Problem Statement

You are hired by one of the leading news channel CNBE who wants to analyze recent elections. This survey was conducted on 1525 voters with 9 variables. You have to build a model, to predict which party a voter will vote for on the basis of the given information, to create an exit poll that will help in predicting overall win and seats covered by a particular party. Our Dataset has following variables:

Data Dictionary

1. vote: Party choice: Conservative or Labour
2. age: in years
3. economic.cond.national: Assessment of current national economic conditions, 1 to 5.
4. economic.cond.household: Assessment of current household economic conditions, 1 to 5.
5. Blair: Assessment of the Labour leader, 1 to 5.
6. Hague: Assessment of the Conservative leader, 1 to 5.
7. Europe: an 11-point scale that measures respondents' attitudes toward European integration. High scores represent 'Eurosceptic' sentiment.
8. political.knowledge: Knowledge of parties' positions on European integration, 0 to 3.
9. gender: female or male.

1. Read the dataset. Do the descriptive statistics and do null value condition check. Write an inference on it. (5 Marks)

Upload Required Libraries

In [965]:
```
import numpy as np
import pandas as pd
import os

import seaborn as sns
import matplotlib.pyplot as plt
import matplotlib.style
plt.style.use('classic')


import warnings
warnings.filterwarnings("ignore")
```

Importing data

In [966]:
```
## Load the csv file available in the working or specified directory

df = pd.read_excel("Election_Data.xlsx")
```

EDA

In [967]:
```
# Check top few records to get a feel of the data structure


df.head()
```

Out[967]:

	Unnamed: 0	vote	age	economic.cond.national	economic.cond.household	Blair	Hague	I
0	1	Labour	43	3	3	4	1	
1	2	Labour	36	4	4	4	4	
2	3	Labour	35	4	4	5	2	
3	4	Labour	24	4	2	2	1	
4	5	Labour	41	2	2	1	1	

Shape

In [968]:
```
print("No of rows: ",df.shape[0], "\n""No of columns: ",df.shape[1])
```

```
No of rows:  1525
No of columns:  10
```

Data type of data features

In [969]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1525 entries, 0 to 1524
Data columns (total 10 columns):
 #   Column                   Non-Null Count  Dtype 
---  ------                   --------------  ----- 
 0   Unnamed: 0               1525 non-null   int64 
 1   vote                     1525 non-null   object
 2   age                      1525 non-null   int64 
 3   economic.cond.national   1525 non-null   int64 
 4   economic.cond.household  1525 non-null   int64 
 5   Blair                    1525 non-null   int64 
 6   Hague                    1525 non-null   int64 
 7   Europe                   1525 non-null   int64 
 8   political.knowledge      1525 non-null   int64 
 9   gender                   1525 non-null   object
dtypes: int64(8), object(2)
memory usage: 119.3+ KB
```

checking for Possible columns which are categorical but are having data type "object"

In [970]:
```
df['vote'].value_counts()
```

Out[970]:
```
Labour          1063
Conservative     462
Name: vote, dtype: int64
```

In [971]:
```
df['gender'].value_counts()
```

Out[971]:
```
female    812
male      713
Name: gender, dtype: int64
```

Remove unnamed column

In [972]:
```
df.drop(df.columns[0],axis=1,inplace=True)
```

In [973]:
```
df
```

Out[973]:

	vote	age	economic.cond.national	economic.cond.household	Blair	Hague	Eur
0	Labour	43	3	3	4	1	
1	Labour	36	4	4	4	4	
2	Labour	35	4	4	5	2	
3	Labour	24	4	2	2	1	
4	Labour	41	2	2	1	1	
...	...	...	...	...	...	...	
1520	Conservative	67	5	3	2	4	
1521	Conservative	73	2	2	4	4	
1522	Labour	37	3	3	5	4	
1523	Conservative	61	3	3	1	4	
1524	Conservative	74	2	3	2	4	

1525 rows × 9 columns

checking for Possible columns which are categorical but are not having data type "object"

In [974]:
```
df['economic.cond.national'].value_counts()
```

Out[974]:
```
3    607
4    542
2    257
5     82
1     37
Name: economic.cond.national, dtype: int64
```

In [975]:
```
df['economic.cond.household'].value_counts()
```

Out[975]:
```
3    648
4    440
2    280
5     92
1     65
Name: economic.cond.household, dtype: int64
```

In [976]:
```
df['Blair'].value_counts()
```

Out[976]:
```
4    836
2    438
5    153
1     97
3      1
Name: Blair, dtype: int64
```

In [977]:
```
df['Hague'].value_counts()
```

Out[977]:
```
2    624
4    558
1    233
5     73
3     37
Name: Hague, dtype: int64
```

In [978]:
```
df['Europe'].value_counts()
```

Out[978]:
```
11    338
6     209
3     129
4     127
5     124
8     112
9     111
1     109
10    101
7      86
2      79
Name: Europe, dtype: int64
```

In [979]:
```
df['political.knowledge'].value_counts()
```

Out[979]:
```
2    782
0    455
3    250
1     38
Name: political.knowledge, dtype: int64
```

Change the data types of these 6 features

***Convert political.knowledge to object type as it is having 0 which will affect in the further process ***

In [980]:
```
cat=["economic.cond.national","economic.cond.household","Blair","Hague","Eu
```

In [981]:
```
for i in cat:
    df[i]=df[i].astype("object")
```

In [982]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1525 entries, 0 to 1524
Data columns (total 9 columns):
 #   Column                   Non-Null Count  Dtype 
---  ------                   --------------  ----- 
 0   vote                     1525 non-null   object
 1   age                      1525 non-null   int64 
 2   economic.cond.national   1525 non-null   object
 3   economic.cond.household  1525 non-null   object
 4   Blair                    1525 non-null   object
 5   Hague                    1525 non-null   object
 6   Europe                   1525 non-null   object
 7   political.knowledge      1525 non-null   object
 8   gender                   1525 non-null   object
dtypes: int64(1), object(8)
memory usage: 107.4+ KB
```

Making different list for categorical columns and numerical columns

In [983]:
```
cat=[]
num=[]
for i in df.columns:
    if df[i].dtype=="object":
        cat.append(i)
    else:
        num.append(i)
print(cat)

print(num)
```

```
['vote', 'economic.cond.national', 'economic.cond.household', 'Blair', 'Ha
gue', 'Europe', 'political.knowledge', 'gender']
['age']
```

In [984]:
```
df.info()
```


```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1525 entries, 0 to 1524
Data columns (total 9 columns):
 #   Column                   Non-Null Count  Dtype 
---  ------                   --------------  ----- 
 0   vote                     1525 non-null   object
 1   age                      1525 non-null   int64 
 2   economic.cond.national   1525 non-null   object
 3   economic.cond.household  1525 non-null   object
 4   Blair                    1525 non-null   object
 5   Hague                    1525 non-null   object
 6   Europe                   1525 non-null   object
 7   political.knowledge      1525 non-null   object
 8   gender                   1525 non-null   object
dtypes: int64(1), object(8)
memory usage: 107.4+ KB
```

In [985]:
```
df.head()
```

Out[985]:

	vote	age	economic.cond.national	economic.cond.household	Blair	Hague	Europe	poli
0	Labour	43	3	3	4	1	2	
1	Labour	36	4	4	4	4	5	
2	Labour	35	4	4	5	2	3	
3	Labour	24	4	2	2	1	4	
4	Labour	41	2	2	1	1	6	

Describe for numerical and categorical columns

In [986]:
```
df[num].describe().T
```

Out[986]:

	count	mean	std	min	25%	50%	75%	max
age	1525.0	54.182295	15.711209	24.0	41.0	53.0	67.0	93.0

In [987]:
```
df[cat].describe().T
```

Out[987]:

	count	unique	top	freq
vote	1525	2	Labour	1063
economic.cond.national	1525	5	3	607
economic.cond.household	1525	5	3	648
Blair	1525	5	4	836
Hague	1525	5	2	624
Europe	1525	11	11	338
political.knowledge	1525	4	2	782
gender	1525	2	female	812

In [988]:
```
# Are there any missing values ?
df.isnull().sum()
```

Out[988]:
```
vote                       0
age                        0
economic.cond.national     0
economic.cond.household    0
Blair                      0
Hague                      0
Europe                     0
political.knowledge        0
gender                     0
dtype: int64
```

There are no missing values

In [989]:
```
## Are there any duplicate records

# Check for duplicate data

dups = df.duplicated()
print('Number of duplicate rows = %d' % (dups.sum()))
print(df.shape)
```

```
Number of duplicate rows = 8
(1525, 9)
```

In [990]:
```
df.drop_duplicates(inplace=True)
```

In [991]:
```
dups = df.duplicated()
print('Number of duplicate rows = %d' % (dups.sum()))
print(df.shape)
```

```
Number of duplicate rows = 0
(1517, 9)
```

Geting unique counts of Target

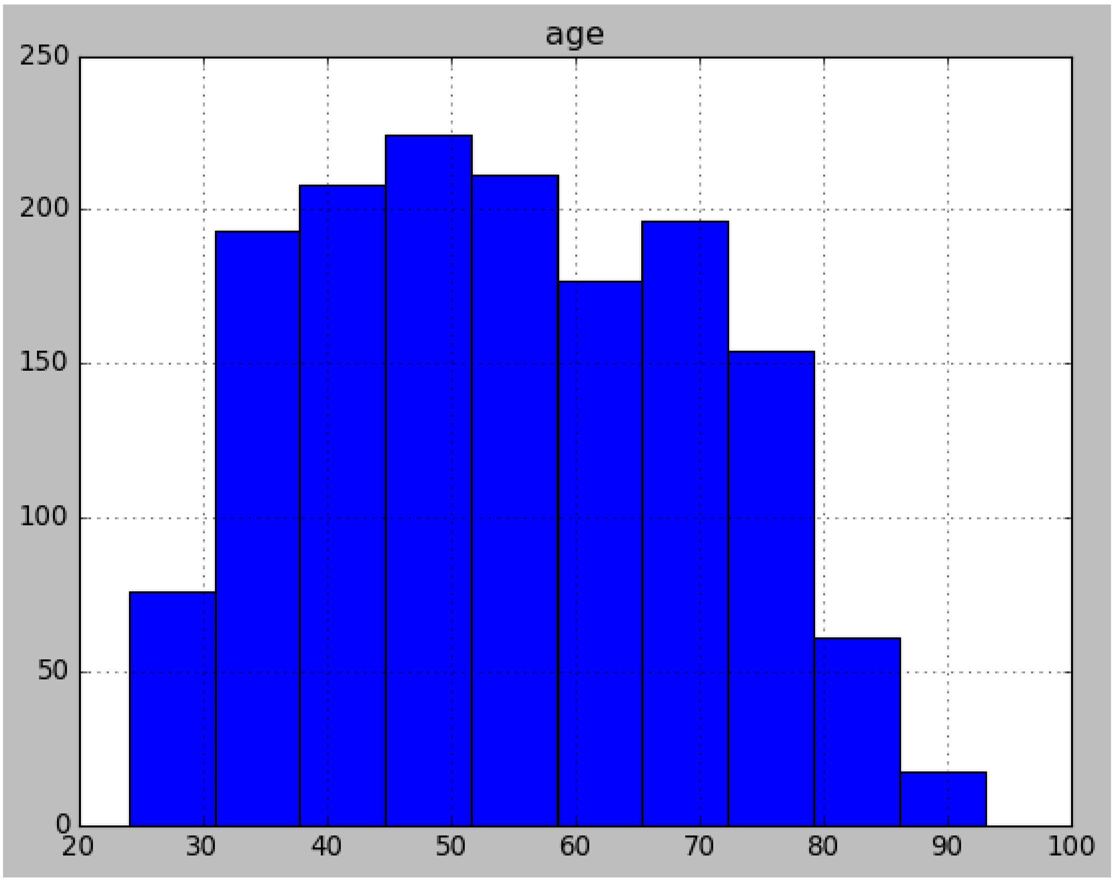
In [992]:
```
df.vote.value_counts(normalize=True)
```

Out[992]:
```
Labour          0.69677
Conservative    0.30323
Name: vote, dtype: float64
```

2. Perform Univariate and Bivariate Analysis. Do exploratory data analysis. Check for Outliers.

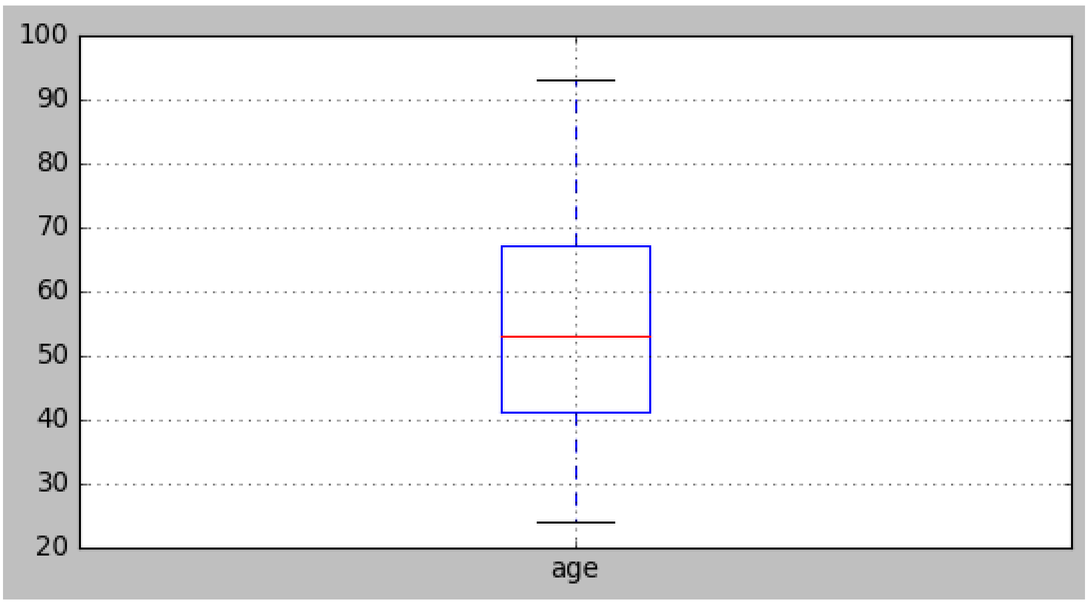
Univariate Analysis

In [993]:
```
fig = plt.figure(figsize = (8,6))
ax = fig.gca()
df.hist(ax=ax)
plt.show()
```



In [994]:
```
fig = plt.figure(figsize = (8,4))
df.boxplot()
```

Out[994]: <AxesSubplot:>



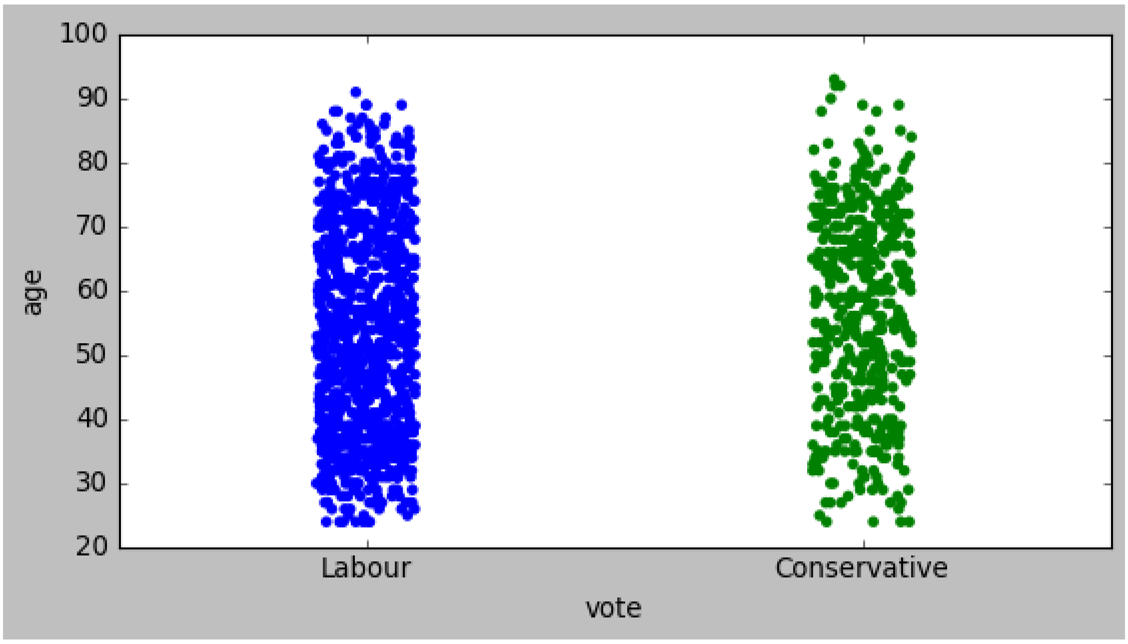
Bivariate and Multivariate Analysis

In [995]:
```
print(num)
```

```
['age']
```

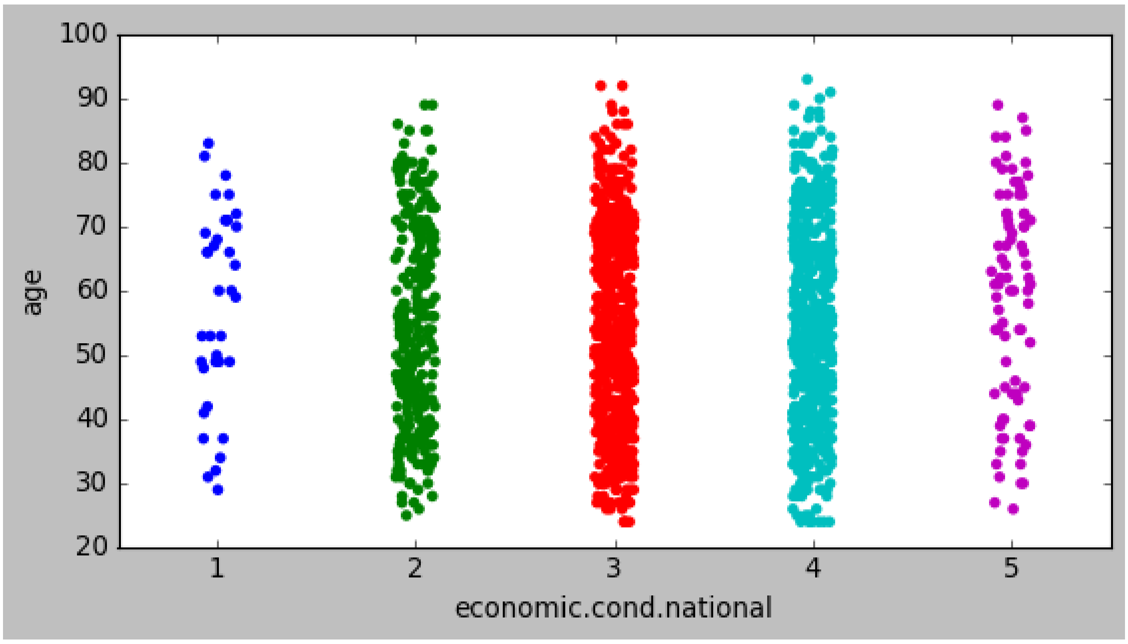
In [996]:
```
fig = plt.figure(figsize = (8,4))
sns.stripplot(df["vote"],df['age'],jitter = True)
```

Out[996]:
```
<AxesSubplot:xlabel='vote', ylabel='age'>
```



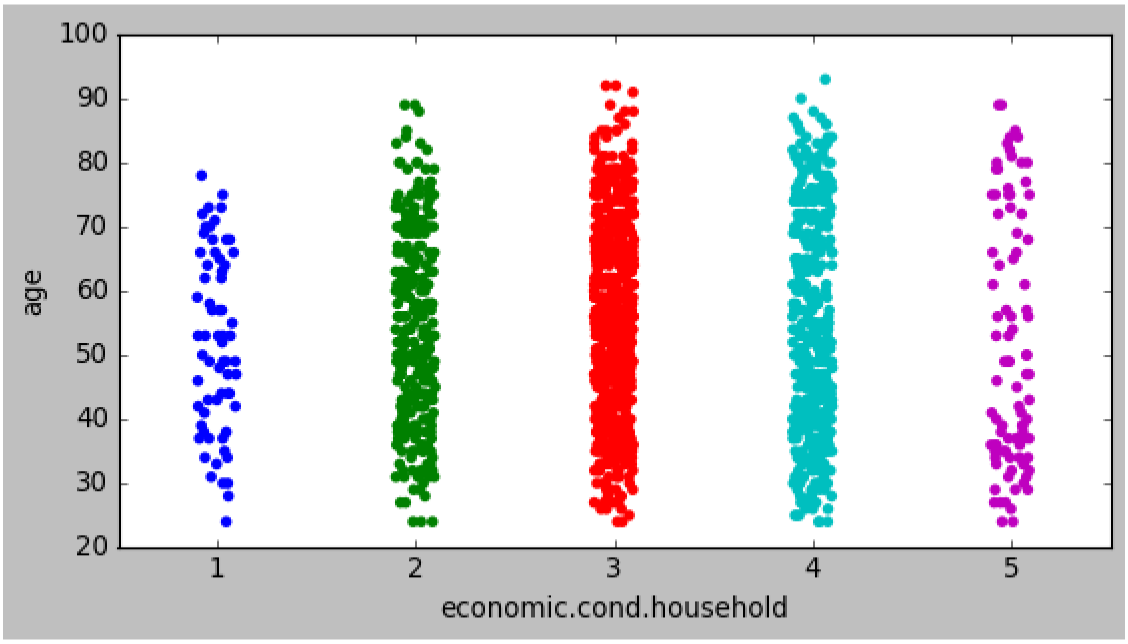
In [997]:
```
fig = plt.figure(figsize = (8,4))
sns.stripplot(df["economic.cond.national"],df['age'],jitter = True)
```

Out[997]:
```
<AxesSubplot:xlabel='economic.cond.national', ylabel='age'>
```



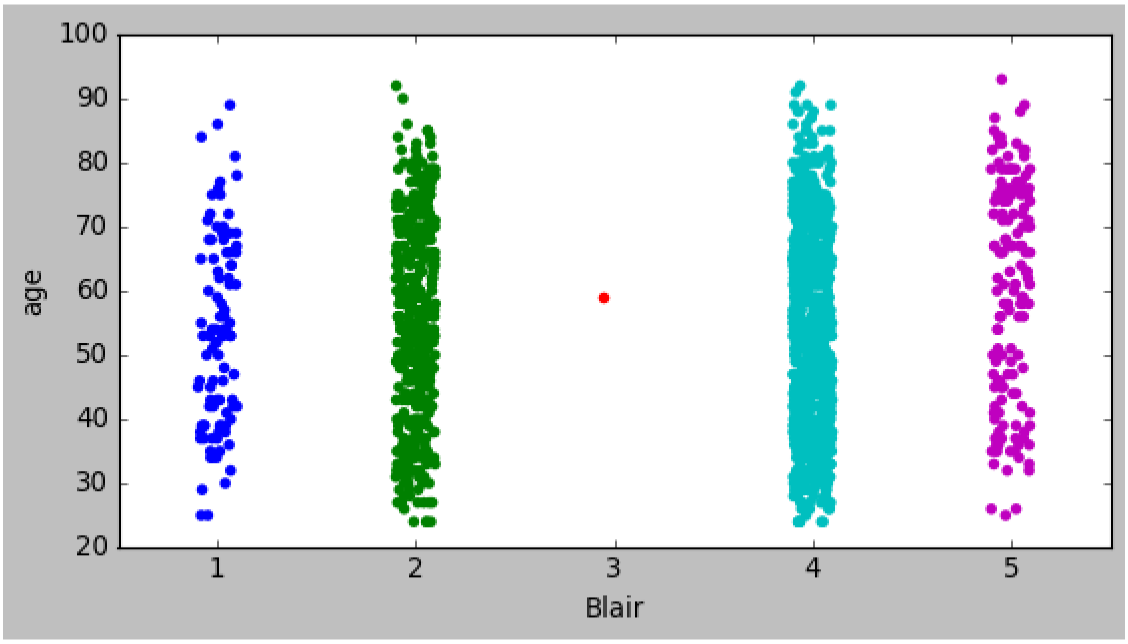
In [998]:
```
fig = plt.figure(figsize = (8,4))
sns.stripplot(df["economic.cond.household"],df['age'],jitter = True)
```

Out[998]: <AxesSubplot:xlabel='economic.cond.household', ylabel='age'>



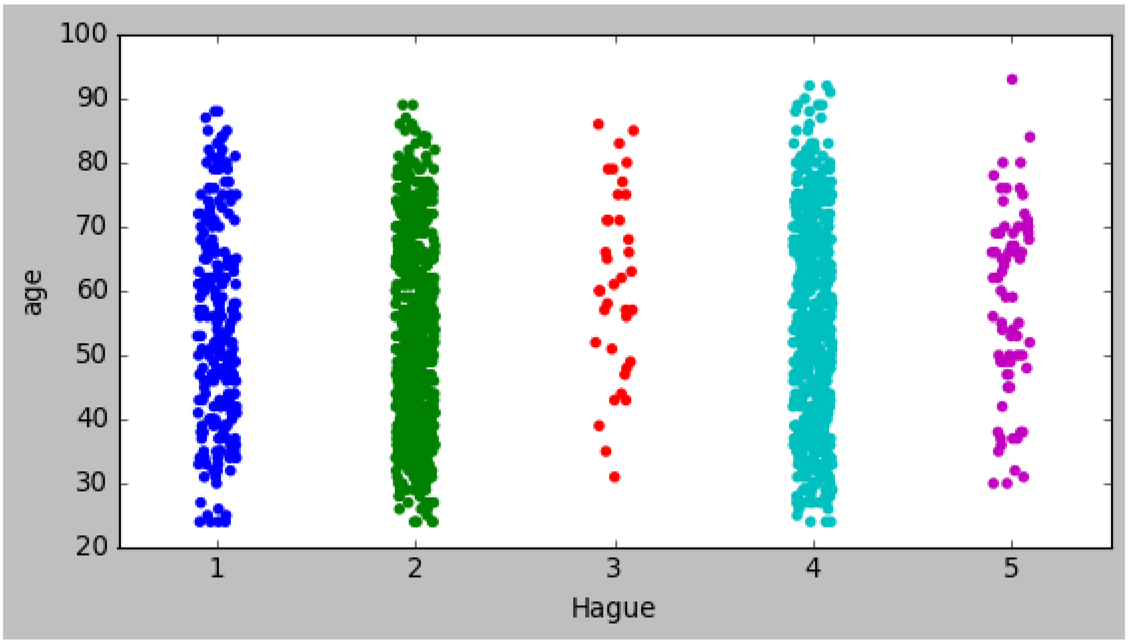
In [999]:
```
fig = plt.figure(figsize = (8,4))
sns.stripplot(df["Blair"],df['age'],jitter = True)
```

Out[999]: <AxesSubplot:xlabel='Blair', ylabel='age'>



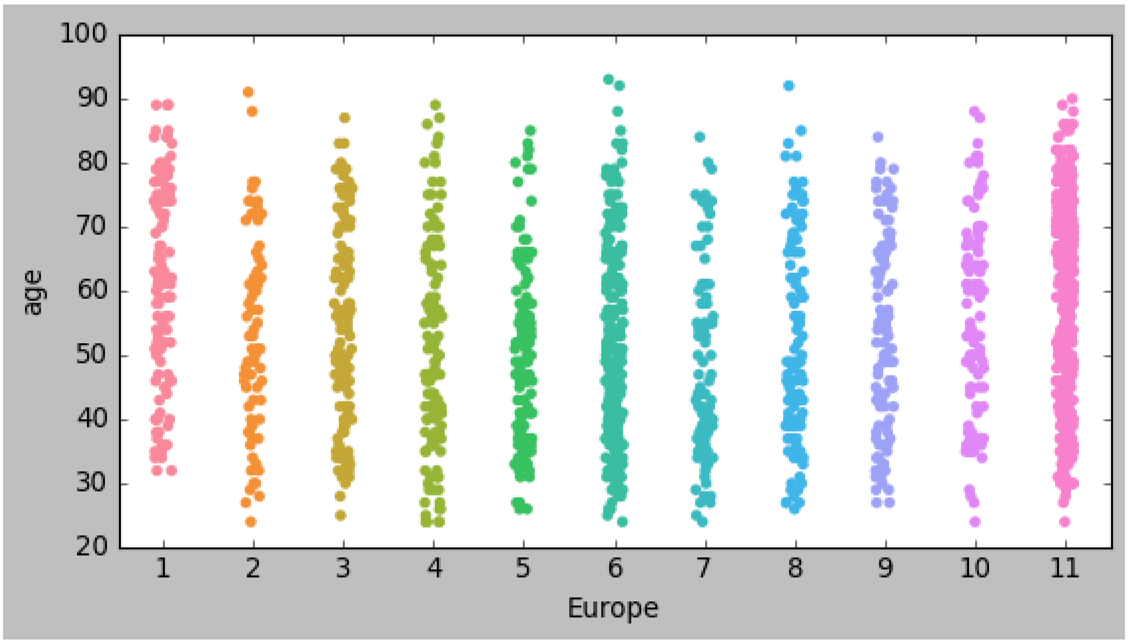
In [1000]:
```
fig = plt.figure(figsize = (8,4))
sns.stripplot(df["Hague"],df['age'],jitter = True)
```

Out[1000]: <AxesSubplot:xlabel='Hague', ylabel='age'>



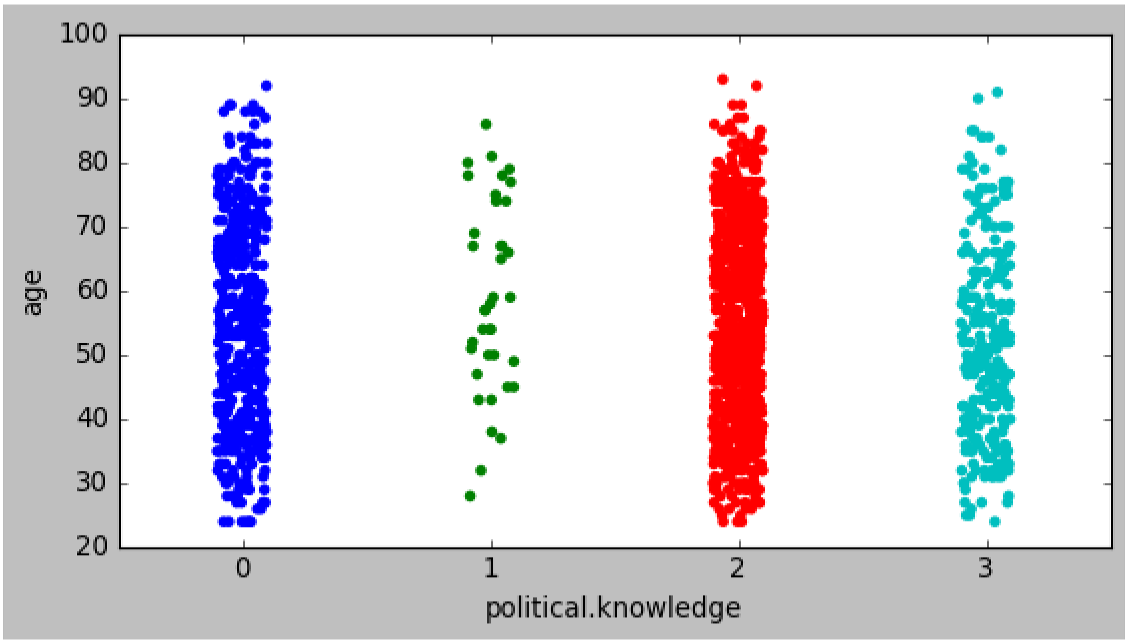
In [1001]:
```
fig = plt.figure(figsize = (8,4))
sns.stripplot(df["Europe"],df['age'],jitter = True)
```

Out[1001]: <AxesSubplot:xlabel='Europe', ylabel='age'>



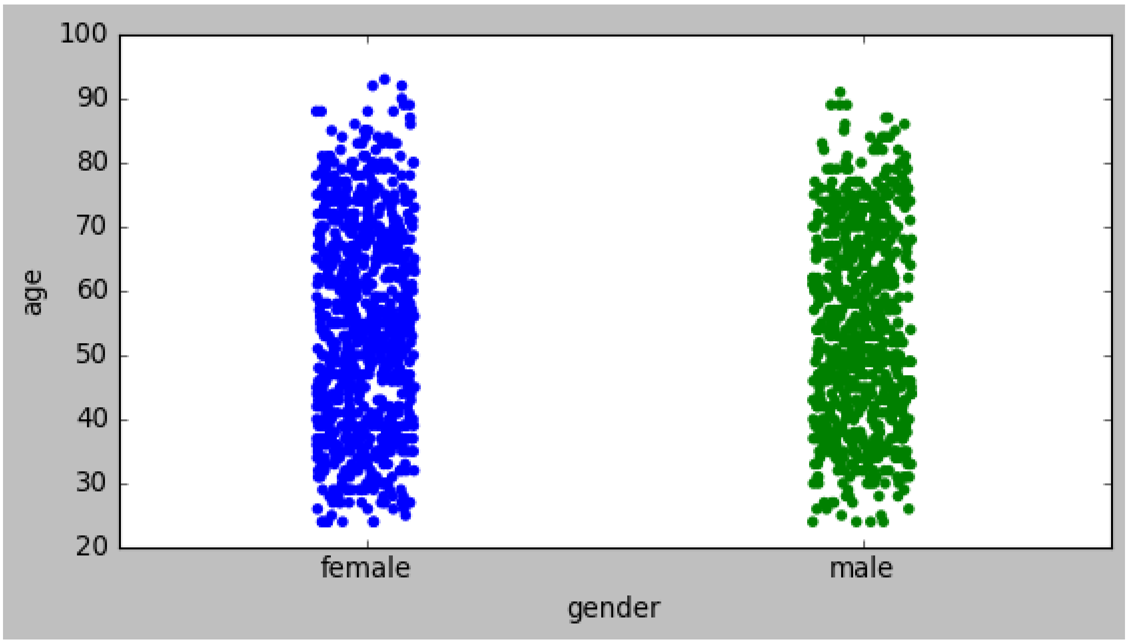
In [1002]:
```
fig = plt.figure(figsize = (8,4))
sns.stripplot(df["political.knowledge"],df['age'],jitter = True)
```

Out[1002]: <AxesSubplot:xlabel='political.knowledge', ylabel='age'>



In [1003]:
```
fig = plt.figure(figsize = (8,4))
sns.stripplot(df["gender"],df['age'],jitter = True)
```

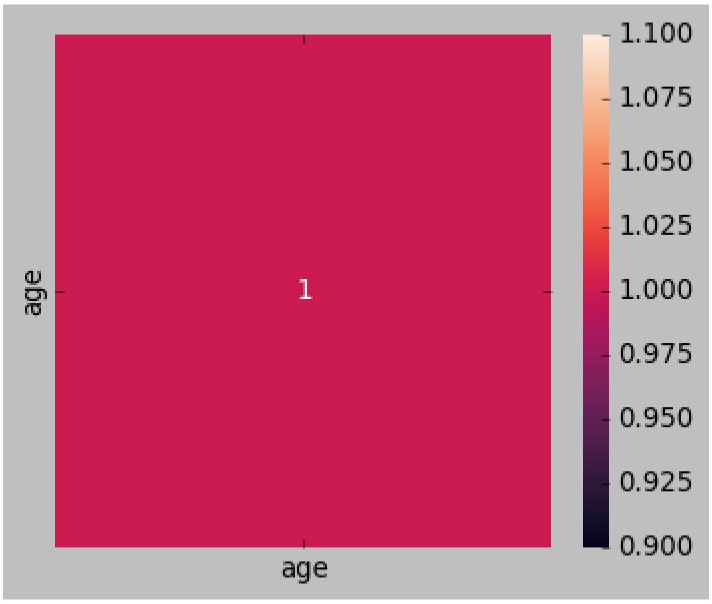
Out[1003]: <AxesSubplot:xlabel='gender', ylabel='age'>



In [ ]:

Correlation Plot

In [1004]:
```
plt.figure(figsize=(5,4))
sns.heatmap(df.corr(),annot=True)
plt.show()
```



Since there is only one continuous variable, correlation cant be known for other features

In [1006]:
```
df.corr()
```

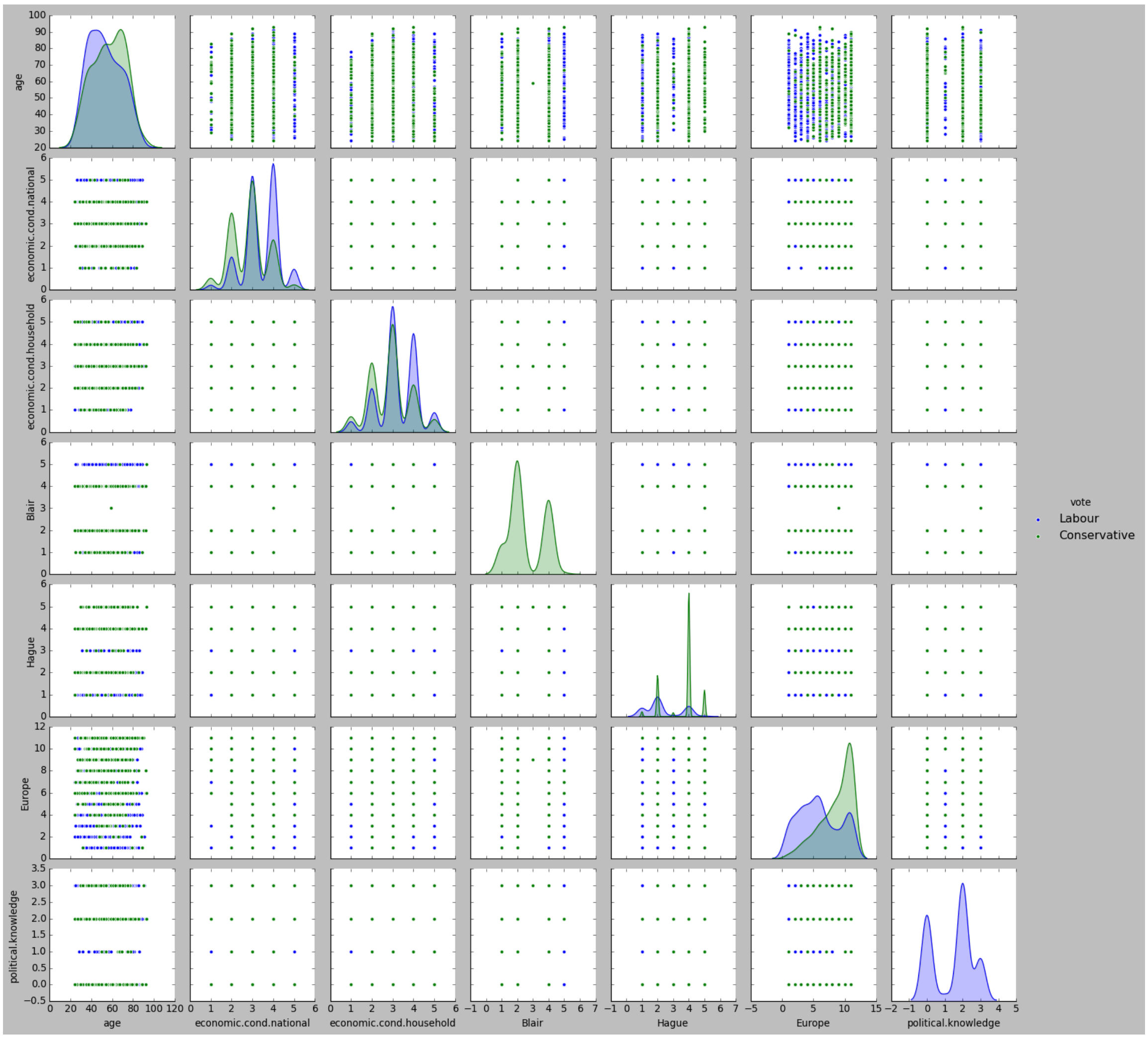
Out[1006]:

	age
age	1.0

In [1007]:
```
plt.figure(figsize=(4,4))
sns.pairplot(df,hue = 'vote', diag_kind = 'kde')
```

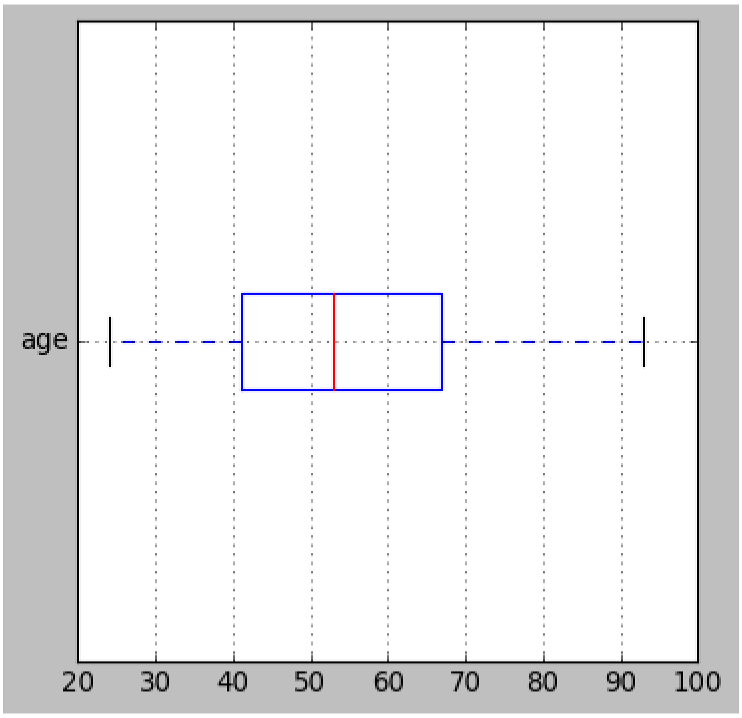
Out[1007]: <seaborn.axisgrid.PairGrid at 0x168c995cfd0>

<Figure size 320x320 with 0 Axes>



Outlier Checks

In [1008]:
```
# construct box plot for continuous variables
plt.figure(figsize=(5,5))
df.iloc[:,:7].boxplot(vert=0)
plt.show()
```



In [1009]:
```
for feature in df.columns:
    if df[feature].dtype == 'object':
        print(feature)
        print(df[feature].value_counts())
        print('\n')
```

```
vote
Labour          1057
Conservative     460
Name: vote, dtype: int64


economic.cond.national
3    604
4    538
2    256
5     82
1     37
Name: economic.cond.national, dtype: int64


economic.cond.household
3    645
4    435
2    280
5     92
1     65
Name: economic.cond.household, dtype: int64


Blair
4    833
2    434
5    152
1     97
3      1
Name: Blair, dtype: int64


Hague
2    617
4    557
1    233
5     73
3     37
Name: Hague, dtype: int64


Europe
11    338
6     207
3     128
4     126
5     123
9     111
8     111
1     109
10    101
7      86
2      77
Name: Europe, dtype: int64


political.knowledge
2    776
0    454
3    249
```

```
1      38
Name: political.knowledge, dtype: int64


gender
female    808
male      709
Name: gender, dtype: int64
```

Data Preparation:

1. Encode the data (having string values) for Modelling. Is Scaling necessary here or not? Data Split: Split the data into train and test (70:30). **

Converting all objects to categorical codes

In [1010]:
```
for feature in df.columns:
    if df[feature].dtype == 'object':
        print('\n')
        print('feature:',feature)
        print(pd.Categorical(df[feature].unique()))
        print(pd.Categorical(df[feature].unique()).codes)
        df[feature] = pd.Categorical(df[feature]).codes
```

```


feature: vote
['Labour', 'Conservative']
Categories (2, object): ['Conservative', 'Labour']
[1 0]


feature: economic.cond.national
[3, 4, 2, 1, 5]
Categories (5, int64): [1, 2, 3, 4, 5]
[2 3 1 0 4]


feature: economic.cond.household
[3, 4, 2, 1, 5]
Categories (5, int64): [1, 2, 3, 4, 5]
[2 3 1 0 4]


feature: Blair
[4, 5, 2, 1, 3]
Categories (5, int64): [1, 2, 3, 4, 5]
[3 4 1 0 2]


feature: Hague
[1, 4, 2, 5, 3]
Categories (5, int64): [1, 2, 3, 4, 5]
[0 3 1 4 2]


feature: Europe
[2, 5, 3, 4, 6, ..., 1, 7, 9, 10, 8]
Length: 11
Categories (11, int64): [1, 2, 3, 4, ..., 8, 9, 10, 11]
[ 1  4  2  3  5 10  0  6  8  9  7]


feature: political.knowledge
[2, 0, 3, 1]
Categories (4, int64): [0, 1, 2, 3]
[2 0 3 1]


feature: gender
['female', 'male']
Categories (2, object): ['female', 'male']
[0 1]
```

In [1011]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1517 entries, 0 to 1524
Data columns (total 9 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   vote                     1517 non-null   int8
 1   age                      1517 non-null   int64
 2   economic.cond.national   1517 non-null   int8
 3   economic.cond.household  1517 non-null   int8
 4   Blair                    1517 non-null   int8
 5   Hague                    1517 non-null   int8
 6   Europe                   1517 non-null   int8
 7   political.knowledge      1517 non-null   int8
 8   gender                   1517 non-null   int8
dtypes: int64(1), int8(8)
memory usage: 75.6 KB
```

In [1012]:
```
df.head()
```

Out[1012]:

	vote	age	economic.cond.national	economic.cond.household	Blair	Hague	Europe	politic
0	1	43	2	2	3	0	1	
1	1	36	3	3	3	3	4	
2	1	35	3	3	4	1	2	
3	1	24	3	1	1	0	3	
4	1	41	1	1	0	0	5	

With Scaling

In [1013]:
```
cat1 = ['economic.cond.national', 'economic.cond.household', 'Blair', 'Hague
```

Scaling the variables as continuous variables have different weightage using min-max technique

In [1014]:
```
df =pd.get_dummies(df, columns=cat1,drop_first=True)
```

In [1015]:
```
df.head()
```

Out[1015]:

	vote	age	economic.cond.national_1	economic.cond.national_2	economic.cond.national_3
0	1	43	0	1	0
1	1	36	0	0	1
2	1	35	0	0	1
3	1	24	0	0	1
4	1	41	1	0	0

5 rows × 32 columns

In [1016]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1517 entries, 0 to 1524
Data columns (total 32 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   vote                      1517 non-null   int8
 1   age                       1517 non-null   int64
 2   economic.cond.national_1  1517 non-null   uint8
 3   economic.cond.national_2  1517 non-null   uint8
 4   economic.cond.national_3  1517 non-null   uint8
 5   economic.cond.national_4  1517 non-null   uint8
 6   economic.cond.household_1 1517 non-null   uint8
 7   economic.cond.household_2 1517 non-null   uint8
 8   economic.cond.household_3 1517 non-null   uint8
 9   economic.cond.household_4 1517 non-null   uint8
 10  Blair_1                   1517 non-null   uint8
 11  Blair_2                   1517 non-null   uint8
 12  Blair_3                   1517 non-null   uint8
 13  Blair_4                   1517 non-null   uint8
 14  Hague_1                   1517 non-null   uint8
 15  Hague_2                   1517 non-null   uint8
 16  Hague_3                   1517 non-null   uint8
 17  Hague_4                   1517 non-null   uint8
 18  Europe_1                  1517 non-null   uint8
 19  Europe_2                  1517 non-null   uint8
 20  Europe_3                  1517 non-null   uint8
 21  Europe_4                  1517 non-null   uint8
 22  Europe_5                  1517 non-null   uint8
 23  Europe_6                  1517 non-null   uint8
 24  Europe_7                  1517 non-null   uint8
 25  Europe_8                  1517 non-null   uint8
 26  Europe_9                  1517 non-null   uint8
 27  Europe_10                 1517 non-null   uint8
 28  political.knowledge_1     1517 non-null   uint8
 29  political.knowledge_2     1517 non-null   uint8
 30  political.knowledge_3     1517 non-null   uint8
 31  gender_1                  1517 non-null   uint8
dtypes: int64(1), int8(1), uint8(30)
memory usage: 109.6 KB
```

In [1026]:
```
print(num)
```

```
['age']
```

In [1027]:
```
df[num] = df[num].apply(lambda x:(x-x.min()) / (x.max()-x.min()))
```

In [1028]:
```
## Check if the variables have been scaled or not
df.head()
```

Out[1028]:

	vote	age	economic.cond.national_1	economic.cond.national_2	economic.cond.nationa
0	1	0.275362	0	1	
1	1	0.173913	0	0	
2	1	0.159420	0	0	
3	1	0.000000	0	0	
4	1	0.246377	1	0	

5 rows × 32 columns

Train-Test Split

In [1029]:
```
df.columns
```

Out[1029]:
```
Index(['vote', 'age', 'economic.cond.national_1', 'economic.cond.national_
2',
       'economic.cond.national_3', 'economic.cond.national_4',
       'economic.cond.household_1', 'economic.cond.household_2',
       'economic.cond.household_3', 'economic.cond.household_4', 'Blair_
1',
       'Blair_2', 'Blair_3', 'Blair_4', 'Hague_1', 'Hague_2', 'Hague_3',
       'Hague_4', 'Europe_1', 'Europe_2', 'Europe_3', 'Europe_4', 'Europe_
5',
       'Europe_6', 'Europe_7', 'Europe_8', 'Europe_9', 'Europe_10',
       'political.knowledge_1', 'political.knowledge_2',
       'political.knowledge_3', 'gender_1'],
      dtype='object')
```

In [1030]:
```
# Copy all the predictor variables into X dataframe
X = df.drop('vote', axis=1)

# Copy target into the y dataframe.
y = df['vote']
```

In [1031]:
```
X.head()
```

Out[1031]:

	age	economic.cond.national_1	economic.cond.national_2	economic.cond.national_3	e
0	0.275362	0	1	0	
1	0.173913	0	0	1	
2	0.159420	0	0	1	
3	0.000000	0	0	1	
4	0.246377	1	0	0	

5 rows × 31 columns

In [1032]:
```
y.head()
```

Out[1032]:
```
0    1
1    1
2    1
3    1
4    1
Name: vote, dtype: int8
```

In [1033]:
```
# Split X and y into training and test set in 70:30 ratio
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 , ra
```

In [1034]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1517 entries, 0 to 1524
Data columns (total 32 columns):
 #   Column                       Non-Null Count  Dtype  
---  ------                       --------------  -----  
 0   vote                         1517 non-null   int8   
 1   age                          1517 non-null   float64
 2   economic.cond.national_1     1517 non-null   uint8  
 3   economic.cond.national_2     1517 non-null   uint8  
 4   economic.cond.national_3     1517 non-null   uint8  
 5   economic.cond.national_4     1517 non-null   uint8  
 6   economic.cond.household_1    1517 non-null   uint8  
 7   economic.cond.household_2    1517 non-null   uint8  
 8   economic.cond.household_3    1517 non-null   uint8  
 9   economic.cond.household_4    1517 non-null   uint8  
 10  Blair_1                      1517 non-null   uint8  
 11  Blair_2                      1517 non-null   uint8  
 12  Blair_3                      1517 non-null   uint8  
 13  Blair_4                      1517 non-null   uint8  
 14  Hague_1                      1517 non-null   uint8  
 15  Hague_2                      1517 non-null   uint8  
 16  Hague_3                      1517 non-null   uint8  
 17  Hague_4                      1517 non-null   uint8  
 18  Europe_1                     1517 non-null   uint8  
 19  Europe_2                     1517 non-null   uint8  
 20  Europe_3                     1517 non-null   uint8  
 21  Europe_4                     1517 non-null   uint8  
 22  Europe_5                     1517 non-null   uint8  
 23  Europe_6                     1517 non-null   uint8  
 24  Europe_7                     1517 non-null   uint8  
 25  Europe_8                     1517 non-null   uint8  
 26  Europe_9                     1517 non-null   uint8  
 27  Europe_10                    1517 non-null   uint8  
 28  political.knowledge_1        1517 non-null   uint8  
 29  political.knowledge_2        1517 non-null   uint8  
 30  political.knowledge_3        1517 non-null   uint8  
 31  gender_1                     1517 non-null   uint8  
dtypes: float64(1), int8(1), uint8(30)
memory usage: 109.6 KB
```

Modelling

1. Apply Logistic Regression and LDA (linear discriminant analysis).

Logistic Regression

In [1035]:
```
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
from sklearn.metrics import roc_auc_score,roc_curve,classification_report,co
```

In [1036]:
```
# Fit the Logistic Regression model
LR_model = LogisticRegression()
LR_model.fit(X_train, y_train)
```

Out[1036]:
```
LogisticRegression()
```

Predicting on Training and Test dataset

In [1037]:
```
ytrain_predict = LR_model.predict(X_train)
ytest_predict = LR_model.predict(X_test)
```

Getting the Predicted Classes and Probs

In [1038]:
```
ytest_predict_prob=LR_model.predict_proba(X_test)
pd.DataFrame(ytest_predict_prob).head()
```

Out[1038]:

	0	1
0	0.641286	0.358714
1	0.231681	0.768319
2	0.021009	0.978991
3	0.908414	0.091586
4	0.107228	0.892772

Logistic Regression Model Evaluation

In [1039]:
```
np.round(LR_model.coef_,decimals = 2)>0
```

Out[1039]:
```
array([[False, False,  True,  True,  True, False,  True,  True, False,
        False, False,  True,  True, False, False, False, False,  True,
         True, False,  True, False, False, False, False, False, False,
        False, False, False,  True]])
```

In [1040]:
```
from sklearn.feature_selection import RFE

predictor=X_train
selector = RFE(LR_model, n_features_to_select = 1)
selector = selector.fit(predictor,y_train)
selector.ranking_
```

Out[1040]:
```
array([ 5, 14, 29,  7,  6, 19, 30, 23, 18, 13, 31,  4,  1, 20, 28,  3,  2,
       27, 22, 17, 21, 26, 12, 10,  8,  9, 11, 25, 15, 16, 24])
```

In [1041]:
```
## Performance Matrix on train data set
y_train_predict = LR_model.predict(X_train)
model_score = LR_model.score(X_train, y_train)
print(model_score)
print(metrics.confusion_matrix(y_train, y_train_predict))
print(metrics.classification_report(y_train, y_train_predict))
```

```
0.8473138548539114
[[208  99]
 [ 63 691]]
              precision    recall  f1-score   support

           0       0.77      0.68      0.72       307
           1       0.87      0.92      0.90       754

    accuracy                           0.85      1061
   macro avg       0.82      0.80      0.81      1061
weighted avg       0.84      0.85      0.84      1061
```

In [1042]:
```
## Performance Matrix on test data set
y_test_predict = LR_model.predict(X_test)
model_score = LR_model.score(X_test, y_test)
print(model_score)
print(metrics.confusion_matrix(y_test, y_test_predict))
print(metrics.classification_report(y_test, y_test_predict))
```

```
0.8245614035087719
[[104  49]
 [ 31 272]]
              precision    recall  f1-score   support

           0       0.77      0.68      0.72       153
           1       0.85      0.90      0.87       303

    accuracy                           0.82       456
   macro avg       0.81      0.79      0.80       456
weighted avg       0.82      0.82      0.82       456
```

In [1068]:
```
#the coefficients for each of the independent attributes

for idx, col_name in enumerate(X_train.columns):
    print("The coefficient for {} is {}".format(col_name, LR_model.coef_[0]
```

```
The coefficient for age is -1.1744662597904778
The coefficient for economic.cond.national_1 is -0.5176819122872669
The coefficient for economic.cond.national_2 is 0.056863771958550656
The coefficient for economic.cond.national_3 is 0.9439766626917371
The coefficient for economic.cond.national_4 is 1.0658858565395501
The coefficient for economic.cond.household_1 is -0.3636165042756915
The coefficient for economic.cond.household_2 is 0.0524779099905074
The coefficient for economic.cond.household_3 is 0.2851715158228671
The coefficient for economic.cond.household_4 is -0.4602130111030272
The coefficient for Blair_1 is -0.7831170915375294
The coefficient for Blair_2 is 0.0
The coefficient for Blair_3 is 0.6346235469331488
The coefficient for Blair_4 is 1.9405714934299025
The coefficient for Hague_1 is -0.4037435130688747
The coefficient for Hague_2 is -0.1127543404691797
The coefficient for Hague_3 is -1.9511780874524054
The coefficient for Hague_4 is -2.910718778649979
The coefficient for Europe_1 is 0.12530103447480292
The coefficient for Europe_2 is 0.21303170378589245
The coefficient for Europe_3 is -0.5400940404325377
The coefficient for Europe_4 is 0.29612321343401515
The coefficient for Europe_5 is -0.14341107825619337
The coefficient for Europe_6 is -0.5982110667160351
The coefficient for Europe_7 is -1.2983305031961327
The coefficient for Europe_8 is -1.4993756060970698
The coefficient for Europe_9 is -1.254106350023709
The coefficient for Europe_10 is -1.2406509400842118
The coefficient for political.knowledge_1 is -0.21622747500023098
The coefficient for political.knowledge_2 is -0.7309469728065002
The coefficient for political.knowledge_3 is -0.6399423403123045
The coefficient for gender_1 is 0.21054384787334637
```

In [1070]:
```
# the intercept for the model

intercept = LR_model.intercept_[0]

print("The intercept for LR model is {}".format(intercept))
```

```
The intercept for LR model is 3.3258468981851084
```

In [1071]:
```
# R square on testing data (coeff of determinant)
LR_model.score(X_test, y_test)
```

Out[1071]: 0.8245614035087719

In [1072]:
```
# R square on training data
LR_model.score(X_train, y_train)
```

Out[1072]: 0.8473138548539114

In [1166]:
```
#RMSE on Training data
predicted_train=LR_model.fit(X_train, y_train).predict(X_train)
np.sqrt(metrics.mean_squared_error(y_train,predicted_train))
```

Out[1166]:
```
0.39075074554770667
```

In [1167]:
```
#RMSE on Testing data
predicted_test=LR_model.fit(X_train, y_train).predict(X_test)
np.sqrt(metrics.mean_squared_error(y_test,predicted_test))
```

Out[1167]:
```
0.4188539082916955
```

Discriminant Analysis

In [1043]:
```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
LDA_model= LinearDiscriminantAnalysis()
LDA_model.fit(X_train, y_train)
```

Out[1043]:
```
LinearDiscriminantAnalysis()
```

Predicting on Training and Test dataset

In [1044]:
```
ytrain_predict = LDA_model.predict(X_train)
ytest_predict = LDA_model.predict(X_test)
```

Getting the Predicted Classes and Probs

In [1045]:
```
ytest_predict_prob=LDA_model.predict_proba(X_test)
pd.DataFrame(ytest_predict_prob).head()
```

Out[1045]:

	0	1
0	0.676144	0.323856
1	0.184856	0.815144
2	0.012266	0.987734
3	0.953492	0.046508
4	0.079953	0.920047

LDA Model Evaluation

In [1046]:
```
np.round(LDA_model.coef_,decimals = 2)>0
```

Out[1046]:
```
array([[False,  True,  True,  True,  True, False, False, False, False,
        False, False,  True,  True, False, False, False, False, False,
        False, False, False, False, False, False, False, False, False,
        False, False, False,  True]])
```

In [1047]:
```
from sklearn.feature_selection import RFE

predictor=X_train
selector = RFE(LDA_model, n_features_to_select = 1)
selector = selector.fit(predictor,y_train)
selector.ranking_
```

Out[1047]:
```
array([ 9, 30, 14, 11, 10, 19, 27, 29, 18, 15, 31,  4,  3, 24, 28,  2,  1,
       23, 21, 17, 22, 20, 16,  8,  5,  6,  7, 25, 12, 13, 26])
```

In [1048]:
```
## Performance Matrix on train data set
y_train_predict = LDA_model.predict(X_train)
model_score = LDA_model.score(X_train, y_train)
print(model_score)
print(metrics.confusion_matrix(y_train, y_train_predict))
print(metrics.classification_report(y_train, y_train_predict))
```

```
0.8444863336475024
[[216  91]
 [ 74 680]]
              precision    recall  f1-score   support

           0       0.74      0.70      0.72       307
           1       0.88      0.90      0.89       754

    accuracy                           0.84      1061
   macro avg       0.81      0.80      0.81      1061
weighted avg       0.84      0.84      0.84      1061
```

In [1050]:
```
## Performance Matrix on test data set
y_test_predict = LDA_model.predict(X_test)
model_score = LDA_model.score(X_test, y_test)
print(model_score)
print(metrics.confusion_matrix(y_test, y_test_predict))
print(metrics.classification_report(y_test, y_test_predict))
```

```
0.8201754385964912
[[107  46]
 [ 36 267]]
              precision    recall  f1-score   support

           0       0.75      0.70      0.72       153
           1       0.85      0.88      0.87       303

    accuracy                           0.82       456
   macro avg       0.80      0.79      0.79       456
weighted avg       0.82      0.82      0.82       456
```

In [1060]:
```
#the coefficients for each of the independent attributes

for idx, col_name in enumerate(X_train.columns):
    print("The coefficient for {} is {}".format(col_name, LDA_model.coef_[0]
```

```
The coefficient for age is -1.6054817728067365
The coefficient for economic.cond.national_1 is 0.011057388537691917
The coefficient for economic.cond.national_2 is 0.8152787250715633
The coefficient for economic.cond.national_3 is 1.6444282399466414
The coefficient for economic.cond.national_4 is 1.8052982289827066
The coefficient for economic.cond.household_1 is -0.734342349289149
The coefficient for economic.cond.household_2 is -0.22969441775054084
The coefficient for economic.cond.household_3 is -0.08000247435303365
The coefficient for economic.cond.household_4 is -0.9087363797046396
The coefficient for Blair_1 is -0.6861893524294167
The coefficient for Blair_2 is -4.0483397333117293e-16
The coefficient for Blair_3 is 1.2397487082601484
The coefficient for Blair_4 is 2.075975766849963
The coefficient for Hague_1 is -0.44448043481909605
The coefficient for Hague_2 is -0.08488123913742465
The coefficient for Hague_3 is -2.4830826464084983
The coefficient for Hague_4 is -4.206117192482931
The coefficient for Europe_1 is -0.48194809336272615
The coefficient for Europe_2 is -0.555347309457755
The coefficient for Europe_3 is -1.288459692863717
The coefficient for Europe_4 is -0.4855053728879808
The coefficient for Europe_5 is -0.7387708641621507
The coefficient for Europe_6 is -1.25244993906857
The coefficient for Europe_7 is -2.2708058440884917
The coefficient for Europe_8 is -2.7585751578394517
The coefficient for Europe_9 is -2.31197928838158
The coefficient for Europe_10 is -2.1361281333340405
The coefficient for political.knowledge_1 is -0.38235599112698015
The coefficient for political.knowledge_2 is -1.068276354917667
The coefficient for political.knowledge_3 is -1.1207650745790063
The coefficient for gender_1 is 0.19959609113653495
```

In [1055]:
```
# the intercept for the model

intercept = LDA_model.intercept_[0]

print("The intercept for LDA model is {}".format(intercept))
```

```
The intercept for LDA model is 4.241648712932388
```

In [1063]:
```
# R square on testing data (coeff of determinant)
LDA_model.score(X_test, y_test)
```

Out[1063]:
```
0.8201754385964912
```

In [1064]:
```
# R square on training data
LDA_model.score(X_train, y_train)
```

Out[1064]:
```
0.8444863336475024
```

In [1065]:
```
#RMSE on Training data
predicted_train=LDA_model.fit(X_train, y_train).predict(X_train)
np.sqrt(metrics.mean_squared_error(y_train,predicted_train))
```

Out[1065]: 0.39435221104045765

In [1067]:
```
#RMSE on Testing data
predicted_test=LDA_model.fit(X_train, y_train).predict(X_test)
np.sqrt(metrics.mean_squared_error(y_test,predicted_test))
```

Out[1067]: 0.4240572619393621

2. Apply KNN Model and Naïve Bayes Model. Interpret the results.

KNN Model

In [1075]:
```
from sklearn.neighbors import KNeighborsClassifier

KNN_model=KNeighborsClassifier()
KNN_model.fit(X_train,y_train)
```

Out[1075]: KNeighborsClassifier()

Predicting on Training and Test dataset

In [1079]:
```
ytrain_predict = KNN_model.predict(X_train)
ytest_predict = KNN_model.predict(X_test)
```

Getting the Predicted Classes and Probs

In [1080]:
```
ytest_predict_prob=KNN_model.predict_proba(X_test)
pd.DataFrame(ytest_predict_prob).head()
```

Out[1080]:

	0	1
0	0.2	0.8
1	0.0	1.0
2	0.0	1.0
3	1.0	0.0
4	0.2	0.8

KNN Model Evaluation

In [1081]:
```
## Performance Matrix on train data set
y_train_predict = KNN_model.predict(X_train)
model_score = KNN_model.score(X_train, y_train)
print(model_score)
print(metrics.confusion_matrix(y_train, y_train_predict))
print(metrics.classification_report(y_train, y_train_predict))
```

```
0.8501413760603205
[[212  95]
 [ 64 690]]
              precision    recall  f1-score   support

           0       0.77      0.69      0.73       307
           1       0.88      0.92      0.90       754

    accuracy                           0.85      1061
   macro avg       0.82      0.80      0.81      1061
weighted avg       0.85      0.85      0.85      1061
```

In [842]:
```
## Performance Matrix on test data set
y_test_predict = KNN_model.predict(X_test)
model_score = KNN_model.score(X_test, y_test)
print(model_score)
print(metrics.confusion_matrix(y_test, y_test_predict))
print(metrics.classification_report(y_test, y_test_predict))
```

```
0.7828947368421053
[[ 91  62]
 [ 37 266]]
              precision    recall  f1-score   support

           0       0.71      0.59      0.65       153
           1       0.81      0.88      0.84       303

    accuracy                           0.78       456
   macro avg       0.76      0.74      0.75       456
weighted avg       0.78      0.78      0.78       456
```

In [1088]:
```
# R square on testing data (coeff of determinant)
KNN_model.score(X_test, y_test)
```

Out[1088]: 0.7828947368421053

In [1089]:
```
# R square on training data
KNN_model.score(X_train, y_train)
```

Out[1089]: 0.8501413760603205

In [1090]:
```
#RMSE on Training data
predicted_train=KNN_model.fit(X_train, y_train).predict(X_train)
np.sqrt(metrics.mean_squared_error(y_train,predicted_train))
```

Out[1090]:
```
0.38711577588581886
```

In [1091]:
```
#RMSE on Testing data
predicted_test=KNN_model.fit(X_train, y_train).predict(X_test)
np.sqrt(metrics.mean_squared_error(y_test,predicted_test))
```

Out[1091]:
```
0.46594555814804667
```

Naive Bayes Model

In [1082]:
```
from sklearn.naive_bayes import GaussianNB
from sklearn import metrics
```

In [1083]:
```
NB_model = GaussianNB()
NB_model.fit(X_train, y_train)
```

Out[1083]:
```
GaussianNB()
```

Predicting on Training and Test dataset

In [1084]:
```
ytrain_predict = NB_model.predict(X_train)
ytest_predict = NB_model.predict(X_test)
```

Getting the Predicted Classes and Probs

In [1085]:
```
ytest_predict_prob=NB_model.predict_proba(X_test)
pd.DataFrame(ytest_predict_prob).head()
```

Out[1085]:

	0	1
0	9.954622e-01	0.004538
1	8.951301e-01	0.104870
2	9.487741e-32	1.000000
3	9.999868e-01	0.000013
4	1.345917e-09	1.000000

Naives Model Evaluation

In [1097]:
```
## Performance Matrix on train data set
y_train_predict = NB_model.predict(X_train)
model_score = NB_model.score(X_train, y_train)
print(model_score)
print(metrics.confusion_matrix(y_train, y_train_predict))
print(metrics.classification_report(y_train, y_train_predict))
```

```
0.7492931196983977
[[248  59]
 [207 547]]
              precision    recall  f1-score   support

           0       0.55      0.81      0.65       307
           1       0.90      0.73      0.80       754

    accuracy                           0.75      1061
   macro avg       0.72      0.77      0.73      1061
weighted avg       0.80      0.75      0.76      1061
```

In [1098]:
```

y_test.value_counts()
```

Out[1098]:
```
1    303
0    153
Name: vote, dtype: int64
```

In [1099]:
```
## Performance Matrix on test data set
y_test_predict = NB_model.predict(X_test)
model_score = NB_model.score(X_test, y_test)
print(model_score)
print(metrics.confusion_matrix(y_test, y_test_predict))
print(metrics.classification_report(y_test, y_test_predict))
```

```
0.7346491228070176
[[120  33]
 [ 88 215]]
              precision    recall  f1-score   support

           0       0.58      0.78      0.66       153
           1       0.87      0.71      0.78       303

    accuracy                           0.73       456
   macro avg       0.72      0.75      0.72       456
weighted avg       0.77      0.73      0.74       456
```

In [1100]:
```
#R square on testing data (coeff of determinant)
NB_model.score(X_test, y_test)
```

Out[1100]:
```
0.7346491228070176
```

In [1101]:
```
# R square on training data
NB_model.score(X_train, y_train)
```
Out[1101]:
```
0.7492931196983977
```

In [1095]:
```
#RMSE on Training data
predicted_train=NB_model.fit(X_train, y_train).predict(X_train)
np.sqrt(metrics.mean_squared_error(y_train,predicted_train))
```
Out[1095]:
```
0.500706381327023
```

In [1096]:
```
#RMSE on Testing data
predicted_test=NB_model.fit(X_train, y_train).predict(X_test)
np.sqrt(metrics.mean_squared_error(y_test,predicted_test))
```
Out[1096]:
```
0.5151221963699317
```

3. Model Tuning, Bagging (Random Forest should be applied for Bagging) and Boosting.

Ada Boost

In [1102]:
```
from sklearn.ensemble import AdaBoostClassifier

ADB_model = AdaBoostClassifier(n_estimators=100,random_state=1)
ADB_model.fit(X_train,y_train)
```
Out[1102]:
```
AdaBoostClassifier(n_estimators=100, random_state=1)
```

In [1103]:
```
## Performance Matrix on train data set
y_train_predict = ADB_model.predict(X_train)
model_score = ADB_model.score(X_train, y_train)
print(model_score)
print(metrics.confusion_matrix(y_train, y_train_predict))
print(metrics.classification_report(y_train, y_train_predict))
```
```
0.8473138548539114
[[211  96]
 [ 66 688]]
              precision    recall  f1-score   support

           0       0.76      0.69      0.72       307
           1       0.88      0.91      0.89       754

    accuracy                           0.85      1061
   macro avg       0.82      0.80      0.81      1061
weighted avg       0.84      0.85      0.84      1061
```

In [1104]:
```
## Performance Matrix on test data set
y_test_predict = ADB_model.predict(X_test)
model_score = ADB_model.score(X_test, y_test)
print(model_score)
print(metrics.confusion_matrix(y_test, y_test_predict))
print(metrics.classification_report(y_test, y_test_predict))
```

```
0.8135964912280702
[[100  53]
 [ 32 271]]
              precision    recall  f1-score   support

           0       0.76      0.65      0.70       153
           1       0.84      0.89      0.86       303

    accuracy                           0.81       456
   macro avg       0.80      0.77      0.78       456
weighted avg       0.81      0.81      0.81       456
```

In [1105]:
```
#R square on testing data (coeff of determinant)
ADB_model.score(X_test, y_test)
```

Out[1105]: 0.8135964912280702

In [1106]:
```
# R square on training data
ADB_model.score(X_train, y_train)
```

Out[1106]: 0.8473138548539114

In [1107]:
```
#RMSE on Training data
predicted_train=ADB_model.fit(X_train, y_train).predict(X_train)
np.sqrt(metrics.mean_squared_error(y_train,predicted_train))
```

Out[1107]: 0.39075074554770667

In [1108]:
```
#RMSE on Testing data
predicted_test=ADB_model.fit(X_train, y_train).predict(X_test)
np.sqrt(metrics.mean_squared_error(y_test,predicted_test))
```

Out[1108]: 0.43174472639735834

Gradient Boosting

In [1109]:
```
from sklearn.ensemble import GradientBoostingClassifier
gbcl = GradientBoostingClassifier(random_state=1)
gbcl = gbcl.fit(X_train, y_train)
```

In [1110]:
```
## Performance Matrix on train data set
y_train_predict = gbcl.predict(X_train)
model_score = gbcl.score(X_train, y_train)
print(model_score)
print(metrics.confusion_matrix(y_train, y_train_predict))
print(metrics.classification_report(y_train, y_train_predict))
```

```
0.884071630537229
[[227  80]
 [ 43 711]]
              precision    recall  f1-score   support

           0       0.84      0.74      0.79       307
           1       0.90      0.94      0.92       754

    accuracy                           0.88      1061
   macro avg       0.87      0.84      0.85      1061
weighted avg       0.88      0.88      0.88      1061
```

In [1111]:
```
## Performance Matrix on test data set
y_test_predict = gbcl.predict(X_test)
model_score = gbcl.score(X_test, y_test)
print(model_score)
print(metrics.confusion_matrix(y_test, y_test_predict))
print(metrics.classification_report(y_test, y_test_predict))
```

```
0.8223684210526315
[[101  52]
 [ 29 274]]
              precision    recall  f1-score   support

           0       0.78      0.66      0.71       153
           1       0.84      0.90      0.87       303

    accuracy                           0.82       456
   macro avg       0.81      0.78      0.79       456
weighted avg       0.82      0.82      0.82       456
```

In [1114]:
```
#R square on testing data (coeff of determinant)
gbcl.score(X_test, y_test)
```

Out[1114]: 0.8223684210526315

In [1115]:
```
# R square on training data
gbcl.score(X_train, y_train)
```

Out[1115]: 0.884071630537229

In [1116]:
```
#RMSE on Training data
predicted_train=gbcl.fit(X_train, y_train).predict(X_train)
np.sqrt(metrics.mean_squared_error(y_train,predicted_train))
```

Out[1116]: 0.3404825538302528

In [1117]:
```
#RMSE on Testing data
predicted_test=gbcl.fit(X_train, y_train).predict(X_test)
np.sqrt(metrics.mean_squared_error(y_test,predicted_test))
```

Out[1117]:
```
0.4214636152117623
```

Decision Tree

In [1118]:
```
from sklearn import tree


DT_model= tree.DecisionTreeClassifier()
DT_model.fit(X_train, y_train)
```

Out[1118]:
```
DecisionTreeClassifier()
```

In [1119]:
```
## Performance Matrix on train data set
y_train_predict = DT_model.predict(X_train)
model_score = DT_model.score(X_train, y_train)
print(model_score)
print(metrics.confusion_matrix(y_train, y_train_predict))
print(metrics.classification_report(y_train, y_train_predict))
```

```
1.0
[[307   0]
 [  0 754]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       307
           1       1.00      1.00      1.00       754

    accuracy                           1.00      1061
   macro avg       1.00      1.00      1.00      1061
weighted avg       1.00      1.00      1.00      1061
```

In [1120]:
```
## Performance Matrix on test data set
y_test_predict = DT_model.predict(X_test)
model_score = DT_model.score(X_test, y_test)
print(model_score)
print(metrics.confusion_matrix(y_test, y_test_predict))
print(metrics.classification_report(y_test, y_test_predict))
```

```
0.756578947368421
[[ 93  60]
 [ 51 252]]
              precision    recall  f1-score   support

           0       0.65      0.61      0.63       153
           1       0.81      0.83      0.82       303

    accuracy                           0.76       456
   macro avg       0.73      0.72      0.72       456
weighted avg       0.75      0.76      0.75       456
```

In [1121]:
```
#R square on testing data (coeff of determinant)
DT_model.score(X_test, y_test)
```

Out[1121]:
```
0.756578947368421
```

In [1122]:
```
# R square on training data
DT_model.score(X_train, y_train)
```

Out[1122]:
```
1.0
```

In [1123]:
```
#RMSE on Training data
predicted_train=DT_model.fit(X_train, y_train).predict(X_train)
np.sqrt(metrics.mean_squared_error(y_train,predicted_train))
```

Out[1123]:
```
0.0
```

In [1124]:
```
#RMSE on Testing data
predicted_test=DT_model.fit(X_train, y_train).predict(X_test)
np.sqrt(metrics.mean_squared_error(y_test,predicted_test))
```

Out[1124]:
```
0.4933771910329651
```

Random Forest

In [1125]:
```

from sklearn.ensemble import RandomForestClassifier

RF_model=RandomForestClassifier(n_estimators=100,random_state=1)
RF_model.fit(X_train, y_train)
```

Out[1125]:
```
RandomForestClassifier(random_state=1)
```

In [1126]:
```
## Performance Matrix on train data set
y_train_predict = RF_model.predict(X_train)
model_score = RF_model.score(X_train, y_train)
print(model_score)
print(metrics.confusion_matrix(y_train, y_train_predict))
print(metrics.classification_report(y_train, y_train_predict))
```

```
1.0
[[307   0]
 [  0 754]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       307
           1       1.00      1.00      1.00       754

    accuracy                           1.00      1061
   macro avg       1.00      1.00      1.00      1061
weighted avg       1.00      1.00      1.00      1061
```

In [1127]:
```
## Performance Matrix on test data set
y_test_predict = RF_model.predict(X_test)
model_score = RF_model.score(X_test, y_test)
print(model_score)
print(metrics.confusion_matrix(y_test, y_test_predict))
print(metrics.classification_report(y_test, y_test_predict))
```

```
0.8026315789473685
[[ 93  60]
 [ 30 273]]
              precision    recall  f1-score   support

           0       0.76      0.61      0.67       153
           1       0.82      0.90      0.86       303

    accuracy                           0.80       456
   macro avg       0.79      0.75      0.77       456
weighted avg       0.80      0.80      0.80       456
```

In [1128]:
```
#R square on testing data (coeff of determinant)
RF_model.score(X_test, y_test)
```

Out[1128]: 0.8026315789473685

In [1129]:
```
# R square on training data
RF_model.score(X_train, y_train)
```

Out[1129]: 1.0

In [1130]:
```
#RMSE on Training data
predicted_train=RF_model.fit(X_train, y_train).predict(X_train)
np.sqrt(metrics.mean_squared_error(y_train,predicted_train))
```

Out[1130]: 0.0

In [1131]:
```
#RMSE on Testing data
predicted_test=RF_model.fit(X_train, y_train).predict(X_test)
np.sqrt(metrics.mean_squared_error(y_test,predicted_test))
```

Out[1131]: 0.4442616583193193

Bagging

In [1137]:
```
from sklearn.ensemble import BaggingClassifier
from sklearn.ensemble import RandomForestClassifier
cart = RandomForestClassifier()
Bagging_model=BaggingClassifier(base_estimator=cart,n_estimators=100,random_
Bagging_model.fit(X_train, y_train)
```

Out[1137]:
```
BaggingClassifier(base_estimator=RandomForestClassifier(), n_estimators=10
0,
                  random_state=1)
```

In [1138]:
```
## Performance Matrix on train data set
y_train_predict = Bagging_model.predict(X_train)
model_score =Bagging_model.score(X_train, y_train)
print(model_score)
print(metrics.confusion_matrix(y_train, y_train_predict))
print(metrics.classification_report(y_train, y_train_predict))
```

```
0.9679547596606974
[[277  30]
 [  4 750]]
              precision    recall  f1-score   support

           0       0.99      0.90      0.94       307
           1       0.96      0.99      0.98       754

    accuracy                           0.97      1061
   macro avg       0.97      0.95      0.96      1061
weighted avg       0.97      0.97      0.97      1061
```

In [1139]:
```
## Performance Matrix on test data set
y_test_predict = Bagging_model.predict(X_test)
model_score = Bagging_model.score(X_test, y_test)
print(model_score)
print(metrics.confusion_matrix(y_test, y_test_predict))
print(metrics.classification_report(y_test, y_test_predict))
```

```
0.8179824561403509
[[ 97  56]
 [ 27 276]]
              precision    recall  f1-score   support

           0       0.78      0.63      0.70       153
           1       0.83      0.91      0.87       303

    accuracy                           0.82       456
   macro avg       0.81      0.77      0.78       456
weighted avg       0.81      0.82      0.81       456
```

SMOTE

In [1140]:
```
from imblearn.over_sampling import SMOTE
```

SMOTE is only applied on the train data set

In [1141]:
```
smt = SMOTE()
```

In [1143]:
```
sm = SMOTE(random_state=2)
X_train_res, y_train_res = sm.fit_resample(X_train, y_train.ravel())
```

In [1144]:
```
## Let's check the shape after SMOTE
X_train_res.shape
```

Out[1144]:
```
(1508, 31)
```

Naive Bayes with SMOTE

In [1145]:
```
from sklearn.naive_bayes import GaussianNB
from sklearn import metrics
```

In [1146]:
```
NB_SM_model = GaussianNB()
NB_SM_model.fit(X_train_res, y_train_res)
```

Out[1146]:
```
GaussianNB()
```

In [1147]:
```
## Performance Matrix on train data set with SMOTE
y_train_predict = NB_SM_model.predict(X_train_res)
model_score = NB_SM_model.score(X_train_res, y_train_res)
print(model_score)
print(metrics.confusion_matrix(y_train_res, y_train_predict))
print(metrics.classification_report(y_train_res ,y_train_predict))
```

```
0.7798408488063661
[[666  88]
 [244 510]]
              precision    recall  f1-score   support

           0       0.73      0.88      0.80       754
           1       0.85      0.68      0.75       754

    accuracy                           0.78      1508
   macro avg       0.79      0.78      0.78      1508
weighted avg       0.79      0.78      0.78      1508
```

In [1148]:
```
## Performance Matrix on test data set
y_test_predict = NB_SM_model.predict(X_test)
model_score = NB_SM_model.score(X_test, y_test)
print(model_score)
print(metrics.confusion_matrix(y_test, y_test_predict))
print(metrics.classification_report(y_test, y_test_predict))
```

```
0.6951754385964912
[[119  34]
 [105 198]]
              precision    recall  f1-score   support

           0       0.53      0.78      0.63       153
           1       0.85      0.65      0.74       303

    accuracy                           0.70       456
   macro avg       0.69      0.72      0.69       456
weighted avg       0.75      0.70      0.70       456
```

KNN With SMOTE

In [1149]:
```
from sklearn.neighbors import KNeighborsClassifier

KNN_SM_model=KNeighborsClassifier()
KNN_SM_model.fit(X_train_res,y_train_res)
```

Out[1149]:
```
KNeighborsClassifier()
```

In [1150]:
```
## Performance Matrix on train data set
y_train_predict = KNN_SM_model.predict(X_train_res)
model_score = KNN_SM_model.score(X_train_res, y_train_res)
print(model_score)
print(metrics.confusion_matrix(y_train_res, y_train_predict))
print(metrics.classification_report(y_train_res, y_train_predict))
```

```
0.8806366047745358
[[725  29]
 [151 603]]
              precision    recall  f1-score   support

           0       0.83      0.96      0.89       754
           1       0.95      0.80      0.87       754

    accuracy                           0.88      1508
   macro avg       0.89      0.88      0.88      1508
weighted avg       0.89      0.88      0.88      1508
```

In [1151]:
```
## Performance Matrix on test data set
y_test_predict = KNN_SM_model.predict(X_test)
model_score = KNN_SM_model.score(X_test, y_test)
print(model_score)
print(metrics.confusion_matrix(y_test, y_test_predict))
print(metrics.classification_report(y_test, y_test_predict))
```

```
0.743421052631579
[[116  37]
 [ 80 223]]
              precision    recall  f1-score   support

           0       0.59      0.76      0.66       153
           1       0.86      0.74      0.79       303

    accuracy                           0.74       456
   macro avg       0.72      0.75      0.73       456
weighted avg       0.77      0.74      0.75       456
```

Conclusion after SMOTE

Cross Validation on Naive Bayes Model

In [1152]:
```
from sklearn.model_selection import cross_val_score
scores = cross_val_score(NB_SM_model, X_train_res, y_train_res, cv=10)
scores
```

Out[1152]:
```
array([0.73509934, 0.78145695, 0.7615894 , 0.65562914, 0.80794702,
       0.8013245 , 0.76821192, 0.82781457, 0.78      , 0.82      ])
```

In [1153]:
```
scores = cross_val_score(NB_SM_model, X_test, y_test, cv=10)
scores
```

Out[1153]:
```
array([0.56521739, 0.54347826, 0.67391304, 0.56521739, 0.54347826,
       0.47826087, 0.57777778, 0.57777778, 0.6       , 0.53333333])
```

In [ ]:
```
## Cross Validation on KNN Model
```

In [1154]:
```
from sklearn.model_selection import cross_val_score
scores = cross_val_score(KNN_SM_model, X_train_res, y_train_res, cv=10)
scores
```

Out[1154]:
```
array([0.78807947, 0.8410596 , 0.84768212, 0.8013245 , 0.89403974,
       0.8410596 , 0.87417219, 0.85430464, 0.81333333, 0.81333333])
```

In [1155]:
```
scores = cross_val_score(KNN_SM_model, X_test, y_test, cv=10)
scores
```

Out[1155]:
```
array([0.76086957, 0.73913043, 0.73913043, 0.73913043, 0.80434783,
       0.80434783, 0.66666667, 0.75555556, 0.68888889, 0.73333333])
```

In [113]:
```
## After 10 fold cross validation, scores both on train and test data set re
## Hence our model is valid.
```

4. Performance Metrics: Check the performance of Predictions on Train and Test sets using Accuracy, Confusion Matrix, Plot ROC curve and get ROC_AUC score for each model. Final Model: Compare the models and write inference which model is best/optimized.

Logistic Regression

In [1156]:
```
np.round(LR_model.coef_,decimals = 2)>0
```

Out[1156]:
```
array([[False, False,  True,  True,  True, False,  True,  True, False,
        False, False,  True,  True, False, False, False, False,  True,
         True, False,  True, False, False, False, False, False, False,
        False, False, False,  True]])
```

In [1157]:
```
from sklearn.feature_selection import RFE

predictor=X_train
selector = RFE(LR_model, n_features_to_select = 1)
selector = selector.fit(predictor,y_train)
selector.ranking_
```

Out[1157]:
```
array([ 5, 14, 29,  7,  6, 19, 30, 23, 18, 13, 31,  4,  1, 20, 28,  3,  2,
       27, 22, 17, 21, 26, 12, 10,  8,  9, 11, 25, 15, 16, 24])
```

In [1158]:
```
## Performance Matrix on train data set
y_train_predict = LR_model.predict(X_train)
model_score = LR_model.score(X_train, y_train)
print(model_score)
print(metrics.confusion_matrix(y_train, y_train_predict))
print(metrics.classification_report(y_train, y_train_predict))
```

```
0.8473138548539114
[[208  99]
 [ 63 691]]
              precision    recall  f1-score   support

           0       0.77      0.68      0.72       307
           1       0.87      0.92      0.90       754

    accuracy                           0.85      1061
   macro avg       0.82      0.80      0.81      1061
weighted avg       0.84      0.85      0.84      1061
```

In [1159]:
```
## Performance Matrix on test data set
y_test_predict = LR_model.predict(X_test)
model_score = LR_model.score(X_test, y_test)
print(model_score)
print(metrics.confusion_matrix(y_test, y_test_predict))
print(metrics.classification_report(y_test, y_test_predict))
```

```
0.8245614035087719
[[104  49]
 [ 31 272]]
              precision    recall  f1-score   support

           0       0.77      0.68      0.72       153
           1       0.85      0.90      0.87       303

    accuracy                           0.82       456
   macro avg       0.81      0.79      0.80       456
weighted avg       0.82      0.82      0.82       456
```

In [1160]:
```
#the coefficients for each of the independent attributes

for idx, col_name in enumerate(X_train.columns):
    print("The coefficient for {} is {}".format(col_name, LR_model.coef_[0]|
```

```
The coefficient for age is -1.1744662597904778
The coefficient for economic.cond.national_1 is -0.5176819122872669
The coefficient for economic.cond.national_2 is 0.05686377195855065
The coefficient for economic.cond.national_3 is 0.9439766626917371
The coefficient for economic.cond.national_4 is 1.0658858565395501
The coefficient for economic.cond.household_1 is -0.3636165042756915
The coefficient for economic.cond.household_2 is 0.0524779099905074
The coefficient for economic.cond.household_3 is 0.2851715158228671
The coefficient for economic.cond.household_4 is -0.4602130111030272
The coefficient for Blair_1 is -0.7831170915375294
The coefficient for Blair_2 is 0.0
The coefficient for Blair_3 is 0.6346235469331488
The coefficient for Blair_4 is 1.9405714934299025
The coefficient for Hague_1 is -0.4037435130688747
The coefficient for Hague_2 is -0.1127543404691797
The coefficient for Hague_3 is -1.9511780874524054
The coefficient for Hague_4 is -2.910718778649979
The coefficient for Europe_1 is 0.12530103447480292
The coefficient for Europe_2 is 0.21303170378589245
The coefficient for Europe_3 is -0.5400940404325377
The coefficient for Europe_4 is 0.29612321343401515
The coefficient for Europe_5 is -0.14341107825619337
The coefficient for Europe_6 is -0.5982110667160351
The coefficient for Europe_7 is -1.2983305031961327
The coefficient for Europe_8 is -1.4993756060970698
The coefficient for Europe_9 is -1.254106350023709
The coefficient for Europe_10 is -1.2406509400842118
The coefficient for political.knowledge_1 is -0.21622747500023098
The coefficient for political.knowledge_2 is -0.7309469728065002
The coefficient for political.knowledge_3 is -0.6399423403123045
The coefficient for gender_1 is 0.21054384787334637
```

```
*The sign of each coefficient indicates the direction of the relationship
betweeen a predictor variable and the response variable**

- Eg : For every 1 unit increase in Bair_3, vote increases by 0.635

       For every 1 unit increase in Hague_3, vote decreases by 1.951

- Positive sign indicates that as the predictor variable increases the
target variable also increases

- Negative sign indicates that as the predictor variable increases the
target variable also decreases.
```

In [ ]:

In [1161]:
```
# the intercept for the model

intercept = LR_model.intercept_[0]

print("The intercept for LR model is {}".format(intercept))
```

```
The intercept for LR model is 3.3258468981851084
```

In [1162]:
```
# R square on testing data (coeff of determinant)
LR_model.score(X_test, y_test)
```

Out[1162]:
```
0.8245614035087719
```

In [1163]:
```
# R square on training data
LR_model.score(X_train, y_train)
```

Out[1163]:
```
0.8473138548539114
```

In [1170]:
```
# RMSE on Training data
predicted_train=LR_model.fit(X_train, y_train).predict(X_train)
np.sqrt(metrics.mean_squared_error(y_train,predicted_train))
```

Out[1170]:
```
0.39075074554770667
```

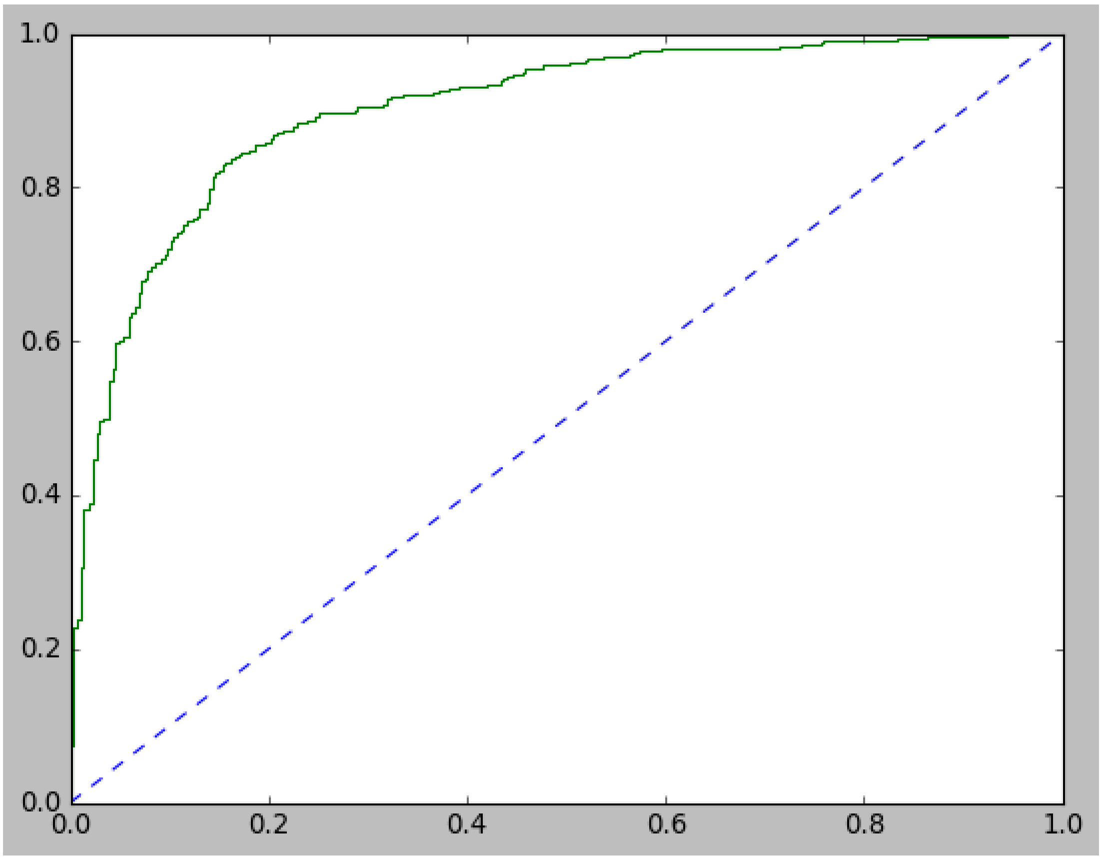
In [1168]:
```
#RMSE on Testing data
predicted_test=LR_model.fit(X_train, y_train).predict(X_test)
np.sqrt(metrics.mean_squared_error(y_test,predicted_test))
```

Out[1168]:
```
0.4188539082916955
```

AUC and ROC for training data

In [1169]:
```
# predict probabilities
probs = LR_model.predict_proba(X_train)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# calculate AUC
auc = roc_auc_score(y_train, probs)
print('AUC: %.3f' % auc)
# calculate roc curve
train_fpr, train_tpr, train_thresholds = roc_curve(y_train, probs)
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
plt.plot(train_fpr, train_tpr);
```

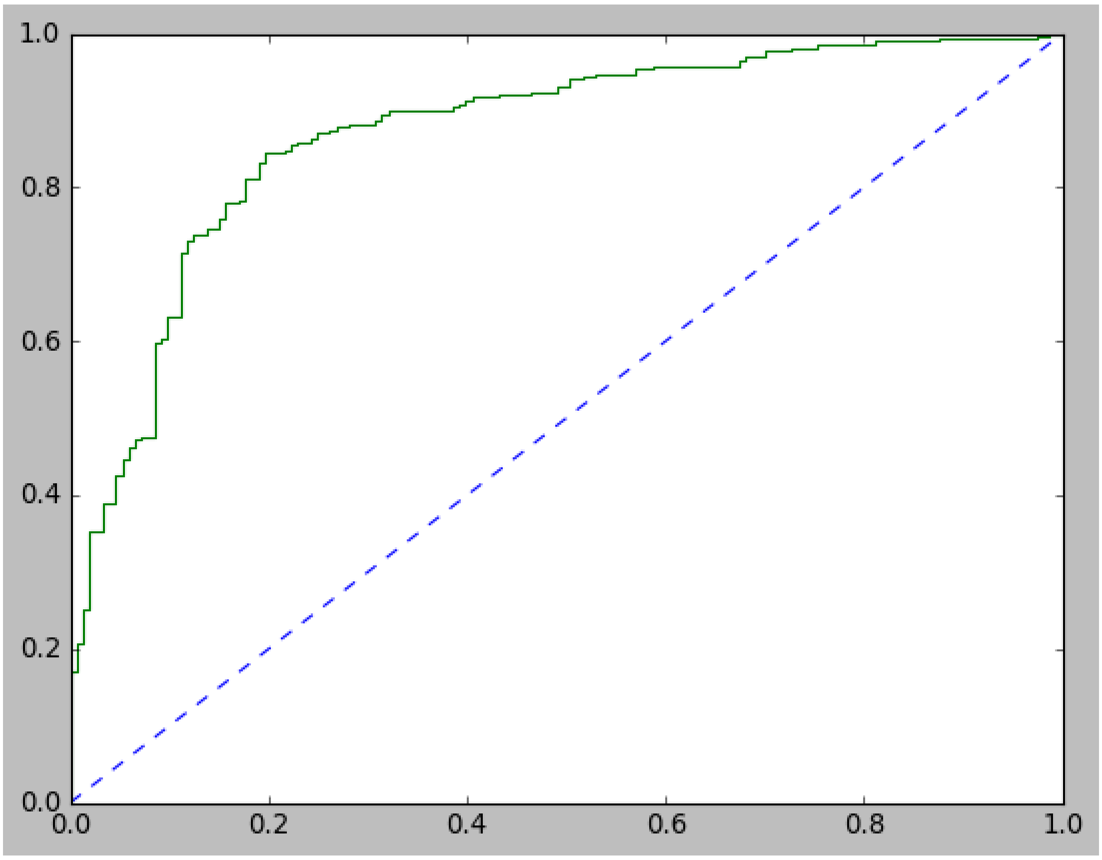
```
AUC: 0.903
```



AUC and ROC for Test data set

In [1171]:
```
# predict probabilities
probs = LR_model.predict_proba(X_test)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# calculate AUC
test_auc = roc_auc_score(y_test, probs)
print('AUC: %.3f' % auc)
# calculate roc curve
test_fpr, test_tpr, test_thresholds = roc_curve(y_test, probs)
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
plt.plot(test_fpr, test_tpr);
```

```
AUC: 0.903
```



Linear Discriminant Analysis

In [1173]:
```
np.round(LDA_model.coef_,decimals = 2)>0
```

Out[1173]:
```
array([[False,  True,  True,  True,  True, False, False, False, False,
        False, False,  True,  True, False, False, False, False, False,
        False, False, False, False, False, False, False, False, False,
        False, False, False,  True]])
```

In [1175]:
```
from sklearn.feature_selection import RFE

predictor=X_train
selector = RFE(LDA_model, n_features_to_select = 1)
selector = selector.fit(predictor,y_train)
selector.ranking_
```

Out[1175]:
```
array([ 9, 30, 14, 11, 10, 19, 27, 29, 18, 15, 31,  4,  3, 24, 28,  2,  1,
       23, 21, 17, 22, 20, 16,  8,  5,  6,  7, 25, 12, 13, 26])
```

In [1177]:
```
## Performance Matrix on train data set
y_train_predict = LDA_model.predict(X_train)
model_score = LDA_model.score(X_train, y_train)
print(model_score)
print(metrics.confusion_matrix(y_train, y_train_predict))
print(metrics.classification_report(y_train, y_train_predict))
```

```
0.8444863336475024
[[216  91]
 [ 74 680]]
              precision    recall  f1-score   support

           0       0.74      0.70      0.72       307
           1       0.88      0.90      0.89       754

    accuracy                           0.84      1061
   macro avg       0.81      0.80      0.81      1061
weighted avg       0.84      0.84      0.84      1061
```

In [1178]:
```
## Performance Matrix on test data set
y_test_predict = LDA_model.predict(X_test)
model_score = LDA_model.score(X_test, y_test)
print(model_score)
print(metrics.confusion_matrix(y_test, y_test_predict))
print(metrics.classification_report(y_test, y_test_predict))
```

```
0.8201754385964912
[[107  46]
 [ 36 267]]
              precision    recall  f1-score   support

           0       0.75      0.70      0.72       153
           1       0.85      0.88      0.87       303

    accuracy                           0.82       456
   macro avg       0.80      0.79      0.79       456
weighted avg       0.82      0.82      0.82       456
```

In [1179]:
```
#the coefficients for each of the independent attributes

for idx, col_name in enumerate(X_train.columns):
    print("The coefficient for {} is {}".format(col_name, LDA_model.coef_[0]
```

```
The coefficient for age is -1.6054817728067365
The coefficient for economic.cond.national_1 is 0.011057388537691917
The coefficient for economic.cond.national_2 is 0.8152787250715633
The coefficient for economic.cond.national_3 is 1.6444282399466414
The coefficient for economic.cond.national_4 is 1.8052982289827066
The coefficient for economic.cond.household_1 is -0.734342349289149
The coefficient for economic.cond.household_2 is -0.22969441775054084
The coefficient for economic.cond.household_3 is -0.08000247435303365
The coefficient for economic.cond.household_4 is -0.9087363797046396
The coefficient for Blair_1 is -0.6861893524294167
The coefficient for Blair_2 is -4.0483397333117293e-16
The coefficient for Blair_3 is 1.2397487082601484
The coefficient for Blair_4 is 2.075975766849963
The coefficient for Hague_1 is -0.44448043481909605
The coefficient for Hague_2 is -0.08488123913742465
The coefficient for Hague_3 is -2.4830826464084983
The coefficient for Hague_4 is -4.206117192482931
The coefficient for Europe_1 is -0.48194809336272615
The coefficient for Europe_2 is -0.555347309457755
The coefficient for Europe_3 is -1.288459692863717
The coefficient for Europe_4 is -0.4855053728879808
The coefficient for Europe_5 is -0.7387708641621507
The coefficient for Europe_6 is -1.25244993906857
The coefficient for Europe_7 is -2.2708058440884917
The coefficient for Europe_8 is -2.7585751578394517
The coefficient for Europe_9 is -2.31197928838158
The coefficient for Europe_10 is -2.1361281333340405
The coefficient for political.knowledge_1 is -0.38235599112698015
The coefficient for political.knowledge_2 is -1.068276354917667
The coefficient for political.knowledge_3 is -1.1207650745790063
The coefficient for gender_1 is 0.19959609113653495
```

In [1180]:
```
# the intercept for the model

intercept = LDA_model.intercept_[0]

print("The intercept for LR model is {}".format(intercept))
```

```
The intercept for LR model is 4.241648712932388
```

In [1181]:
```
# R square on testing data (coeff of determinant)
LDA_model.score(X_test, y_test)
```

Out[1181]: 0.8201754385964912

In [1182]:
```
# R square on training data
LDA_model.score(X_train, y_train)
```

Out[1182]: 0.8444863336475024

In [1183]:
```
# RMSE on Training data
predicted_train=LDA_model.fit(X_train, y_train).predict(X_train)
np.sqrt(metrics.mean_squared_error(y_train,predicted_train))
```

Out[1183]: 0.39435221104045765

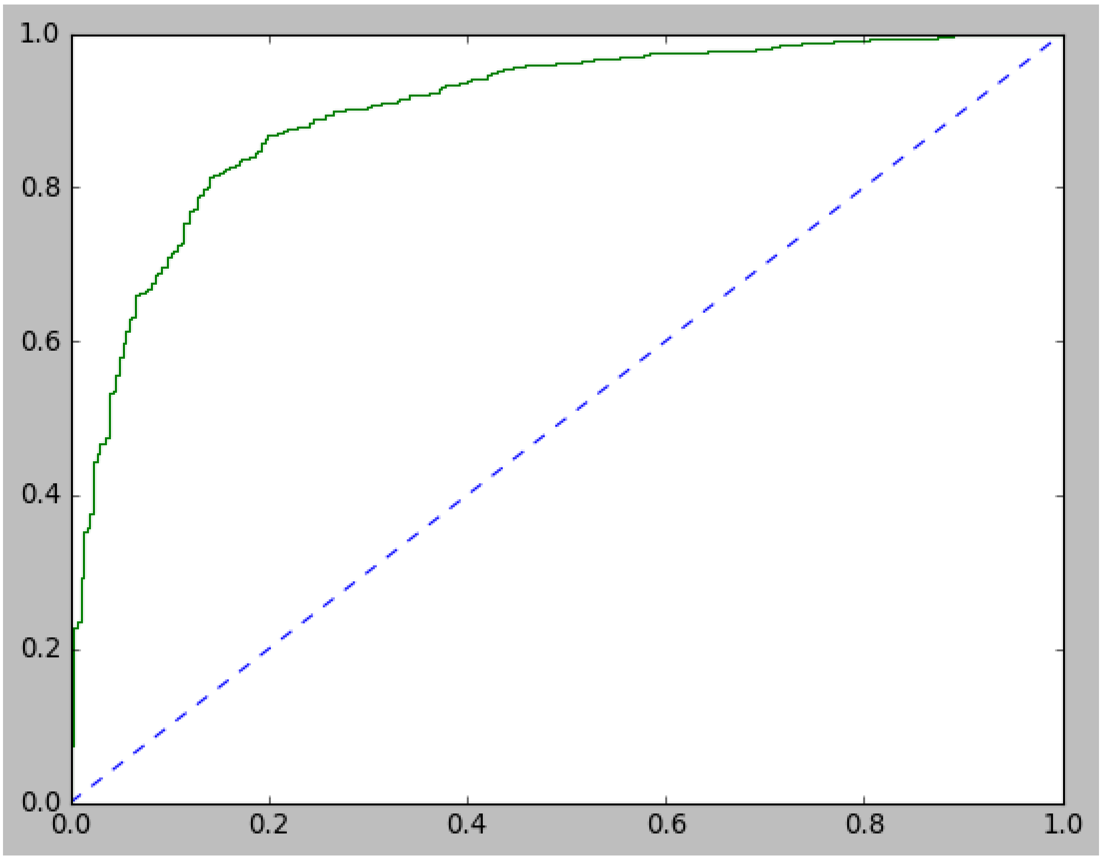
In [1184]:
```
#RMSE on Testing data
predicted_test=LDA_model.fit(X_train, y_train).predict(X_test)
np.sqrt(metrics.mean_squared_error(y_test,predicted_test))
```

Out[1184]: 0.4240572619393621

AUC and ROC for Training Data

In [1185]:
```
# predict probabilities
probs = LDA_model.predict_proba(X_train)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# calculate AUC
auc = roc_auc_score(y_train, probs)
print('AUC: %.3f' % auc)
# calculate roc curve
train_fpr, train_tpr, train_thresholds = roc_curve(y_train, probs)
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
plt.plot(train_fpr, train_tpr);
```

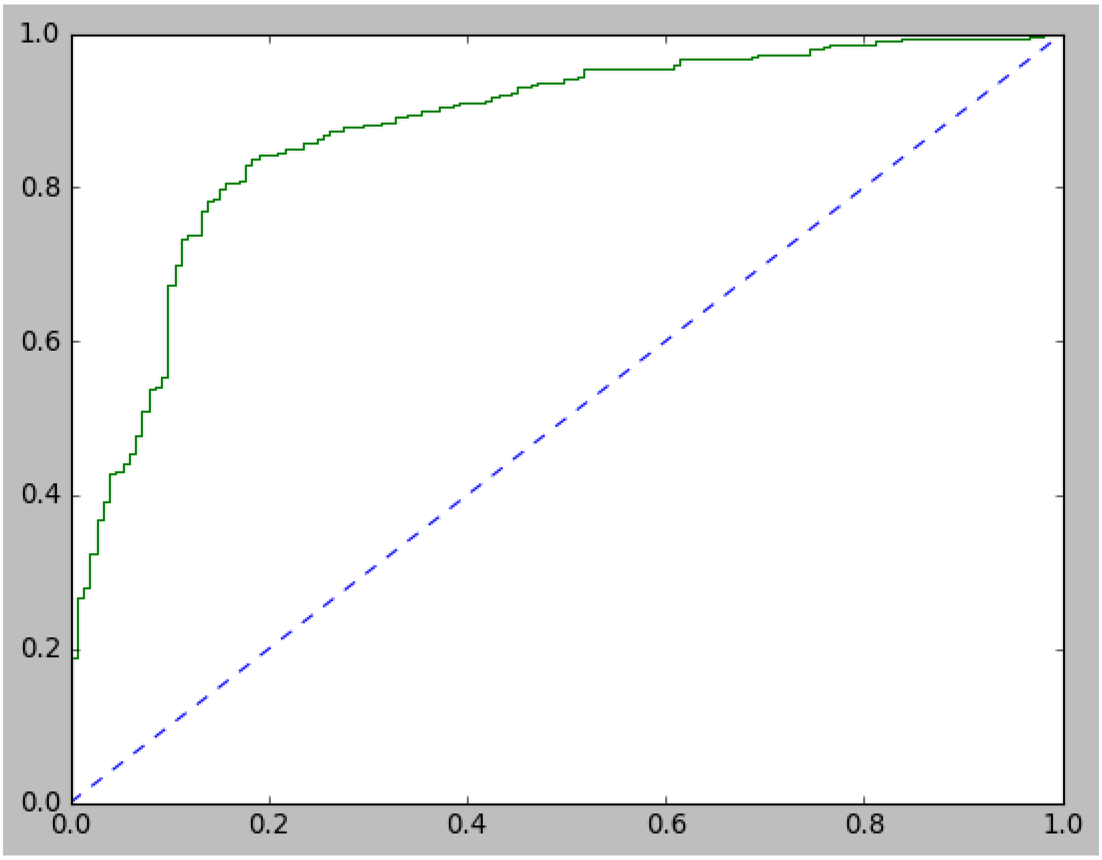
AUC: 0.902



AUC and ROC for Test Data

In [1186]:
```
# predict probabilities
probs = LDA_model.predict_proba(X_test)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# calculate AUC
test_auc = roc_auc_score(y_test, probs)
print('AUC: %.3f' % auc)
# calculate roc curve
test_fpr, test_tpr, test_thresholds = roc_curve(y_test, probs)
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
plt.plot(test_fpr, test_tpr);
```

```
AUC: 0.902
```



KNN Model

In [1188]:
```
## Performance Matrix on train data set
y_train_predict = KNN_model.predict(X_train)
model_score = KNN_model.score(X_train, y_train)
print(model_score)
print(metrics.confusion_matrix(y_train, y_train_predict))
print(metrics.classification_report(y_train, y_train_predict))
```

```
0.8501413760603205
[[212  95]
 [ 64 690]]
              precision    recall  f1-score   support

           0       0.77      0.69      0.73       307
           1       0.88      0.92      0.90       754

    accuracy                           0.85      1061
   macro avg       0.82      0.80      0.81      1061
weighted avg       0.85      0.85      0.85      1061
```

In [1189]:
```
## Performance Matrix on test data set
y_test_predict = KNN_model.predict(X_test)
model_score = KNN_model.score(X_test, y_test)
print(model_score)
print(metrics.confusion_matrix(y_test, y_test_predict))
print(metrics.classification_report(y_test, y_test_predict))
```

```
0.7828947368421053
[[ 91  62]
 [ 37 266]]
              precision    recall  f1-score   support

           0       0.71      0.59      0.65       153
           1       0.81      0.88      0.84       303

    accuracy                           0.78       456
   macro avg       0.76      0.74      0.75       456
weighted avg       0.78      0.78      0.78       456
```

In [1191]:
```
# R square on testing data (coeff of determinant)
KNN_model.score(X_test, y_test)
```

Out[1191]: 0.7828947368421053

In [1192]:
```
# R square on training data
KNN_model.score(X_train, y_train)
```

Out[1192]: 0.8501413760603205

In [1193]:
```
# RMSE on Training data
predicted_train=KNN_model.fit(X_train, y_train).predict(X_train)
np.sqrt(metrics.mean_squared_error(y_train,predicted_train))
```

Out[1193]: 0.38711577588581886

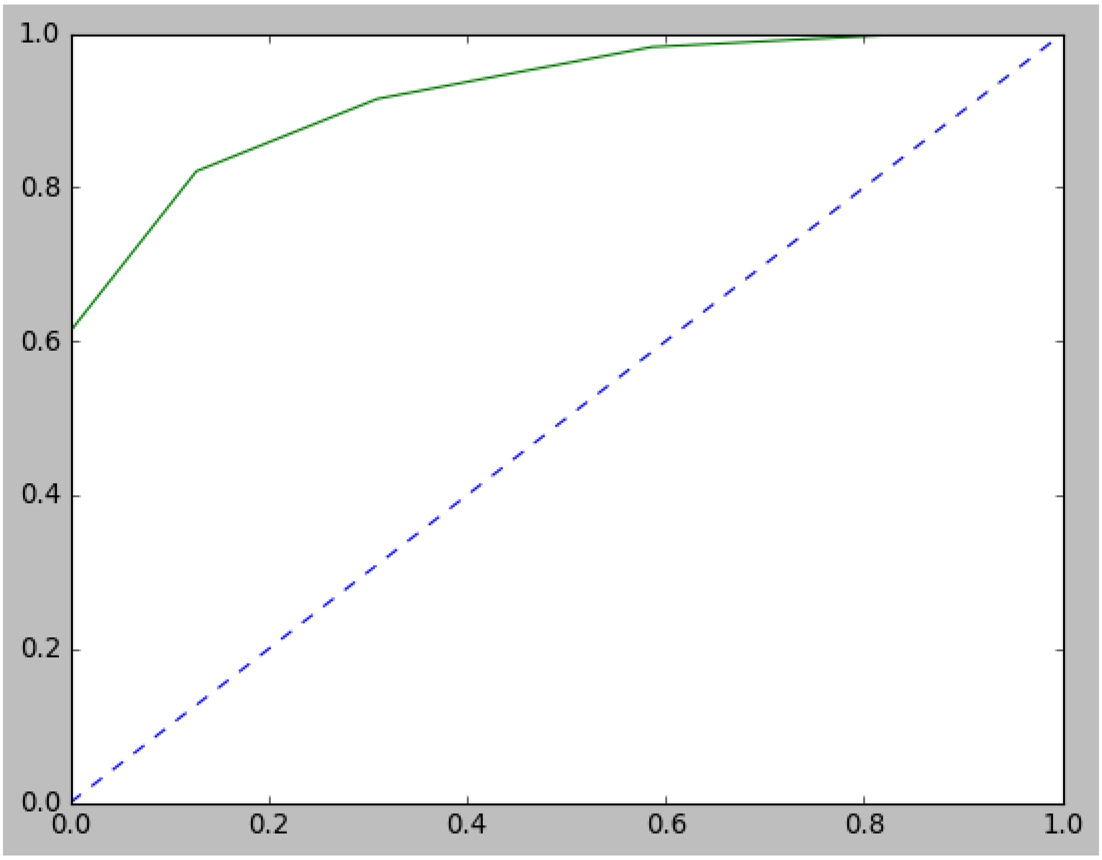
In [1194]:
```
#RMSE on Testing data
predicted_test=KNN_model.fit(X_train, y_train).predict(X_test)
np.sqrt(metrics.mean_squared_error(y_test,predicted_test))
```

Out[1194]: 0.46594555814804667

AUC and ROC on the training data

In [1190]:
```
# predict probabilities
probs =KNN_model.predict_proba(X_train)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# calculate AUC
auc = roc_auc_score(y_train, probs)
print('AUC: %.3f' % auc)
# calculate roc curve
train_fpr, train_tpr, train_thresholds = roc_curve(y_train, probs)
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
plt.plot(train_fpr, train_tpr);
```

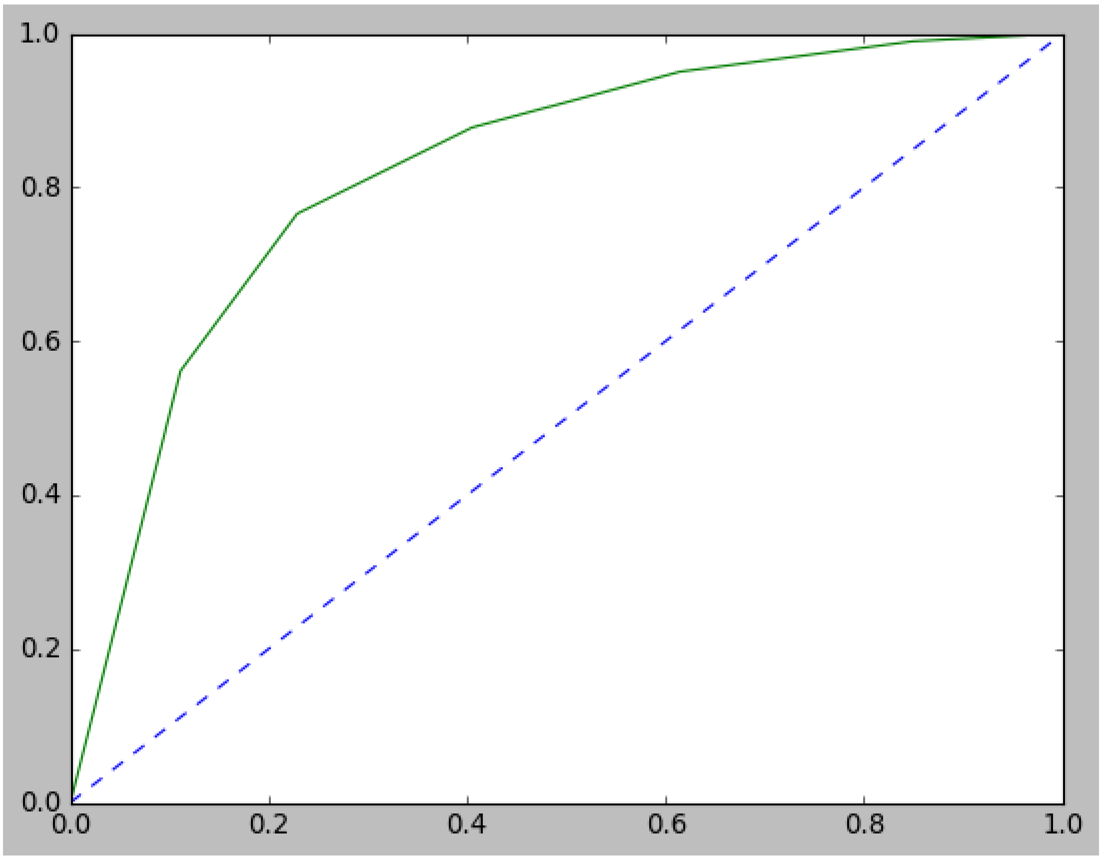
AUC: 0.924



AUC and ROC for Test data

In [1196]:
```
# predict probabilities
probs = KNN_model.predict_proba(X_test)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# calculate AUC
test_auc = roc_auc_score(y_test, probs)
print('AUC: %.3f' % auc)
# calculate roc curve
test_fpr, test_tpr, test_thresholds = roc_curve(y_test, probs)
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
plt.plot(test_fpr, test_tpr);
```

```
AUC: 0.924
```



Naives Bayes Model

In [1198]:
```
## Performance Matrix on train data set
y_train_predict = NB_model.predict(X_train)
model_score = NB_model.score(X_train, y_train)
print(model_score)
print(metrics.confusion_matrix(y_train, y_train_predict))
print(metrics.classification_report(y_train, y_train_predict))
```

```
0.7492931196983977
[[248  59]
 [207 547]]
              precision    recall  f1-score   support

           0       0.55      0.81      0.65       307
           1       0.90      0.73      0.80       754

    accuracy                           0.75      1061
   macro avg       0.72      0.77      0.73      1061
weighted avg       0.80      0.75      0.76      1061
```

In [1199]:
```
## Performance Matrix on test data set
y_test_predict = NB_model.predict(X_test)
model_score = NB_model.score(X_test, y_test)
print(model_score)
print(metrics.confusion_matrix(y_test, y_test_predict))
print(metrics.classification_report(y_test, y_test_predict))
```

```
0.7346491228070176
[[120  33]
 [ 88 215]]
              precision    recall  f1-score   support

           0       0.58      0.78      0.66       153
           1       0.87      0.71      0.78       303

    accuracy                           0.73       456
   macro avg       0.72      0.75      0.72       456
weighted avg       0.77      0.73      0.74       456
```

In [1200]:
```
# R square on testing data (coeff of determinant)
NB_model.score(X_test, y_test)
```

Out[1200]: 0.7346491228070176

In [1201]:
```
# R square on training data
NB_model.score(X_train, y_train)
```

Out[1201]: 0.7492931196983977

In [1202]:
```
# RMSE on Training data
predicted_train=NB_model.fit(X_train, y_train).predict(X_train)
np.sqrt(metrics.mean_squared_error(y_train,predicted_train))
```

Out[1202]: 0.500706381327023

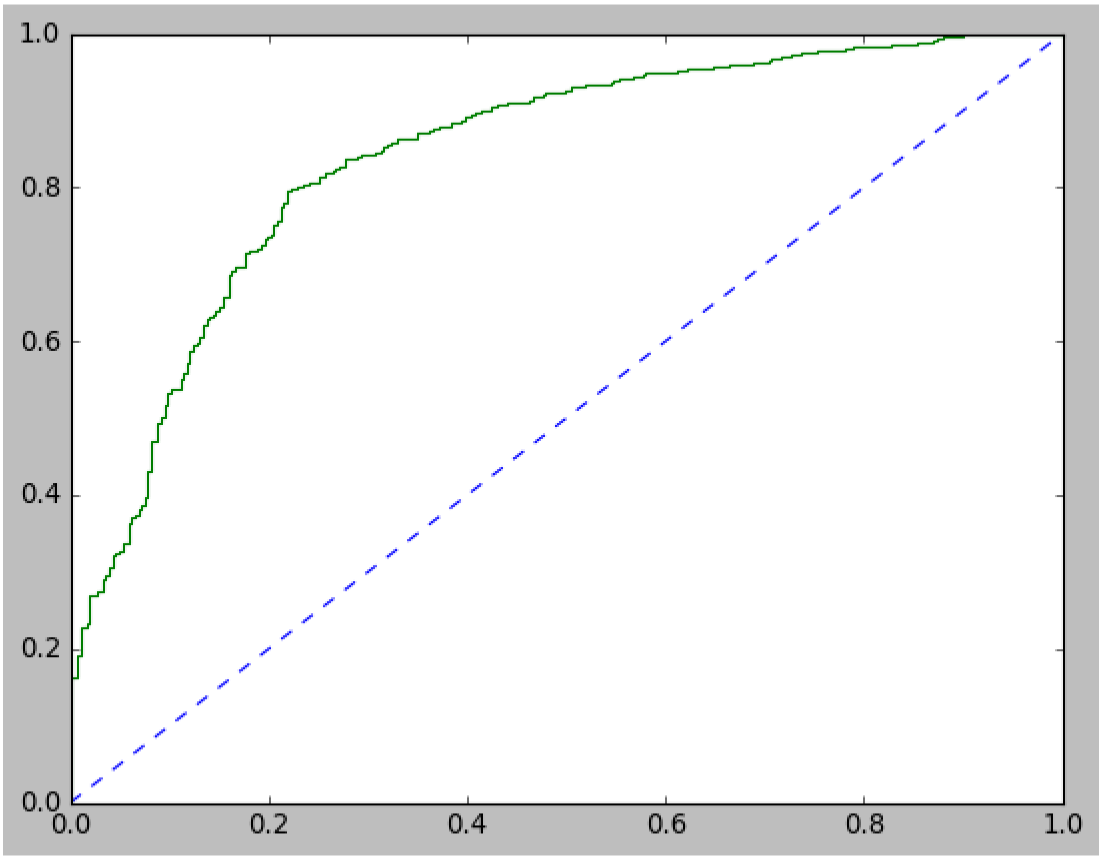
In [1203]:
```
#RMSE on Testing data
predicted_test=NB_model.fit(X_train, y_train).predict(X_test)
np.sqrt(metrics.mean_squared_error(y_test,predicted_test))
```

Out[1203]: 0.5151221963699317

AUC and ROC for Training Data

In [1204]:
```
# predict probabilities
probs =NB_model.predict_proba(X_train)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# calculate AUC
auc = roc_auc_score(y_train, probs)
print('AUC: %.3f' % auc)
# calculate roc curve
train_fpr, train_tpr, train_thresholds = roc_curve(y_train, probs)
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
plt.plot(train_fpr, train_tpr);
```

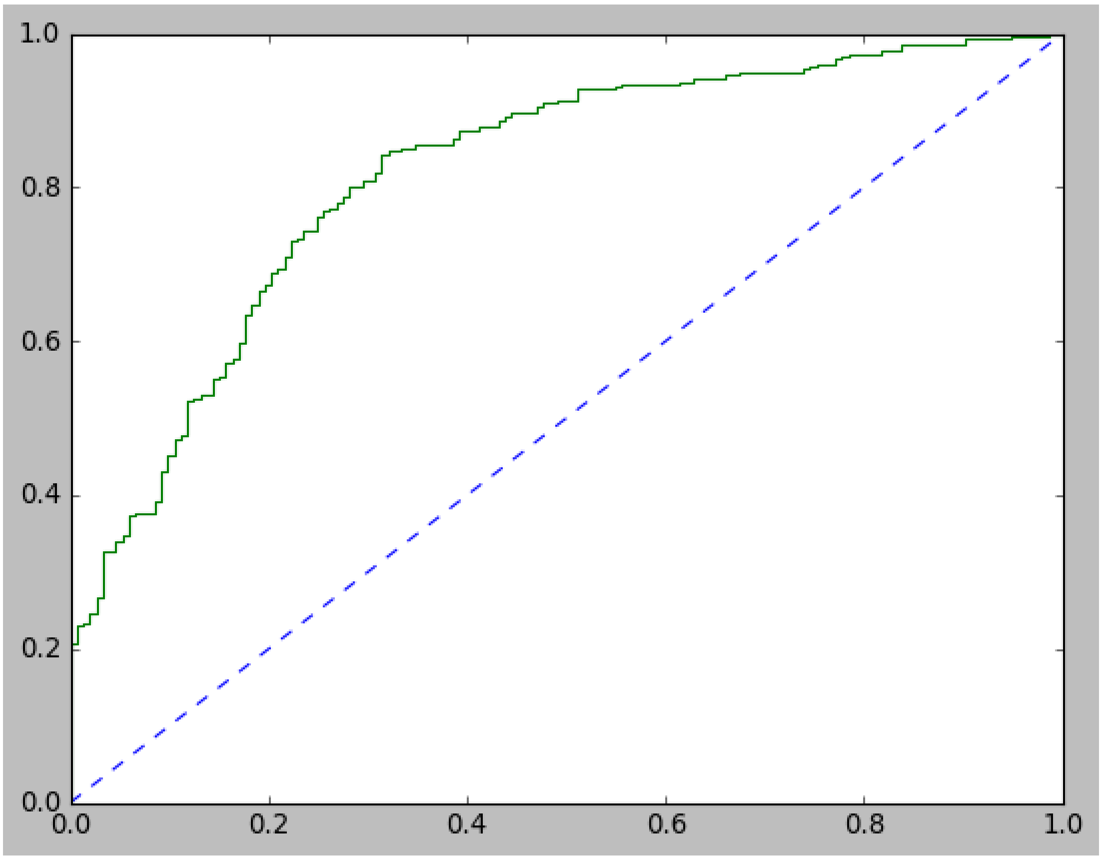
AUC: 0.843



AUC and ROC for test data

In [1205]:
```
# predict probabilities
probs = NB_model.predict_proba(X_test)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# calculate AUC
test_auc = roc_auc_score(y_test, probs)
print('AUC: %.3f' % auc)
# calculate roc curve
test_fpr, test_tpr, test_thresholds = roc_curve(y_test, probs)
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
plt.plot(test_fpr, test_tpr);
```

```
AUC: 0.843
```



Ada Boost

In [1207]:
```
## Performance Matrix on train data set
y_train_predict = ADB_model.predict(X_train)
model_score = ADB_model.score(X_train, y_train)
print(model_score)
print(metrics.confusion_matrix(y_train, y_train_predict))
print(metrics.classification_report(y_train, y_train_predict))
```

```
0.8473138548539114
[[211  96]
 [ 66 688]]
              precision    recall  f1-score   support

           0       0.76      0.69      0.72       307
           1       0.88      0.91      0.89       754

    accuracy                           0.85      1061
   macro avg       0.82      0.80      0.81      1061
weighted avg       0.84      0.85      0.84      1061
```

In [1214]:
```
## Performance Matrix on test data set
y_test_predict = ADB_model.predict(X_test)
model_score = ADB_model.score(X_test, y_test)
print(model_score)
print(metrics.confusion_matrix(y_test, y_test_predict))
print(metrics.classification_report(y_test, y_test_predict))
```

```
0.8135964912280702
[[100  53]
 [ 32 271]]
              precision    recall  f1-score   support

           0       0.76      0.65      0.70       153
           1       0.84      0.89      0.86       303

    accuracy                           0.81       456
   macro avg       0.80      0.77      0.78       456
weighted avg       0.81      0.81      0.81       456
```

In [1228]:
```
# R square on testing data (coeff of determinant)
ADB_model.score(X_test, y_test)
```

Out[1228]: 0.8135964912280702

In [1236]:
```
# R square on training data
ADB_model.score(X_train, y_train)
```

Out[1236]: 0.8473138548539114

In [1244]:
```
# RMSE on Training data
predicted_train=ADB_model.fit(X_train, y_train).predict(X_train)
np.sqrt(metrics.mean_squared_error(y_train,predicted_train))
```

Out[1244]: 0.39075074554770667

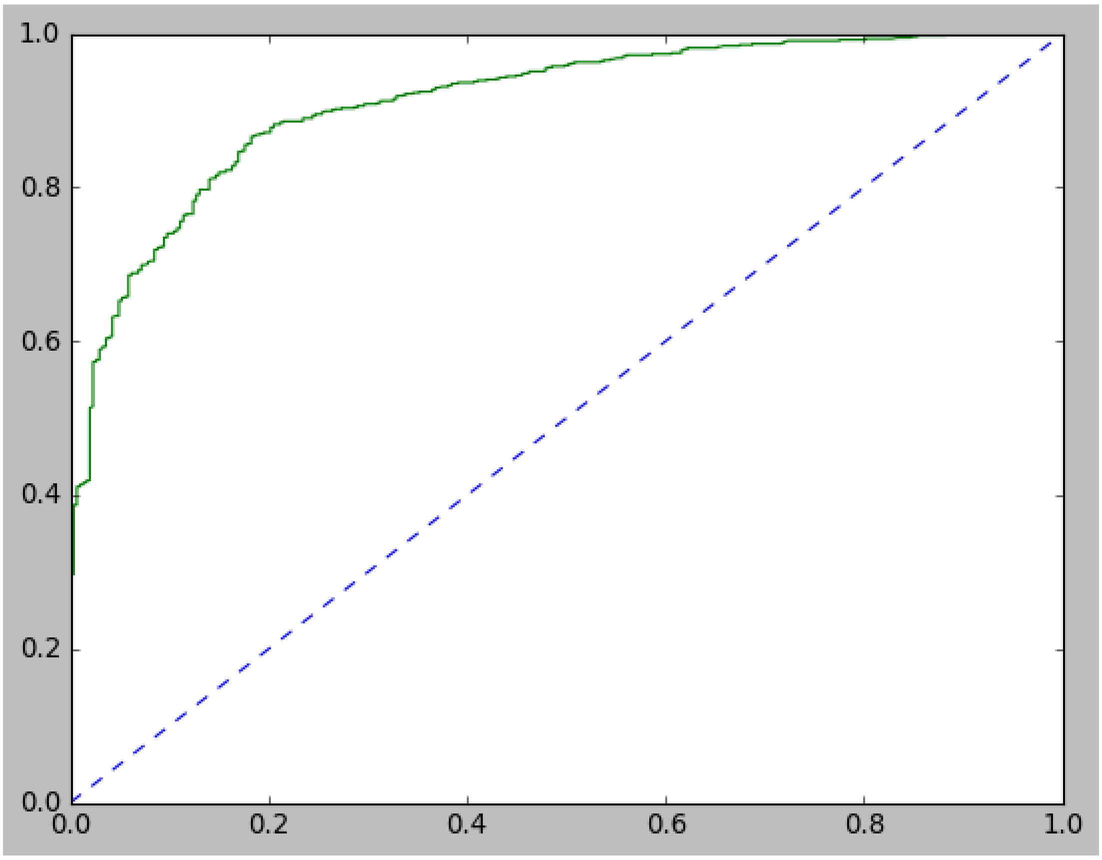
In [1252]:
```
#RMSE on Testing data
predicted_test=ADB_model.fit(X_train, y_train).predict(X_test)
np.sqrt(metrics.mean_squared_error(y_test,predicted_test))
```

Out[1252]: 0.43174472639735834

AUC and ROC for training data

In [1259]:
```
# predict probabilities
probs =ADB_model.predict_proba(X_train)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# calculate AUC
auc = roc_auc_score(y_train, probs)
print('AUC: %.3f' % auc)
# calculate roc curve
train_fpr, train_tpr, train_thresholds = roc_curve(y_train, probs)
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
plt.plot(train_fpr, train_tpr);
```

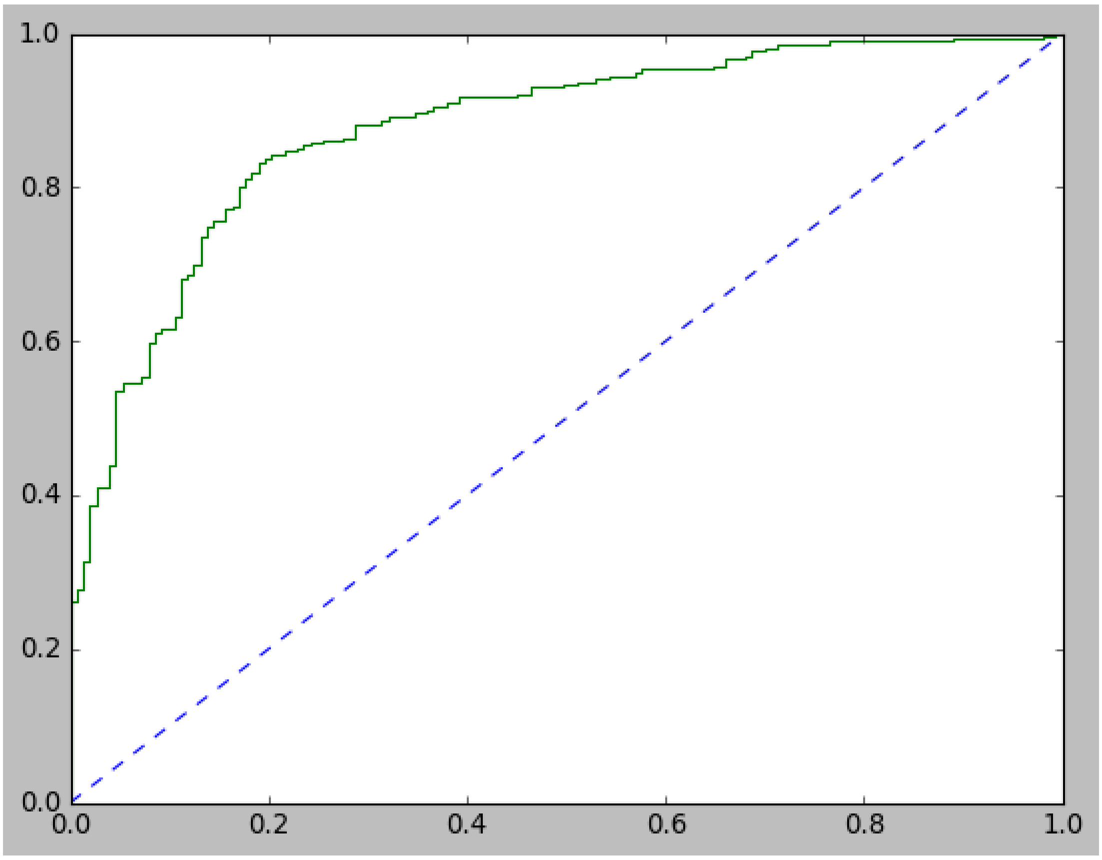
AUC: 0.912



AUC and ROC for test data

In [1266]:
```
# predict probabilities
probs = ADB_model.predict_proba(X_test)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# calculate AUC
test_auc = roc_auc_score(y_test, probs)
print('AUC: %.3f' % auc)
# calculate roc curve
test_fpr, test_tpr, test_thresholds = roc_curve(y_test, probs)
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
plt.plot(test_fpr, test_tpr);
```

```
AUC: 0.912
```



Gradient Boost

In [1208]:
```
## Performance Matrix on train data set
y_train_predict = gbcl.predict(X_train)
model_score = gbcl.score(X_train, y_train)
print(model_score)
print(metrics.confusion_matrix(y_train, y_train_predict))
print(metrics.classification_report(y_train, y_train_predict))
```

```
0.884071630537229
[[227  80]
 [ 43 711]]
              precision    recall  f1-score   support

           0       0.84      0.74      0.79       307
           1       0.90      0.94      0.92       754

    accuracy                           0.88      1061
   macro avg       0.87      0.84      0.85      1061
weighted avg       0.88      0.88      0.88      1061
```

In [1215]:
```
## Performance Matrix on test data set
y_test_predict = gbcl.predict(X_test)
model_score = gbcl.score(X_test, y_test)
print(model_score)
print(metrics.confusion_matrix(y_test, y_test_predict))
print(metrics.classification_report(y_test, y_test_predict))
```

```
0.8223684210526315
[[101  52]
 [ 29 274]]
              precision    recall  f1-score   support

           0       0.78      0.66      0.71       153
           1       0.84      0.90      0.87       303

    accuracy                           0.82       456
   macro avg       0.81      0.78      0.79       456
weighted avg       0.82      0.82      0.82       456
```

In [1227]:
```
# R square on testing data (coeff of determinant)
gbcl.score(X_test, y_test)
```

Out[1227]: 0.8223684210526315

In [1235]:
```
# R square on training data
gbcl.score(X_train, y_train)
```

Out[1235]: 0.884071630537229

In [1242]:
```
# RMSE on Training data
predicted_train=gbcl.fit(X_train, y_train).predict(X_train)
np.sqrt(metrics.mean_squared_error(y_train,predicted_train))
```

Out[1242]: 0.3404825538302528

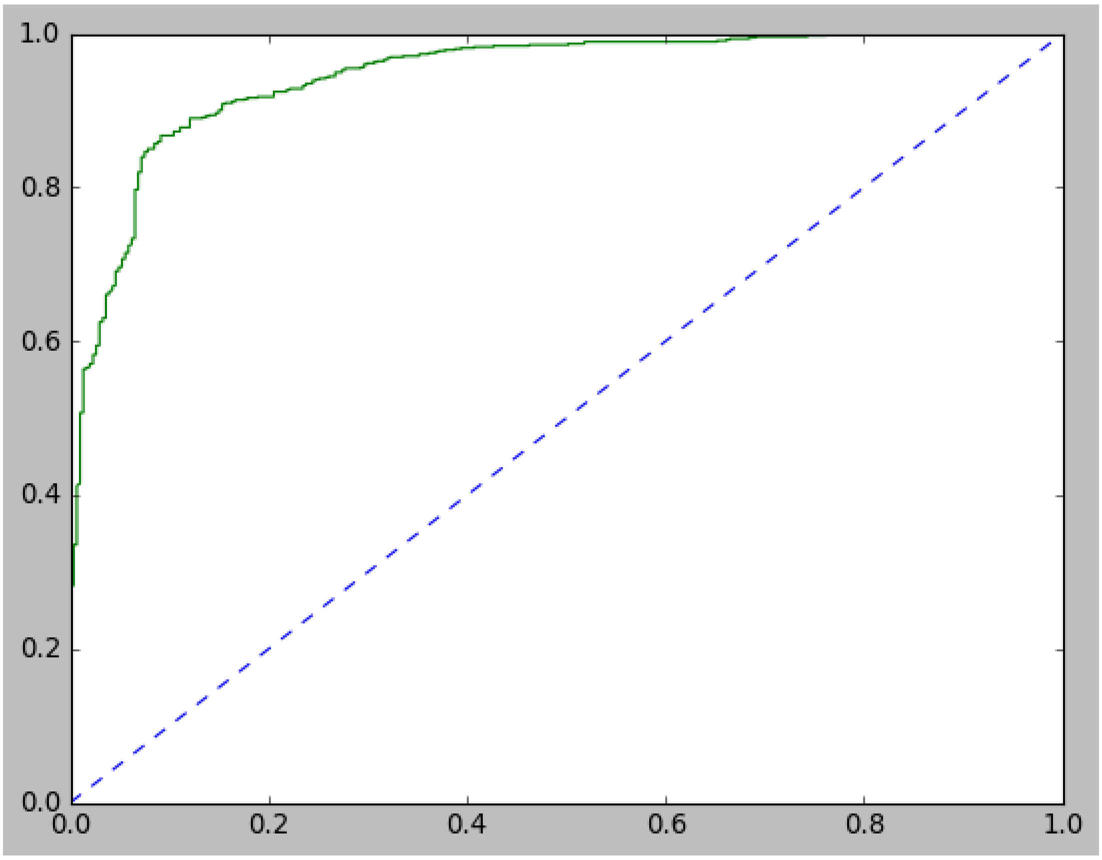
In [1250]:
```
#RMSE on Testing data
predicted_test=gbcl.fit(X_train, y_train).predict(X_test)
np.sqrt(metrics.mean_squared_error(y_test,predicted_test))
```

Out[1250]:
```
0.4214636152117623
```

AUC and ROC for training data

In [1258]:
```
# predict probabilities
probs =gbcl.predict_proba(X_train)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# calculate AUC
auc = roc_auc_score(y_train, probs)
print('AUC: %.3f' % auc)
# calculate roc curve
train_fpr, train_tpr, train_thresholds = roc_curve(y_train, probs)
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
plt.plot(train_fpr, train_tpr);
```

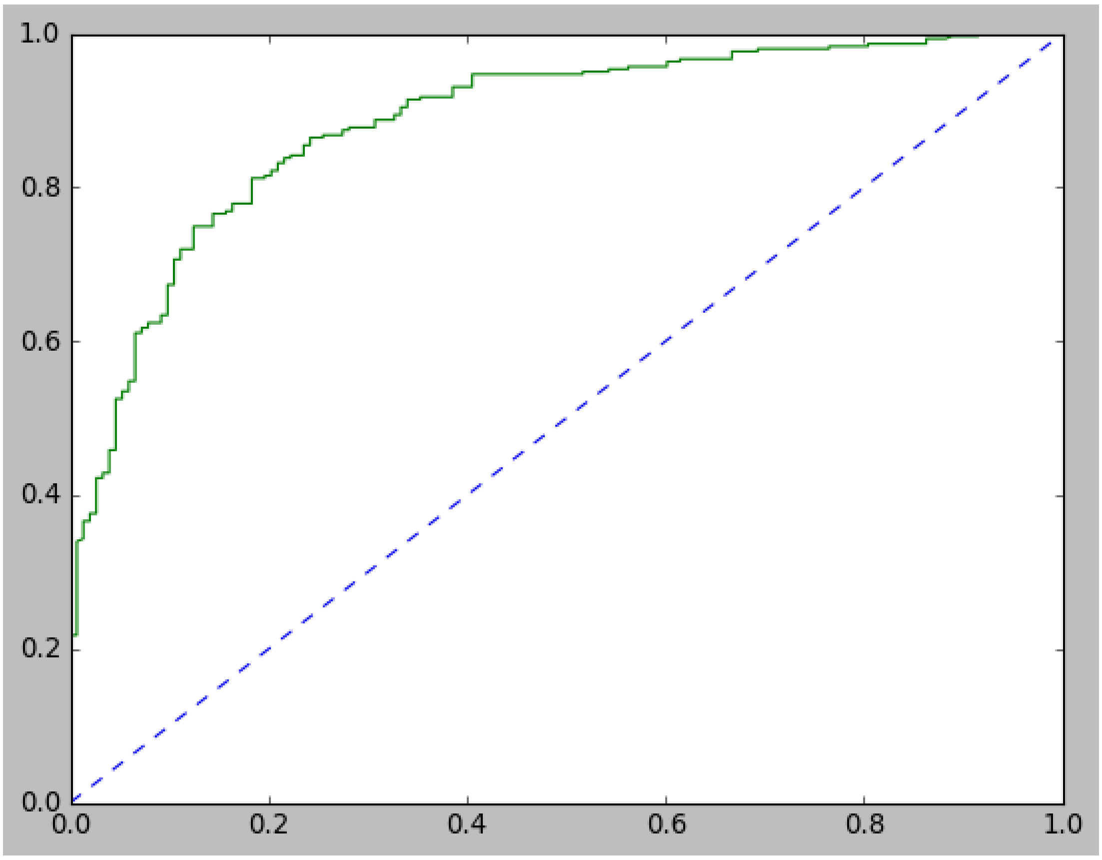
```
AUC: 0.945
```



AUC and ROC for test data

In [1265]:
```
# predict probabilities
probs = gbcl.predict_proba(X_test)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# calculate AUC
test_auc = roc_auc_score(y_test, probs)
print('AUC: %.3f' % auc)
# calculate roc curve
test_fpr, test_tpr, test_thresholds = roc_curve(y_test, probs)
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
plt.plot(test_fpr, test_tpr);
```

```
AUC: 0.912
```



Decision Tree

In [ ]:

In [ ]:

In [1209]:
```
## Performance Matrix on train data set
y_train_predict = DT_model.predict(X_train)
model_score = DT_model.score(X_train, y_train)
print(model_score)
print(metrics.confusion_matrix(y_train, y_train_predict))
print(metrics.classification_report(y_train, y_train_predict))
```

```
1.0
[[307   0]
 [  0 754]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       307
           1       1.00      1.00      1.00       754

    accuracy                           1.00      1061
   macro avg       1.00      1.00      1.00      1061
weighted avg       1.00      1.00      1.00      1061
```

In [1216]:
```
## Performance Matrix on test data set
y_test_predict = DT_model.predict(X_test)
model_score = DT_model.score(X_test, y_test)
print(model_score)
print(metrics.confusion_matrix(y_test, y_test_predict))
print(metrics.classification_report(y_test, y_test_predict))
```

```
0.756578947368421
[[ 94  59]
 [ 52 251]]
              precision    recall  f1-score   support

           0       0.64      0.61      0.63       153
           1       0.81      0.83      0.82       303

    accuracy                           0.76       456
   macro avg       0.73      0.72      0.72       456
weighted avg       0.75      0.76      0.76       456
```

In [1226]:
```
# R square on testing data (coeff of determinant)
DT_model.score(X_test, y_test)
```

Out[1226]: 0.756578947368421

In [1233]:
```
# R square on training data
DT_model.score(X_train, y_train)
```

Out[1233]: 1.0

In [1241]:
```
# RMSE on Training data
predicted_train=DT_model.fit(X_train, y_train).predict(X_train)
np.sqrt(metrics.mean_squared_error(y_train,predicted_train))
```

Out[1241]: 0.0

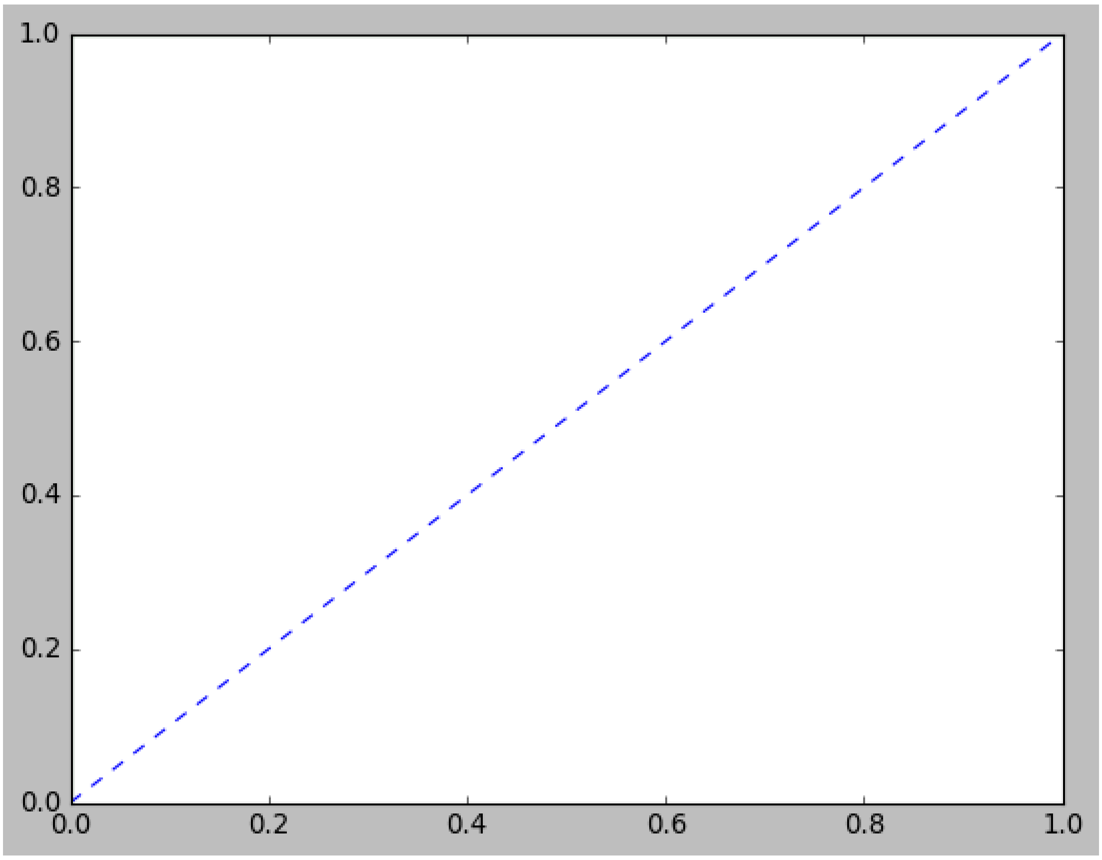
In [1249]:
```
#RMSE on Testing data
predicted_test=DT_model.fit(X_train, y_train).predict(X_test)
np.sqrt(metrics.mean_squared_error(y_test,predicted_test))
```

Out[1249]: 0.49559462778335206

AUC and ROC for training data

In [1257]:
```
# predict probabilities
probs =DT_model.predict_proba(X_train)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# calculate AUC
auc = roc_auc_score(y_train, probs)
print('AUC: %.3f' % auc)
# calculate roc curve
train_fpr, train_tpr, train_thresholds = roc_curve(y_train, probs)
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
plt.plot(train_fpr, train_tpr);
```

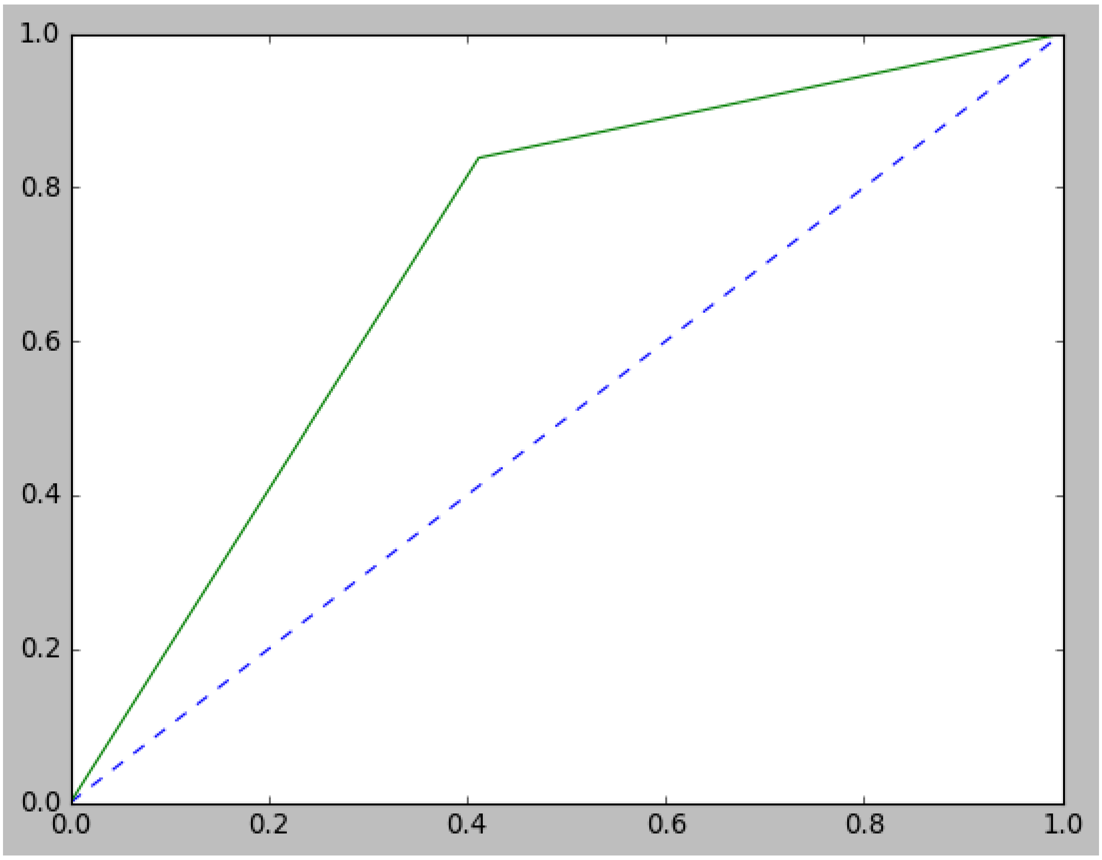
AUC: 1.000



AUC and ROC for test data

In [1264]:
```
# predict probabilities
probs = DT_model.predict_proba(X_test)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# calculate AUC
test_auc = roc_auc_score(y_test, probs)
print('AUC: %.3f' % auc)
# calculate roc curve
test_fpr, test_tpr, test_thresholds = roc_curve(y_test, probs)
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
plt.plot(test_fpr, test_tpr);
```

```
AUC: 0.912
```



Random Forest

In [1210]:
```
## Performance Matrix on train data set
y_train_predict = RF_model.predict(X_train)
model_score = RF_model.score(X_train, y_train)
print(model_score)
print(metrics.confusion_matrix(y_train, y_train_predict))
print(metrics.classification_report(y_train, y_train_predict))
```

```
1.0
[[307   0]
 [  0 754]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       307
           1       1.00      1.00      1.00       754

    accuracy                           1.00      1061
   macro avg       1.00      1.00      1.00      1061
weighted avg       1.00      1.00      1.00      1061
```

In [1217]:
```
## Performance Matrix on test data set
y_test_predict = RF_model.predict(X_test)
model_score = RF_model.score(X_test, y_test)
print(model_score)
print(metrics.confusion_matrix(y_test, y_test_predict))
print(metrics.classification_report(y_test, y_test_predict))
```

```
0.8026315789473685
[[ 93  60]
 [ 30 273]]
              precision    recall  f1-score   support

           0       0.76      0.61      0.67       153
           1       0.82      0.90      0.86       303

    accuracy                           0.80       456
   macro avg       0.79      0.75      0.77       456
weighted avg       0.80      0.80      0.80       456
```

In [1225]:
```
# R square on testing data (coeff of determinant)
RF_model.score(X_test, y_test)
```

Out[1225]: 0.8026315789473685

In [1232]:
```
# R square on training data
RF_model.score(X_train, y_train)
```

Out[1232]: 1.0

In [1240]:
```
# RMSE on Training data
predicted_train=RF_model.fit(X_train, y_train).predict(X_train)
np.sqrt(metrics.mean_squared_error(y_train,predicted_train))
```

Out[1240]: 0.0

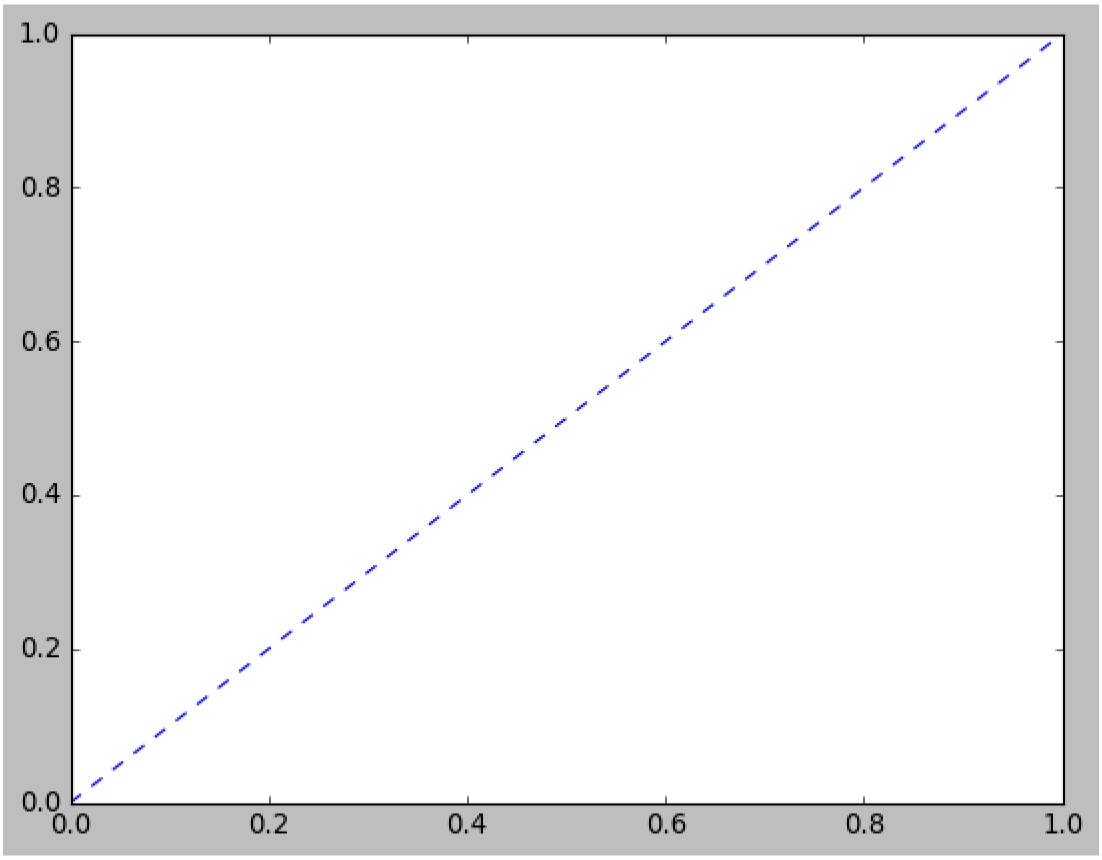
In [1248]:
```
#RMSE on Testing data
predicted_test=RF_model.fit(X_train, y_train).predict(X_test)
np.sqrt(metrics.mean_squared_error(y_test,predicted_test))
```

Out[1248]: 0.4442616583193193

AUC and ROC for training data

In [1256]:
```
# predict probabilities
probs =RF_model.predict_proba(X_train)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# calculate AUC
auc = roc_auc_score(y_train, probs)
print('AUC: %.3f' % auc)
# calculate roc curve
train_fpr, train_tpr, train_thresholds = roc_curve(y_train, probs)
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
plt.plot(train_fpr, train_tpr);
```

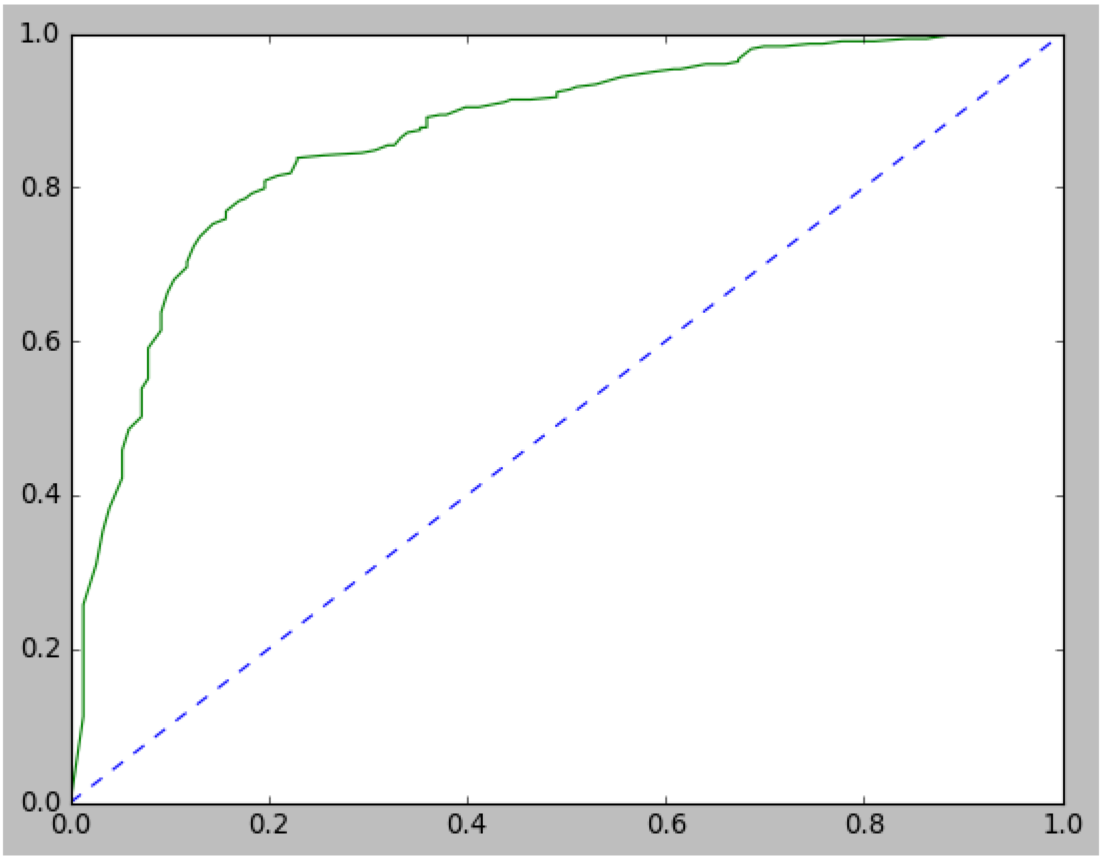
```
AUC: 1.000
```



AUC and ROC for test data

In [1263]:
```
# predict probabilities
probs = RF_model.predict_proba(X_test)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# calculate AUC
test_auc = roc_auc_score(y_test, probs)
print('AUC: %.3f' % auc)
# calculate roc curve
test_fpr, test_tpr, test_thresholds = roc_curve(y_test, probs)
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
plt.plot(test_fpr, test_tpr);
```

```
AUC: 0.912
```



Bagging with Random Forest

In [1211]:
```
## Performance Matrix on train data set
y_train_predict = Bagging_model.predict(X_train)
model_score = Bagging_model.score(X_train, y_train)
print(model_score)
print(metrics.confusion_matrix(y_train, y_train_predict))
print(metrics.classification_report(y_train, y_train_predict))
```

```
0.9679547596606974
[[277  30]
 [  4 750]]
              precision    recall  f1-score   support

           0       0.99      0.90      0.94       307
           1       0.96      0.99      0.98       754

    accuracy                           0.97      1061
   macro avg       0.97      0.95      0.96      1061
weighted avg       0.97      0.97      0.97      1061
```

In [1218]:
```
## Performance Matrix on test data set
y_test_predict = Bagging_model.predict(X_test)
model_score = Bagging_model.score(X_test, y_test)
print(model_score)
print(metrics.confusion_matrix(y_test, y_test_predict))
print(metrics.classification_report(y_test, y_test_predict))
```

```
0.8179824561403509
[[ 97  56]
 [ 27 276]]
              precision    recall  f1-score   support

           0       0.78      0.63      0.70       153
           1       0.83      0.91      0.87       303

    accuracy                           0.82       456
   macro avg       0.81      0.77      0.78       456
weighted avg       0.81      0.82      0.81       456
```

In [1224]:
```
# R square on testing data (coeff of determinant)
Bagging_model.score(X_test, y_test)
```

Out[1224]: 0.8179824561403509

In [1231]:
```
# R square on training data
Bagging_model.score(X_train, y_train)
```

Out[1231]: 0.9679547596606974

In [1239]:
```
# RMSE on Training data
predicted_train=Bagging_model.fit(X_train, y_train).predict(X_train)
np.sqrt(metrics.mean_squared_error(y_train,predicted_train))
```

Out[1239]: 0.17901184413133828

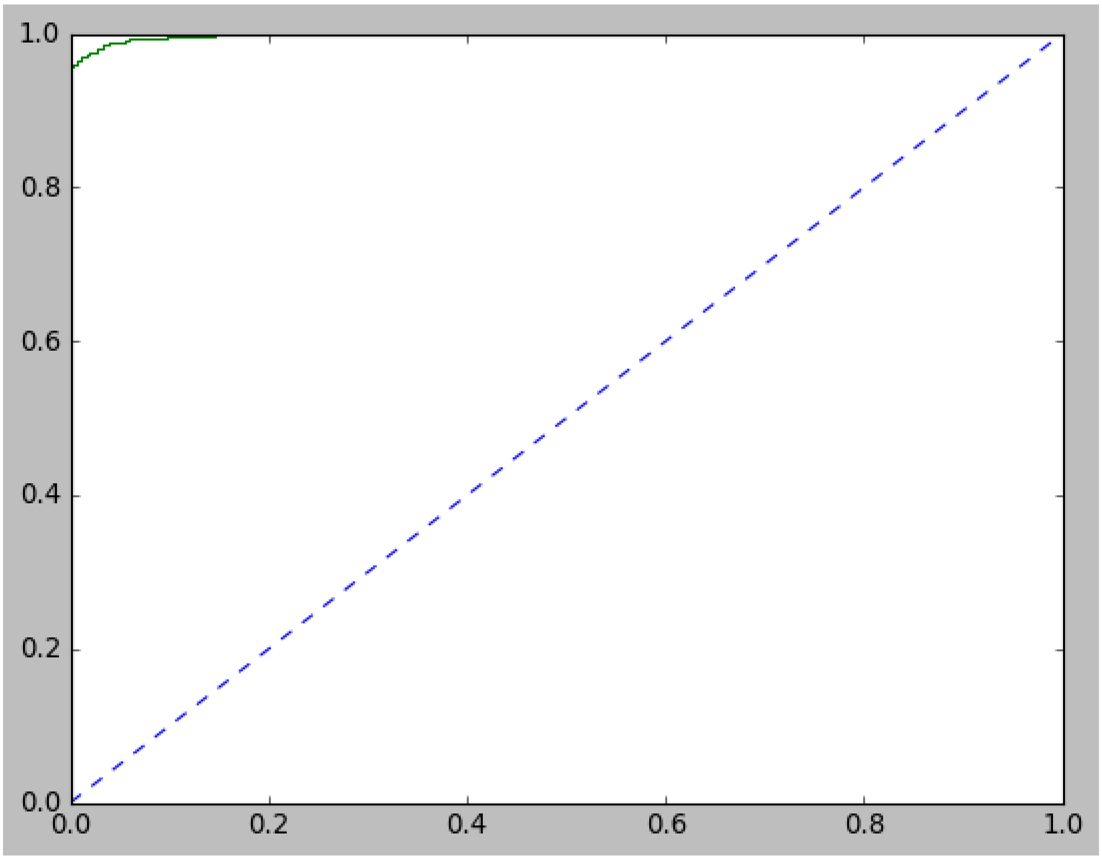
In [1247]:
```
#RMSE on Testing data
predicted_test=Bagging_model.fit(X_train, y_train).predict(X_test)
np.sqrt(metrics.mean_squared_error(y_test,predicted_test))
```

Out[1247]: 0.426635141379199

AUC and ROC for training data

In [1255]:
```
# predict probabilities
probs =Bagging_model.predict_proba(X_train)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# calculate AUC
auc = roc_auc_score(y_train, probs)
print('AUC: %.3f' % auc)
# calculate roc curve
train_fpr, train_tpr, train_thresholds = roc_curve(y_train, probs)
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
plt.plot(train_fpr, train_tpr);
```

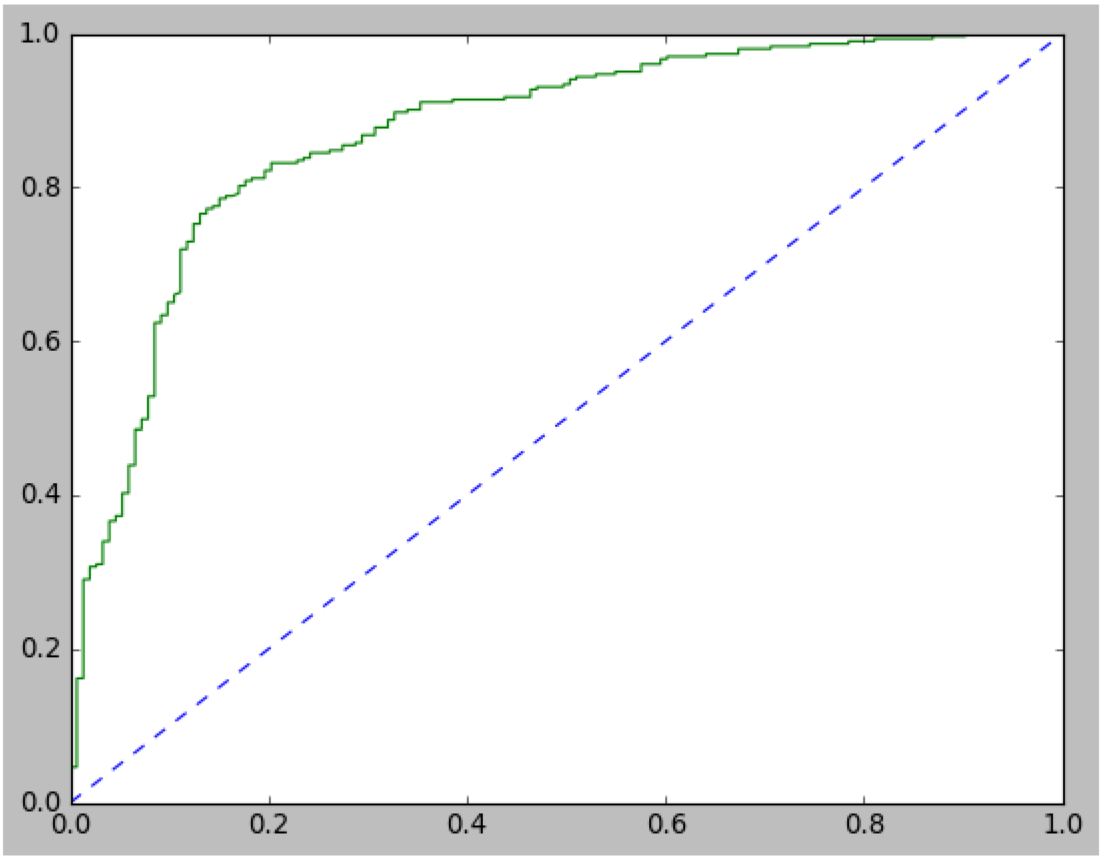
AUC: 0.998



AUC and ROC for Test data

In [1262]:
```
# predict probabilities
probs = Bagging_model.predict_proba(X_test)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# calculate AUC
test_auc = roc_auc_score(y_test, probs)
print('AUC: %.3f' % auc)
# calculate roc curve
test_fpr, test_tpr, test_thresholds = roc_curve(y_test, probs)
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
plt.plot(test_fpr, test_tpr);
```

```
AUC: 0.912
```



KNN with SMOTE

In [1212]:
```
## Performance Matrix on train data set
y_train_predict = KNN_SM_model.predict(X_train)
model_score = KNN_SM_model.score(X_train, y_train)
print(model_score)
print(metrics.confusion_matrix(y_train, y_train_predict))
print(metrics.classification_report(y_train, y_train_predict))
```

```
0.8360037700282752
[[284  23]
 [151 603]]
              precision    recall  f1-score   support

           0       0.65      0.93      0.77       307
           1       0.96      0.80      0.87       754

    accuracy                           0.84      1061
   macro avg       0.81      0.86      0.82      1061
weighted avg       0.87      0.84      0.84      1061
```

In [1219]:
```
## Performance Matrix on test data set
y_test_predict = KNN_SM_model.predict(X_test)
model_score = KNN_SM_model.score(X_test, y_test)
print(model_score)
print(metrics.confusion_matrix(y_test, y_test_predict))
print(metrics.classification_report(y_test, y_test_predict))
```

```
0.743421052631579
[[116  37]
 [ 80 223]]
              precision    recall  f1-score   support

           0       0.59      0.76      0.66       153
           1       0.86      0.74      0.79       303

    accuracy                           0.74       456
   macro avg       0.72      0.75      0.73       456
weighted avg       0.77      0.74      0.75       456
```

In [1223]:
```
# R square on testing data (coeff of determinant)
KNN_SM_model.score(X_test, y_test)
```

Out[1223]: 0.743421052631579

In [1230]:
```
# R square on training data
KNN_SM_model.score(X_train, y_train)
```

Out[1230]: 0.8360037700282752

In [1238]:
```
# RMSE on Training data
predicted_train=KNN_SM_model.fit(X_train, y_train).predict(X_train)
np.sqrt(metrics.mean_squared_error(y_train,predicted_train))
```

Out[1238]: 0.3871157758858186

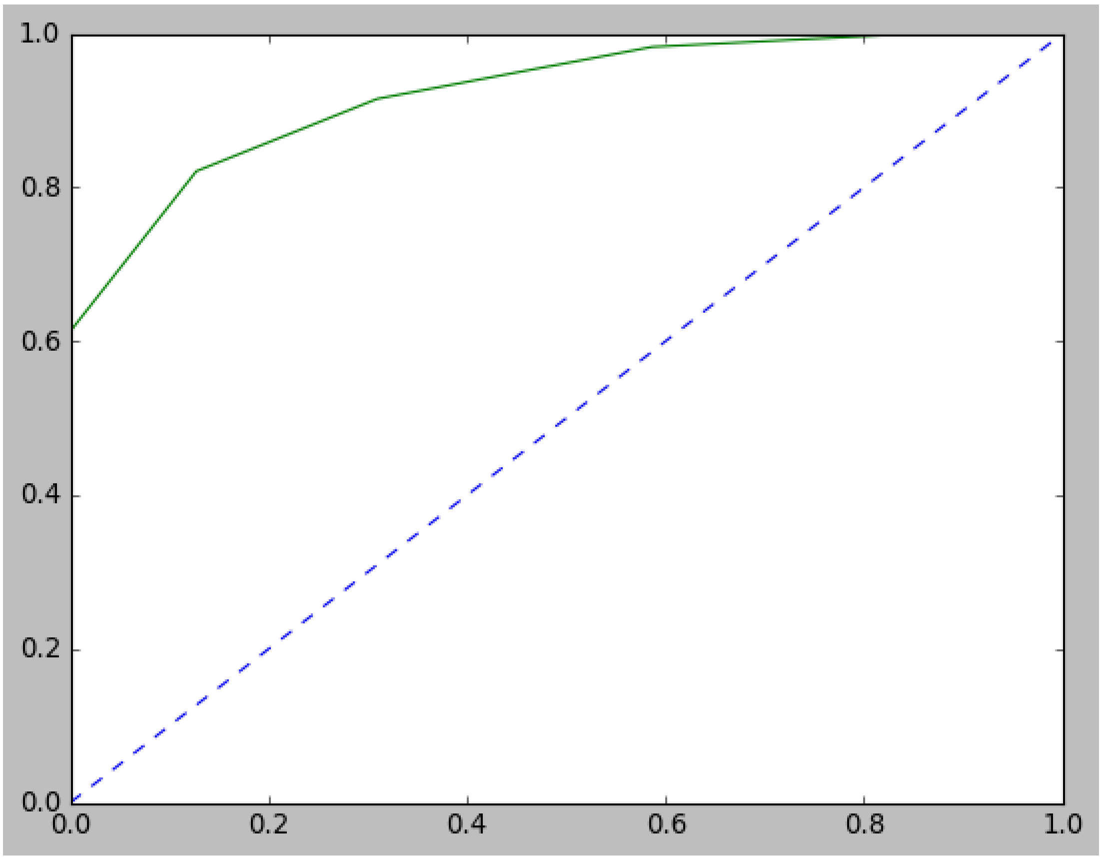
In [1246]:
```
#RMSE on Testing data
predicted_test=KNN_SM_model.fit(X_train, y_train).predict(X_test)
np.sqrt(metrics.mean_squared_error(y_test,predicted_test))
```

Out[1246]: 0.46594555814804667

AUC and ROC for Training data

In [1254]:
```
# predict probabilities
probs =KNN_SM_model.predict_proba(X_train)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# calculate AUC
auc = roc_auc_score(y_train, probs)
print('AUC: %.3f' % auc)
# calculate roc curve
train_fpr, train_tpr, train_thresholds = roc_curve(y_train, probs)
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
plt.plot(train_fpr, train_tpr);
```

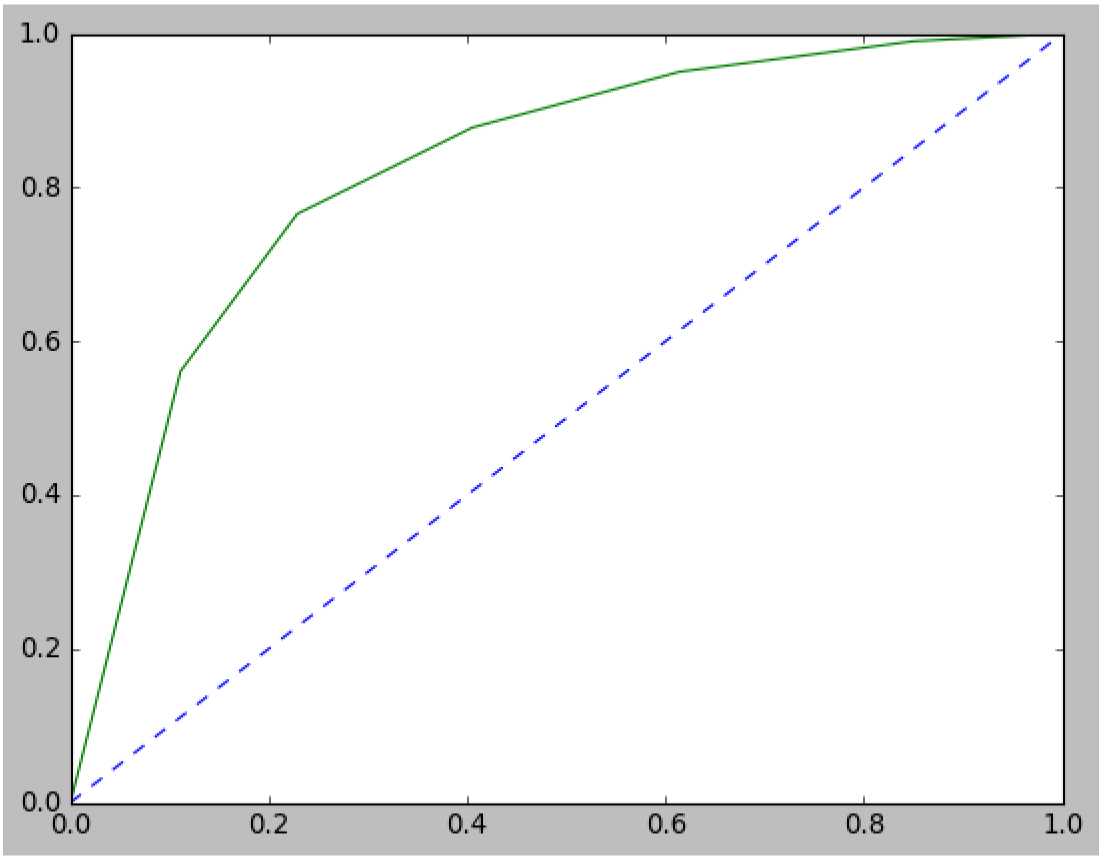
```
AUC: 0.924
```



AUC and ROC for test data

In [1261]:
```
# predict probabilities
probs = KNN_SM_model.predict_proba(X_test)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# calculate AUC
test_auc = roc_auc_score(y_test, probs)
print('AUC: %.3f' % auc)
# calculate roc curve
test_fpr, test_tpr, test_thresholds = roc_curve(y_test, probs)
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
plt.plot(test_fpr, test_tpr);
```

```
AUC: 0.912
```



NB with SMOTE

In [1213]:
```
## Performance Matrix on train data set
y_train_predict = NB_SM_model.predict(X_train)
model_score = NB_SM_model.score(X_train, y_train)
print(model_score)
print(metrics.confusion_matrix(y_train, y_train_predict))
print(metrics.classification_report(y_train, y_train_predict))
```

```
0.7115928369462771
[[245  62]
 [244 510]]
              precision    recall  f1-score   support

           0       0.50      0.80      0.62       307
           1       0.89      0.68      0.77       754

    accuracy                           0.71      1061
   macro avg       0.70      0.74      0.69      1061
weighted avg       0.78      0.71      0.72      1061
```

In [1267]:
```
## Performance Matrix on test data set
y_test_predict = NB_SM_model.predict(X_test)
model_score = KNN_SM_model.score(X_test, y_test)
print(model_score)
print(metrics.confusion_matrix(y_test, y_test_predict))
print(metrics.classification_report(y_test, y_test_predict))
```

```
0.7828947368421053
[[120  33]
 [ 88 215]]
              precision    recall  f1-score   support

           0       0.58      0.78      0.66       153
           1       0.87      0.71      0.78       303

    accuracy                           0.73       456
   macro avg       0.72      0.75      0.72       456
weighted avg       0.77      0.73      0.74       456
```

In [1222]:
```
# R square on testing data (coeff of determinant)
NB_SM_model.score(X_test, y_test)
```

Out[1222]: 0.6951754385964912

In [1229]:
```
# R square on training data
NB_SM_model.score(X_train, y_train)
```

Out[1229]: 0.7115928369462771

In [1237]:
```
# RMSE on Training data
predicted_train=NB_SM_model.fit(X_train, y_train).predict(X_train)
np.sqrt(metrics.mean_squared_error(y_train,predicted_train))
```

Out[1237]: 0.500706381327023

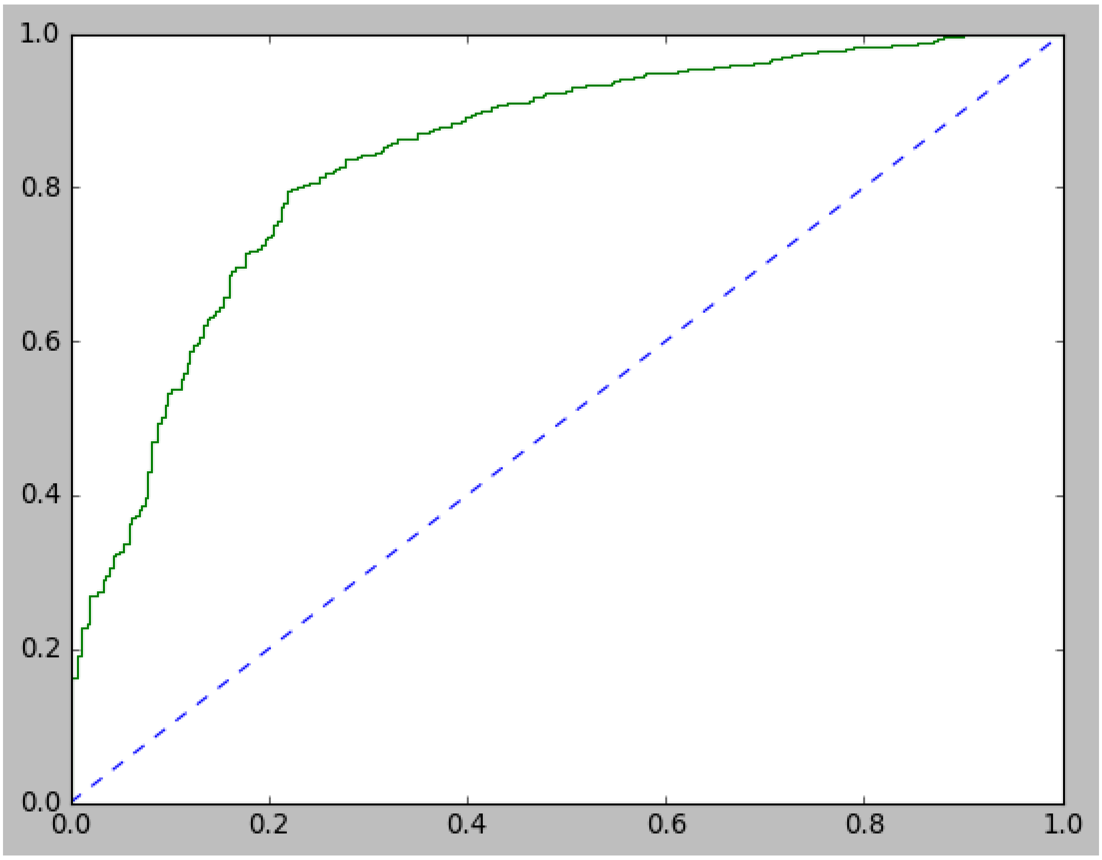
In [1245]:
```
#RMSE on Testing data
predicted_test=NB_SM_model.fit(X_train, y_train).predict(X_test)
np.sqrt(metrics.mean_squared_error(y_test,predicted_test))
```

Out[1245]: 0.5151221963699317

AUC and ROC for training data

In [1253]:
```
# predict probabilities
probs =NB_SM_model.predict_proba(X_train)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# calculate AUC
auc = roc_auc_score(y_train, probs)
print('AUC: %.3f' % auc)
# calculate roc curve
train_fpr, train_tpr, train_thresholds = roc_curve(y_train, probs)
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
plt.plot(train_fpr, train_tpr);
```

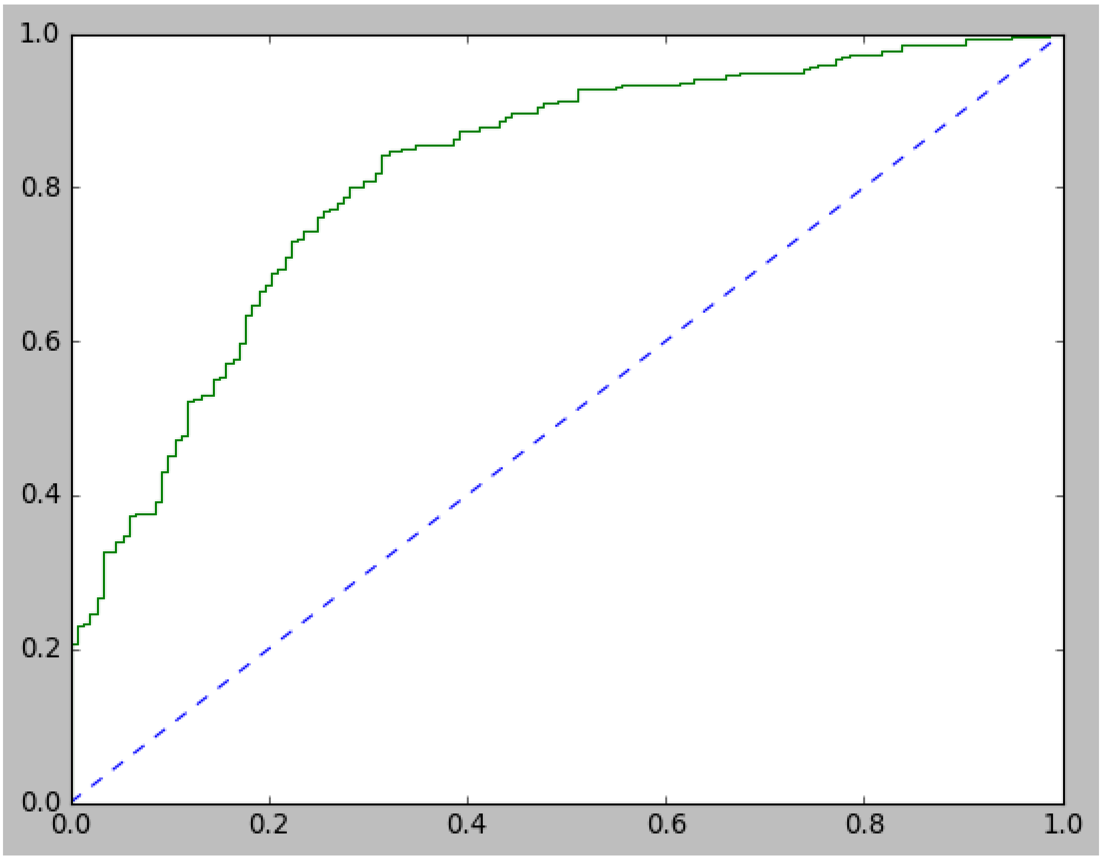
AUC: 0.843



AUC and ROC for test data

In [1260]:
```
# predict probabilities
probs = NB_SM_model.predict_proba(X_test)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# calculate AUC
test_auc = roc_auc_score(y_test, probs)
print('AUC: %.3f' % auc)
# calculate roc curve
test_fpr, test_tpr, test_thresholds = roc_curve(y_test, probs)
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
plt.plot(test_fpr, test_tpr);
```

```
AUC: 0.912
```



In [ ]: