Santander Customer Transaction Prediction

Chandini C

4th December 2019

Contents

Chapter 1

Introduction

1.1 Problem Statement

At Santander, mission is to help people and businesses prosper. We are always looking for ways to help our customers understand their financial health and identify which products and services might help them achieve their monetary goals. Our data science team is continually challenging our machine learning algorithms, working with the global data science community to make sure we can more accurately identify new ways to solve our most common challenge, binary classification problems such as: is a customer satisfied? Will a customer buy this product? Can a customer pay this loan? In this challenge, we need to identify which customers will make a specific transaction in the future, irrespective of the amount of money transacted.

1.2 Data

By looking at the problem statement, we can say that it is a binary classification problem. Here, our task is to build a classification model which will classify customers who will make transaction in future and who will not transact money based on the 'target' variable.

The data provided here has the same structure as the real data they have available to solve this problem. Models are evaluated on area under the ROC curve between the predicted probability and the observed target. The dataset is given in two files labeled "train.csv" and "test.csv", 200k observation and 202 variables each. Variables are ID_code, target, and var_0 to var_199. The data in test.csv does not have target variable therefore cannot be used in testing the model. The data in train.csv file much be split (80/20) for training and testing the model. A sample of data sets used is given below:

Table 1.1:A sample of top 5 rows in train data

ID_code	**target**	**var_0**	**var_1**	**var_2**	**var_3**	**var_4**	**var_5**	**var_6**	**var_7**	**...**	**var_190**	**var_191**	**var_192**	**var_193**	**var_194**	**var_195**	**var_196**	**var_197**	**var_198**	**var_199**
train_0	0	8.9255	-6.7863	11.9081	5.0930	11.4607	-9.2834	5.1187	18.6266	...	4.4354	3.9642	3.1364	1.6910	18.5227	-2.3978	7.8784	8.5635	12.7803	-1.0914
train_1	0	11.5006	-4.1473	13.8588	5.3890	12.3622	7.0433	5.6208	16.5338	...	7.6421	7.7214	2.5837	10.9516	15.4305	2.0339	8.1267	8.7889	18.3560	1.9518
train_2	0	8.6093	-2.7457	12.0805	7.8928	10.5825	-9.0837	6.9427	14.6155	...	2.9057	9.7905	1.6704	1.6858	21.6042	3.1417	-6.5213	8.2675	14.7222	0.3965

train_3	0	11.0604	-2.1518	8.9522	7.1957	12.5846	-1.8361	5.8428	14.9250	...	4.4666	4.7433	0.7178	1.4214	23.0347	-1.2706	-2.9275	10.2922	17.9697	-8.9996
train_4	0	9.8369	-1.4834	12.8746	6.6375	12.2772	2.4486	5.9405	19.2514	...	-1.4905	9.5214	-0.1508	9.1942	13.2876	-1.5121	3.9267	9.5031	17.9974	-8.8104

Table 1.2: sample of top 5 rows in test sample

ID_code	**var_0**	**var_1**	**var_2**	**var_3**	**var_4**	**var_5**	**var_6**	**var_7**	**var_8**	**...**	**var_190**	**var_191**	**var_192**	**var_193**	**var_194**	**var_195**	**var_196**	**var_197**	**var_198**	**var_199**
test_0	11.0656	7.7798	12.9536	9.4292	11.4327	-2.3805	5.8493	18.2675	2.1337	...	-2.1556	11.8495	-1.4300	2.4508	13.7112	2.4669	4.3654	10.7200	15.4722	-8.7197
test_1	8.5304	1.2543	11.3047	5.1858	9.1974	-4.0117	6.0196	18.6316	-4.4131	...	10.6165	8.8349	0.9403	10.1282	15.5765	0.4773	-1.4852	9.8714	19.1293	-20.9760
test_2	5.4827	-10.3581	10.1407	7.0479	10.2628	9.8052	4.8950	20.2537	1.5233	...	-0.7484	10.9935	1.9803	2.1800	12.9813	2.1281	-7.1086	7.0618	19.8956	-23.1794
test_3	8.5374	-1.3222	12.0220	6.5749	8.8458	3.1744	4.9397	20.5660	3.3755	...	9.5702	9.0766	1.6580	3.5813	15.1874	3.1656	3.9567	9.2295	13.0168	-4.2108
test_4	11.7058	-0.1327	14.1295	7.7506	9.1035	-8.5848	6.8595	10.6048	2.9890	...	4.2259	9.1723	1.2835	3.3778	19.5542	-0.2860	-5.1612	7.2882	13.9260	-9.1846

Chapter 2

METHODOLOGY

2.1 Data Pre-processing

In real world data are often unstructured, incomplete, inconsistent, lacking certain behaviour or having different data types and also tend to have many errors. It is important that we resolve these issues by making the data readable before feeding the data into the model. Here comes the data pre-processing part where this technique helps in transforming these raw data into understandable data format.

Raw data may contain incomplete datasets or missing values, unwanted or noisy attributes, outliers, etc. These issues may affect the model so this have to be removed before modeling. This process is also called as *exploratory data analysis*. Here we analyse the data with visualizations, count the number of 1's and 0's where 1 represent that the customer will make transaction in future and represent that the customer do not make transaction in future. Target variable consists of 89.95% of 0's and 10.05% of 1's in train data, which is an imbalanced data set. A pie chart in Fig.2.1(a) represents the target variable with 1's and 0's. (visualizations of distribution of columns per target class and code is given in the appendix)

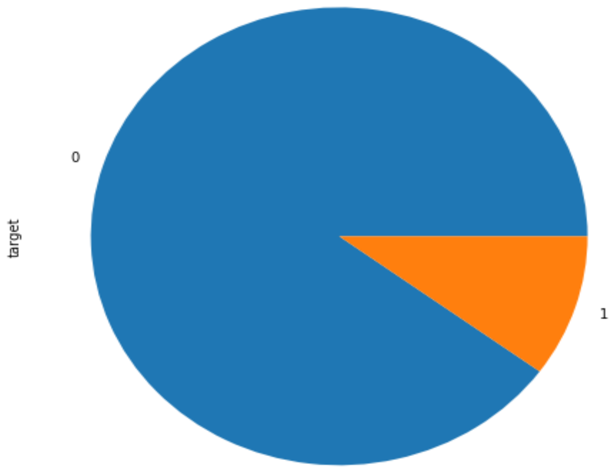


Fig.2.1(a) Pie chart representing target variable

Steps involved in data pre-processing:

2.1.1 Cleaning The data:

This step involves handling missing values, noisy data/ outliers, etc.

a) **Missing Value Analysis:**

A data may contain multiple missing values. If the percentage of missing value in a variable>30% then that variable is deleted as it does not contain any meaningful data to explain the problem statement.
Following steps can be performed to deal with these missing data:

- **Ignore tuple:**
  We can ignore the tuples only when there is a large data set with multiple missing values.
- **Fill the Missing Values:**
  There are several steps which helps to fill the missing values. We can fill the missing values manually by attribute mean, median or by using KNN.

Here, the given data does not contain any missing values, so we can move to next step.

b) Outlier Analysis:

A random variance in a measured variable is known as noisy data. These data also refers to outliers. This may be considered as an abnormal data but also can be used in fraud detection. Here, our aim is to know how many customers will make the transaction. So, we have to remove these outliers in order to get better accuracy. I chose boxplot analysis to check whether outliers are present in the data and by looking at the plot in Fig.2.1(b). It is clear that there are noisy data present in the data so we have to remove the outliers to get better model.

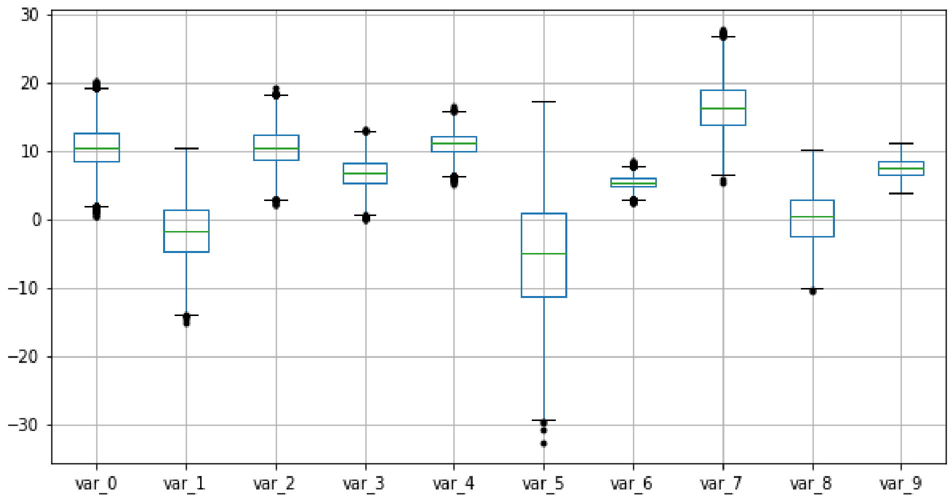


Fig.2.1(b) var_0 to 9 in train dataset after before removal of outliers

After removal of the outliers the data is less noisy and do not contain any unwanted data. Fig 2.1(c) shows the variable 0 to 9 after removing the outliers.

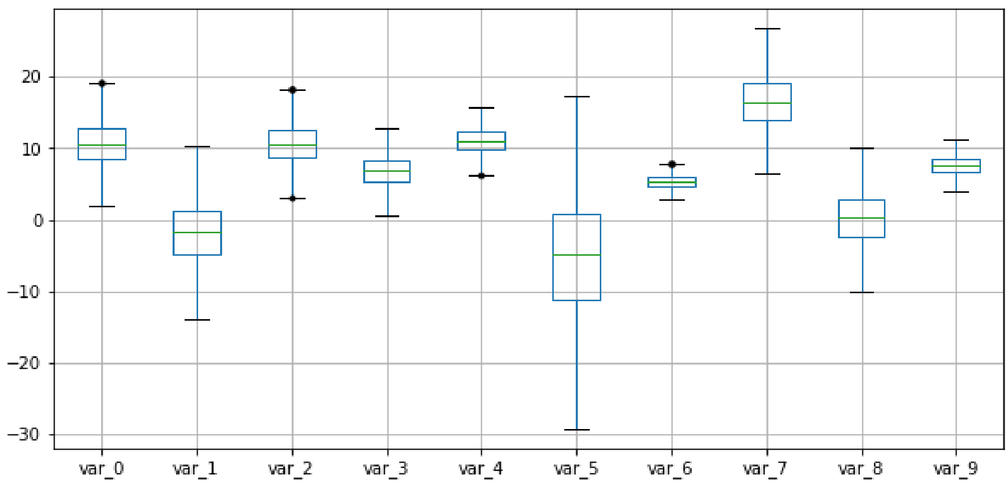


Fig.2.1(c) var_0 to 9 in train dataset after removal of outliers(see python and r code in appendix)

2.1.2 Feature Selection

Feature selection or attribute selection is a process of selecting a subset of relevant features(variable predictors) for use in model construction. As we are dealing with numeric data correlation analysis is applied on the train data. Fig. 2.1.2 depicts the correlation plot. After getting correlation matrix we can observe that the max and min values of correlation plot is 0.009, -0.01 and also we can observe from correlation distribution plot that the correlation between the train and test attributes is very small, it means that features are independent each other so we can ignore it. Now, the data is good to go for next step and there is no need for the correlation analysis on the data.

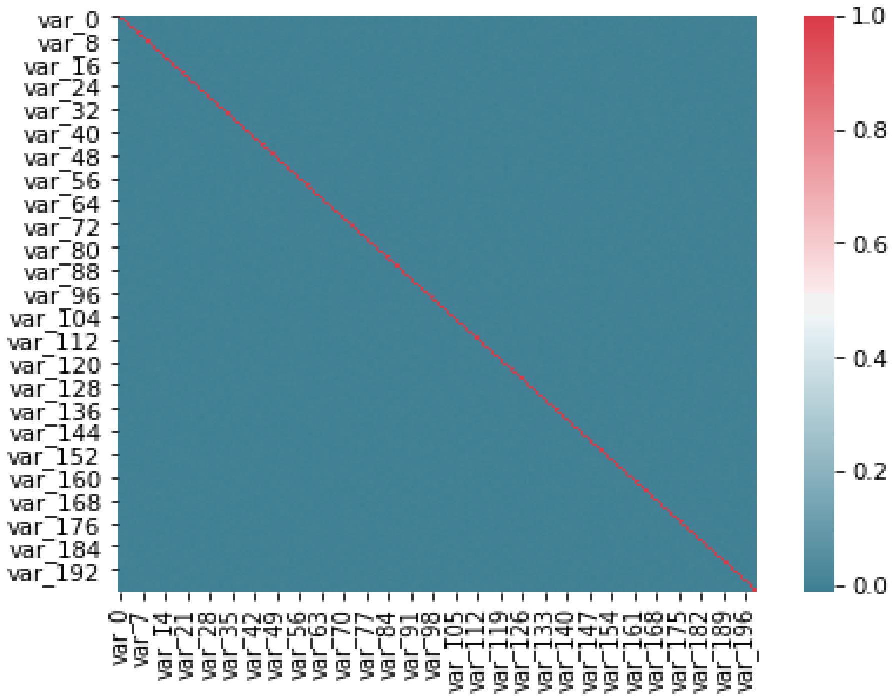


Fig.2.1.2 correlation plot(see appendix for python code)

2.1.3 Feature Scaling

Feature scaling is a process of reducing unwanted variations within the variables and limit the range of variable on common ground. There are two types of feature scaling techniques:

a) Normalization: Rescales the range of values between [0,1]
b) Standardization: Rescales the data to have a mean of 0 or standard deviation of 1 unit variance

After performing outlier analysis we can say that the variation within the variables have been reduced and Fig. 2.1.3 in appendix shows that most of the variables are well distributed as there is no much variation in the data and there is no use of normalization or standardization techniques for the given data set.

Histogram plots for mean frequency, median frequency, standard deviation frequency, skewness and other inferential plots can be drawn from this. (These plots can be viewed in appendix)

2.2 Modeling

Before feeding data to the model it is necessary that we split the data. We are provided with two data sets train and test, where train data consists of the target variable. So, we have to split train data with test_size=0.3(70% of train data and 30% of test data from train dataset). This is don using stratified sampling method.

2.2.1 Model Selection:

While exploring the data(target variable with classes 1's and 0's) and looking at the problem statement we can conclude that it is a binary classification problem, with binary target variables. As it is a classification problem we need to train the data using train data and predict the test cases on the model. As we are provided with two data sets train and test, we use train data to train the model as it contains target variable and it will be easy to choose the best performing model to predict test cases.

You always start your model building from the most simplest to more complex. Therefore, we use Random Forest.

2.2.2 Decision tree

```
#Decision Tree
#Replace target categories with Yes or No
train['target']=train['target'].replace(0,'No')

#apply on train data
c50_model=tree.DecisionTreeClassifier(criterion='entropy').fit(X_train,
Y_train)

#Apply on test data
Y_pred=c50_model.predict(X_test)
```

```
#create confusion matrix
CM=confusion_matrix(Y_test,Y_pred)
CM=pd.crosstab(Y_test,Y_pred)

#let us save TP,TN,FN,FP
TN=CM.iloc[0,0]
FP=CM.iloc[1,0]
TP=CM.iloc[1,1]
FN=CM.iloc[0,1]
print(CM)
```

col_0 target	0	1
0	43162	4229
1	4152	979

```
#ROC_AUC score
roc_score_dt = np.round(roc_auc_score(Y_test, Y_pred),2)
print('ROC score :',roc_score_dt)
#ROC_AUC curve
plt.figure()
false_positive_rate,recall,thresholds=roc_curve(Y_test,Y_pred)
roc_auc_dt=auc(false_positive_rate,recall)
plt.title('Reciver Operating Characteristics(ROC)')
plt.plot(false_positive_rate,recall,'b',label='ROC(area=%0.3f)' %roc_auc_dt
)
plt.legend()
plt.plot([0,1],[0,1],'r--')
plt.xlim([0.0,1.0])
plt.ylim([0.0,1.0])
plt.ylabel('Recall(True Positive Rate)')
plt.xlabel('False Positive Rate')
plt.show()
print('AUC:',roc_auc_dt)
```

ROC score : 0.55

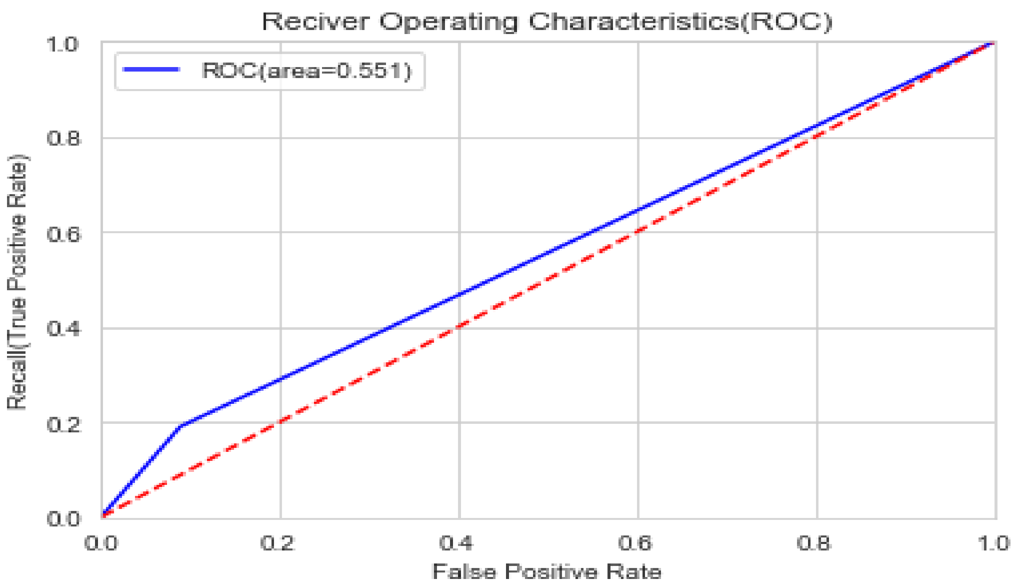


Fig.2.2.2 AUC-ROC curve for Decision Tree

AUC: 0.5507823302768308

AUC score obtained from the AUC-ROC curve shown in Fig.2.2.2 is very less and the performance of the model is poor even though accuracy of the model is 84%. AUC score should be close to 1 then the model performs well if it is close to 0.5 it will not be able to classify between 1's and 0's.

Similarly we try for Random Forest we get a good accuracy of 90.22% but AUC score is ~=0.5(visualization of the curve and code is given in the appendix)

2.2.3 Logistic Regression

```
#Logistic Regression
lrg = LogisticRegression(random_state=42)
lrg.fit(X_train, Y_train)
y_pred_lrg = lrg.predict(X_test)
#Cross validation prediction
cv_predict=cross_val_predict(lrg,X_test,Y_test,cv=5)
#Cross validation score
cv_score=cross_val_score(lrg,X_test,Y_test,cv=5)
print('cross_val_score :',np.average(cv_score))
#ROC_AUC score
roc_score_lrg=roc_auc_score(Y_test,cv_predict)
print('ROC score :',roc_score_lrg)
#ROC_AUC curve
plt.figure()
false_positive_rate,recall,thresholds=roc_curve(Y_test,cv_predict)
roc_auc_lrg=auc(false_positive_rate,recall)
plt.title('Reciver Operating Characteristics(ROC)')
plt.plot(false_positive_rate,recall,'b',label='ROC(area=%0.3f)' %roc_au
c_lrg)
plt.legend()
plt.plot([0,1],[0,1],'r--')
```

```
plt.xlim([0.0,1.0])
plt.ylim([0.0,1.0])
plt.ylabel('Recall(True Positive Rate)')
plt.xlabel('False Positive Rate')
plt.show()
print('AUC:',roc_auc_lrg)
```

ROC score : 0.6244763923405998

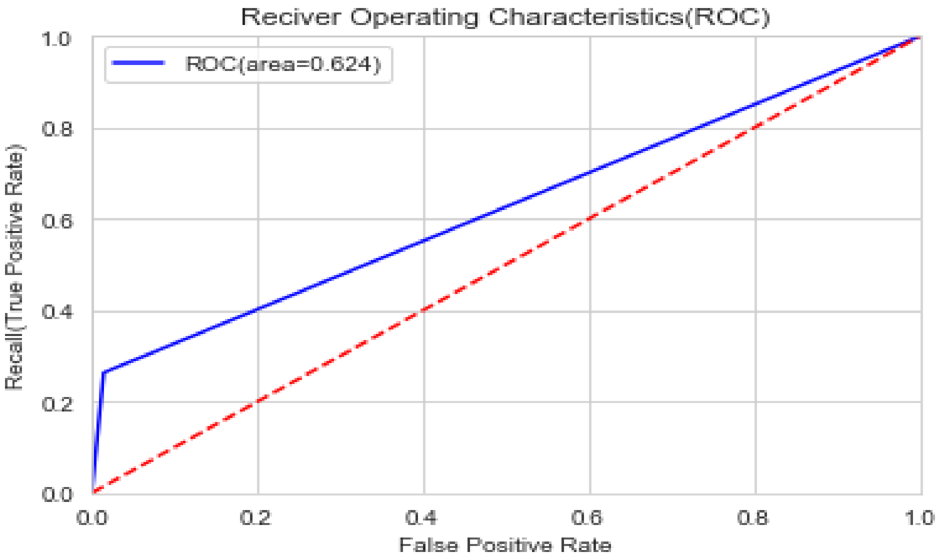


Fig.2.2.3 AUC-ROC for logistic regression

AUC: 0.6244763923405998

Though we get an accuracy of 92% it's AUC score is not up to the mark for the model to perform well. Therefore, this model also fails to get a good performance score.

Similarly, we obtain an accuracy of 92% and an AUC score of 0.67 while using NaiveBayes algorithm.(see appendix for code in r and python with its AUC-ROC curve)

2.2.4 Synthetic Minority Oversampling Technique(SMOTE)

As the data is imbalanced precision and recall metrics are biased towards the majority class(i.e., 0's) gives more precision value to 0's and lesser value to 1's. So there is a necessity to balance the data set. SMOTE uses a nearest neighbors algorithm to generate new and synthetic data to use for training the model. This technique helps to balance the data and hence we get a better AUC score. We have implemented Logistic Regression on SMOTE. Below code explains the logistic regression model using SMOTE.

```
#SMOTE
#Synthetic Minority Oversampling Technique
sm = SMOTE(random_state=42, ratio=1.0)
#Generating synthetic data points
```

```
X_smote,y_smote=sm.fit_sample(X_train,Y_train)
X_smote_v,y_smote_v=sm.fit_sample(X_test,Y_test)
#Logistic regression model for SMOTE
smote=LogisticRegression(random_state=42)
#fitting the smote model
smote.fit(X_smote,y_smote)
#Accuracy of the model
smote_score=smote.score(X_smote,y_smote)
print('Accuracy of the smote_model :',smote_score)
```

Accuracy of the smote_model : 0.7985422186852839

```
#Cross validation prediction
cv_pred=cross_val_predict(smote,X_smote_v,y_smote_v,cv=5)
#Cross validation score
cv_score=cross_val_score(smote,X_smote_v,y_smote_v,cv=5)
print('cross_val_score :',np.average(cv_score))
```

```
cross_val_score : 0.7970819415096922
```

```
#Confusion matrix
cm=confusion_matrix(y_smote_v,cv_pred)
cm=pd.crosstab(y_smote_v,cv_pred)
cm
```

col_0	0	1
row_0		
0	37392	9999
1	9234	38157

```
#Classification report
scores=classification_report(y_smote_v,cv_pred)
print(scores)
```

```
              precision    recall  f1-score   support

           0       0.80      0.79      0.80     47391
           1       0.79      0.81      0.80     47391

    accuracy                           0.80     94782
   macro avg       0.80      0.80      0.80     94782
weighted avg       0.80      0.80      0.80     94782
```

```
#ROC_AUC score
roc_score=roc_auc_score(y_smote_v,cv_pred)
print('ROC score :',roc_score)
```

```
#ROC_AUC curve
plt.figure()
false_positive_rate,recall,thresholds=roc_curve(y_smote_v,cv_pred)
roc_auc=auc(false_positive_rate,recall)
plt.title('Reciver Operating Characteristics(ROC)')
plt.plot(false_positive_rate,recall,'b',label='ROC(area=%0.3f)' %roc_auc)
plt.legend()
plt.plot([0,1],[0,1],'r--')
plt.xlim([0.0,1.0])
plt.ylim([0.0,1.0])
plt.ylabel('Recall(True Positive Rate)')
plt.xlabel('False Positive Rate')
plt.show()
print('AUC:',roc_auc)
```

ROC score : 0.7970817243780465

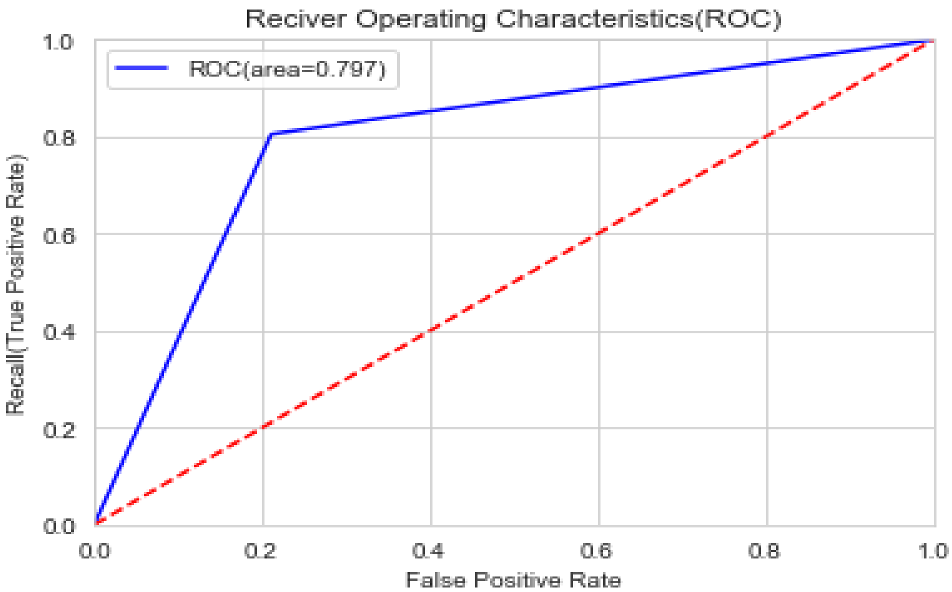


Fig.2.2.4 AUC-ROC curve using SMOTE(logistic regression model)

AUC: 0.7970817243780465

AUC score of SMOTE(logistic regression model) is better when compared to other models but the accuracy is little less compared to other models.

Chapter 3

Conclusion

3.1 Model Evaluation Metric

As we have done few models, now we have to decide which model to choose. There are multiple metrics on which we select the model. Few of the model evaluation metrics are given below:

1. Accuracy

2. Precision

3. Recall

4. AUC

AUC is an important metric for checking any classification model's performance as it tells how much model is capable of distinguishing between the classes. Higher the AUC, better the model is at predicting 0's as 0's and 1's as 1's. Using Confusion Matrix we can manually find accuracy and recall.

3.2 Model Selection

Accuracy of the model is not the best metric to use when evaluating the imbalanced datasets as it may be misleading. So, we are going to change the performance metric to AUC as it is one of the best evaluation metric for classification model. As SMOTE(Logistic Regression Model) has the highest AUC score, we can use this model for predicting the test dataset.

```
#Predicting the model
X_test=test.drop(['ID_code'],axis=1)
smote_pred=smote.predict(X_test)
print(smote_pred)

print('\nWe can observe that smote model is performing well on imbalance da
ta compare to Random Forest,logistic regression or any  other models')
```

```
[1 1 0 ... 0 0 1]
```

```
#final submission
sub_df=pd.DataFrame({'ID_code':test['ID_code'].values})
sub_df['smote_pred']=smote_pred
sub_df.to_csv('submission.csv',index=False)
sub_df.head()
```

	ID_code	smote_pred
0	test_0	1
1	test_1	1
2	test_2	0
3	test_3	1
4	test_4	0

Fig.3.2

The above table consists of predicted target values for the test data set. Where, 1 represents the number of customer who will make transaction and 0 represent that there is no transaction done by the customer. (A whole test data set with its target variable is attached in the appendix page)

Appendix A -Python Code

```
#import libraries
import numpy as np
import pandas as pd
import seaborn as sns
import os
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.special import erfc
from sklearn.model_selection import train_test_split,cross_val_predict,
cross_val_score
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn import tree
from random import randrange, uniform
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from scipy.stats import chi2_contingency
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score,roc_curve,auc
!pip install imblearn
from imblearn.over_sampling import SMOTE, RandomOverSampler
%matplotlib inline

os.chdir("C:/Users/chandini c/Desktop")
os.getcwd()
```

```
C:\\Users\\chandini c\\Desktop'
```

```
#load data
train=pd.read_csv("train.csv")
test=pd.read_csv("test.csv")
#shape of train and test data
train.shape, test.shape
```

```
((200000, 202), (200000, 201))
```

```
#checking types
train.dtypes
```

```
ID_code     object
target       int64
var_0      float64
var_1      float64
var_2      float64
var_3      float64
var_4      float64
```

```
var_5       float64
var_6       float64
var_7       float64
var_8       float64
var_9       float64
var_10      float64
var_11      float64
var_12      float64
var_13      float64
var_14      float64
var_15      float64
var_16      float64
var_17      float64
var_18      float64
var_19      float64
var_20      float64
var_21      float64
var_22      float64
var_23      float64
var_24      float64
var_25      float64
var_26      float64
var_27      float64
              ...
var_170     float64
var_171     float64
var_172     float64
var_173     float64
var_174     float64
var_175     float64
var_176     float64
var_177     float64
var_178     float64
var_179     float64
var_180     float64
var_181     float64
var_182     float64
var_183     float64
var_184     float64
var_185     float64
var_186     float64
var_187     float64
var_188     float64
var_189     float64
var_190     float64
var_191     float64
var_192     float64
```

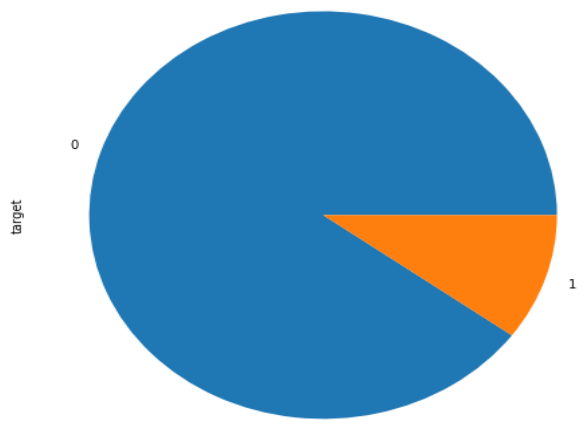
```
var_193    float64
var_194    float64
var_195    float64
var_196    float64
var_197    float64
var_198    float64
var_199    float64
Length: 202, dtype: object
```

```
#observing data
train.head(5)
#observing test data
test.head(5)
```

```
#counting observations per target class
train.target.value_counts()
```

```
0    179902
1     20098
Name: target, dtype: int64
```

```
#plotting pie chart for target class
train['target'].value_counts().plot(kind='pie', figsize=(8,8))
```



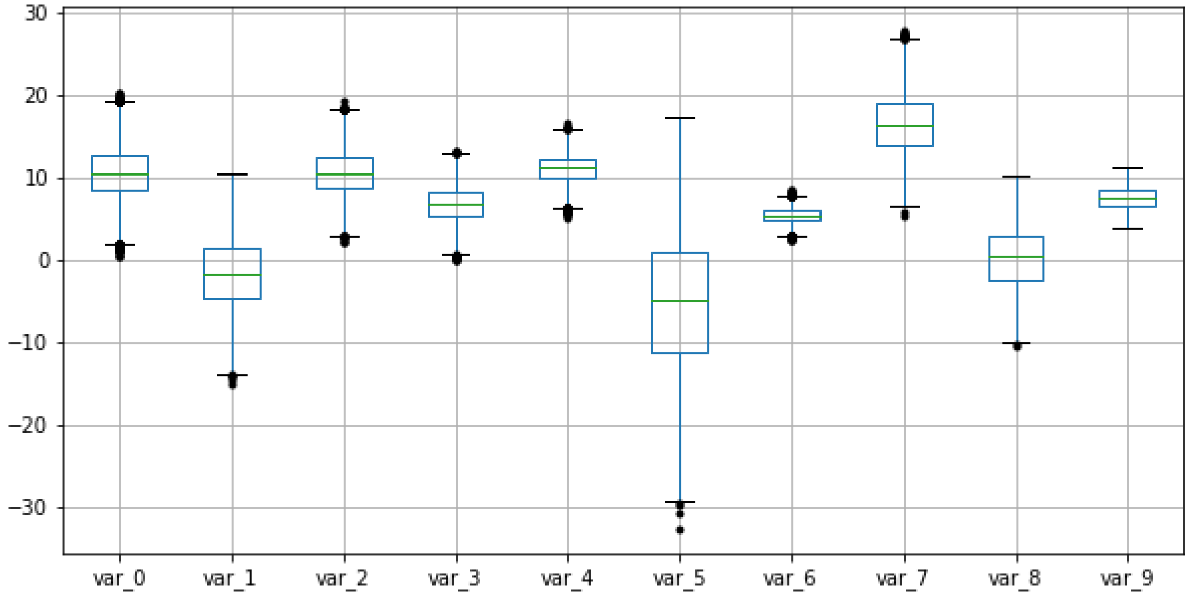

```
#checking for missing values in train data
train.isna().sum().sum()
```

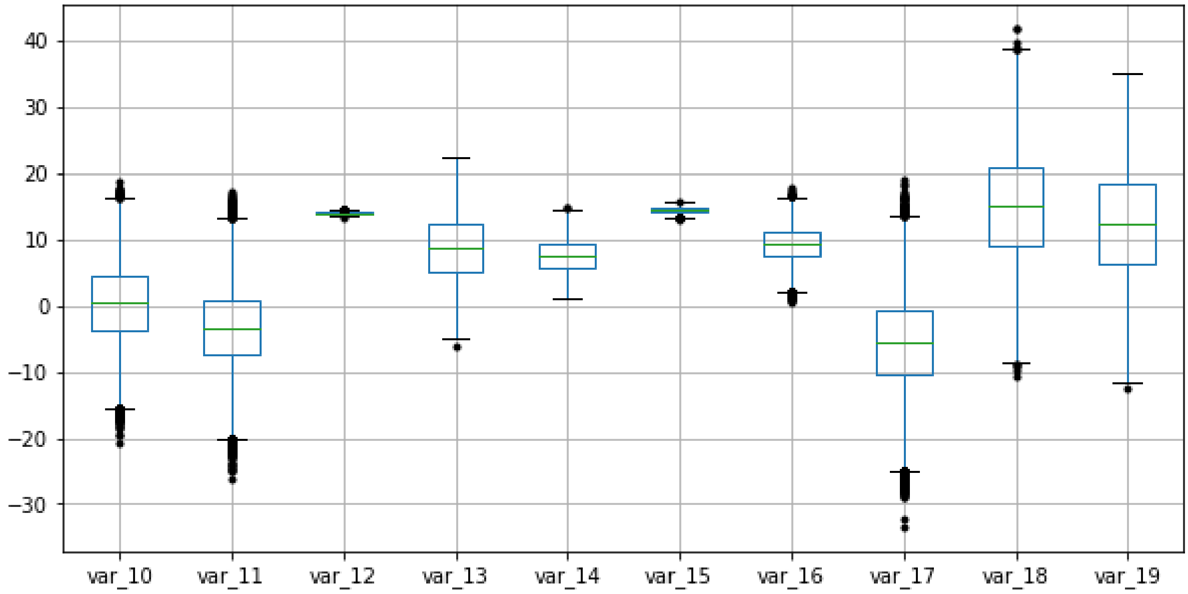
0

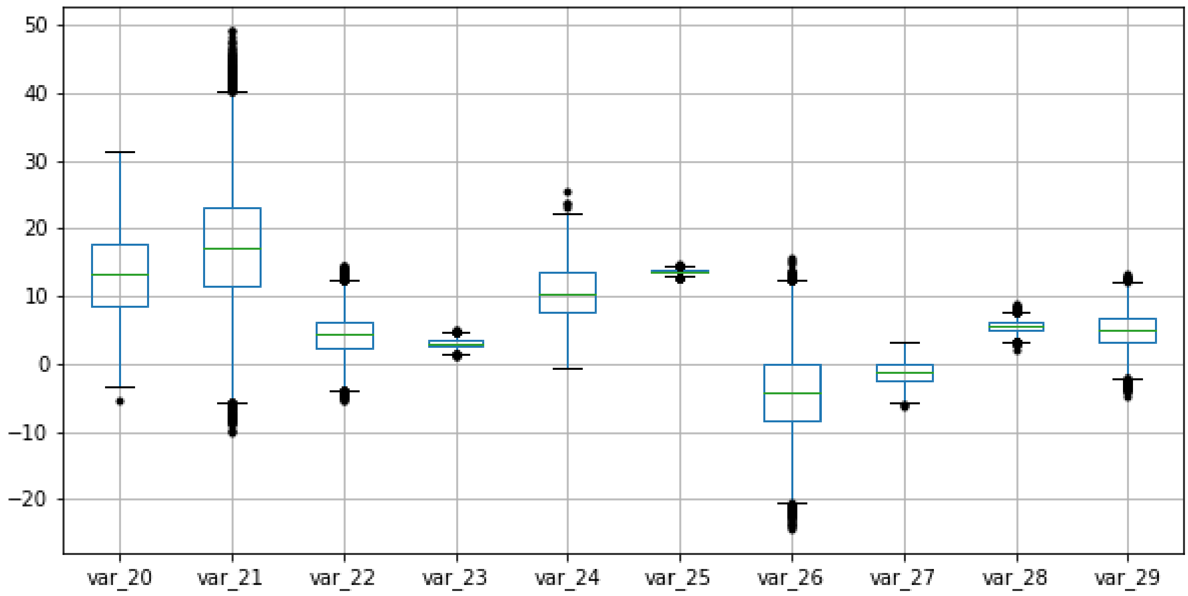
```
#outlier analysis
#putting all the df colname in a list
dfcols = list(train.columns)
```

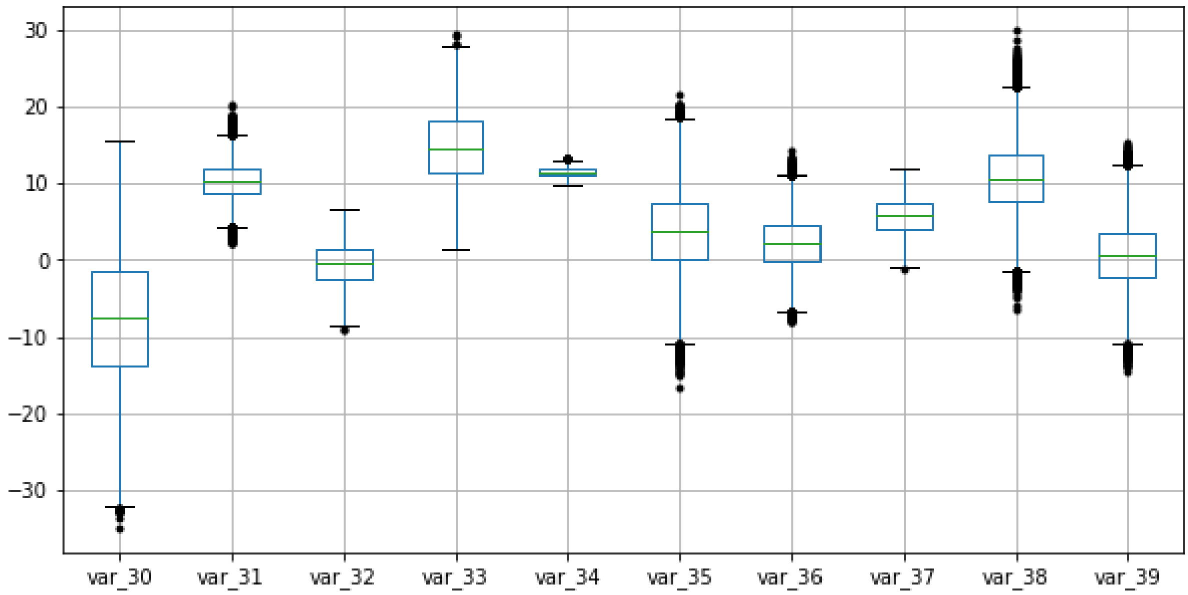
```
# exculdig target and index columns
```

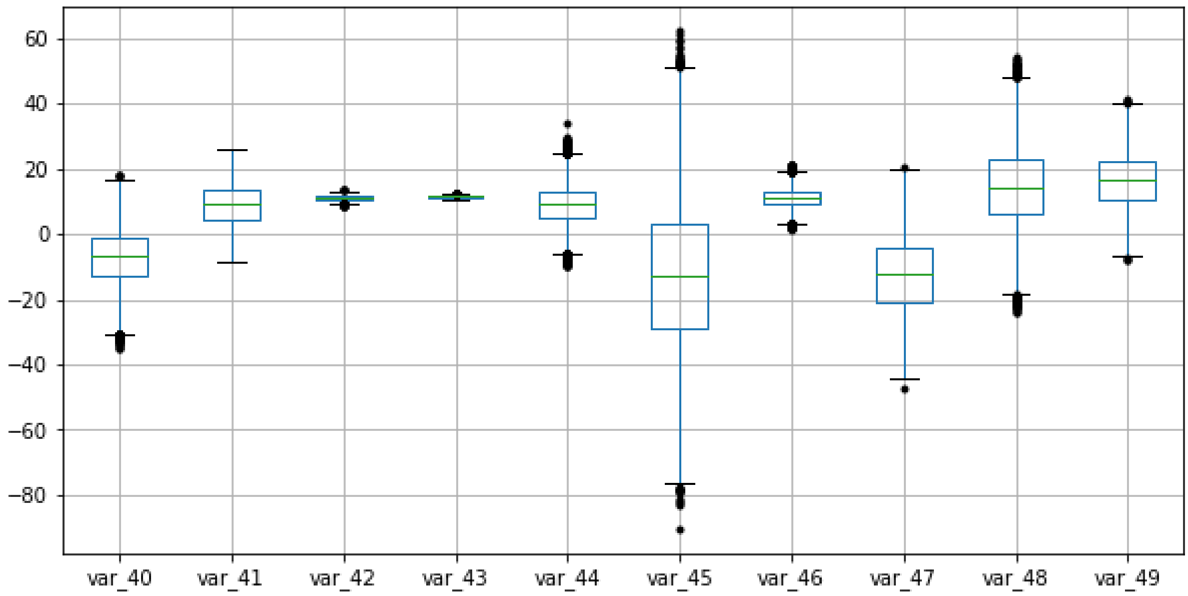
```
variables = dfcols[2:]

# splitting the list every n elements:
n = 10
chunks = [variables[x:x + n] for x in range(0, len(variables), n)]

# displaying a boxplot every n columns before removal of outliers(train)
for i in chunks:
    plt.show(train.boxplot(column = i, sym='k.', figsize=(10,5)))
```

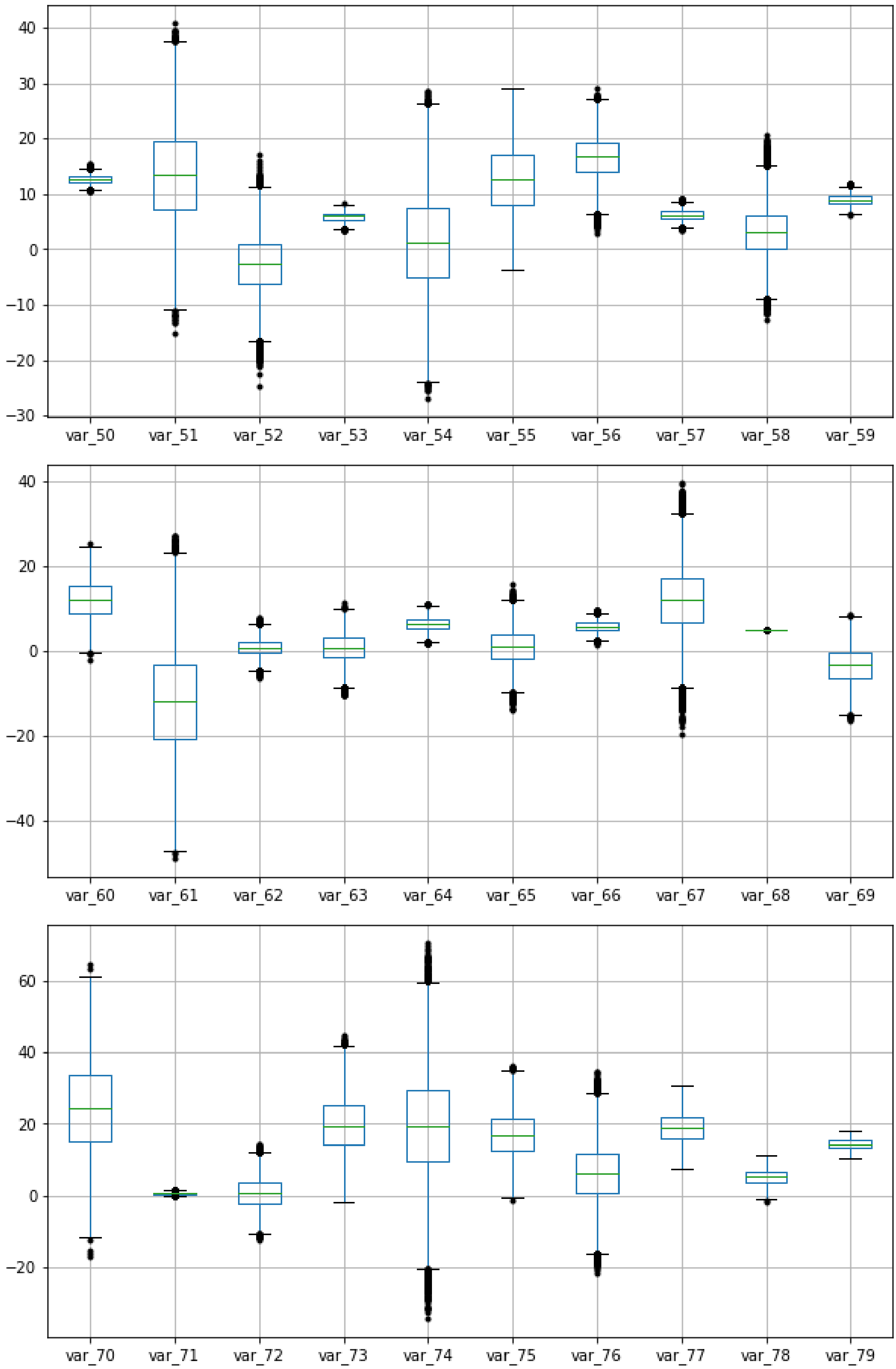


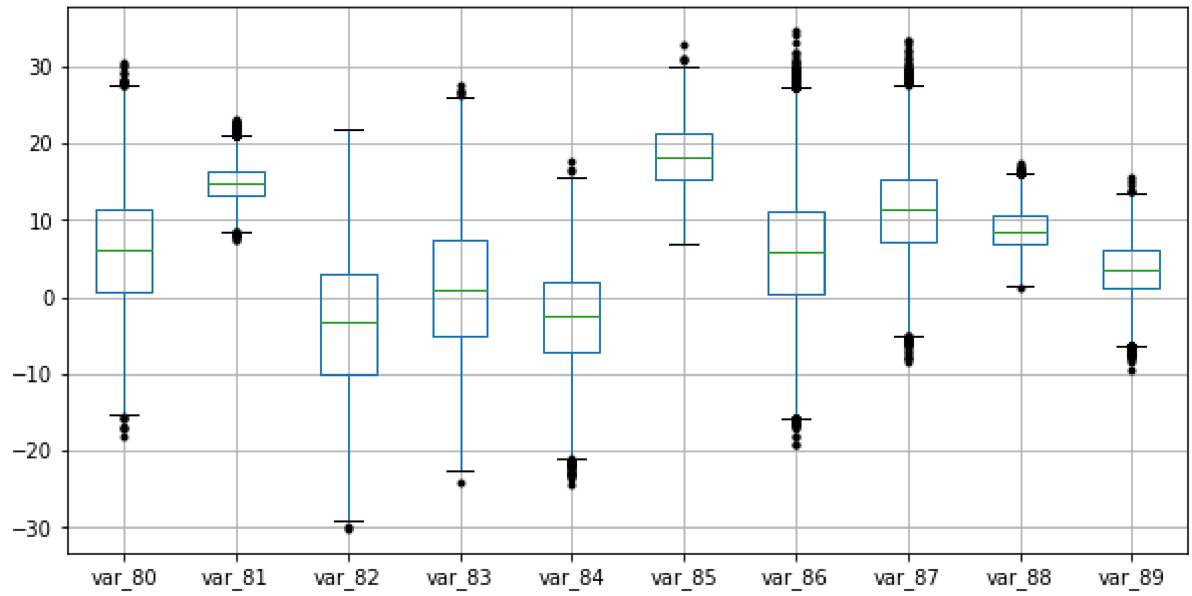


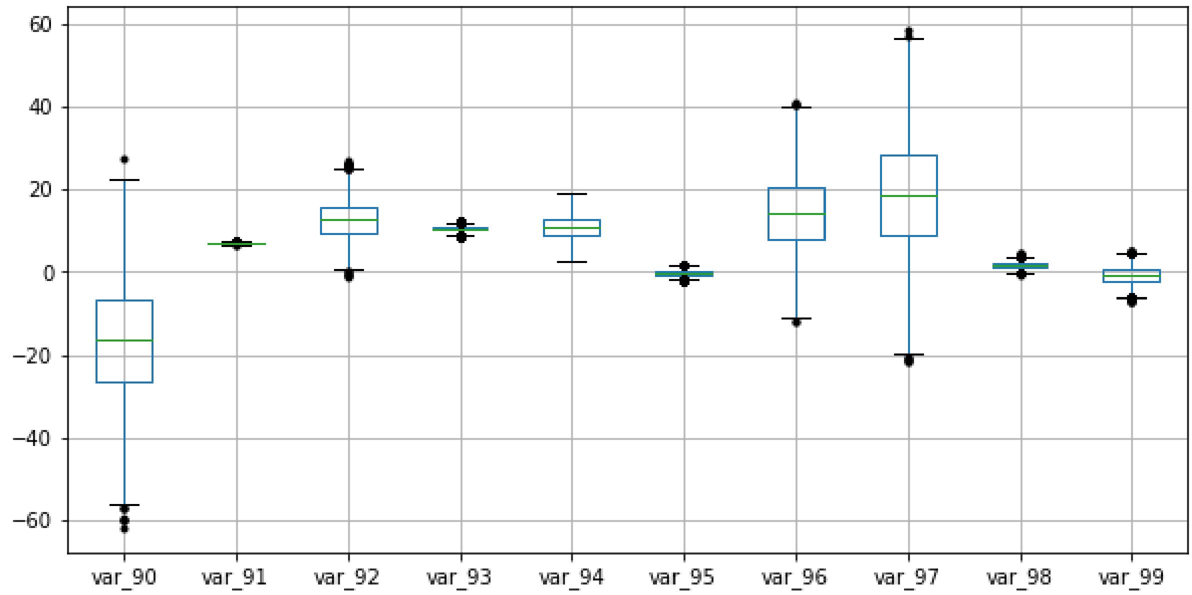

50
40
30
20
10
0
−10
−20
var_20
var_21
var_22
var_23
var_24
var_25
var_26
var_27
var_28
var_29

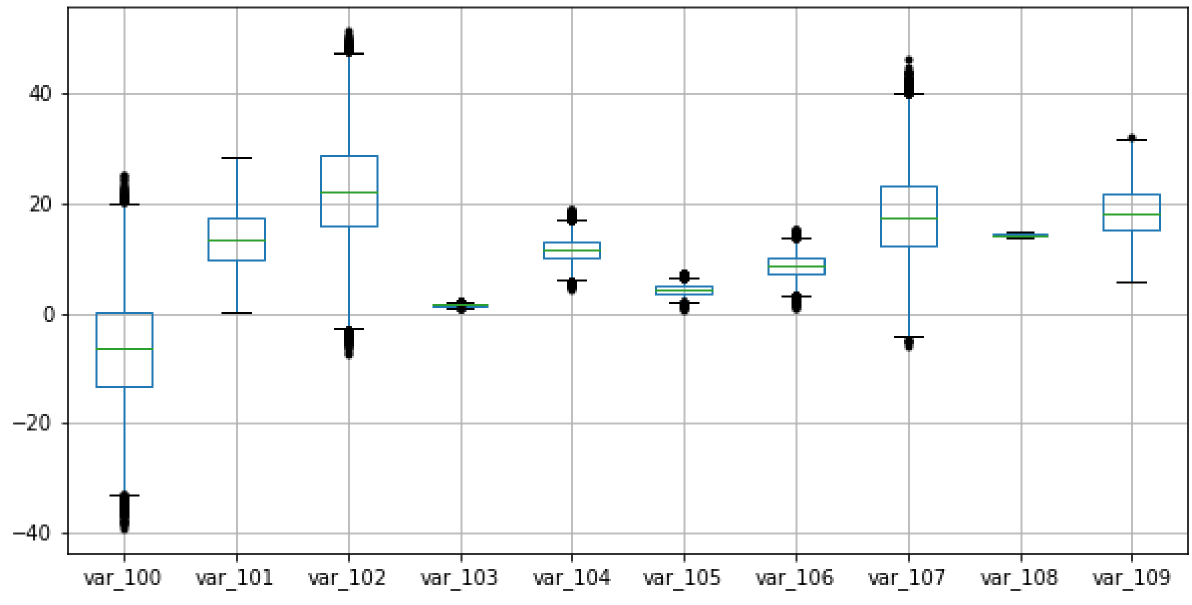

30
20
10
0
−10
−20
−30
var_30
var_31
var_32
var_33
var_34
var_35
var_36
var_37
var_38
var_39

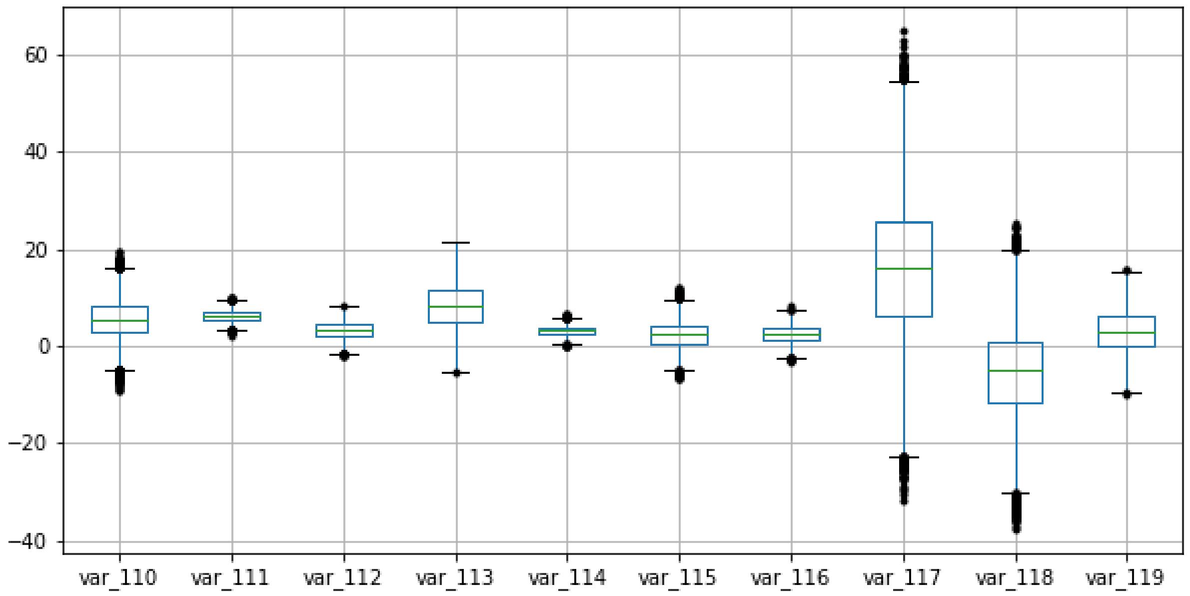

60
40
20
0
−20
−40
−60
−80
var_40
var_41
var_42
var_43
var_44
var_45
var_46
var_47
var_48
var_49

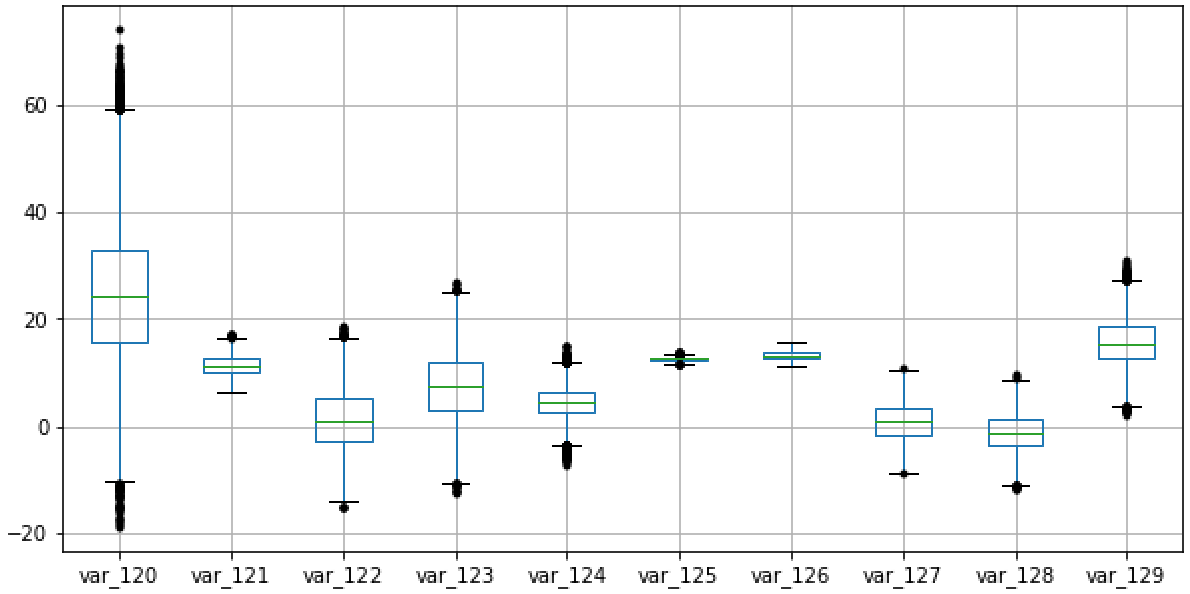

40
30
20
10
0
−10
−20
−30
var_50
var_51
var_52
var_53
var_54
var_55
var_56
var_57
var_58
var_59
40
20
0
−20
−40
var_60
var_61
var_62
var_63
var_64
var_65
var_66
var_67
var_68
var_69
60
40
20
0
−20
var_70
var_71
var_72
var_73
var_74
var_75
var_76
var_77
var_78
var_79

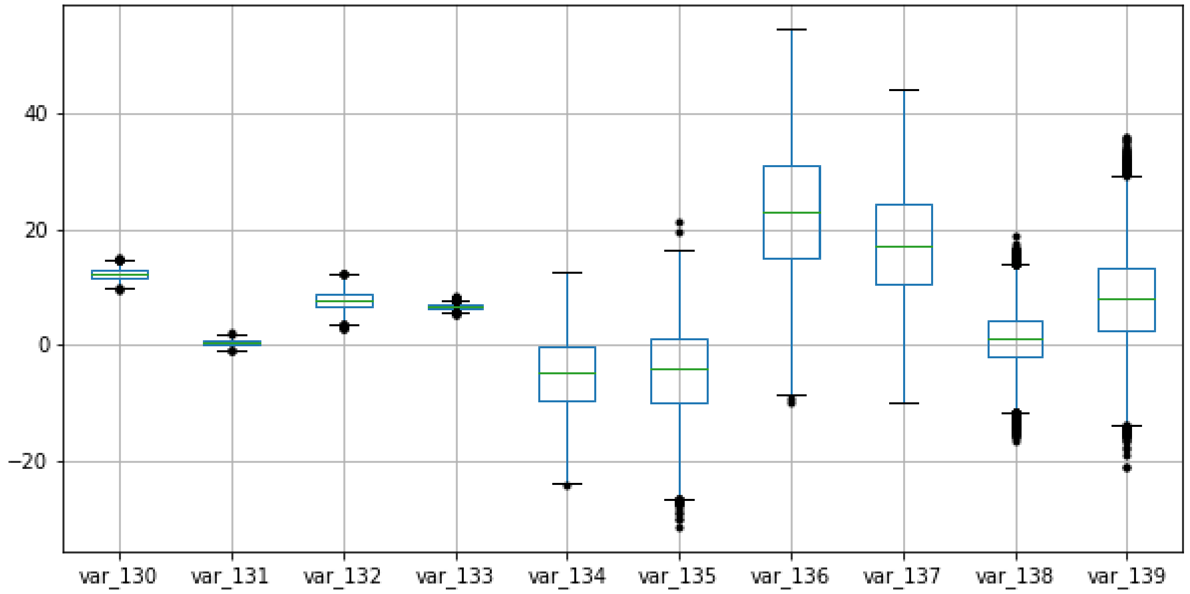

30
20
10
0
−10
−20
−30
var_80
var_81
var_82
var_83
var_84
var_85
var_86
var_87
var_88
var_89

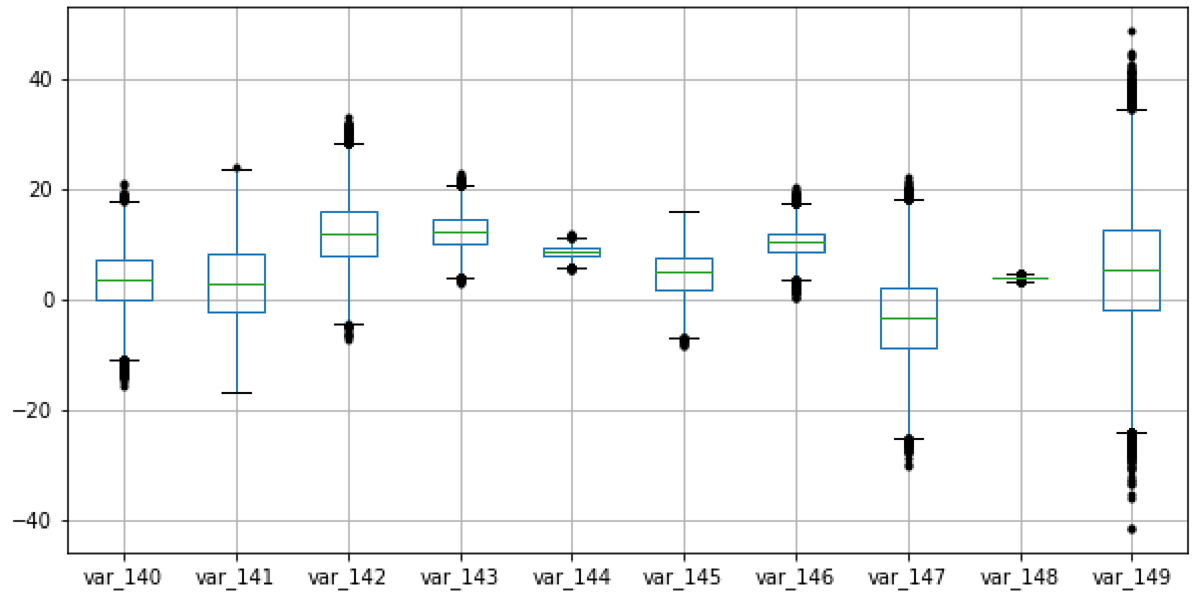

60
40
20
0
−20
−40
−60
var_90
var_91
var_92
var_93
var_94
var_95
var_96
var_97
var_98
var_99

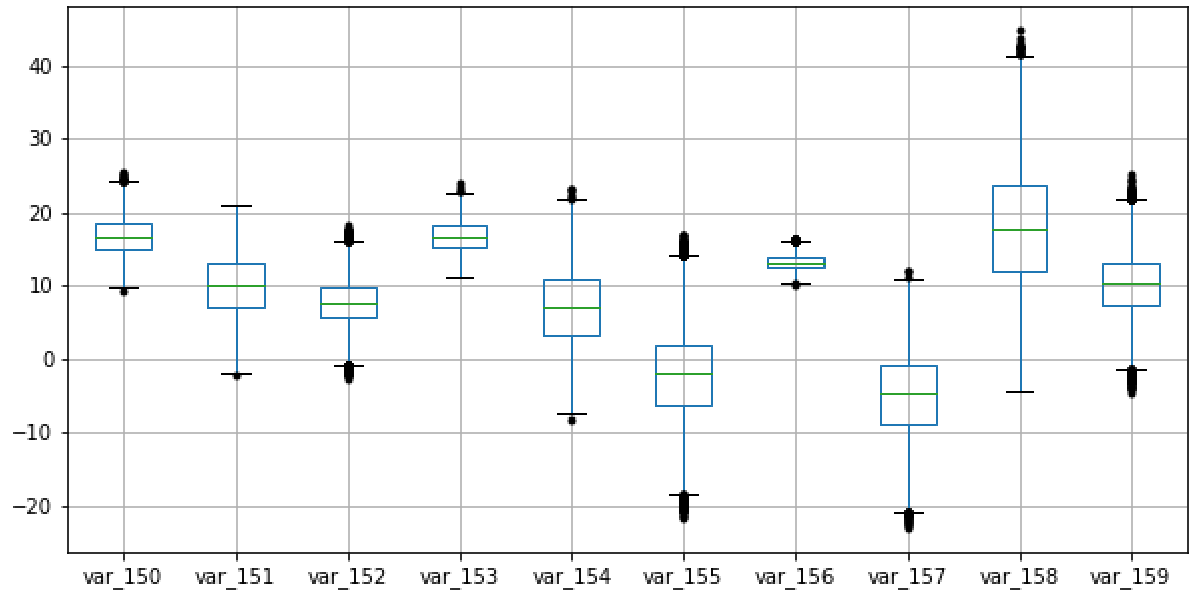

40
20
0
−20
−40
var_100
var_101
var_102
var_103
var_104
var_105
var_106
var_107
var_108
var_109

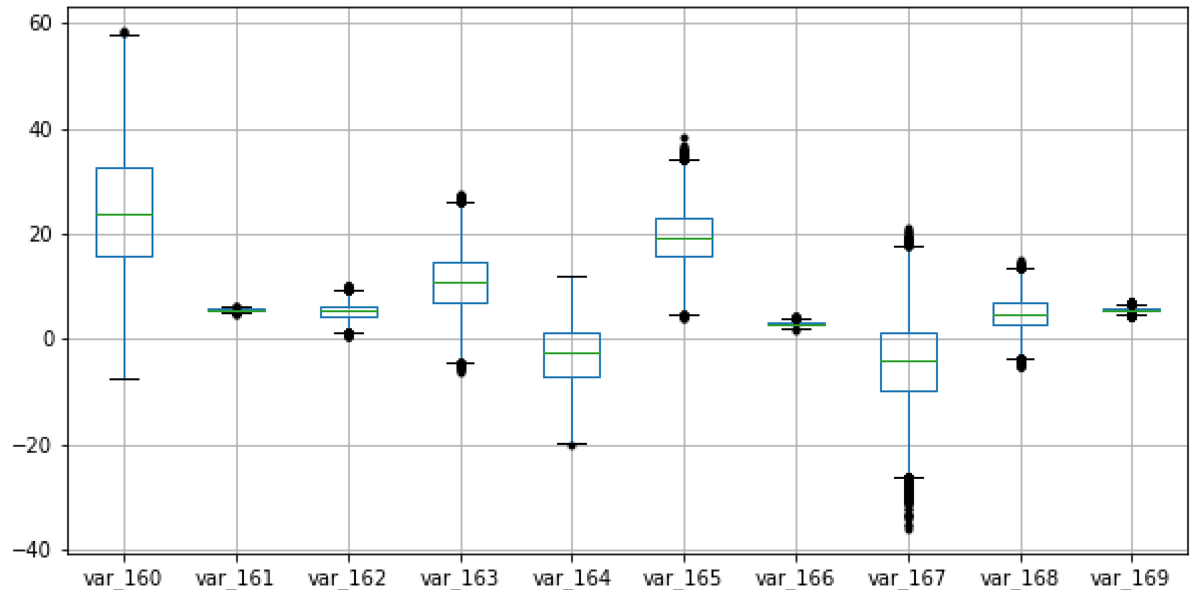

60
40
20
0
−20
−40
var_110
var_111
var_112
var_113
var_114
var_115
var_116
var_117
var_118
var_119

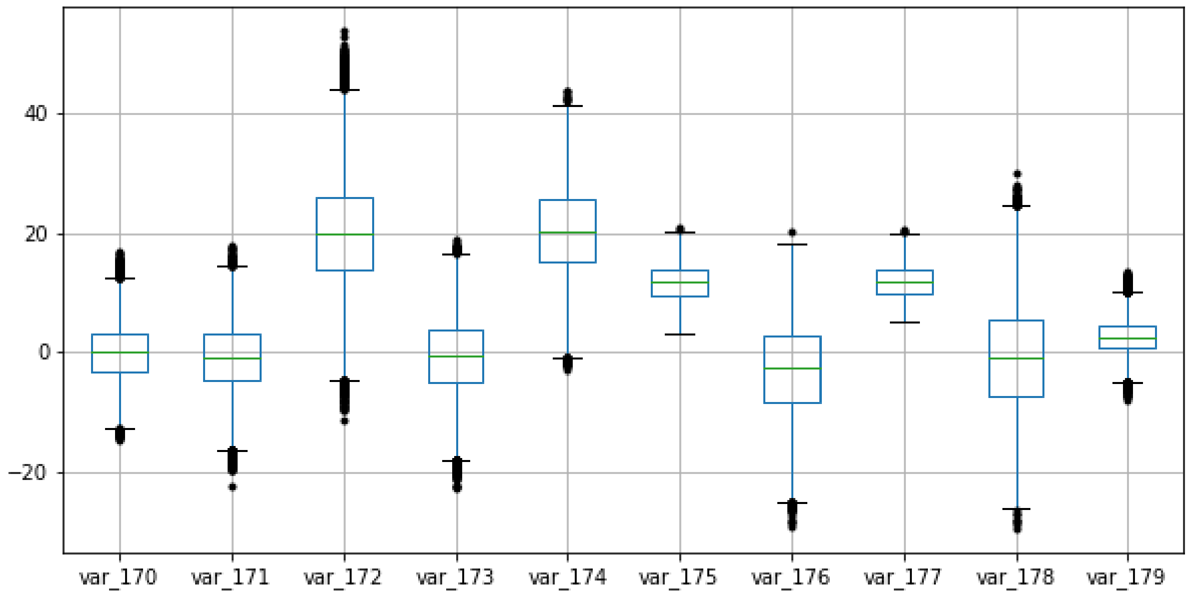

60
40
20
0
−20
var_120
var_121
var_122
var_123
var_124
var_125
var_126
var_127
var_128
var_129


40
20
0
−20
var_130
var_131
var_132
var_133
var_134
var_135
var_136
var_137
var_138
var_139

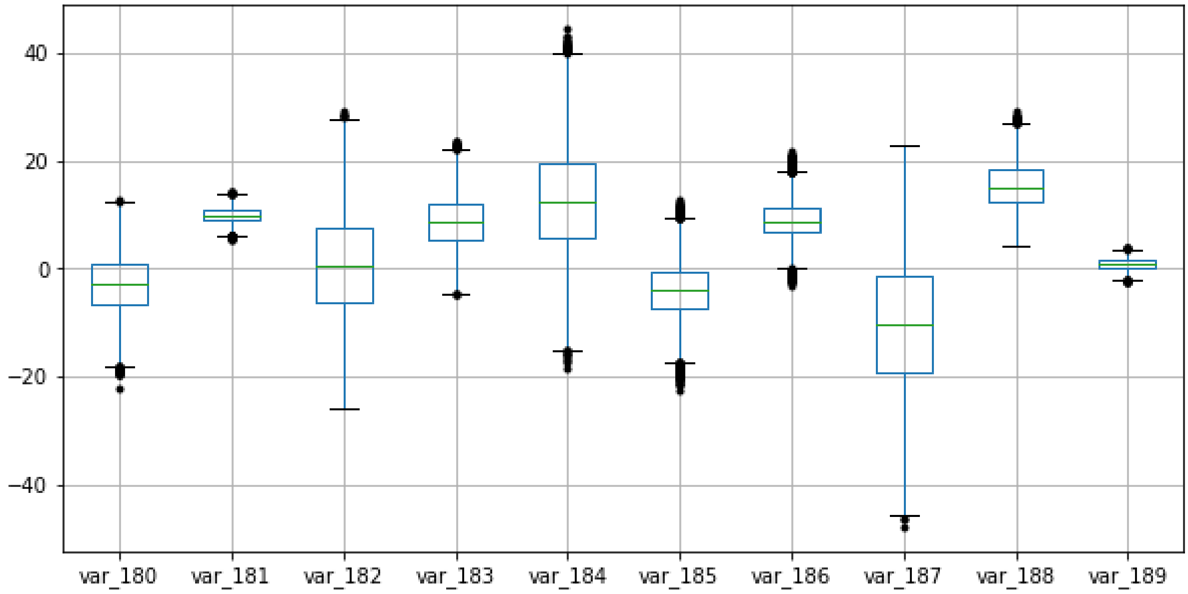

40
20
0
−20
−40
var_140
var_141
var_142
var_143
var_144
var_145
var_146
var_147
var_148
var_149


40
30
20
10
0
−10
−20
var_150
var_151
var_152
var_153
var_154
var_155
var_156
var_157
var_158
var_159

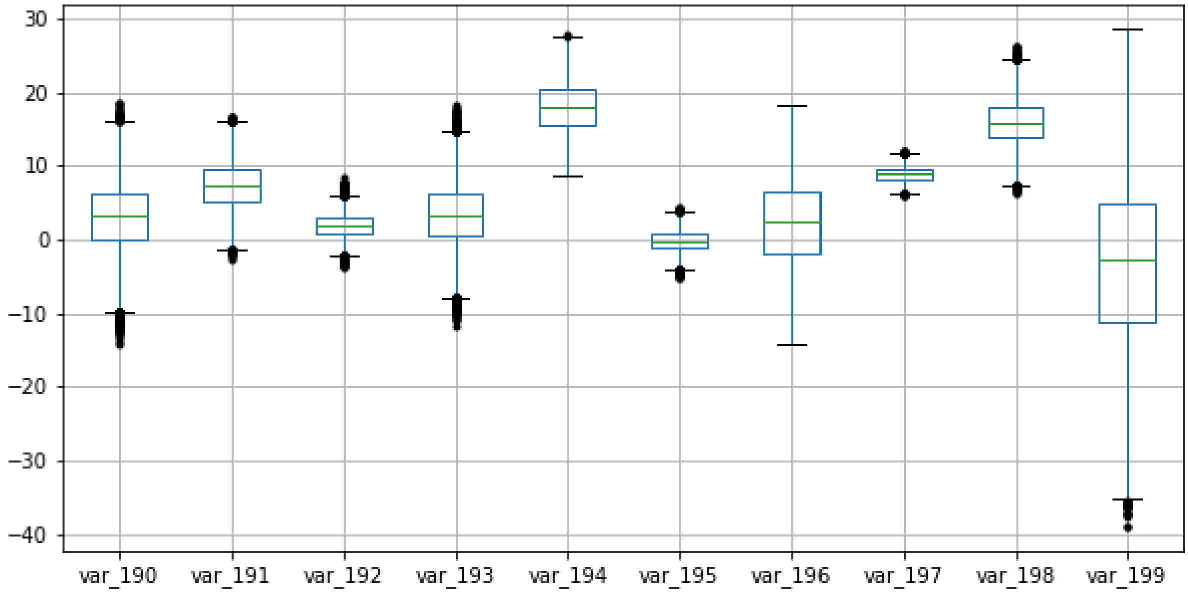

60
40
20
0
−20
−40
var_160
var_161
var_162
var_163
var_164
var_165
var_166
var_167
var_168
var_169



40
20
0
−20
var_170
var_171
var_172
var_173
var_174
var_175
var_176
var_177
var_178
var_179



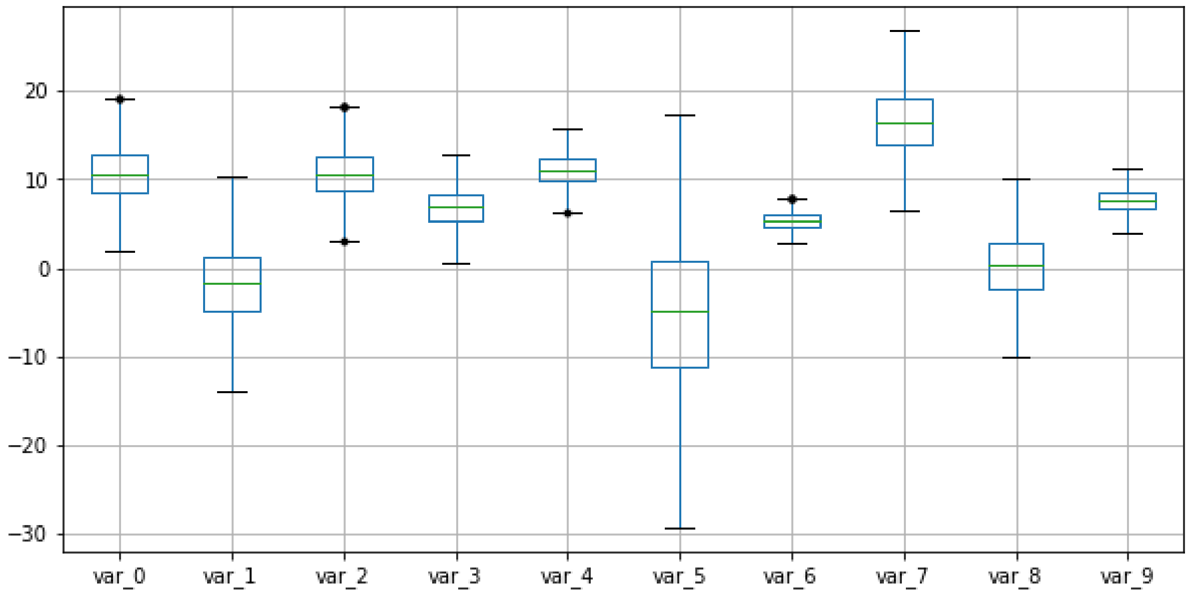
40
20
0
−20
−40
var_180
var_181
var_182
var_183
var_184
var_185
var_186
var_187
var_188
var_189

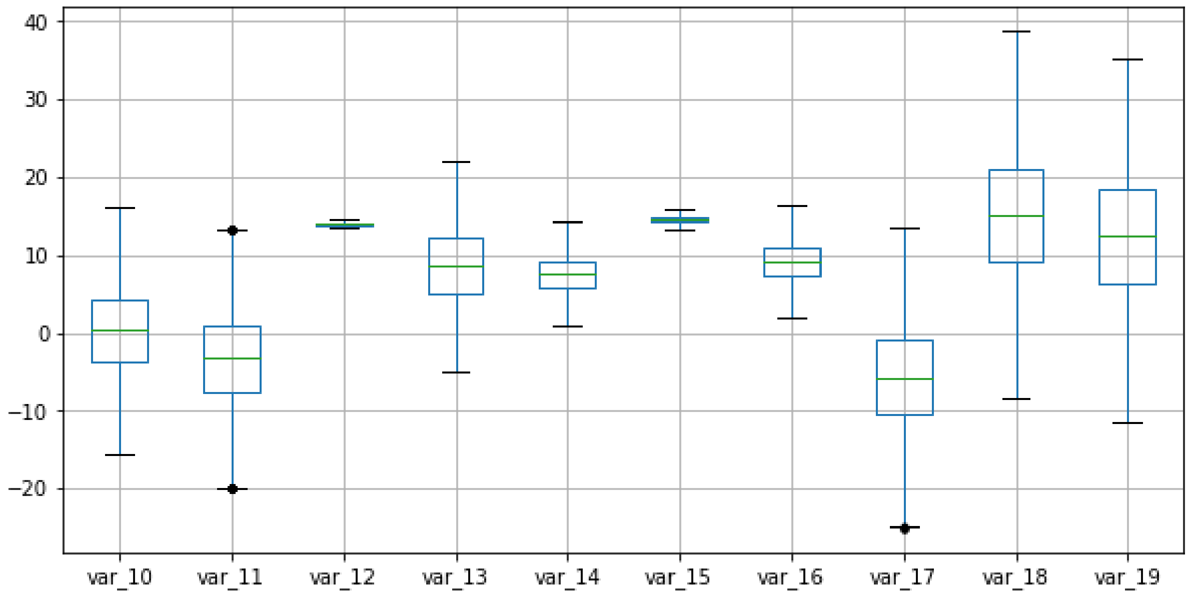


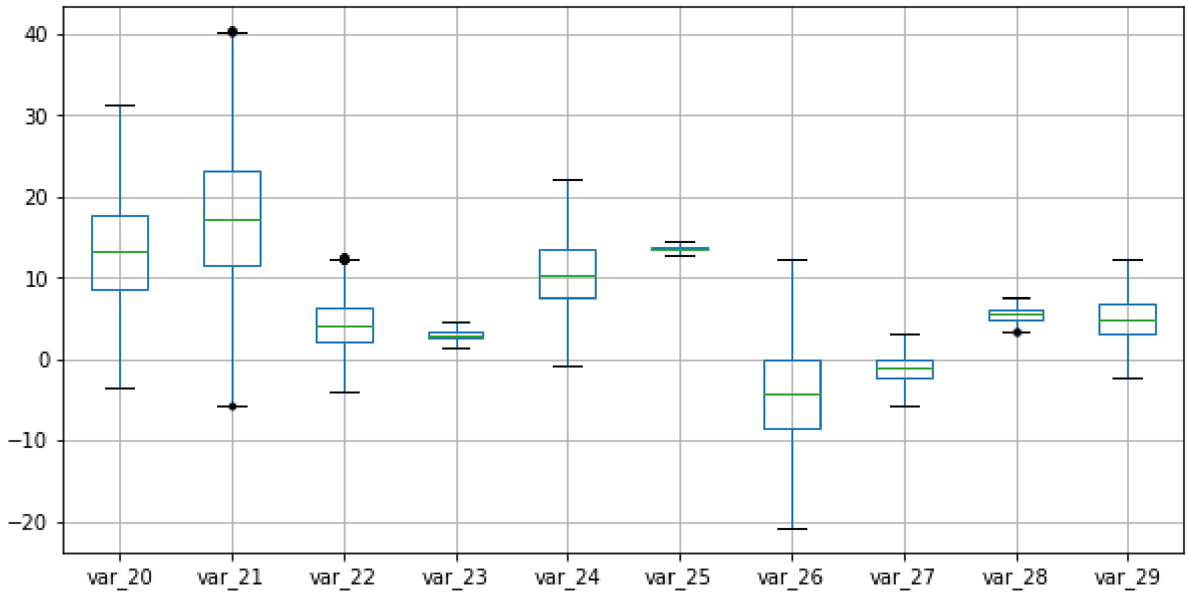
30
20
10
0
−10
−20
−30
−40
var_190
var_191
var_192
var_193
var_194
var_195
var_196
var_197
var_198
var_199

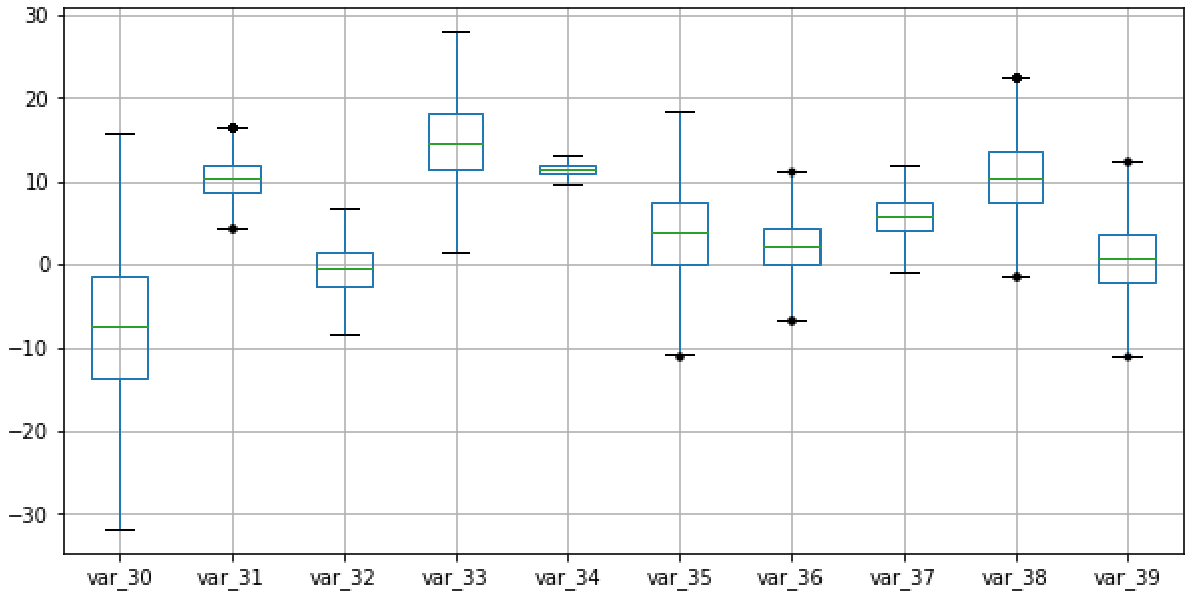
```python
numerical_features=train.columns[2:]

#outliers in each variable in train data
train_outliers = dict()
for col in [col for col in numerical_features]:
    q75,q25=np.percentile(train.loc[:,col],[75,25])
    Q=q75-q25
    min=q25-(Q*1.5)
    max=q75+(Q*1.5)
    #print(min)
    #print(max)
    train=train.drop(train[train.loc[:,col]<min].index)
    train=train.drop(train[train.loc[:,col]>max].index)
```

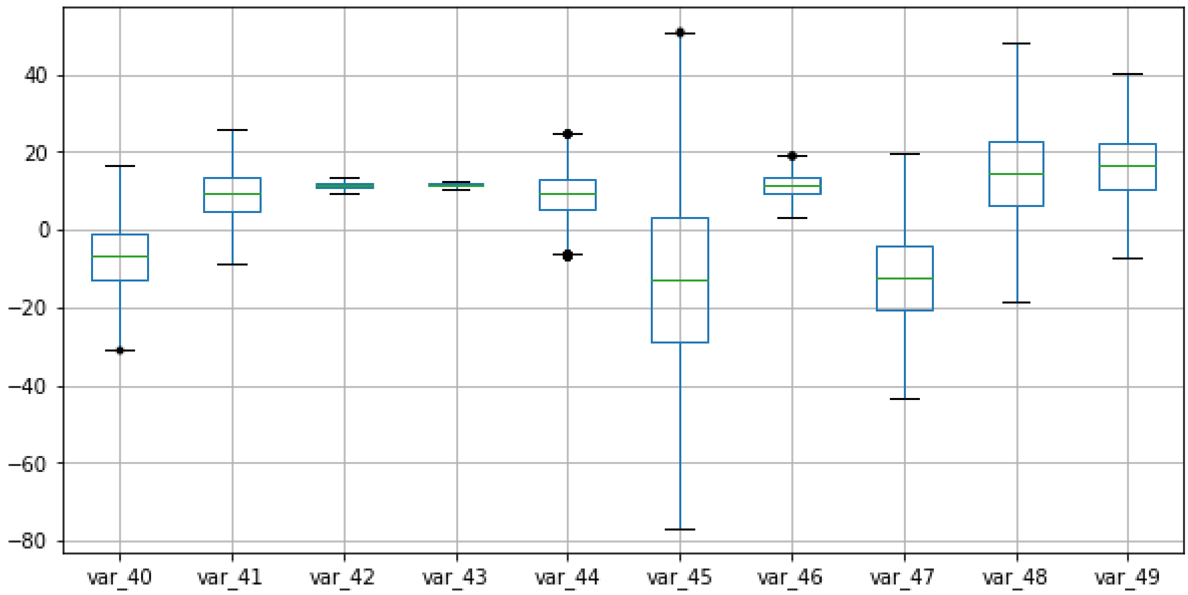
```python
# displaying a boxplot every n columns after removal of outliers:
for i in chunks:
    plt.show(train.boxplot(column = i, sym='k.', figsize=(10,5)))
```

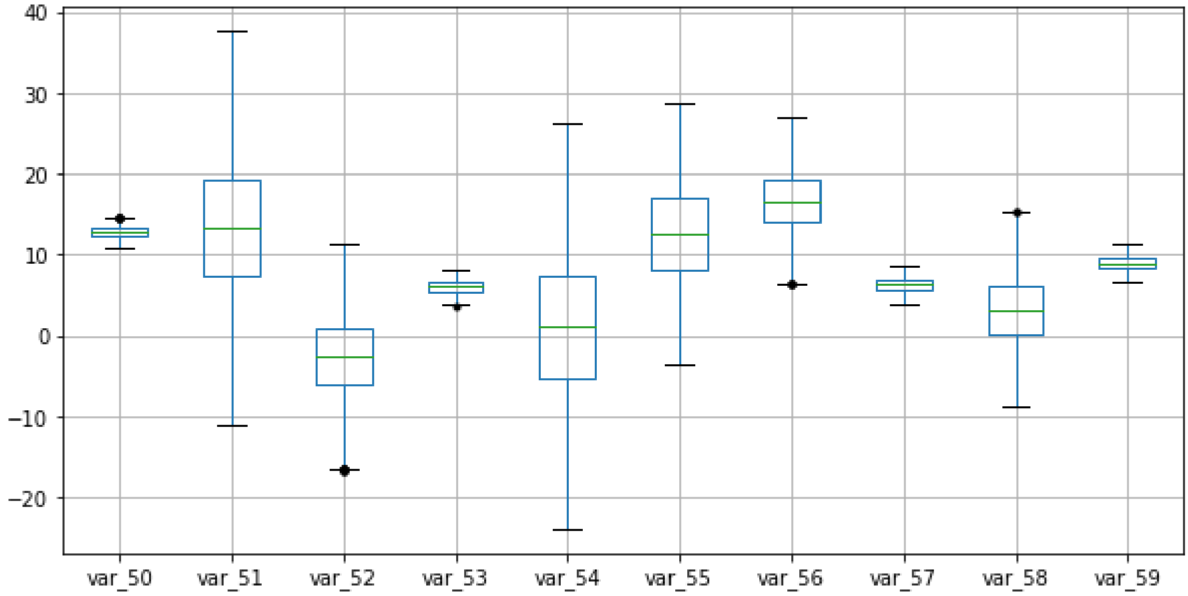


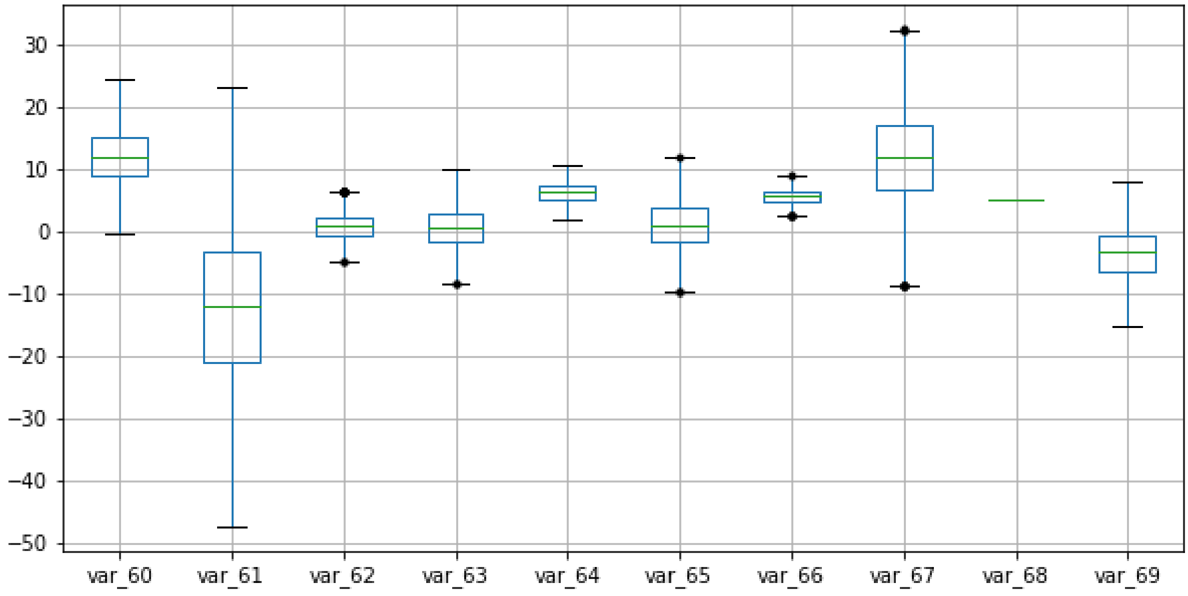

40
30
20
10
0
−10
−20
var_10
var_11
var_12
var_13
var_14
var_15
var_16
var_17
var_18
var_19

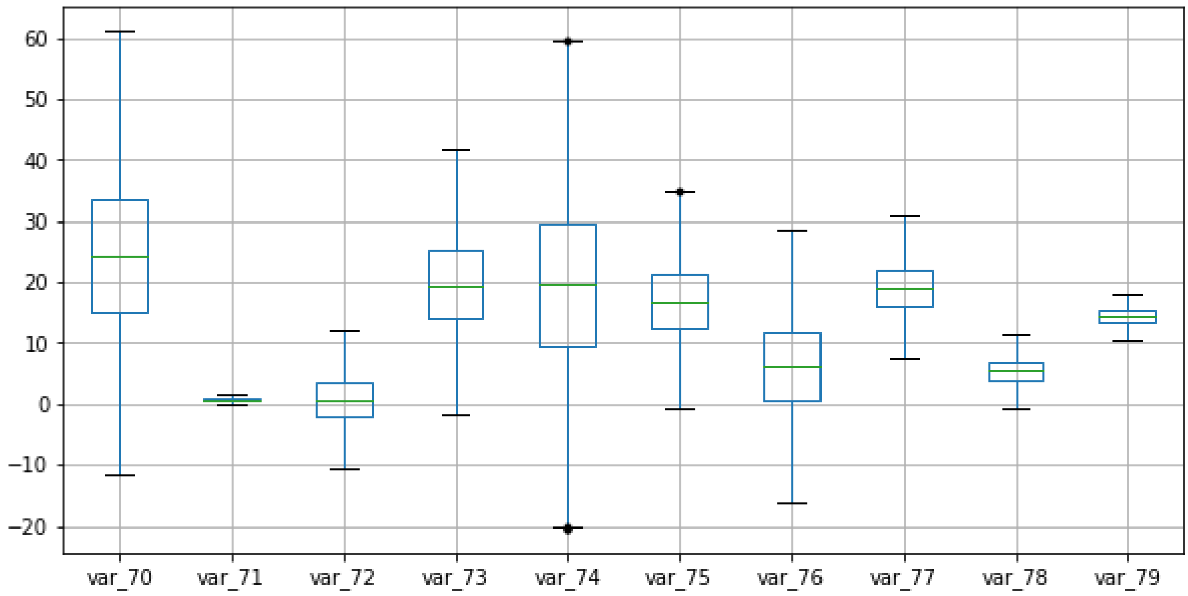

40
30
20
10
0
−10
−20
var_20
var_21
var_22
var_23
var_24
var_25
var_26
var_27
var_28
var_29

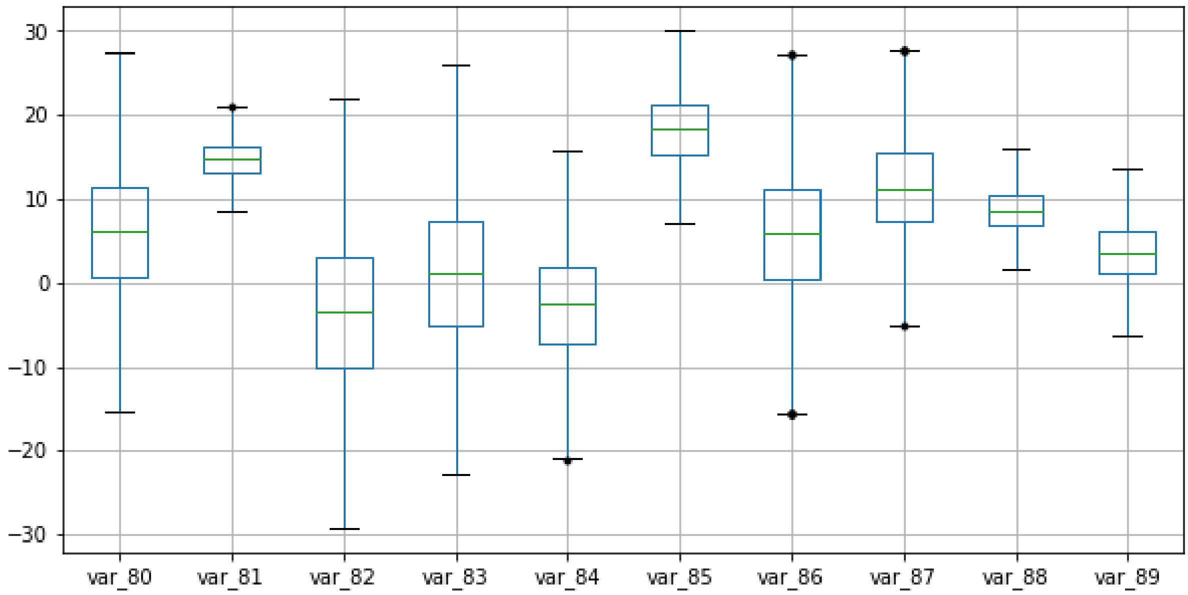

30
20
10
0
−10
−20
−30
var_30
var_31
var_32
var_33
var_34
var_35
var_36
var_37
var_38
var_39

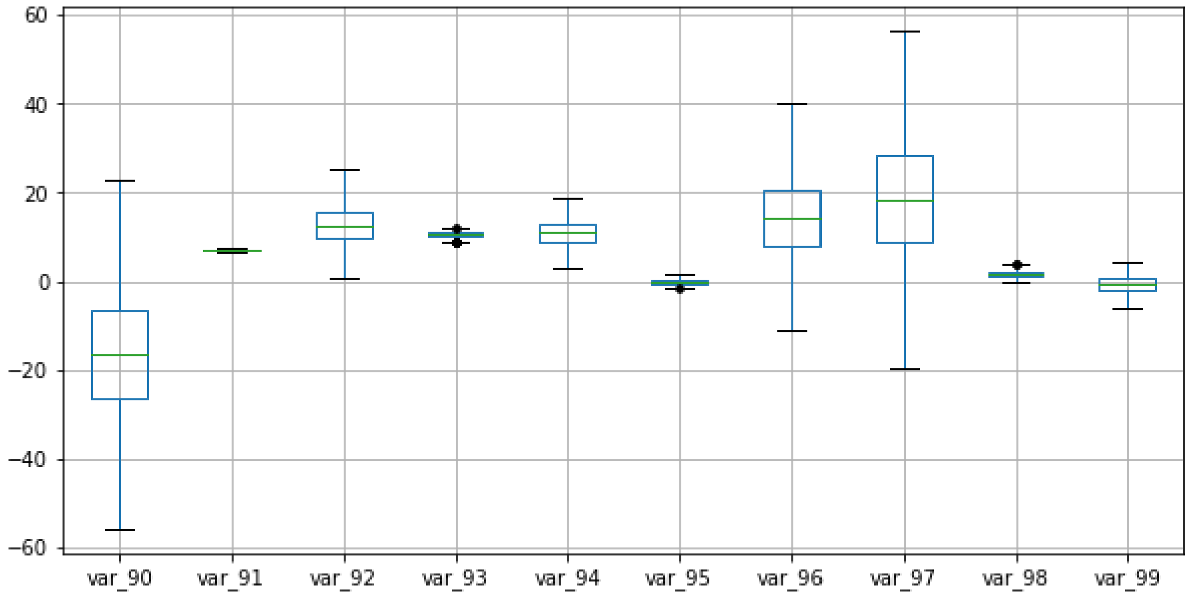

40
20
0
−20
−40
−60
−80
var_40 var_41 var_42 var_43 var_44 var_45 var_46 var_47 var_48 var_49

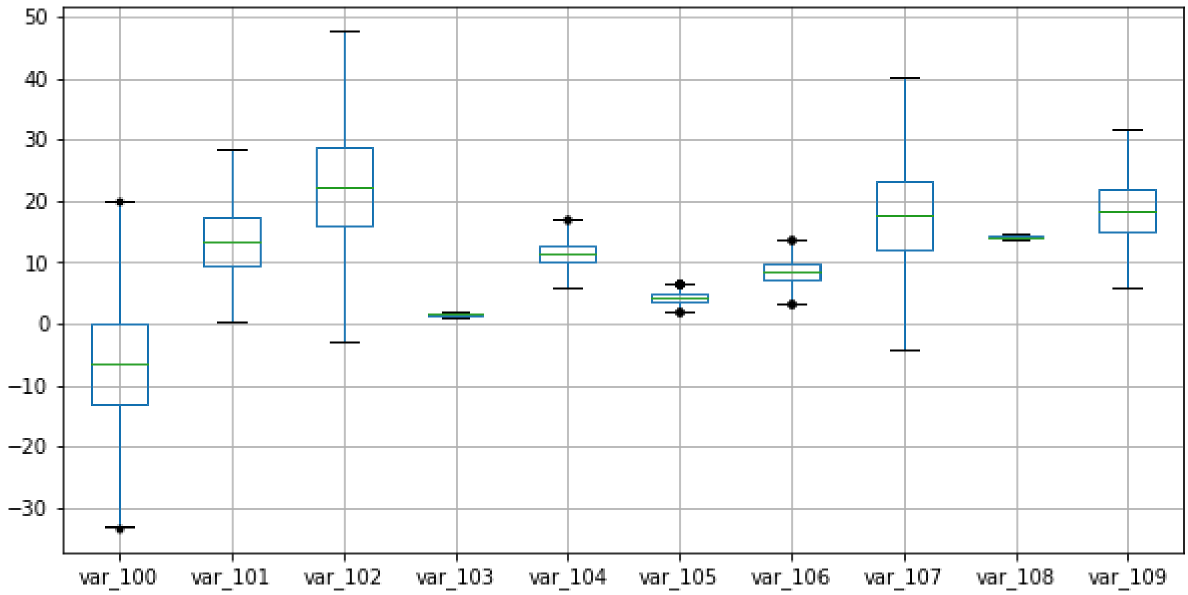

40
30
20
10
0
−10
−20
var_50 var_51 var_52 var_53 var_54 var_55 var_56 var_57 var_58 var_59


30
20
10
0
−10
−20
−30
−40
−50
var_60 var_61 var_62 var_63 var_64 var_65 var_66 var_67 var_68 var_69

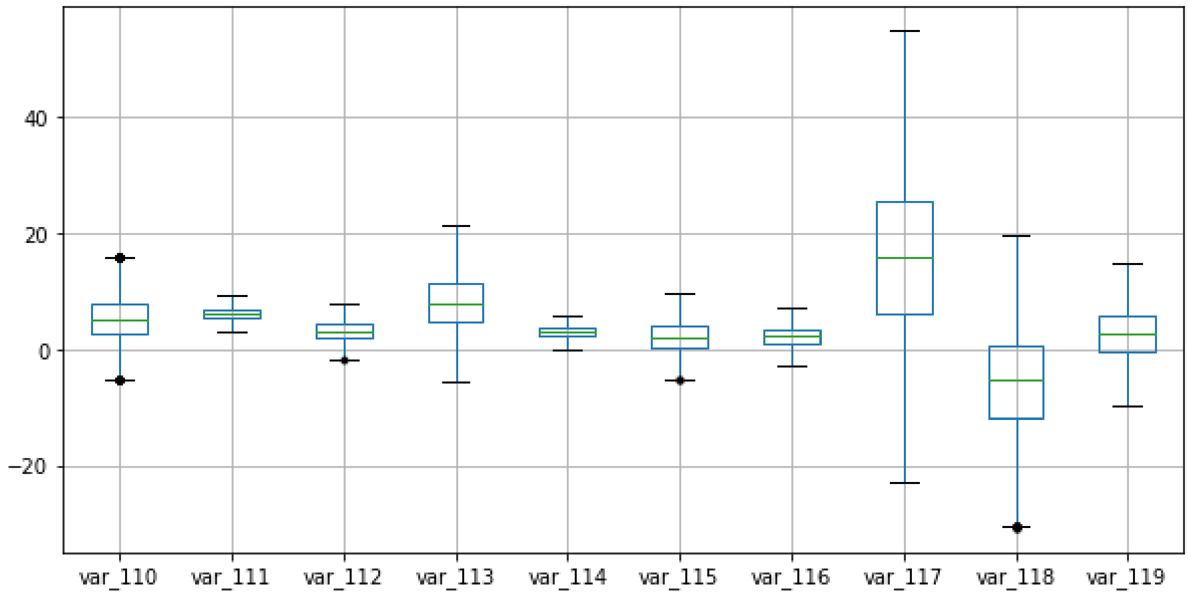

60
50
40
30
20
10
0
−10
−20
var_70
var_71
var_72
var_73
var_74
var_75
var_76
var_77
var_78
var_79


30
20
10
0
−10
−20
−30
var_80
var_81
var_82
var_83
var_84
var_85
var_86
var_87
var_88
var_89

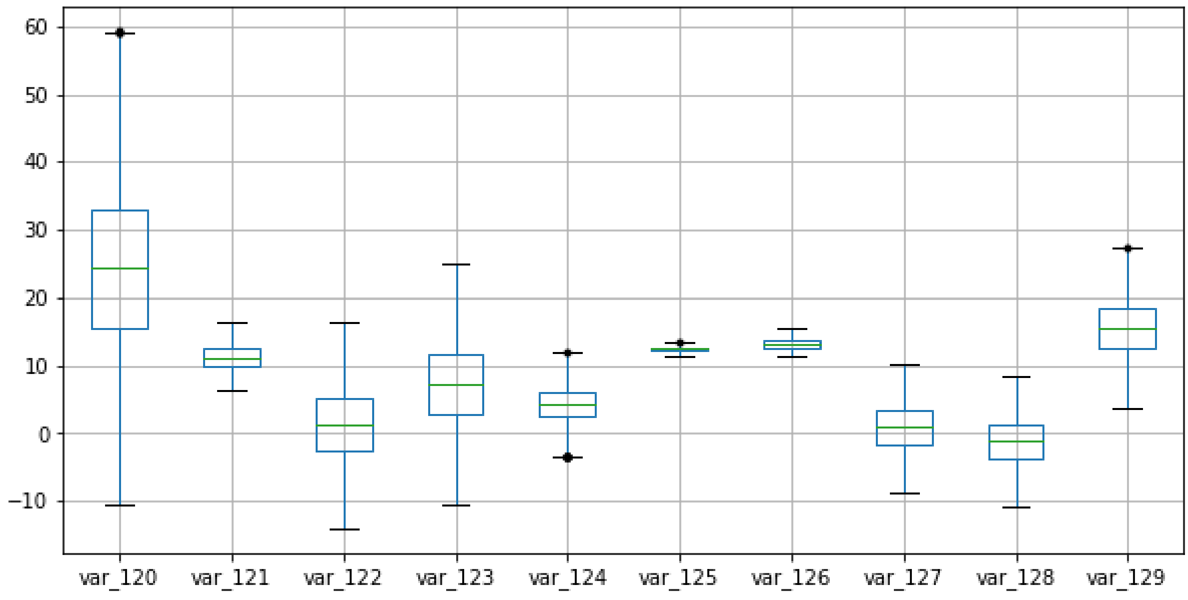

60
40
20
0
−20
−40
−60
var_90
var_91
var_92
var_93
var_94
var_95
var_96
var_97
var_98
var_99



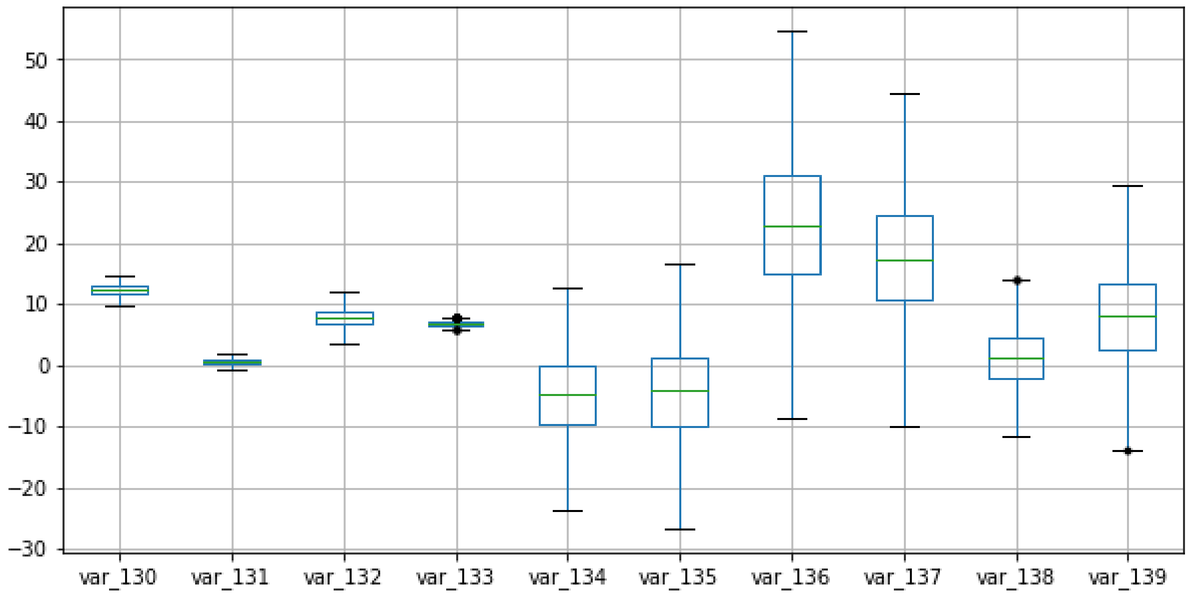
50
40
30
20
10
0
−10
−20
−30
var_100
var_101
var_102
var_103
var_104
var_105
var_106
var_107
var_108
var_109

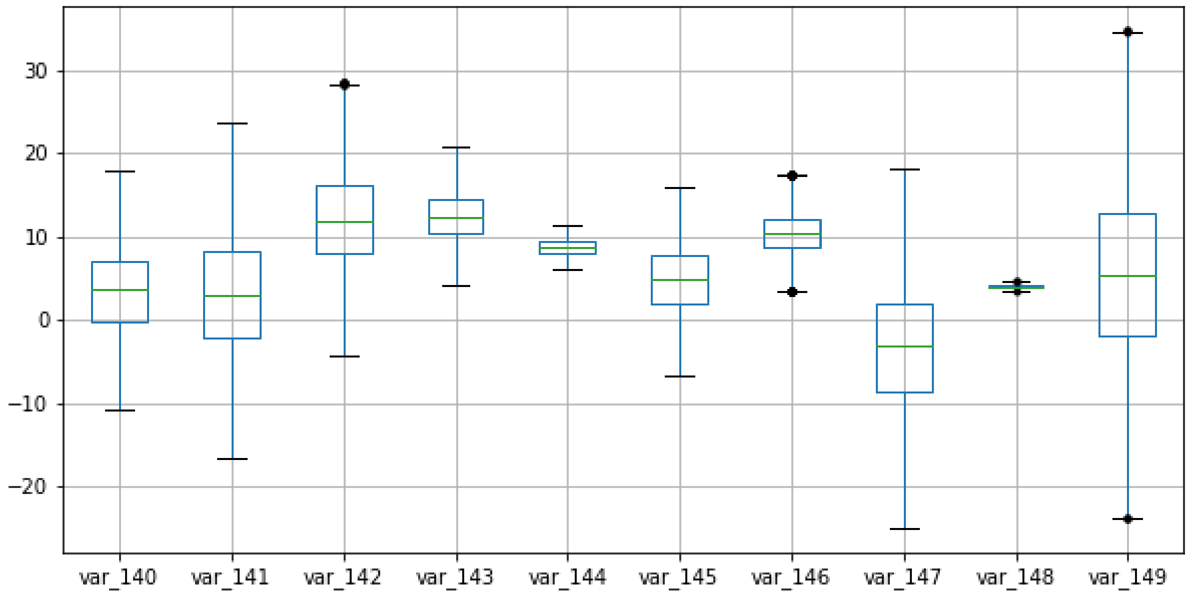


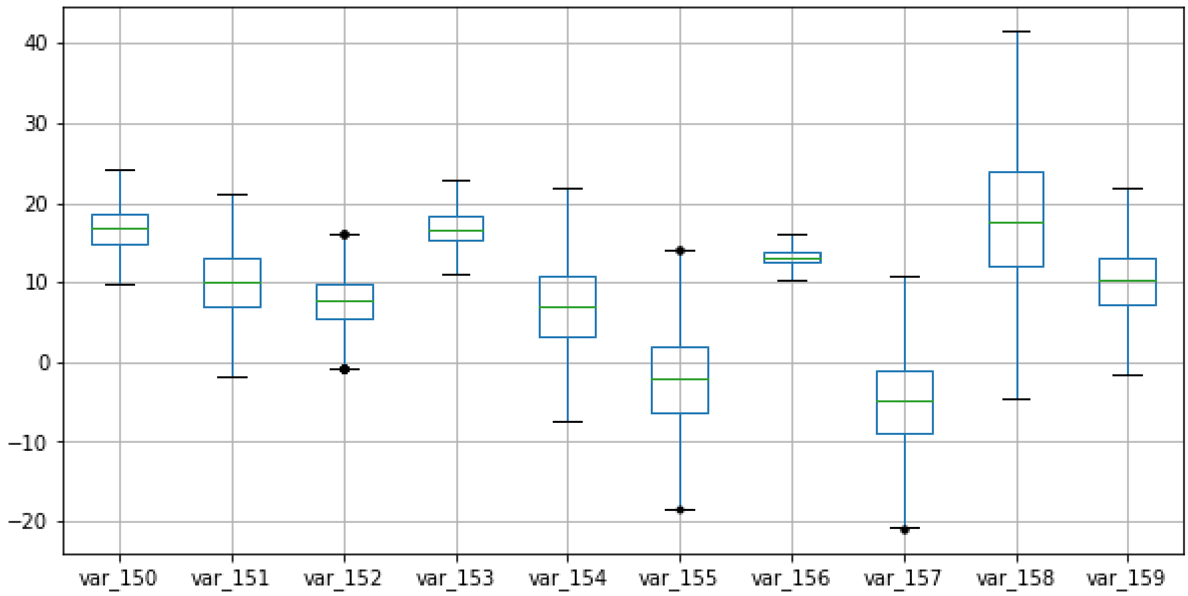
40
20
0
−20
var_110
var_111
var_112
var_113
var_114
var_115
var_116
var_117
var_118
var_119

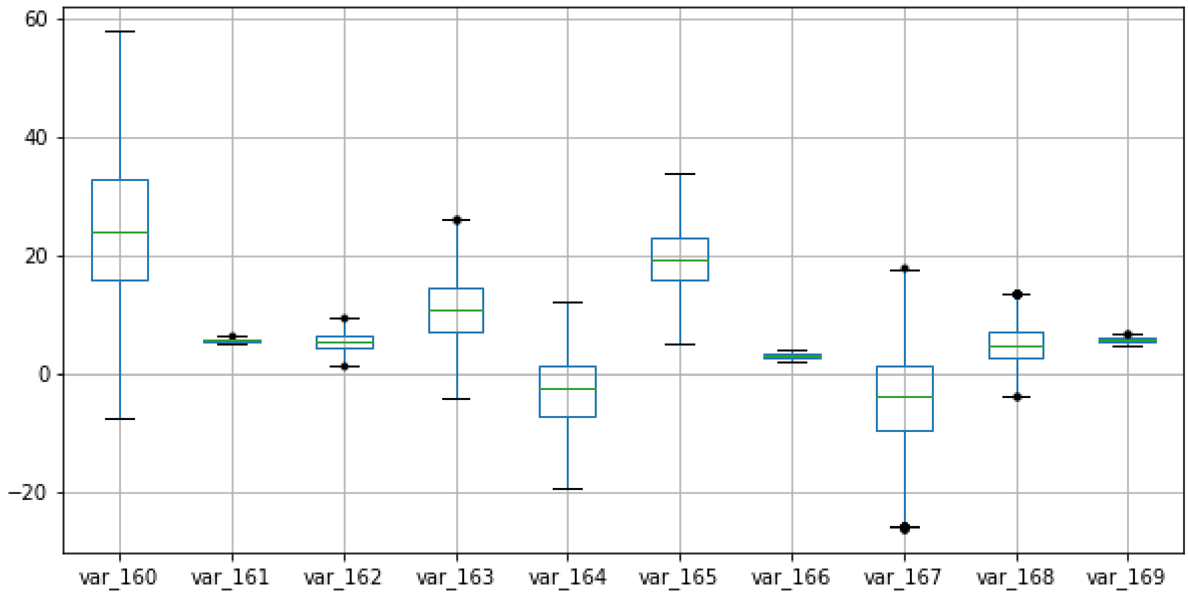


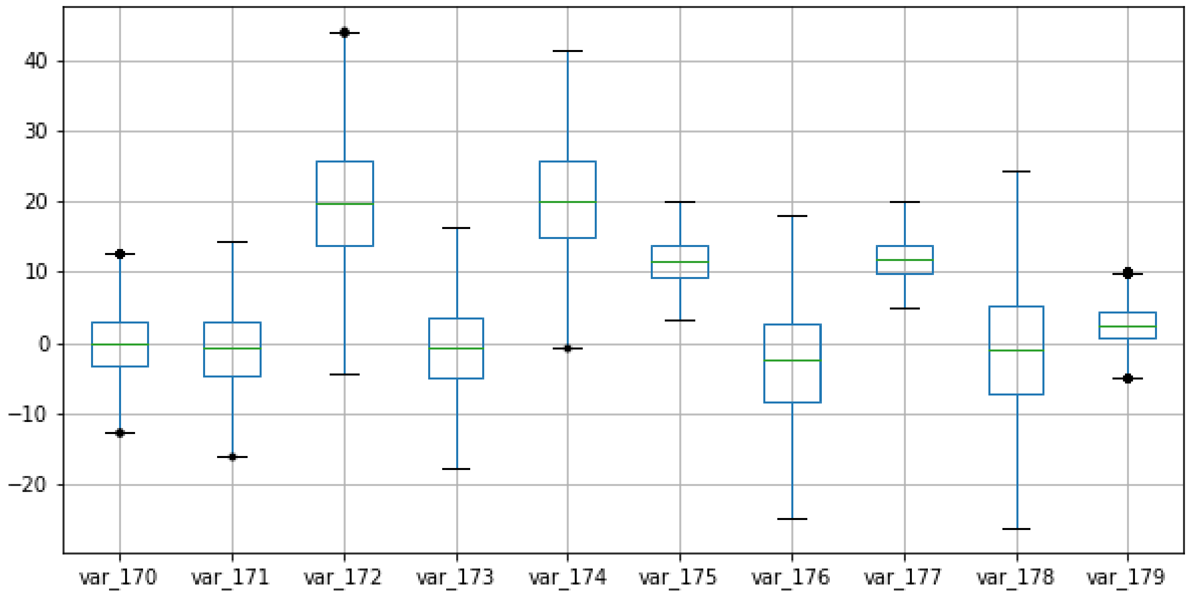
60
50
40
30
20
10
0
−10
var_120
var_121
var_122
var_123
var_124
var_125
var_126
var_127
var_128
var_129

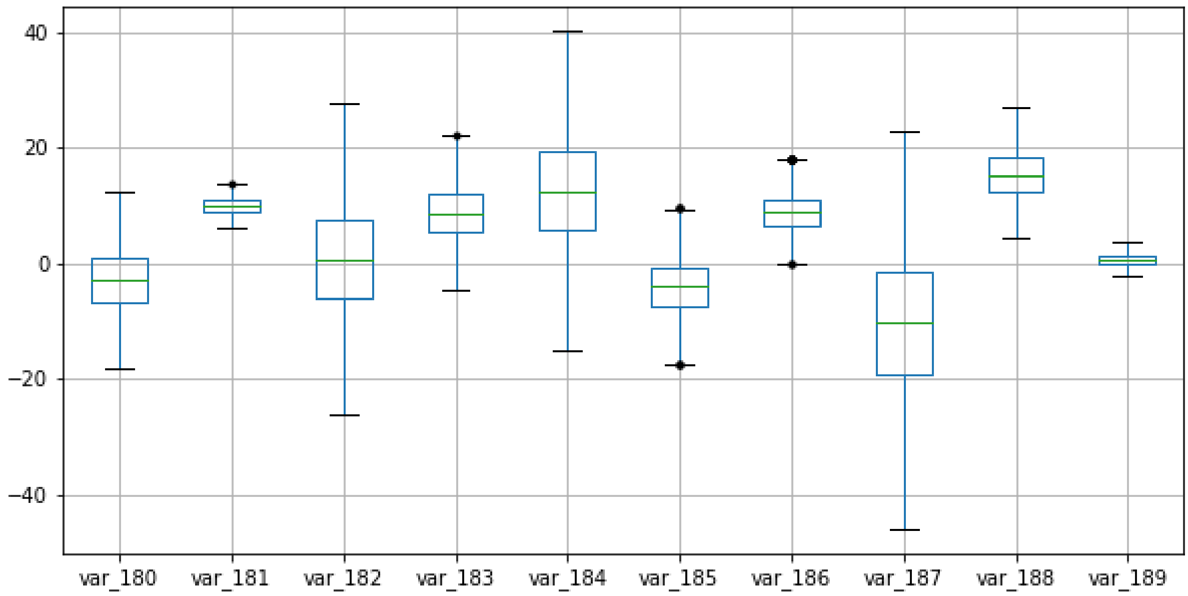

50
40
30
20
10
0
−10
−20
−30
var_130
var_131
var_132
var_133
var_134
var_135
var_136
var_137
var_138
var_139

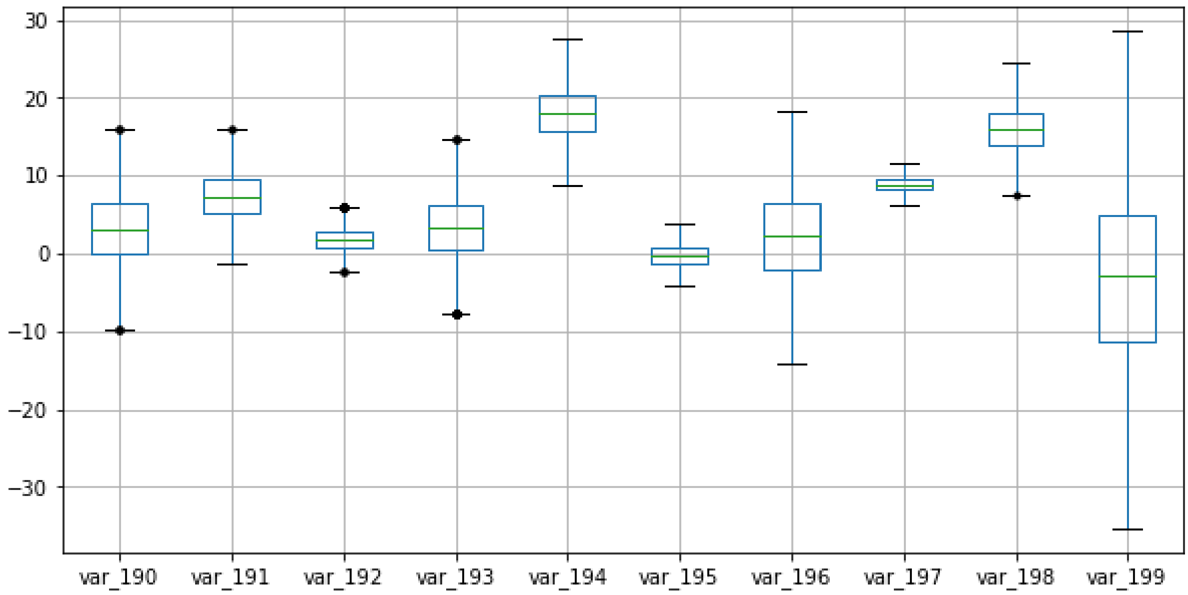

30
20
10
0
−10
−20
var_140
var_141
var_142
var_143
var_144
var_145
var_146
var_147
var_148
var_149


40
30
20
10
0
−10
−20
var_150
var_151
var_152
var_153
var_154
var_155
var_156
var_157
var_158
var_159

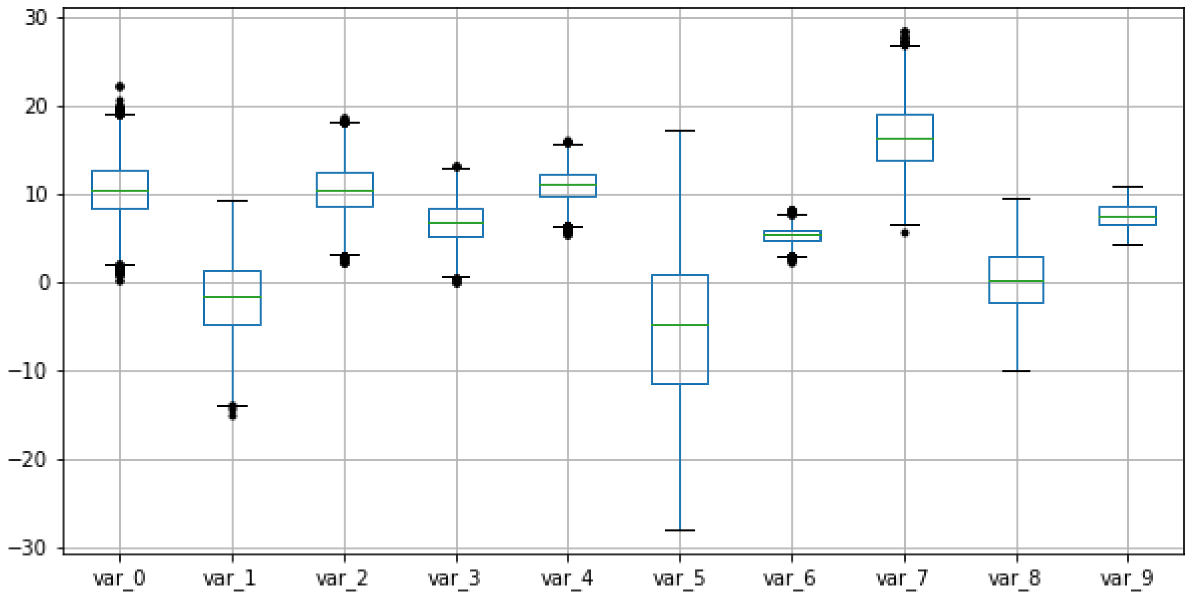

60
40
20
0
−20
var_160
var_161
var_162
var_163
var_164
var_165
var_166
var_167
var_168
var_169

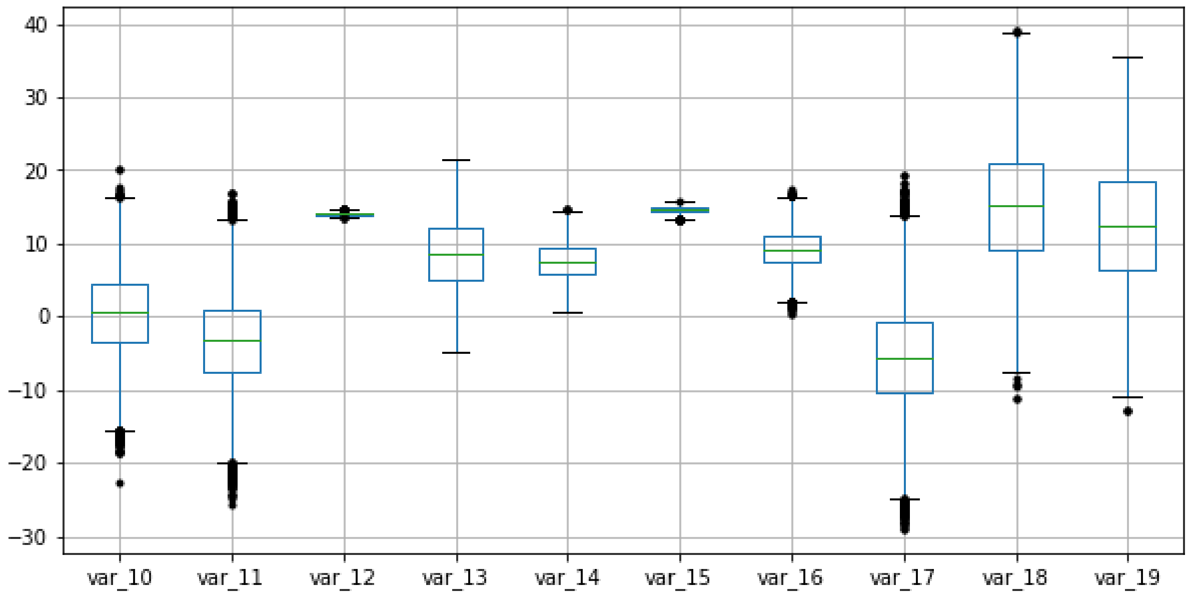

40
30
20
10
0
−10
−20
var_170
var_171
var_172
var_173
var_174
var_175
var_176
var_177
var_178
var_179

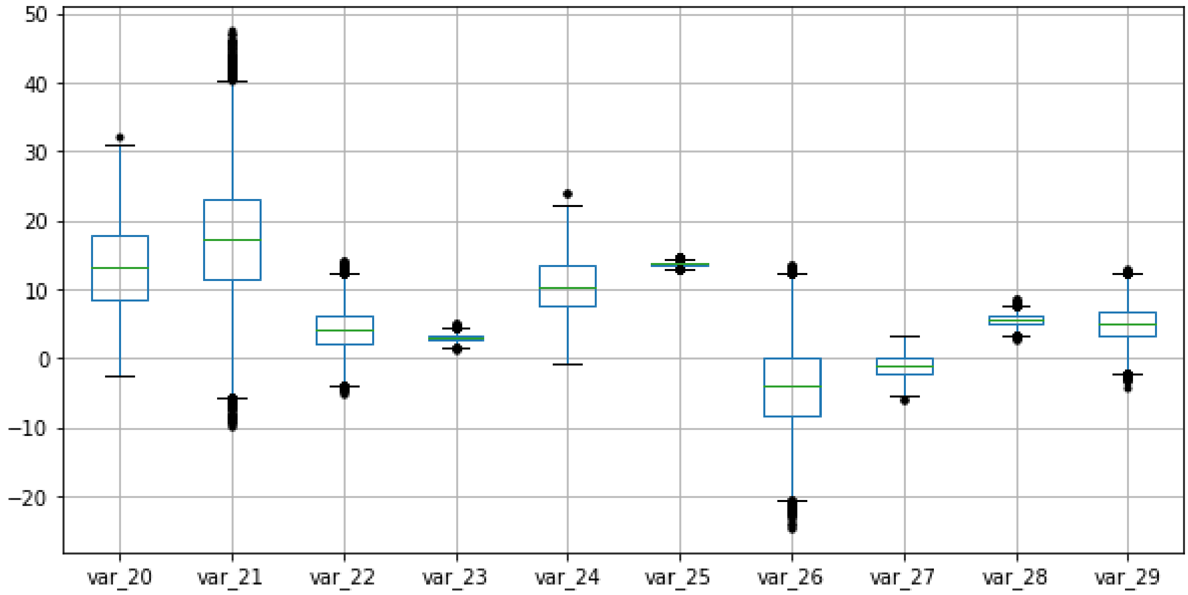

40
20
0
−20
−40
var_180
var_181
var_182
var_183
var_184
var_185
var_186
var_187
var_188
var_189

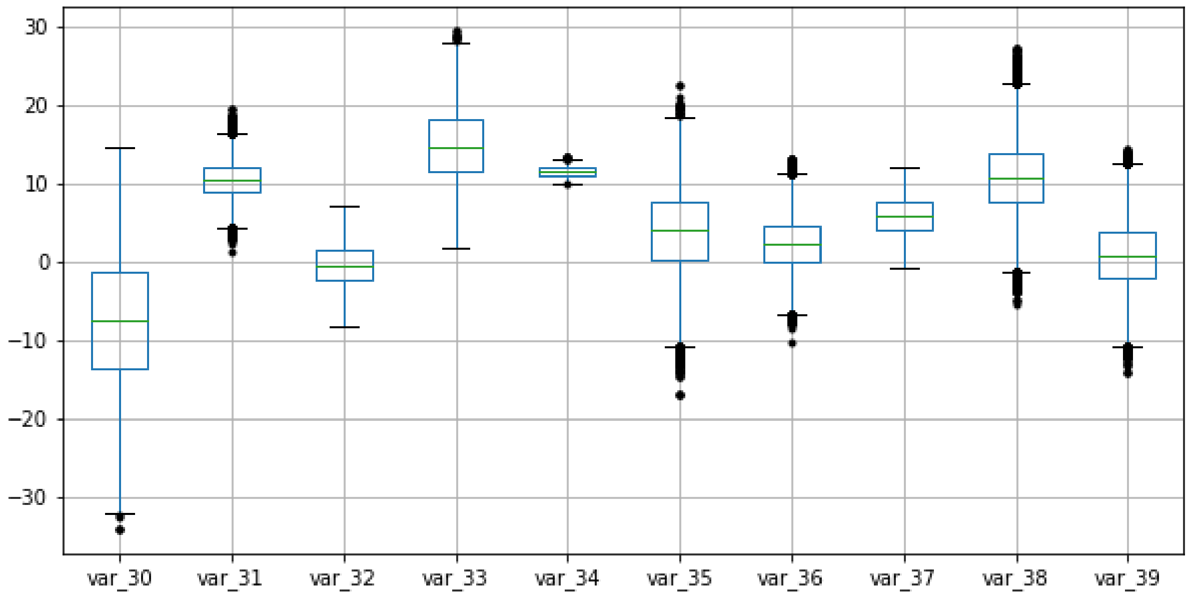


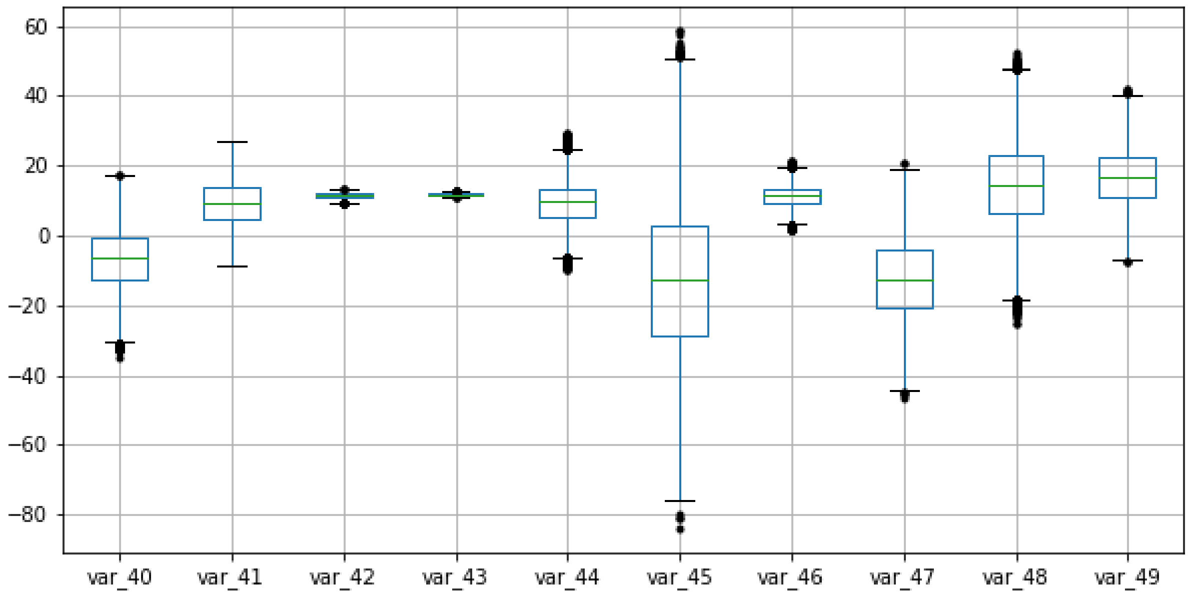

```python
# displaying a boxplot every n columns before removal of outliers (test)
for i in chunks:
    plt.show(test.boxplot(column = i, sym='k.', figsize=(10,5)))
```

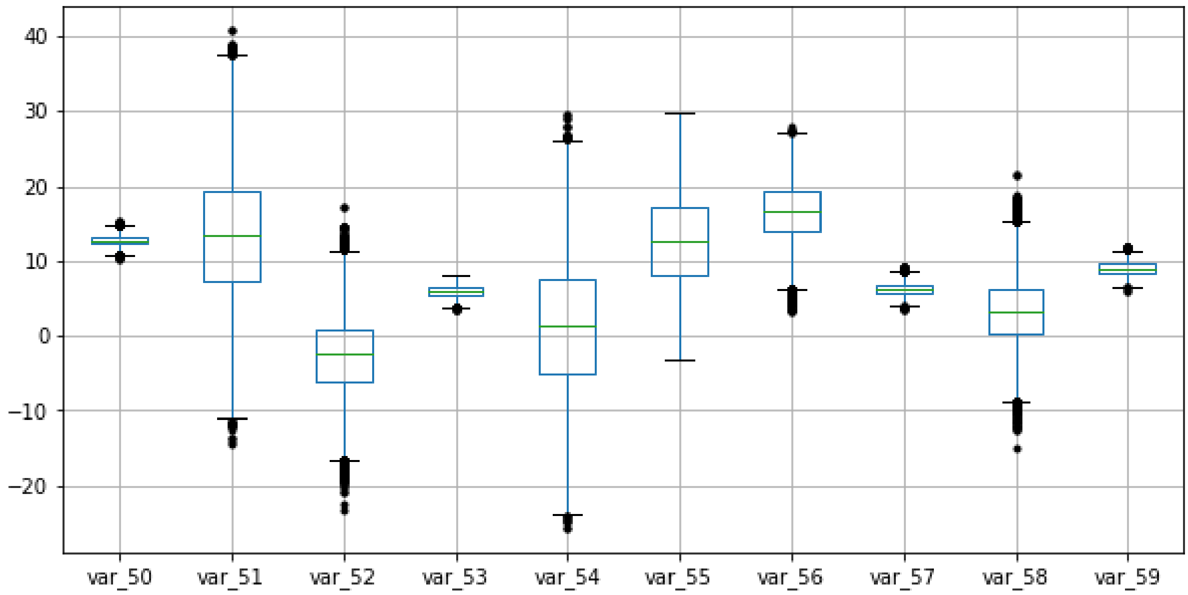


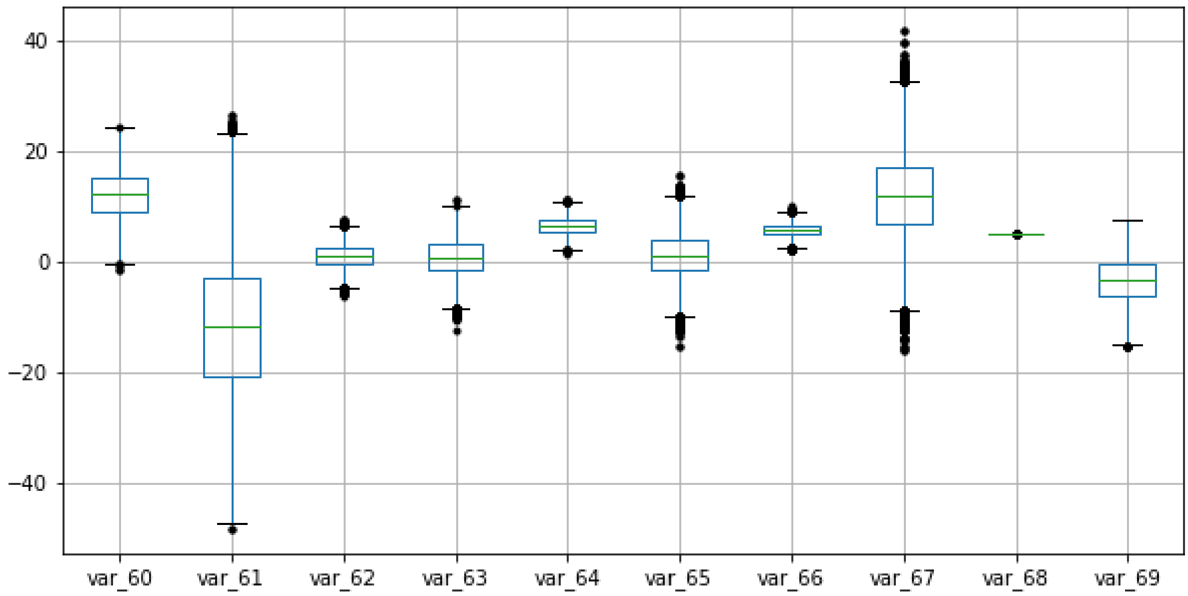

40
30
20
10
0
−10
−20
−30
var_10 var_11 var_12 var_13 var_14 var_15 var_16 var_17 var_18 var_19

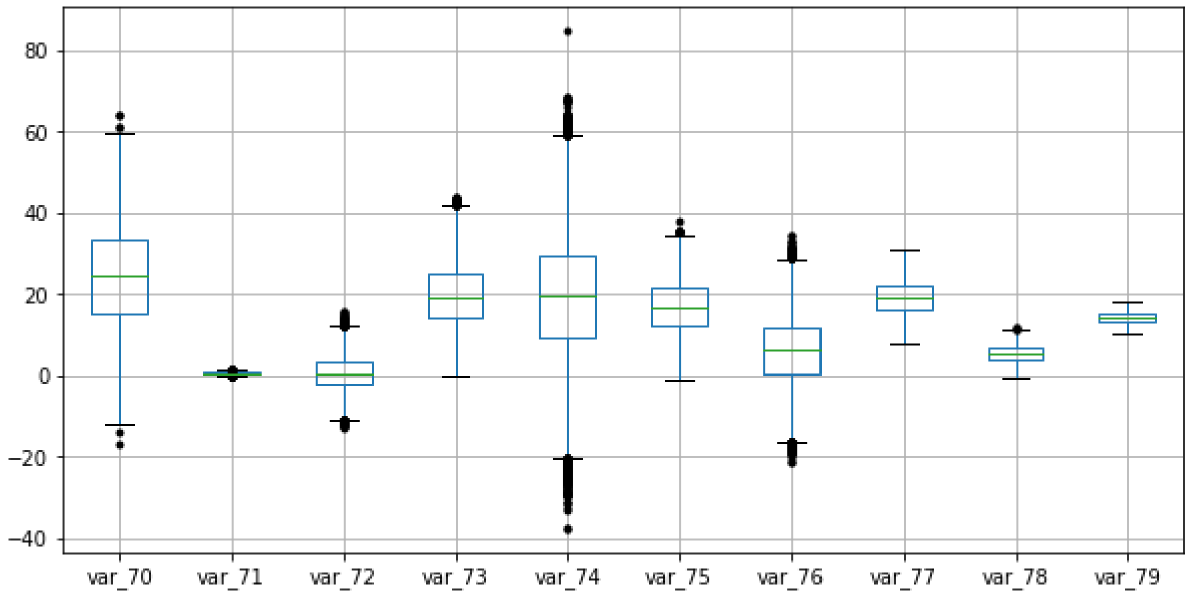

50
40
30
20
10
0
−10
−20
var_20 var_21 var_22 var_23 var_24 var_25 var_26 var_27 var_28 var_29

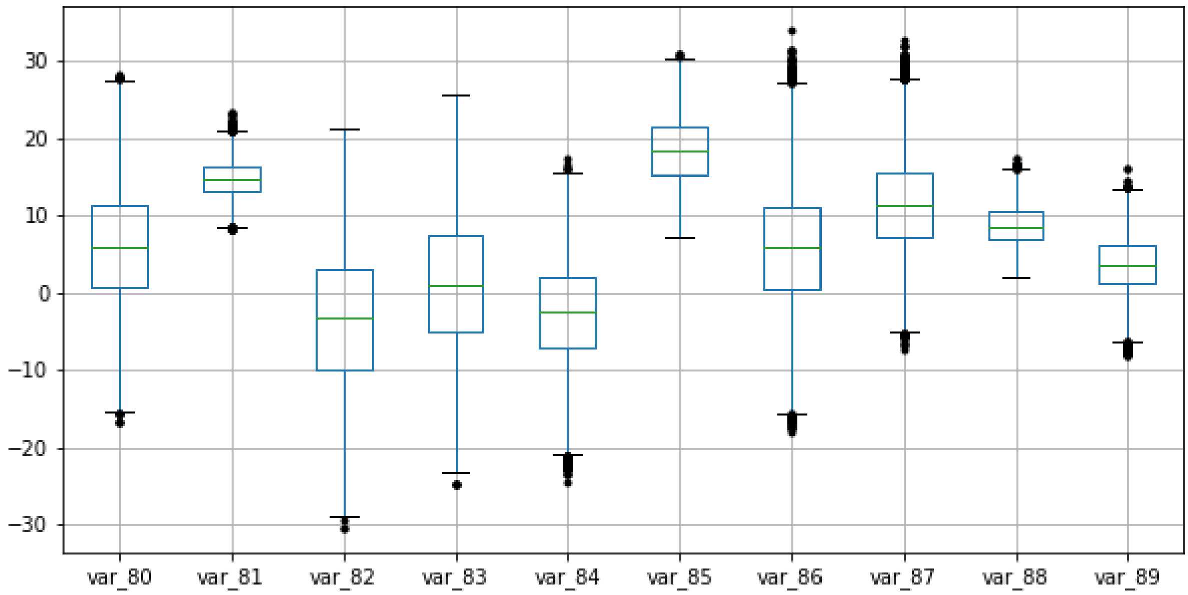

30
20
10
0
−10
−20
−30
var_30 var_31 var_32 var_33 var_34 var_35 var_36 var_37 var_38 var_39

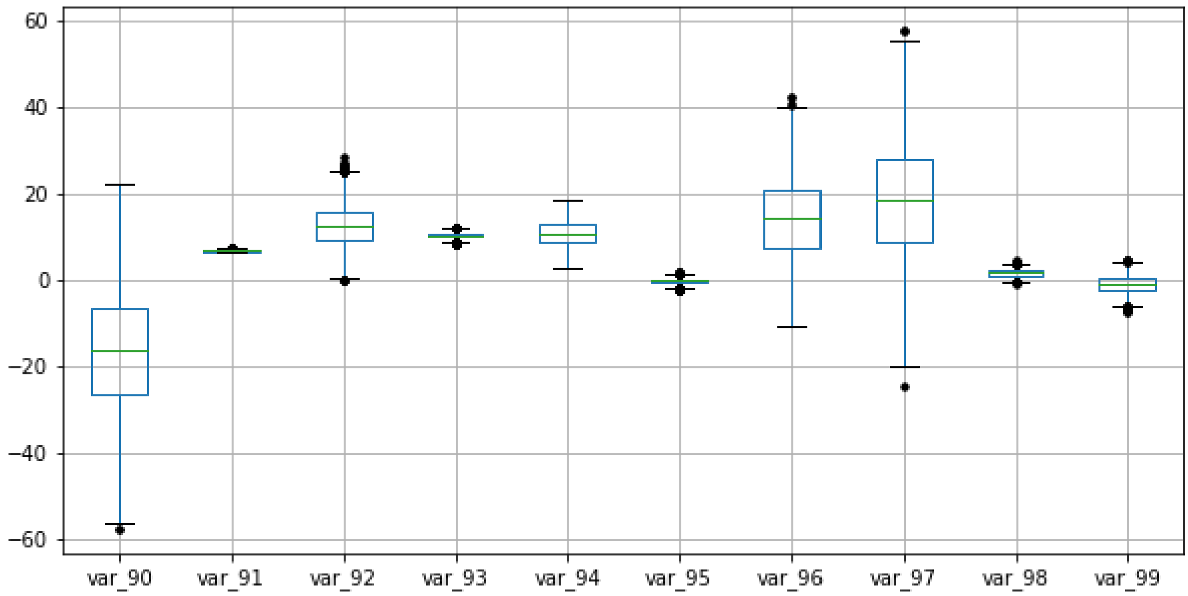

60
40
20
0
−20
−40
−60
−80
var_40
var_41
var_42
var_43
var_44
var_45
var_46
var_47
var_48
var_49

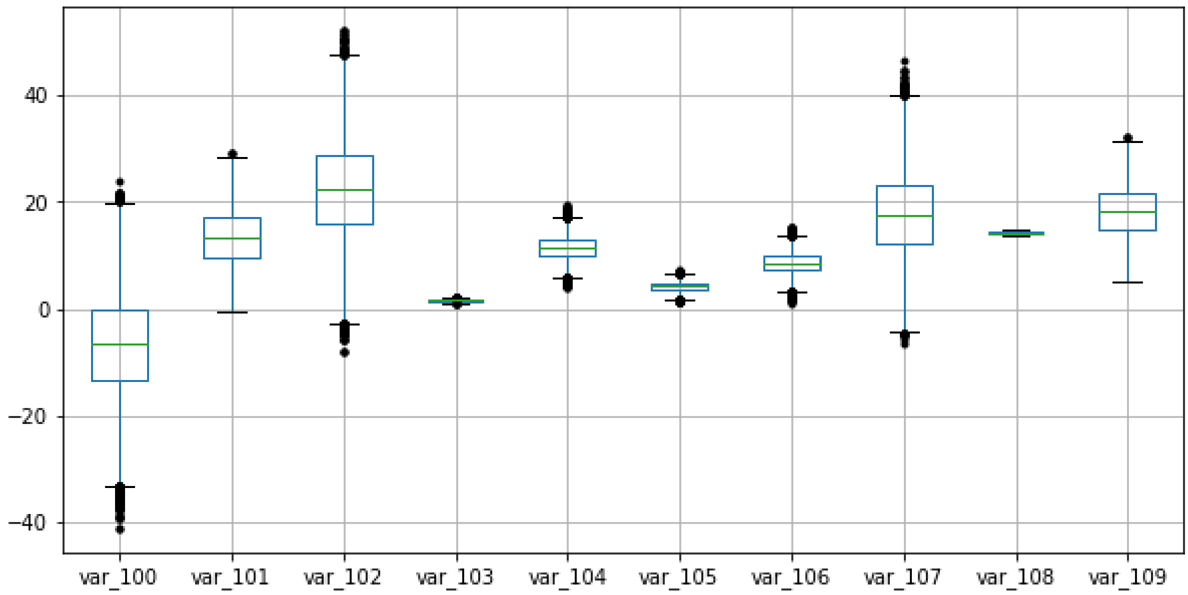

40
30
20
10
0
−10
−20
var_50
var_51
var_52
var_53
var_54
var_55
var_56
var_57
var_58
var_59

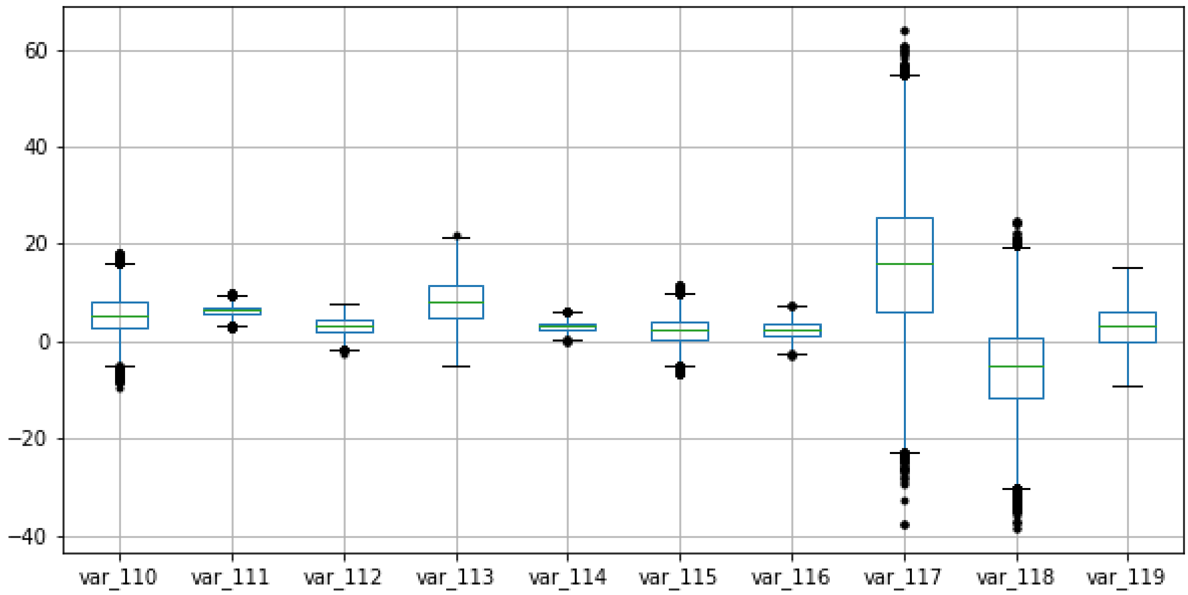

40
20
0
−20
−40
var_60
var_61
var_62
var_63
var_64
var_65
var_66
var_67
var_68
var_69

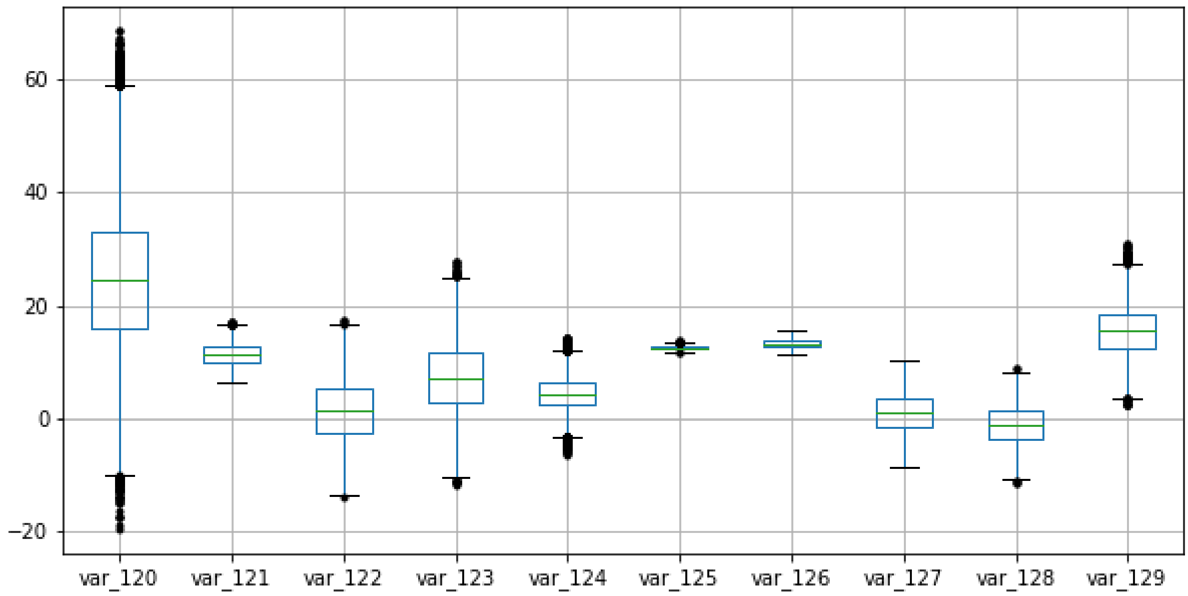

80
60
40
20
0
−20
−40
var_70
var_71
var_72
var_73
var_74
var_75
var_76
var_77
var_78
var_79

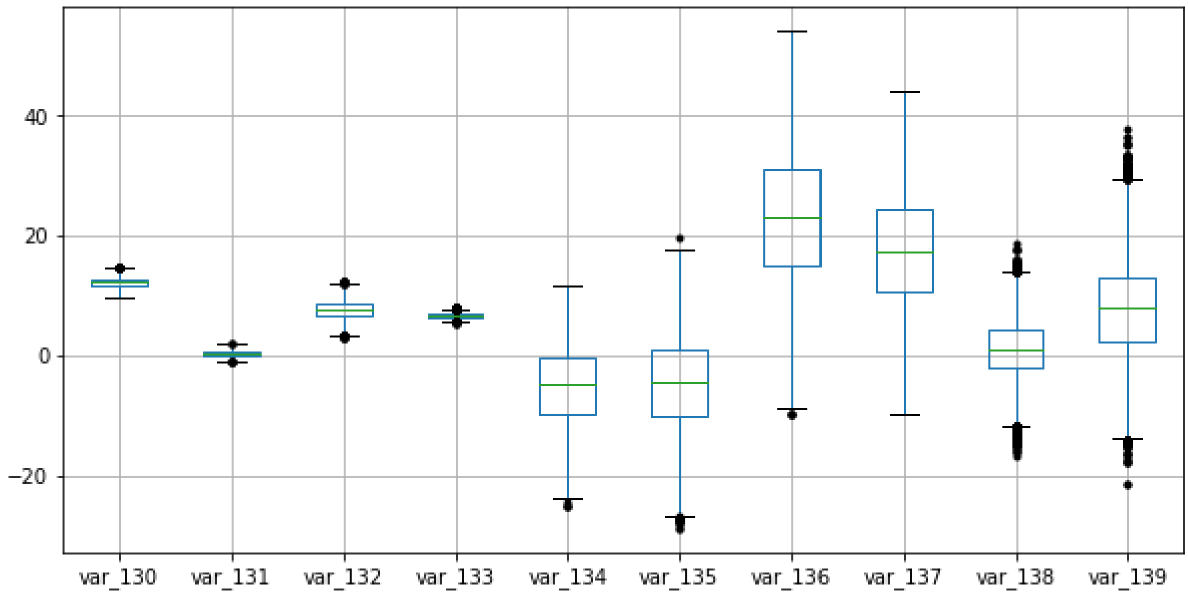

30
20
10
0
−10
−20
−30
var_80
var_81
var_82
var_83
var_84
var_85
var_86
var_87
var_88
var_89

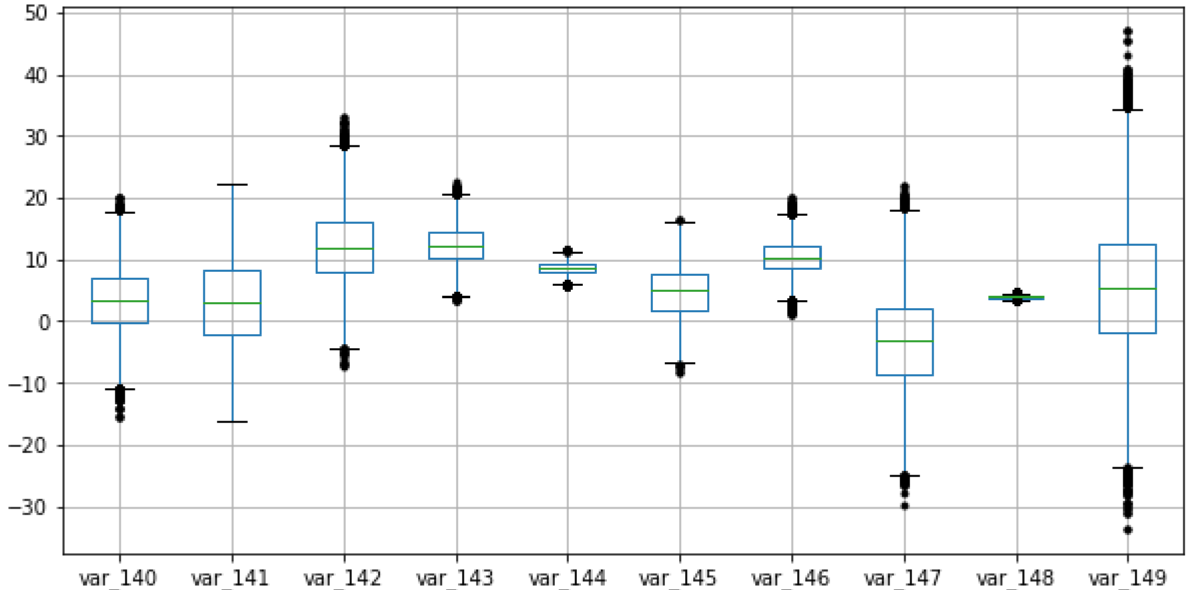

60
40
20
0
−20
−40
−60
var_90
var_91
var_92
var_93
var_94
var_95
var_96
var_97
var_98
var_99

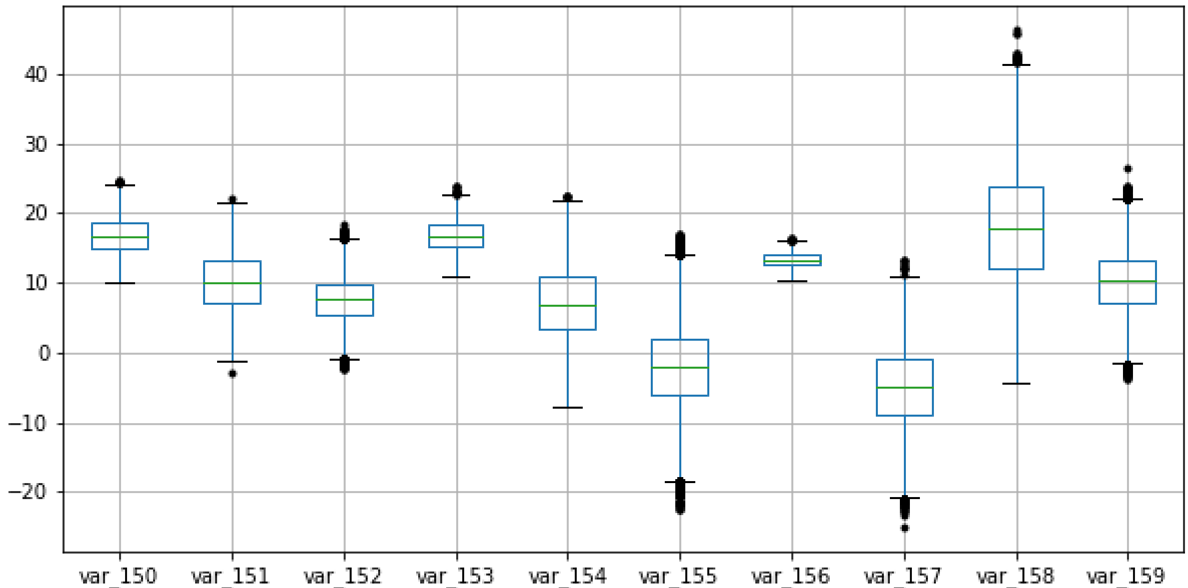

40
20
0
−20
−40
var_100 var_101 var_102 var_103 var_104 var_105 var_106 var_107 var_108 var_109

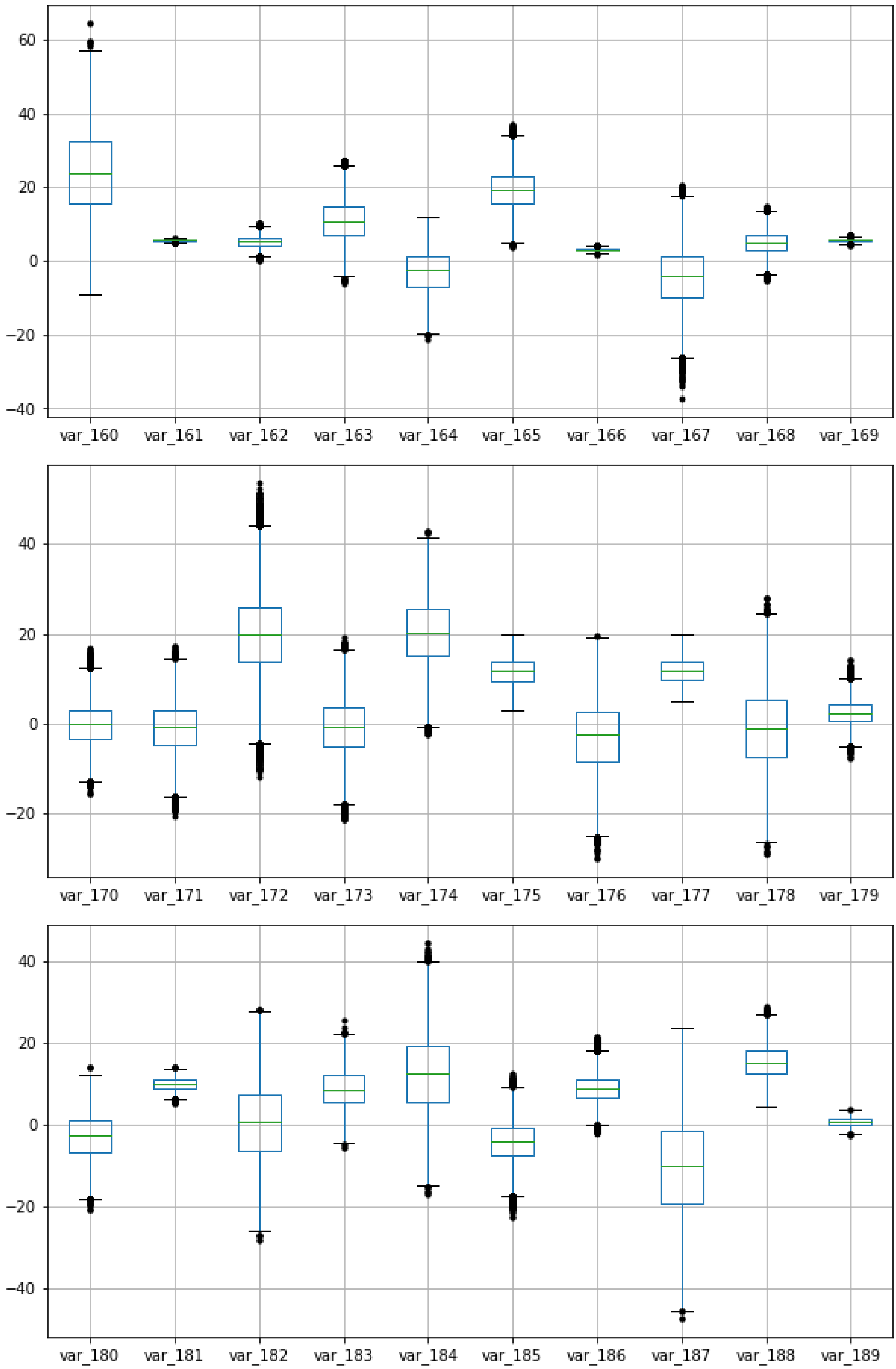

60
40
20
0
−20
−40
var_110 var_111 var_112 var_113 var_114 var_115 var_116 var_117 var_118 var_119

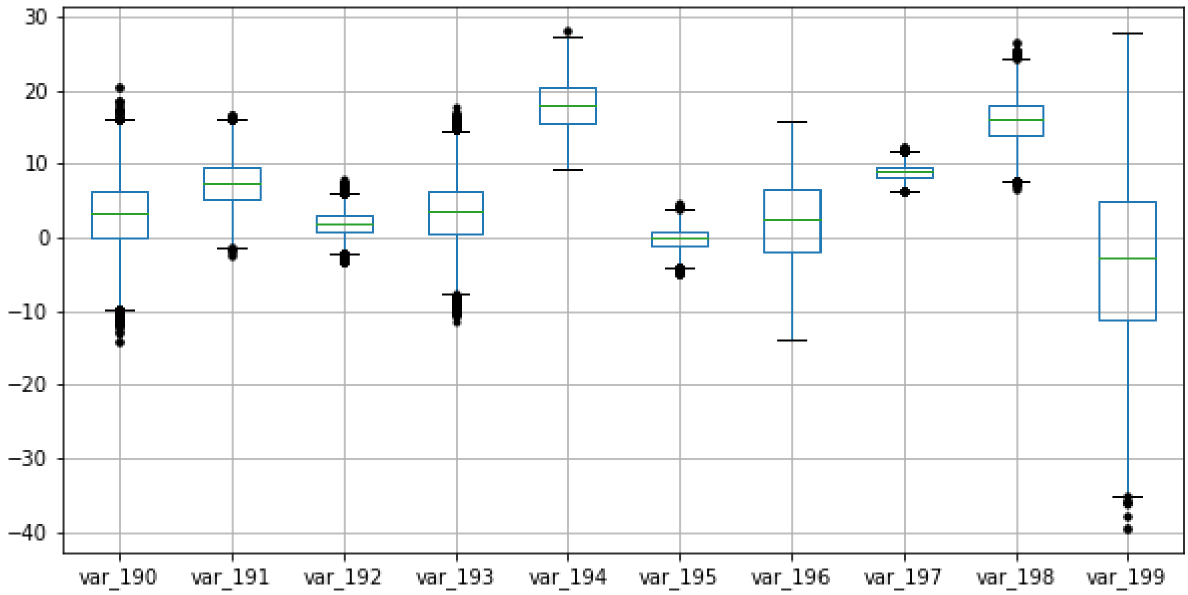

60
40
20
0
−20
var_120 var_121 var_122 var_123 var_124 var_125 var_126 var_127 var_128 var_129


40
20
0
−20
var_130
var_131
var_132
var_133
var_134
var_135
var_136
var_137
var_138
var_139

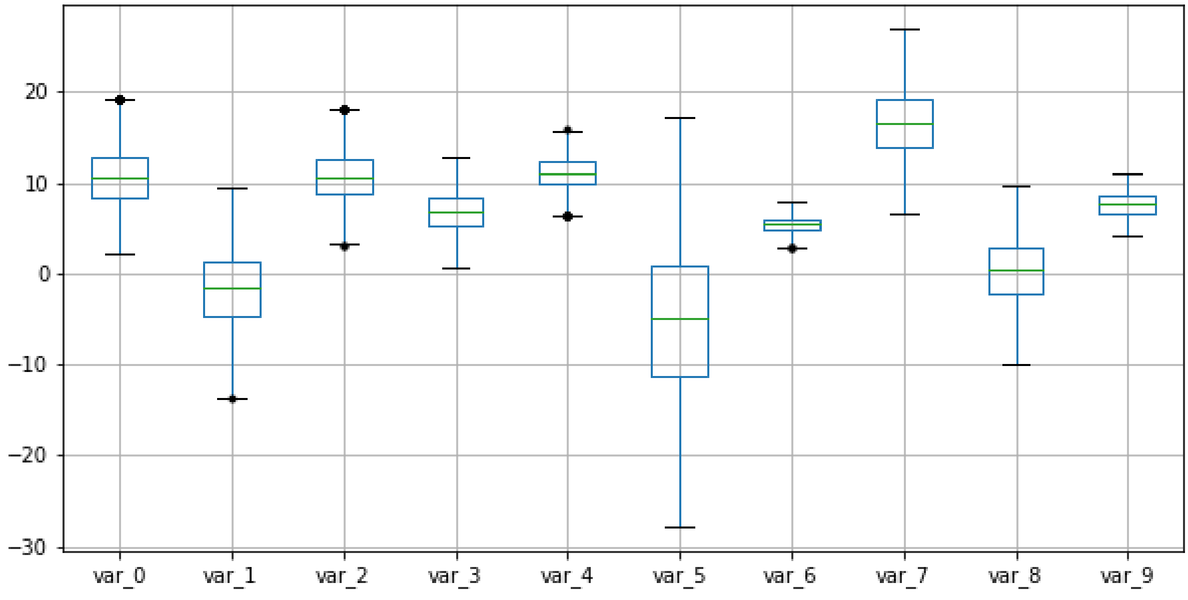

50
40
30
20
10
0
−10
−20
−30
var_140
var_141
var_142
var_143
var_144
var_145
var_146
var_147
var_148
var_149

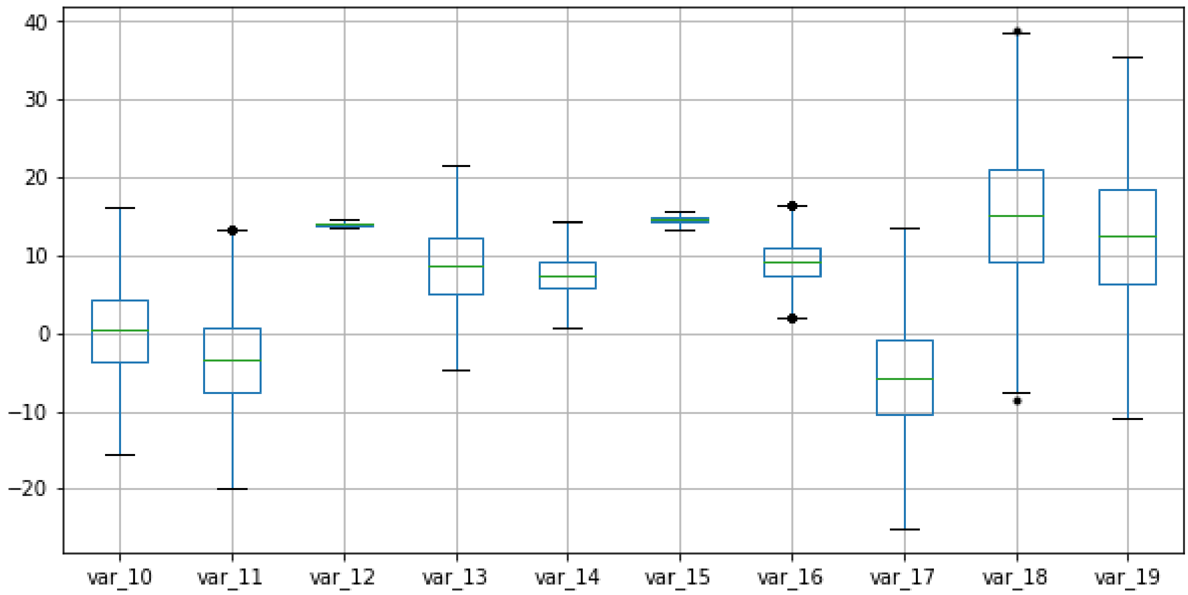

40
30
20
10
0
−10
−20
var_150
var_151
var_152
var_153
var_154
var_155
var_156
var_157
var_158
var_159

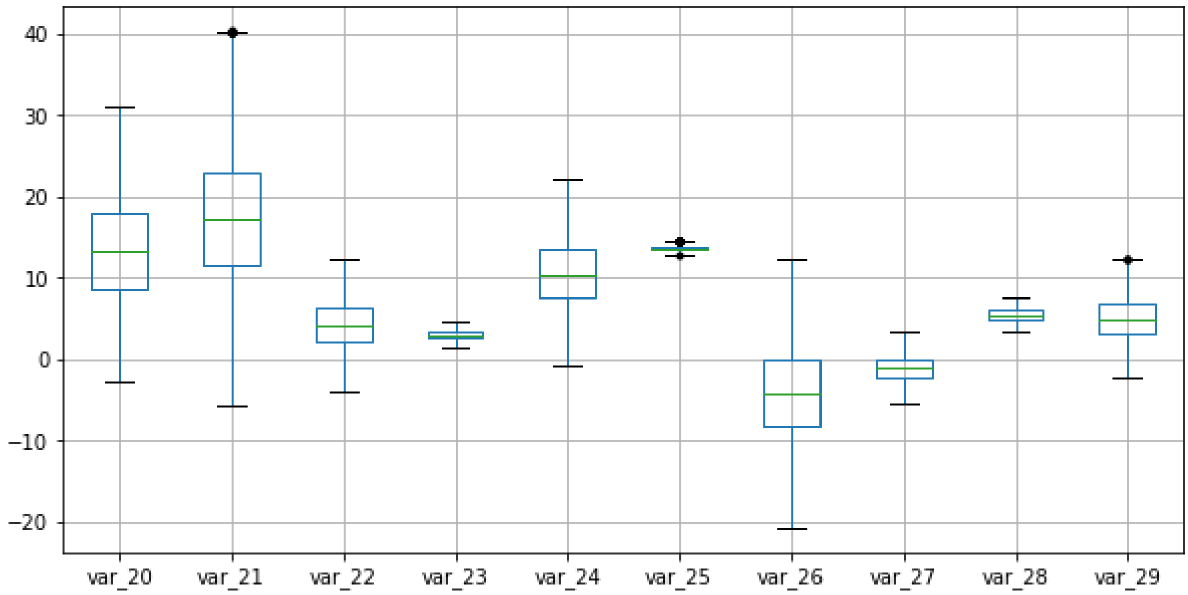

60
40
20
0
−20
−40
var_160
var_161
var_162
var_163
var_164
var_165
var_166
var_167
var_168
var_169
40
20
0
−20
var_170
var_171
var_172
var_173
var_174
var_175
var_176
var_177
var_178
var_179
40
20
0
−20
−40
var_180
var_181
var_182
var_183
var_184
var_185
var_186
var_187
var_188
var_189

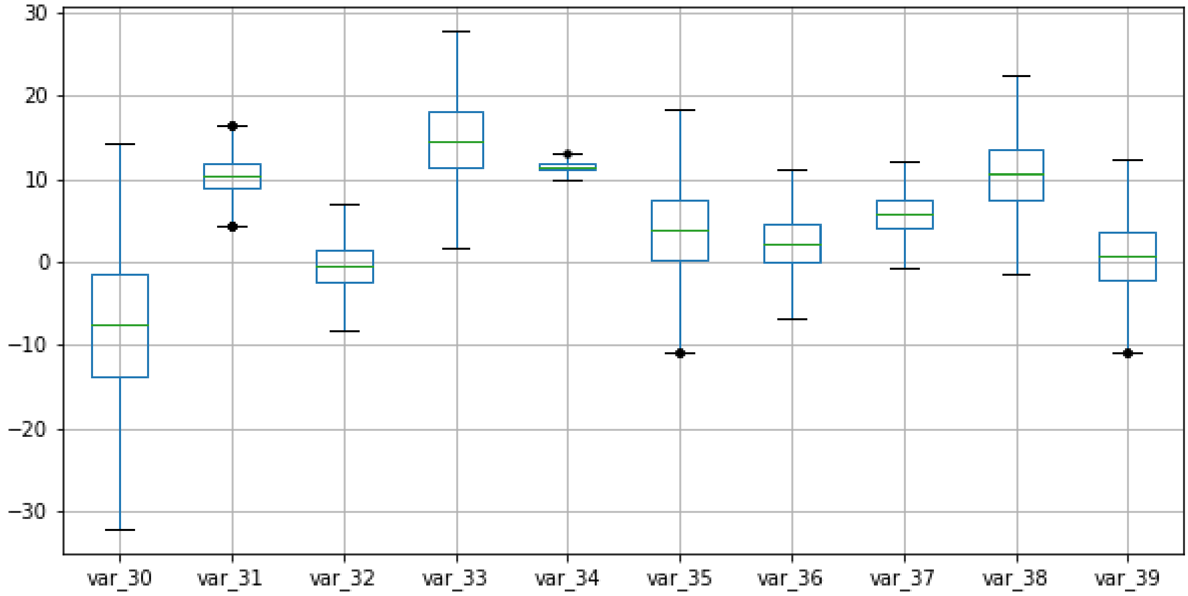


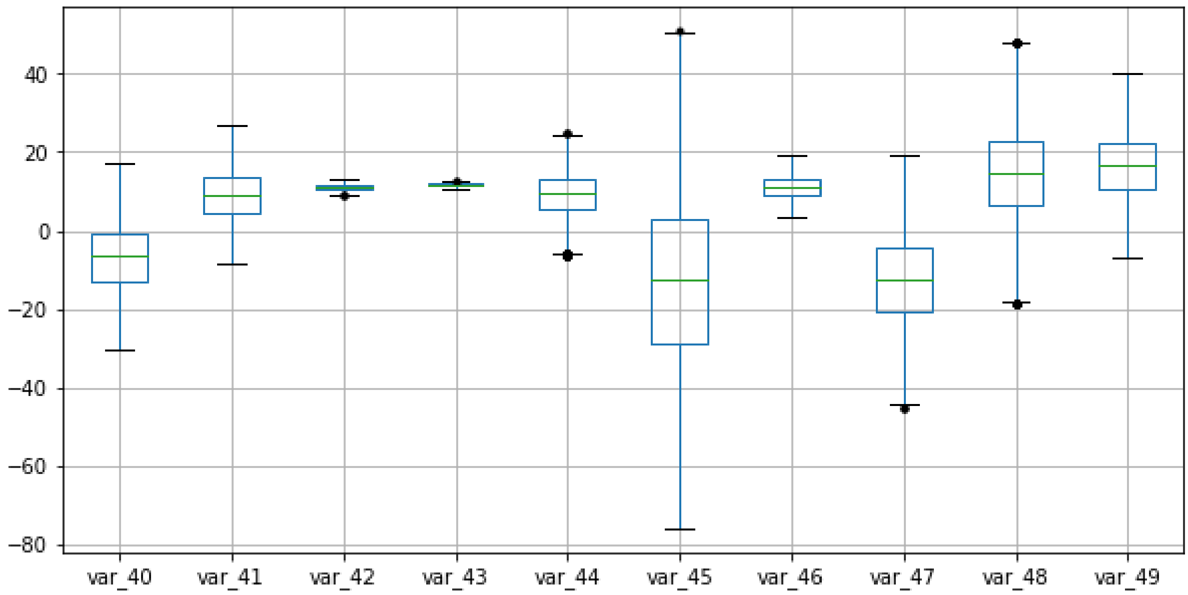

```
#outliers in each variable in test data
test_outliers = dict()
for col in [col for col in numerical_features]:
    q75,q25=np.percentile(train.loc[:,col],[75,25])
    Q=q75-q25
    min=q25-(Q*1.5)
    max=q75+(Q*1.5)
    #print(min)
    #print(max)
    test=test.drop(test[test.loc[:,col]<min].index)
    test=test.drop(test[test.loc[:,col]>max].index)

# displaying a boxplot every n columns after removal of outliers:
for i in chunks:
    plt.show(test.boxplot(column = i, sym='k.', figsize=(10,5)))
```

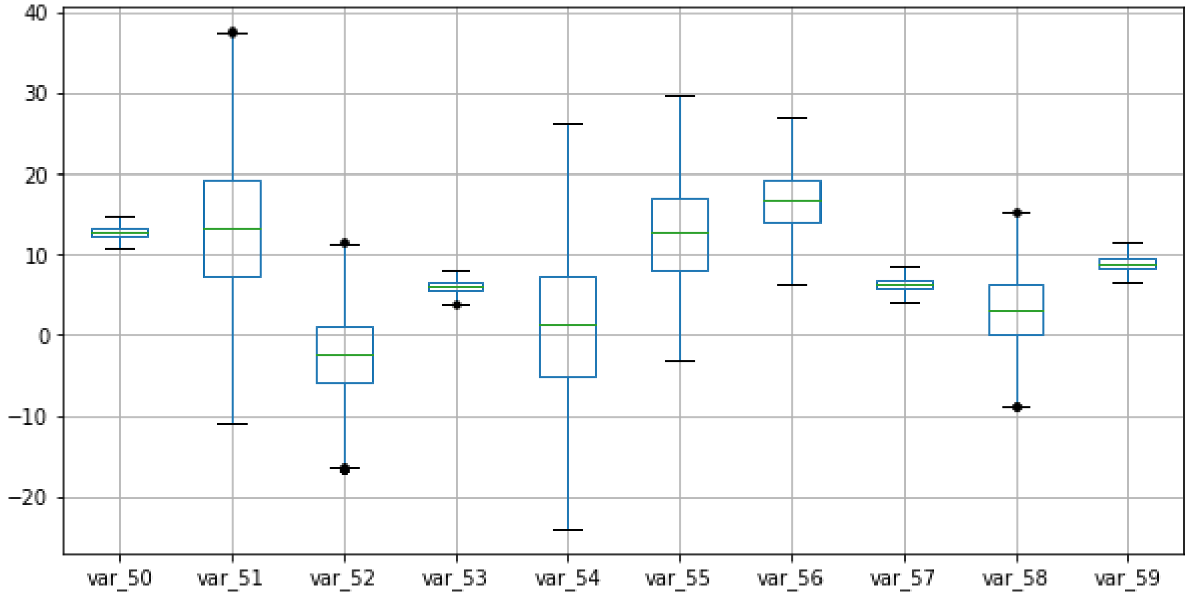


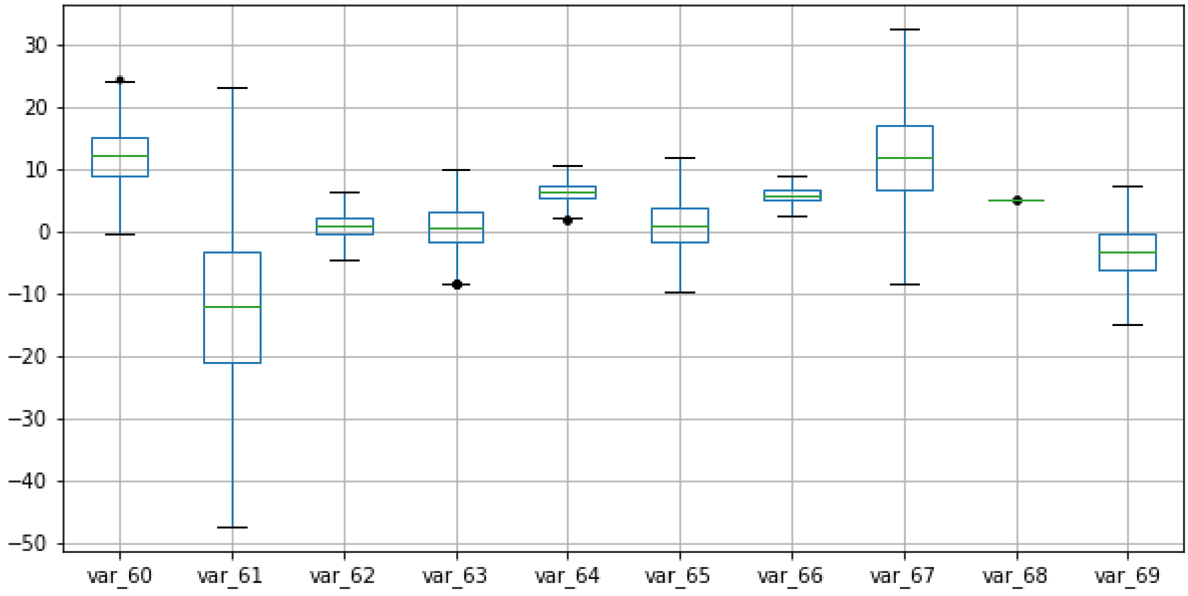

40
30
20
10
0
−10
−20
var_10
var_11
var_12
var_13
var_14
var_15
var_16
var_17
var_18
var_19

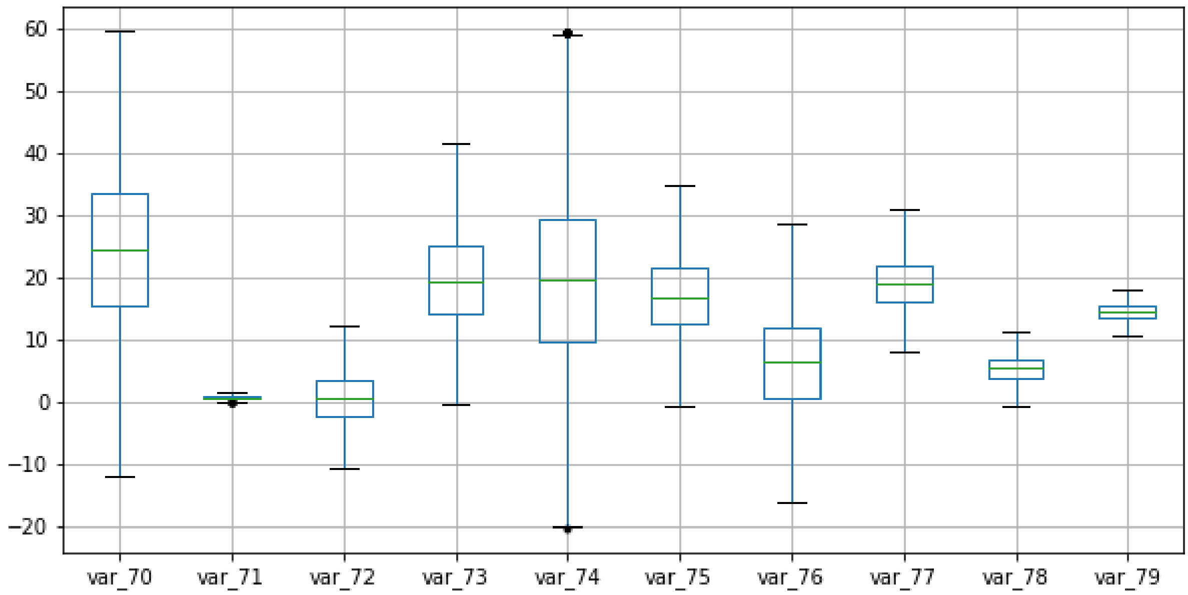

40
30
20
10
0
−10
−20
var_20
var_21
var_22
var_23
var_24
var_25
var_26
var_27
var_28
var_29

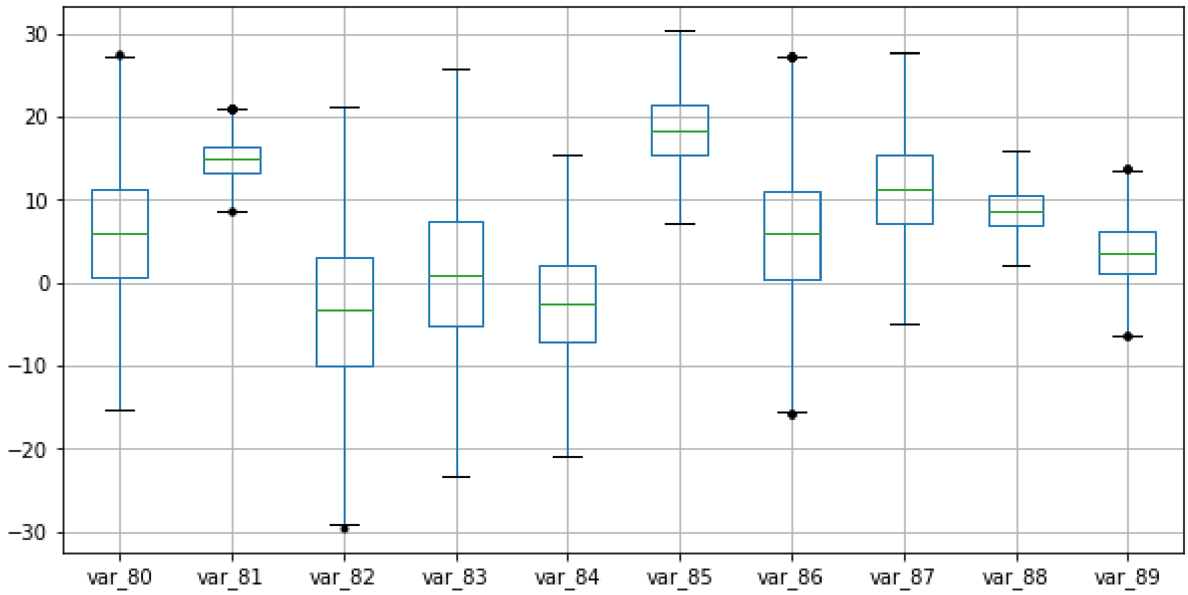

30
20
10
0
−10
−20
−30
var_30
var_31
var_32
var_33
var_34
var_35
var_36
var_37
var_38
var_39

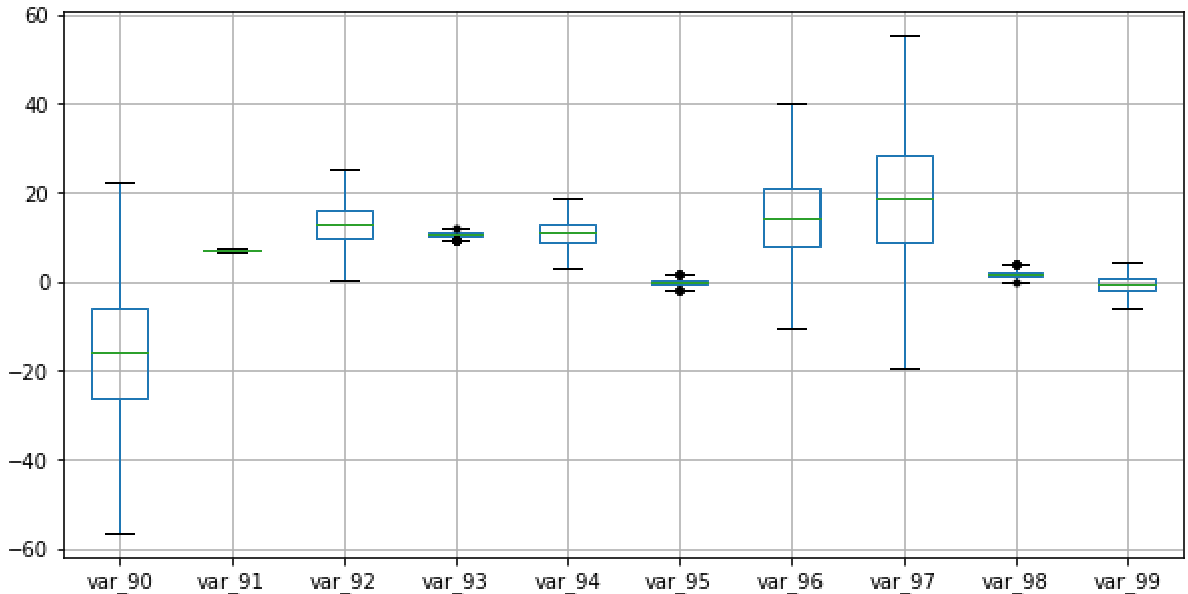

40
20
0
−20
−40
−60
−80
var_40
var_41
var_42
var_43
var_44
var_45
var_46
var_47
var_48
var_49

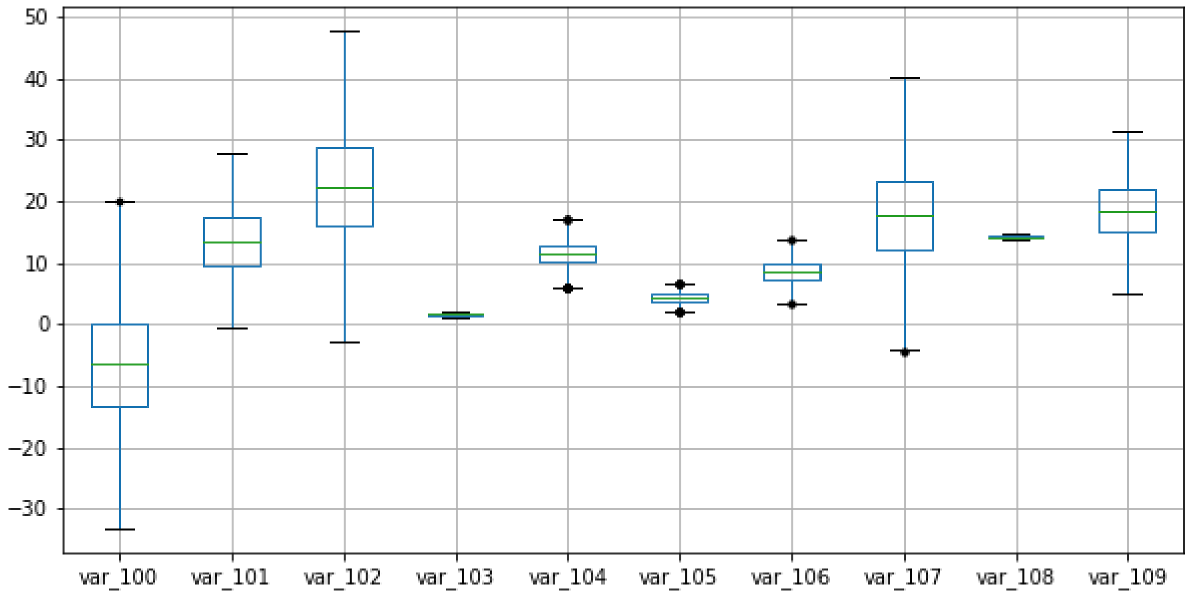

40
30
20
10
0
−10
−20
var_50
var_51
var_52
var_53
var_54
var_55
var_56
var_57
var_58
var_59

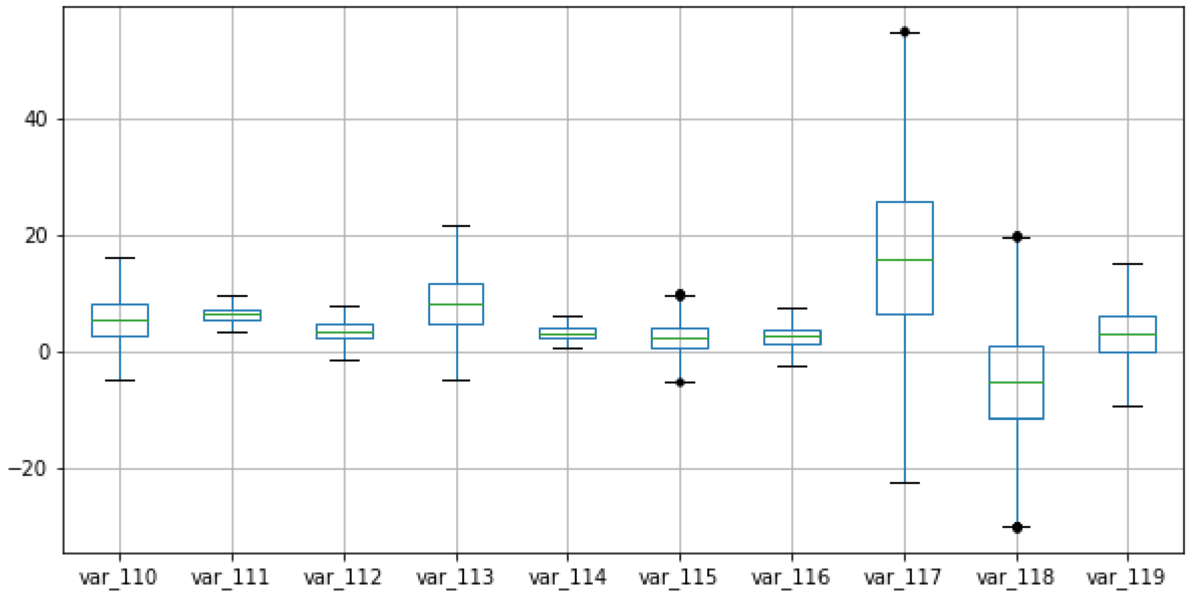

30
20
10
0
−10
−20
−30
−40
−50
var_60
var_61
var_62
var_63
var_64
var_65
var_66
var_67
var_68
var_69

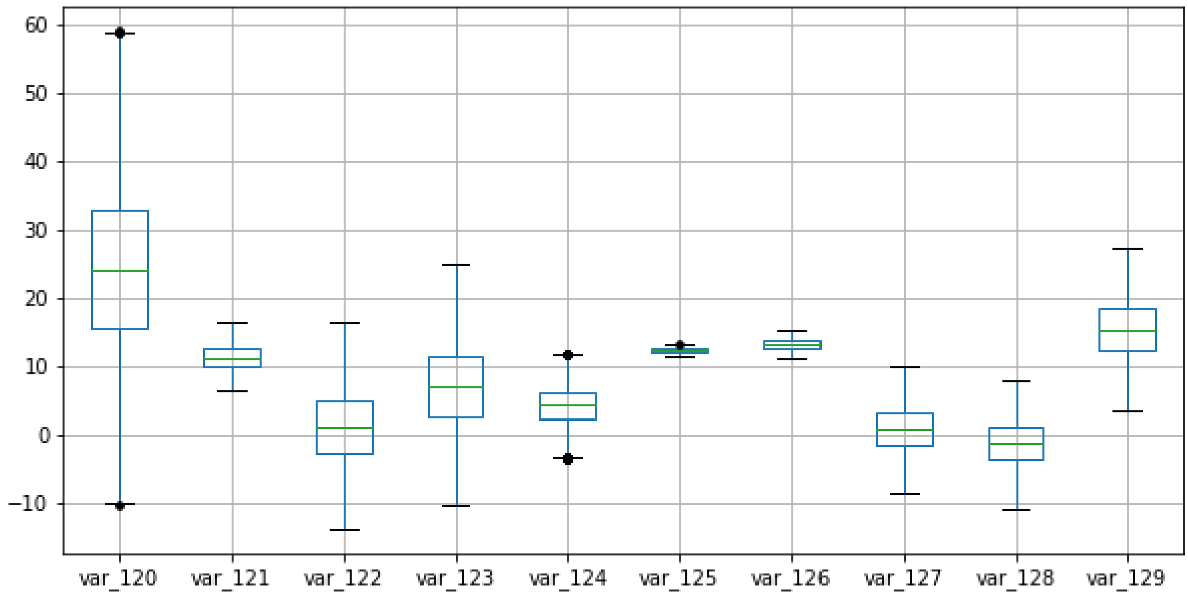

60
50
40
30
20
10
0
−10
−20
var_70
var_71
var_72
var_73
var_74
var_75
var_76
var_77
var_78
var_79

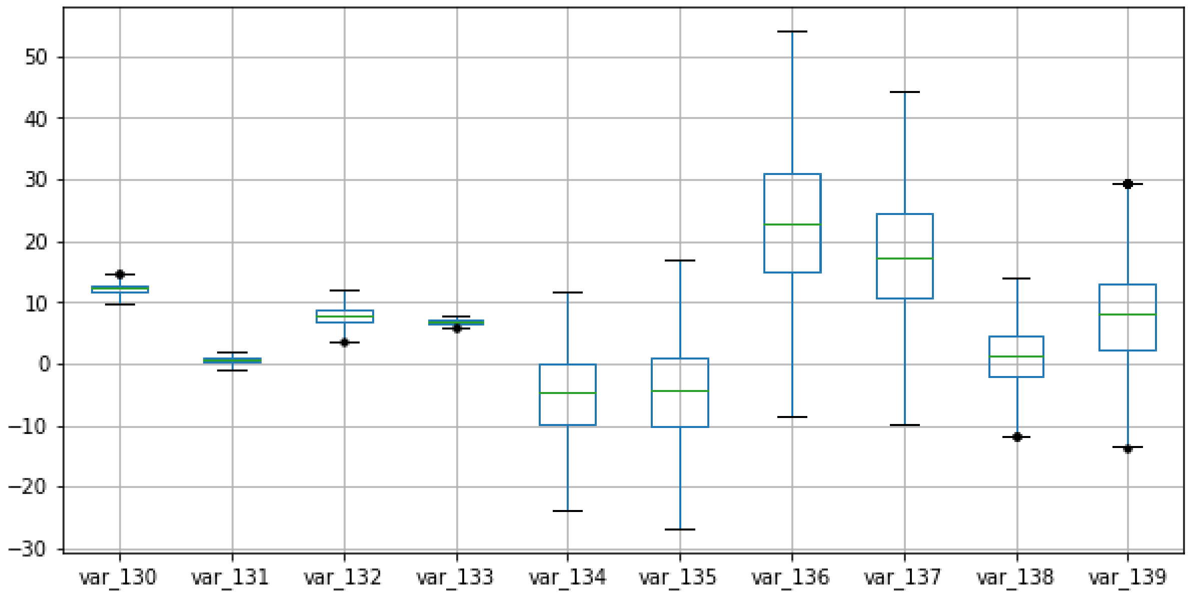

30
20
10
0
−10
−20
−30
var_80
var_81
var_82
var_83
var_84
var_85
var_86
var_87
var_88
var_89

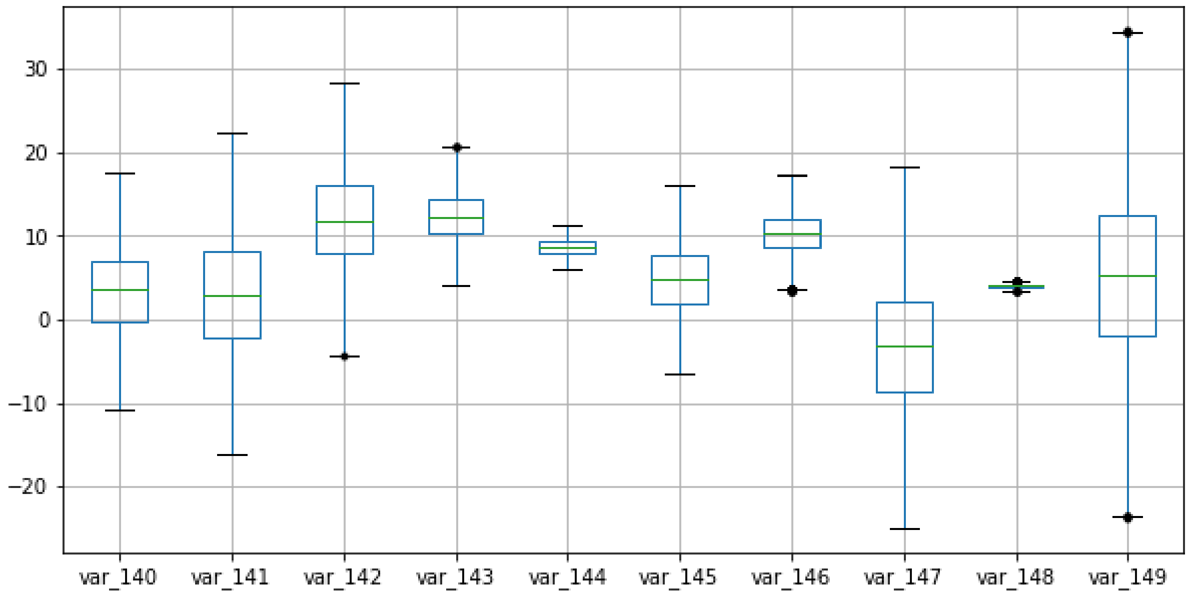

60
40
20
0
−20
−40
−60
var_90
var_91
var_92
var_93
var_94
var_95
var_96
var_97
var_98
var_99

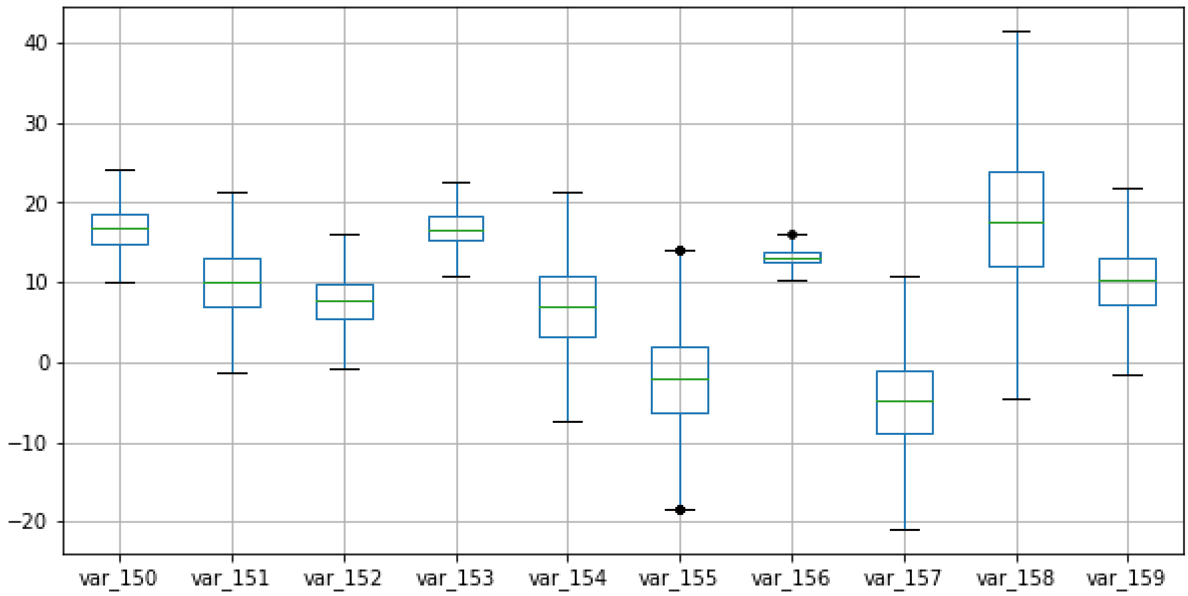

50
40
30
20
10
0
−10
−20
−30
var_100
var_101
var_102
var_103
var_104
var_105
var_106
var_107
var_108
var_109

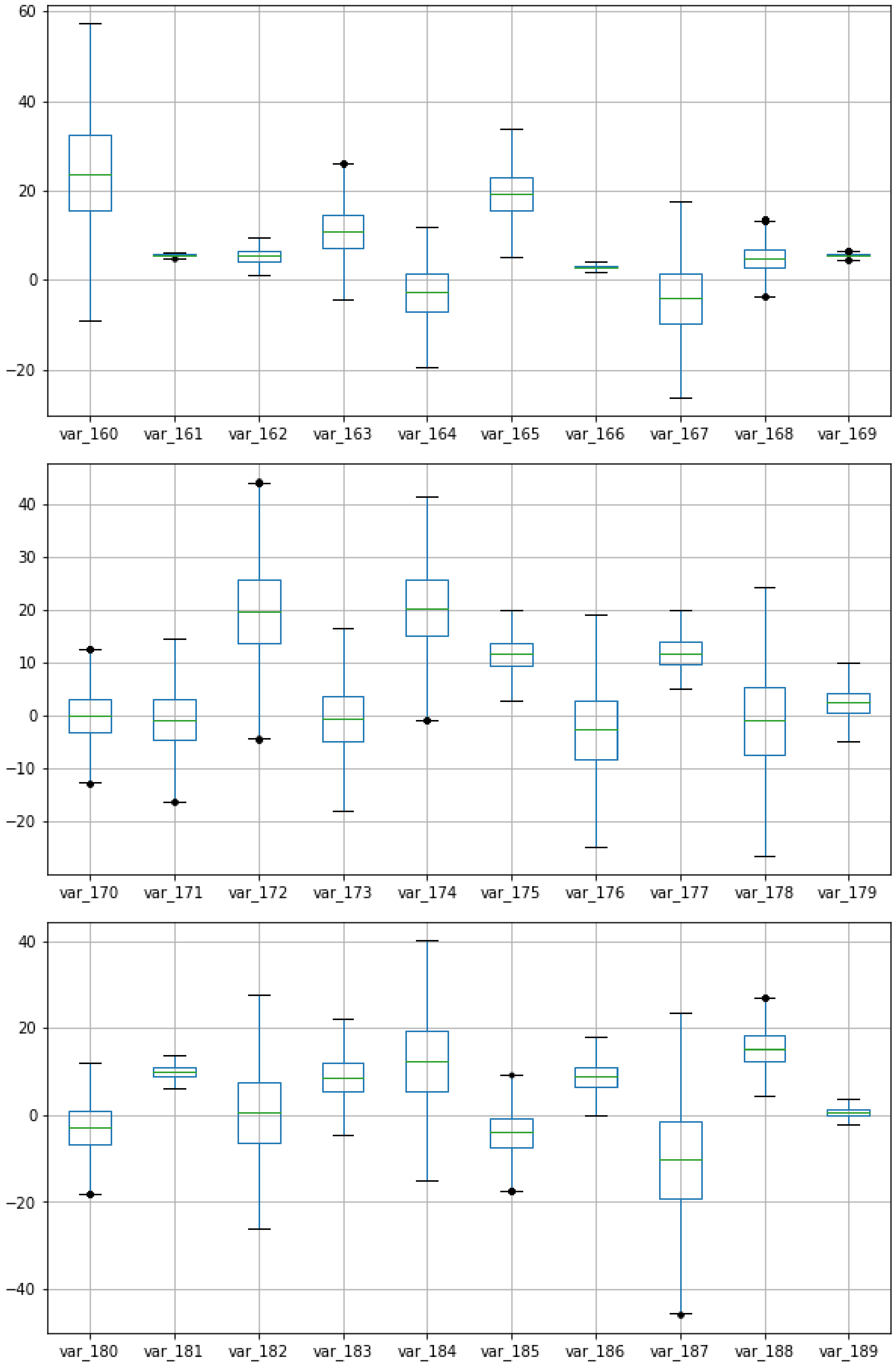

40
20
0
−20
var_110
var_111
var_112
var_113
var_114
var_115
var_116
var_117
var_118
var_119

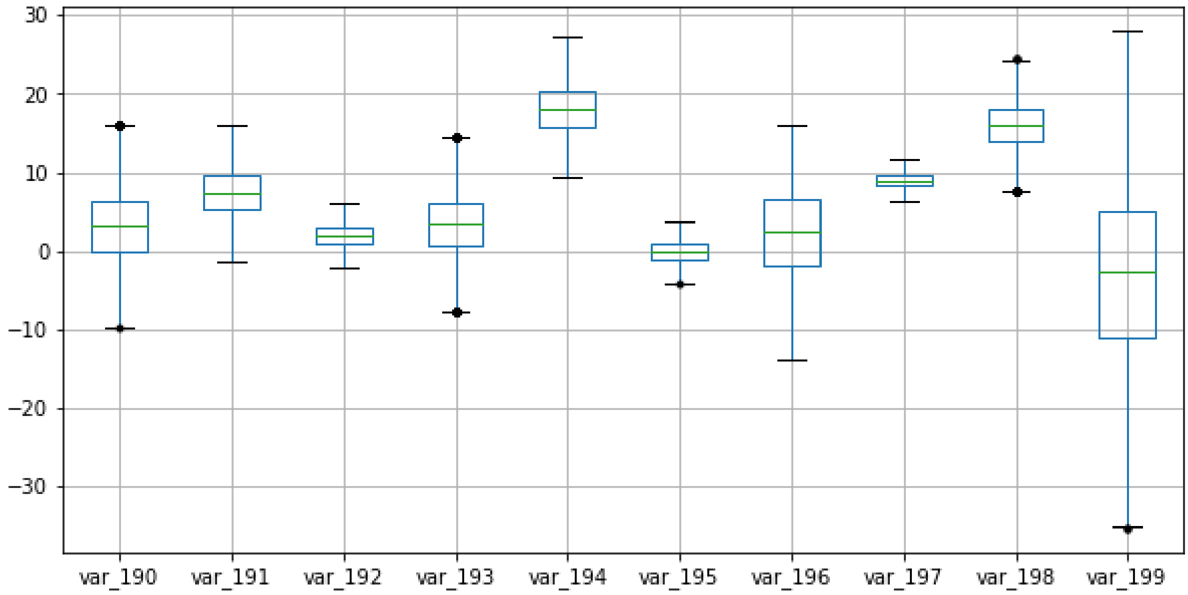

60
50
40
30
20
10
0
−10
var_120
var_121
var_122
var_123
var_124
var_125
var_126
var_127
var_128
var_129


50
40
30
20
10
0
−10
−20
−30
var_130
var_131
var_132
var_133
var_134
var_135
var_136
var_137
var_138
var_139


30
20
10
0
−10
−20
var_140
var_141
var_142
var_143
var_144
var_145
var_146
var_147
var_148
var_149


40
30
20
10
0
−10
−20
var_150
var_151
var_152
var_153
var_154
var_155
var_156
var_157
var_158
var_159


60
40
20
0
−20
var_160
var_161
var_162
var_163
var_164
var_165
var_166
var_167
var_168
var_169
40
30
20
10
0
−10
−20
var_170
var_171
var_172
var_173
var_174
var_175
var_176
var_177
var_178
var_179
40
20
0
−20
−40
var_180
var_181
var_182
var_183
var_184
var_185
var_186
var_187
var_188
var_189

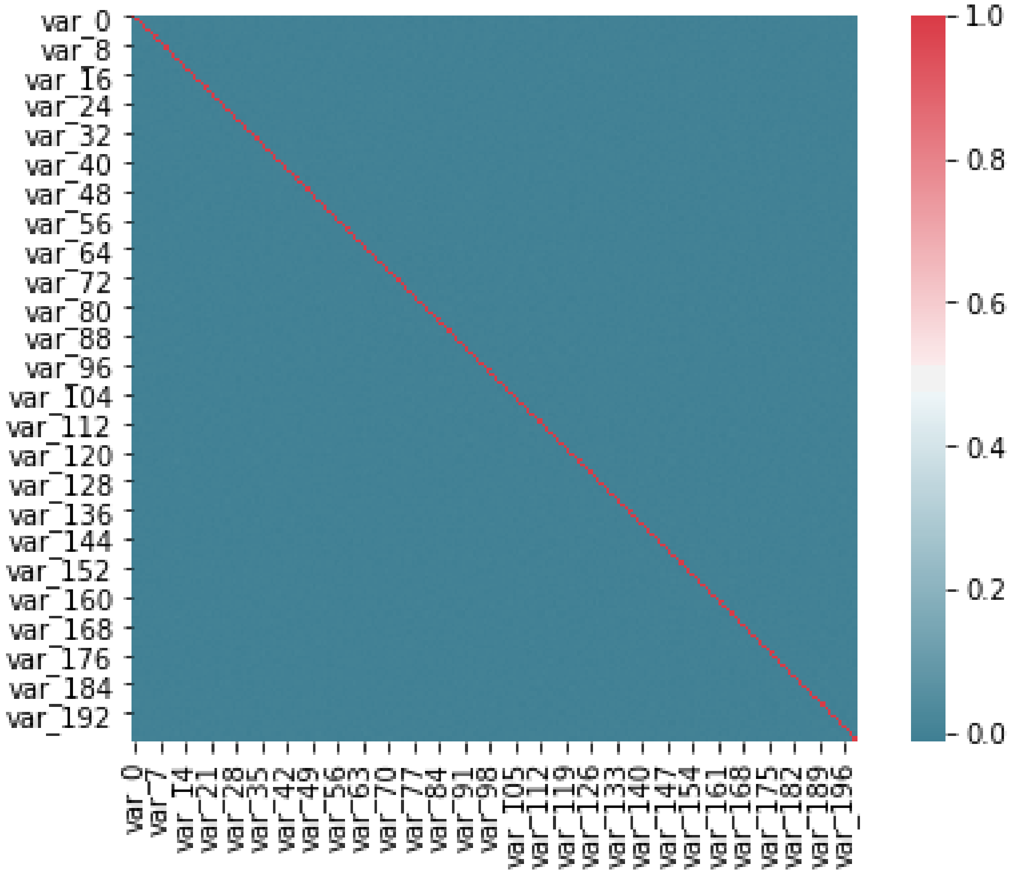



```
#shape of train and test data after removal of outliers
train.shape,test.shape
```

```
((175073, 202), (174011, 201))
```

```
dfcorr=train.loc[:,numerical_features]
dfcorr.shape
f,ax=plt.subplots(figsize=(7,5))
corr=dfcorr.corr()
sns.heatmap(corr,mask=np.zeros_like(corr,dtype=np.bool),cmap=sns.diverging_
palette(220,10,as_cmap=True),square=True,ax=ax)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x16b7f3d8ac8>
```



```
# Compute the correlation matrix
np.fill_diagonal(corr.values,np.nan)
corr.max().max(),corr.min().min()
```

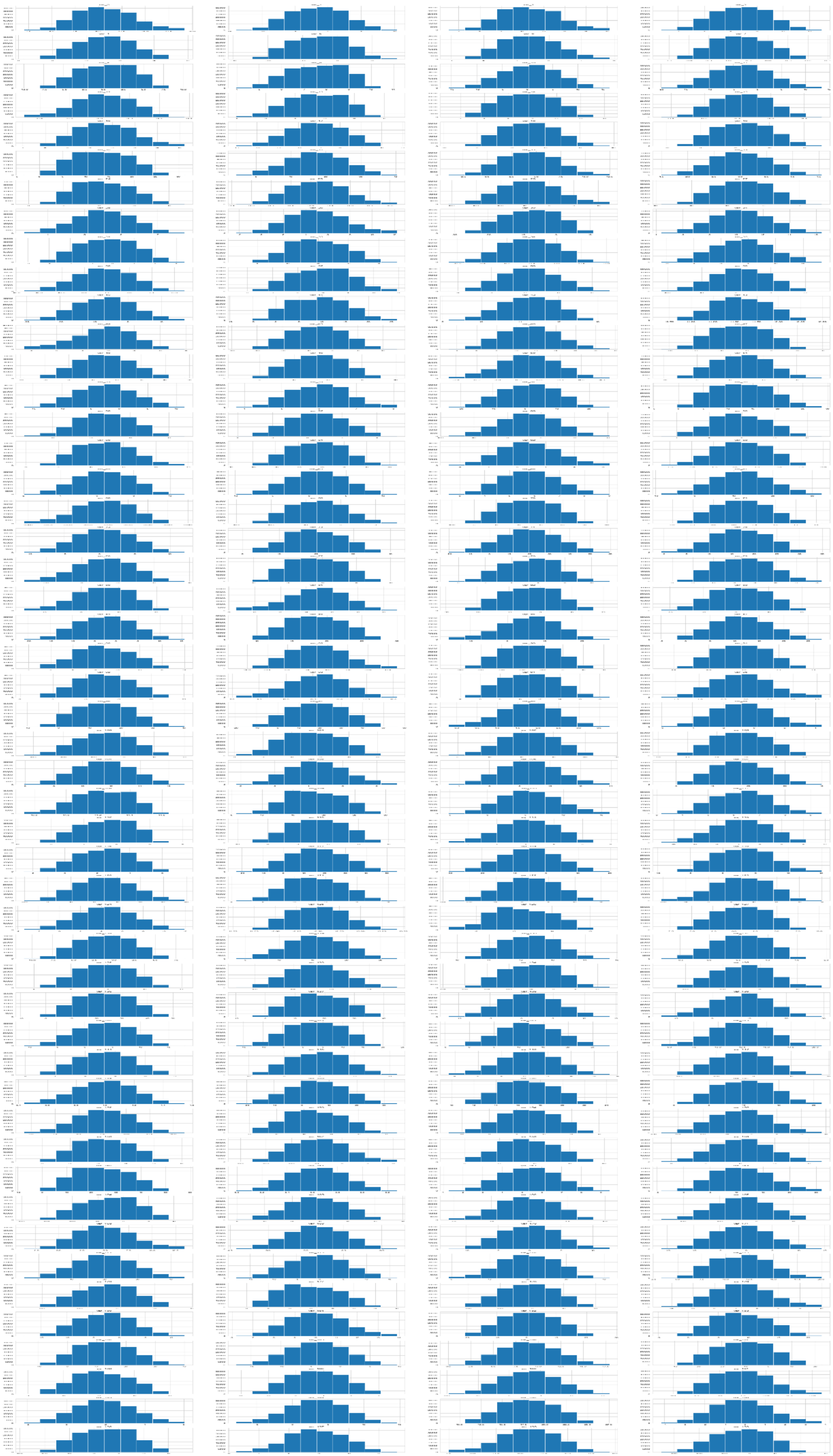
```
 (0.009824411895648928, -0.010286443734441413)
```

```
print("Distribution of columns per target class")
sns.set_style('whitegrid')
plt.figure(figsize=(40,200))
for i,col in enumerate(numerical_features):
    plt.subplot(50,4,i+1)
    sns.distplot(train[train['target']==0][col],hist=False,label='0',color=
'green')
    sns.distplot(train[train['target']==1][col],hist=False,label='1',color=
'red')
```

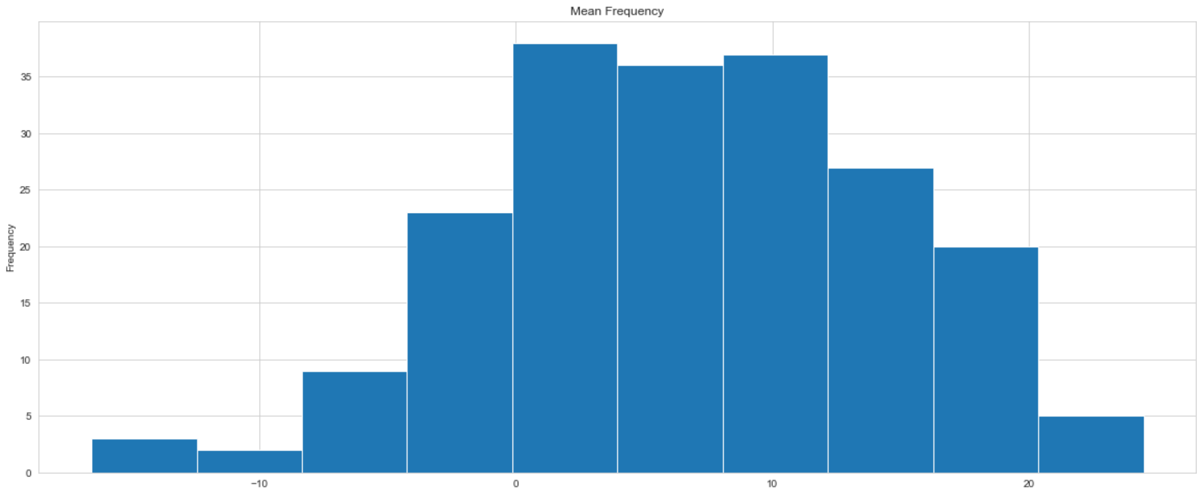
```
Distribution of columns per target class
```

```
#hisograms are used to check distribution of data
#draw histograms of numeric data in training set
print("Distribution of Columns")
plt.figure(figsize=(40,200))
for i,col in enumerate(numerical_features):
    plt.subplot(50,4,i+1)
    plt.hist(train[col])
    plt.title(col)
```

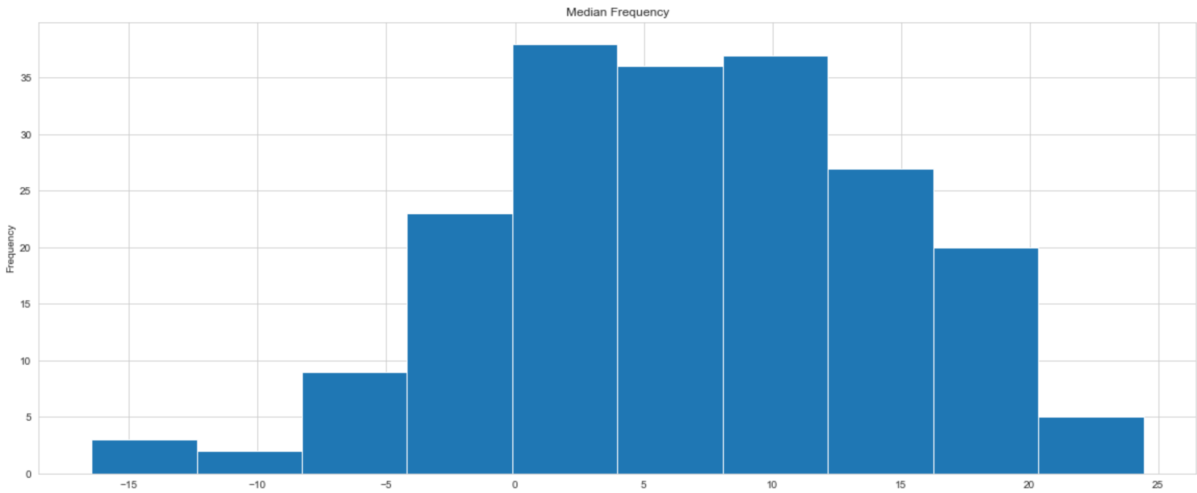
```
Distribution of Columns
```



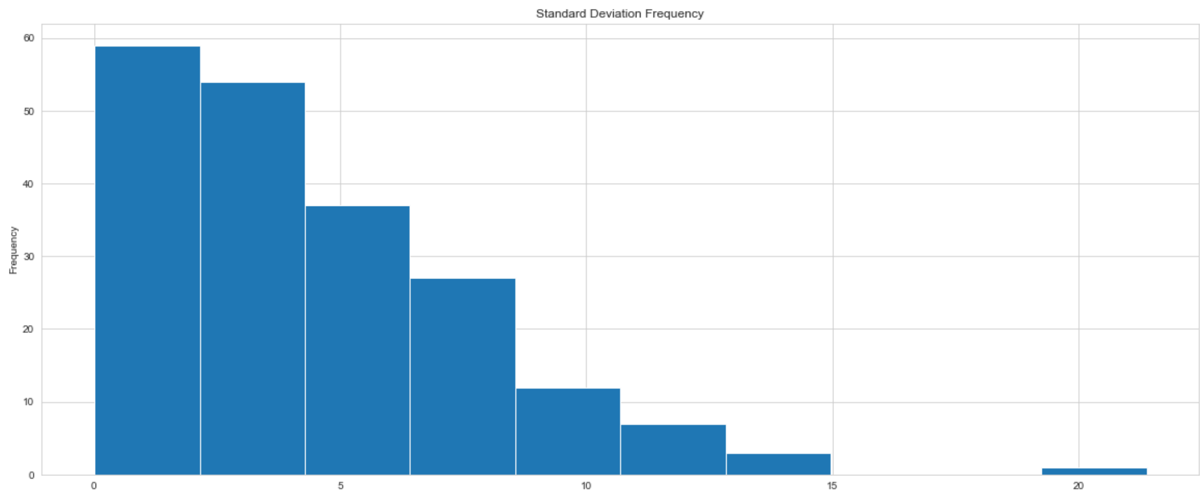
```
plt.figure(figsize=(20, 8))
train[numerical_features].mean().plot('hist');
plt.title('Mean Frequency');
```




```
plt.figure(figsize=(20, 8))
train[numerical_features].median().plot('hist');
plt.title('Median Frequency');
```

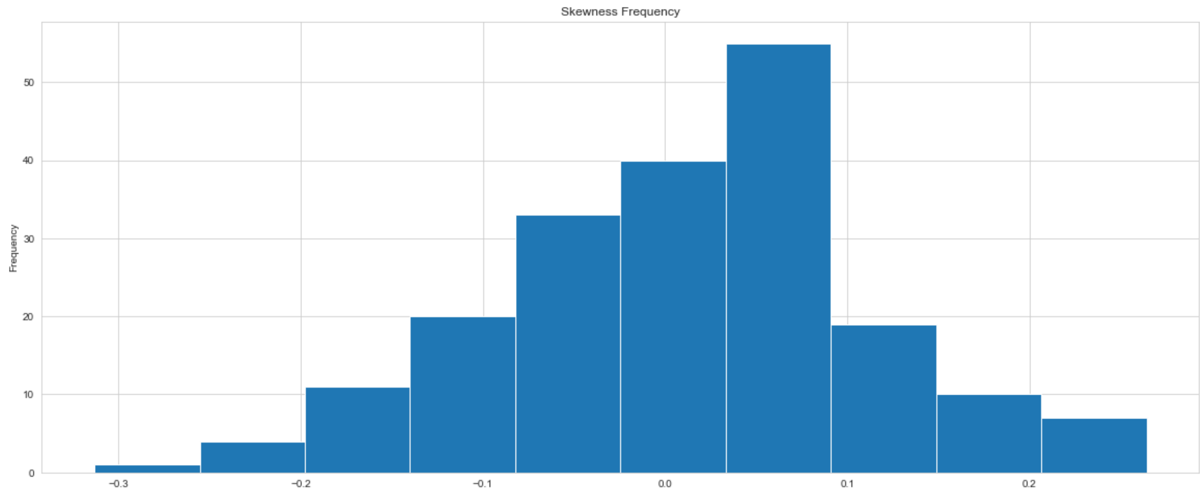



```
plt.figure(figsize=(20, 8))
train[numerical_features].std().plot('hist');
plt.title('Standard Deviation Frequency');
```




```
plt.figure(figsize=(20, 8))
train[numerical_features].skew().plot('hist');
plt.title('Skewness Frequency')
```

```
Text(0.5, 1.0, 'Skewness Frequency')
```




```
y = train['target']
x = train.drop(['target', "ID_code"], axis=1)
```

```
X_train, X_test, Y_train, Y_test =train_test_split(x, y, stratify = y, rand
om_state=42, test_size=0.3)
```

```
print(X_train.shape)
print(Y_train.shape)
print(X_test.shape)
print(Y_test.shape)
```

```
(122551, 200)
(122551,)
(52522, 200)
(52522,)
```

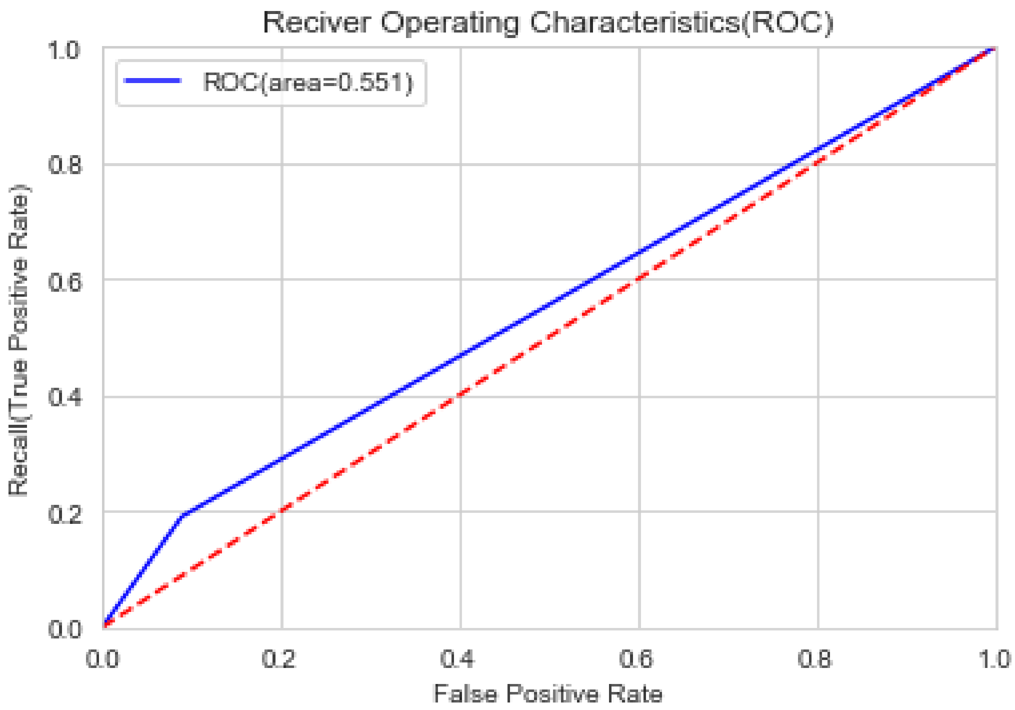
```
#Decision Tree
#Replace target categories with Yes or No
train['target']=train['target'].replace(0,'No')

#apply on train data
```

```
c50_model=tree.DecisionTreeClassifier(criterion='entropy').fit(X_train,Y_tr
ain)

#Apply on test data
Y_pred=c50_model.predict(X_test)
#create confusion matrix
CM=confusion_matrix(Y_test,Y_pred)
CM=pd.crosstab(Y_test,Y_pred)

#let us save TP,TN,FN,FP
TN=CM.iloc[0,0]
FP=CM.iloc[1,0]
TP=CM.iloc[1,1]
FN=CM.iloc[0,1]
print(CM)
```

```
col_0       0     1
target
0       43162  4229
1        4152   979
```

```
#ROC_AUC score
roc_score_dt = np.round(roc_auc_score(Y_test, Y_pred),2)
print('ROC score :',roc_score_dt)
#ROC_AUC curve
plt.figure()
false_positive_rate,recall,thresholds=roc_curve(Y_test,Y_pred)
roc_auc_dt=auc(false_positive_rate,recall)
plt.title('Reciver Operating Characteristics(ROC)')
plt.plot(false_positive_rate,recall,'b',label='ROC(area=%0.3f)' %roc_auc_dt
)
plt.legend()
plt.plot([0,1],[0,1],'r--')
plt.xlim([0.0,1.0])
plt.ylim([0.0,1.0])
plt.ylabel('Recall(True Positive Rate)')
plt.xlabel('False Positive Rate')
plt.show()
print('AUC:',roc_auc_dt)
```

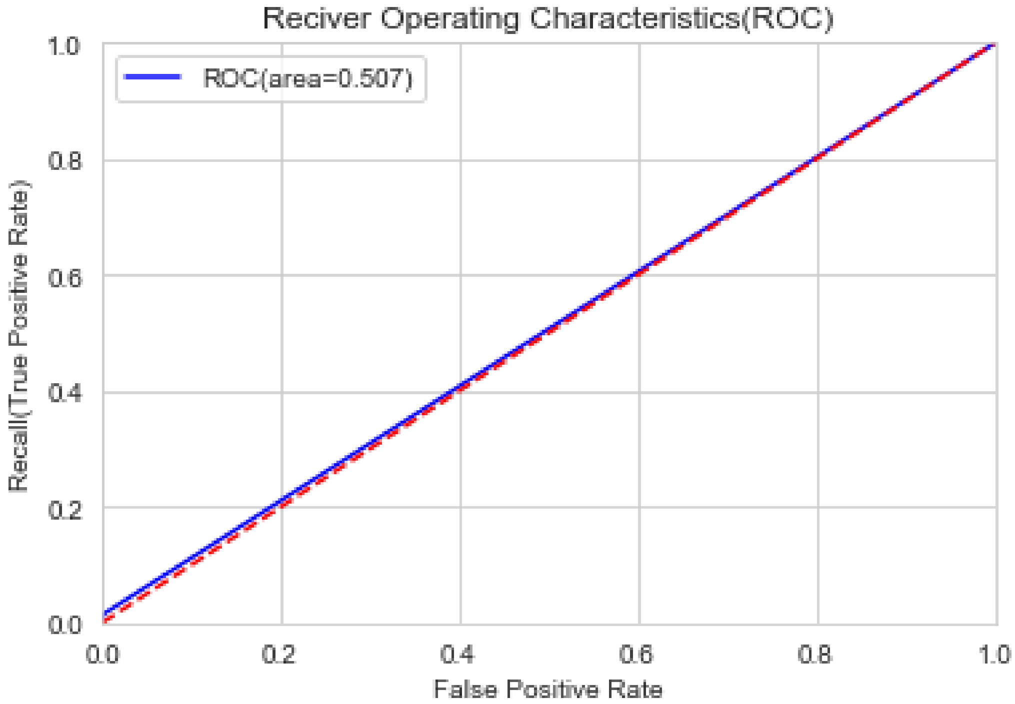
```
ROC score : 0.55
```




```
AUC: 0.5507823302768308
#Classification report
scores=classification_report(Y_test,Y_pred)
print(scores)
              precision    recall  f1-score   support

           0       0.91      0.91      0.91     47391
           1       0.19      0.19      0.19      5131

    accuracy                           0.84     52522
   macro avg       0.55      0.55      0.55     52522
weighted avg       0.84      0.84      0.84     52522

#Random Forest
RF_model=RandomForestClassifier(n_estimators=10).fit(X_train,Y_train)
RF_Pred=RF_model.predict(X_test)
#ROC_AUC score
roc_score_rf=roc_auc_score(Y_test,RF_Pred)
print('ROC score :',roc_score_rf)
#ROC_AUC curve
plt.figure()
false_positive_rate,recall,thresholds=roc_curve(Y_test,RF_Pred)
roc_auc_rf=auc(false_positive_rate,recall)
plt.title('Reciver Operating Characteristics(ROC)')
plt.plot(false_positive_rate,recall,'b',label='ROC(area=%0.3f)' %roc_auc_rf
)
plt.legend()
plt.plot([0,1],[0,1],'r--')
```

```
plt.xlim([0.0,1.0])
plt.ylim([0.0,1.0])
plt.ylabel('Recall(True Positive Rate)')
plt.xlabel('False Positive Rate')
plt.show()
print('AUC:',roc_auc_rf)
```

ROC score : 0.5067226716823265



AUC: 0.5067226716823265

```
#Classification report
scores=classification_report(Y_test,RF_Pred)
print(scores)
```

	precision	recall	f1-score	support
0	0.90	1.00	0.95	47391
1	0.51	0.02	0.03	5131
accuracy			0.90	52522
macro avg	0.71	0.51	0.49	52522
weighted avg	0.87	0.90	0.86	52522

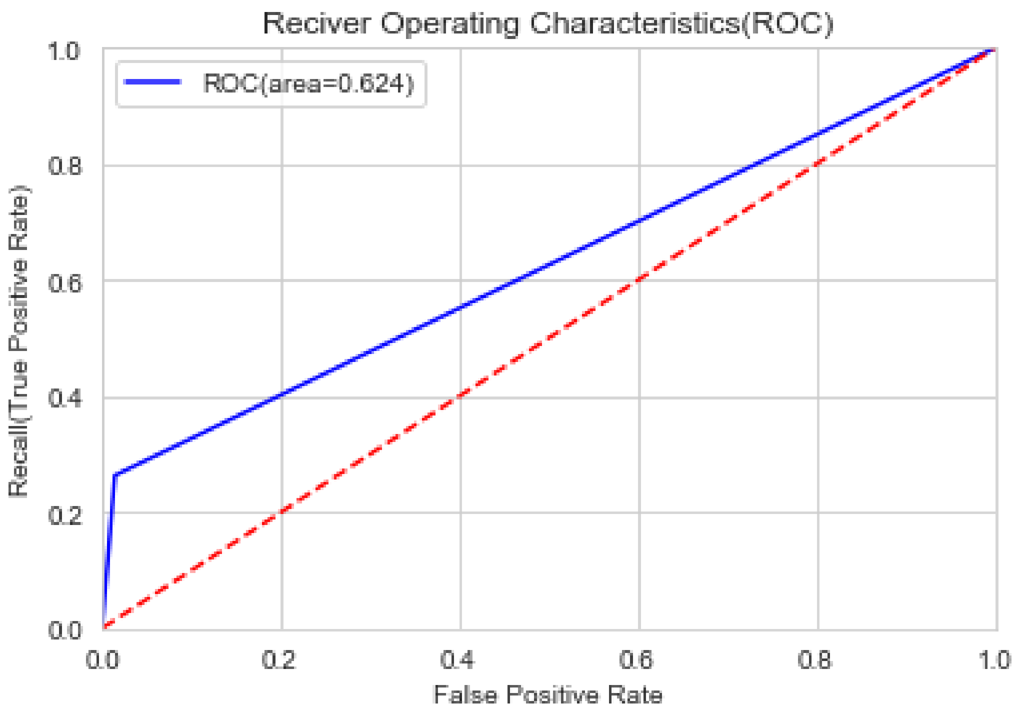
```
#Logistic Regression
lrg = LogisticRegression(random_state=42)
lrg.fit(X_train, Y_train)
y_pred_lrg = lrg.predict(X_test)

#Cross validation prediction
cv_predict=cross_val_predict(lrg,X_test,Y_test,cv=5)
#Cross validation score
```

```
cv_score=cross_val_score(lrg,X_test,Y_test,cv=5)
print('cross_val_score :',np.average(cv_score))
```

cross_val_score : 0.9150832208599764

```
roc_score_lrg=roc_auc_score(Y_test,cv_predict)
print('ROC score :',roc_score_lrg)
#ROC_AUC curve
plt.figure()
false_positive_rate,recall,thresholds=roc_curve(Y_test,cv_predict)
roc_auc_lrg=auc(false_positive_rate,recall)
plt.title('Reciver Operating Characteristics(ROC)')
plt.plot(false_positive_rate,recall,'b',label='ROC(area=%0.3f)' %roc_auc_lr
g)
plt.legend()
plt.plot([0,1],[0,1],'r--')
plt.xlim([0.0,1.0])
plt.ylim([0.0,1.0])
plt.ylabel('Recall(True Positive Rate)')
plt.xlabel('False Positive Rate')
plt.show()
print('AUC:',roc_auc_lrg)
```

ROC score : 0.6244763923405998



AUC: 0.6244763923405998

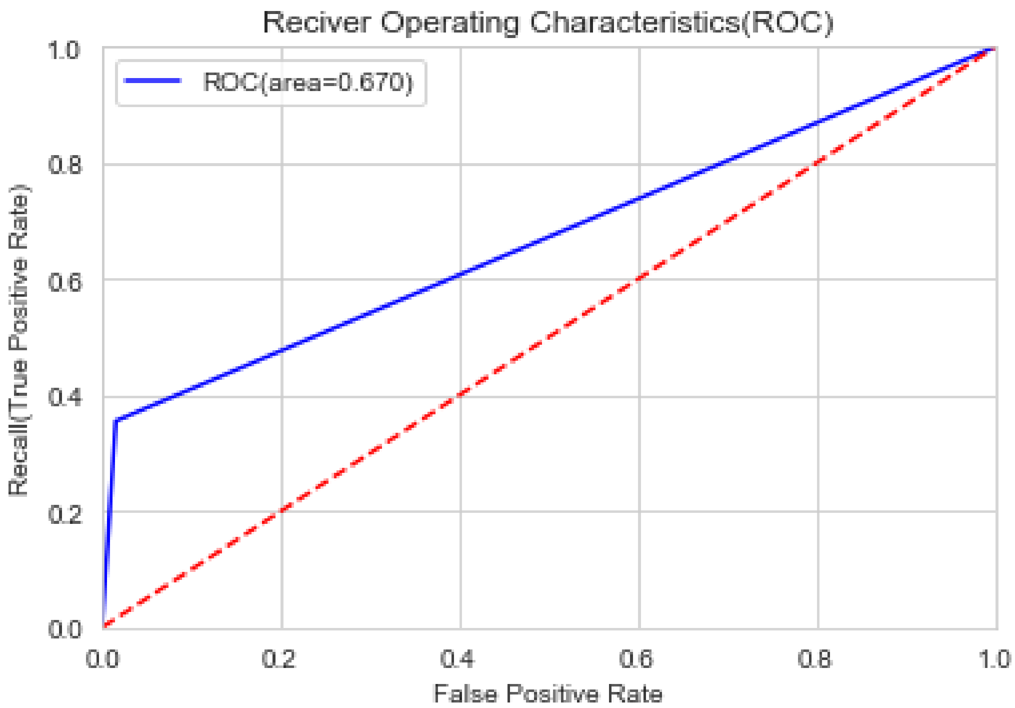
```
#Classification report
scores=classification_report(Y_test,y_pred_lrg)
print(scores)
```

```
              precision    recall  f1-score   support
```

```
           0       0.93      0.99      0.96     47391
           1       0.69      0.26      0.38      5131

    accuracy                           0.92     52522
   macro avg       0.81      0.63      0.67     52522
weighted avg       0.90      0.92      0.90     52522
```

```
#Naive Bayes
gnb = GaussianNB()
gnb.fit(X_train, Y_train)
y_pred_gnb = gnb.predict(X_test)

#ROC_AUC score
roc_score_gnb=roc_auc_score(Y_test,y_pred_gnb)
print('ROC score :',roc_score_gnb)

#ROC_AUC curve
plt.figure()
false_positive_rate,recall,thresholds=roc_curve(Y_test,y_pred_gnb)
roc_auc_gnb=auc(false_positive_rate,recall)
plt.title('Reciver Operating Characteristics(ROC)')
plt.plot(false_positive_rate,recall,'b',label='ROC(area=%0.3f)' %roc_auc_gn
b)
plt.legend()
plt.plot([0,1],[0,1],'r--')
plt.xlim([0.0,1.0])
plt.ylim([0.0,1.0])
plt.ylabel('Recall(True Positive Rate)')
plt.xlabel('False Positive Rate')
plt.show()
print('AUC:',roc_auc_gnb)
```

```
ROC score : 0.6700517961965967
```




```
AUC: 0.6700517961965967
#Classification report
scores=classification_report(Y_test,y_pred_gnb)
print(scores)
              precision    recall  f1-score   support

           0       0.93      0.98      0.96     47391
           1       0.71      0.36      0.47      5131


    accuracy                           0.92     52522
   macro avg       0.82      0.67      0.72     52522
weighted avg       0.91      0.92      0.91     52522
#SMOTE
#Synthetic Minority Oversampling Technique
sm = SMOTE(random_state=42, ratio=1.0)
#Generating synthetic data points
X_smote,y_smote=sm.fit_sample(X_train,Y_train)
X_smote_v,y_smote_v=sm.fit_sample(X_test,Y_test)

#Logistic regression model for SMOTE
smote=LogisticRegression(random_state=42)
#fitting the smote model
smote.fit(X_smote,y_smote)

LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True
,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='warn', n_jobs=None, penalty='l2',
                   random_state=42, solver='warn', tol=0.0001, verbose=0,
                   warm_start=False)
```

```
#Accuracy of the model
smote_score=smote.score(X_smote,y_smote)
print('Accuracy of the smote_model :',smote_score)
```

```
Accuracy of the smote_model : 0.7985422186852839
```

```
#Cross validation prediction
cv_pred=cross_val_predict(smote,X_smote_v,y_smote_v,cv=5)
#Cross validation score
cv_score=cross_val_score(smote,X_smote_v,y_smote_v,cv=5)
print('cross_val_score :',np.average(cv_score))
```

```
cross_val_score : 0.7970819415096922
```

```
#Confusion matrix
cm=confusion_matrix(y_smote_v,cv_pred)
cm=pd.crosstab(y_smote_v,cv_pred)
```

```
cm
```

col_0	0	1
row_0		
0	37392	9999
1	9234	38157

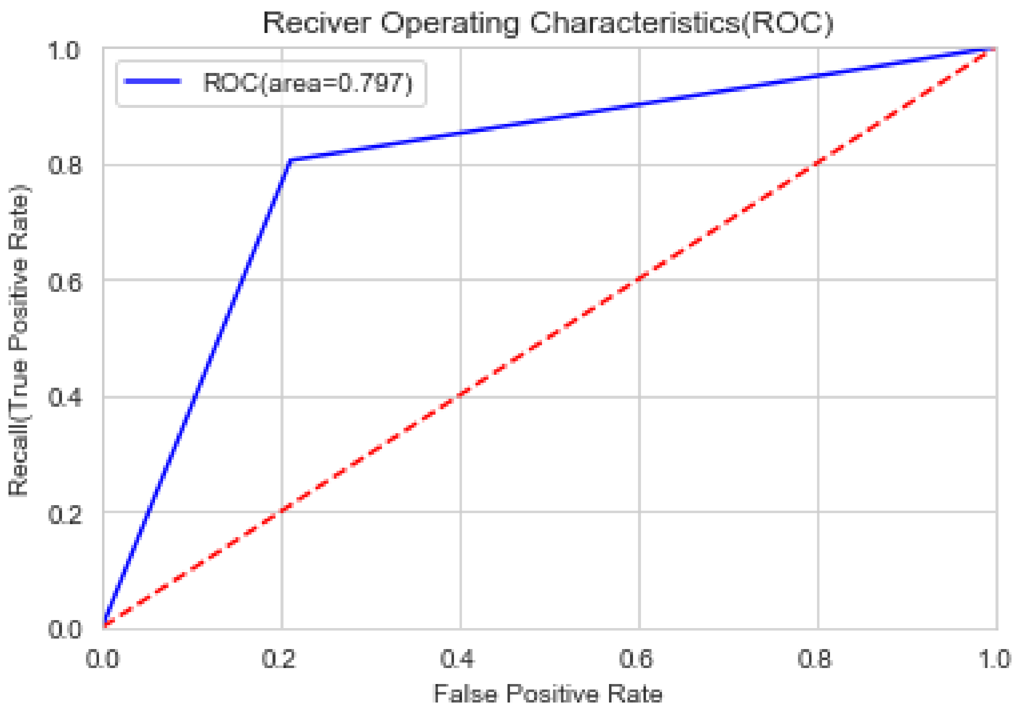
```
#Classification report
scores=classification_report(y_smote_v,cv_pred)
print(scores)
```

```
              precision    recall  f1-score   support

           0       0.80      0.79      0.80     47391
           1       0.79      0.81      0.80     47391

    accuracy                           0.80     94782
   macro avg       0.80      0.80      0.80     94782
weighted avg       0.80      0.80      0.80     94782
```

```
#ROC_AUC score
roc_score=roc_auc_score(y_smote_v,cv_pred)
print('ROC score :',roc_score)

#ROC_AUC curve
plt.figure()
false_positive_rate,recall,thresholds=roc_curve(y_smote_v,cv_pred)
roc_auc=auc(false_positive_rate,recall)
plt.title('Reciver Operating Characteristics(ROC)')
```

```
plt.plot(false_positive_rate,recall,'b',label='ROC(area=%0.3f)' %roc_auc)
plt.legend()
plt.plot([0,1],[0,1],'r--')
plt.xlim([0.0,1.0])
plt.ylim([0.0,1.0])
plt.ylabel('Recall(True Positive Rate)')
plt.xlabel('False Positive Rate')
plt.show()
print('AUC:',roc_auc)
```

```
ROC score : 0.7970817243780465
```




```
AUC: 0.7970817243780465
```

```
#Predicting the model
X_test=test.drop(['ID_code'],axis=1)
smote_pred=smote.predict(X_test)
print(smote_pred)

print('\nWe can observe that smote model is performing well on imbalance da
ta compare to Random Forest,logistic regression or any  other models')
```

```
[1 1 0 ... 0 0 1]

We can observe that smote model is performing well on imbalance data compar
e to Random Forest,logistic regression or any  other models
```

```
#final submission
sub_df=pd.DataFrame({'ID_code':test['ID_code'].values})
sub_df['smote_pred']=smote_pred
sub_df.to_csv('submission.csv',index=False)
sub_df.head()
```

	ID_code	smote_pred
0	test_0	1
1	test_1	1
2	test_2	0
3	test_3	1
4	test_4	0

Python code:

file:///C:/Users/chandini%20c/Downloads/Project%201(Santander%20Customer%20Transaction).html

Output test data with target variable:

submission.csv

Appendix B -R Code

```
#import libraries and set working directory

rm(list=ls())

setwd("C:/Users/chandini c/Desktop")

getwd()

x = c("ggplot2","glmnet","pROC", "corrgram", "DMwR", "caret", "randomForest", "unbalanced",
"C50", "dummies", "e1071", "Information",

    "MASS", "rpart", "gbm", "ROSE", 'sampling', 'DataCombine', 'inTrees')


#Load Libraries

lapply(x, require, character.only = TRUE)

rm(x)


#Load Data Sets

train = read.csv("train.csv")

test=read.csv("test.csv")

head(train)


#Dimension of data sets

dim(train)

dim(test)


#Summary of the dataset

str(train)

str(test)


#convert to factor
```

```
train$target=as.factor(train$target)

require(gridExtra)
#Count of target classes
table(train$target)
#Percenatge counts of target classes
table(train$target)/length(train$target)*100
#Bar plot for count of target classes
plot1=ggplot(train,aes(target))+theme_bw()+geom_bar(stat='count',fill='lightgreen')
grid.arrange(plot1)

###MISSING VALUE ANALYSIS
#Finding the missing values in train data
missing_val=data.frame(missing_val=apply(train,2,function(x){sum(is.na(x))}))
missing_val=sum(missing_val)
missing_val

#Finding the missing values in test data
missing_val=data.frame(missing_val=apply(test,2,function(x){sum(is.na(x))}))
missing_val=sum(missing_val)
missing_val


##OUTLIER ANALYSIS
#Outlier Analysis in train dataset
numeric_index=sapply(train,is.numeric)
numeric_data=train[,numeric_index]
numerical_features=colnames(numeric_data)

#Removal of outliers in train data:
for(i in numerical_features){
```

```r
  val=train[,i][train[,i]%in%boxplot.stats(train[,i])$out]
  train=train[which(!train[,i]%in%val),]
}

#Outlier Analysis in test dataset
numeric_index=sapply(test,is.numeric)
numeric_data=test[,numeric_index]
numerical_features=colnames(numeric_data)

#Removal of outliers in test data:
for(i in numerical_features){
  val=test[,i][test[,i]%in%boxplot.stats(test[,i])$out]
  test=test[which(!test[,i]%in%val),]
}



##CORRELATION ANALYSIS
#Correlations in train data
#convert factor to int
train$target=as.numeric(train$target)
train_correlations=cor(train[,c(2:202)])
train_correlations

#Correlations in test data
test_correlations=cor(test[,c(2:201)])
test_correlations



##VISUALIZATIONS
#Distribution of train attributes
for (var in names(train)[c(3:202)]){
```

```
  target=train$target
  plot=ggplot(train, aes(x=train[[var]],fill=target)) +
    geom_density(kernel='gaussian') + ggtitle(var)+theme_classic()
  print(plot)
}
#Distribution of test attributes
for (var in names(test)[c(2:201)]){
  target=test$target
  plot=ggplot(test, aes(x=test[[var]])) +
    geom_density(kernel='gaussian') + ggtitle(var)+theme_classic()
  print(plot)
}


#Applying the function to find mean values per column in train and test data.
train_mean=apply(train[,-c(1,2)],MARGIN=2,FUN=mean)
test_mean=apply(test[,-c(1)],MARGIN=2,FUN=mean)
ggplot()+

#Distribution of mean values per column in train data
geom_density(aes(x=train_mean),kernel='gaussian',show.legend=TRUE,color='blue')+theme_classic(
)+

#Distribution of mean values per column in test data
geom_density(aes(x=test_mean),kernel='gaussian',show.legend=TRUE,color='green')+
labs(x='mean values per column',title="Distribution of mean values per row in train and test
dataset")


#Applying the function to find sd values per column in train and test data.
train_sd=apply(train[,-c(1,2)],MARGIN=2,FUN=sd)
test_sd=apply(test[,-c(1)],MARGIN=2,FUN=sd)
```

```
ggplot()+

#Distribution of sd values per column in train data
geom_density(aes(x=train_sd),kernel='gaussian',show.legend=TRUE,color='red')+theme_classic()+
  #Distribution of sd values per column in test data
geom_density(aes(x=test_sd),kernel='gaussian',show.legend=TRUE,color='blue')+
labs(x='sd values per column',title="Distribution of std values per column in train and test dataset")



#Applying the function to find skewness values per column in train and test data.
train_skew=apply(train[,-c(1,2)],MARGIN=2,FUN=skewness)
test_skew=apply(test[,-c(1)],MARGIN=2,FUN=skewness)
ggplot()+

#Distribution of skewness values per column in train data

geom_density(aes(x=train_skew),kernel='gaussian',show.legend=TRUE,color='green')+theme_classic
()+

#Distribution of skewness values per column in test data
  geom_density(aes(x=test_skew),kernel='gaussian',show.legend=TRUE,color='blue')+
  labs(x='skewness values per column',title="Distribution of skewness values per column in train and
test dataset")

##MODELING##
#splitting the data using stratified sampling method:
rm(list=c('numeric_data','test_correlations','train_correlations','i','numeric_index','numerical_featur
es','val'))
set.seed(1234)

#convert to factor
train$target=as.factor(train$target)
```

```
train_index=createDataPartition(train$target,p=.70,list=FALSE)
train_data=train[train_index,]

#validation data
valid_data=train[-train_index,]

#dimension of train and validation data
dim(train_data)
dim(valid_data)

##RANDOM FOREST:
#convert to int to factor
train_data$target=as.factor(train_data$target)

#Random Forest Model on train data
RF_model=randomForest(target~.,train_data[,-c(1)],importance=TRUE,ntree=10)

#Model performance on validation data set
RF_pred=predict(RF_model,valid_data[,-2])

#confusion matrix
confmatrix_RF=table(valid_data$target,RF_pred)
confusionMatrix(confmatrix_RF)

#Accuracy=90.22
#FNR=(FN*100)/(FN+TP)=50.86
(147*100)/(147+142)
#Recall=49.13
(142*100)/(142+147)

#ROC_AUC score and curve
```

```
set.seed(843)
#convert to numeric
RF_pred=as.numeric(RF_pred)
roc(valid_data$target,predictor=RF_pred,auc=TRUE,plot=TRUE)
#AUC=51.23

#LOGISTIC REGRESSION:
#Training dataset
X_t=as.matrix(train_data[,-c(1,2)])
y_t=as.matrix(train_data$target)

#validation dataset
X_v=as.matrix(valid_data[,-c(1,2)])
y_v=as.matrix(valid_data$target)

#test dataset
test_df=as.matrix(test[,-c(1)])

#logistic regression model
library(glmnet)
set.seed(667) # to reproduce results
lr_model=glmnet(X_t,y_t, family = "binomial")
summary(lr_model)

#cross validation:
cv_lr=cv.glmnet(X_t,y_t,family = "binomial", type.measure = "class")

#Model performance on validation dataset
set.seed(5363)
cv_predict=predict(cv_lr,X_v,type = "class")
```

```
#Confusion matrix
set.seed(689)

#actual target variable
target=valid_data$target

#convert to factor
target=as.factor(target)

#predicted target variable
#convert to factor
cv_predict=as.factor(cv_predict)
confMat=table(valid_data$target,cv_predict)
confusionMatrix(confMat)

#Accuracy=91.56%
#Recall=74.87
#FNR=(FN*100)/(FN+TP)=25.12
(353*100)/(353+1052)
#Recall=(TP*100)/(TP+FN)=74.87
(1052*100)/(1052+353)
#ROC_AUC score and curve
set.seed(843)
#convert to numeric
cv_predict=as.numeric(cv_predict)
roc(valid_data$target,predictor=cv_predict,auc=TRUE,plot=TRUE)
#AUC=59.55

#Naive bayes
library(e1071)
#Develop model
```

```
NB_model=naiveBayes(target~.,data=train_data)
#Predict on test cases
NB_pred=predict(NB_model,X_v,type="class")
#Confusion matrix
confmatrix_NB=table(observed=y_v,predicted=NB_pred)
confusionMatrix(confmatrix_NB)

#Accuracy=92.33%
#FNR=24.5
#Recall=71.77
(713*100)/(713+1813)
(1813*100)/(713+1813)

#ROC_AUC score and curve
set.seed(843)
#convert to numeric
NB_pred=as.numeric(NB_pred)
roc(valid_data$target,predictor=NB_pred,auc=TRUE,plot=TRUE)
#AUC=66.92

#Random Oversampling Examples(ROSE) for Logistic Reg
library(ROSE)
set.seed(699)
train.rose = ROSE(target~., data =train_data[,-c(1)],seed=32)$data

#target classes in balanced train data
table(train.rose$target)
valid.rose = ROSE(target~., data =valid_data[,-c(1)],seed=42)$data

#target classes in balanced valid data
table(valid.rose$target)
```

```
#Logistic regression model
set.seed(462)
lr_rose =glmnet(data.matrix(train.rose),data.matrix(train.rose$target), family = "binomial")
summary(lr_rose)

#Cross validation prediction
set.seed(473)
cv_rose = cv.glmnet(data.matrix(valid.rose),data.matrix(valid.rose$target),family = "binomial",
type.measure = "class")
cv_rose

#Model performance on validation dataset
set.seed(442)
predict.rose=predict(cv_rose,data.matrix(valid.rose),s = "lambda.min", type = "class")
predict.rose

#Confusion matrix
set.seed(478)
#actual target variable
target=valid.rose$target
#convert to factor
target=as.factor(target)
#predicted target variable
#convert to factor
predict.rose=as.factor(predict.rose)
#Confusion matrix
confmat_rose=table(valid.rose$target,predict.rose)
confusionMatrix(confmat_rose)
#accuracy=100%
```

#ROC_AUC score and curve

set.seed(843)

#convert to numeric

predict.rose=as.numeric(predict.rose)

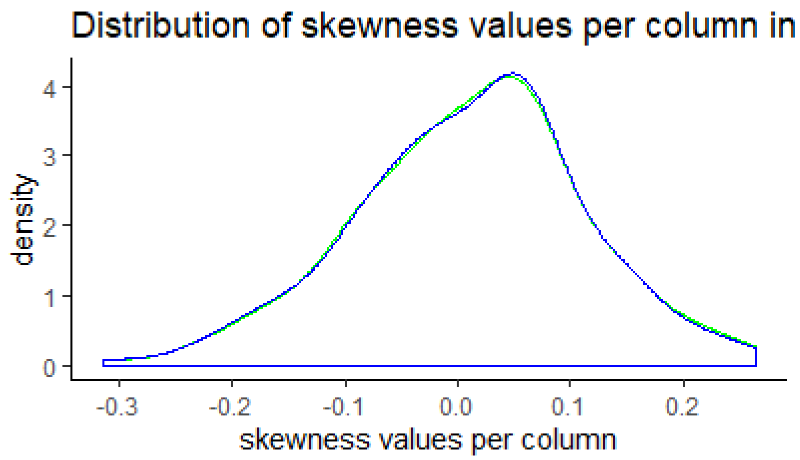
roc(valid.rose$target,predictor=predict.rose,auc=TRUE,plot=TRUE)

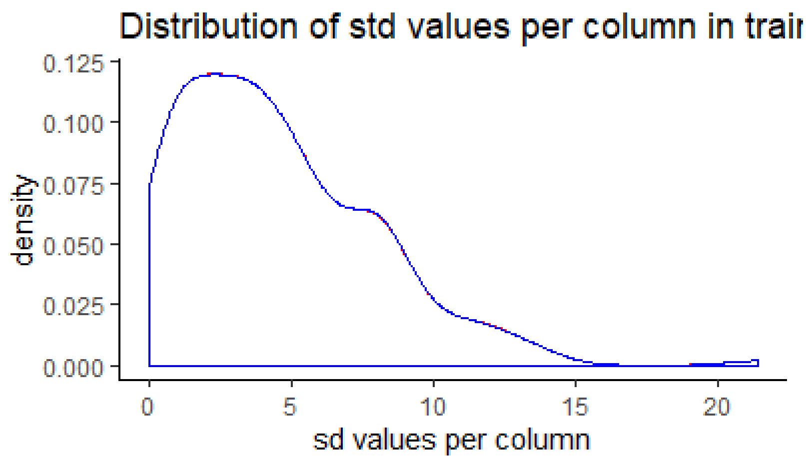
#AUC=100

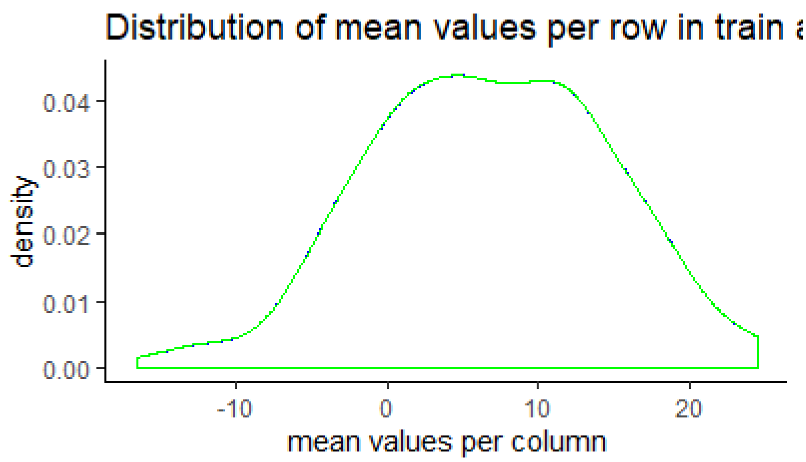
R Code File:

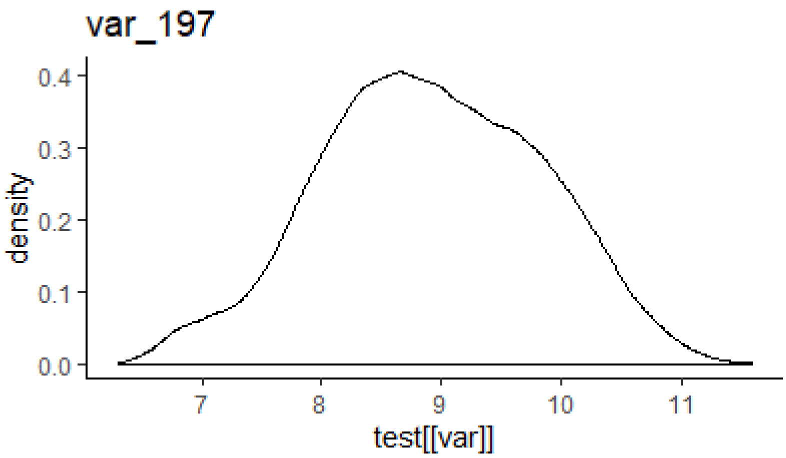
santander_project.R

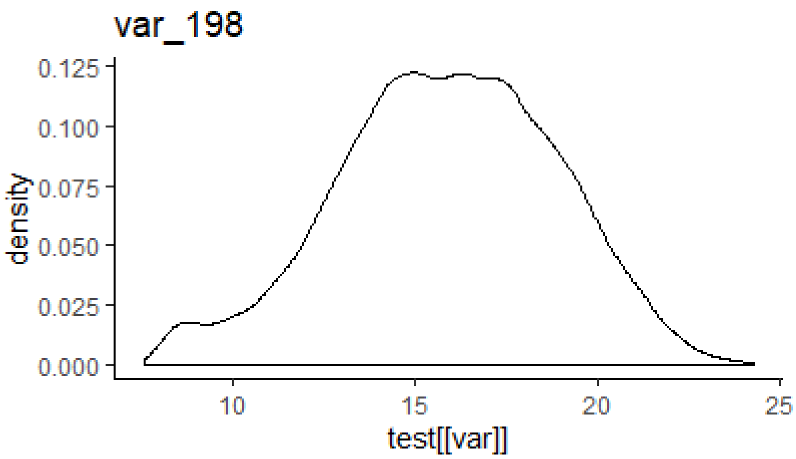
Appendix B -Extra Figures in R CODE

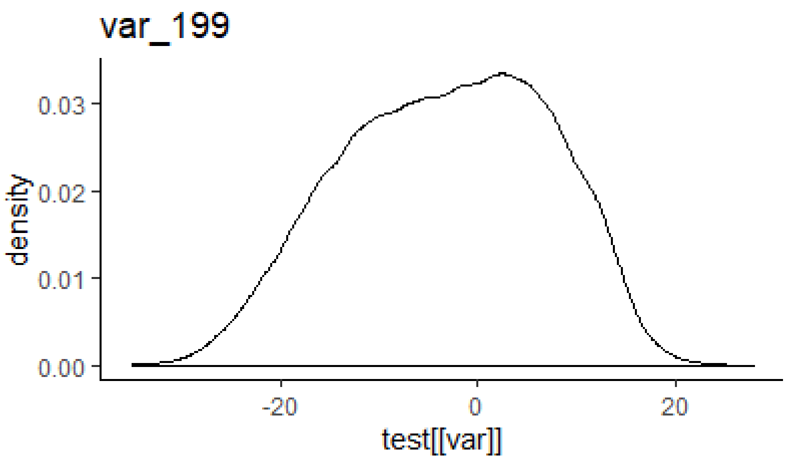


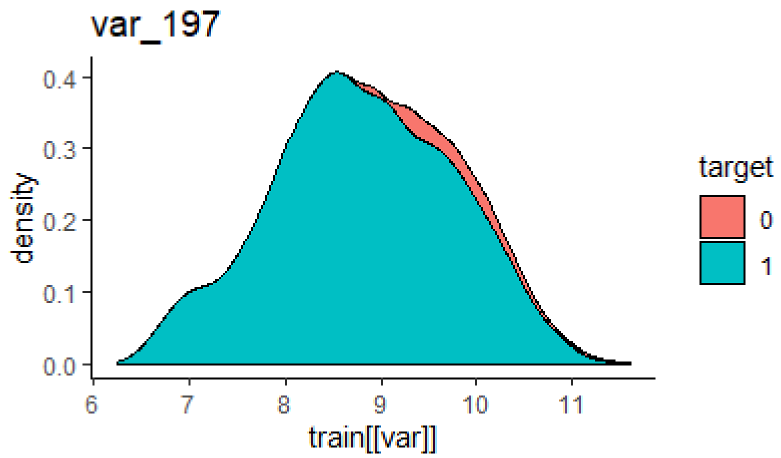


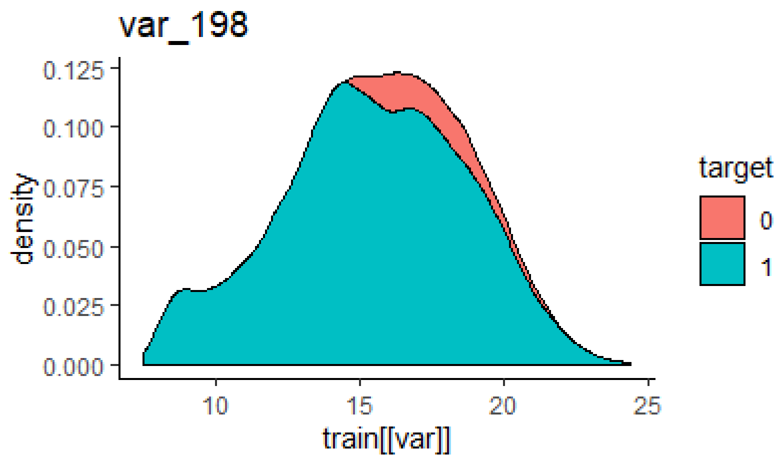


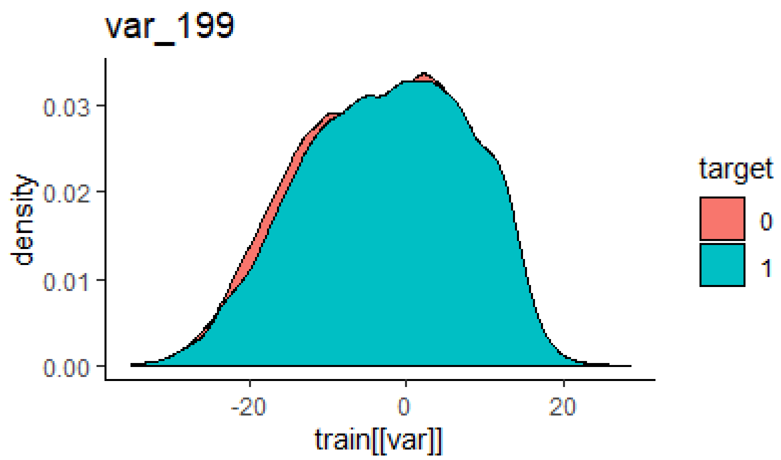

var_197
density
0.4
0.3
0.2
0.1
0.0
7
8
9
10
11
test[[var]]


var_198
density
0.125
0.100
0.075
0.050
0.025
0.000
10
15
20
25
test[[var]]


var_199
density
0.03
0.02
0.01
0.00
-20
0
20
test[[var]]


var_197
density
train[[var]]
target
0
1


var_198
density
train[[var]]
target
0
1


var_199
density
train[[var]]
target
0
1

References

https://hackernoon.com/what-steps-should-one-take-while-doing-data-preprocessing-502c993e1caa

https://www.geeksforgeeks.org/data-preprocessing-in-data-mining/

https://developer.ibm.com/articles/data-preprocessing-in-detail/

https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5