Problem Statement

Context

CNBE, a prominent news channel, is gearing up to provide insightful coverage of recent elections, recognizing the importance of data-driven analysis. A comprehensive survey has been conducted, capturing the perspectives of 1525 voters across various demographic and socio-economic factors. This dataset encompasses 9 variables, offering a rich source of information regarding voters' characteristics and preferences.

Objective

The primary objective is to leverage machine learning to build a predictive model capable of forecasting which political party a voter is likely to support. This predictive model, developed based on the provided information, will serve as the foundation for creating an exit poll. The exit poll aims to contribute to the accurate prediction of the overall election outcomes, including determining which party is likely to secure the majority of seats.

Data Dictionary

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

Importing required libraries

In [1]:
```
# Pandas and Numpy libraries
import pandas as pd
import numpy as np

# libaries to help with data visualization
import matplotlib.pyplot as plt
import seaborn as sns

# to scale the data using z-score
from sklearn.preprocessing import StandardScaler
```

```
# To split the dataset into train and test datasets
from sklearn.model_selection import train_test_split

# To tune different models
from sklearn.model_selection import GridSearchCV

# To model the KNeighbors Classifier
from sklearn.neighbors import KNeighborsClassifier

# To model the Gaussian Navie Bayes classifier
from sklearn.naive_bayes import GaussianNB

# To model the Random Forest Classifier
from sklearn.ensemble import RandomForestClassifier

# To model the Bagging classifier
from sklearn.ensemble import BaggingClassifier

# To model the AdaBoost classifier
from sklearn.ensemble import AdaBoostClassifier

# To model the Gradient Boosting classifier
from sklearn.ensemble import GradientBoostingClassifier

# To calculate the accuracy score of the model
from sklearn import metrics
from sklearn.metrics import roc_auc_score,roc_curve,classification_report,confusion

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

Understanding the structure of data

In [2]:
```
df = pd.read_excel('Election_Data.xlsx') # Importing the data
```

In [3]:
```
df.head() # Returns first 5 rows
```

Out[3]:

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

Number of rows and columns in the dataset

In [4]:
```
# checking shape of the data

rows = str(df.shape[0])
columns = str(df.shape[1])

print(f"There are \033[1m" + rows + "\033[0m rows and \033[1m" + columns + "\033[0m
```

```
There are 1525 rows and 10 columns in the dataset.
```

Datatypes of the different columns in the dataset

In [5]:
```
df.info() # Concise summary of dataset
```

```
<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
```

There are 10 columns in the dataset. Out of which 8 have integer data type and 2 have object data type.

Finding missing values in the dataset

In [6]:
```
df.isna().sum()  # Count NaN values in all columns of dataset
```

Out[6]:
```
Unnamed: 0                0
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 any of the column in dataset.

Checking for Duplicates

In [7]:
```
df.duplicated().sum()
```

Out[7]:
```
0
```

There are no duplicate rows in the dataset.

Removing first column in the dataset

In [8]:
```
# Removing first column in the dataset as it is a serial number

df.drop('Unnamed: 0', axis=1, inplace=True)
```

Removed first column from the dataset as it is a serial number.

In [9]:
```
df.head()  # Returns first 5 rows
```

Out[9]:

	vote	age	economic.cond.national	economic.cond.household	Blair	Hague	Europe
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

Checking Summary Statistic

In [10]:
```
df.describe(include='all').T
```

Out[10]:

	count	unique	top	freq	mean	std	min	25%
vote	1525	2	Labour	1063	NaN	NaN	NaN	NaN
age	1525.0	NaN	NaN	NaN	54.182295	15.711209	24.0	41.0
economic.cond.national	1525.0	NaN	NaN	NaN	3.245902	0.880969	1.0	3.0
economic.cond.household	1525.0	NaN	NaN	NaN	3.140328	0.929951	1.0	3.0
Blair	1525.0	NaN	NaN	NaN	3.334426	1.174824	1.0	2.0
Hague	1525.0	NaN	NaN	NaN	2.746885	1.230703	1.0	2.0
Europe	1525.0	NaN	NaN	NaN	6.728525	3.297538	1.0	4.0
political.knowledge	1525.0	NaN	NaN	NaN	1.542295	1.083315	0.0	0.0
gender	1525	2	female	812	NaN	NaN	NaN	NaN

Observations and Insights:

1. Average age of voter is 54 years (minimum - 24 years, maximum - 93 years).
2. Most voters have assessed national economic condition as good in the survey.
3. Most voters have assessed household economic condition as good in the survey.
4. Most voters have assessed Labor leader as good in the survey.
5. Most voters have assessed Conservative leader as average in the survey.
6. 'Eurosceptic' sentiment (attitude toward European integration) is higher among the voters.
7. Both political parties are having fair knowledge in European integration.
8. Female voters are more than the Male voters.

Exploratory Data Analysis (EDA)

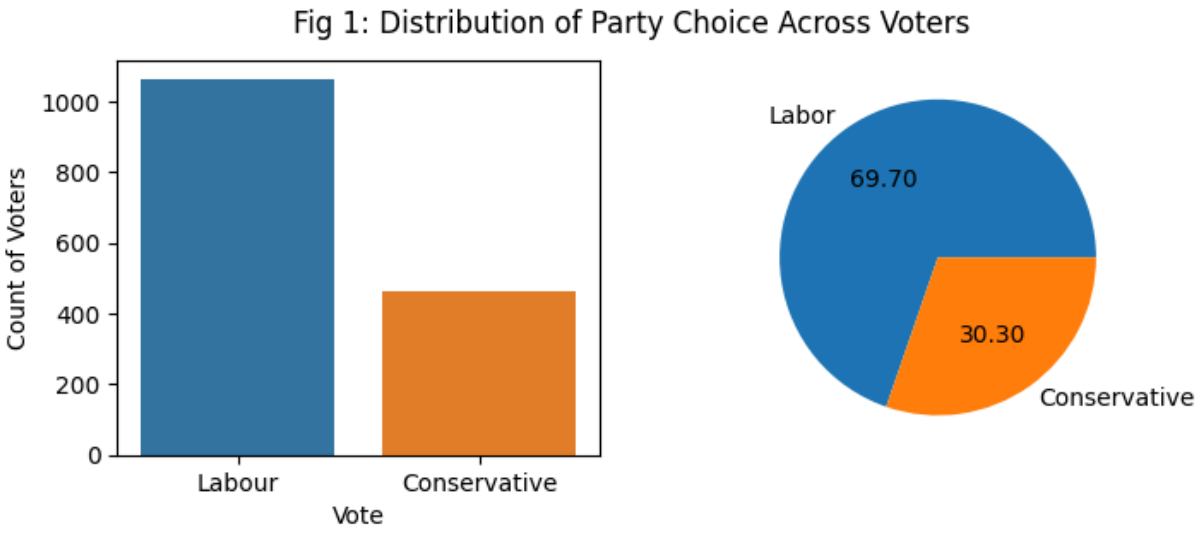
Univariate Analysis

vote

In [11]:
```
df['vote'].value_counts().sort_values() # Frequency of each distinct value in the v
```

Out[11]:
```
vote
Conservative     462
Labour          1063
Name: count, dtype: int64
```

In [12]:
```
# Count Plot and Pie Chart - Distribution of party choice across voters

fig, ax = plt.subplots(1,2, figsize=(8,3))
sns.countplot(data=df, x='vote', order = df['vote'].value_counts().index, ax=ax[0])
ax[0].set(xlabel = 'Vote', ylabel = 'Count of Voters')
ax[1]=plt.pie(df['vote'].value_counts(), labels=['Labor', 'Conservative'], autopct=
fig.suptitle('Fig 1: Distribution of Party Choice Across Voters')
plt.show()
```



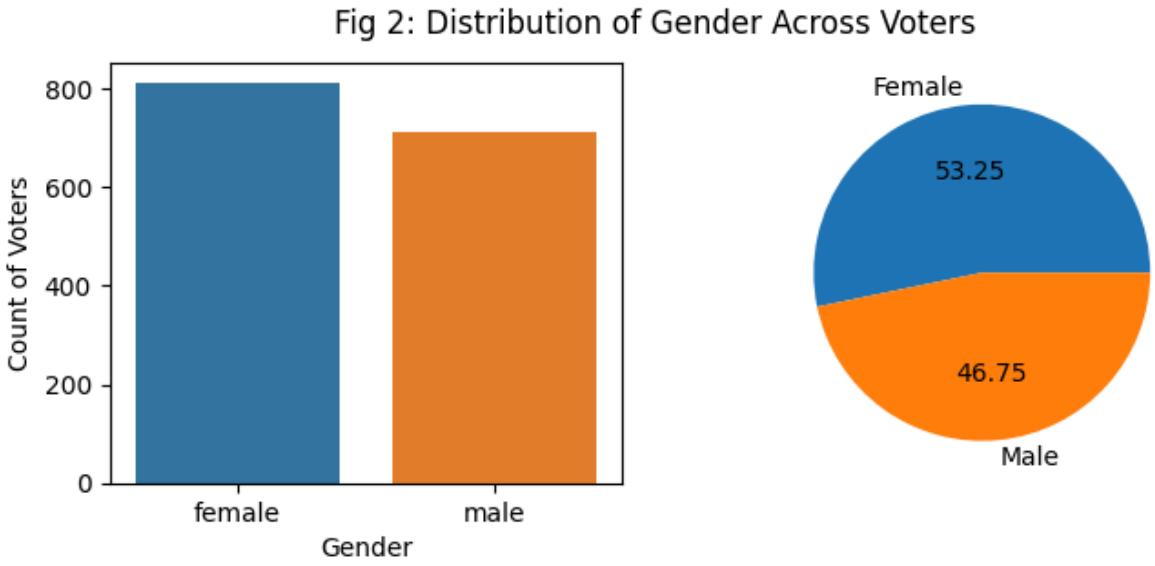
Most voters have preferred Labor leader (**60.70%**) instead of Conservative leader (**30.30%**) in the survey.

gender

In [13]:
```
df['gender'].value_counts().sort_values() # Frequency of each distinct value in the
```

Out[13]:
```
gender
male      713
female    812
Name: count, dtype: int64
```

In [14]:
```
# Count Plot and Pie Chart - Distribution of gender across voters

fig, ax = plt.subplots(1,2, figsize=(8,3))
sns.countplot(data=df, x='gender', order = df['gender'].value_counts().index, ax=ax
ax[0].set(xlabel = 'Gender', ylabel = 'Count of Voters')
ax[1]=plt.pie(df['gender'].value_counts(), labels=['Female', 'Male'], autopct='%.2f
fig.suptitle('Fig 2: Distribution of Gender Across Voters')
plt.show()
```

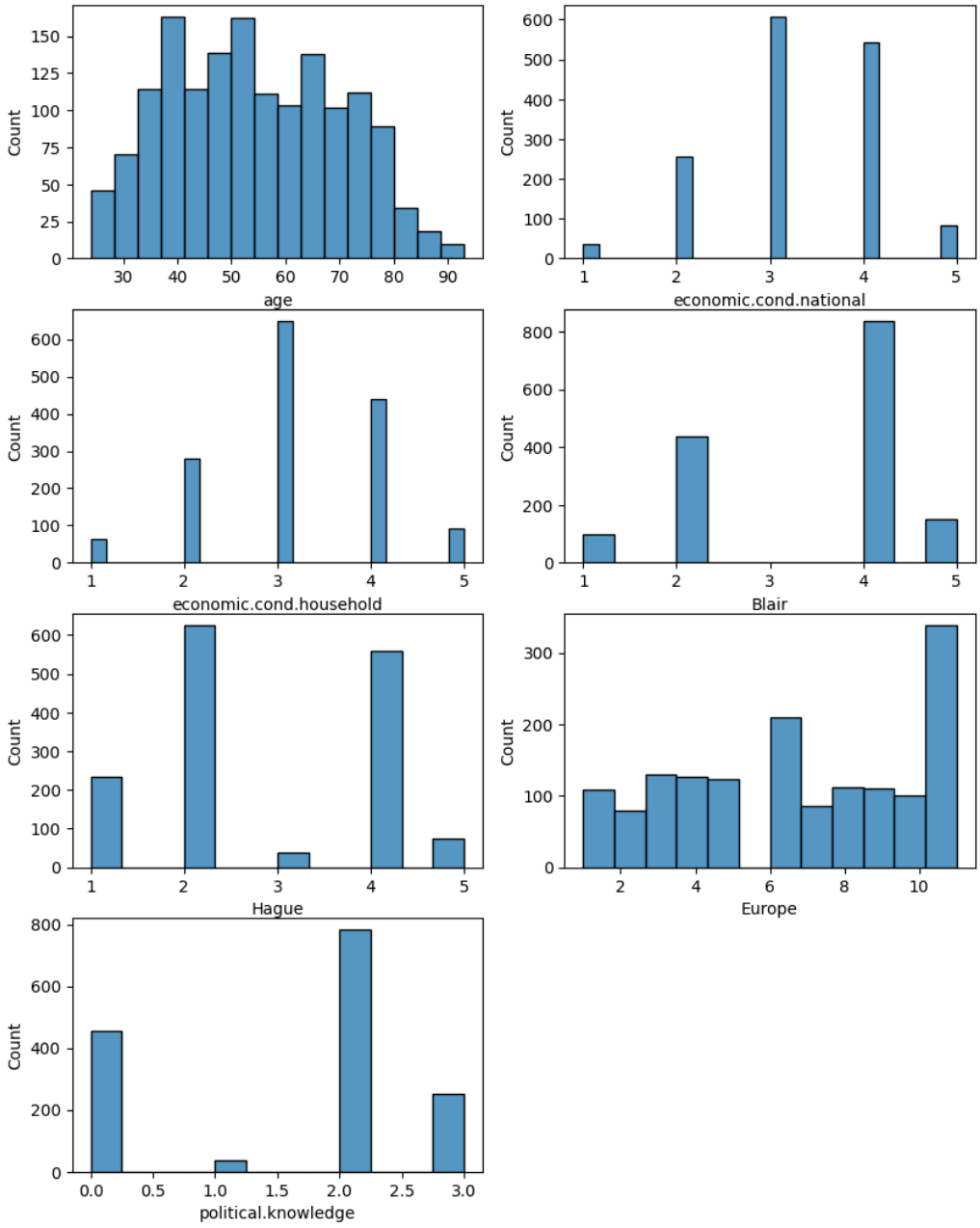


Female voters (**53.23%**) are higher in numbers than the Male voters (**46.75%**) in the survey.

In [15]:
```
# Hist Plots for age, economic.cond.national, economic.cond.household, Blair, Hague

fig, axes = plt.subplots(4,2, figsize=(10, 13))

sns.histplot(ax=axes[0, 0], data=df, x='age')
sns.histplot(ax=axes[0, 1], data=df, x='economic.cond.national')
sns.histplot(ax=axes[1, 0], data=df, x='economic.cond.household')
sns.histplot(ax=axes[1, 1], data=df, x='Blair')
sns.histplot(ax=axes[2, 0], data=df, x='Hague')
sns.histplot(ax=axes[2, 1], data=df, x='Europe')
sns.histplot(ax=axes[3, 0], data=df, x='political.knowledge')
axes[3,1].axis("off")
```

```
plt.suptitle('Fig 3: Hist Plots: age, economic.cond.national, economic.cond.household

plt.show()
```

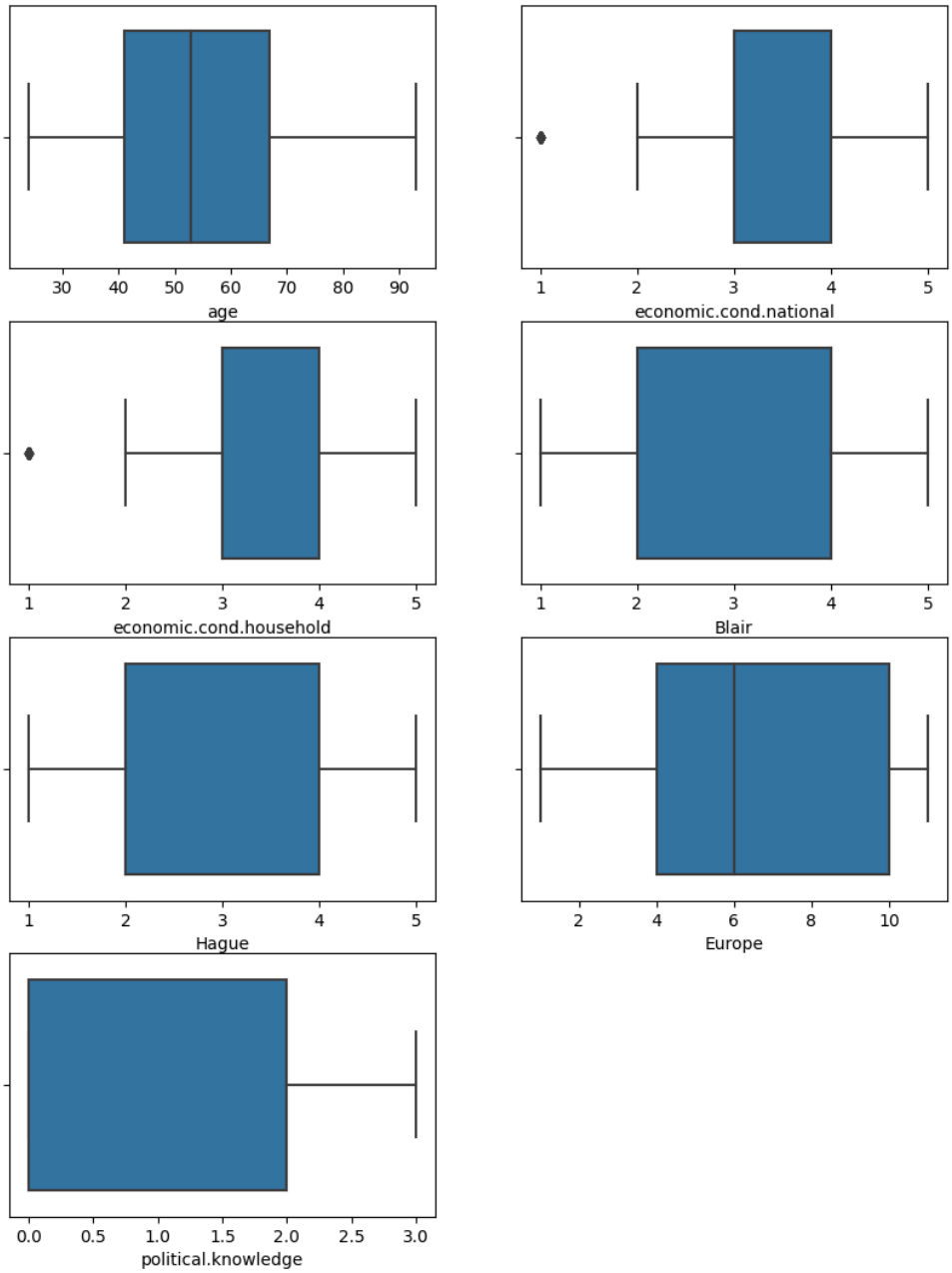
Fig 3: Hist Plots: age, economic.cond.national, economic.cond.household, Blair, Hague, Europe and political.knowledge



In [16]:
```
# Box Plots for age, economic.cond.national, economic.cond.household, Blair, Hague,

fig, axes = plt.subplots(4,2, figsize=(10, 13))

sns.boxplot(ax=axes[0, 0], data=df, x='age')
sns.boxplot(ax=axes[0, 1], data=df, x='economic.cond.national')
sns.boxplot(ax=axes[1, 0], data=df, x='economic.cond.household')
sns.boxplot(ax=axes[1, 1], data=df, x='Blair')
sns.boxplot(ax=axes[2, 0], data=df, x='Hague')
sns.boxplot(ax=axes[2, 1], data=df, x='Europe')
sns.boxplot(ax=axes[3, 0], data=df, x='political.knowledge')
axes[3,1].axis("off")
```

```
plt.suptitle('Fig 4: Box Plots: age, economic.cond.national, economic.cond.household
plt.show()
```

Fig 4: Box Plots: age, economic.cond.national, economic.cond.household, Blair, Hague, Europe and political.knowledge



Observations and Insights:

- No distribution is evenly distributed (symmetric).
- age distribution is Positively Skewed i.e. mean age of voter is more than the mode age of voter. Minimum age of voter is 24 years and maximum age of voter is 93 years.
- economic.cond.national and economic.cond.household distributions are Negatively Skewed i.e mean assessment of economic condition (national and household) is less than the mode assessment of economic condition (national and household). Both assessments are similarily distributed as well between the range of 1 to 5.

- Blair distribution is Negatively Skewed i.e. mean assessment of Labor leader is less than the mode assessment of Labor leader.
- Hauge distribution is Positively Skewed i.e. mean assessment of Conservative leader is more than the mode assessment of Conservative leader.
- 'Europe' sentiment (attitude toward European integration) is Negatively Skewed i.e. mean 'Europe' sentiment of voter is less than the mode 'Europe' sentiment of voter.
- political.knowledge distribution is Negatively Skewed i.e. mean assessment of political knowledge is less than the mode assessment of political.knowledge for both political parties.
- economic.cond.national and economic.cond.household distributions have few outliers.

Multivariate Analysis

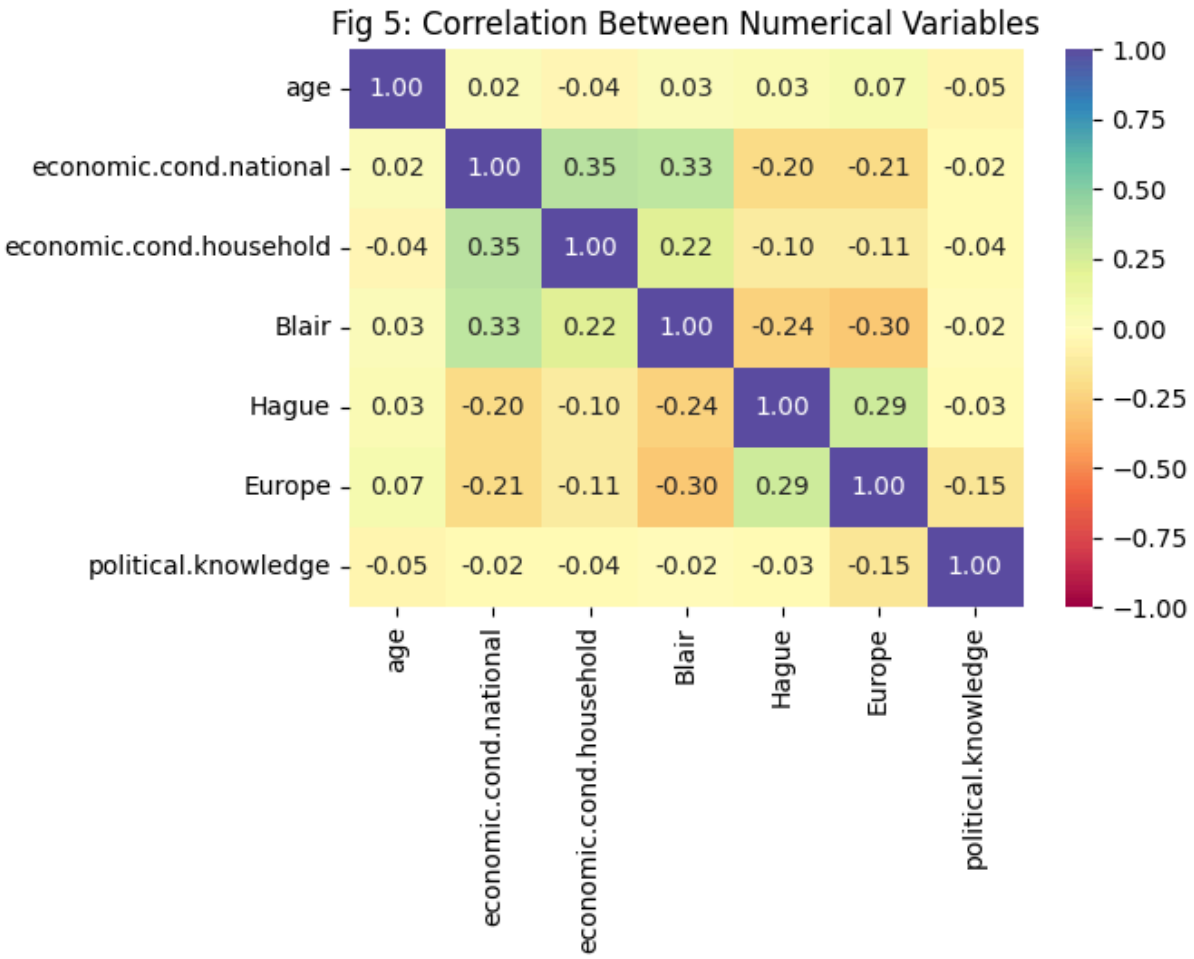
Correlation among variables

In [17]:
```
# Correlation between all numerical variables in the dataset

df_corr = df.select_dtypes(include=[np.number])
df_corr.corr()
```

Out[17]:

	age	economic.cond.national	economic.cond.household	
age	1.000000	0.018567	-0.041587	0.(
economic.cond.national	0.018567	1.000000	0.346303	0.3
economic.cond.household	-0.041587	0.346303	1.000000	0.2
Blair	0.030218	0.326878	0.215273	1.(
Hague	0.034626	-0.199766	-0.101956	-0.2
Europe	0.068880	-0.209429	-0.114885	-0.2
political.knowledge	-0.048490	-0.023624	-0.037810	-0.(

In [18]:
```
# Heatmap to plot correlation between all numerical variables in the dataset

plt.figure(figsize=(6, 4))
sns.heatmap(df_corr.corr(), annot=True, vmin=-1, vmax=1, fmt=".2f", cmap="Spectral"
plt.title('Fig 5: Correlation Between Numerical Variables')
plt.show()
```

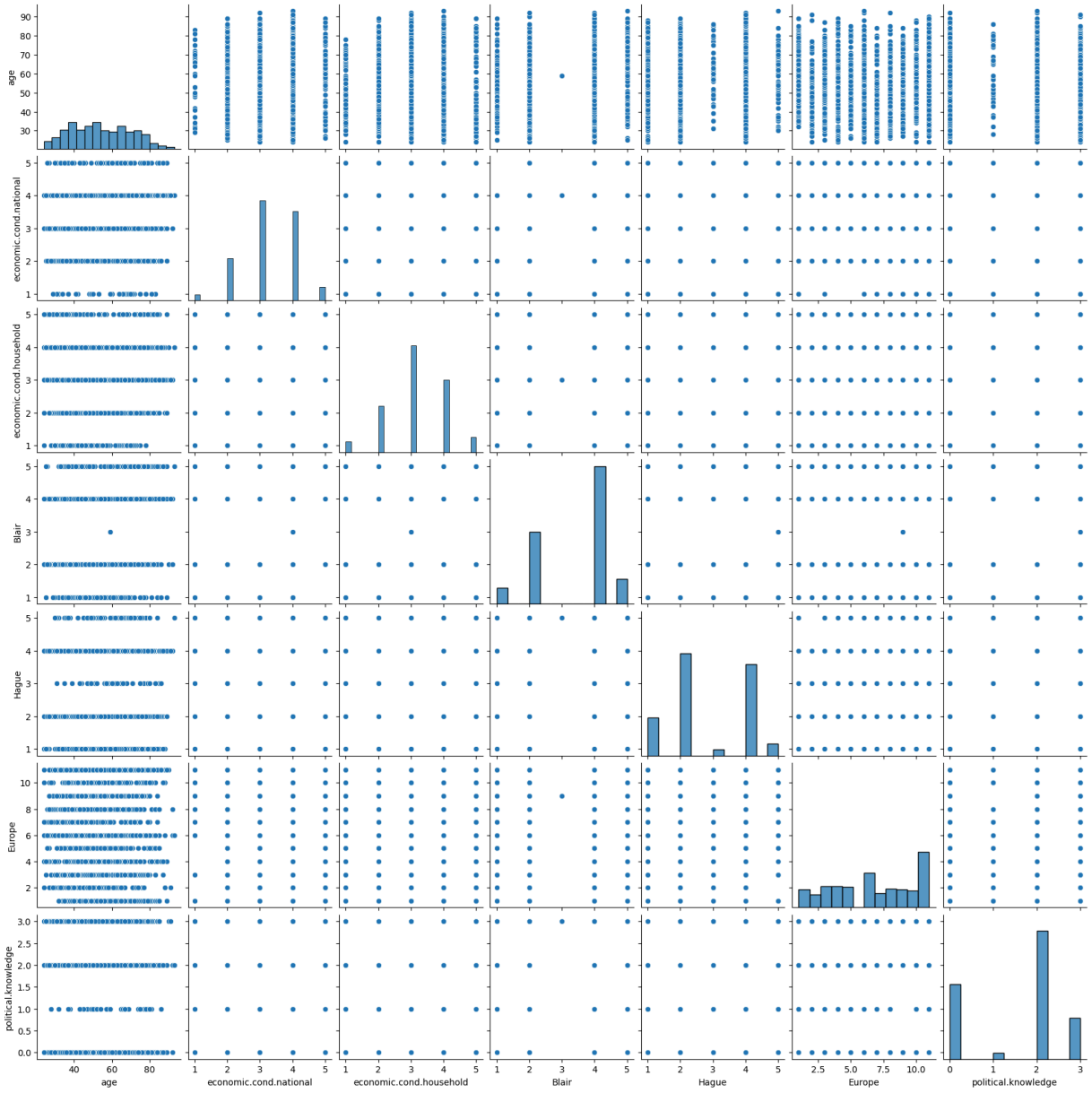


Observations and Insights:

- There is no strong correlation between numerical variables.

In [19]:
```
# Pair Plot to plot correlation between all numerical variables in the dataset

g = sns.pairplot(df)
g.fig.suptitle('Fig 6: Correlation Between Numerical Variables', y=1.02)
plt.show()
```

Fig 6: Correlation Between Numerical Variables



Observations and Insights:

- There is no strong correlation between numerical variables.

Converting all objects to categorical codes

In [20]:
```
# We are coding up the vote variable in an ordinal manner

df['vote']=np.where(df['vote'] =='Labour', '0', df['vote'])
df['vote']=np.where(df['vote'] =='Conservative', '1', df['vote'])
```

In [21]:
```
# We are coding up the gender variable in an ordinal manner

df['gender']=np.where(df['gender'] =='female', '0', df['gender'])
df['gender']=np.where(df['gender'] =='male', '1', df['gender'])
```

In [22]:
```
df['vote'].value_counts() # Frequency of each distinct value in the vote column
```

Out[22]:
```
vote
0    1063
1     462
Name: count, dtype: int64
```

In [23]:
```
df['gender'].value_counts() # Frequency of each distinct value in the gender column
```

Out[23]:
```
gender
0    812
1    713
Name: count, dtype: int64
```

In [24]:
```
df.head() # Returns first 5 rows
```

Out[24]:

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

Converting object variables to numeric variables

In [25]:
```
## Converting object variables to numeric variables

df['vote'] = df['vote'].astype('int64')
df['gender'] = df['gender'].astype('int64')
```

In [26]:
```
df.info() # Concise summary of dataset
```

```
<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   int64
 1   age                     1525 non-null   int64
 2   economic.cond.national  1525 non-null   int64
 3   economic.cond.household 1525 non-null   int64
 4   Blair                   1525 non-null   int64
 5   Hague                   1525 non-null   int64
 6   Europe                  1525 non-null   int64
 7   political.knowledge     1525 non-null   int64
 8   gender                  1525 non-null   int64
dtypes: int64(9)
memory usage: 107.4 KB
```

There are 9 columns in the dataset. All columns are now having int64 data type.

Scaling

In [27]:
```
# Copy all the predictor variables into X dataframe

X = df.drop('vote', axis=1)

# Copy target into the y dataframe

y = df['vote']
```

In [28]:
```
# scaling the data before model building

std_scaler = StandardScaler()
scaled_df = std_scaler.fit_transform(X)
```

In [29]:
```
# Creating dataframe after scaling

X = pd.DataFrame(scaled_df, columns=['age','economic.cond.national','economic.cond.
                                     'political.knowledge','gender'])
```

In [30]:
```
X.head() # Returns first 5 rows
```

Out[30]:

	age	economic.cond.national	economic.cond.household	Blair	Hague	Eu
0	-0.711973	-0.279218	-0.150948	0.566716	-1.419886	-1.43
1	-1.157661	0.856268	0.924730	0.566716	1.018544	-0.52
2	-1.221331	0.856268	0.924730	1.418187	-0.607076	-1.13
3	-1.921698	0.856268	-1.226625	-1.136225	-1.419886	-0.82
4	-0.839313	-1.414704	-1.226625	-1.987695	-1.419886	-0.22

Rational for Scaling

- Dataset is scaled as age variable unit is in years so it will get more importance (mean and median is high) as other variables values are ordinal numbers.

Train-Test Split

Split X and y into train and test sets (70:30 ratio)

In [31]:
```
# Split X and y into training and test set in 70:30 ratio

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30 , random_s
```

In [32]:
```
X_train.head() # Returns first 5 rows
```

Out[32]:

	age	economic.cond.national	economic.cond.household	Blair	Hague
1453	0.497751	-0.279218	-0.150948	-1.136225	-0.607076
275	-0.329955	-0.279218	-0.150948	-1.136225	-0.607076
1130	1.261787	0.856268	0.924730	0.566716	1.018544
1153	0.179402	-1.414704	-0.150948	0.566716	-0.607076
1172	-1.921698	0.856268	2.000408	0.566716	1.018544

In [33]:
```
X_test.head() # Returns first 5 rows
```

Out[33]:

	age	economic.cond.national	economic.cond.household	Blair	Hague
91	-0.329955	-2.550189	-2.302303	-1.136225	1.018544
1194	-1.285001	-0.279218	-0.150948	-1.136225	1.018544
201	-0.202616	-1.414704	-1.226625	0.566716	1.018544
613	-1.539680	-1.414704	-0.150948	0.566716	1.018544
283	-0.775643	-0.279218	-0.150948	-1.136225	-0.607076

Metrics of Choice

Accuracy, Precision and Recall

Rational: Dataset is balanced (minority class in the dataset constitutes more than 30% of total data).

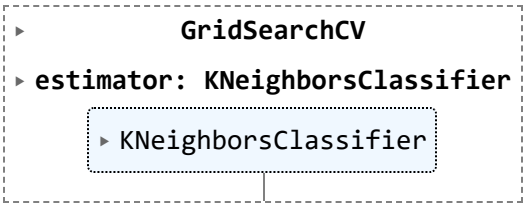
KNN Model

Find optimal value for 'n_neighbors'

In [34]:
```
#create a KNN model

KNN = KNeighborsClassifier()

#create a dictionary of all values we want to test for n_neighbors

param_grid = {'n_neighbors': np.arange(1, 25)}

#use gridsearch to test all values for n_neighbors

knn_gscv = GridSearchCV(KNN, param_grid, cv=5)

#fit model to data
```

```
knn_gscv.fit(X, y)
```

Out[34]:



In [35]:
```
#check top performing n_neighbors value

print('Top performing n_neighbors value:', knn_gscv.best_params_)
```

```
Top performing n_neighbors value: {'n_neighbors': 24}
```

In [36]:
```
#check mean score for the top performing value of n_neighbors

print('Mean score for the top performing value of n_neighbors:', knn_gscv.best_scor
```

```
Mean score for the top performing value of n_neighbors: 0.8321311475409836
```

Build KNN Model using optimal value of 'n_neighbors'

In [37]:
```
KNN_model=KNeighborsClassifier(n_neighbors=24)
KNN_model.fit(X_train,y_train)
```

Out[37]:

KNeighborsClassifier

```
KNeighborsClassifier(n_neighbors=24)
```

Predicting on Training and Test dataset

In [38]:
```
ytrain_predict = KNN_model.predict(X_train)
ytest_predict = KNN_model.predict(X_test)
```

Getting the Predicted Classes and Prob

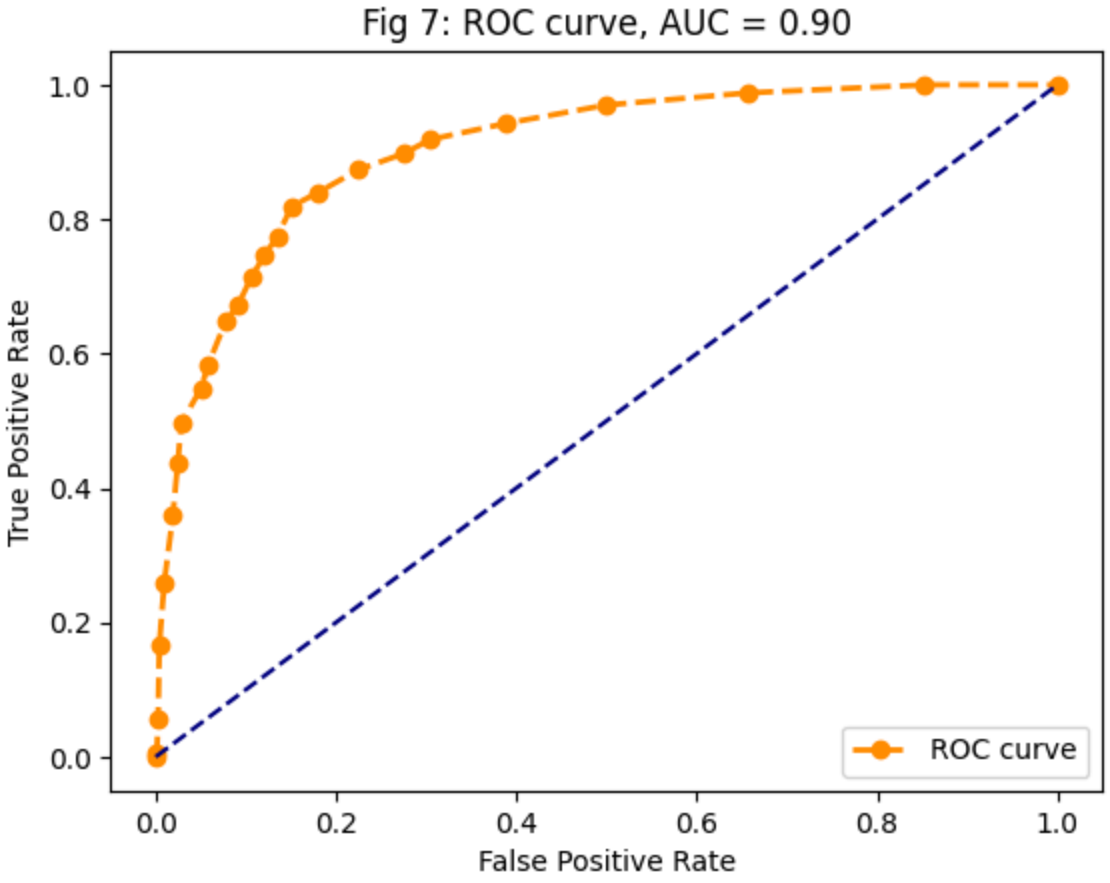
In [39]:
```
ytest_predict_prob = KNN_model.predict_proba(X_test)
pd.DataFrame(ytest_predict_prob).head()
```

Out[39]:

	0	1
0	0.208333	0.791667
1	0.291667	0.708333
2	0.750000	0.250000
3	0.625000	0.375000
4	0.833333	0.166667

AUC and ROC for the Training data

In [40]:
```
# 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)
# calculate roc curve
train_fpr, train_tpr, train_thresholds = roc_curve(y_train, probs)
# plot the roc curve for the model
plt.plot(train_fpr, train_tpr, linestyle='--', marker='o', color='darkorange', lw =
plt.plot([0, 1], [0, 1], color='navy', linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Fig 7: ROC curve, AUC = %.2f'%auc)
plt.legend(loc="lower right")
plt.show()
```

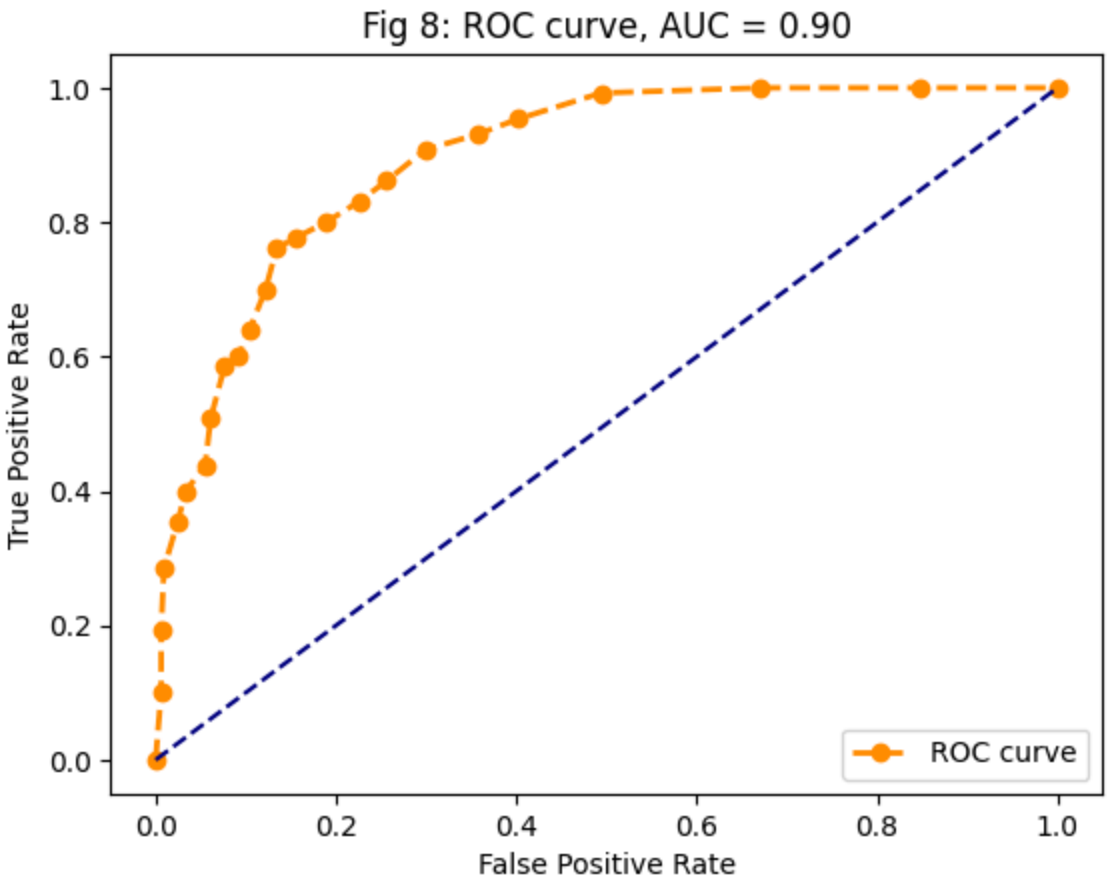


Observations and Insights:

- AOC (Area under the ROC Curve) is 0.90 (good).
- ROC curve is closer to the top-left corner which indicates a better performance.

AUC and ROC for the Test data

In [41]:
```
# 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)
# calculate roc curve
test_fpr, test_tpr, test_thresholds = roc_curve(y_test, probs)
# plot the roc curve for the model
plt.plot(test_fpr, test_tpr, linestyle='--', marker='o', color='darkorange', lw = 2
plt.plot([0, 1], [0, 1], color='navy', linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Fig 8: ROC curve, AUC = %.2f'%auc)
plt.legend(loc="lower right")
plt.show()
```



Observations and Insights:

- AOC (Area under the ROC Curve) is 0.90 (good).
- ROC curve is closer to the top-left corner which indicates a better performance.

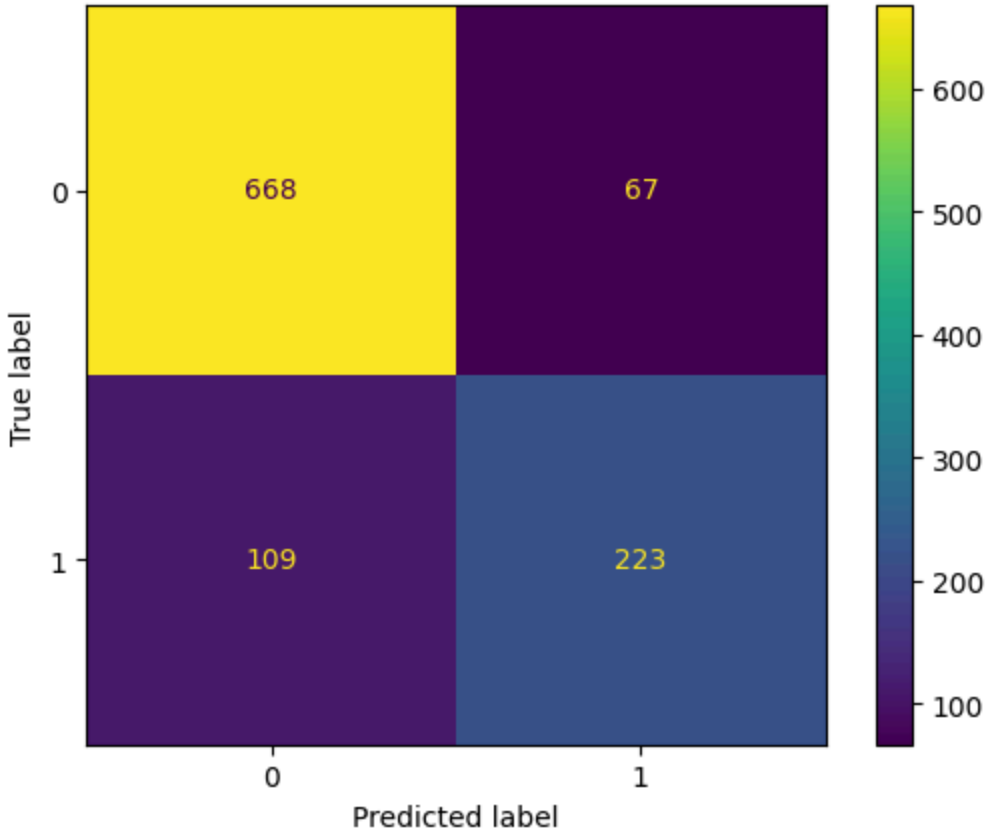
Confusion Matrix for the Training data

In [42]:
```
cm_train = confusion_matrix(y_train, ytrain_predict)
cm_train
```

Out[42]:
```
array([[668,  67],
       [109, 223]], dtype=int64)
```

In [43]:
```
disp = ConfusionMatrixDisplay(confusion_matrix=cm_train, display_labels=KNN_model.c
disp.plot()
```

Out[43]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x21be22e5ae0>
```



In [44]:
```
print(classification_report(y_train, ytrain_predict))
```

```
              precision    recall  f1-score   support

           0       0.86      0.91      0.88       735
           1       0.77      0.67      0.72       332

    accuracy                           0.84      1067
   macro avg       0.81      0.79      0.80      1067
weighted avg       0.83      0.84      0.83      1067
```

Model Performance (KNN Model - Training data):

For predicting party choice as Labor (Label 0):

Precision (86%) – 86% of voters predicted are actually voting the Labor party out of all voters predicted to vote Labor party.

Recall (91%) – Out of all voters actually voting the Labor party, 91% of voters are predicted to vote Labor party.

For predicting party choice as Conservative (Label 1):

Precision (77%) – 77% of voters predicted are actually voting the Conservative party out of all voters predicted to vote Conservative party.

Recall (67%) – Out of all voters actually voting the Conservative party, 67% of voters are predicted to vote Conservative party.

Overall accuracy of the model – 84% of total predictions are correct.

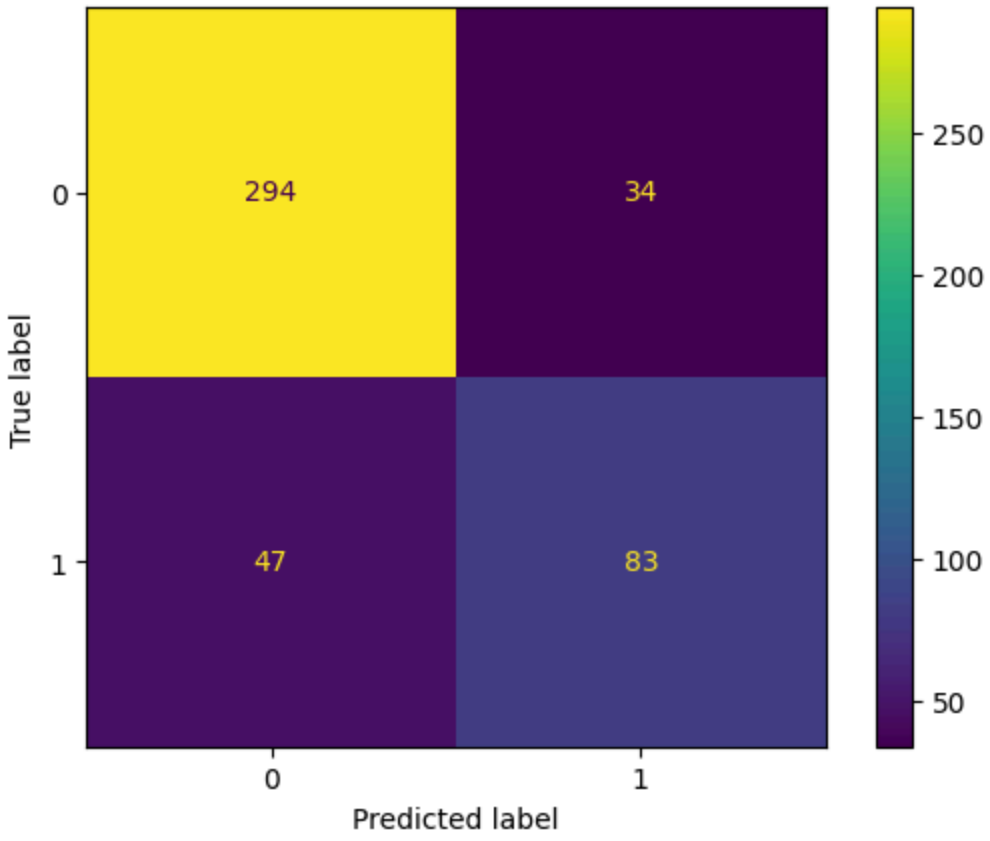
Confusion Matrix for the Test data

In [45]:
```
cm_test = confusion_matrix(y_test, ytest_predict)
cm_test
```

Out[45]:
```
array([[294,  34],
       [ 47,  83]], dtype=int64)
```

In [46]:
```
disp = ConfusionMatrixDisplay(confusion_matrix=cm_test, display_labels=KNN_model.cl
disp.plot()
```

Out[46]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x21be30eae60>
```



In [47]:
```
print(classification_report(y_test, ytest_predict))
```

```
              precision    recall  f1-score   support

           0       0.86      0.90      0.88       328
           1       0.71      0.64      0.67       130

    accuracy                           0.82       458
   macro avg       0.79      0.77      0.78       458
weighted avg       0.82      0.82      0.82       458
```

Model Performance (KNN Model - Test data):

For predicting party choice as Labor (Label 0):

Precision (86%) – 86% of voters predicted are actually voting the Labor party out of all voters predicted to vote Labor party.

Recall (90%) – Out of all voters actually voting the Labor party, 90% of voters are predicted to vote Labor party.

For predicting party choice as Conservative (Label 1):

Precision (71%) – 71% of voters predicted are actually voting the Conservative party out of all voters predicted to vote Conservative party.

Recall (64%) – Out of all voters actually voting the Conservative party, 64% of voters are predicted to vote Conservative party.

Overall accuracy of the model – 82% of total predictions are correct.

Navie Bayes Model

In [48]:
```
#Build Navie Bayes Model

NB_model = GaussianNB()
NB_model.fit(X_train, y_train)
```

Out[48]:
```
▾ GaussianNB

GaussianNB()
```

Predicting on Training and Test dataset

In [49]:
```
ytrain_predict = NB_model.predict(X_train)
ytest_predict = NB_model.predict(X_test)
```

Getting the Predicted Classes and Prob

In [50]:
```
ytest_predict_prob = NB_model.predict_proba(X_test)
```

```
pd.DataFrame(ytest_predict_prob).head()
```

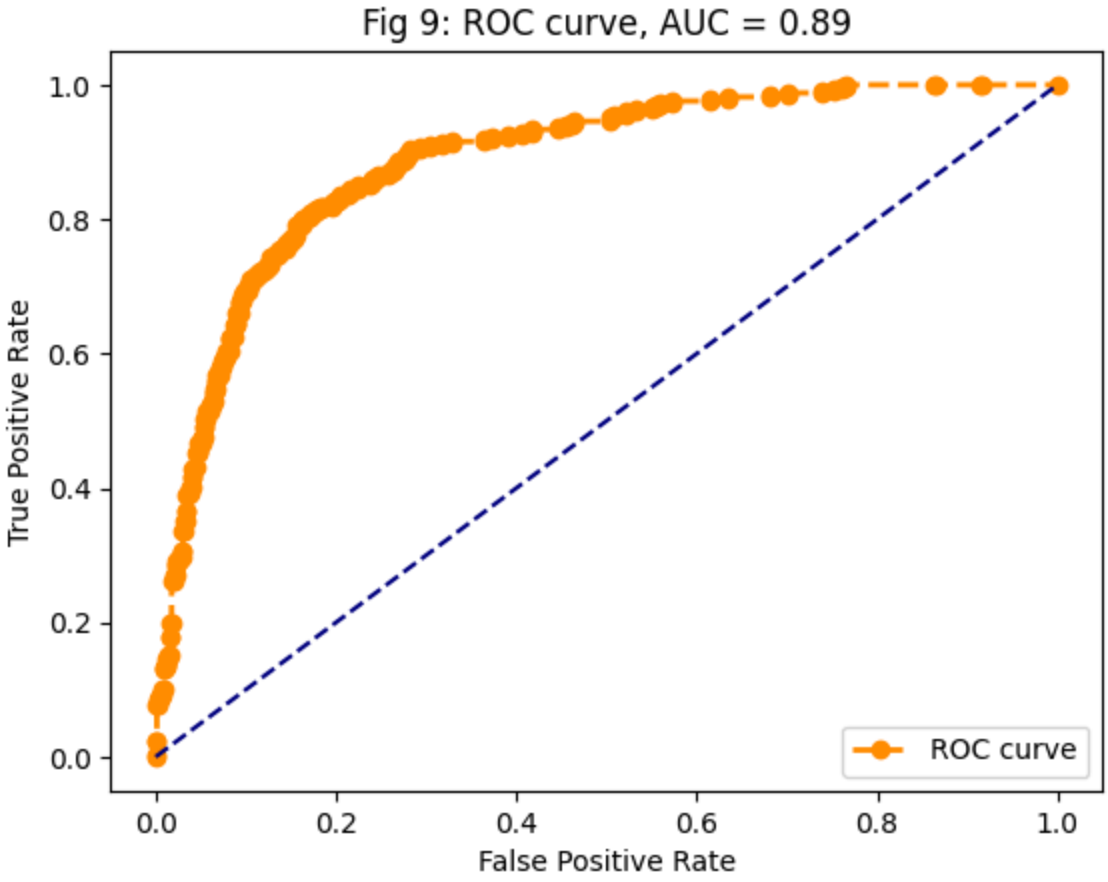
Out[50]:

	0	1
0	0.007418	0.992582
1	0.127536	0.872464
2	0.565517	0.434483
3	0.463956	0.536044
4	0.757823	0.242177

AUC and ROC for the Training data

In [51]:

```
# 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)
# calculate roc curve
train_fpr, train_tpr, train_thresholds = roc_curve(y_train, probs)
# plot the roc curve for the model
plt.plot(train_fpr, train_tpr, linestyle='--', marker='o', color='darkorange', lw =
plt.plot([0, 1], [0, 1], color='navy', linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Fig 9: ROC curve, AUC = %.2f'%auc)
plt.legend(loc="lower right")
plt.show()
```

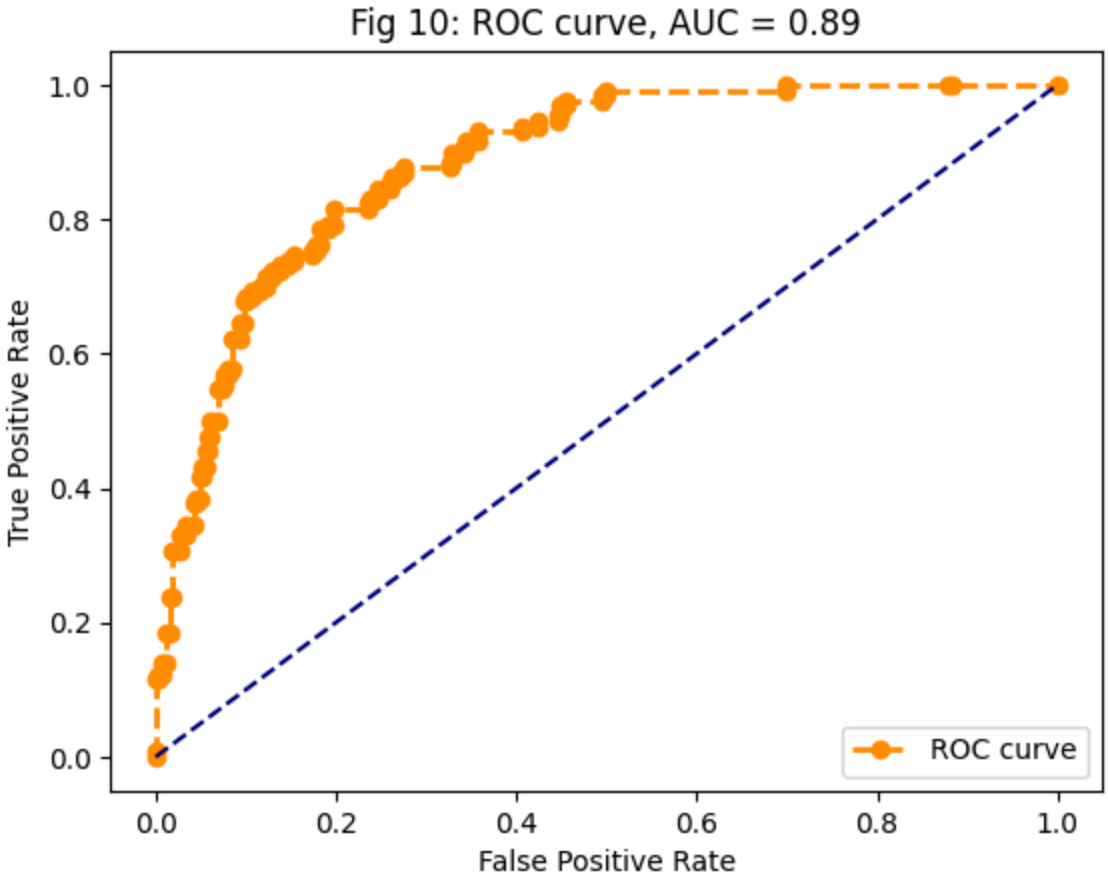


Observations and Insights:

- AOC (Area under the ROC Curve) is 0.89 (good).
- ROC curve is closer to the top-left corner which indicates a better performance.

AUC and ROC for the Test data

In [52]:
```
# 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)
# calculate roc curve
test_fpr, test_tpr, test_thresholds = roc_curve(y_test, probs)
# plot the roc curve for the model
plt.plot(test_fpr, test_tpr, linestyle='--', marker='o', color='darkorange', lw = 2
plt.plot([0, 1], [0, 1], color='navy', linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Fig 10: ROC curve, AUC = %.2f'%auc)
plt.legend(loc="lower right")
plt.show()
```



Observations and Insights:

- AOC (Area under the ROC Curve) is 0.89 (good).
- ROC curve is closer to the top-left corner which indicates a better performance.

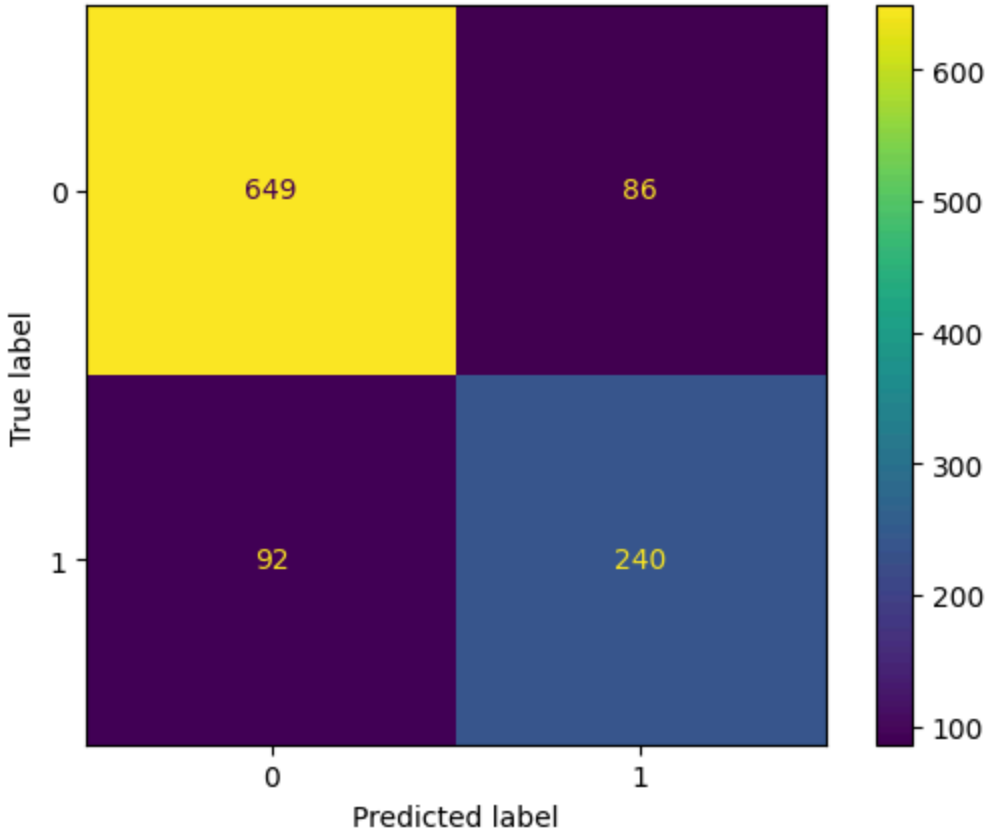
Confusion Matrix for the Training data

In [53]:
```
cm_train = confusion_matrix(y_train, ytrain_predict)
cm_train
```

Out[53]:
```
array([[649,  86],
       [ 92, 240]], dtype=int64)
```

In [54]:
```
disp = ConfusionMatrixDisplay(confusion_matrix=cm_train, display_labels=NB_model.cl
disp.plot()
```

Out[54]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x21be2fc3d60>
```



In [55]:
```
print(classification_report(y_train, ytrain_predict))
```

```
              precision    recall  f1-score   support

           0       0.88      0.88      0.88       735
           1       0.74      0.72      0.73       332

    accuracy                           0.83      1067
   macro avg       0.81      0.80      0.80      1067
weighted avg       0.83      0.83      0.83      1067
```

Model Performance (Navie Bayes Model - Training data):

For predicting party choice as Labor (Label 0):

Precision (88%) – 88% of voters predicted are actually voting the Labor party out of all voters predicted to vote Labor party.

Recall (88%) – Out of all voters actually voting the Labor party, 88% of voters are predicted to vote Labor party.

For predicting party choice as Conservative (Label 1):

Precision (74%) – 74% of voters predicted are actually voting the Conservative party out of all voters predicted to vote Conservative party.

Recall (72%) – Out of all voters actually voting the Conservative party, 72% of voters are predicted to vote Conservative party.

Overall accuracy of the model – 83% of total predictions are correct.

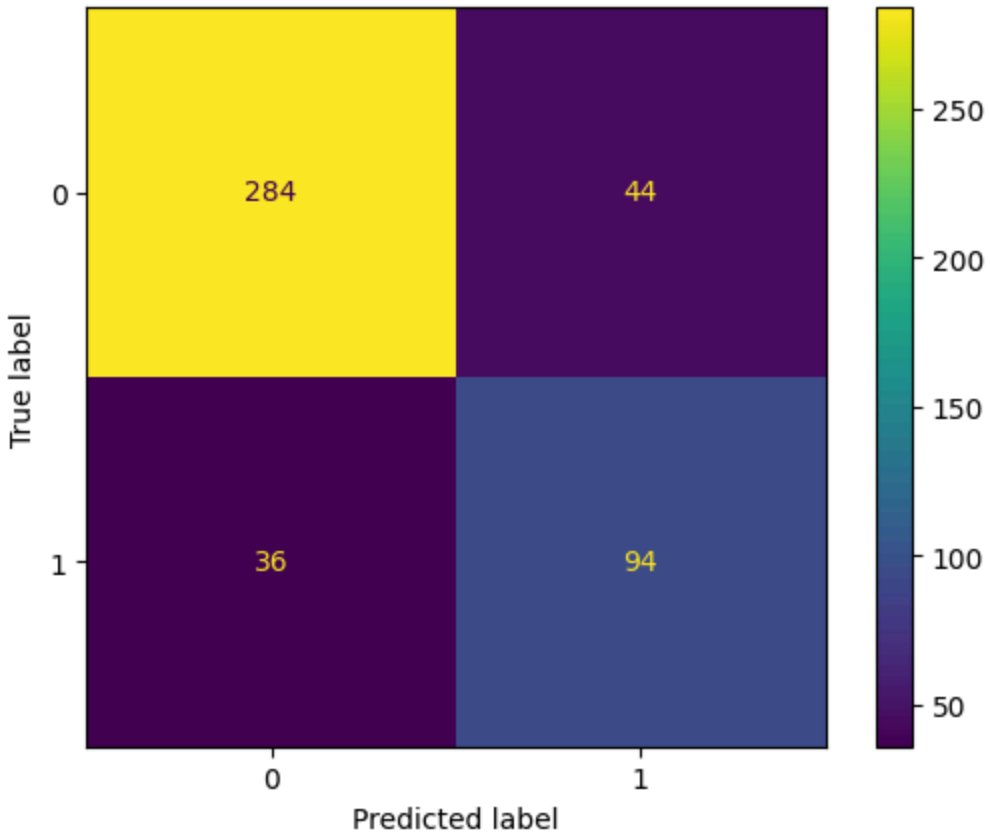
Confusion Matrix for the Test data

In [56]:
```
cm_test = confusion_matrix(y_test, ytest_predict)
cm_test
```

Out[56]:
```
array([[284,  44],
       [ 36,  94]], dtype=int64)
```

In [57]:
```
disp = ConfusionMatrixDisplay(confusion_matrix=cm_test, display_labels=NB_model.cla
disp.plot()
```

Out[57]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x21be2ef28c0>
```



In [58]:
```
print(classification_report(y_test, ytest_predict))
```

```
              precision    recall  f1-score   support

           0       0.89      0.87      0.88       328
           1       0.68      0.72      0.70       130

    accuracy                           0.83       458
   macro avg       0.78      0.79      0.79       458
weighted avg       0.83      0.83      0.83       458
```

Model Performance (Navie Bayes Model - Test data):

For predicting party choice as Labor (Label 0):

Precision (89%) – 89% of voters predicted are actually voting the Labor party out of all voters predicted to vote Labor party.

Recall (87%) – Out of all voters actually voting the Labor party, 87% of voters are predicted to vote Labor party.

For predicting party choice as Conservative (Label 1):

Precision (68%) – 68% of voters predicted are actually voting the Conservative party out of all voters predicted to vote Conservative party.

Recall (72%) – Out of all voters actually voting the Conservative party, 72% of voters are predicted to vote Conservative party.

Overall accuracy of the model – 83% of total predictions are correct

Random Forest Model

In [59]:
```
# Initialise a Random Forest Classifier

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

Out[59]:
```
RandomForestClassifier(random_state=1)
```

Predicting on Training and Test dataset

In [60]:
```
ytrain_predict = RF_model.predict(X_train)
ytest_predict = RF_model.predict(X_test)
```

Getting the Predicted Probabilities

In [61]:
```
ytest_predict_prob=RF_model.predict_proba(X_test)
```

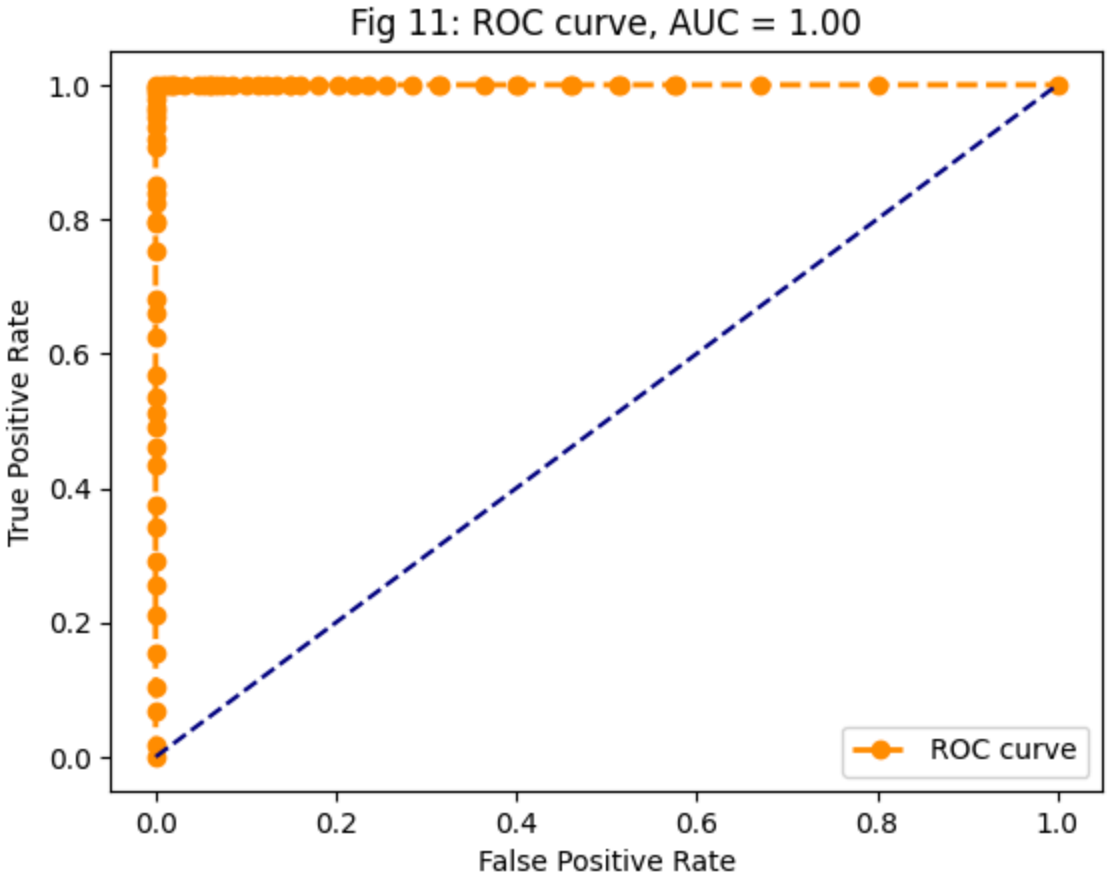
In [62]:
```
pd.DataFrame(ytest_predict_prob).head()
```

Out[62]:

	0	1
0	0.26	0.74
1	0.17	0.83
2	0.74	0.26
3	0.60	0.40
4	0.89	0.11

AUC and ROC for the Training data

In [63]:
```
# 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)
# calculate roc curve
train_fpr, train_tpr, train_thresholds = roc_curve(y_train, probs)
# plot the roc curve for the model
plt.plot(train_fpr, train_tpr, linestyle='--', marker='o', color='darkorange', lw =
plt.plot([0, 1], [0, 1], color='navy', linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Fig 11: ROC curve, AUC = %.2f'%auc)
plt.legend(loc="lower right")
plt.show()
```

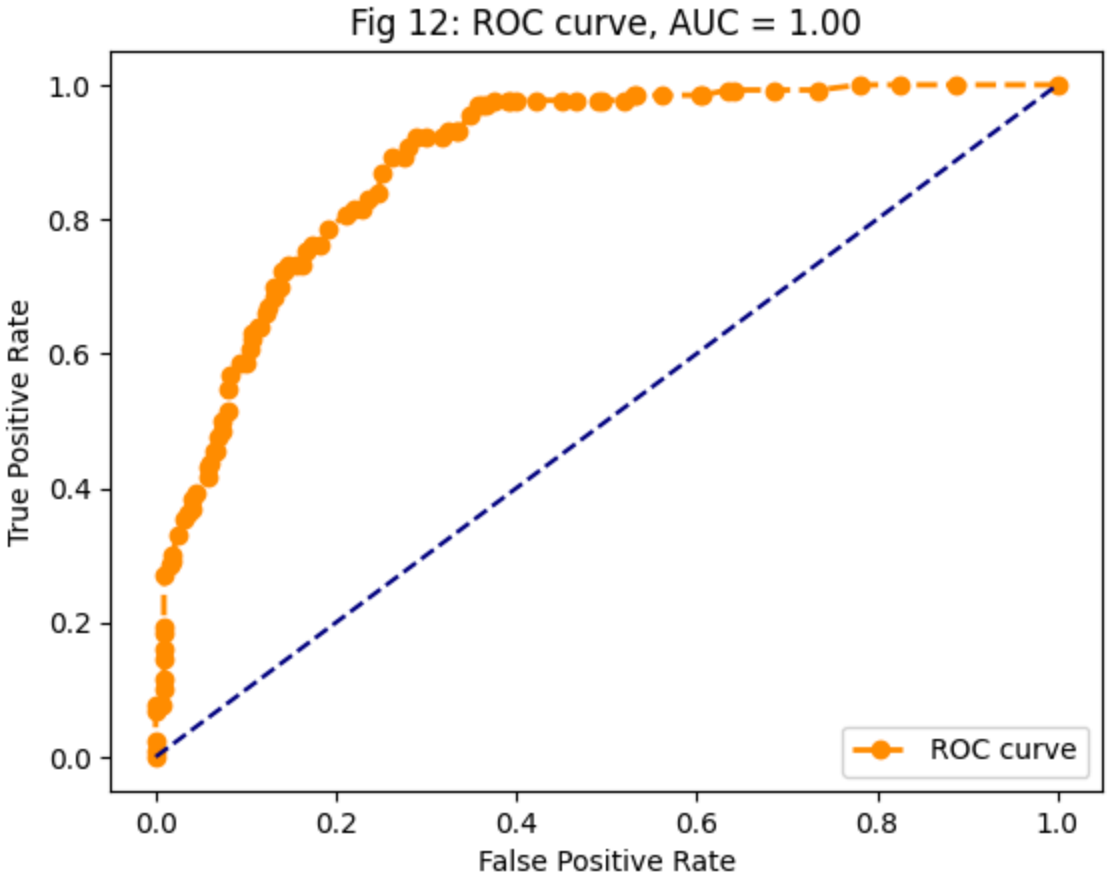


Observations and Insights:

- AOC (Area under the ROC Curve) is 1.00 (excellent).
- ROC curve is closer to the top-left corner which indicates a better performance.

AUC and ROC for the Test data

In [64]:
```
# 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)
# calculate roc curve
test_fpr, test_tpr, test_thresholds = roc_curve(y_test, probs)
# plot the roc curve for the model
plt.plot(test_fpr, test_tpr, linestyle='--', marker='o', color='darkorange', lw = 2
plt.plot([0, 1], [0, 1], color='navy', linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Fig 12: ROC curve, AUC = %.2f'%auc)
plt.legend(loc="lower right")
plt.show()
```



Observations and Insights:

- AOC (Area under the ROC Curve) is 1.00 (excellent).
- ROC curve is closer to the top-left corner which indicates a better performance.

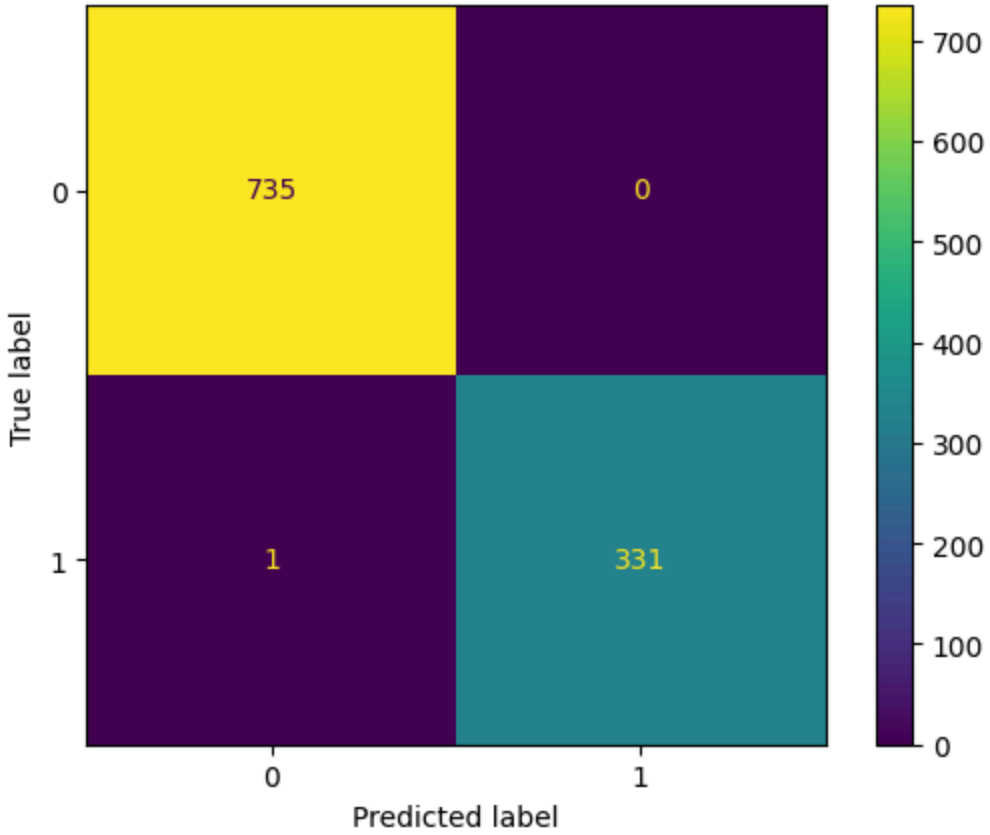
Confusion Matrix for the Training data

In [65]:
```
cm_train = confusion_matrix(y_train, ytrain_predict)
cm_train
```

Out[65]:
```
array([[735,   0],
       [  1, 331]], dtype=int64)
```

In [66]:
```
disp = ConfusionMatrixDisplay(confusion_matrix=cm_train, display_labels=RF_model.cl
disp.plot()
```

Out[66]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x21be31b7e20>
```



In [67]:
```
print(classification_report(y_train, ytrain_predict))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       735
           1       1.00      1.00      1.00       332

    accuracy                           1.00      1067
   macro avg       1.00      1.00      1.00      1067
weighted avg       1.00      1.00      1.00      1067
```

Model Performance (Random Forest Model - Training data):

For predicting party choice as Labor (Label 0):

Precision (100%) – 100% of voters predicted are actually voting the Labor party out of all voters predicted to vote Labor party.

Recall (100%) – Out of all voters actually voting the Labor party, 100% of voters are predicted to vote Labor party.

For predicting party choice as Conservative (Label 1):

Precision (100%) – 100% of voters predicted are actually voting the Conservative party out of all voters predicted to vote Conservative party.

Recall (100%) – Out of all voters actually voting the Conservative party, 100% of voters are predicted to vote Conservative party.

Overall accuracy of the model – 100% of total predictions are correct.

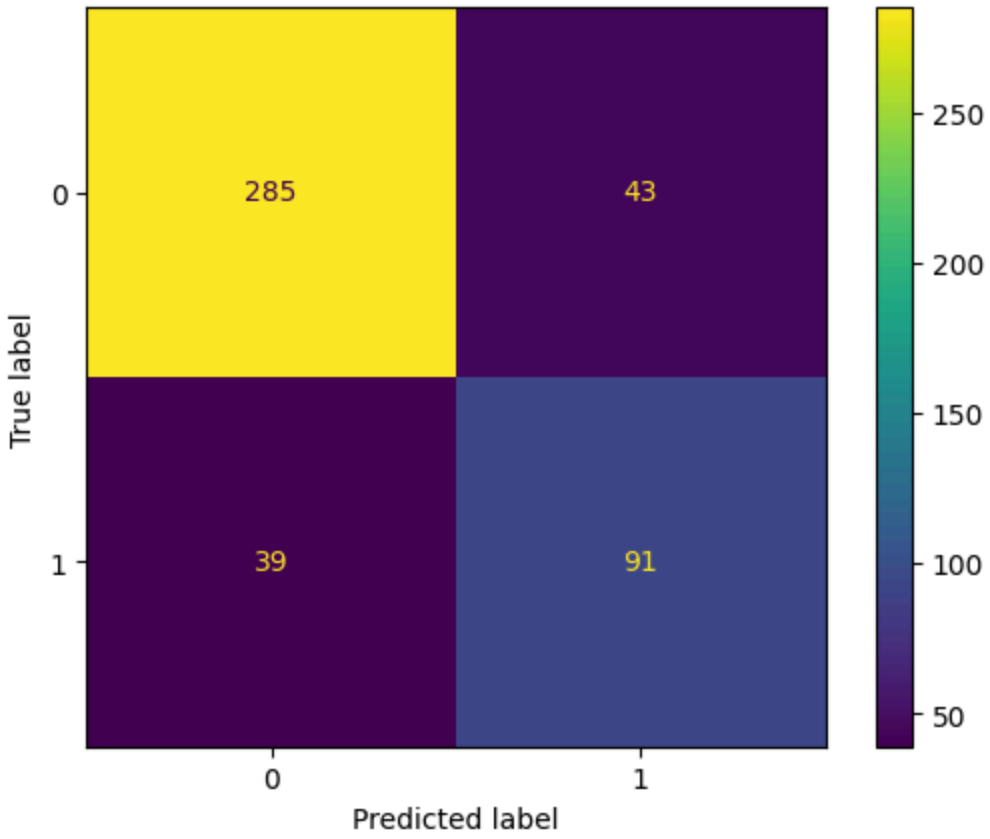
Confusion Matrix for the Test data

In [68]:
```
cm_test = confusion_matrix(y_test, ytest_predict)
cm_test
```

Out[68]:
```
array([[285,  43],
       [ 39,  91]], dtype=int64)
```

In [69]:
```
disp = ConfusionMatrixDisplay(confusion_matrix=cm_test, display_labels=RF_model.cla
disp.plot()
```

Out[69]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x21be31ff850>
```



In [70]:
```
print(classification_report(y_test, ytest_predict))
```

```
              precision    recall  f1-score   support

           0       0.88      0.87      0.87       328
           1       0.68      0.70      0.69       130

    accuracy                           0.82       458
   macro avg       0.78      0.78      0.78       458
weighted avg       0.82      0.82      0.82       458
```

Model Performance (Random Forest Model - Test data):

For predicting party choice as Labor (Label 0):

Precision (88%) – 88% of voters predicted are actually voting the Labor party out of all voters predicted to vote Labor party.

Recall (87%) – Out of all voters actually voting the Labor party, 87% of voters are predicted to vote Labor party.

For predicting party choice as Conservative (Label 1):

Precision (68%) – 68% of voters predicted are actually voting the Conservative party out of all voters predicted to vote Conservative party.

Recall (70%) – Out of all voters actually voting the Conservative party, 70% of voters are predicted to vote Conservative party.

Overall accuracy of the model – 82% of total predictions are correct.

Bagging Model

In [71]:
```
# Initialise a Bagging Classifier

BGG_model = BaggingClassifier(random_state=1)
BGG_model.fit(X_train, y_train)
```

Out[71]:
```
BaggingClassifier(random_state=1)
```

Predicting on Training and Test dataset

In [72]:
```
ytrain_predict = BGG_model.predict(X_train)
ytest_predict = BGG_model.predict(X_test)
```

Getting the Predicted Probabilities

In [73]:
```
ytest_predict_prob=BGG_model.predict_proba(X_test)
```

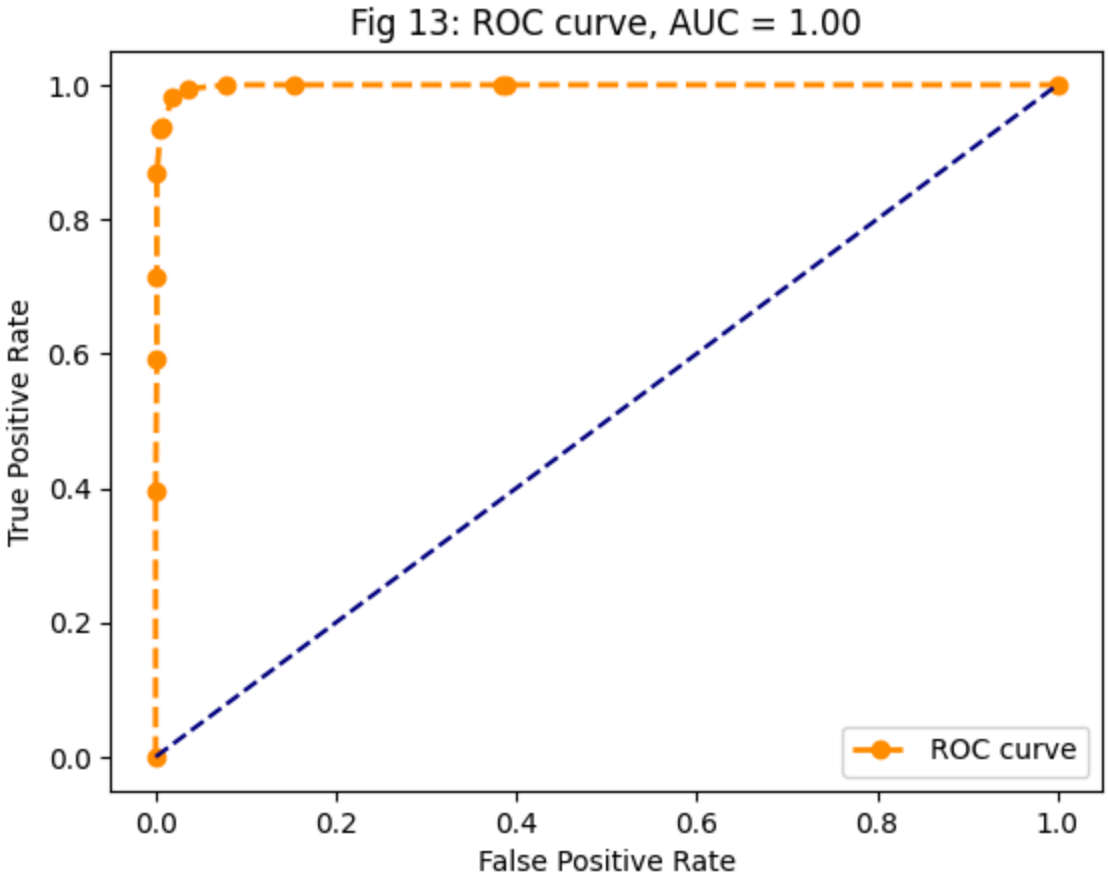
In [74]:
```
pd.DataFrame(ytest_predict_prob).head()
```

Out[74]:

	0	1
0	0.2	0.8
1	0.0	1.0
2	0.7	0.3
3	0.7	0.3
4	0.9	0.1

AUC and ROC for the Training data

In [75]:
```
# predict probabilities
probs = BGG_model.predict_proba(X_train)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# calculate AUC
auc = roc_auc_score(y_train, probs)
# calculate roc curve
train_fpr, train_tpr, train_thresholds = roc_curve(y_train, probs)
# plot the roc curve for the model
plt.plot(train_fpr, train_tpr, linestyle='--', marker='o', color='darkorange', lw =
plt.plot([0, 1], [0, 1], color='navy', linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Fig 13: ROC curve, AUC = %.2f'%auc)
plt.legend(loc="lower right")
plt.show()
```

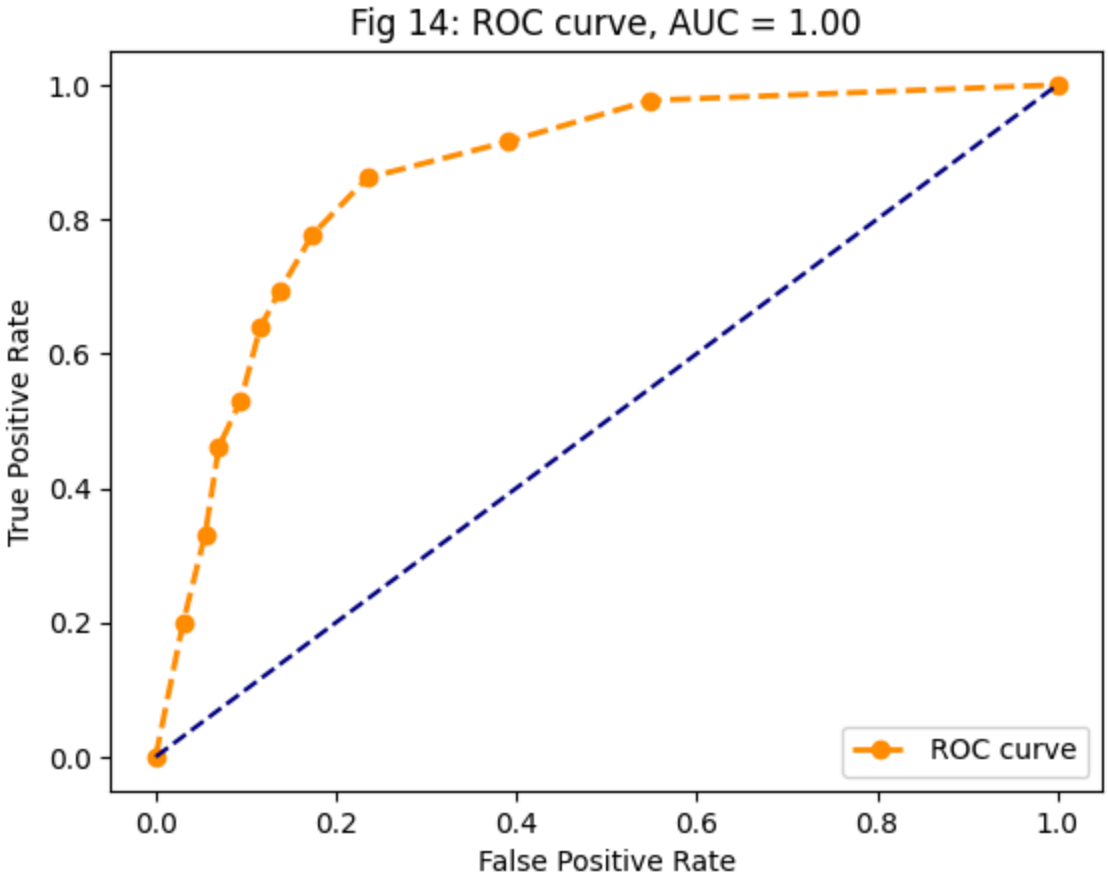


Observations and Insights:

- AOC (Area under the ROC Curve) is 1.00 (excellent).
- ROC curve is closer to the top-left corner which indicates a better performance.

AUC and ROC for the Test data

In [76]:
```
# predict probabilities
probs = BGG_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)
# calculate roc curve
test_fpr, test_tpr, test_thresholds = roc_curve(y_test, probs)
# plot the roc curve for the model
plt.plot(test_fpr, test_tpr, linestyle='--', marker='o', color='darkorange', lw = 2
plt.plot([0, 1], [0, 1], color='navy', linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Fig 14: ROC curve, AUC = %.2f'%auc)
plt.legend(loc="lower right")
plt.show()
```



Observations and Insights:

- AOC (Area under the ROC Curve) is 1.00 (excellent).
- ROC curve is closer to the top-left corner which indicates a better performance.

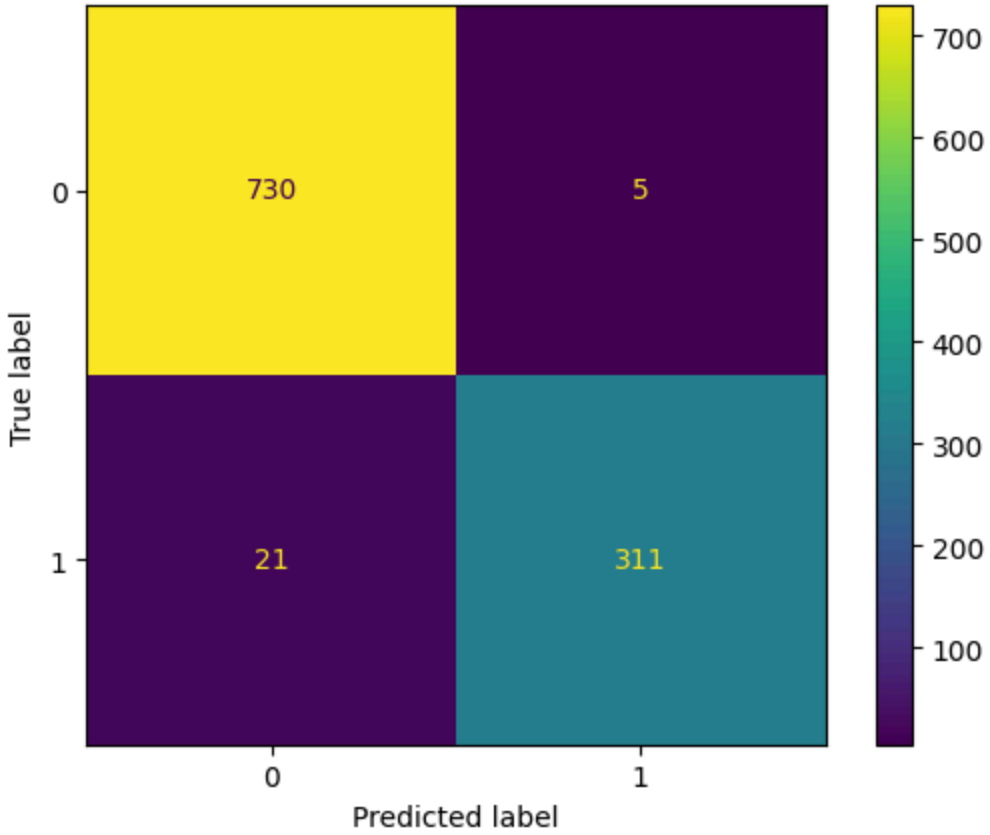
Confusion Matrix for the Training data

In [77]:
```
cm_train = confusion_matrix(y_train, ytrain_predict)
cm_train
```

Out[77]:
```
array([[730,   5],
       [ 21, 311]], dtype=int64)
```

In [78]:
```
disp = ConfusionMatrixDisplay(confusion_matrix=cm_train, display_labels=BGG_model.c
disp.plot()
```

Out[78]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x21be453eb90>
```



In [79]:
```
print(classification_report(y_train, ytrain_predict))
```

```
              precision    recall  f1-score   support

           0       0.97      0.99      0.98       735
           1       0.98      0.94      0.96       332

    accuracy                           0.98      1067
   macro avg       0.98      0.96      0.97      1067
weighted avg       0.98      0.98      0.98      1067
```

Model Performance (before tuning of Bagging Model - Training data):

For predicting party choice as Labor (Label 0):

Precision (97%) – 97% of voters predicted are actually voting the Labor party out of all voters predicted to vote Labor party.

Recall (99%) – Out of all voters actually voting the Labor party, 99% of voters are predicted to vote Labor party.

For predicting party choice as Conservative (Label 1):

Precision (98%) – 98% of voters predicted are actually voting the Conservative party out of all voters predicted to vote Conservative party.

Recall (94%) – Out of all voters actually voting the Conservative party, 94% of voters are predicted to vote Conservative party.

Overall accuracy of the model – 98% of total predictions are correct.

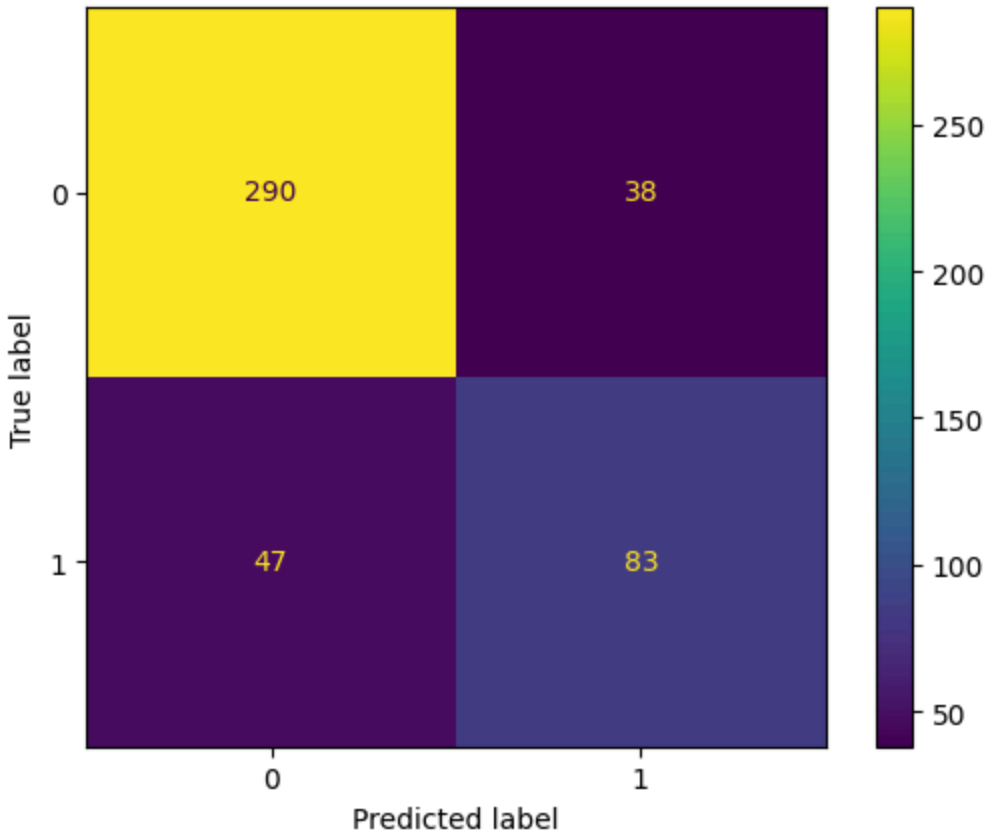
Confusion Matrix for the Test data

In [80]:
```
cm_test = confusion_matrix(y_test, ytest_predict)
cm_test
```

Out[80]:
```
array([[290,  38],
       [ 47,  83]], dtype=int64)
```

In [81]:
```
disp = ConfusionMatrixDisplay(confusion_matrix=cm_test, display_labels=BGG_model.cl
disp.plot()
```

Out[81]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x21be44001f0>
```



In [82]:
```
print(classification_report(y_test, ytest_predict))
```

	precision	recall	f1-score	support
0	0.86	0.88	0.87	328
1	0.69	0.64	0.66	130
accuracy			0.81	458
macro avg	0.77	0.76	0.77	458
weighted avg	0.81	0.81	0.81	458

Model Performance (before tuning of Bagging Model - Test data):

For predicting party choice as Labor (Label 0):

Precision (86%) – 86% of voters predicted are actually voting the Labor party out of all voters predicted to vote Labor party.

Recall (88%) – Out of all voters actually voting the Labor party, 88% of voters are predicted to vote Labor party.

For predicting party choice as Conservative (Label 1):

Precision (69%) – 69% of voters predicted are actually voting the Conservative party out of all voters predicted to vote Conservative party.

Recall (64%) – Out of all voters actually voting the Conservative party, 64% of voters are predicted to vote Conservative party.

Overall accuracy of the model – 81% of total predictions are correct.

Hyperparameter Tuning - Bagging Model

In [83]:
```
# Choose the type of classifier.
BGG_model_tuned = BaggingClassifier(random_state=1)

# Grid of parameters to choose from
parameters = {
    'n_estimators': [10, 50, 100],
    'max_samples': [0.5, 1.0, 1.5],
    'bootstrap': [True, False],
    'bootstrap_features': [True, False]
}

# Run the grid search
grid_obj = GridSearchCV(BGG_model_tuned, parameters, cv=5)
grid_obj = grid_obj.fit(X_train, y_train)

# Set the clf to the best combination of parameters
BGG_model_tuned = grid_obj.best_estimator_

# Fit the best algorithm to the data.
BGG_model_tuned.fit(X_train, y_train)
```

Out[83]:

BaggingClassifier

BaggingClassifier(max_samples=0.5, n_estimators=100, random_state=1)

Predicting on Training and Test dataset

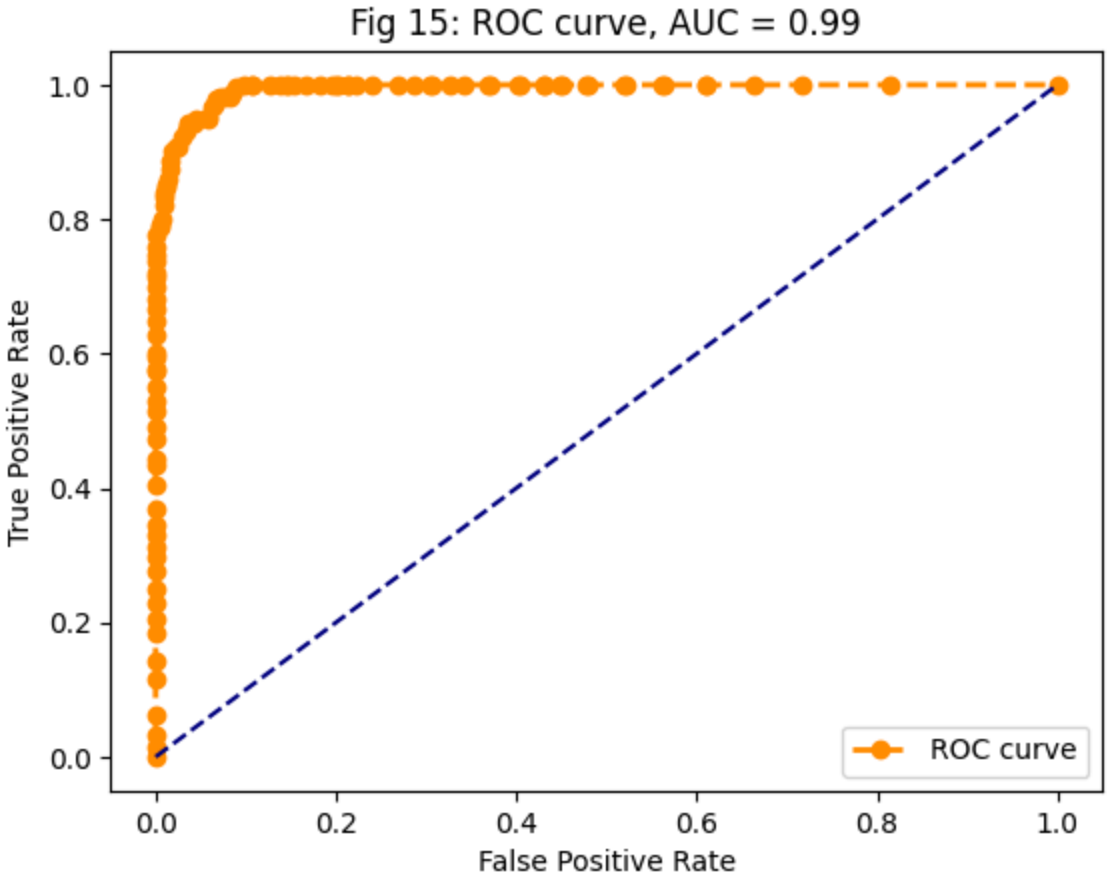
In [84]:

```
ytrain_predict = BGG_model_tuned.predict(X_train)
ytest_predict = BGG_model_tuned.predict(X_test)
```

AUC and ROC for the Training and Test data

In [85]:

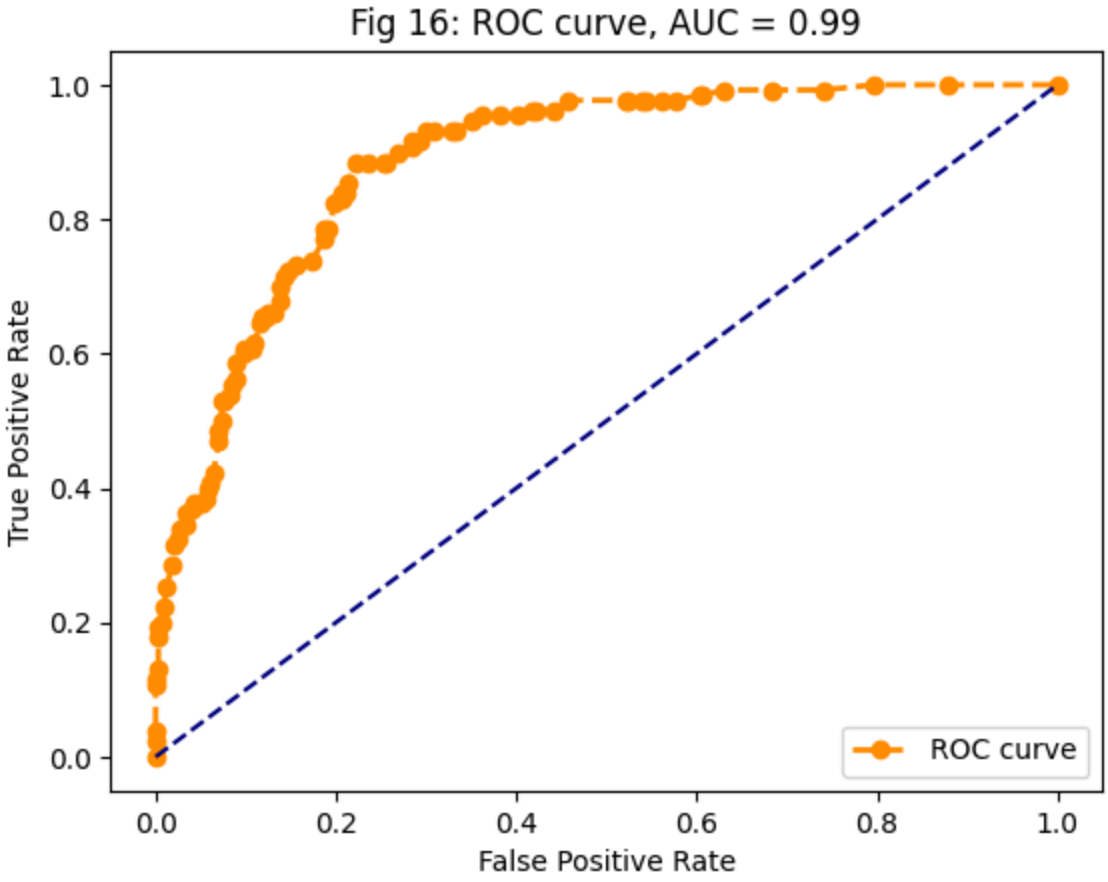
```
# predict probabilities
probs = BGG_model_tuned.predict_proba(X_train)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# calculate AUC
auc = roc_auc_score(y_train, probs)
# calculate roc curve
train_fpr, train_tpr, train_thresholds = roc_curve(y_train, probs)
# plot the roc curve for the model
plt.plot(train_fpr, train_tpr, linestyle='--', marker='o', color='darkorange', lw =
plt.plot([0, 1], [0, 1], color='navy', linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Fig 15: ROC curve, AUC = %.2f'%auc)
plt.legend(loc="lower right")
plt.show()
```



Observations and Insights:

- AOC (Area under the ROC Curve) is 0.99 (excellent).
- ROC curve is closer to the top-left corner which indicates a better performance.

In [86]:
```
# predict probabilities
probs = BGG_model_tuned.predict_proba(X_test)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# calculate AUC
test_auc = roc_auc_score(y_test, probs)
# calculate roc curve
test_fpr, test_tpr, test_thresholds = roc_curve(y_test, probs)
# plot the roc curve for the model
plt.plot(test_fpr, test_tpr, linestyle='--', marker='o', color='darkorange', lw = 2
plt.plot([0, 1], [0, 1], color='navy', linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Fig 16: ROC curve, AUC = %.2f'%auc)
plt.legend(loc="lower right")
plt.show()
```



Observations and Insights:

- AOC (Area under the ROC Curve) is 0.99 (excellent).
- ROC curve is closer to the top-left corner which indicates a better performance.

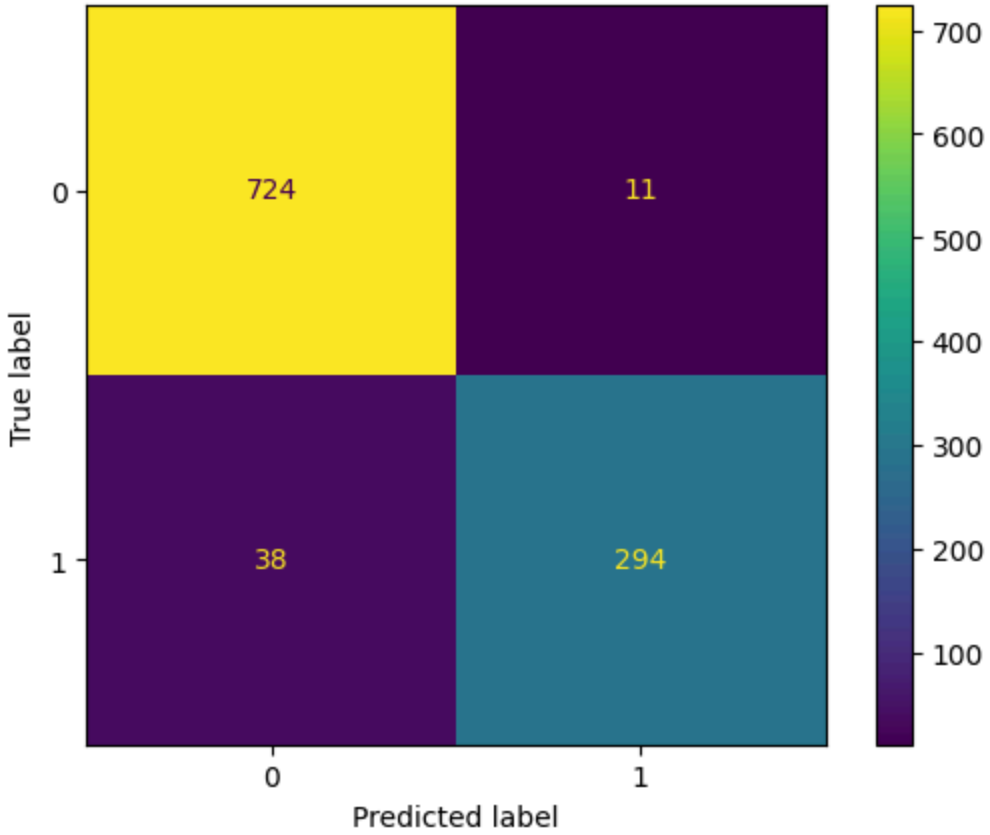
Confusion Matrix for the Training and Test data

In [87]:
```
cm_train_tuned = confusion_matrix(y_train, ytrain_predict)
cm_train_tuned
```

Out[87]:
```
array([[724,  11],
       [ 38, 294]], dtype=int64)
```

In [88]:
```
disp = ConfusionMatrixDisplay(confusion_matrix=cm_train_tuned, display_labels=BGG_m
disp.plot()
```

Out[88]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x21be4606f50>
```



In [89]:
```
print(classification_report(y_train, ytrain_predict))
```

```
              precision    recall  f1-score   support

           0       0.95      0.99      0.97       735
           1       0.96      0.89      0.92       332

    accuracy                           0.95      1067
   macro avg       0.96      0.94      0.95      1067
weighted avg       0.95      0.95      0.95      1067
```

Model Performance (after tuning of Bagging Model - Training data):

For predicting party choice as Labor (Label 0):

Precision (95%) – 95% of voters predicted are actually voting the Labor party out of all voters predicted to vote Labor party.

Recall (99%) – Out of all voters actually voting the Labor party, 99% of voters are predicted to vote Labor party.

For predicting party choice as Conservative (Label 1):

Precision (96%) – 96% of voters predicted are actually voting the Conservative party out of all voters predicted to vote Conservative party.

Recall (89%) – Out of all voters actually voting the Conservative party, 89% of voters are predicted to vote Conservative party.

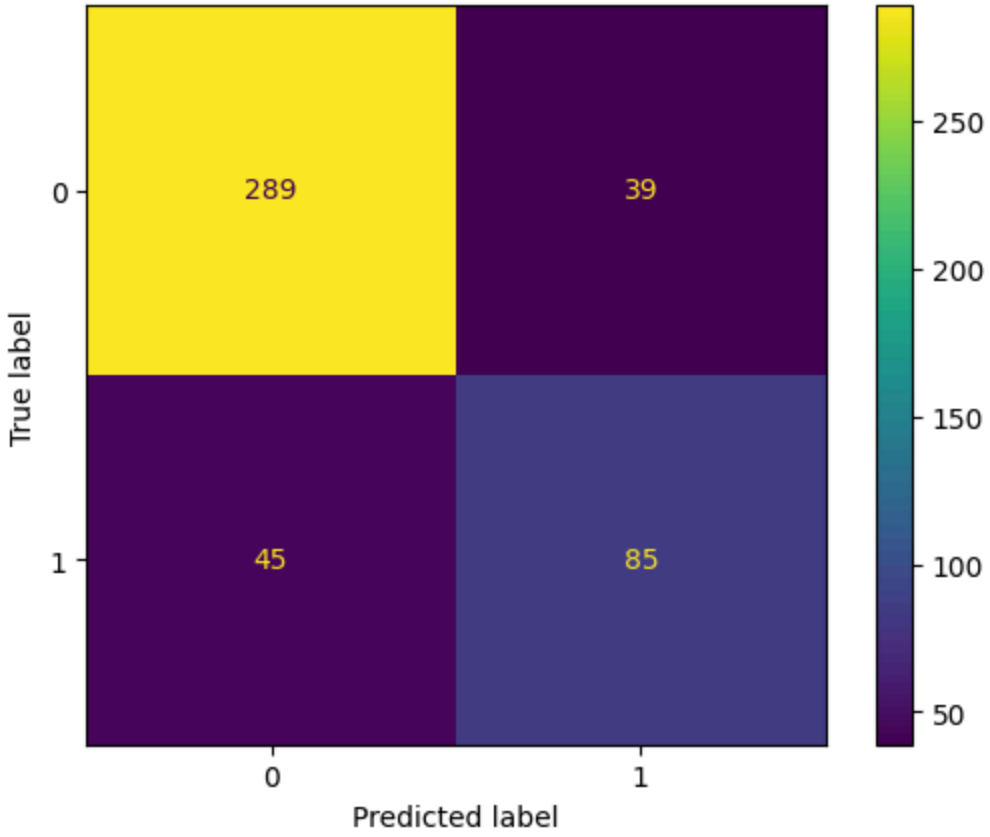
Overall accuracy of the model – 95% of total predictions are correct.

In [90]:
```
cm_test_tuned = confusion_matrix(y_test, ytest_predict)
cm_test_tuned
```

Out[90]:
```
array([[289,  39],
       [ 45,  85]], dtype=int64)
```

In [91]:
```
disp = ConfusionMatrixDisplay(confusion_matrix=cm_test_tuned, display_labels=BGG_mo
disp.plot()
```

Out[91]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x21be47851b0>
```



In [92]:
```
print(classification_report(y_test, ytest_predict))
```

```
              precision    recall  f1-score   support

           0       0.87      0.88      0.87       328
           1       0.69      0.65      0.67       130

    accuracy                           0.82       458
   macro avg       0.78      0.77      0.77       458
weighted avg       0.81      0.82      0.82       458
```

Model Performance (after tuning of Bagging Model - Test data):

For predicting party choice as Labor (Label 0):

Precision (87%) – 87% of voters predicted are actually voting the Labor party out of all voters predicted to vote Labor party.

Recall (88%) – Out of all voters actually voting the Labor party, 88% of voters are predicted to vote Labor party.

For predicting party choice as Conservative (Label 1):

Precision (69%) – 69% of voters predicted are actually voting the Conservative party out of all voters predicted to vote Conservative party.

Recall (65%) – Out of all voters actually voting the Conservative party, 65% of voters are predicted to vote Conservative party.

Overall accuracy of the model – 82% of total predictions are correct. Overall accuracy of model is increased by 1% after tuning the Bagging Model.

Boosting Models

AdaBoost Model

In [93]:
```
# Initialise AdaBoost Classifier

BST_model = AdaBoostClassifier(random_state=1)
BST_model.fit(X_train, y_train)
```

Out[93]:
```
AdaBoostClassifier(random_state=1)
```

Predicting on Training and Test dataset

In [94]:
```
ytrain_predict = BST_model.predict(X_train)
ytest_predict = BST_model.predict(X_test)
```

Getting the Predicted Probabilities

In [95]:
```
ytest_predict_prob=BST_model.predict_proba(X_test)
```

In [96]:
```
pd.DataFrame(ytest_predict_prob).head() # Returns first 5 rows
```

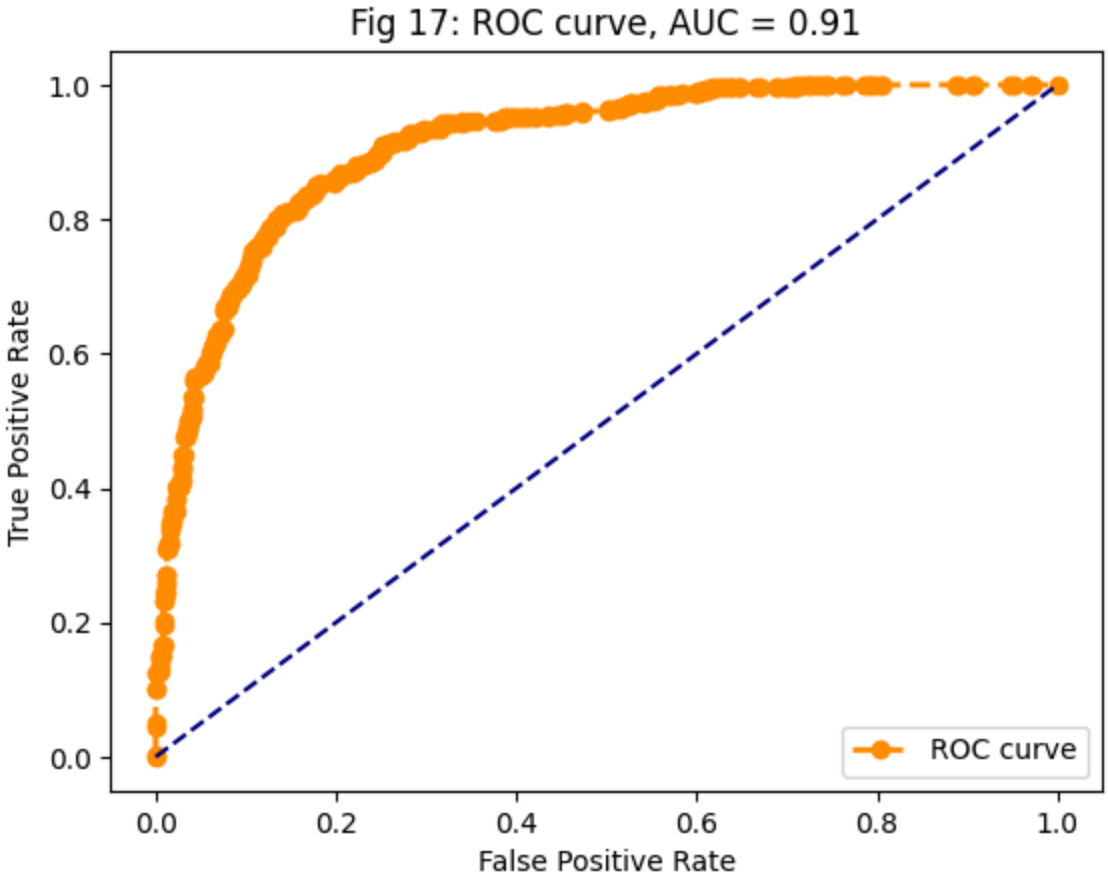
Out[96]:

	0	1
0	0.491090	0.508910
1	0.493893	0.506107
2	0.501883	0.498117
3	0.497560	0.502440
4	0.507384	0.492616

AUC and ROC for the Training data

In [97]:

```
# predict probabilities
probs = BST_model.predict_proba(X_train)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# calculate AUC
auc = roc_auc_score(y_train, probs)
# calculate roc curve
train_fpr, train_tpr, train_thresholds = roc_curve(y_train, probs)
# plot the roc curve for the model
plt.plot(train_fpr, train_tpr, linestyle='--', marker='o', color='darkorange', lw =
plt.plot([0, 1], [0, 1], color='navy', linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Fig 17: ROC curve, AUC = %.2f'%auc)
plt.legend(loc="lower right")
plt.show()
```

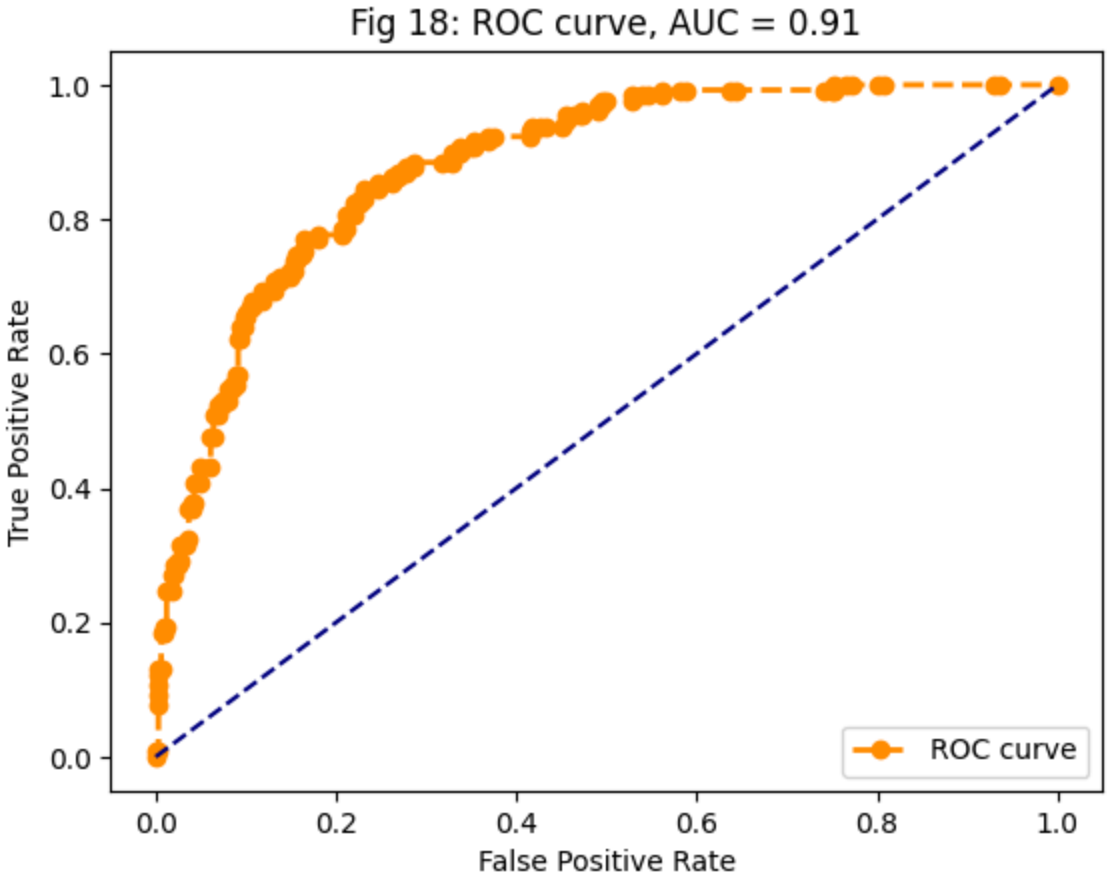


Observations and Insights:

- AOC (Area under the ROC Curve) is 0.91 (excellent).
- ROC curve is closer to the top-left corner which indicates a better performance.

AUC and ROC for the Test data

In [98]:
```
# predict probabilities
probs = BST_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)
# calculate roc curve
test_fpr, test_tpr, test_thresholds = roc_curve(y_test, probs)
# plot the roc curve for the model
plt.plot(test_fpr, test_tpr, linestyle='--', marker='o', color='darkorange', lw = 2
plt.plot([0, 1], [0, 1], color='navy', linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Fig 18: ROC curve, AUC = %.2f'%auc)
plt.legend(loc="lower right")
plt.show()
```



Observations and Insights:

- AOC (Area under the ROC Curve) is 0.91 (excellent).
- ROC curve is closer to the top-left corner which indicates a better performance.

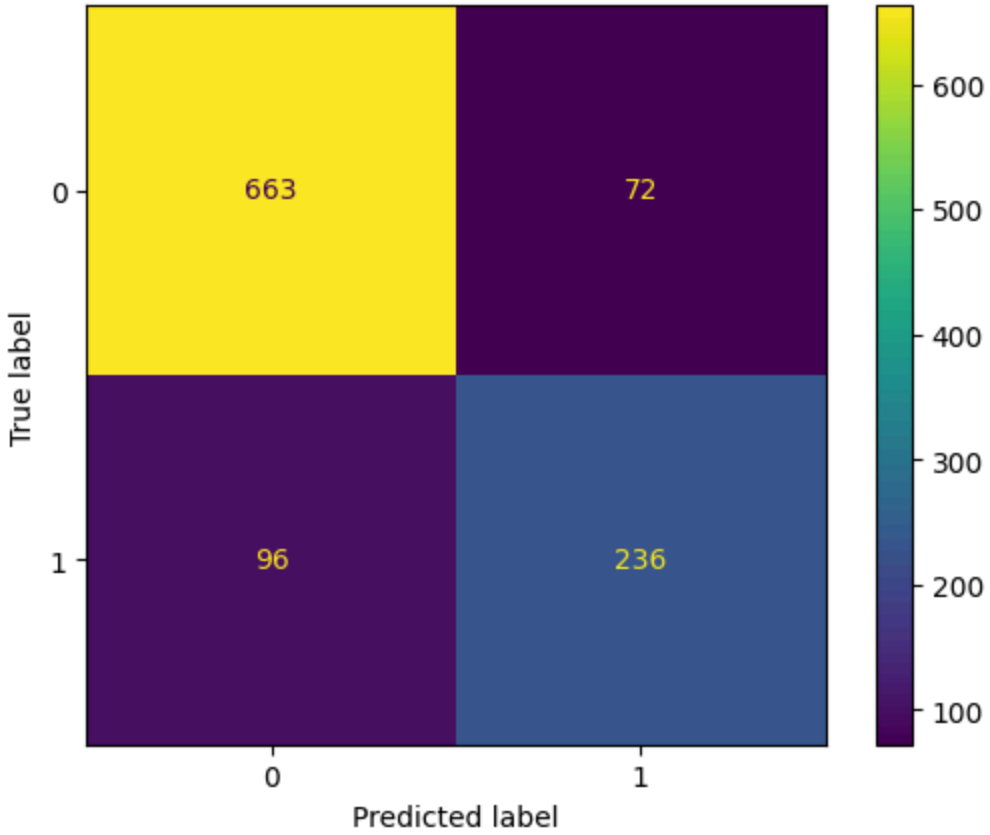
Confusion Matrix for the Training data

In [99]:
```
cm_train = confusion_matrix(y_train, ytrain_predict)
cm_train
```

Out[99]:
```
array([[663,  72],
       [ 96, 236]], dtype=int64)
```

In [100…
```
disp = ConfusionMatrixDisplay(confusion_matrix=cm_train, display_labels=BST_model.c
disp.plot()
```

Out[100…
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x21be5a9b5b0>
```



In [101…

```
print(classification_report(y_train, ytrain_predict))
```

```
              precision    recall  f1-score   support

           0       0.87      0.90      0.89       735
           1       0.77      0.71      0.74       332

    accuracy                           0.84      1067
   macro avg       0.82      0.81      0.81      1067
weighted avg       0.84      0.84      0.84      1067
```

Model Performance (before tuning of AdaBoost Model - Training data):

For predicting party choice as Labor (Label 0):

Precision (87%) – 87% of voters predicted are actually voting the Labor party out of all voters predicted to vote Labor party.

Recall (90%) – Out of all voters actually voting the Labor party, 90% of voters are predicted to vote Labor party.

For predicting party choice as Conservative (Label 1):

Precision (77%) – 77% of voters predicted are actually voting the Conservative party out of all voters predicted to vote Conservative party.

Recall (71%) – Out of all voters actually voting the Conservative party, 71% of voters are predicted to vote Conservative party.

Overall accuracy of the model – 84% of total predictions are correct.

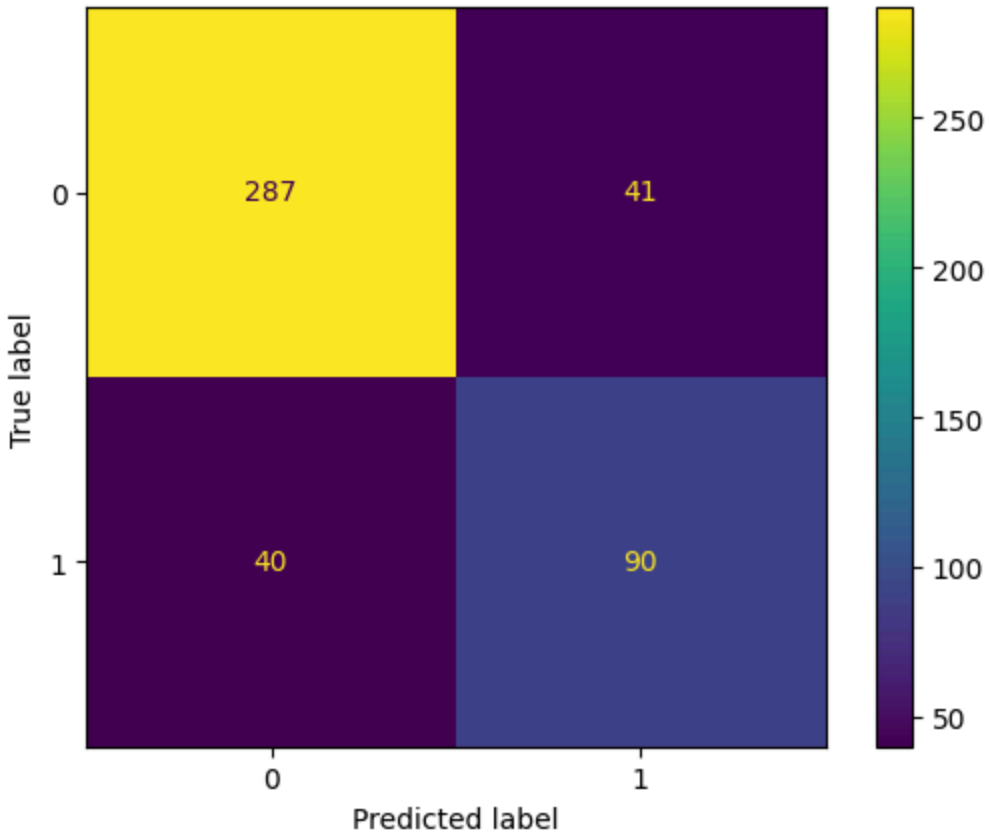
Confusion Matrix for the Test data

In [102…
```
cm_test = confusion_matrix(y_test, ytest_predict)
cm_test
```

Out[102…
```
array([[287,  41],
       [ 40,  90]], dtype=int64)
```

In [103…
```
disp = ConfusionMatrixDisplay(confusion_matrix=cm_test, display_labels=BST_model.cl
disp.plot()
```

Out[103…
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x21be4787bb0>
```



In [104…
```
print(classification_report(y_test, ytest_predict))
```

```
              precision    recall  f1-score   support

           0       0.88      0.88      0.88       328
           1       0.69      0.69      0.69       130

    accuracy                           0.82       458
   macro avg       0.78      0.78      0.78       458
weighted avg       0.82      0.82      0.82       458
```

Model Performance (before tuning of AdaBoost Model - Test data):

For predicting party choice as Labor (Label 0):

Precision (88%) – 88% of voters predicted are actually voting the Labor party out of all voters predicted to vote Labor party.

Recall (88%) – Out of all voters actually voting the Labor party, 88% of voters are predicted to vote Labor party.

For predicting party choice as Conservative (Label 1):

Precision (69%) – 69% of voters predicted are actually voting the Conservative party out of all voters predicted to vote Conservative party.

Recall (69%) – Out of all voters actually voting the Conservative party, 69% of voters are predicted to vote Conservative party.

Overall accuracy of the model – 82% of total predictions are correct.

Hyperparameter Tuning - AdaBoost Model

In [105…

```
# Choose the type of classifier.
BST_model_tuned = AdaBoostClassifier(random_state=1)

# Grid of parameters to choose from
parameters = {
    'n_estimators': [10, 50, 100],
    'learning_rate': [0.01, 0.1, 1.0]
}

# Run the grid search
grid_obj = GridSearchCV(BST_model_tuned, parameters, cv=5)
grid_obj = grid_obj.fit(X_train, y_train)

# Set the clf to the best combination of parameters
BST_model_tuned = grid_obj.best_estimator_

# Fit the best algorithm to the data.
BST_model_tuned.fit(X_train, y_train)
```

Out[105…

AdaBoostClassifier

```
AdaBoostClassifier(learning_rate=0.1, n_estimators=100, random_state=1)
```

Predicting on Training and Test dataset

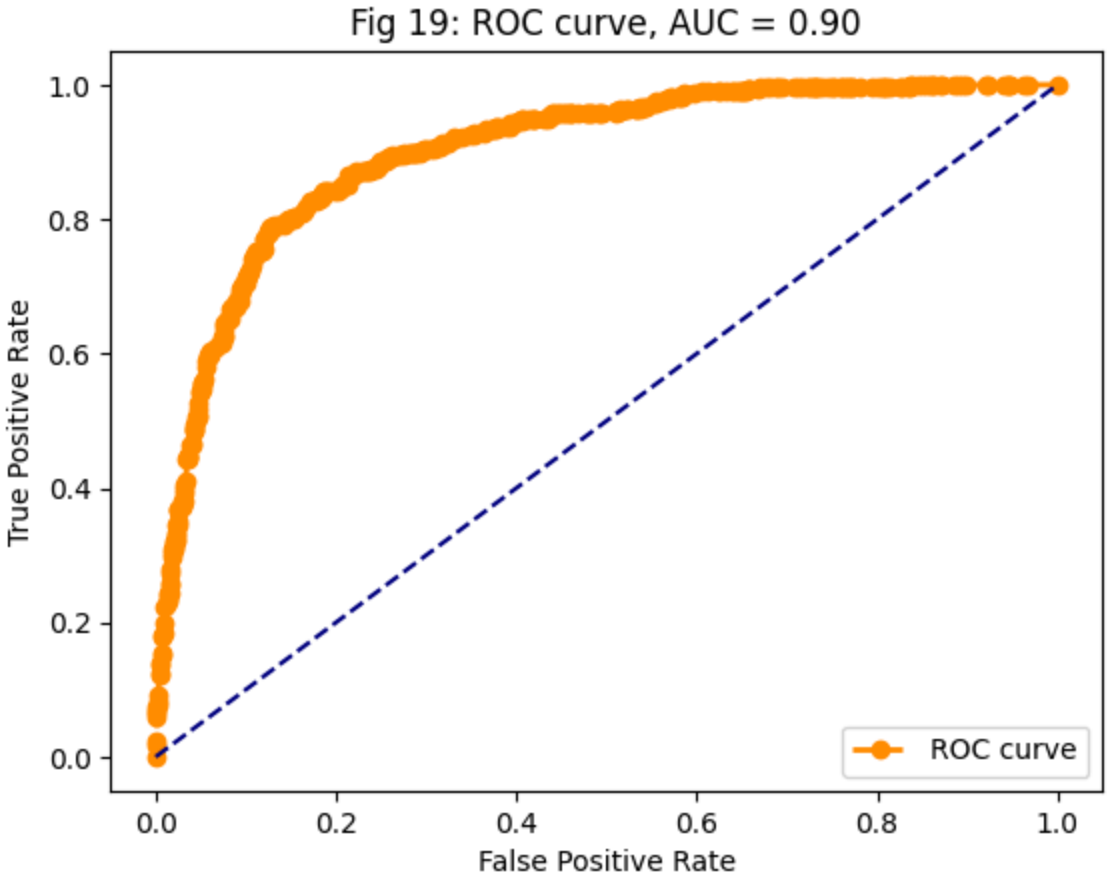
In [106…

```
ytrain_predict = BST_model_tuned.predict(X_train)
ytest_predict = BST_model_tuned.predict(X_test)
```

AUC and ROC for the Training and Test data

In [107…

```
# predict probabilities
probs = BST_model_tuned.predict_proba(X_train)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# calculate AUC
auc = roc_auc_score(y_train, probs)
# calculate roc curve
train_fpr, train_tpr, train_thresholds = roc_curve(y_train, probs)
# plot the roc curve for the model
plt.plot(train_fpr, train_tpr, linestyle='--', marker='o', color='darkorange', lw =
plt.plot([0, 1], [0, 1], color='navy', linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Fig 19: ROC curve, AUC = %.2f'%auc)
plt.legend(loc="lower right")
plt.show()
```

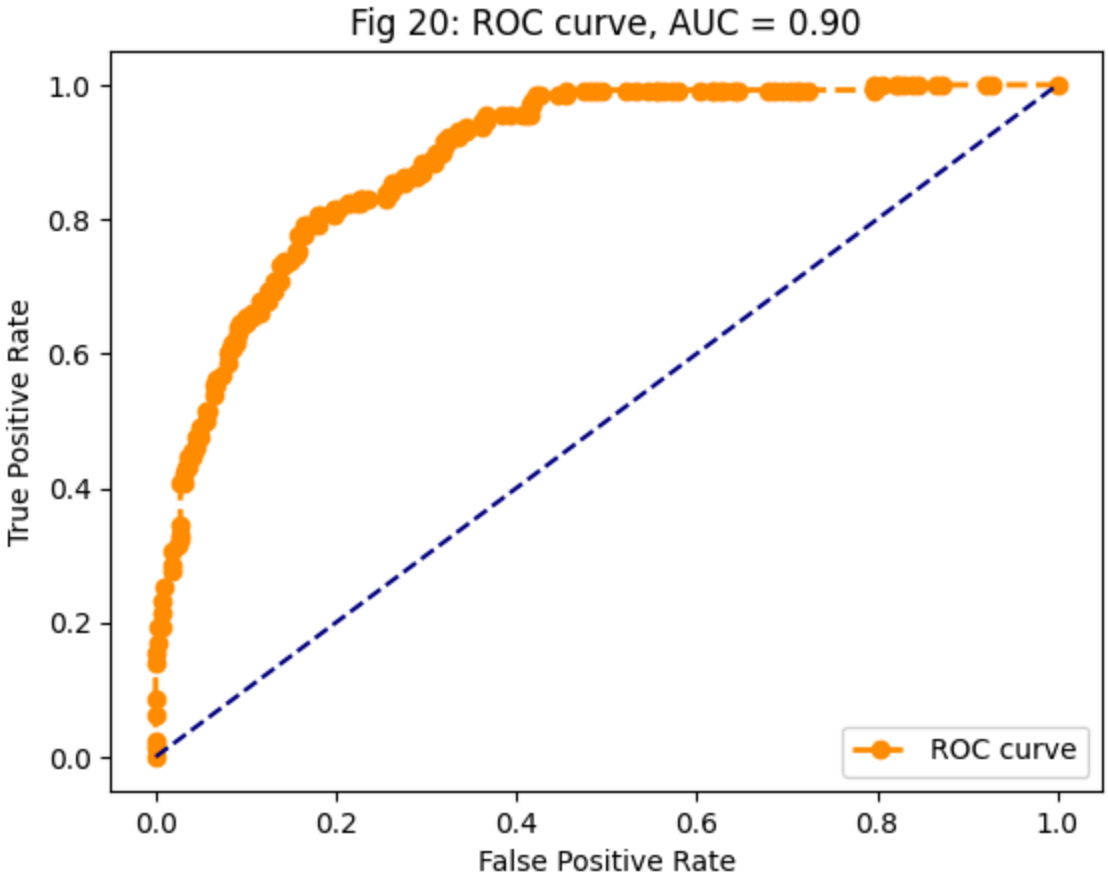


Observations and Insights:

- AOC (Area under the ROC Curve) is 0.90 (good).
- ROC curve is closer to the top-left corner which indicates a better performance.

In [108…

```
# predict probabilities
probs = BST_model_tuned.predict_proba(X_test)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# calculate AUC
test_auc = roc_auc_score(y_test, probs)
# calculate roc curve
test_fpr, test_tpr, test_thresholds = roc_curve(y_test, probs)
# plot the roc curve for the model
plt.plot(test_fpr, test_tpr, linestyle='--', marker='o', color='darkorange', lw = 2
plt.plot([0, 1], [0, 1], color='navy', linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Fig 20: ROC curve, AUC = %.2f'%auc)
plt.legend(loc="lower right")
plt.show()
```



Observations and Insights:

- AOC (Area under the ROC Curve) is 0.90 (good).
- ROC curve is closer to the top-left corner which indicates a better performance.

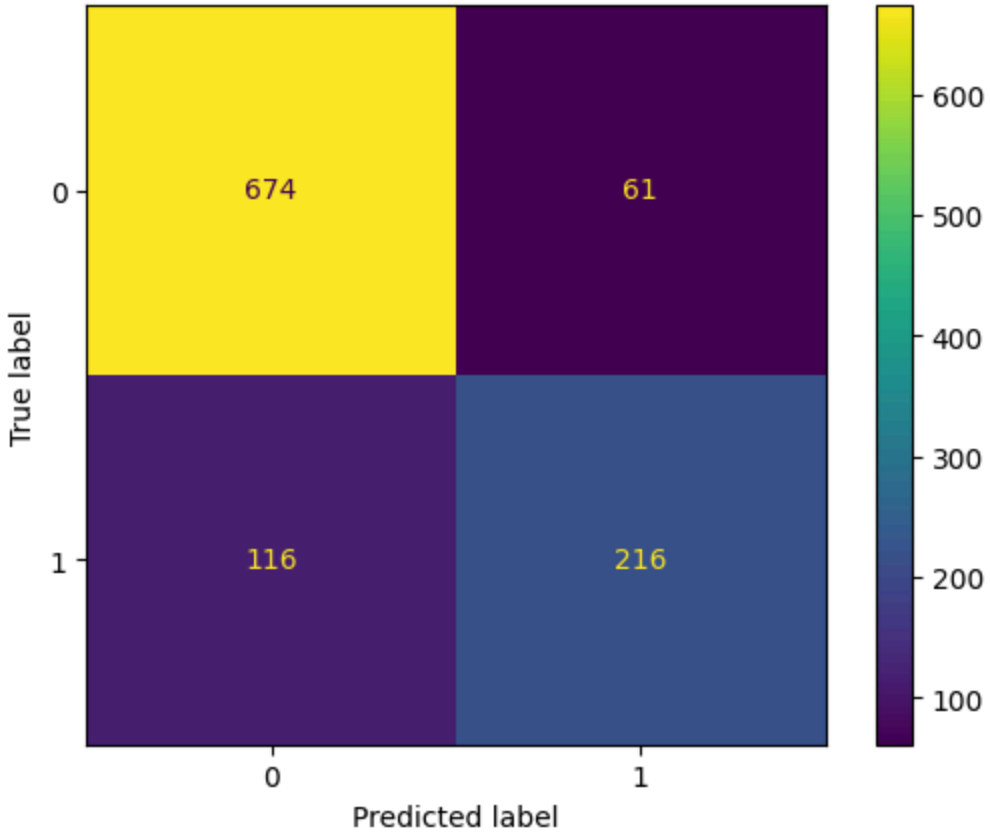
Confusion Matrix for the Training and Test data

In [109…
```
cm_train_tuned = confusion_matrix(y_train, ytrain_predict)
cm_train_tuned
```

Out[109…
```
array([[674,  61],
       [116, 216]], dtype=int64)
```

In [110…
```
disp = ConfusionMatrixDisplay(confusion_matrix=cm_train_tuned, display_labels=BST_m
disp.plot()
```

Out[110…
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x21be5a9aa70>
```



In [111…

```
print(classification_report(y_train, ytrain_predict))
```

```
              precision    recall  f1-score   support

           0       0.85      0.92      0.88       735
           1       0.78      0.65      0.71       332

    accuracy                           0.83      1067
   macro avg       0.82      0.78      0.80      1067
weighted avg       0.83      0.83      0.83      1067
```

Model Performance (after tuning of AdaBoost Model - Training data):

For predicting party choice as Labor (Label 0):

Precision (85%) – 85% of voters predicted are actually voting the Labor party out of all voters predicted to vote Labor party.

Recall (92%) – Out of all voters actually voting the Labor party, 92% of voters are predicted to vote Labor party.

For predicting party choice as Conservative (Label 1):

Precision (78%) – 78% of voters predicted are actually voting the Conservative party out of all voters predicted to vote Conservative party.

Recall (65%) – Out of all voters actually voting the Conservative party, 65% of voters are predicted to vote Conservative party.

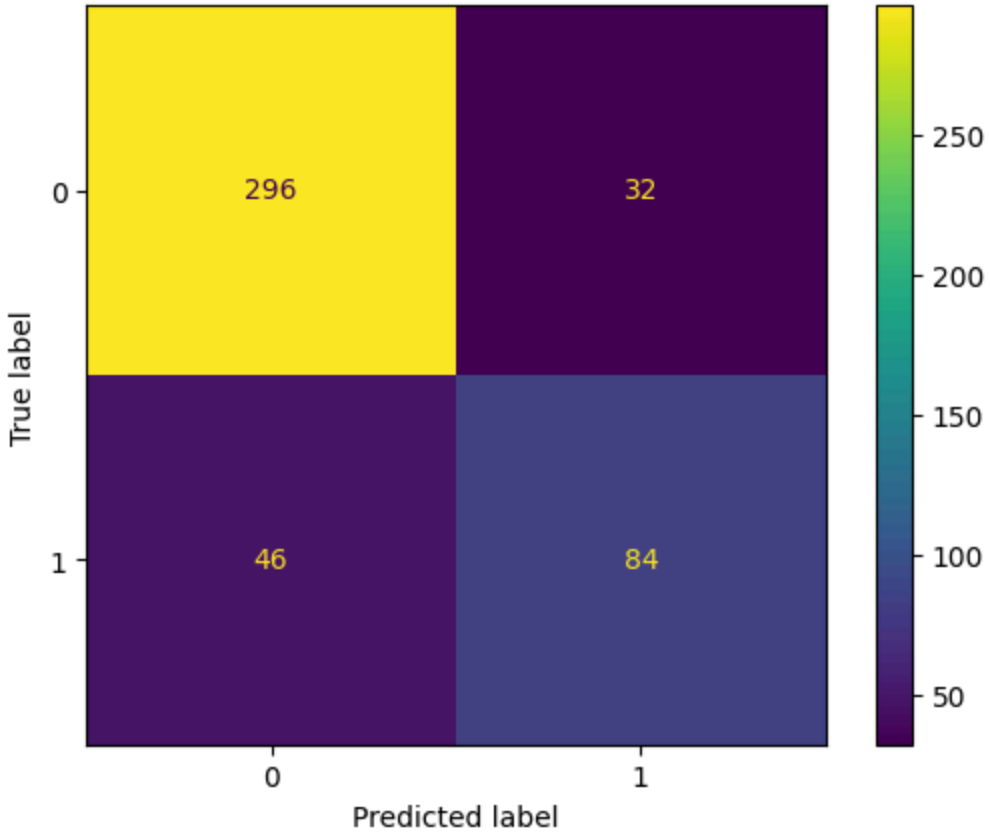
Overall accuracy of the model – 83% of total predictions are correct.

In [112…
```
cm_test_tuned = confusion_matrix(y_test, ytest_predict)
cm_test_tuned
```

Out[112…
```
array([[296,  32],
       [ 46,  84]], dtype=int64)
```

In [113…
```
disp = ConfusionMatrixDisplay(confusion_matrix=cm_test_tuned, display_labels=BST_mo
disp.plot()
```

Out[113…
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x21be5a99450>
```



In [114…
```
print(classification_report(y_test, ytest_predict))
```

```
              precision    recall  f1-score   support

           0       0.87      0.90      0.88       328
           1       0.72      0.65      0.68       130

    accuracy                           0.83       458
   macro avg       0.79      0.77      0.78       458
weighted avg       0.83      0.83      0.83       458
```

Model Performance (after tuning of AdaBoost Model - Test data):

For predicting party choice as Labor (Label 0):

Precision (87%) – 87% of voters predicted are actually voting the Labor party out of all voters predicted to vote Labor party.

Recall (90%) – Out of all voters actually voting the Labor party, 90% of voters are predicted to vote Labor party.

For predicting party choice as Conservative (Label 1):

Precision (72%) – 72% of voters predicted are actually voting the Conservative party out of all voters predicted to vote Conservative party.

Recall (65%) – Out of all voters actually voting the Conservative party, 65% of voters are predicted to vote Conservative party.

Overall accuracy of the model – 83% of total predictions are correct. Overall accuracy of model is increased by 1% after tuning the AdaBoost Model.

Gradient Boosting Model

In [115…
```
# Initialise Gradient Boosting Classifier

BGT_model = GradientBoostingClassifier(random_state=1)
BGT_model.fit(X_train, y_train)
```

Out[115…
```
GradientBoostingClassifier
GradientBoostingClassifier(random_state=1)
```

Predicting on Training and Test dataset

In [116…
```
ytrain_predict = BGT_model.predict(X_train)
ytest_predict = BGT_model.predict(X_test)
```

Getting the Predicted Probabilities

In [117…
```
ytest_predict_prob=BGT_model.predict_proba(X_test)
```

In [118…
```
pd.DataFrame(ytest_predict_prob).head() # Returns first 5 rows
```

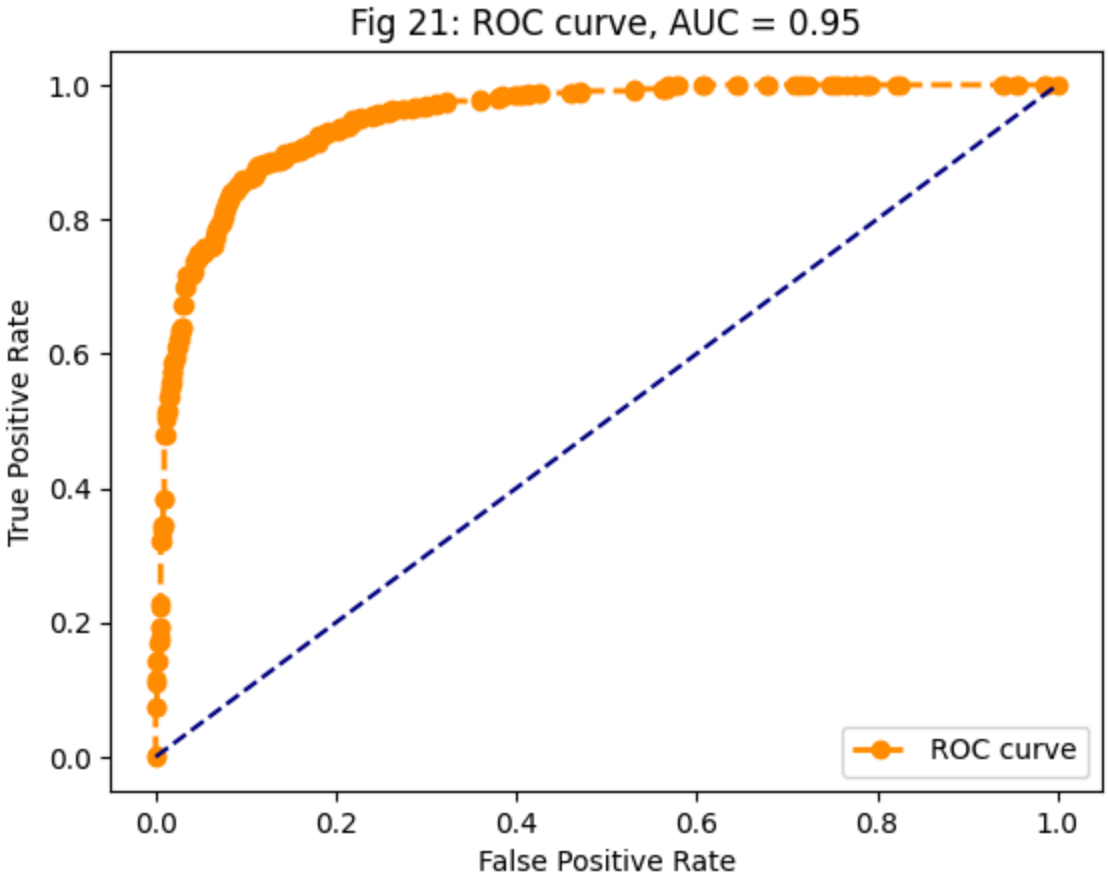
Out[118…

	0	1
0	0.123040	0.876960
1	0.160127	0.839873
2	0.827060	0.172940
3	0.631505	0.368495
4	0.765814	0.234186

AUC and ROC for the Training data

In [119…

```
# predict probabilities
probs = BGT_model.predict_proba(X_train)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# calculate AUC
auc = roc_auc_score(y_train, probs)
# calculate roc curve
train_fpr, train_tpr, train_thresholds = roc_curve(y_train, probs)
# plot the roc curve for the model
plt.plot(train_fpr, train_tpr, linestyle='--', marker='o', color='darkorange', lw =
plt.plot([0, 1], [0, 1], color='navy', linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Fig 21: ROC curve, AUC = %.2f'%auc)
plt.legend(loc="lower right")
plt.show()
```



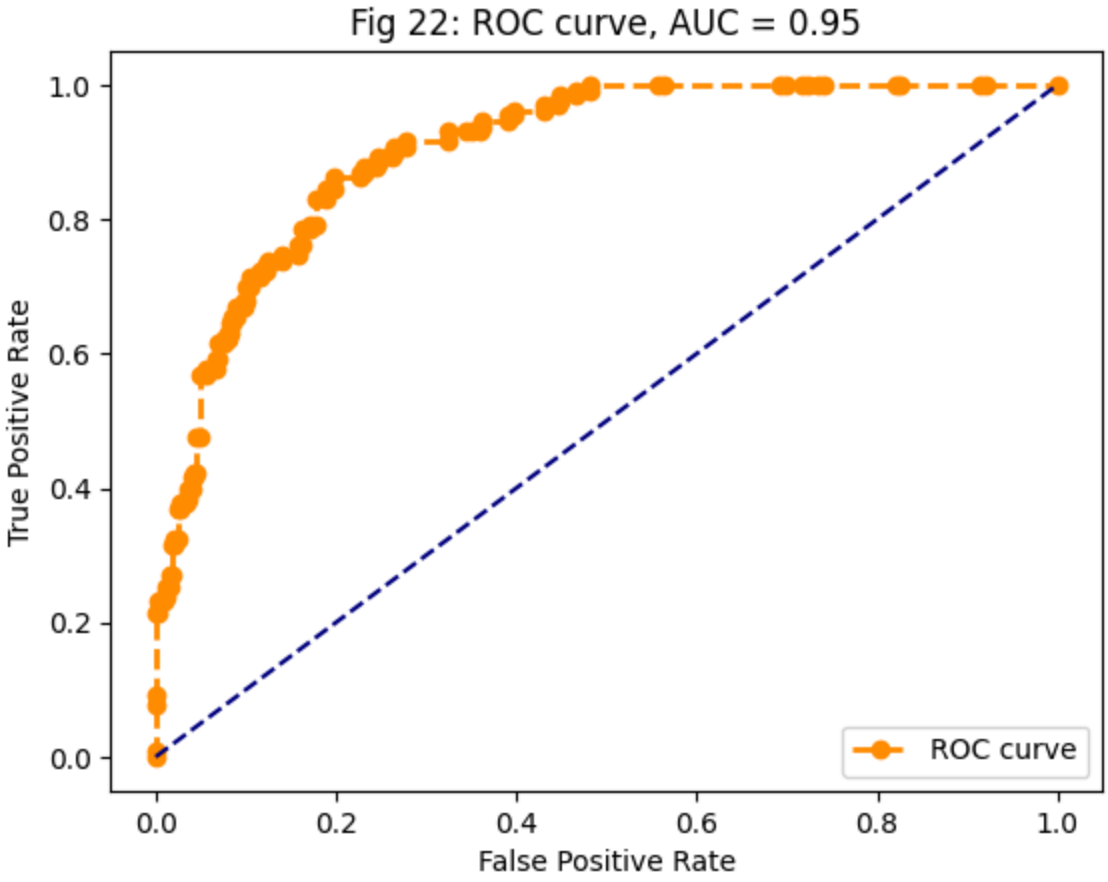
Observations and Insights:

- AOC (Area under the ROC Curve) is 0.95 (excellent).
- ROC curve is closer to the top-left corner which indicates a better performance.

AUC and ROC for the Test data

In [120…

```
# predict probabilities
probs = BGT_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)
# calculate roc curve
test_fpr, test_tpr, test_thresholds = roc_curve(y_test, probs)
# plot the roc curve for the model
plt.plot(test_fpr, test_tpr, linestyle='--', marker='o', color='darkorange', lw = 2
plt.plot([0, 1], [0, 1], color='navy', linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Fig 22: ROC curve, AUC = %.2f'%auc)
plt.legend(loc="lower right")
plt.show()
```



Observations and Insights:

- AOC (Area under the ROC Curve) is 0.95 (excellent).
- ROC curve is closer to the top-left corner which indicates a better performance.

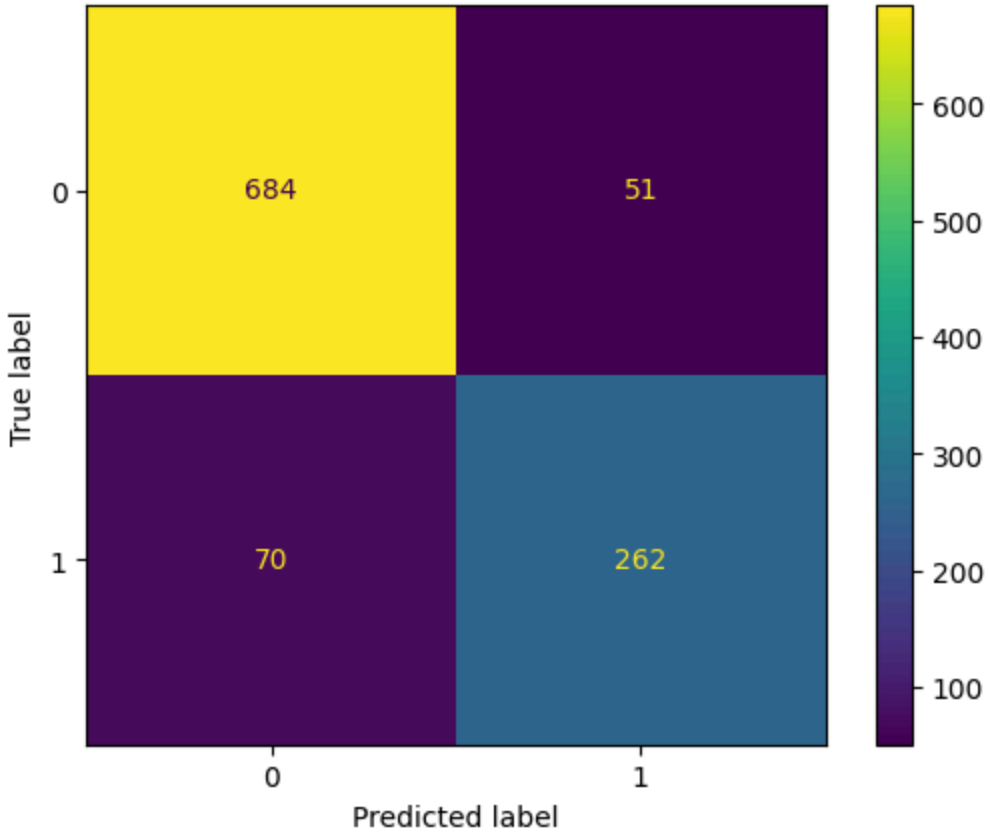
Confusion Matrix for the Training data

In [121…
```
cm_train = confusion_matrix(y_train, ytrain_predict)
cm_train
```

Out[121…
```
array([[684,  51],
       [ 70, 262]], dtype=int64)
```

In [122…
```
disp = ConfusionMatrixDisplay(confusion_matrix=cm_train, display_labels=BGT_model.c
disp.plot()
```

Out[122…
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x21be2fa0a90>
```



In [123…

```
print(classification_report(y_train, ytrain_predict))
```

```
              precision    recall  f1-score   support

           0       0.91      0.93      0.92       735
           1       0.84      0.79      0.81       332

    accuracy                           0.89      1067
   macro avg       0.87      0.86      0.87      1067
weighted avg       0.89      0.89      0.89      1067
```

Model Performance (before tuning of Gradient Boosting Model - Training data):

For predicting party choice as Labor (Label 0):

Precision (91%) – 91% of voters predicted are actually voting the Labor party out of all voters predicted to vote Labor party.

Recall (93%) – Out of all voters actually voting the Labor party, 93% of voters are predicted to vote Labor party.

For predicting party choice as Conservative (Label 1):

Precision (84%) – 84% of voters predicted are actually voting the Conservative party out of all voters predicted to vote Conservative party.

Recall (79%) – Out of all voters actually voting the Conservative party, 79% of voters are predicted to vote Conservative party.

Overall accuracy of the model – 89% of total predictions are correct.

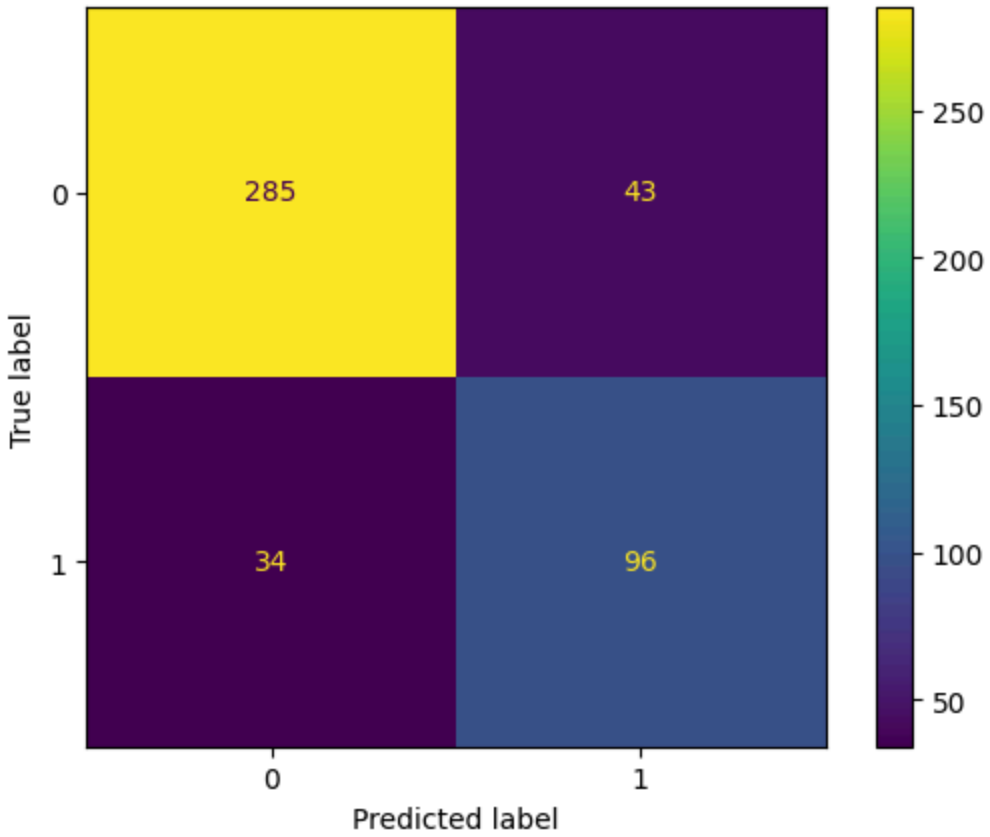
Confusion Matrix for the Test data

In [124…
```
cm_test = confusion_matrix(y_test, ytest_predict)
cm_test
```

Out[124…
```
array([[285,  43],
       [ 34,  96]], dtype=int64)
```

In [125…
```
disp = ConfusionMatrixDisplay(confusion_matrix=cm_test, display_labels=BGT_model.cl
disp.plot()
```

Out[125…
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x21be2fa1a80>
```



In [126…
```
print(classification_report(y_test, ytest_predict))
```

```
              precision    recall  f1-score   support

           0       0.89      0.87      0.88       328
           1       0.69      0.74      0.71       130

    accuracy                           0.83       458
   macro avg       0.79      0.80      0.80       458
weighted avg       0.84      0.83      0.83       458
```

Model Performance (before tuning of Gradient Boosting Model - Test data):

For predicting party choice as Labor (Label 0):

Precision (89%) – 89% of voters predicted are actually voting the Labor party out of all voters predicted to vote Labor party.

Recall (87%) – Out of all voters actually voting the Labor party, 87% of voters are predicted to vote Labor party.

For predicting party choice as Conservative (Label 1):

Precision (69%) – 69% of voters predicted are actually voting the Conservative party out of all voters predicted to vote Conservative party.

Recall (74%) – Out of all voters actually voting the Conservative party, 74% of voters are predicted to vote Conservative party.

Overall accuracy of the model – 83% of total predictions are correct.

Hyperparameter Tuning - Gradient Boosting Model

In [127…

```
# Choose the type of classifier.
BGT_model_tuned = GradientBoostingClassifier(random_state=1)

# Grid of parameters to choose from
param_grid = {
    'n_estimators': [10, 50, 100],
    'learning_rate': [0.01, 0.1, 1.0],
    'max_depth': [5,7,10],
    'max_features': [4,6,8],
    'min_samples_leaf': [5,10],
    'min_samples_split': [50,100]
}

# Run the grid search
grid_obj = GridSearchCV(BGT_model_tuned, parameters, cv=5)
grid_obj = grid_obj.fit(X_train, y_train)

# Set the clf to the best combination of parameters
BGT_model_tuned = grid_obj.best_estimator_
```

```
# Fit the best algorithm to the data.
BGT_model_tuned.fit(X_train, y_train)
```

Out[127…

GradientBoostingClassifier

```
GradientBoostingClassifier(n_estimators=50, random_state=1)
```

Predicting on Training and Test dataset

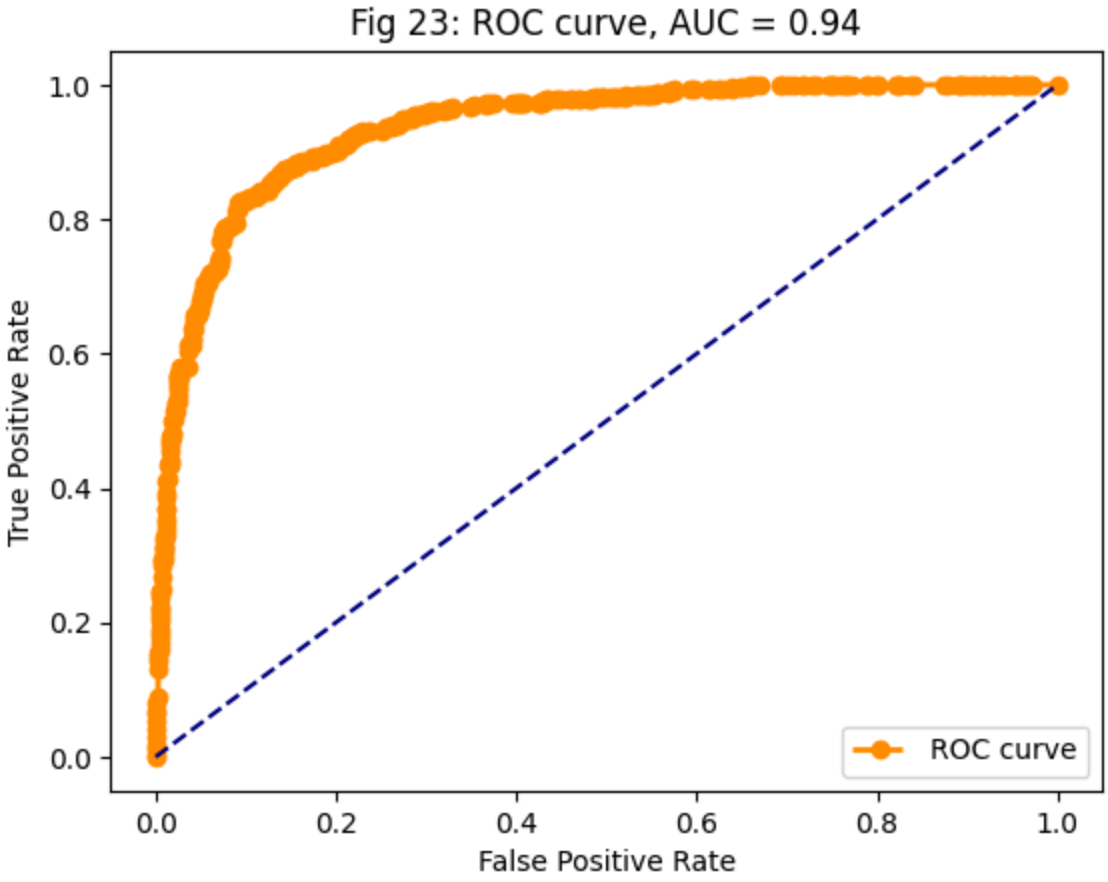
In [128…

```
ytrain_predict = BST_model_tuned.predict(X_train)
ytest_predict = BST_model_tuned.predict(X_test)
```

AUC and ROC for the Training and Test data

In [129…

```
# predict probabilities
probs = BGT_model_tuned.predict_proba(X_train)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# calculate AUC
auc = roc_auc_score(y_train, probs)
# calculate roc curve
train_fpr, train_tpr, train_thresholds = roc_curve(y_train, probs)
# plot the roc curve for the model
plt.plot(train_fpr, train_tpr, linestyle='--', marker='o', color='darkorange', lw =
plt.plot([0, 1], [0, 1], color='navy', linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Fig 23: ROC curve, AUC = %.2f'%auc)
plt.legend(loc="lower right")
plt.show()
```

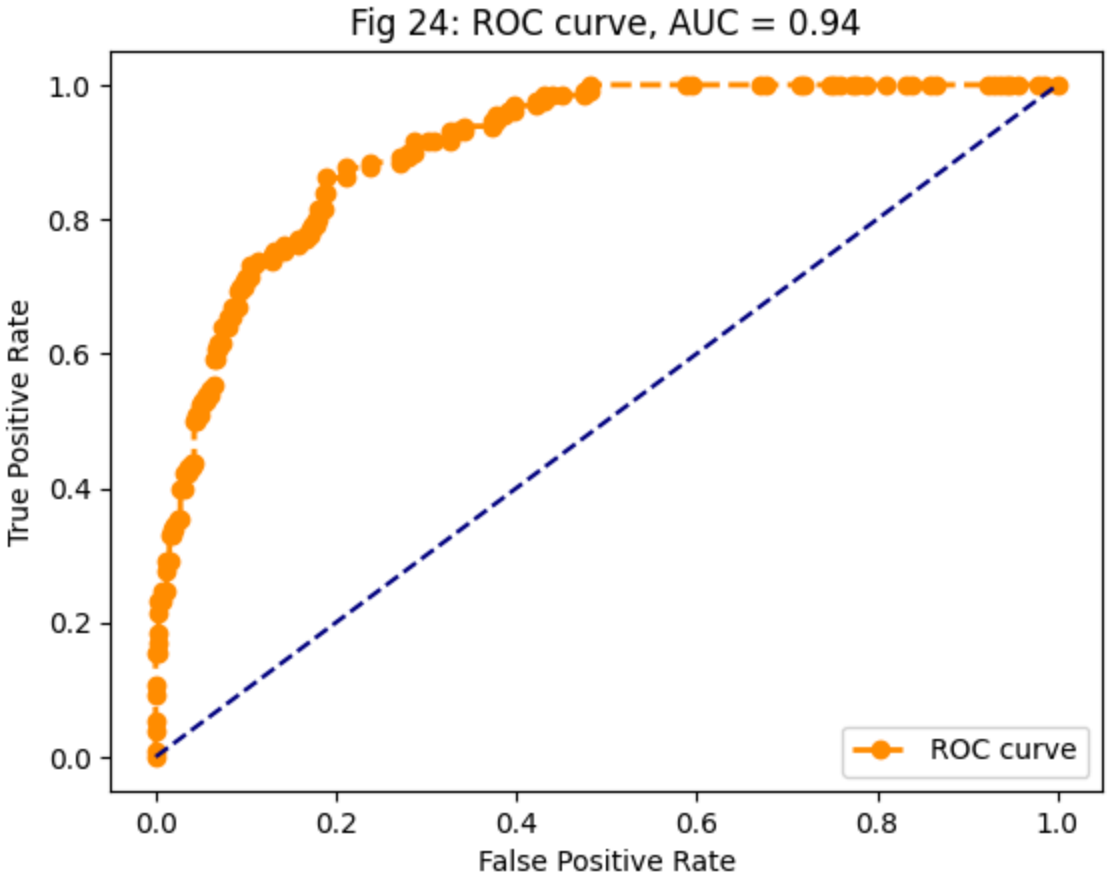


Observations and Insights:

- AOC (Area under the ROC Curve) is 0.94 (excellent).
- ROC curve is closer to the top-left corner which indicates a better performance.

In [130…

```
# predict probabilities
probs = BGT_model_tuned.predict_proba(X_test)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# calculate AUC
test_auc = roc_auc_score(y_test, probs)
# calculate roc curve
test_fpr, test_tpr, test_thresholds = roc_curve(y_test, probs)
# plot the roc curve for the model
plt.plot(test_fpr, test_tpr, linestyle='--', marker='o', color='darkorange', lw = 2
plt.plot([0, 1], [0, 1], color='navy', linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Fig 24: ROC curve, AUC = %.2f'%auc)
plt.legend(loc="lower right")
plt.show()
```



Observations and Insights:

- AOC (Area under the ROC Curve) is 0.94 (excellent).
- ROC curve is closer to the top-left corner which indicates a better performance.

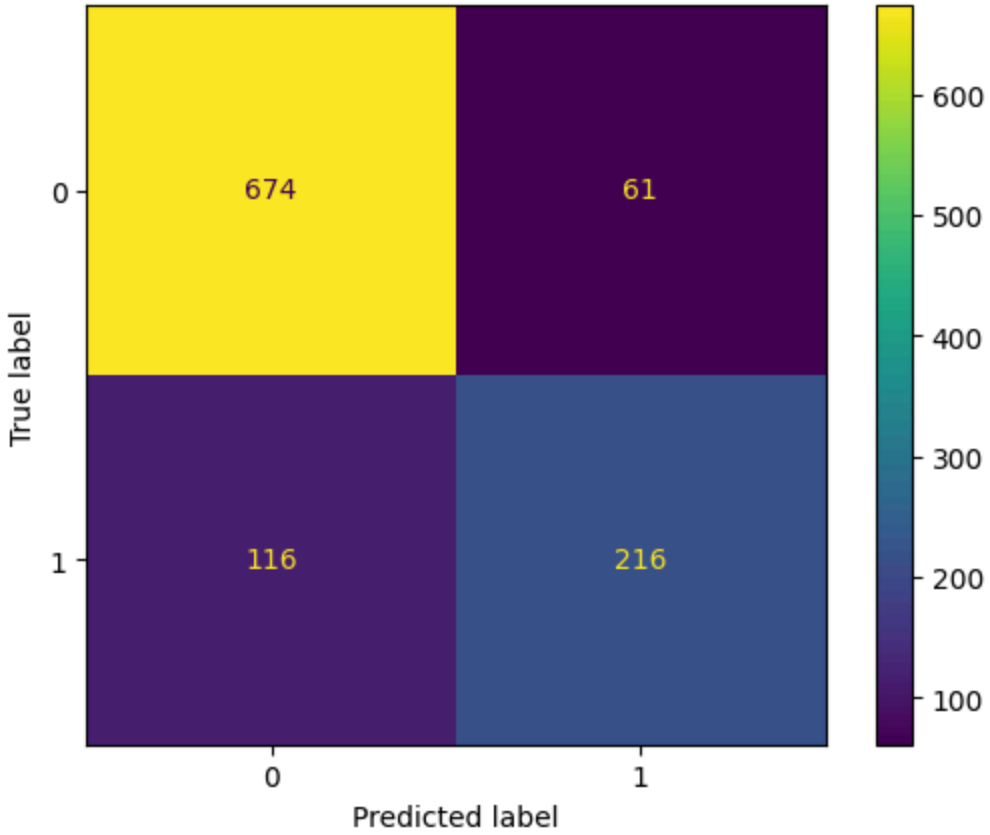
Confusion Matrix for the Training and Test data

In [131…
```
cm_train_tuned = confusion_matrix(y_train, ytrain_predict)
cm_train_tuned
```

Out[131…
```
array([[674,  61],
       [116, 216]], dtype=int64)
```

In [132…
```
disp = ConfusionMatrixDisplay(confusion_matrix=cm_train_tuned, display_labels=BGT_m
disp.plot()
```

Out[132…
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x21be5c5b790>
```



In [133…

```
print(classification_report(y_train, ytrain_predict))
```

```
              precision    recall  f1-score   support

           0       0.85      0.92      0.88       735
           1       0.78      0.65      0.71       332

    accuracy                           0.83      1067
   macro avg       0.82      0.78      0.80      1067
weighted avg       0.83      0.83      0.83      1067
```

Model Performance (after tuning of Gradient Boosting Model - Training data):

For predicting party choice as Labor (Label 0):

Precision (85%) – 85% of voters predicted are actually voting the Labor party out of all voters predicted to vote Labor party.

Recall (92%) – Out of all voters actually voting the Labor party, 92% of voters are predicted to vote Labor party.

For predicting party choice as Conservative (Label 1):

Precision (78%) – 78% of voters predicted are actually voting the Conservative party out of all voters predicted to vote Conservative party.

Recall (65%) – Out of all voters actually voting the Conservative party, 65% of voters are predicted to vote Conservative party.

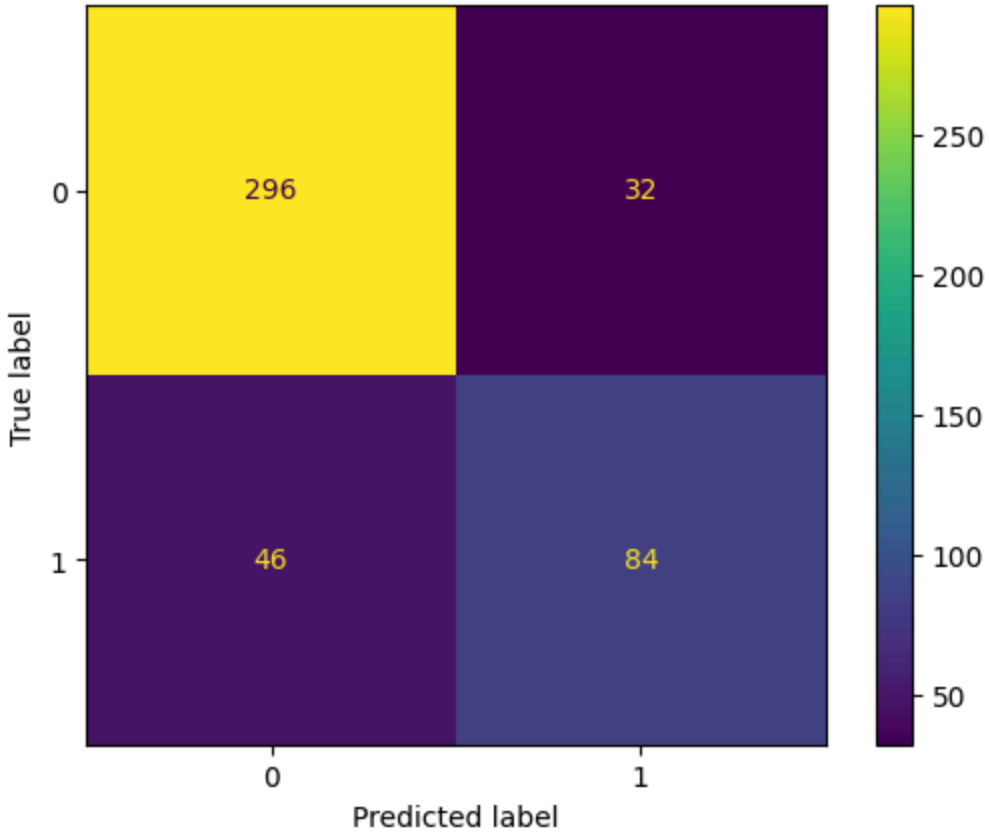
Overall accuracy of the model – 83% of total predictions are correct.

In [134…
```
cm_test_tuned = confusion_matrix(y_test, ytest_predict)
cm_test_tuned
```

Out[134…
```
array([[296,  32],
       [ 46,  84]], dtype=int64)
```

In [135…
```
disp = ConfusionMatrixDisplay(confusion_matrix=cm_test_tuned, display_labels=BGT_mo
disp.plot()
```

Out[135…
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x21be5cffa90>
```



In [136…
```
print(classification_report(y_test, ytest_predict))
```

```
              precision    recall  f1-score   support

           0       0.87      0.90      0.88       328
           1       0.72      0.65      0.68       130

    accuracy                           0.83       458
   macro avg       0.79      0.77      0.78       458
weighted avg       0.83      0.83      0.83       458
```

Model Performance (after tuning of Gradient Boosting Model - Test data):

For predicting party choice as Labor (Label 0):

Precision (87%) – 87% of voters predicted are actually voting the Labor party out of all voters predicted to vote Labor party.

Recall (90%) – Out of all voters actually voting the Labor party, 90% of voters are predicted to vote Labor party.

For predicting party choice as Conservative (Label 1):

Precision (72%) – 72% of voters predicted are actually voting the Conservative party out of all voters predicted to vote Conservative party.

Recall (65%) – Out of all voters actually voting the Conservative party, 65% of voters are predicted to vote Conservative party.

Overall accuracy of the model – 83% of total predictions are correct. There is no change in the overall accuracy of model after tuning the Gradient Boosting Model.

Final Model

Best Models: **Navie Bayes Model, AdaBoost Model (after tuning), Gradient Boosting Model (after tuning)**

Rationale:

- **Comparison of Accuracy on test data:** Navie Bayes Model: 83%, AdaBoost Model (after tuning): 83%, Gradient Boosting Model (after tuning): 83%, KNN Model: 82%, Random Forest Model: 82%, Bagging Model (after tuning): 82%. Navie Bayes Model, AdaBoost Model (after tuning) and Gradient Boosting Model (after tuning) are having highest Accuracy on test data.
- **Navie Bayes Model, AdaBoost Model (after tuning), Gradient Boosting Model (after tuning):** Accuracy, AUC, Precision and Recall for test data is almost inline with training data. This proves no overfitting or underfitting has happened, and overall the models are good for classification.

Important features of the AdaBoost Model (after tuning)

In [137…

```
# importance of features in the AdaBoost Model (after tuning)

print (pd.DataFrame(BST_model_tuned.feature_importances_, columns = ["Imp"], index
```

```
                          Imp
Hague                    0.24
Blair                    0.20
Europe                   0.19
age                      0.12
economic.cond.national   0.10
political.knowledge      0.10
economic.cond.household  0.05
gender                   0.00
```

Features importance (AdaBoost Model (after tuning)):

Hague, Blair, Europe, age, economic.cond.national, political.knowledge, economic.cond.household and gender (in same order of preference) are the most important Features in determining if a voter will vote for a particular political party (Labor or Conservative).

Important features of the Gradient Boosting Model (after tuning)

In [138…

```
# importance of features in the Gradient Boosting Model (after tuning)

print (pd.DataFrame(BGT_model_tuned.feature_importances_, columns = ["Imp"], index
```

```
                          Imp
Hague                0.369026
Europe               0.191889
Blair                0.187477
political.knowledge  0.111081
age                  0.087185
economic.cond.national   0.033154
economic.cond.household  0.018543
gender               0.001644
```

Features importance (Gradient Boosting Model (after tuning)):

Hague, Europe, Blair, political.knowledge, age, economic.cond.national, economic.cond.household and gender (in same order of preference) are the most important Features in determining if a voter will vote for a particular political party (Labor or Conservative).

Actionable Insights:

- Labor and Conservative leaders image among the voters is the most important factor.
- Knowledge of polititcal parties on European integration is the second most important factor.
- Age of voters playes an important role in selecting the candidate of a political party (Labor or Conservative).
- Current status of economic condition of nation and household among voters (especially Females) also plays an important role in selecting the candidate of a political party

(Labor or Conservative).

Business Recommendations:

- Conservative party can increase number of seats if they focus on improving the image of their leaders among voters.
- Labor party can increase number of seats as well if they focus on further improving the image of their leaders among voters.
- 'Eurosceptic' sentiment (attitude toward European integration) is higher among the voters. So each political party need to improve their knowledge in European integration to get more seats.
- Both political parties can increase number of seats if they focus on improving the image of their leaders and by showcasing the knowledge in European integration among voters of different age groups.
- Both political parties can increase number of seats if they can showcase the roadmap of further improving the economic condition of nation and household among voters (especially Females).

In [ ]: