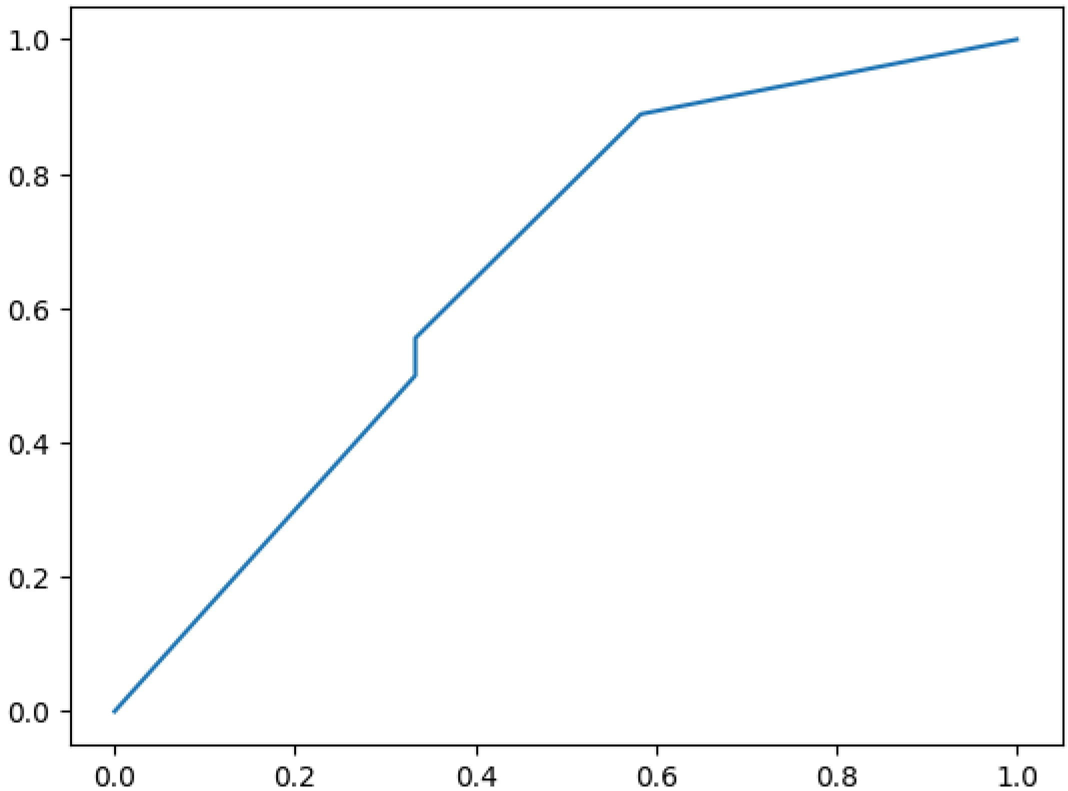


In [288]:
```
# Plot ROC curve for each model
# Assign a value to y_pred_prob_rf
y_pred_prob_rf = rf_grid_search.predict_proba(X_test)[:, 1]  # Assuming

if y_pred_prob_rf is not None:
    fpr_rf, tpr_rf, _ = roc_curve(y_test, y_pred_prob_rf)
    plt.plot(fpr_rf, tpr_rf, label=f'Random Forest (AUC = {accuracy_rf:.
```

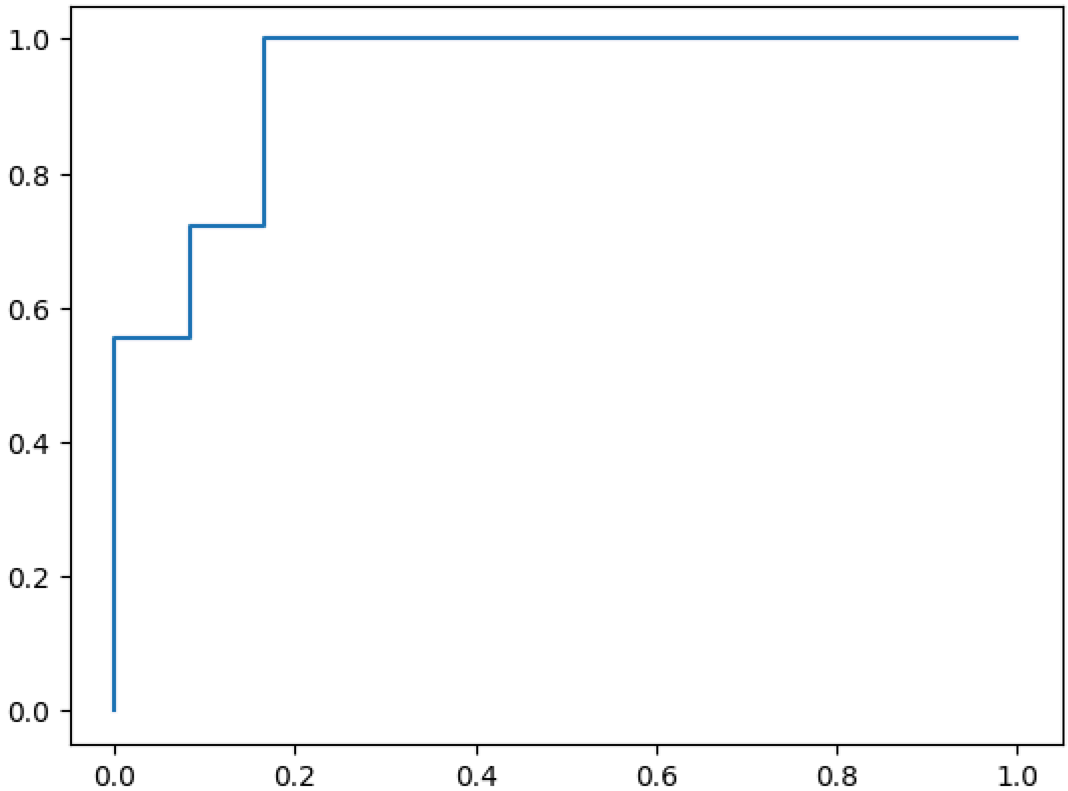


In [289]:
```
# Plot ROC curve for each model
# Calculate and assign a value to y_pred_prob_dt
y_pred_prob_dt = dt_grid_search.predict_proba(X_test)[:, 1]  # Assuming
```

In [290]:
```
if y_pred_prob_dt is not None:
    fpr_dt, tpr_dt, _ = roc_curve(y_test, y_pred_prob_dt)
    plt.plot(fpr_dt, tpr_dt, label=f'Decision Tree (AUC = {accuracy_dt:.
```

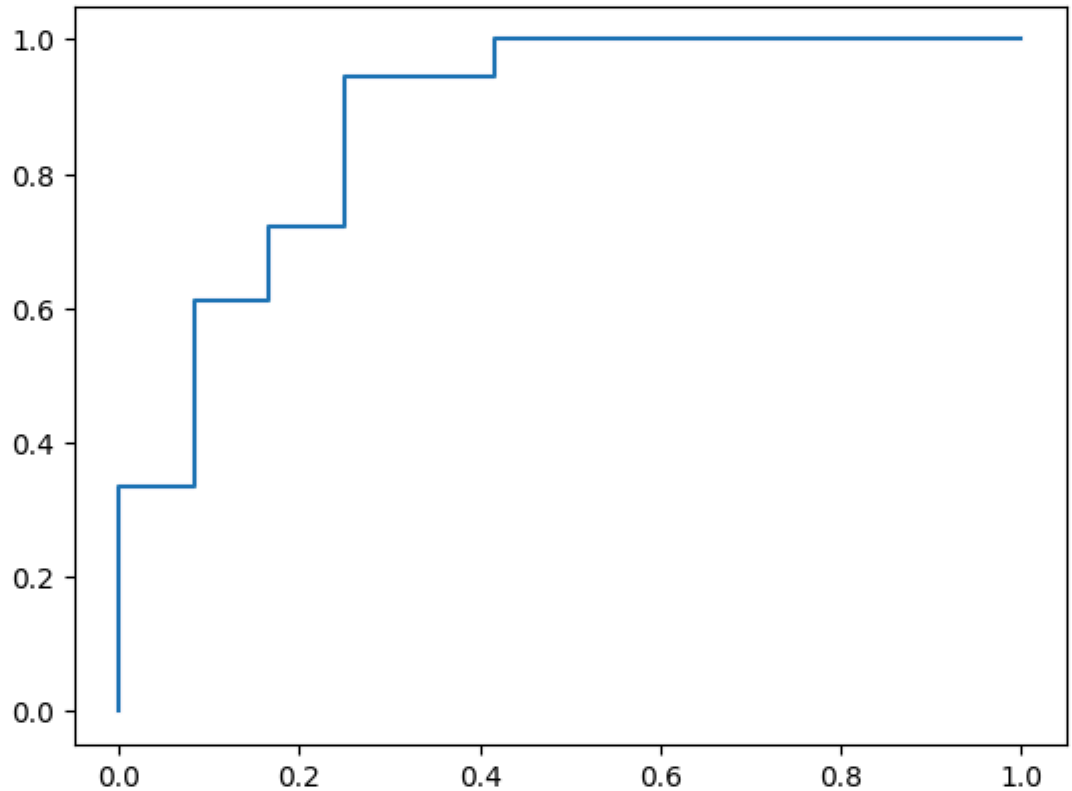


In [291]:
```
if y_pred_prob_svm is not None:
    plt.plot(fpr_svm, tpr_svm, label=f'SVM (AUC = {accuracy_svm:.4f})')
```



In [292]:
```
# Plot ROC curve for each model
# Calculate and assign a value to y_pred_prob_gb
y_pred_prob_gb = gb_grid_search.predict_proba(X_test)[:, 1]  # Assuming 
```

In [293]:
```
if y_pred_prob_gb is not None:
    fpr_gb, tpr_gb, _ = roc_curve(y_test, y_pred_prob_gb)
    plt.plot(fpr_gb, tpr_gb, label=f'Gradient Boosting (AUC = {accuracy_
```

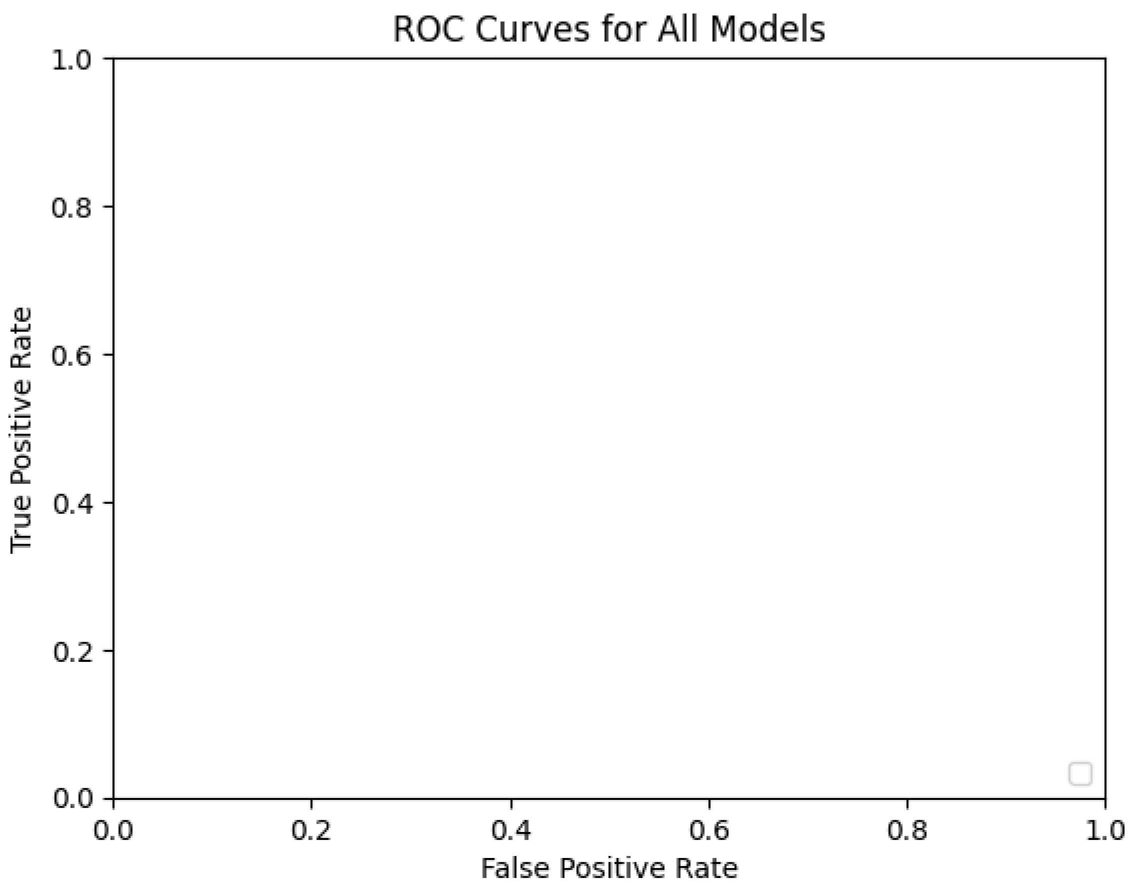


In [295]:

```
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curves for All Models")
plt.legend(loc="lower right")
```

```
WARNING:matplotlib.legend:No artists with labels found to put in legen
d.  Note that artists whose label start with an underscore are ignored
when legend() is called with no argument.
```

Out[295]: <matplotlib.legend.Legend at 0x7f9d257e1510>



In [ ]: