creditanalysis

April 20, 2020

[1]:
```
#importing usefull liberary
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

```
/home/sanjukta/anaconda/lib/python3.7/site-
packages/statsmodels/tools/_testing.py:19: FutureWarning: pandas.util.testing is
deprecated. Use the functions in the public API at pandas.testing instead.
  import pandas.util.testing as tm
```

1 Data Preparation:

[2]:
```
df=pd.read_csv("credit_data.csv")
df.head()
```

[2]:
```
   age gender       education    occupation organization_type    seniority  \
0   19   Male        Graduate  Professional              None         None
1   18   Male  Under Graduate  Professional              None         None
2   29   Male  Under Graduate      Salaried              None        Entry
3   18   Male        Graduate       Student              None         None
4   26   Male   Post Graduate      Salaried              None  Mid-level 1

   annual_income  disposable_income house_type vehicle_type marital_status  \
0         186319              21625     Family         None        Married
1         277022              20442     Rented         None        Married
2         348676              24404     Rented         None        Married
3         165041               2533     Rented         None        Married
4         348745              19321     Rented         None        Married

   no_card  default
0        0        1
1        0        1
2        1        1
3        0        1
4        1        1
```

[3]:

```
print(df.shape)
print(df.describe())
```

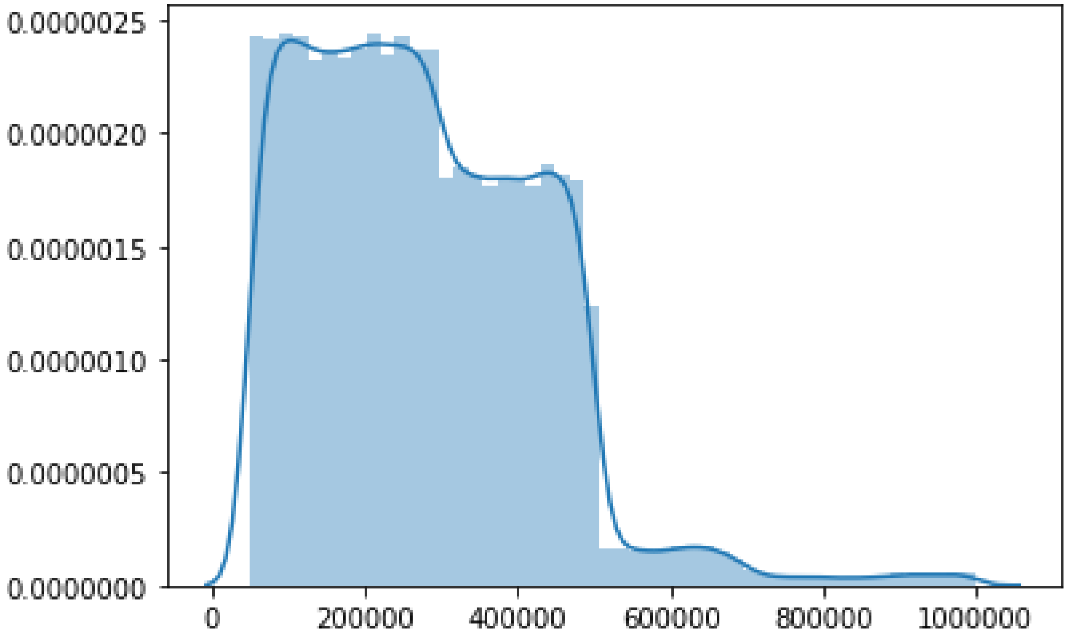
```
(50636, 13)
                age  annual_income  disposable_income       no_card  \
count  50636.000000   50636.000000       50636.000000  50636.000000
mean      29.527411  277243.989889       18325.788569      0.509815
std        8.816532  153838.973755       12677.864844      0.669883
min       18.000000   50000.000000        1000.000000      0.000000
25%       25.000000  154052.250000        8317.750000      0.000000
50%       27.000000  258860.500000       15770.000000      0.000000
75%       30.000000  385071.500000       24135.000000      1.000000
max       64.000000  999844.000000       49999.000000      2.000000

            default
count  50636.000000
mean       0.158425
std        0.365142
min        0.000000
25%        0.000000
50%        0.000000
75%        0.000000
max        1.000000
```

[4]:

```
# distribution of annual_income
amount = [df['annual_income'].values]
sns.distplot(amount)
```

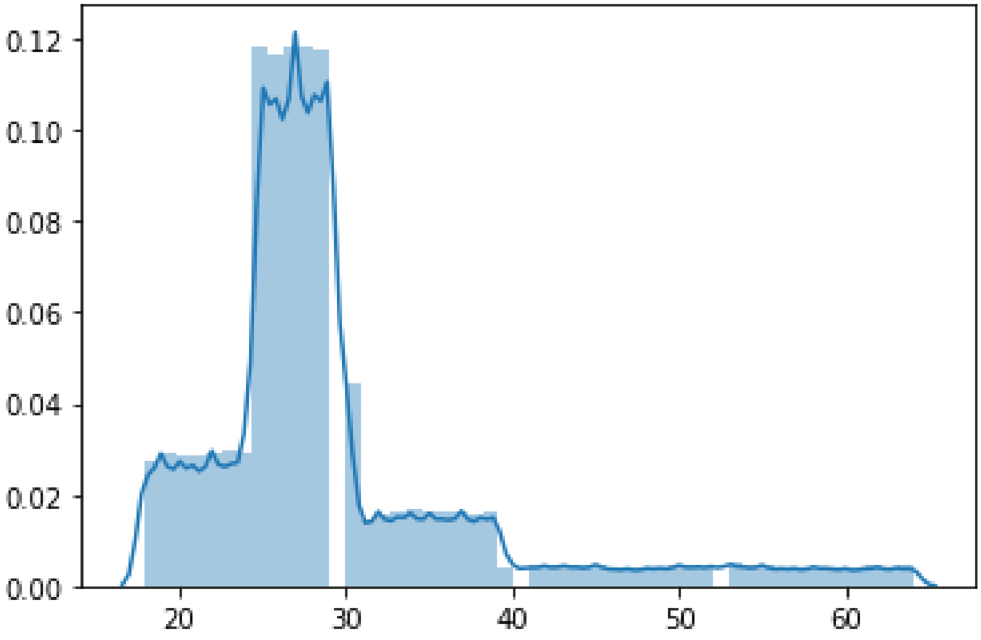
[4]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f19a7d8e2e8>
```



[5]:
```
# distribution of age
amount = [df['age'].values]
sns.distplot(amount)
```

[5]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f19a8436d68>
```



[6]:
```
'''from matplotlib import gridspec
# distribution of anomalous features
features = df.iloc[:,0:13].columns

plt.figure(figsize=(12,13*4))
gs = gridspec.GridSpec(13, 1)
for i, c in enumerate(df[features]):
    ax = plt.subplot(gs[i])
    sns.distplot(df[c][df.default == 1], bins=50)
    sns.distplot(df[c][df.default == 0], bins=50)
    ax.set_xlabel('')
    ax.set_title('histogram of feature: ' + str(c))
plt.show()'''
```

[6]:
```
"from matplotlib import gridspec\n# distribution of anomalous features\nfeatures
= df.iloc[:,0:13].columns\n\nplt.figure(figsize=(12,13*4))\ngs =
gridspec.GridSpec(13, 1)\nfor i, c in enumerate(df[features]):\n    ax =
plt.subplot(gs[i])\n    sns.distplot(df[c][df.default == 1], bins=50)\n
sns.distplot(df[c][df.default == 0], bins=50)\n    ax.set_xlabel('')\n
```

```
ax.set_title('histogram of feature: ' + str(c))\nplt.show()"
```

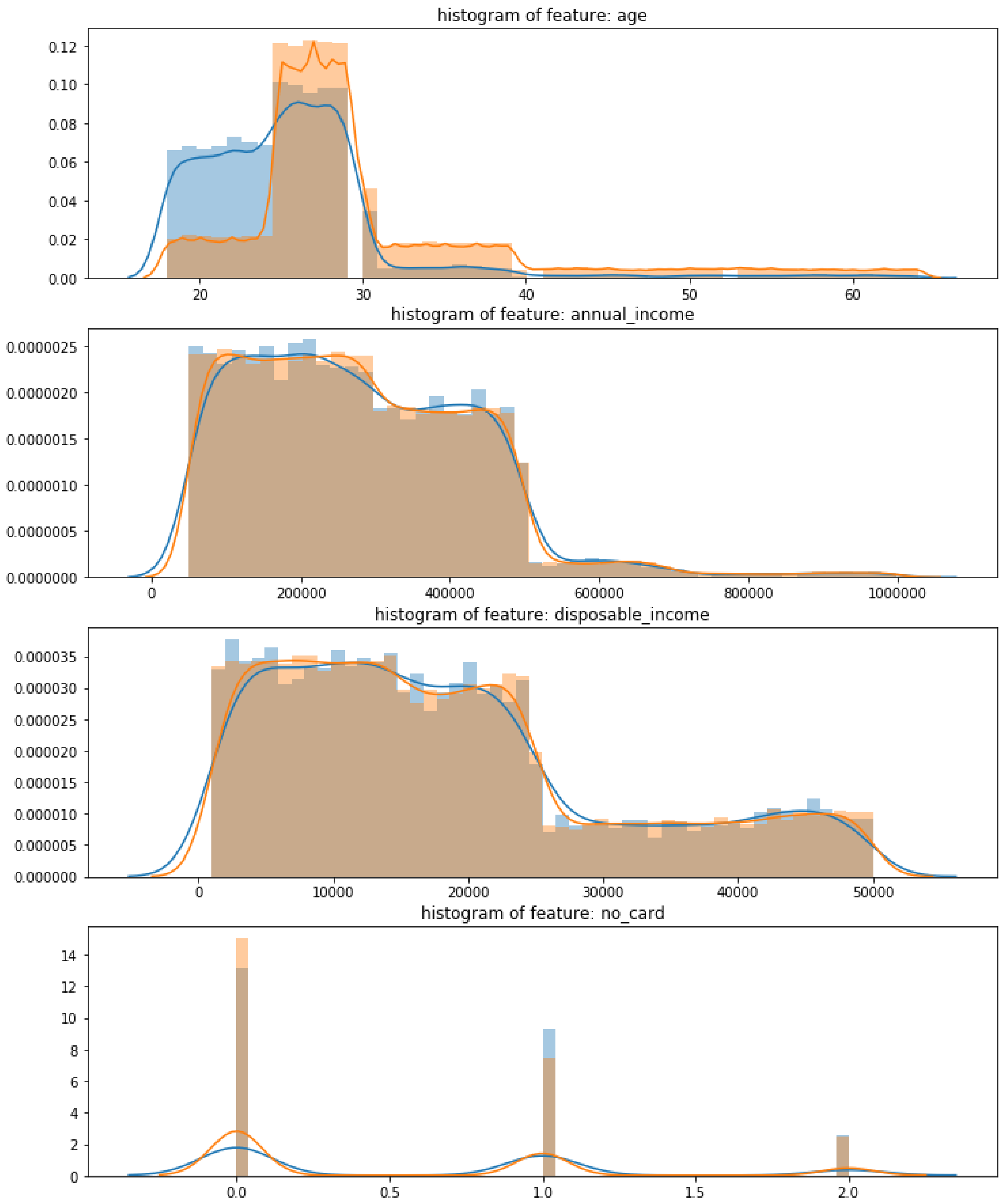
[7]:

```
df.rename(columns=lambda x: x.lower(), inplace =True)
#Base values: famale,other,student, none,none,Rented,none,other,

#Gender
df['Male'] = (df['gender']==1).astype('int')
df.drop('gender', axis = 1,inplace =True)

#Education
df['Graduate'] = (df['education']==1).astype('int')
df['Under Graduate'] = (df['education']==2).astype('int')
df['Post Graduate'] = (df['education']==3).astype('int')
df.drop('education', axis = 1,inplace =True)

#occupation
df['Professional'] = (df['occupation']==1).astype('int')
df['Salaried'] = (df['occupation']==2).astype('int')
df['Business'] = (df['occupation']==3).astype('int')
df.drop('occupation', axis = 1,inplace =True)

#Orgnization
df['Tier 1'] = (df['organization_type']==1).astype('int')
df['Tier 2'] = (df['organization_type']==2).astype('int')
df['Tier 3'] = (df['organization_type']==3).astype('int')
df.drop('organization_type', axis = 1,inplace =True)

#Seniority
df['Entry'] = (df['seniority']==1).astype('int')
df['Junior'] = (df['seniority']==2).astype('int')
df['Mid-level 1'] = (df['seniority']==3).astype('int')
df['Mid-level 2'] = (df['seniority']==4).astype('int')
df['Senior'] = (df['seniority']==5).astype('int')
df.drop('seniority', axis = 1,inplace =True)

#House Type
df['Rented'] = (df['house_type']==1).astype('int')
df['Owned'] = (df['house_type']==2).astype('int')
df['Family'] = (df['house_type']==3).astype('int')
df.drop('house_type', axis = 1,inplace =True)

#vehicle
df['Two Wheeler'] = (df['vehicle_type']==1).astype('int')
df['Four Wheeler'] = (df['vehicle_type']==2).astype('int')
df.drop('vehicle_type', axis = 1,inplace =True)

#Maritial status
```

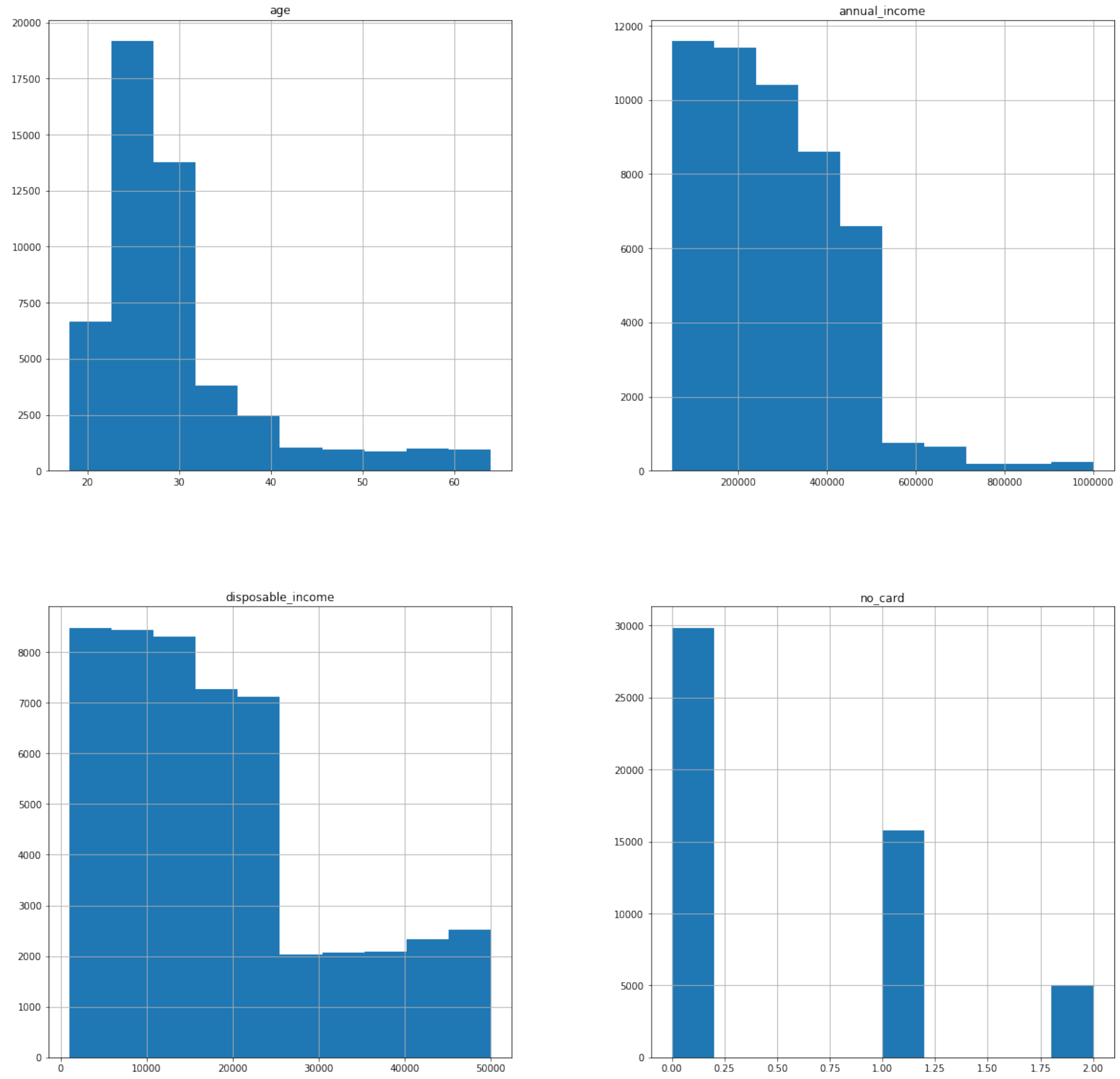
```
df['Married'] = (df['marital_status']==1).astype('int')
df['Single'] = (df['marital_status']==2).astype('int')
df.drop('marital_status', axis = 1,inplace =True)
```

[8]:

```
from matplotlib import gridspec
# distribution of anomalous features
features = ['age','annual_income','disposable_income','no_card']

plt.figure(figsize=(12,13*4))
gs = gridspec.GridSpec(13, 1)
for i, c in enumerate(df[features]):
    ax = plt.subplot(gs[i])
    sns.distplot(df[c][df.default == 1], bins=50)
    sns.distplot(df[c][df.default == 0], bins=50)
    ax.set_xlabel('')
    ax.set_title('histogram of feature: ' + str(c))
plt.show()
```



[9]:
```
# Plot histograms of each parameter
df[features].hist(figsize = (20, 20))
plt.show()
```



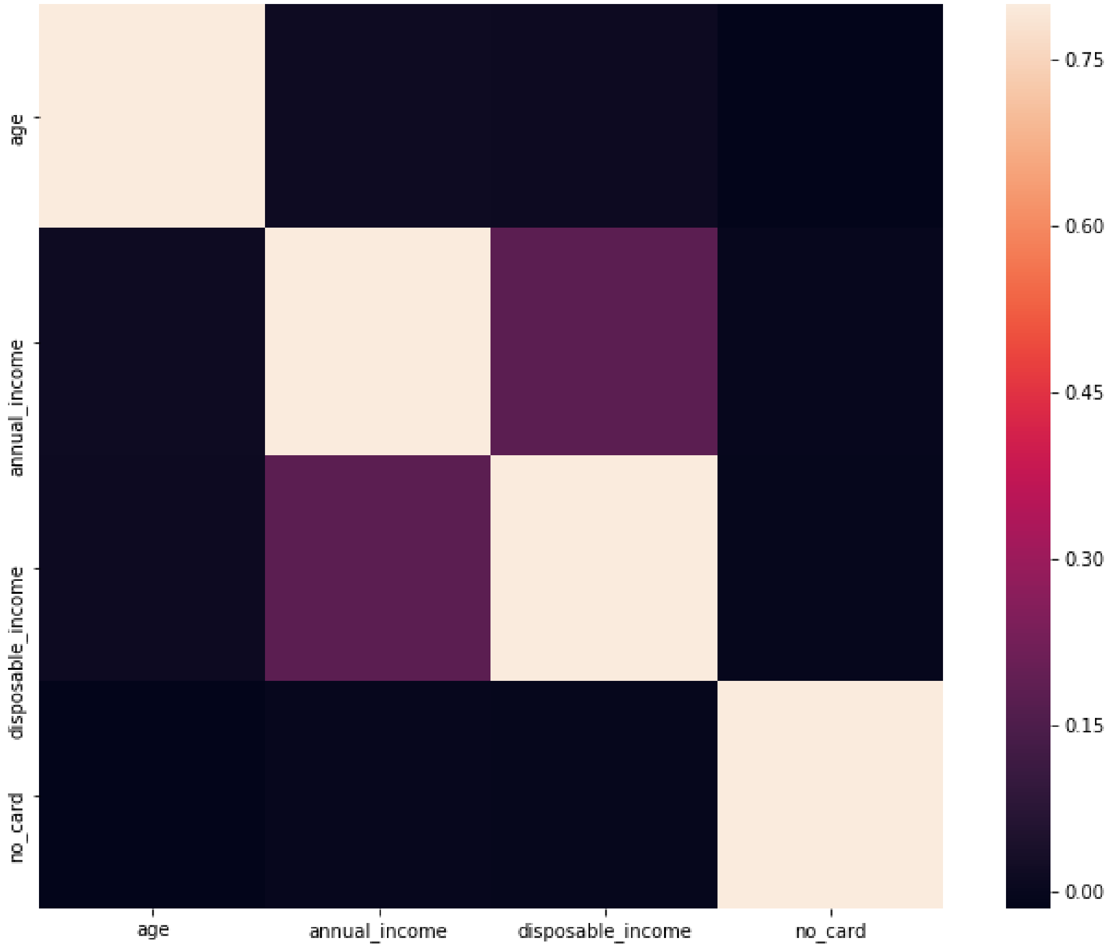
[10]:

```
# Determine number of fraud cases in dataset

Fraud = df[df['default'] == 1]
Valid = df[df['default'] == 0]

outlier_fraction = len(Fraud)/float(len(Valid))
print(outlier_fraction)

print('Fraud Cases: {}'.format(len(df[df['default'] == 1])))
print('Valid Transactions: {}'.format(len(df[df['default'] == 0])))
```

```
0.18824799361712113
Fraud Cases: 8022
Valid Transactions: 42614
```

[11]:

```
# Correlation matrix
corrmat = df[features].corr()
fig = plt.figure(figsize = (12, 9))

sns.heatmap(corrmat, vmax = .8, square = True)
plt.show()
```



[12]:

```
#seperating the X and the Y from the dataset
X=df.drop(['default'], axis=1)
Y=df["default"]
print(X.shape)
print(Y.shape)
#getting just the values for the sake of processing (its a numpy array with no␣
 ↪columns)
X_data=X.values
Y_data=Y.values
```

```
(50636, 26)
(50636,)
```

[13]:

```
X_data
```

[13]:

```
array([[    19, 186319,  21625, ...,      0,      0,      0],
       [    18, 277022,  20442, ...,      0,      0,      0],
       [    29, 348676,  24404, ...,      0,      0,      0],
       ...,
       [    25, 333840,  11636, ...,      0,      0,      0],
       [    29, 306053,  43751, ...,      0,      0,      0],
       [    25, 385157,  32684, ...,      0,      0,      0]])
```

[14]:

```
# Using Skicit-learn to split data into training and testing sets
from sklearn.model_selection import train_test_split
# Split the data into training and testing sets
X_train, X_test, Y_train, Y_test = train_test_split(X_data, Y_data, test_size =␣
 ↪0.2, random_state = 42)
```

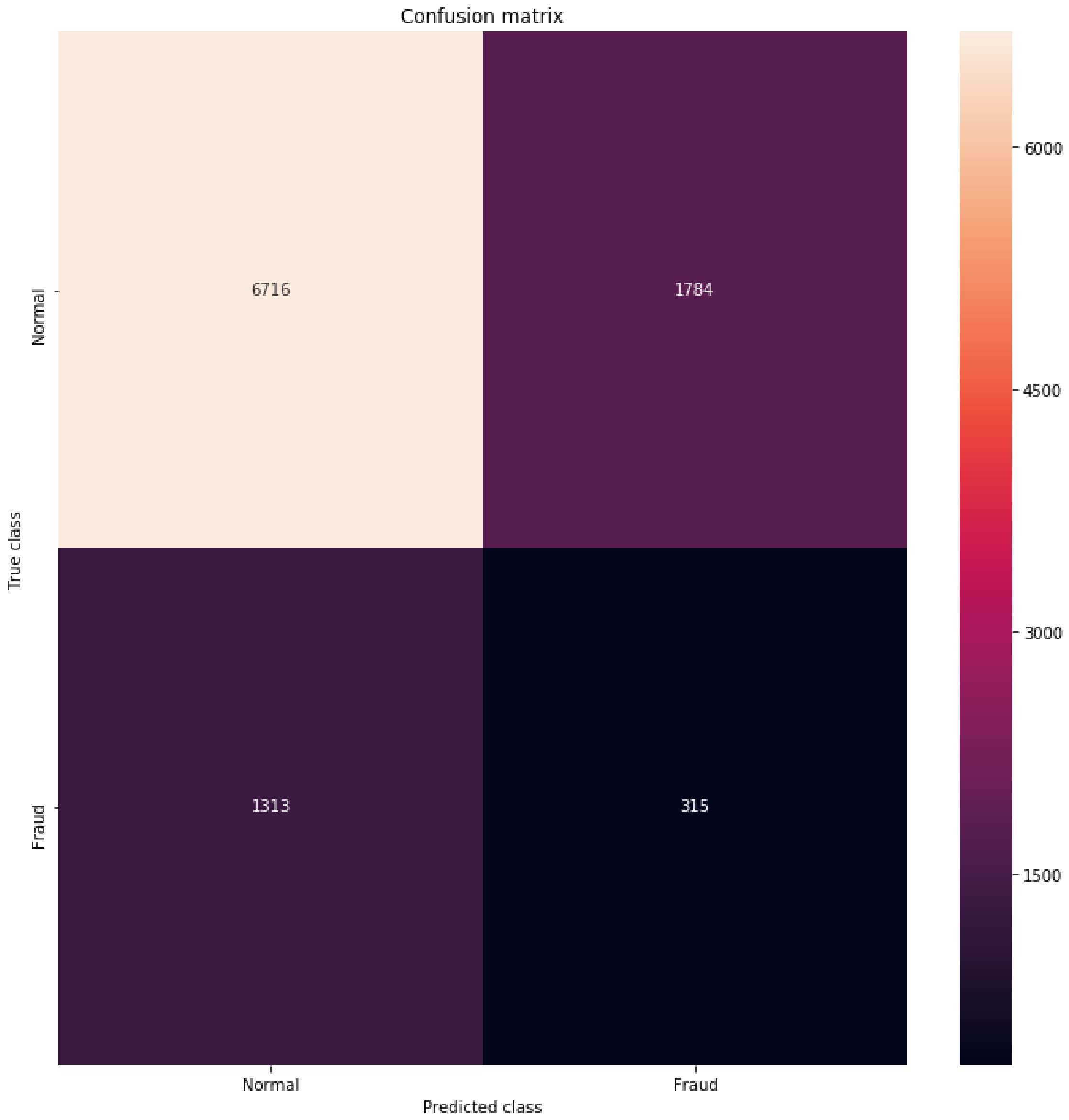
1.0.1 IsolationForest –> is a kind of algorithm to determine fraud detection

[15]:

```
#Building another model/classifier ISOLATION FOREST
from sklearn.ensemble import IsolationForest
from sklearn.metrics import classification_report,␣
 ↪accuracy_score,precision_score,recall_score,f1_score,matthews_corrcoef
from sklearn.metrics import confusion_matrix
ifc=IsolationForest(max_samples=len(X_train),
                    contamination=outlier_fraction,random_state=1)
ifc.fit(X_train)
scores_pred = ifc.decision_function(X_train)
y_pred = ifc.predict(X_test)


# Reshape the prediction values to 0 for valid, 1 for fraud.
y_pred[y_pred == 1] = 0
y_pred[y_pred == -1] = 1

n_errors = (y_pred != Y_test).sum()
#evaluation of the model
#printing every score of the classifier
#scoring in any thing
from sklearn.metrics import confusion_matrix
n_outliers = len(Fraud)
print("the Model used is {}".format("Isolation Forest"))
acc= accuracy_score(Y_test,y_pred)
print("The accuracy is  {}".format(acc))
prec= precision_score(Y_test,y_pred)
print("The precision is {}".format(prec))
```

```python
rec= recall_score(Y_test,y_pred)
print("The recall is {}".format(rec))
f1= f1_score(Y_test,y_pred)
print("The F1-Score is {}".format(f1))
MCC=matthews_corrcoef(Y_test,y_pred)
print("The Matthews correlation coefficient is{}".format(MCC))

#printing the confusion matrix
LABELS = ['Normal', 'Fraud']
conf_matrix = confusion_matrix(Y_test, y_pred)
plt.figure(figsize=(12, 12))
sns.heatmap(conf_matrix, xticklabels=LABELS,
            yticklabels=LABELS, annot=True, fmt="d");
plt.title("Confusion matrix")
plt.ylabel('True class')
plt.xlabel('Predicted class')
plt.show()

# Run classification metrics
plt.figure(figsize=(9, 7))
print('{}: {}'.format("Isolation Forest", n_errors))
print(accuracy_score(Y_test, y_pred))
print(classification_report(Y_test, y_pred))
```

```
the Model used is Isolation Forest
The accuracy is  0.6942140600315956
The precision is 0.1500714626012387
The recall is 0.1934889434889435
The F1-Score is 0.16903675878722832
The Matthews correlation coefficient is-0.014854874980789749
```



```
Isolation Forest: 3097
0.6942140600315956
              precision    recall  f1-score   support

           0       0.84      0.79      0.81      8500
           1       0.15      0.19      0.17      1628

    accuracy                           0.69     10128
   macro avg       0.49      0.49      0.49     10128
weighted avg       0.73      0.69      0.71     10128
```

```
<Figure size 648x504 with 0 Axes>
```

1.0.2 RandomForestClassifier

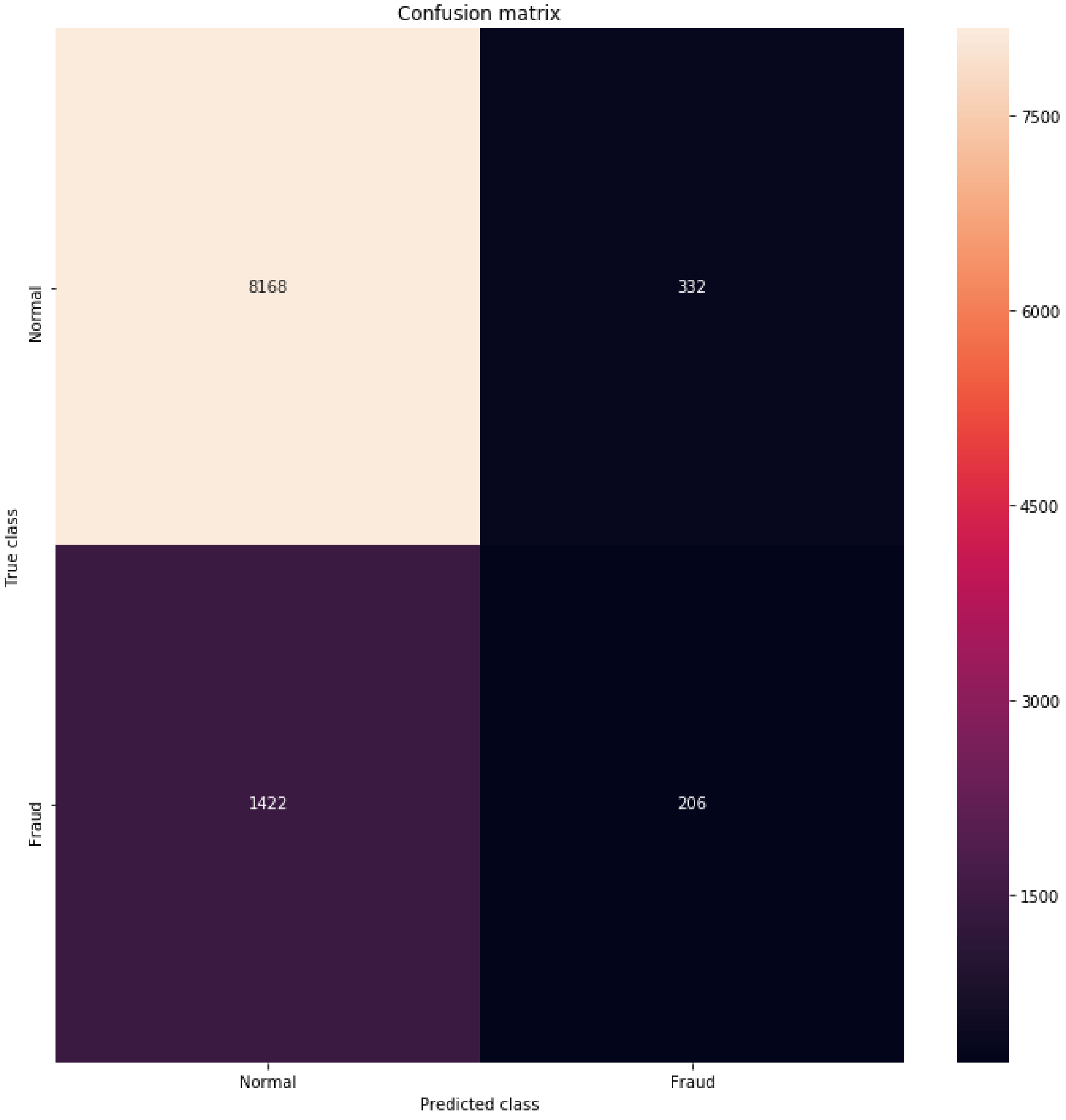
[16]:

```
# Building the Random Forest Classifier (RANDOM FOREST)
from sklearn.ensemble import RandomForestClassifier
# random forest model creation
rfc = RandomForestClassifier()
rfc.fit(X_train,Y_train)
# predictions
y_pred = rfc.predict(X_test)
```

[17]:

```
#Evaluating the classifier
#printing every score of the classifier
#scoring in any thing
from sklearn.metrics import classification_report,␣
 ↪accuracy_score,precision_score,recall_score,f1_score,matthews_corrcoef
from sklearn.metrics import confusion_matrix
n_outliers = len(Fraud)
n_errors = (y_pred != Y_test).sum()
print("The model used is Random Forest classifier")
acc= accuracy_score(Y_test,y_pred)
print("The accuracy is  {}".format(acc))
prec= precision_score(Y_test,y_pred)
print("The precision is {}".format(prec))
rec= recall_score(Y_test,y_pred)
print("The recall is {}".format(rec))
f1= f1_score(Y_test,y_pred)
print("The F1-Score is {}".format(f1))
MCC=matthews_corrcoef(Y_test,y_pred)
print("The Matthews correlation coefficient is {}".format(MCC))


#printing the confusion matrix
LABELS = ['Normal', 'Fraud']
conf_matrix = confusion_matrix(Y_test, y_pred)
plt.figure(figsize=(12, 12))
sns.heatmap(conf_matrix, xticklabels=LABELS, yticklabels=LABELS, annot=True,␣
 ↪fmt="d");
plt.title("Confusion matrix")
plt.ylabel('True class')
plt.xlabel('Predicted class')
plt.show()

# Run classification metrics
plt.figure(figsize=(9, 7))
print('{}: {}'.format("Isolation Forest", n_errors))
print(accuracy_score(Y_test, y_pred))
print(classification_report(Y_test, y_pred))
```

```
The model used is Random Forest classifier
The accuracy is  0.8268167456556083
The precision is 0.3828996282527881
The recall is 0.12653562653562653
The F1-Score is 0.1902123730378578
The Matthews correlation coefficient is 0.14326137802889452
```



```
Isolation Forest: 1754
0.8268167456556083
```

```
              precision    recall  f1-score   support

           0       0.85      0.96      0.90      8500
           1       0.38      0.13      0.19      1628

    accuracy                           0.83     10128
   macro avg       0.62      0.54      0.55     10128
weighted avg       0.78      0.83      0.79     10128
```

```
<Figure size 648x504 with 0 Axes>
```

1.0.3 DecisionTreeRegressor

[29]:

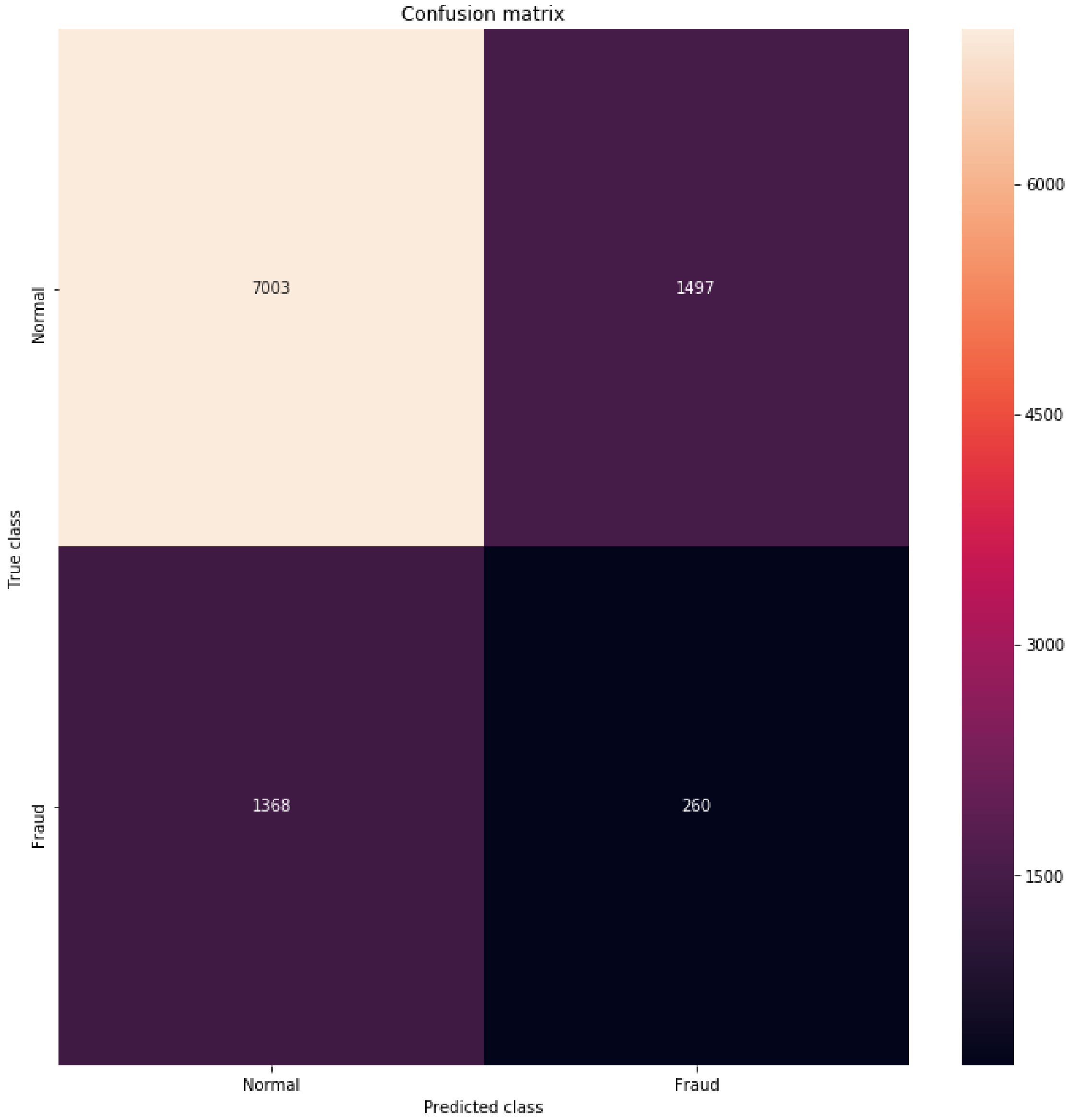
```python
# Building the Random Forest Classifier (Decission Tree)
#from sklearn.ensemble import DecisionTreeClassifier
from sklearn.tree import DecisionTreeRegressor
# Decission Tree model creation
rfc = DecisionTreeClassifier()
rfc.fit(X_train,Y_train)
# predictions
y_pred = rfc.predict(X_test)
```

[31]:

```python
#Evaluating the classifier
#printing every score of the classifier
#scoring in any thing
from sklearn.metrics import classification_report, accuracy_score,precision_score,recall_score,f1_score,matthews_corrcoef
from sklearn.metrics import confusion_matrix
n_outliers = len(Fraud)
n_errors = (y_pred != Y_test).sum()
print("The model used is Decission Tree classifier")
acc= accuracy_score(Y_test,y_pred)
print("The accuracy is  {}".format(acc))
prec= precision_score(Y_test,y_pred)
print("The precision is {}".format(prec))
rec= recall_score(Y_test,y_pred)
print("The recall is {}".format(rec))
f1= f1_score(Y_test,y_pred)
print("The F1-Score is {}".format(f1))
MCC=matthews_corrcoef(Y_test,y_pred)
print("The Matthews correlation coefficient is {}".format(MCC))


#printing the confusion matrix
LABELS = ['Normal', 'Fraud']
conf_matrix = confusion_matrix(Y_test, y_pred)
```

```
plt.figure(figsize=(12, 12))
sns.heatmap(conf_matrix, xticklabels=LABELS, yticklabels=LABELS, annot=True, fmt="d");
plt.title("Confusion matrix")
plt.ylabel('True class')
plt.xlabel('Predicted class')
plt.show()

# Run classification metrics
plt.figure(figsize=(9, 7))
print('{}: {}'.format("Isolation Forest", n_errors))
print(accuracy_score(Y_test, y_pred))
print(classification_report(Y_test, y_pred))
```

```
The model used is Decission Tree classifier
The accuracy is  0.7171208530805687
The precision is 0.14797951052931133
The recall is 0.1597051597051597
The F1-Score is 0.1536189069423929
The Matthews correlation coefficient is -0.015919759988362737
```



```
Isolation Forest: 2865
0.7171208530805687
              precision    recall  f1-score   support

           0       0.84      0.82      0.83      8500
           1       0.15      0.16      0.15      1628

    accuracy                           0.72     10128
   macro avg       0.49      0.49      0.49     10128
weighted avg       0.73      0.72      0.72     10128
```

```
<Figure size 648x504 with 0 Axes>
```

2 Conclusion

1. DecisionTreeRegressor (Accuracy) –> 72%
2. RandomForestClassifier (Accuracy) –> 83%
3. IsolationForest (Accuracy) –> 69%

[ ]:

```

```