

Project Document

Credit Card Fraud Detection Analysis

Harish Chandra Jyoshi
16262845

GitHub link: <https://github.com/HarishChandra95/ISL/tree/master/project/Source>

Summary

- The datasets consists of transactions made by credit cards in September 2013 by European cardholders. This dataset presents transactions that occurred in two days, where we have 492 frauds out of 284,807 transactions. The dataset is highly unbalanced, the positive class (frauds) account for 0.172% of all transactions.
- It contains only numerical input variables which are the result of a PCA transformation. Features V1, V2, ... V28 are the principal components obtained with PCA, the only features which have not been transformed with PCA are 'Time' and 'Amount'. Feature 'Class' is the response variable and it takes value 1 in case of fraud and 0 otherwise.

Approaches Used:

- **Data Preprocessing:**
 - Amount and Time columns in the dataset are scaled
 - Created a subsample of dataframe in order to have equal amount of fraud and non-fraud transactions
 - Using the original dataset may cause overfitting
 - Used Undersampling technique to create a new dataset comprising of an equal representation from both classes.
 - The main goal is to fit the data on the undersampled data and test on original dataset.
- **For predicting the job classifications:**
 - KNN approach
 - Decision Tree Model
 - Random Forest

- Support Vector Machine
- Logistic Regression Model

For each approach generated a model using training data. Evaluated the generated models on predicted versus actual values using confusion matrix. Computed the accuracy and the recall value for each model. The Logistic Regression approach has less error when compared to other approaches like KNN, decision trees etc.

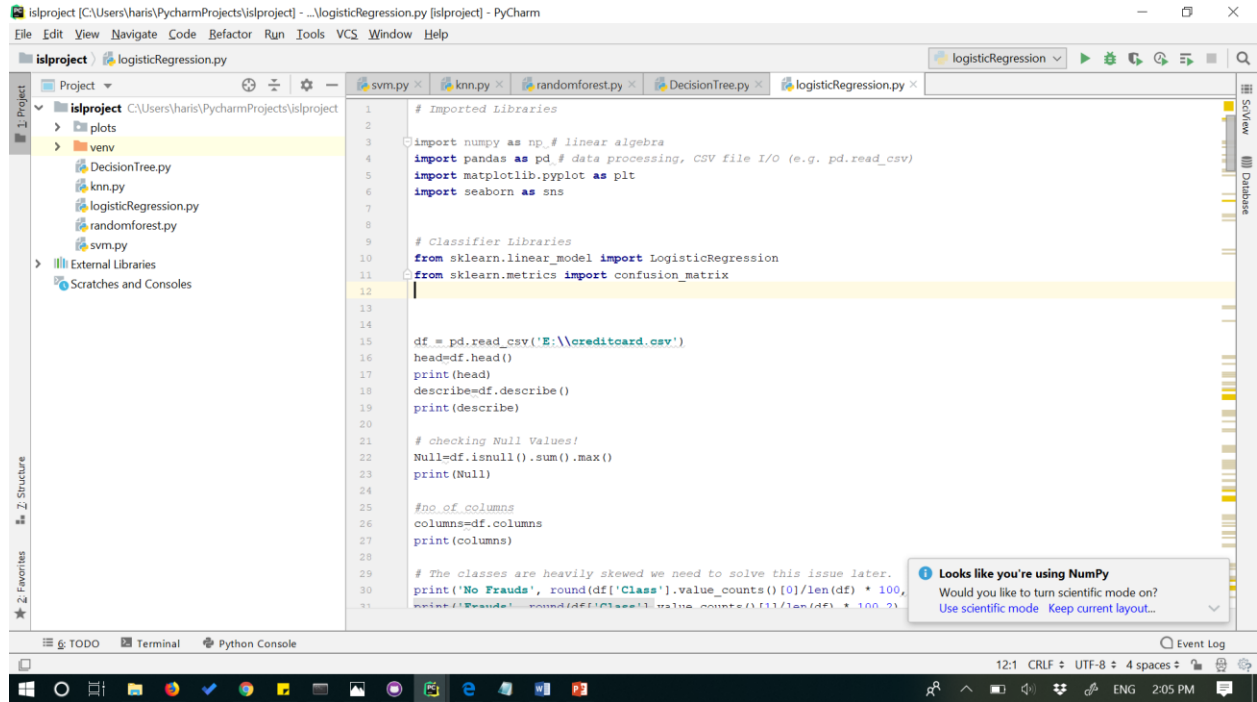
Logistic Regression classifier is more accurate than the other three classifiers in most cases. GridSearchCV is used to determine the parameters that gives the best predictive score for the classifier. Logistic regression model is tested against the full dataset and hit an accuracy of 97.9%.

The credit card fraud detection analysis done using the model created from Logistic Regression approach are accurate. The main issue with "Random Under-Sampling" is that we run the risk that our classification models will not perform as accurate as we would like to since there is a great deal of information loss. In our under-sample data, our model is unable to detect for a large number of cases non-fraud transactions correctly and instead, misclassifies those non fraud transactions as fraud cases. predictions and accuracies may be subjected to change since I implemented data shuffling on both types of data-frames. The main thing is to see if our models are able to correctly classify no fraud and fraud transactions.

Technical Appendix

- Steps:

- a) Imported the required libraries



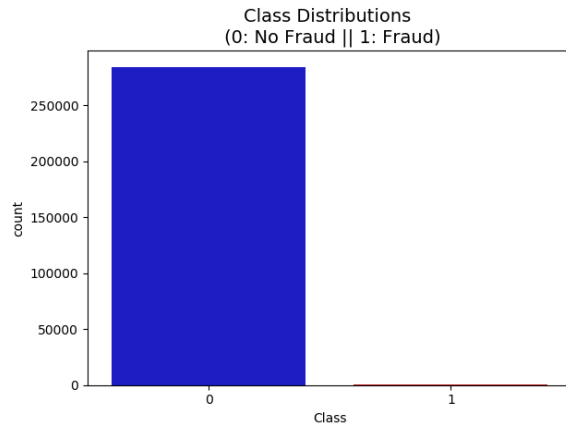
- b) Loading the dataset

```
#loading the dataset
df = pd.read_csv('E:\\creditcard.csv')
head=df.head()
print(head)
describe=df.describe()
print(describe)
```

- c) Initial Data Structure

```
colors = ["#0101DF", "#DF0101"]

sns.countplot('Class', data=df, palette=colors)
plt.title('Class Distributions \n (0: No Fraud || 1: Fraud)', fontsize=14)
plt.show()
```



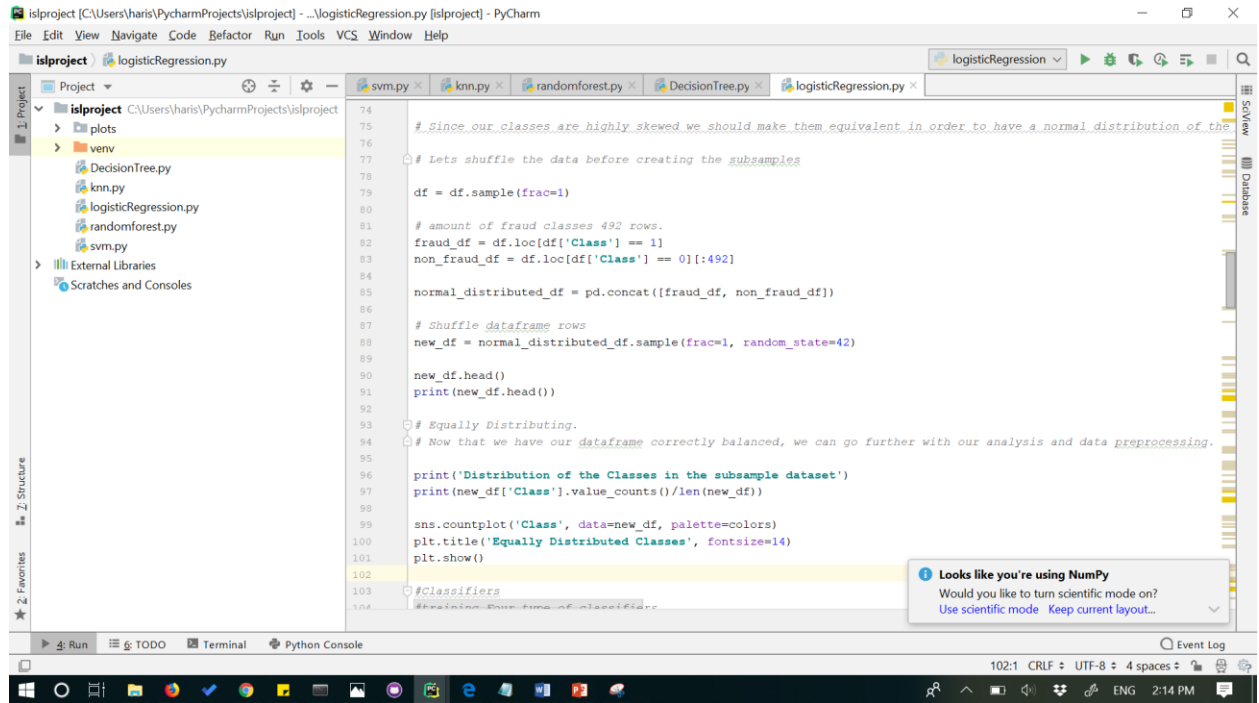
d) Scaling of the dataset and creating sub sample of dataframe

```
is1project [C:\Users\haris\PycharmProjects\is1project] - ... \logisticRegression.py [is1project] - PyCharm
File Edit View Navigate Code Refactor Run Tools VCS Window Help

logisticRegression.py
#scaling of the dataset and creating sub sample of dataframe
# Since most of our data has already been scaled we should scale the columns that are left to scale (Amount and Time)
from sklearn.preprocessing import StandardScaler, RobustScaler
# RobustScaler is less prone to outliers.
rob_scaler = RobustScaler()
df['scaled_amount'] = rob_scaler.fit_transform(df['Amount'].values.reshape(-1,1))
df['scaled_time'] = rob_scaler.fit_transform(df['Time'].values.reshape(-1,1))
df.drop(['Time', 'Amount'], axis=1, inplace=True)
scaled_amount = df['scaled_amount']
scaled_time = df['scaled_time']
df.drop(['scaled_amount', 'scaled_time'], axis=1, inplace=True)
df.insert(0, 'scaled_amount', scaled_amount)
df.insert(1, 'scaled_time', scaled_time)
# Amount and Time are Scaled!
df.head()
print(df.head())
```

e) Random Undersampling of the data.

Since our classes are highly skewed we should make them equivalent in order to have a normal distribution of the classes.



The screenshot shows the PyCharm IDE with a project named 'islproject'. The file explorer on the left shows a directory structure with files like 'DecisionTree.py', 'knn.py', 'logisticRegression.py', 'randomForest.py', and 'svm.py'. The main editor window displays the code in 'logisticRegression.py'. The code implements random undersampling to balance the classes. It starts by shuffling the data and then uses `df.sample(frac=1)` to create a new dataset. It then separates the data into 'fraud' and 'non_fraud' classes based on the 'Class' column. The 'fraud' class is sampled to match the size of the 'non_fraud' class. The resulting dataset is concatenated and shuffled. Finally, it prints the distribution of the classes and displays a countplot.

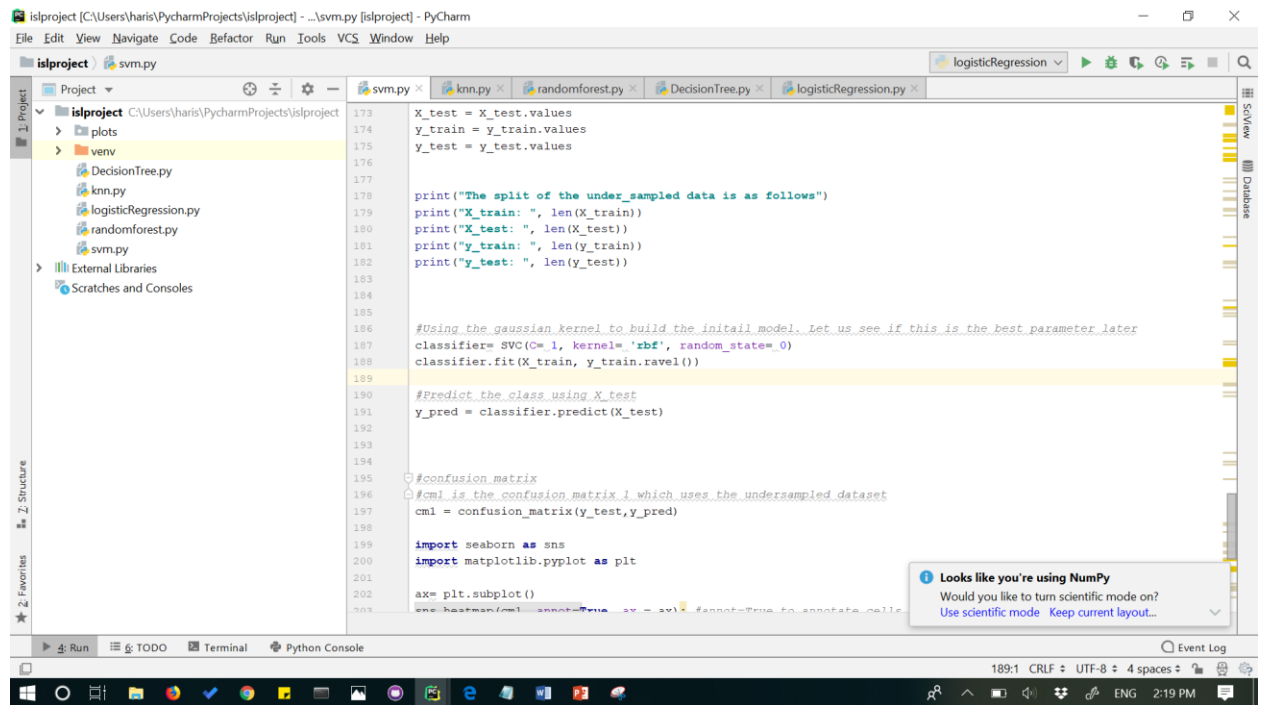
```
74 # Since our classes are highly skewed we should make them equivalent in order to have a normal distribution of the
75
76
77 # Lets shuffle the data before creating the subsamples
78
79 df = df.sample(frac=1)
80
81 # amount of fraud classes 492 rows.
82 fraud_df = df.loc[df['Class'] == 1]
83 non_fraud_df = df.loc[df['Class'] == 0][:492]
84
85 normal_distributed_df = pd.concat([fraud_df, non_fraud_df])
86
87 # Shuffle dataframe rows
88 new_df = normal_distributed_df.sample(frac=1, random_state=42)
89
90 new_df.head()
91 print(new_df.head())
92
93 # Equally Distributing.
94 # Now that we have our dataframe correctly balanced, we can go further with our analysis and data preprocessing.
95
96 print('Distribution of the Classes in the subsample dataset')
97 print(new_df['Class'].value_counts()/len(new_df))
98
99 sns.countplot('Class', data=new_df, palette=colors)
100 plt.title('Equally Distributed Classes', fontsize=14)
101 plt.show()
102
103 #Classifiers
104 #Random Forest Classifier
```

A notification bubble in the bottom right corner of the IDE says: "Looks like you're using NumPy. Would you like to turn scientific mode on? Use scientific mode Keep current layout..."

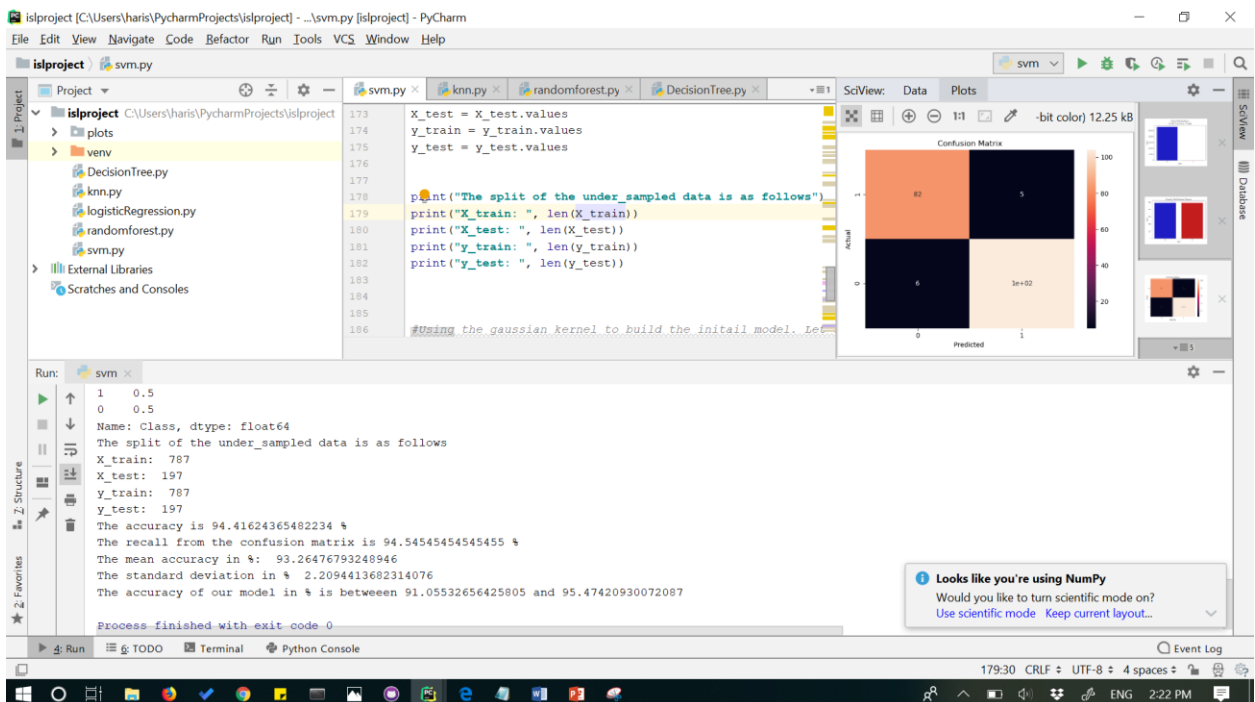


f) Different Classifiers and their outputs

Support vector machine:



Accuracy and recall along with the confusion matrix:



■ KNN approach:

```

175 # This is explicitly used for undersampling.
176 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
177
178 # Turn the values into an array for feeding the classification algorithms.
179 X_train = X_train.values
180 X_test = X_test.values
181 y_train = y_train.values
182 y_test = y_test.values
183
184 print("The split of the under_sampled data is as follows")
185 print("X_train: ", len(X_train))
186 print("X_test: ", len(X_test))
187 print("y_train: ", len(y_train))
188 print("y_test: ", len(y_test))
189
190 #Using the gaussian kernel to build the initial model. Let us see if this is the best parameter later
191 classifier = KNeighborsClassifier(n_neighbors=17)
192 classifier.fit(X_train, y_train.ravel())
193
194 #Predict the class using X_test
195 y_pred = classifier.predict(X_test)
196
197 #confusion matrix
198 #cm1 is the confusion matrix 1 which uses the undersampled dataset
199 cm1 = confusion_matrix(y_test, y_pred)
200
201
202
203
204
205
206
207
208
209

```

Looks like you're using NumPy
Would you like to turn scientific mode on?
Use scientific mode Keep current layout...

Accuracy and recall along with confusion matrix:

```

217 ax.set_xlabel('Predicted '); ax.set_ylabel('Actual');
218 ax.set_title('Confusion Matrix');
219 ax.xaxis.set_ticklabels(['0', '1']); ax.yaxis.set_ticklabels(['0', '1']);
220 plt.show()
221
222 print("The accuracy is " + str((cm1[1, 1] + cm1[0, 0]) / (cm1[1, 1] + cm1[0, 1] + cm1[1, 0] + cm1[0, 0])))
223 print("The recall from the confusion matrix is " + str(cm1[1, 1] / (cm1[1, 1] + cm1[0, 1])))
224
225 #Applying 10 fold cross validation
226 accuracies = cross_val_score(estimator=classifier, X=X_train, y=y_train, cv=10)
227 mean_accuracy = accuracies.mean() * 100
228 std_accuracy = accuracies.std() * 100
229 print("The mean accuracy in %: ", accuracies.mean() * 100)
230 print("The standard deviation in %: ", accuracies.std() * 100)
231 print("The accuracy of our model in % is between () and ()")
232

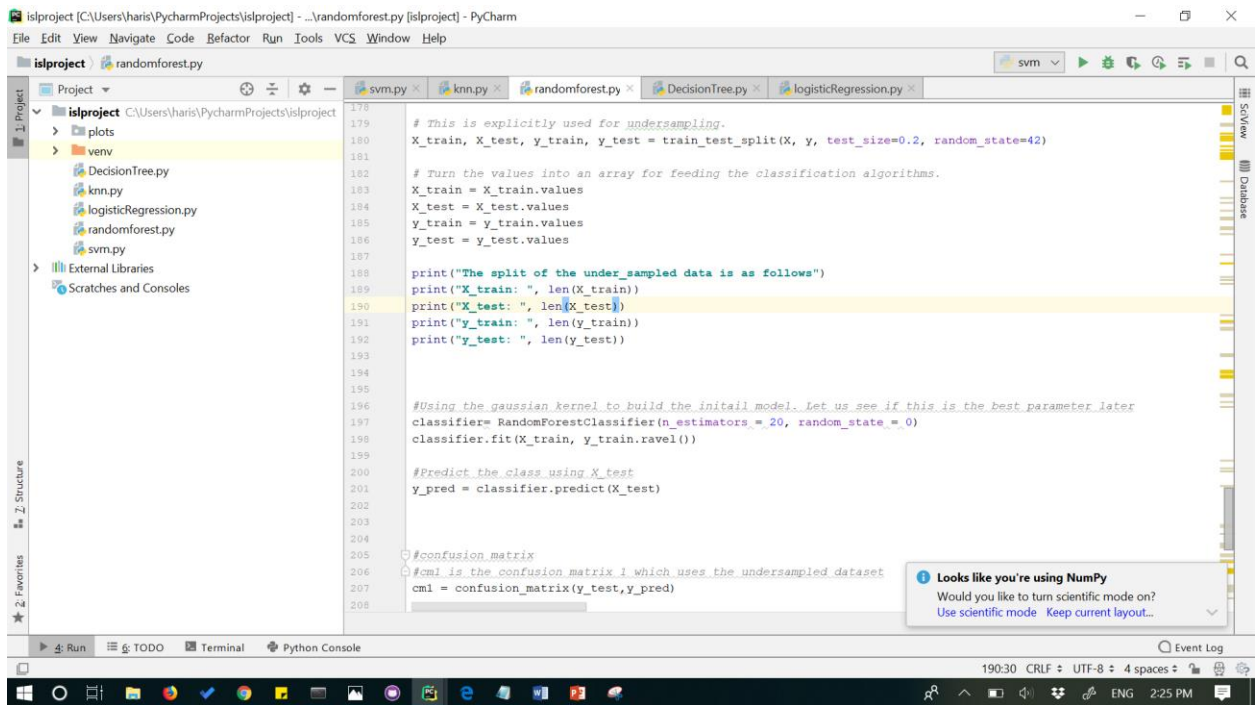
```

Run: knn x
y_test: 197
The accuracy is 92.89340101522842 %
The recall from the confusion matrix is 88.18181818181819 %
The mean accuracy in %: 93.27600616682895
The standard deviation in %: 2.7754246118397754
The accuracy of our model in % is between 90.50058155498917 and 96.0514307786687 %
Process finished with exit code 0

Confusion Matrix

	Predicted 0	Predicted 1
Actual 0	86	13
Actual 1	1	97

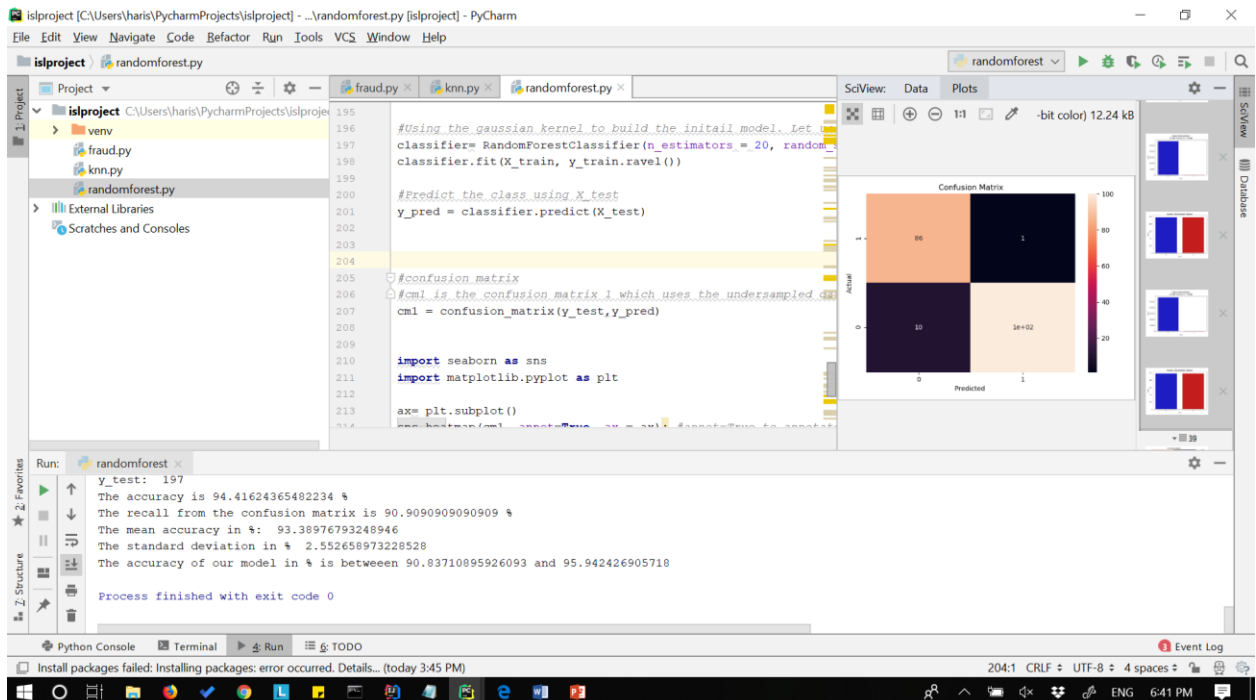
■ Random Forest Approach:



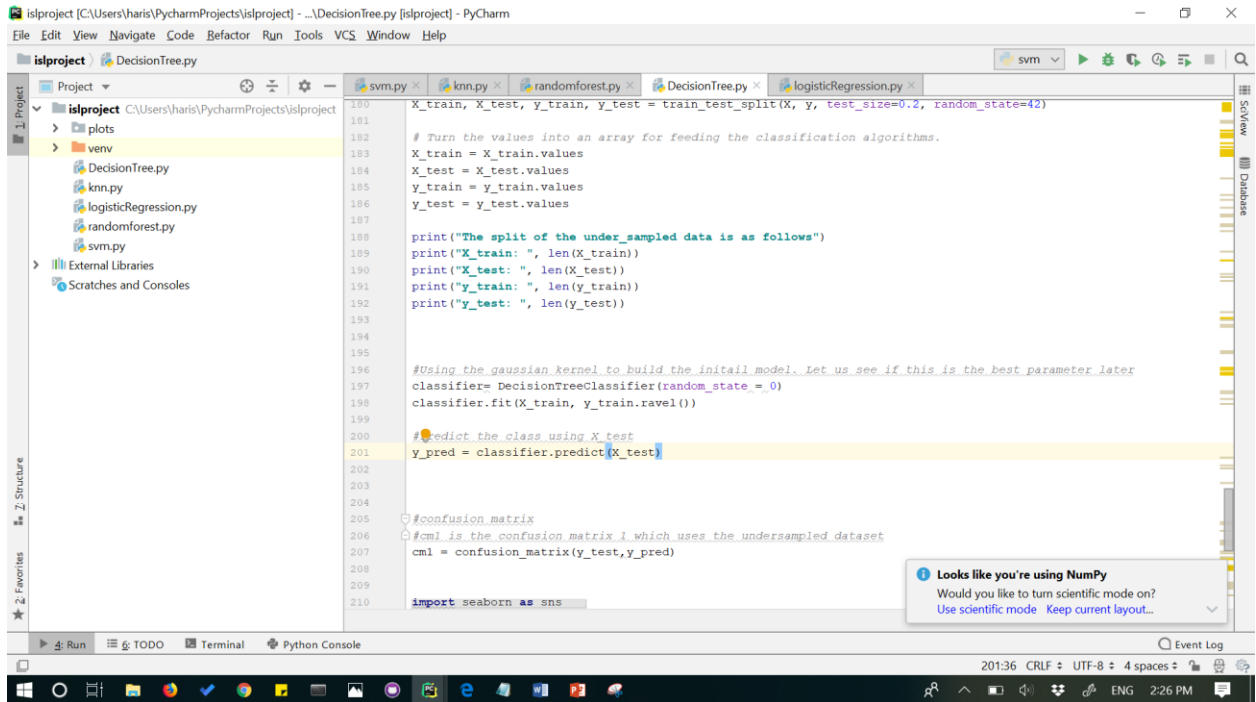
```
178
179 # This is explicitly used for undersampling.
180 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
181
182 # Turn the values into an array for feeding the classification algorithms.
183 X_train = X_train.values
184 X_test = X_test.values
185 y_train = y_train.values
186 y_test = y_test.values
187
188 print("The split of the under_sampled data is as follows")
189 print("X_train: ", len(X_train))
190 print("X_test: ", len(X_test))
191 print("y_train: ", len(y_train))
192 print("y_test: ", len(y_test))
193
194
195
196 #Using the gaussian kernel to build the initail model. Let us see if this is the best parameter later
197 classifier= RandomForestClassifier(n_estimators = 20, random_state = 0)
198 classifier.fit(X_train, y_train.ravel())
199
200 #Predict the class using X_test
201 y_pred = classifier.predict(X_test)
202
203
204
205 #confusion matrix
206 #cm1 is the confusion matrix 1 which uses the undersampled dataset
207 cm1 = confusion_matrix(y_test, y_pred)
208
```

Looks like you're using NumPy
Would you like to turn scientific mode on?
Use scientific mode Keep current layout...

Accuracy and Recall along with confusion matrix:



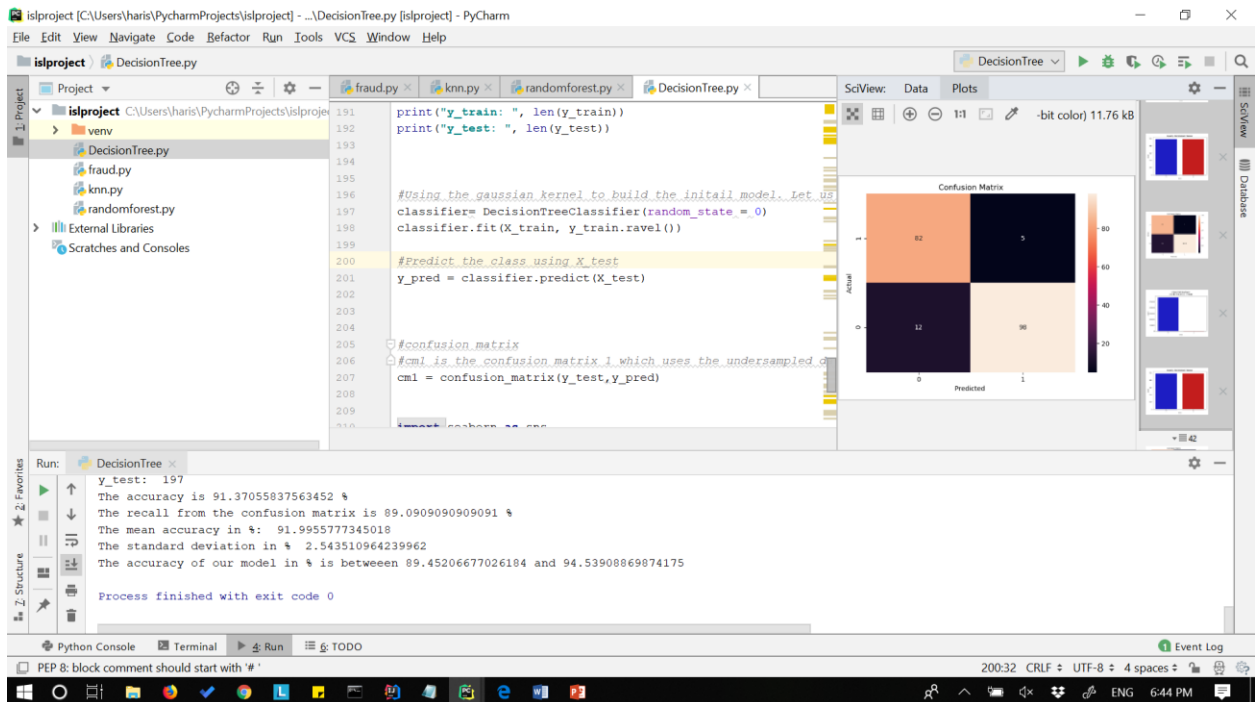
Decision Tree approach:



```
180 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
181
182 # Turn the values into an array for feeding the classification algorithms.
183 X_train = X_train.values
184 X_test = X_test.values
185 y_train = y_train.values
186 y_test = y_test.values
187
188 print("The split of the under_sampled data is as follows")
189 print("X_train: ", len(X_train))
190 print("X_test: ", len(X_test))
191 print("y_train: ", len(y_train))
192 print("y_test: ", len(y_test))
193
194
195
196 #Using the gaussian kernel to build the initail model. Let us see if this is the best parameter later
197 classifier= DecisionTreeClassifier(random_state=0)
198 classifier.fit(X_train, y_train.ravel())
199
200 #Predict the class using X_test
201 y_pred = classifier.predict(X_test)
202
203
204
205 #confusion matrix
206 #cml is the confusion matrix 1 which uses the undersampled dataset
207 cml = confusion_matrix(y_test,y_pred)
208
209
210 import seaborn as sns
```

Looks like you're using NumPy
Would you like to turn scientific mode on?
Use scientific mode Keep current layout...

Accuracy and Recall along with the confusion matrix:



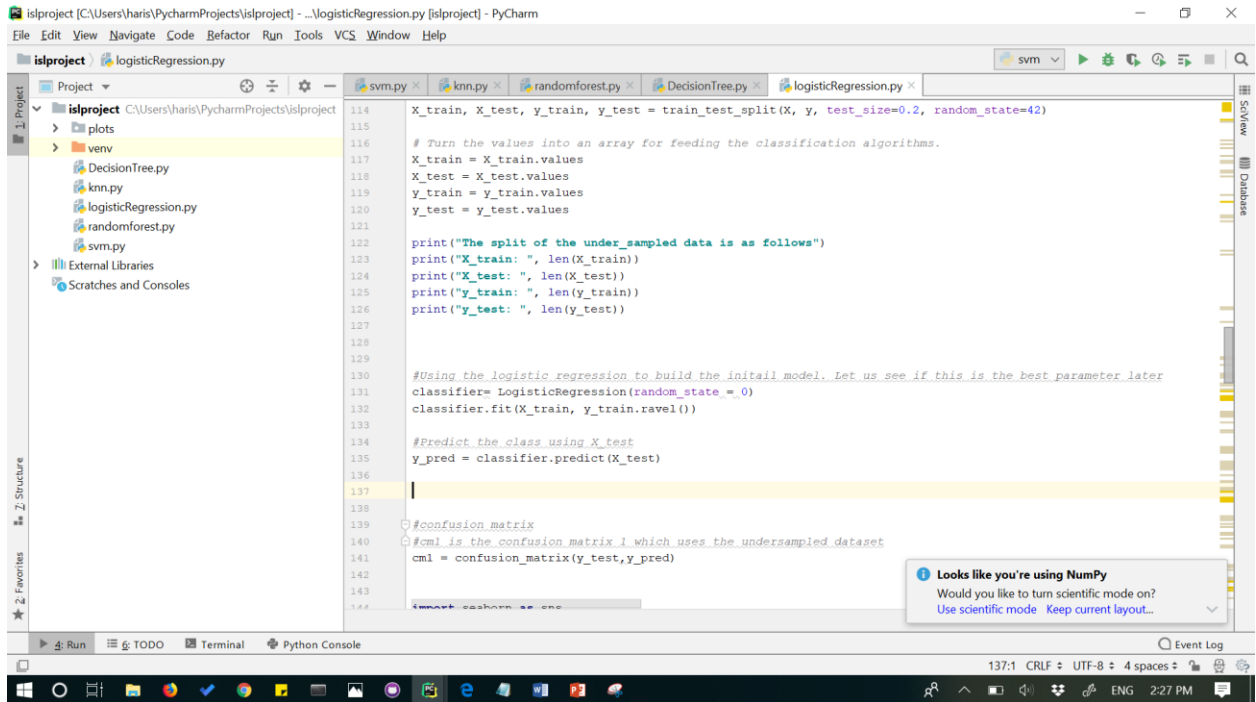
```
191 print("y_train: ", len(y_train))
192 print("y_test: ", len(y_test))
193
194
195
196 #Using the gaussian kernel to build the initail model. Let us
197 classifier= DecisionTreeClassifier(random_state=0)
198 classifier.fit(X_train, y_train.ravel())
199
200 #Predict the class using X_test
201 y_pred = classifier.predict(X_test)
202
203
204
205 #confusion matrix
206 #cml is the confusion matrix 1 which uses the undersampled d
207 cml = confusion_matrix(y_test,y_pred)
208
209
210 import seaborn as sns
```

Confusion Matrix

	0	1
0	82	5
1	12	36

Run: DecisionTree
y_test: 197
The accuracy is 91.37055837563452 %
The recall from the confusion matrix is 89.09090909090909 %
The mean accuracy in %: 91.9955777345018
The standard deviation in %: 2.543510964239962
The accuracy of our model in % is between 89.45206677026184 and 94.53908869874175
Process finished with exit code 0

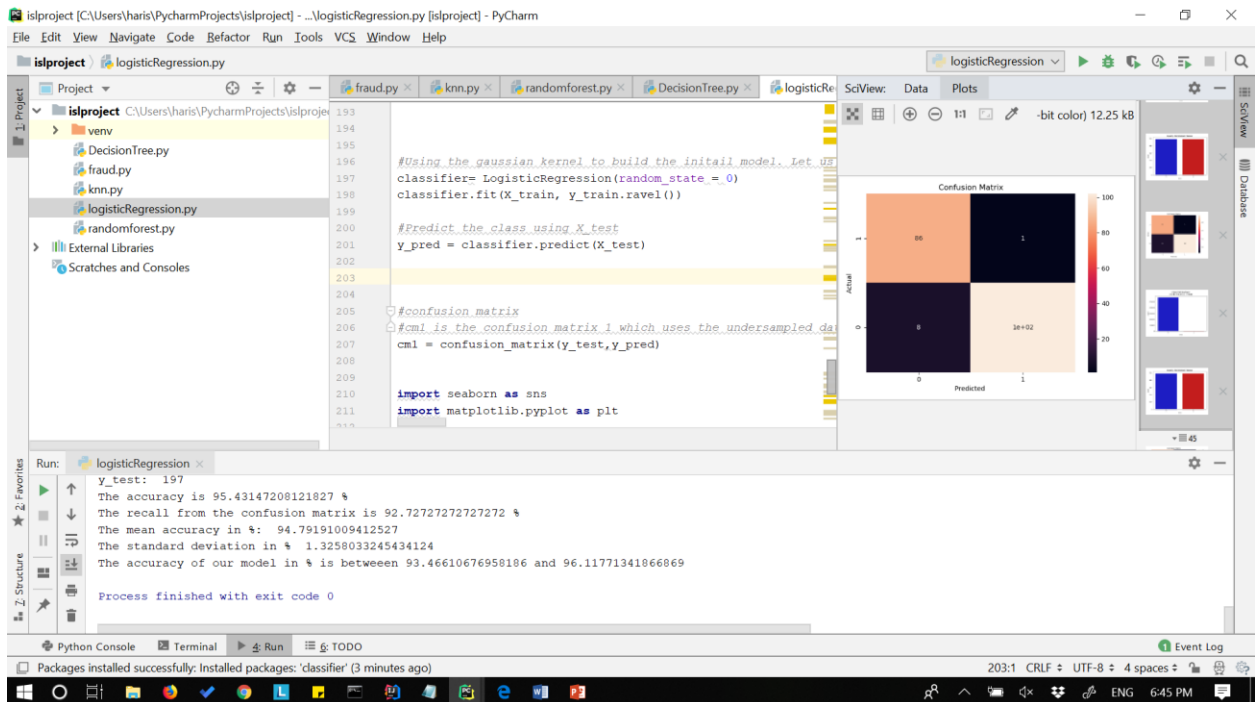
Logistic Regression approach:



```
114 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
115
116 # Turn the values into an array for feeding the classification algorithms.
117 X_train = X_train.values
118 X_test = X_test.values
119 y_train = y_train.values
120 y_test = y_test.values
121
122 print("The split of the under_sampled data is as follows")
123 print("X_train: ", len(X_train))
124 print("X_test: ", len(X_test))
125 print("y_train: ", len(y_train))
126 print("y_test: ", len(y_test))
127
128
129
130
131 #Using the logistic regression to build the initial model. Let us see if this is the best parameter later
132 classifier = LogisticRegression(random_state=0)
133 classifier.fit(X_train, y_train.ravel())
134
135 #Predict the class using X_test
136 y_pred = classifier.predict(X_test)
137
138
139
140 #confusion matrix
141 #cm1 is the confusion matrix 1 which uses the undersampled dataset
142 cm1 = confusion_matrix(y_test, y_pred)
143
144
```

Looks like you're using NumPy
Would you like to turn scientific mode on?
Use scientific mode Keep current layout...

Accuracy and recall using confusion matrix:



- Logistic Regression when performed with best parameters Using grid search:

```

165 # Use GridSearchCV to find the best parameters.
166 from sklearn.model_selection import GridSearchCV
167
168 # Logistic Regression
169 log_reg_params = {'penalty': ['l1', 'l2'], 'C': [0.001, 0.01, 0.1, 1, 10, 100, 1000]}
170
171
172
173
174
175
176
177
178 grid_log_reg = GridSearchCV(LogisticRegression(), log_reg_params)
179 grid_search=grid_log_reg.fit(X_train, y_train)
180 best_accuracy=grid_search.best_score_
181 print("The best accuracy using gridSearch is", best_accuracy)
182 # We automatically get the logistic regression with the best parameters.
183 log_reg = grid_log_reg.best_estimator_
184 # print("The best parameters for using this model is", log_reg)
185
186
187 #fitting the model with the best parameters
188 classifier_with_best_parameters = LogisticRegression(C=0.1, class_weight=None, dual=False, fit_intercept=True,
189 intercept_scaling=1, max_iter=100, multi_class='warn',
190 n_jobs=None, penalty='l1', random_state=None, solver='warn',
191 tol=0.0001, verbose=0, warm_start=False)
192 classifier_with_best_parameters.fit(X_train, y_train.ravel())
193
194 #predicting the Class
195 y_pred_best_parameters = classifier_with_best_parameters.predict(X_test)
196
197 #creating a confusion matrix
198 #cm is the confusion matrix which uses the best parameters
199 cm2 = confusion_matrix(y_test, y_pred_best_parameters)
200

```

Looks like you're using NumPy
Would you like to turn scientific mode on?
Use scientific mode Keep current layout...

Accuracy and recall along with confusion matrix:

```

206 ax= plt.subplot()
207 sns.heatmap(cm2, annot=True, ax= ax); #annot=True to annotate
208
209 # labels, title and ticks
210 ax.set_xlabel('Predicted ');ax.set_ylabel('Actual');
211 ax.set_title('Confusion Matrix2 using best parameters');
212 ax.xaxis.set_ticklabels(['0', '1']); ax.yaxis.set_ticklabels(['0', '1'])
213 plt.show()
214
215 print("The accuracy is " + str((cm2[1, 1] + cm2[0, 0]) / (cm2
216 print("The recall from the confusion matrix 2 is " + str(cm2[1, 1]
217
218 #
219 #
220 # Testing the model against the full dataset(skewed)
221 # creating a new dataset to test our model
222 # datanew= df.copy()
223 #
224 # Now to test the model with the whole dataset
225 #skewed_data=skewed_data.fit_transform(datanew)
226

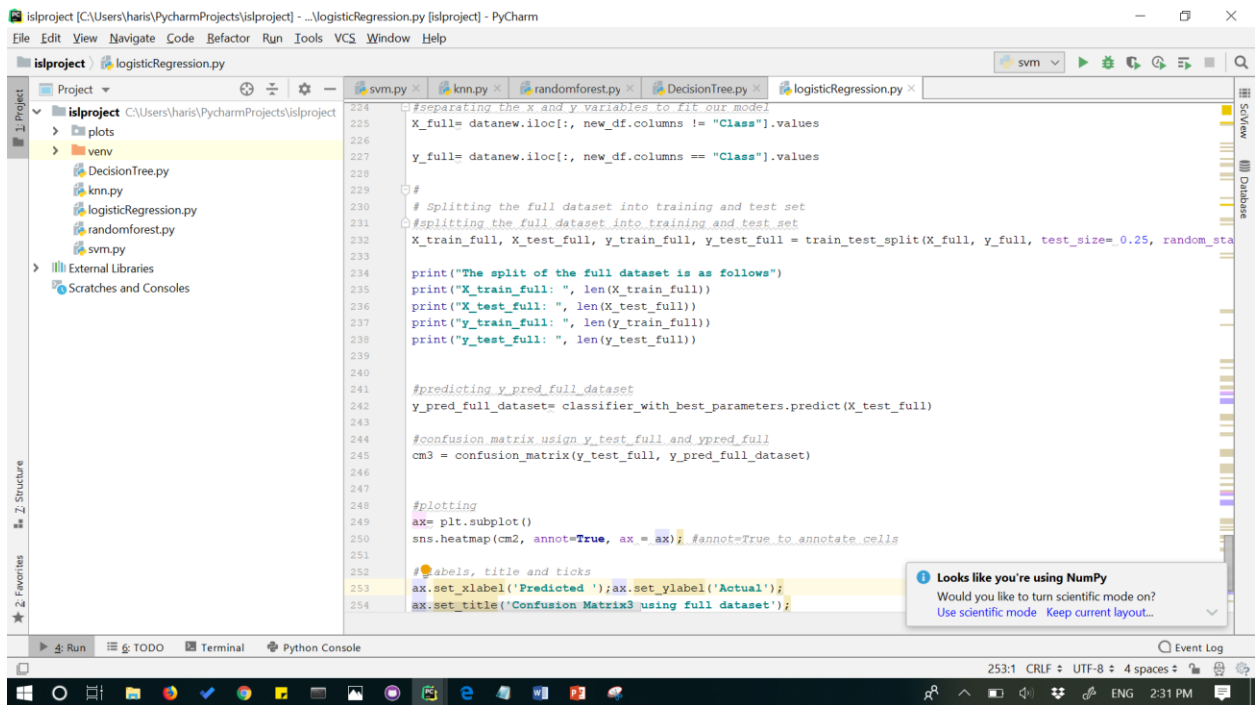
```

Run: logisticRegression
FutureWarning
C:\Users\haris\PycharmProjects\isl\venv\lib\site-packages\sklearn\linear_model\logistic.py:432: FutureWarning: Default solver will be changed to 'lbfgs' in 0.2
FutureWarning
The best accuracy using gridSearch is 0.9453621346886912
The accuracy is 94.41624365482234 %
The recall from the confusion matrix 2 is 91.81818181818183 %
Process finished with exit code 0

Confusion Matrix2 using best parameters

	Predicted 0	Predicted 1
Actual 0	95	2
Actual 1	9	38=92

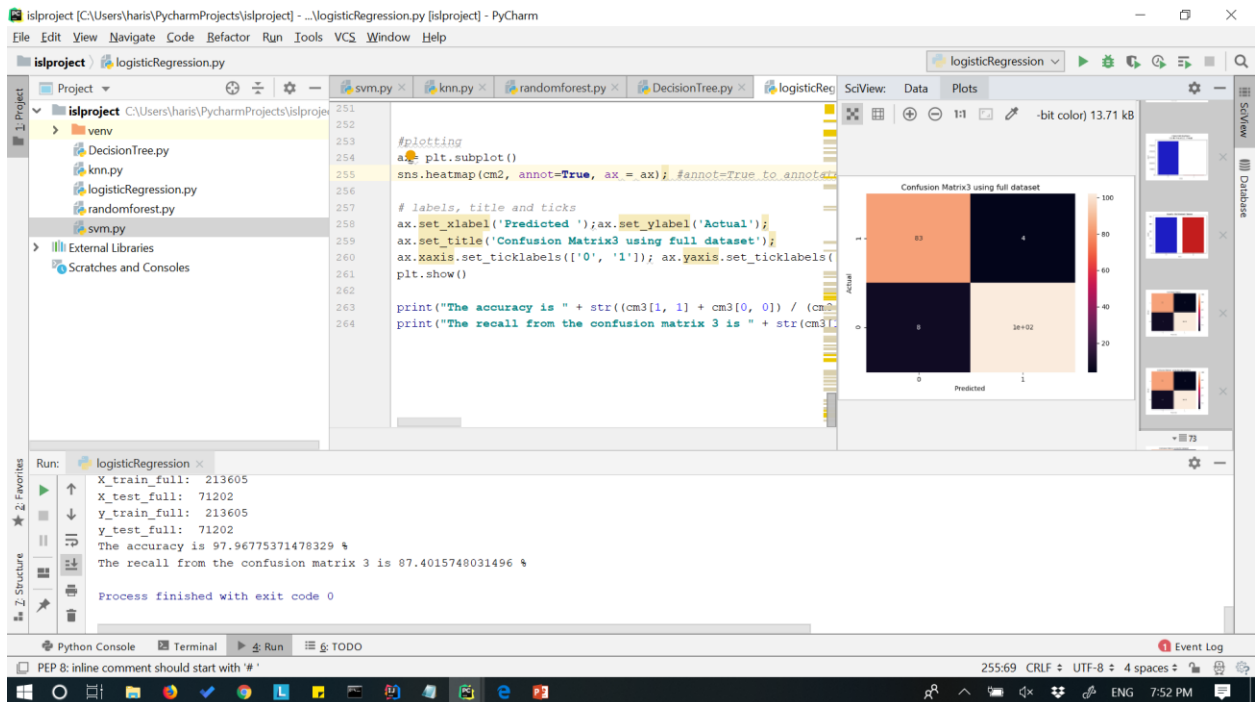
■ Logistic Regression with the test on the actual Dataset.



```
224 # separating the x and y variables to fit our model
225 X_full= datanew.iloc[:, new_df.columns != "Class"].values
226
227 y_full= datanew.iloc[:, new_df.columns == "Class"].values
228
229 #
230 # Splitting the full dataset into training and test set
231 # splitting the full dataset into training and test set
232 X_train_full, X_test_full, y_train_full, y_test_full = train_test_split(X_full, y_full, test_size= 0.25, random_state= 42)
233
234 print("The split of the full dataset is as follows")
235 print("X_train_full: ", len(X_train_full))
236 print("X_test_full: ", len(X_test_full))
237 print("y_train_full: ", len(y_train_full))
238 print("y_test_full: ", len(y_test_full))
239
240
241 # predicting y_pred_full dataset
242 y_pred_full_dataset= classifier_with_best_parameters.predict(X_test_full)
243
244 # confusion matrix using y_test_full and y_pred_full
245 cm3 = confusion_matrix(y_test_full, y_pred_full_dataset)
246
247
248 # plotting
249 ax= plt.subplot()
250 sns.heatmap(cm2, annot=True, ax= ax); #annot=True to annotate cells
251
252 # labels, title and ticks
253 ax.set_xlabel('Predicted '); ax.set_ylabel('Actual');
254 ax.set_title('Confusion Matrix3 using full dataset');
```

Looks like you're using NumPy
Would you like to turn scientific mode on?
Use scientific mode Keep current layout...

Accuracy and recall along with confusion matrix.



References:

1. <https://www.kaggle.com/metetik/classification-algorithms-comparison/notebook>
2. <https://www.kaggle.com/mlg-ulb/creditcardfraud>