Code for 4.3 a) -------------------------------> Univariate Feature Selection

In [20]:
```
# Importing the essential libraries
import numpy as np # For Numerical computations
import pandas as pd # For Data analysis and Manipulation
import matplotlib.pyplot as plt # For Data Visualization
import seaborn as sns # For more better visualization
from warnings import simplefilter # import warnings filter
simplefilter(action='ignore', category=FutureWarning) # ignore all future warnings

# Load csv into pandas dataframe
adult_DF = pd.read_csv("adult.csv")

# Handling missing values
adult_DF = adult_DF.replace('?', np.NaN)
adult_DF= adult_DF.dropna()

# Encoding the categorical features
from sklearn.preprocessing import LabelEncoder
encoder = LabelEncoder()
adult_DF = adult_DF.apply(encoder.fit_transform)

# Dropping the similar column
adult_DF= adult_DF.drop(['educational-num'], axis=1)

# Seperating out the target feature column from all other features
predictorDF = adult_DF.iloc[:, :-1]
targetDF = adult_DF.iloc[:, 13]

# Dividing our dataset into train and test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(predictorDF, targetDF, test_size = 0.2, random_s
tate = 0, stratify=targetDF)

# Only the training data needs to be balanced, not the entire dataset
from imblearn.over_sampling import SMOTE
sm = SMOTE(random_state=0)
# Oversampling the training data instances
X_train, y_train = sm.fit_sample(X_train, y_train)

#------------------------------------------- UNIVARIATE FEATURE SELECTION--------------------------
-----------------------------

# Importing the necessary libraries
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
# Apply the SelectKBest to extract the top 10 features
bestFeatures = SelectKBest(score_func=chi2, k=10)
fit= bestFeatures.fit(X_train, y_train)
# Creating the individual data-frames of scores of individual feature with target variable and featu
re name
scoresDF = pd.DataFrame(fit.scores_)
featuresDF = pd.DataFrame(predictorDF.columns)
# Now combine these 2 dataframes
featuresScoresDF= pd.concat([featuresDF, scoresDF], axis=1)
# Naming the columns of combined dataframe
featuresScoresDF.columns= ['Feature_Name', 'Relationship Score of Feature with target variable']
# Print the features in decending order of scores
print(featuresScoresDF.nlargest(13, 'Relationship Score of Feature with target variable'))
# Removing the 3 features with least scores (retaining the top 10 features) from X_train and X_test
# Removing the race, native-country and occupation feature
X_train = np.delete(X_train, [5,7,12], axis=1)
X_test = np.delete(X_test, [5,7,12], axis=1)

# Scaling the dataset
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
predictorDF = scaler.fit_transform(predictorDF)
```

```
      Feature_Name  Relationship Score of Feature with target variable
9     capital-gain                                      405056.295251
10    capital-loss                                       55023.810547
0              age                                       27946.513014
6     relationship                                       14415.386575
2           fnlwgt                                       12019.297737
11  hours-per-week                                       11239.134583
4   marital-status                                        3413.044179
8           gender                                         509.309371
3        education                                         265.391648
1        workclass                                         161.140730
12  native-country                                         107.136322
5       occupation                                          42.656172
7             race                                          32.429086
```

Function to evaluate the performance after feature selection

In [3]:
```
# Function to calculate the performance after univariate feature selection
def performance(predictorDF_train, predictorDF_test, targetDF_train, targetDF_test):
        # Importing the required libraries
        from sklearn.metrics import accuracy_score
        from sklearn.metrics import classification_report
        from sklearn.metrics import confusion_matrix
        from sklearn.metrics import f1_score
        from sklearn.ensemble import RandomForestClassifier
        from sklearn.ensemble import GradientBoostingClassifier

        # Models with the best hyper-parameters
        tunedRF = RandomForestClassifier(bootstrap= True, max_depth=70, max_features= 'auto', min_sa
mples_leaf= 4, min_samples_split= 10, n_estimators= 400, random_state=0)
        tunedGB = GradientBoostingClassifier(learning_rate=0.01, n_estimators=1500, max_depth=4, min
_samples_split=40, min_samples_leaf=7, max_features=4,subsample=0.95, random_state=0)

        model_names = ["Random Forest", "Gradient Boosting"]
        tuned_models = [tunedRF, tunedGB]

        # Test set results after Univariate Feature Selection
        for model_name, tuned_model in zip(model_names, tuned_models):
            tuned_model = tuned_model.fit(predictorDF_train, targetDF_train)
            results= tuned_model.predict(predictorDF_test)
            accuracy = accuracy_score(results, targetDF_test)
            print("\nTEST ACCURACY on tuned {} is:{} ".format(model_name, accuracy))
            f1Score = f1_score(results, targetDF_test)
            print("F1 SCORE on tuned {} is:{} ".format(model_name, f1Score))
            report = classification_report(targetDF_test, results)
            print("The CLASSIFICATION REPORT for tuned {} is: {}\n".format(model_name, report))
            print("The CONFUSION MATRIX of {} is:".format(model_name))
            print(confusion_matrix(targetDF_test,results))
            print("--------------------------------------------------------------------------------
-------")
            print("\n")
```

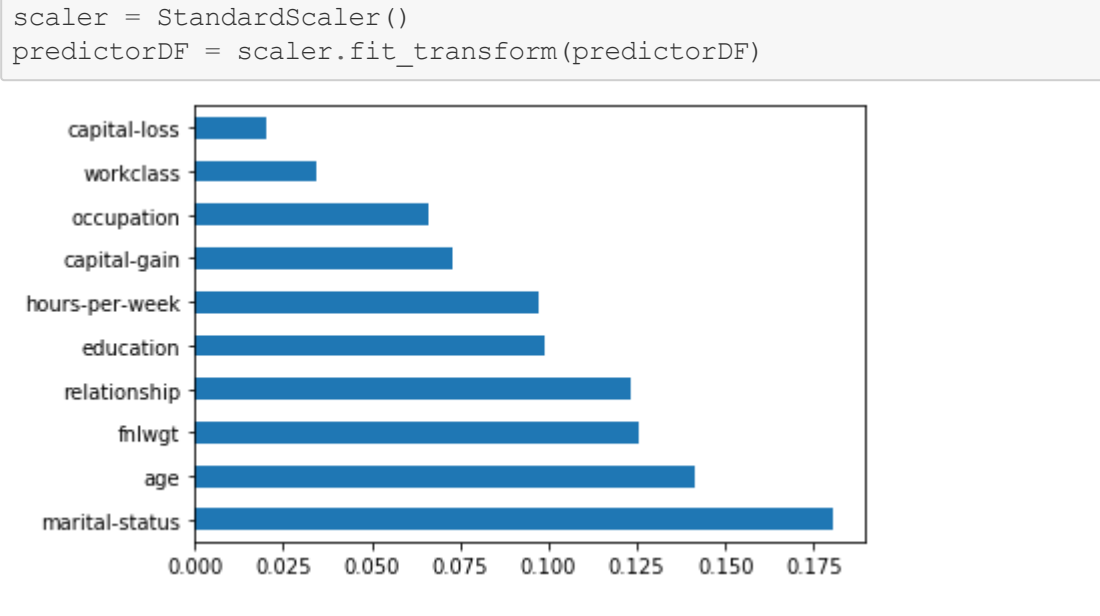
Performance of the model after Univariate feature selection

In [39]:
```
# Calling the function which calculates the performance of the models after feature selection
performance(X_train, X_test, y_train, y_test)
```

```
TEST ACCURACY on tuned Random Forest is:0.8352681039248203
F1 SCORE on tuned Random Forest is:0.6857865879375791
The CLASSIFICATION REPORT for tuned Random Forest is:               precision    recall  f1-scor
e   support

           0       0.91      0.87      0.89      6803
           1       0.65      0.73      0.69      2242

    accuracy                           0.84      9045
   macro avg       0.78      0.80      0.79      9045
weighted avg       0.84      0.84      0.84      9045


The CONFUSION MATRIX of Random Forest is:
[[5929  874]
 [ 616 1626]]
-------------------------------------------------------------------------------------------



TEST ACCURACY on tuned Gradient Boosting is:0.8337202874516307
F1 SCORE on tuned Gradient Boosting is:0.6960388035569927
The CLASSIFICATION REPORT for tuned Gradient Boosting is:               precision    recall  f1-
score   support

           0       0.92      0.86      0.89      6803
           1       0.64      0.77      0.70      2242

    accuracy                           0.83      9045
   macro avg       0.78      0.81      0.79      9045
weighted avg       0.85      0.83      0.84      9045


The CONFUSION MATRIX of Gradient Boosting is:
[[5819  984]
 [ 520 1722]]
-------------------------------------------------------------------------------------------
```

Code for 4.3 b) -------------------------------> Tree based feature selection (Feature Importance)

In [22]:
```
# Importing the essential libraries
import numpy as np # For Numerical computations
import pandas as pd # For Data analysis and Manipulation
import matplotlib.pyplot as plt # For Data Visualization
import seaborn as sns # For more better visualization
from warnings import simplefilter # import warnings filter
simplefilter(action='ignore', category=FutureWarning) # ignore all future warnings

# Load csv into pandas dataframe
adult_DF = pd.read_csv("adult.csv")

# Handling missing values
adult_DF = adult_DF.replace('?', np.NaN)
adult_DF= adult_DF.dropna()

# Encoding the categorical features
from sklearn.preprocessing import LabelEncoder
encoder = LabelEncoder()
adult_DF = adult_DF.apply(encoder.fit_transform)

# Dropping the similar column
adult_DF= adult_DF.drop(['educational-num'], axis=1)

# Seperating out the target feature column from all other features
predictorDF = adult_DF.iloc[:, :-1]
targetDF = adult_DF.iloc[:, 13]

# Dividing our dataset into train and test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(predictorDF, targetDF, test_size = 0.2, random_s
tate = 0, stratify=targetDF)

# Only the training data needs to be balanced, not the entire dataset
from imblearn.over_sampling import SMOTE
sm = SMOTE(random_state=0)
# Oversampling the training data instances
X_train, y_train = sm.fit_sample(X_train, y_train)

# ------------------------------------------TREE BASED FEATURE SELECTION----------------------------
-------------------

# Importing the libraries
from sklearn.ensemble import RandomForestClassifier
import matplotlib.pyplot as plt
# Using Random forest classifier for extracting the top 10 features
model = RandomForestClassifier(n_estimators=700)
model.fit(X_train, y_train)
# Using the feature importance attribute of ExtraTreesClassifier class
importances = model.feature_importances_
# What are the top 10 features on the basis of scores
feature_scores = pd.Series(importances, index = predictorDF.columns)
feature_scores.nlargest(10).plot(kind='barh')
plt.show()
# Removing the race, native-country and gender feature which have the least scores
X_train = np.delete(X_train, [8,7,12], axis=1)
X_test = np.delete(X_test, [8,7,12], axis=1)

# Scaling the dataset
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
predictorDF = scaler.fit_transform(predictorDF)
```



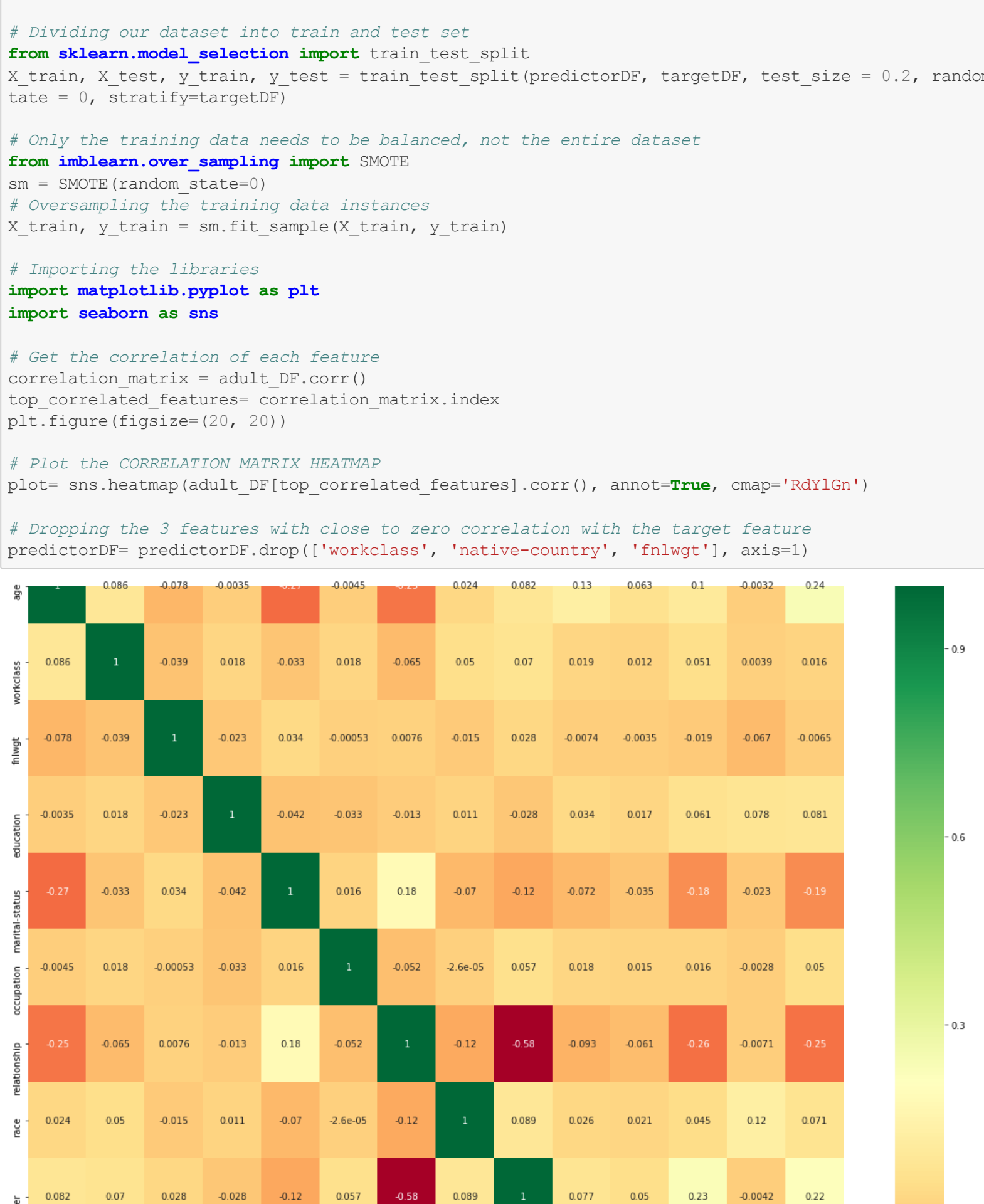
Performance of the model after Tree Based Feature Selection

In [44]:
```
# Calling the function which calculates the performance of the tuned models after feature selection
performance(X_train, X_test, y_train, y_test)
```

```
TEST ACCURACY on tuned Random Forest is:0.8364842454394693
F1 SCORE on tuned Random Forest is:0.6855198809270678
The CLASSIFICATION REPORT for tuned Random Forest is:               precision    recall  f1-scor
e   support

           0       0.90      0.88      0.89      6803
           1       0.66      0.72      0.69      2242

    accuracy                           0.84      9045
   macro avg       0.78      0.80      0.79      9045
weighted avg       0.84      0.84      0.84      9045


The CONFUSION MATRIX of Random Forest is:
[[5954  849]
 [ 630 1612]]
-------------------------------------------------------------------------------------------



TEST ACCURACY on tuned Gradient Boosting is:0.8371475953565506
F1 SCORE on tuned Gradient Boosting is:0.6979700635636661
The CLASSIFICATION REPORT for tuned Gradient Boosting is:               precision    recall  f1-
score   support

           0       0.92      0.86      0.89      6803
           1       0.65      0.76      0.70      2242

    accuracy                           0.84      9045
   macro avg       0.78      0.81      0.79      9045
weighted avg       0.85      0.84      0.84      9045


The CONFUSION MATRIX of Gradient Boosting is:
[[5870  933]
 [ 540 1702]]
-------------------------------------------------------------------------------------------
```

Code for 4.3 c) -------------------------------> Correlation Matrix with Heat Map

In [45]:
```
# Importing the essential libraries
import numpy as np # For Numerical computations
import pandas as pd # For Data analysis and Manipulation
import matplotlib.pyplot as plt # For Data Visualization
import seaborn as sns # For more better visualization
from warnings import simplefilter # import warnings filter
simplefilter(action='ignore', category=FutureWarning) # ignore all future warnings

# Load csv into pandas dataframe
adult_DF = pd.read_csv("adult.csv")

# Handling missing values
adult_DF = adult_DF.replace('?', np.NaN)
adult_DF= adult_DF.dropna()

# Encoding the categorical features
from sklearn.preprocessing import LabelEncoder
encoder = LabelEncoder()
adult_DF = adult_DF.apply(encoder.fit_transform)

# Dropping the similar column
adult_DF= adult_DF.drop(['educational-num'], axis=1)

# Seperating out the target feature column from all other features
predictorDF = adult_DF.iloc[:, :-1]
targetDF = adult_DF.iloc[:, 13]

# Scaling the dataset
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
predictorDF = scaler.fit_transform(predictorDF)

# Dividing our dataset into train and test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(predictorDF, targetDF, test_size = 0.2, random_s
tate = 0, stratify=targetDF)

# Only the training data needs to be balanced, not the entire dataset
from imblearn.over_sampling import SMOTE
sm = SMOTE(random_state=0)
# Oversampling the training data instances
X_train, y_train = sm.fit_sample(X_train, y_train)

# Importing the libraries
import matplotlib.pyplot as plt
import seaborn as sns

# Get the correlation of each feature
correlation_matrix = adult_DF.corr()
top_correlated_features= correlation_matrix.index
plt.figure(figsize=(20, 20))

# Plot the CORRELATION MATRIX HEATMAP
plot= sns.heatmap(adult_DF[top_correlated_features].corr(), annot=True, cmap='RdYlGn')

# Dropping the 3 features with close to zero correlation with the target feature
predictorDF= predictorDF.drop(['workclass', 'native-country', 'fnlwgt'], axis=1)
```



Performance of the model after Correlation Heat Map

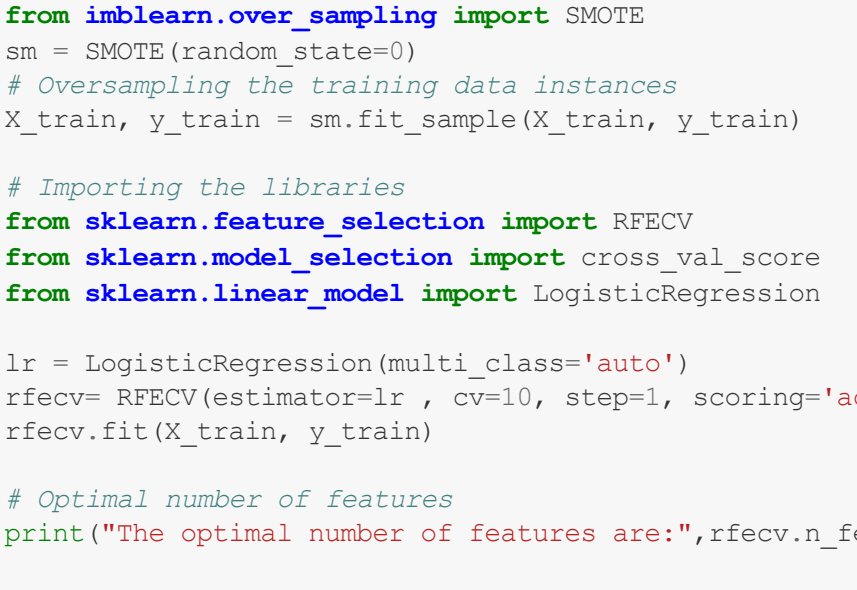
In [46]:
```
# Calling the function which calculates the performance of the tuned models after feature selection
 using correlation matrix
performance(X_train, X_test, y_train, y_test)
```

```
TEST ACCURACY on tuned Random Forest is:0.8367053620784964
F1 SCORE on tuned Random Forest is:0.6875396657499472
The CLASSIFICATION REPORT for tuned Random Forest is:               precision    recall  f1-scor
e   support

           0       0.91      0.87      0.89      6803
           1       0.65      0.72      0.69      2242

    accuracy                           0.84      9045
   macro avg       0.78      0.80      0.79      9045
weighted avg       0.84      0.84      0.84      9045


The CONFUSION MATRIX of Random Forest is:
[[5943  860]
 [ 617 1625]]
-------------------------------------------------------------------------------------------



TEST ACCURACY on tuned Gradient Boosting is:0.8360420121614152
F1 SCORE on tuned Gradient Boosting is:0.6964176049129989
The CLASSIFICATION REPORT for tuned Gradient Boosting is:               precision    recall  f1-
score   support

           0       0.92      0.86      0.89      6803
           1       0.64      0.76      0.70      2242

    accuracy                           0.84      9045
   macro avg       0.78      0.81      0.79      9045
weighted avg       0.85      0.84      0.84      9045


The CONFUSION MATRIX of Gradient Boosting is:
[[5861  942]
 [ 541 1701]]
-------------------------------------------------------------------------------------------
```

Code for 4.3 d) -------------> Greedy Feature Selection (Recursive feature elimination with cross validation)

In [26]:
```
# Importing the essential libraries
import numpy as np # For Numerical computations
import pandas as pd # For Data analysis and Manipulation
import matplotlib.pyplot as plt # For Data Visualization
import seaborn as sns # For more better visualization
from warnings import simplefilter # import warnings filter
simplefilter(action='ignore', category=FutureWarning) # ignore all future warnings

# Load csv into pandas dataframe
adult_DF = pd.read_csv("adult.csv")

# Handling missing values
adult_DF = adult_DF.replace('?', np.NaN)
adult_DF= adult_DF.dropna()

# Encoding the categorical features
from sklearn.preprocessing import LabelEncoder
encoder = LabelEncoder()
adult_DF = adult_DF.apply(encoder.fit_transform)

# Dropping the similar column
adult_DF= adult_DF.drop(['educational-num'], axis=1)

# Seperating out the target feature column from all other features
predictorDF = adult_DF.iloc[:, :-1]
targetDF = adult_DF.iloc[:, 13]

# Scaling the dataset
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
predictorDF = scaler.fit_transform(predictorDF)

# Dividing our dataset into train and test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(predictorDF, targetDF, test_size = 0.2, random_s
tate = 0, stratify=targetDF)

# Only the training data needs to be balanced, not the entire dataset
from imblearn.over_sampling import SMOTE
sm = SMOTE(random_state=0)
# Oversampling the training data instances
X_train, y_train = sm.fit_sample(X_train, y_train)

# Importing the libraries
from sklearn.feature_selection import RFECV
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression

lr = LogisticRegression(multi_class='auto')
rfecv= RFECV(estimator=lr , cv=10, step=1, scoring='accuracy')
rfecv.fit(X_train, y_train)

# Optimal number of features
print("The optimal number of features are:",rfecv.n_features_)

# How many features gave the highest score
plt.figure()
plt.title('Logistic Regression CV score vs No of Features')
plt.xlabel("Number of features selected")
plt.ylabel("Cross validation score")
plt.plot(range(1, len(rfecv.grid_scores_) + 1), rfecv.grid_scores_)
plt.show()
```

```
The optimal number of features are: 11
```



In [ ]:
```
# Remove the features fnlwgt and native country which are marked least important by our RFECV techni
que of feature selection
X_train = X_train[:, rfecv.support_]
X_test = X_test[:, rfecv.support_]
```

In [8]:
```
# Calling the function which calculates the performance of the tuned models after feature selection
 using RFECV
performance(X_train, X_test, y_train, y_test)
```

```
TEST ACCURACY on tuned Random Forest is:0.8398009950248756
F1 SCORE on tuned Random Forest is:0.6991903674486194
The CLASSIFICATION REPORT for tuned Random Forest is:               precision    recall  f1-scor
e   support

           0       0.91      0.87      0.89      6803
           1       0.65      0.75      0.70      2242

    accuracy                           0.84      9045
   macro avg       0.78      0.81      0.80      9045
weighted avg       0.85      0.84      0.84      9045


The CONFUSION MATRIX of Random Forest is:
[[5912  891]
 [ 558 1684]]
-------------------------------------------------------------------------------------------



TEST ACCURACY on tuned Gradient Boosting is:0.8402432283029297
F1 SCORE on tuned Gradient Boosting is:0.7068370866301482
The CLASSIFICATION REPORT for tuned Gradient Boosting is:               precision    recall  f1-
score   support

           0       0.92      0.86      0.89      6803
           1       0.65      0.78      0.71      2242

    accuracy                           0.84      9045
   macro avg       0.78      0.82      0.80      9045
weighted avg       0.85      0.84      0.84      9045


The CONFUSION MATRIX of Gradient Boosting is:
[[5858  945]
 [ 500 1742]]
-------------------------------------------------------------------------------------------
```