Credit Card Fraud Anomaly Detection Project

Alan Danque
Spring Semester
GitHub Portfolio:
https://adanque.github.io/
https://github.com/adanque/Anomaly-Detection

Purpose:
To analyze factors that identify fraudulent credit card transactions.

Abstract:
Payments using credit cards is one of the most convenient ways to pay for products or services. There are many types of monetary transactions that can be completed easily using credit cards. With a simple swipe of a magnetic strip. Insert of a digital chip. Briefly passing a wireless RFID scanner. Voicing one's credit card numbers over the telephone. Saving the credentials on a browser. A credit card customer can purchase anything from any vendor in person or online. From a small pack of gum from a gas station to airline tickets at the airport. To buying electronic goods from a nearby Target store. With all these convenient ways to pay - comes the opportunity for one's credit card information to be stolen. And then used to fraudulently to buy items they would normally not buy. According to the author Roman Chuprina, "Unauthorized card operations hit an astonishing amount of 16.7 million victims in 2017. Additionally, as reported by the Federal Trade Commission (FTC), the number of credit card fraud claims in 2017 was 40% higher than the previous year's number. There were around 13,000 reported cases in California and 8,000 in Florida, which are the largest states per capita for such type of crime. The amount of money at stake will exceed approximately $30 billion by 2020." (Chuprina, 2021) That is where anomaly detection for credit card fraud can come in. By analyzing the deviation between what is normal and expected behavior, it is possible to identify fraudulent purchases. (Vemula, 2020)

Questions:
The goals of my project are to answer the following questions.

- Since the credit card data may likely be streaming in real time, will it be possible to detect credit card fraud?
    - Answer: Yes, given that this project was able to use one period of data it is still able to predict.
- What visualizations can be used to help identify credit card fraud?
    - Answer: The visualizations that I found are helpful with identifying credit card fraud are boxplots and scatterplots.
- What are the factors that can lead to credit card fraud?
    - Answer: Since my variables were PCA translated by the provider of the dataset, I was able to use the resulting data frame export from the correlation matrix to find that my variables V4, V11, V2 and V21 are factors that lead to credit card fraud.
- Which algorithms can be used to detect credit card fraud?
    - Answer: After having tested many algorithms, my tests found that ExtraTreesClassifier was the best algorithm for my model.
- How many variables can be used to detect credit card fraud?

- Answer: Using the elbow method with a PCA Cumulative Explained Variance plot, I found that 4 components explain 90% of the variance in my dataset.
- Is it possible to accidentally mistaken a fraudulent credit card charge for a real charge?
    - Answer: Although my test measures per accuracy, recall & precision therefore F1 score are respectively a little higher than 99%, between 72-88% and about 85% there is a low possibility of the model accidentally identifying a normal charge with a fraudulent charge.
- Are there any variables with multicollinearity?
    - Answer: Yes, as can see in the correlation matrix there appears to be multicollinearity with variables V1 & V2, V6 & V7 and V8, V11 & V12, V21 & V22, V27 & V28.
- Can we still create a model if our dataset contains masked variables?
    - Answer: Absolutely, in this project I was able to make accurate predictions - even though the labels for most of the variables were masked. However, the was able to do so since the target response variable was available.
- How can we measure the accuracy of our detection models?
    - Answer: In this project, I was able to use a variety of measures. These include, MAE, MSE, RMSE, r2 Score, F1 Score and perform pipeline validations tests of predicted values.
- How accurate will the detection of credit card fraud be?
    - Answer: It can be as accurate as higher than 99%.

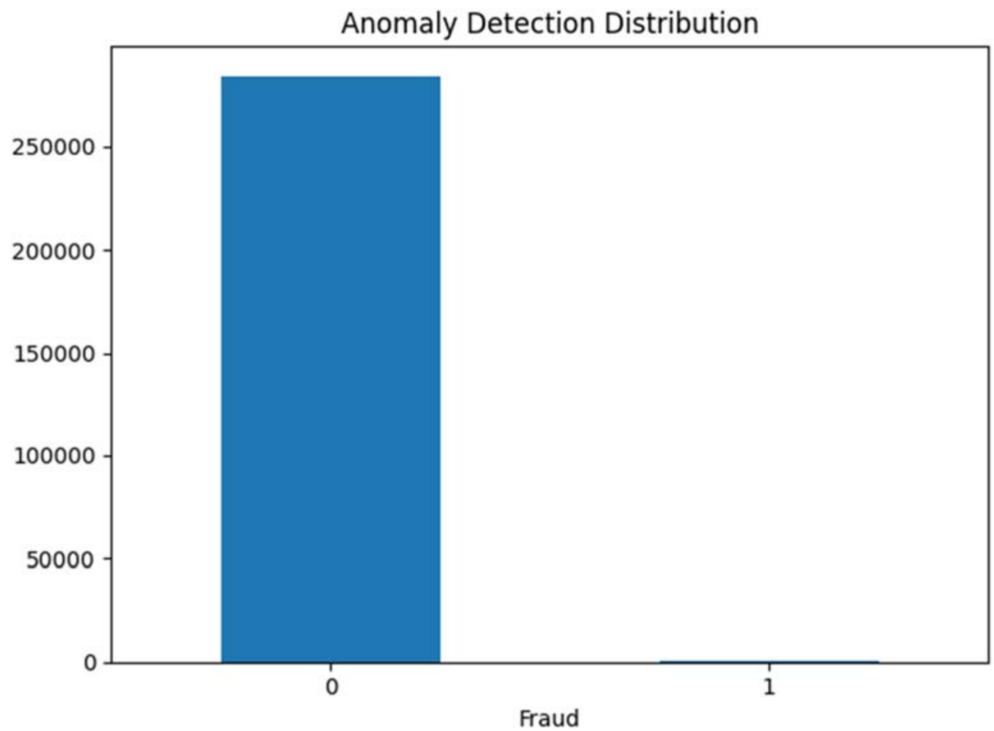
Methods:

1. I used the Pandas profiling library to assist with generating graphs for exploring the distribution of my data and identify possible fields that need cleaning or removal.
2. I then generated plots to visualize the distribution of my data, PCA & Inertia plots to understand the grouping, correlation matrix to review relationships of my dataset's fields, boxplots to analyze the outliers in my dataset and scatter plots to review the distributed spread of my normal and fraudulent classes.
3. Split my dataset using the sklearn RepeatedStratifiedKFold for model training.
4. Use the algorithms: Local outlier factor, One Class SVM, Isolation Forest and Elliptic Envelop to attempt to make predictions. Then to further increase specificity over sensitivity, I used pyCaret to help identify the best performing model.
5. Review and measure the performance of my predictive model using a Precision Recall, F1 Score, MCC, Kapp, r2 score, MAE, MSE, RMSE and the time to train.
6. Test and verify predictions of my model using test data.
7. Visualize the Decision Tree of my resulting model using scikit learn's export_graphviz for model explain ability.

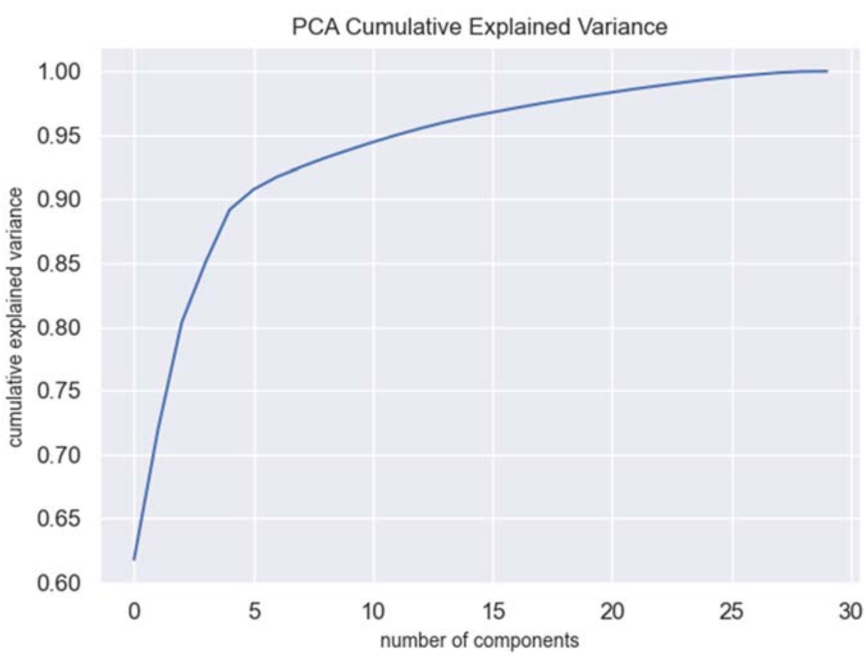
Project Dataset:

Type**:**	CSV
Columns:	31
Rows:	284,807

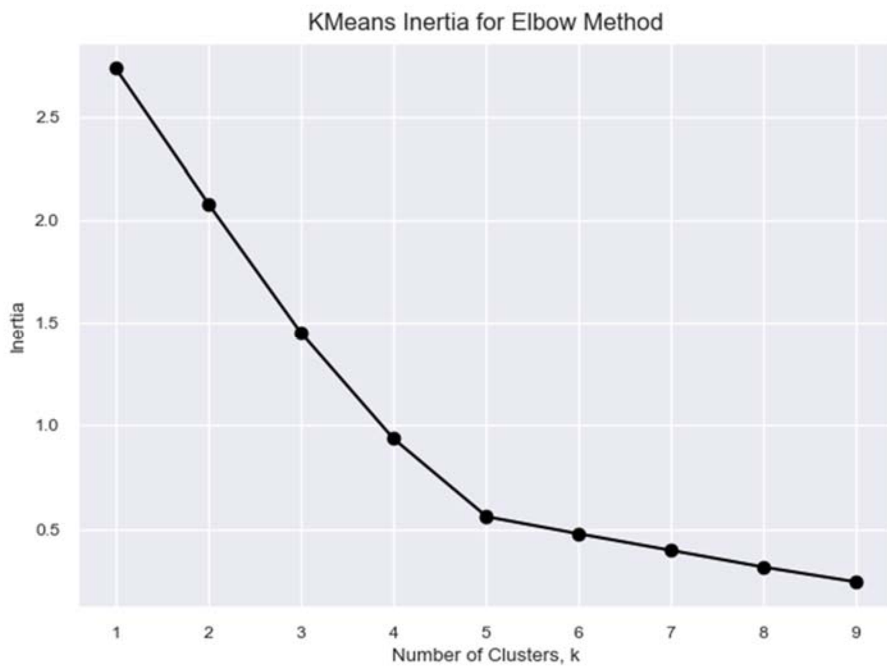
Plots:



My project dataset is an imbalanced dataset with 492 fraud records out of 284,807 transactions.

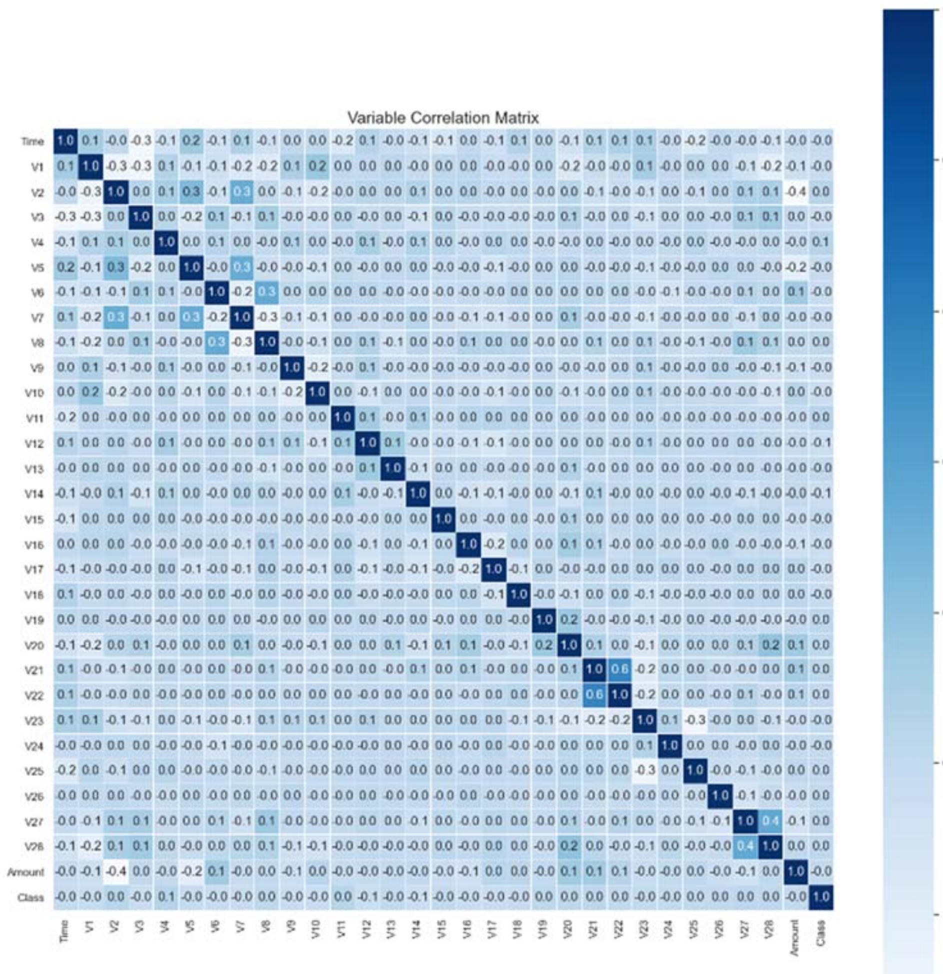


The above indicates that 4-5 components can explain 90% of data variances.



The k means inertia elbow graph above indicates that my dataset can be optimally clustered into 4-5 groups.

Correlation Matrices



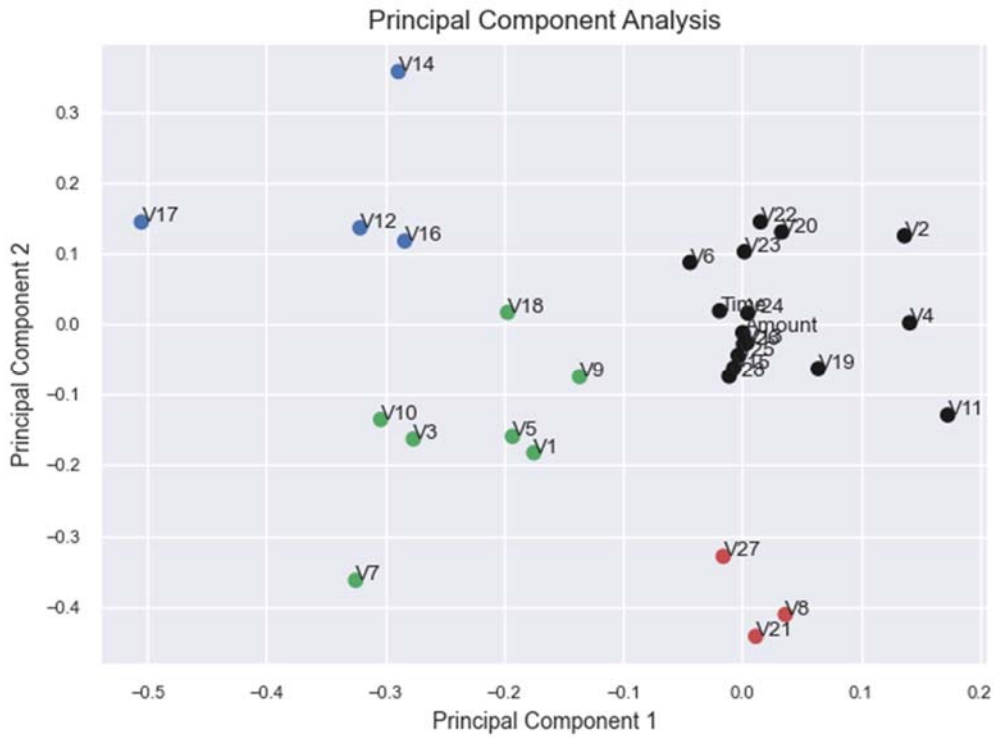
I found that if I export the correlation matrix to csv and then sort the feature correlation values by Class.

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	V13
Time	1	0.1278	-0.004	-0.317	-0.095	0.1504	-0.074	0.078	-0.069	0.0117	0.0096	-0.152	0.0504	-0.03
V1	0.1278	1	-0.279	-0.305	0.0614	-0.07	-0.062	-0.156	-0.174	0.0948	0.1685	0.0086	0.0017	0.0317
V2	-0.004	-0.279	1	0.0485	0.1074	0.2535	-0.076	0.3294	0.001	-0.074	-0.16	-0.009	0.034	0.0193
V3	-0.317	-0.305	0.0485	1	0.0418	-0.152	0.122	-0.052	0.1198	-0.021	-0.05	0.0373	-0.035	0.0049
V4	-0.095	0.0614	0.1074	0.0418	1	0.0096	0.0532	0.0367	-0.005	0.0752	0.0276	-0.008	0.1022	-0.016
V5	0.1504	-0.07	0.2535	-0.152	0.0096	1	-0.013	0.337	-0.045	-0.043	-0.108	0.0071	-0.035	0.0115
V6	-0.074	-0.062	-0.076	0.122	0.0532	-0.013	1	-0.204	0.3461	0.0185	0.0435	0.0348	0.0253	-0.002
V7	0.078	-0.156	0.3294	-0.052	0.0367	0.337	-0.204	1	-0.279	-0.076	-0.146	0.0023	-0.032	-0.01
V8	-0.069	-0.174	0.001	0.1198	-0.005	-0.045	0.3461	-0.279	1	-0.002	-0.084	0.0328	0.0713	-0.096
V9	0.0117	0.0948	-0.074	-0.021	0.0752	-0.043	0.0185	-0.076	-0.002	1	-0.204	-0.022	0.0793	-0.035
V10	0.0096	0.1685	-0.16	-0.05	0.0276	-0.108	0.0435	-0.146	-0.084	-0.204	1	0.0227	-0.113	0.0002
V11	-0.152	0.0086	-0.009	0.0373	-0.008	0.0071	0.0348	0.0023	0.0328	-0.022	0.0227	1	0.1381	-0.024
V12	0.0504	0.0017	0.034	-0.035	0.1022	-0.035	0.0253	-0.032	0.0713	0.0793	-0.113	0.1381	1	0.1415
V13	-0.03	0.0317	0.0193	0.0049	-0.016	0.0115	-0.002	-0.01	-0.096	-0.035	0.0002	-0.024	0.1415	1
V14	-0.061	-0.016	0.0623	-0.095	0.0666	0.0287	-0.048	0.0452	0.0286	-0.023	0.0051	0.103	-0.032	-0.07
V15	-0.135	0.018	0.0256	0.0358	0.0396	-0.029	-0.033	-0.036	-0.011	-0.002	-0.019	-0.018	-0.047	0.0096
V16	0.0016	0.0264	0.0454	-0.032	-0.033	-0.006	-0.011	-0.085	0.0633	-0.026	-0.035	0.0462	-0.08	0.0002
V17	-0.069	-0.039	-0.004	-0.022	0.0344	-0.103	-0.021	-0.1	0.0498	-0.047	-0.102	0.0497	-0.138	-0.012
V18	0.0608	-0.029	0.0103	-0.031	0.0135	-0.009	0.0393	-0.047	0.0461	-0.008	-0.039	0.0435	-0.044	-0.007
V19	0.0168	0.0226	-6E-04	-0.011	-0.015	0.0089	0.0271	0.0001	-0.022	0.0012	0.0311	0.005	0.0134	-0.004
V20	-0.08	-0.158	0.0381	0.0743	-0.01	0.0412	0.0361	0.1145	0.0045	-0.017	-0.061	0.0286	0.0088	0.0676
V21	0.0652	-0.045	-0.069	-0.024	0.0276	-0.029	0.0246	-0.016	0.056	-0.046	-0.031	-6E-04	0.0207	-0.033
V22	0.089	-0.02	-0.039	0.0082	-0.002	-0.009	0.029	-0.017	0.0149	0.0009	0.0275	0.0007	-0.009	0.0045
V23	0.0992	0.1132	-0.068	-0.096	0.006	-0.077	-0.036	-0.13	0.0513	0.0511	0.0527	0.0209	0.0548	0.0015
V24	-0.01	-2E-04	0.0083	0.0032	-0.005	-0.024	-0.103	-0.008	-0.02	-0.001	-0.006	-0.001	-7E-04	0.004
V25	-0.172	0.0296	-0.053	0.0157	0.0433	-0.012	-0.007	-0.031	-0.051	-0.016	-0.01	-0.004	0.0022	0.0019
V26	######	0.0045	0.0183	-0.013	-0.038	0.0194	-0.015	0.0087	-0.005	0.0234	-0.018	-0.004	0.0091	-0.003
V27	-0.023	-0.093	0.0894	0.1039	-0.015	0.0053	0.0534	-0.077	0.1444	-6E-04	-0.013	-0.018	0.0192	0.0025
V28	-0.093	-0.216	0.0949	0.142	0.0086	-0.026	0.0147	0.0392	0.0686	-0.071	-0.094	-0.018	-0.006	-0.004
Amount	-0.027	-0.063	-0.352	######	-0.015	-0.216	0.1401	-0.023	0.0035	-0.052	0.0378	-0.03	-0.025	-0.004
Class	-0.01	-0.035	0.0417	-0.048	0.0515	-0.025	-0.031	-0.039	0.0185	-0.04	-0.049	0.0491	-0.051	-0.003

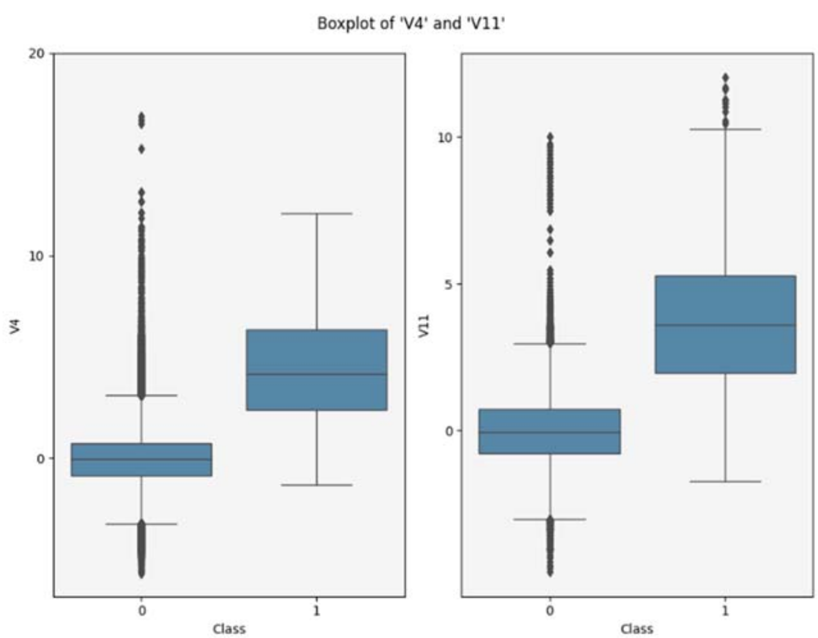
	V14	V15	V16	V17	V18	V19	V20	V21	V22	V23	V24	V25	V26	V27	V28	Amount	Class
Time	-0.061	-0.135	0.0016	-0.069	0.0608	0.0168	-0.08	0.0652	0.089	0.0992	-0.01	-0.172	######	-0.023	-0.093	-0.027	-0.01
V1	-0.016	0.018	0.0264	-0.039	-0.029	0.0226	-0.158	-0.045	-0.02	0.1132	-2E-04	0.0296	0.0045	-0.093	-0.216	-0.063	-0.035
V2	0.0623	0.0256	0.0454	-0.004	0.0103	-6E-04	0.0381	-0.069	-0.039	-0.068	0.0083	-0.053	0.0183	0.0894	0.0949	-0.352	0.0417
V3	-0.095	0.0358	-0.032	-0.022	-0.031	-0.011	0.0743	-0.024	0.0082	-0.096	0.0032	0.0157	-0.013	0.1039	0.142	######	-0.048
V4	0.0666	0.0396	-0.033	0.0344	0.0135	-0.015	-0.01	0.0276	-0.002	0.006	-0.005	0.0433	-0.038	-0.015	0.0086	-0.015	0.0515
V5	0.0287	-0.029	-0.006	-0.103	-0.009	0.0089	0.0412	-0.029	-0.009	-0.077	-0.024	-0.012	0.0194	0.0053	-0.026	-0.216	-0.025
V6	-0.048	-0.033	-0.011	-0.021	0.0393	0.0271	0.0361	0.0246	0.029	-0.036	-0.103	-0.007	-0.015	0.0534	0.0147	0.1401	-0.031
V7	0.0452	-0.036	-0.085	-0.1	-0.047	0.0001	0.1145	-0.016	-0.017	-0.13	-0.008	-0.031	0.0087	-0.077	0.0392	-0.023	-0.039
V8	0.0286	-0.011	0.0633	0.0498	0.0461	-0.022	0.0045	0.056	0.0149	0.0513	-0.02	-0.051	-0.005	0.1444	0.0686	0.0035	0.0185
V9	-0.023	-0.002	-0.026	-0.047	-0.008	0.0012	-0.017	-0.046	0.0009	0.0511	-0.001	-0.016	0.0234	-6E-04	-0.071	-0.052	-0.04
V10	0.0051	-0.019	-0.035	-0.102	-0.039	0.0311	-0.061	-0.031	0.0275	0.0527	-0.006	-0.01	-0.018	-0.013	-0.094	0.0378	-0.049
V11	0.103	-0.018	0.0462	0.0497	0.0435	0.005	0.0286	-6E-04	0.0007	0.0209	-0.001	-0.004	-0.004	-0.018	-0.018	-0.03	0.0491
V12	-0.032	-0.047	-0.08	-0.138	-0.044	0.0134	0.0088	0.0207	-0.009	0.0548	-7E-04	0.0022	0.0091	0.0192	-0.006	-0.025	-0.051
V13	-0.07	0.0096	0.0002	-0.012	-0.007	-0.004	0.0676	-0.033	0.0045	0.0015	0.004	0.0019	-0.003	0.0025	-0.004	-0.004	-0.003
V14	1	0.0122	-0.085	-0.112	-0.016	0.0145	-0.073	0.0581	-0.006	0.011	-0.015	0.0242	-0.002	-0.065	-0.015	-0.003	-0.053
V15	0.0122	1	0.0088	-0.015	0.0085	-0.039	0.0641	0.0206	0.0006	0.021	0.0151	0.0083	-0.01	0.022	0.0293	-0.048	-0.002
V16	-0.085	0.0088	1	-0.171	0.0299	0.011	0.1249	0.0604	-0.034	0.0305	-0.021	-0.01	0.0255	-0.03	-0.007	-0.072	-0.041
V17	-0.112	-0.015	-0.171	1	-0.113	0.0312	-0.012	-0.006	0.0082	0.0252	0.0358	0.0028	-0.006	0.0309	0.0443	0.0366	-0.036
V18	-0.016	0.0085	0.0299	-0.113	1	-0.002	-0.072	0.0193	0.0384	-0.075	-0.033	-0.004	-0.018	0.0131	0.0152	0.0312	-0.028
V19	0.0145	-0.039	0.011	0.0312	-0.002	1	0.1622	-0.004	-0.009	-0.054	-0.039	0.003	-0.01	-0.015	-0.03	-0.006	0.0184
V20	-0.073	0.0641	0.1249	-0.012	-0.072	0.1622	1	0.0827	0.023	-0.142	0.028	0.0309	0.0302	0.075	0.2001	0.141	0.0176
V21	0.0581	0.0206	0.0604	-0.006	0.0193	-0.004	0.0827	1	0.5589	-0.179	0.022	0.0251	-0.024	-0.018	0.0453	0.133	0.0289
V22	-0.006	0.0006	-0.034	0.0082	0.0384	-0.009	0.023	0.5589	1	-0.164	0.0104	0.0179	-0.021	0.0501	-0.026	0.0542	0.0017
V23	0.011	0.021	0.0305	0.0252	-0.075	-0.054	-0.142	-0.179	-0.164	1	0.0656	-0.305	-4E-04	0.0307	-0.054	-0.039	-0.004
V24	-0.015	0.0151	-0.021	0.0358	-0.033	-0.039	0.028	0.022	0.0104	0.0656	1	0.0008	0.0029	-0.024	0.0311	-0.008	-0.008
V25	0.0242	0.0083	-0.01	0.0028	-0.004	0.003	0.0309	0.0251	0.0179	-0.305	0.0008	1	-0.036	-0.058	-0.035	0.0156	0.0038
V26	-0.002	-0.01	0.0255	-0.006	-0.018	-0.01	0.0302	-0.024	-0.021	-4E-04	0.0029	-0.036	1	-0.111	-0.015	-0.046	0.0045
V27	-0.065	0.022	-0.03	0.0309	0.0131	-0.015	0.075	-0.018	0.0501	0.0307	-0.024	-0.058	-0.111	1	0.3553	-0.088	0.0231
V28	-0.015	0.0293	-0.007	0.0443	0.0152	-0.03	0.2001	0.0453	-0.026	-0.054	0.0311	-0.035	-0.015	0.3553	1	0.0097	0.0167
Amount	-0.003	-0.048	-0.072	0.0366	0.0312	-0.006	0.141	0.133	0.0542	-0.039	-0.008	0.0156	-0.046	-0.088	0.0097	1	-0.007
Class	-0.053	-0.002	-0.041	-0.036	-0.028	0.0184	0.0176	0.0289	0.0017	-0.004	-0.008	0.0038	0.0045	0.0231	0.0167	-0.007	1

	A	B	C	D	E
1	Field	Class			
2	Class	1			
3	V4	0.051476		V4	0.051476
4	V11	0.049107		V11	0.049107
5	V2	0.041692		V2	0.041692
6	V21	0.028938		V21	0.028938
7	V27	0.023116			
8	V8	0.01854			
9	V19	0.018409			
10	V20	0.017615			
11	V28	0.016671			
12	V26	0.004463			
13	V25	0.003823			
14	V22	0.0017			
15	V15	-0.00232			
16	V13	-0.00298			
17	V23	-0.0041			
18	Amount	-0.00682			
19	V24	-0.0075			
20	Time	-0.00955			
21	V5	-0.02462			
22	V18	-0.02847			
23	V6	-0.03136			
24	V1	-0.03464			
25	V17	-0.0362			
26	V7	-0.03944			
27	V9	-0.04042			
28	V16	-0.04077			
29	V3	-0.0484			
30	V10	-0.04863			
31	V12	-0.05133			
32	V14	-0.05276			

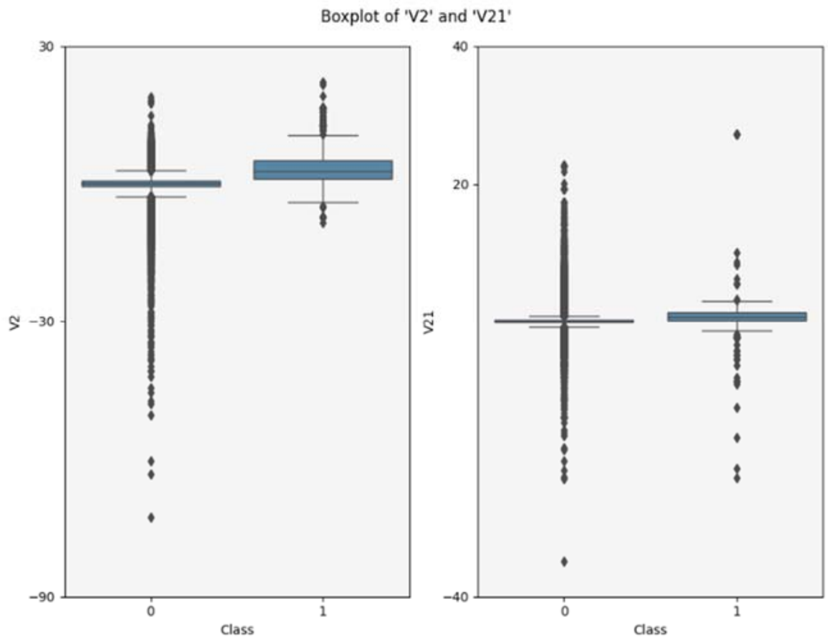
I can easily obtain the top 4 correlated features I can use in my predictive model.
Note: The number 4 was obtained from the PCA Cumulative Explained Variance above.



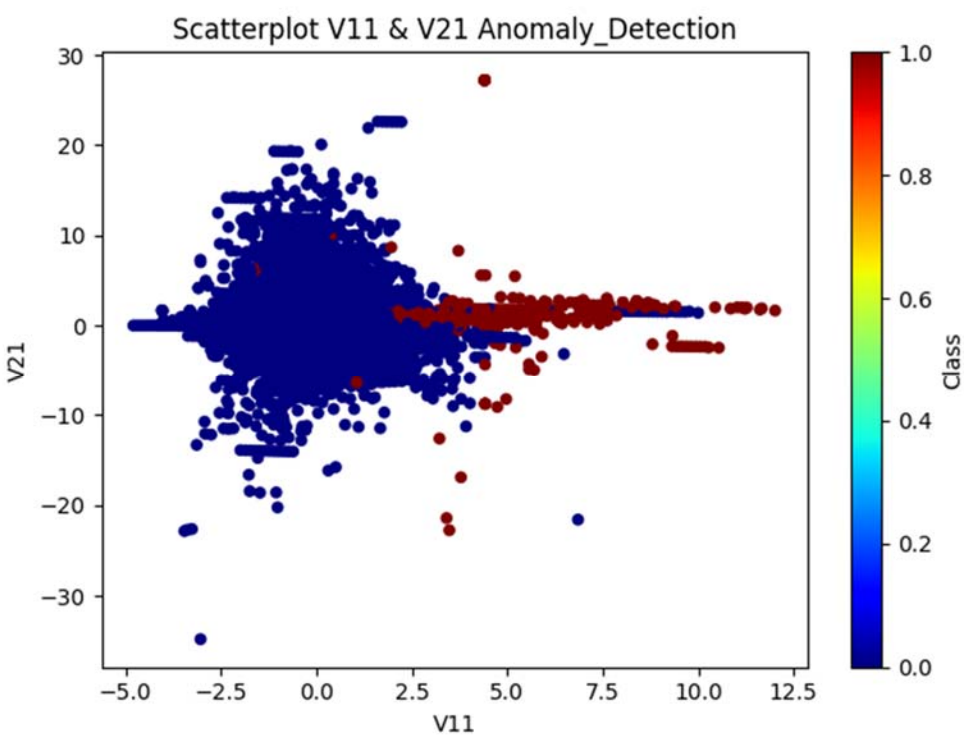
Notice the proximity indicating correlations between the clustered features marked in black and one in the red group. This aligns with the 4 features identified earlier per V2, V4, V11 and V21.



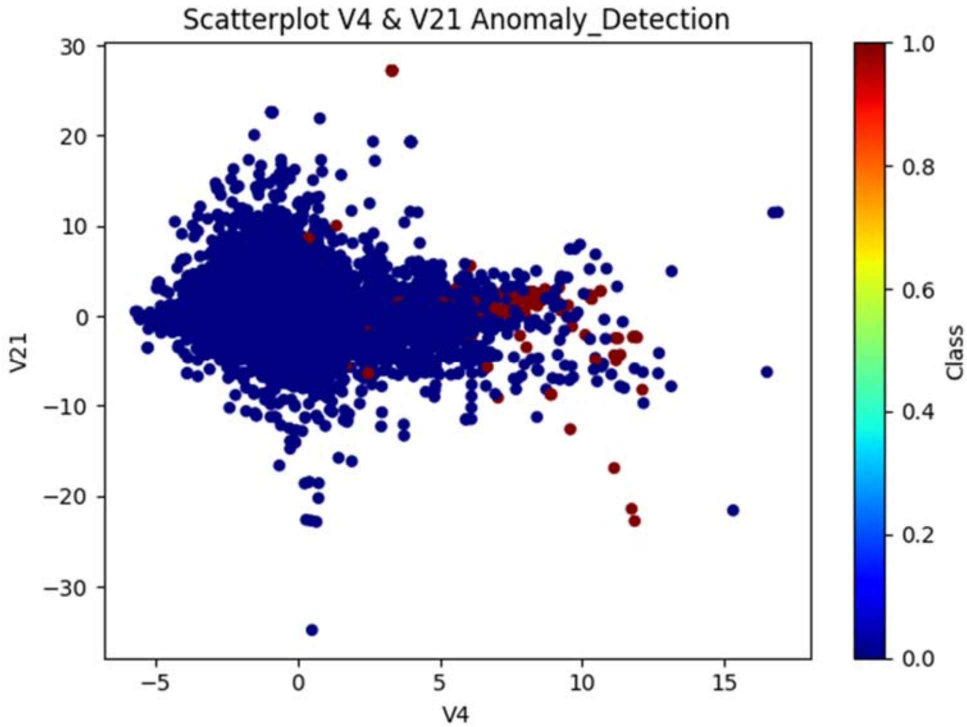
Notice the easy to find outliers on the V11 boxplot for Fraud class of 1.



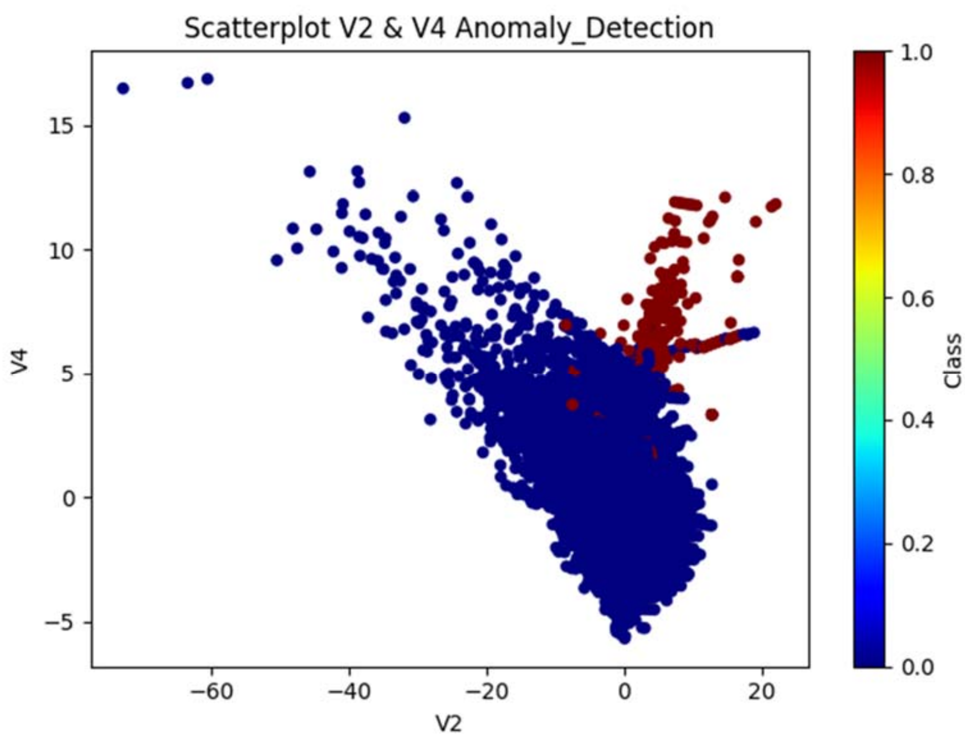
The above graph also shows an interesting amount of outlier for component V21.



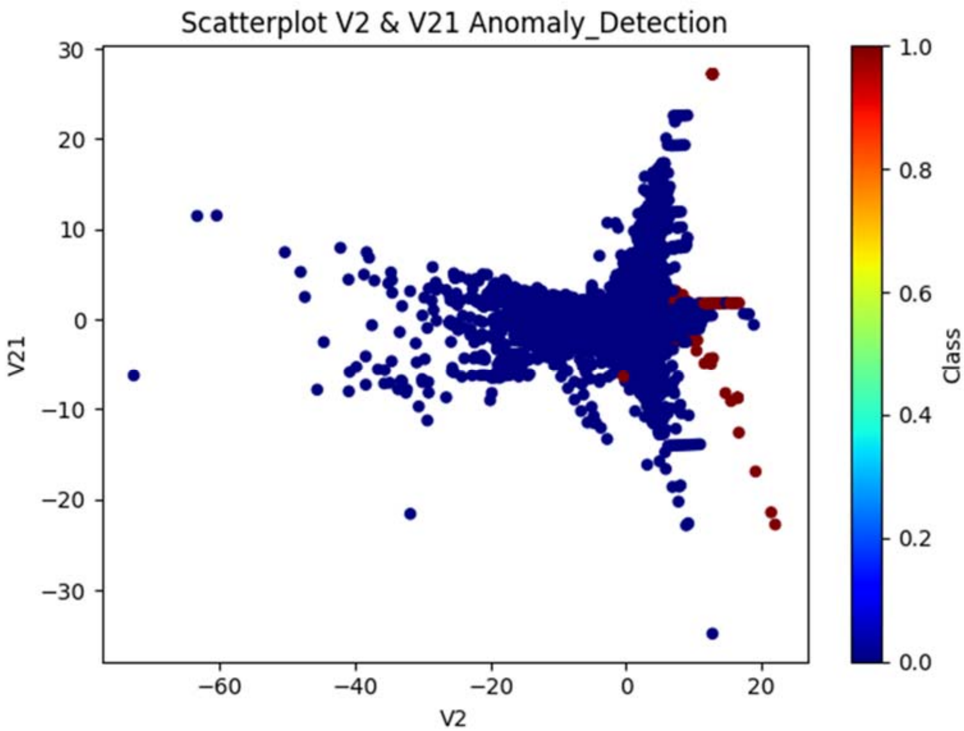
The above scatter plot displays some specific indicator in red - that can help assist with possible fraud customers.



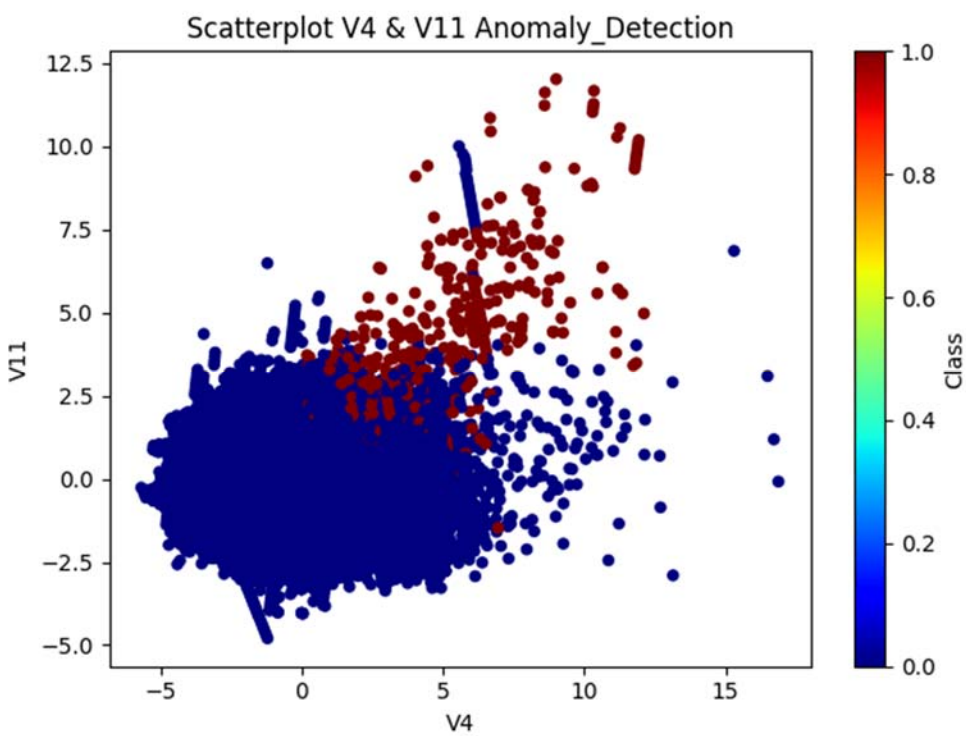
The above scatter plot displays some specific indicator that can help assist with possible fraud customers.



The above scatter plot does a better job of identifying likely fraud customers as noted in the red markers.



The above scatter plot displays some specific indicator that can help assist with possible fraud customers.



The above scatter plot appears to do the best job of identifying likely fraud customers. Notice the obvious red grouping outside of those who do not fraud in blue.

Model Algorithms
For my project I had earlier planned to test out 5 algorithms for my model.

These algorithms included:

Linear Regression
Isolation Forest
Elliptic Envelope
Local Outlier Factor
One Class SVM

The metrics I used to review the results of these algorithms include: **MAE, MSE, RMSE and r2 Score.**

```
AnomalyDetectionModelPredict
LinearRegression
[ 0.00045597 -0.00161791  0.00252932 ...  0.00155776  0.00290815
 -0.00070254]
MAE: 0.003
MSE: 0.001
RMSE: 0.029
r2 Score: 0.468
LinearRegression Complete Duration: --- 0.3464233875274658 seconds ---
IsolationForest
[1 1 1 ... 1 1 1]
(171738, 30) (171738,)
MAE: 0.002
MSE: 0.002
RMSE: 0.039
r2 Score: 0.027
IsolationForest Complete Duration: --- 26.20404052734375 seconds ---
EllipticEnvelope
[1 1 1 ... 1 1 1]
(170020, 30) (170020,)
MAE: 0.009
MSE: 0.016
RMSE: 0.127
r2 Score: -9.221
EllipticEnvelope Complete Duration: --- 60.35044598579407 seconds ---
LocalOutlierFactor
[1 1 1 ... 1 1 1]
(167007, 30) (167007,)
MAE: 0.009
MSE: 0.013
RMSE: 0.115
r2 Score: -7.396
LocalOutlierFactor Complete Duration: --- 669.6039683818817 seconds ---
OneClassSVM
[1 1 1 ... 1 1 1]
(165184, 30) (165184,)
MAE: 0.009
MSE: 0.013
RMSE: 0.116
r2 Score: -7.510
OneClassSVM Complete Duration: --- 68.86110067367554 seconds ---
Complete Duration: --- 827.1672956943512 seconds ---
```

Out of the 5 algorithms, I found that the Isolation Forest algorithm had the best MAE: of .002, MSE: .002 which highlights the accuracy by way of measuring how far our prediction came from the actual values.

However, the r2 score for Isolation Forest only came back up with .027 which is very weak to explain variances in our dependent variable due to variability of our independent variables.

The second-best algorithm was Linear Regression per MAE of .003 and MSE of .001. However, the r2 score was moderately good with a .468. Meaning it was better at explaining variances.

I later added an algorithm review using PyCaret to test more algorithms for the best performance.

PyCaret Model Review

```
pyCaretTest
C:\Users\aland\PycharmProjects\RentalPricePrediction\venv\Scripts\python.exe C:/Users/aland/
                  Model  Accuracy     AUC  Recall   Prec.      F1   Kappa  \
lr  Logistic Regression    0.9991  0.9293   0.598  0.8275  0.6845  0.6841

       MCC  TT (Sec)
lr  0.6979      5.57
                      Model  Accuracy     AUC  Recall   Prec.      F1   Kappa  \
lr      Logistic Regression    0.9991  0.9293  0.5980  0.8275  0.6845  0.6841
knn  K Neighbors Classifier    0.9984  0.6028  0.0537  0.9667  0.1005  0.1004

        MCC  TT (Sec)
lr   0.6979     5.570
knn  0.2204     1.374
                      Model  Accuracy     AUC  Recall   Prec.      F1   Kappa  \
lr      Logistic Regression    0.9991  0.9293  0.5980  0.8275  0.6845  0.6841
knn  K Neighbors Classifier    0.9984  0.6028  0.0537  0.9667  0.1005  0.1004
nb              Naive Bayes    0.9924  0.9634  0.6184  0.1300  0.2146  0.2124

        MCC  TT (Sec)
lr   0.6979     5.570
knn  0.2204     1.374
nb   0.2810     0.144
                        Model  Accuracy     AUC  Recall   Prec.      F1  \
lr        Logistic Regression    0.9991  0.9293  0.5980  0.8275  0.6845
dt   Decision Tree Classifier    0.9990  0.8569  0.7143  0.7191  0.7146
knn    K Neighbors Classifier    0.9984  0.6028  0.0537  0.9667  0.1005
nb                Naive Bayes    0.9924  0.9634  0.6184  0.1300  0.2146

      Kappa     MCC  TT (Sec)
lr   0.6841  0.6979     5.570
dt   0.7141  0.7151     2.323
knn  0.1004  0.2204     1.374
nb   0.2124  0.2810     0.144
```

PyCaret evaluations between algorithms: Logistic Regression, K Nearest Neighbors, Naïve Bayes, and Decision Tree Classifier.

	Model	Accuracy	AUC	Recall	Prec.	F1
lr	Logistic Regression	0.9991	0.9293	0.5980	0.8275	0.6845
dt	Decision Tree Classifier	0.9990	0.8569	0.7143	0.7191	0.7146
knn	K Neighbors Classifier	0.9984	0.6028	0.0537	0.9667	0.1005
svm	SVM - Linear Kernel	0.9982	0.0000	0.0059	0.0286	0.0098
nb	Naive Bayes	0.9924	0.9634	0.6184	0.1300	0.2146

	Kappa	MCC	TT (Sec)
lr	0.6841	0.6979	5.570
dt	0.7141	0.7151	2.323
knn	0.1004	0.2204	1.374
svm	0.0095	0.0126	2.751
nb	0.2124	0.2810	0.144

During the first phases of evaluations, we can see K Nearest Neighbors leads in both Accuracy and Precision however, the F1 score that combines an evaluation of both precision and recall is low.

	Model	Accuracy	AUC	Recall	Prec.
rf	Random Forest Classifier	0.9995	0.9430	0.7589	0.9606
lr	Logistic Regression	0.9991	0.9293	0.5980	0.8275
dt	Decision Tree Classifier	0.9990	0.8569	0.7143	0.7191
ridge	Ridge Classifier	0.9988	0.0000	0.3944	0.8429
knn	K Neighbors Classifier	0.9984	0.6028	0.0537	0.9667
svm	SVM - Linear Kernel	0.9982	0.0000	0.0059	0.0286
nb	Naive Bayes	0.9924	0.9634	0.6184	0.1300
qda	Quadratic Discriminant Analysis	0.9747	0.9675	0.8481	0.0540

	F1	Kappa	MCC	TT (Sec)
rf	0.8450	0.8448	0.8521	27.441
lr	0.6845	0.6841	0.6979	5.570
dt	0.7146	0.7141	0.7151	2.323
ridge	0.5299	0.5294	0.5714	0.149
knn	0.1005	0.1004	0.2204	1.374
svm	0.0098	0.0095	0.0126	2.751
nb	0.2146	0.2124	0.2810	0.144
qda	0.1015	0.0987	0.2102	0.717

As the PyCaret evaluation proceeds, I find it interesting how K Nearest Neighbors hangs in there with high precision while contending with algorithms: Ridge Classifier and Random Forest Classifier. Here we can see that as even more algorithms get added to the evaluation, K Nearest Neighbors continues to hold well to Precision despite not leading accuracy. We can see that the Random Forest Classifier and Ridge Classifier continues to comparatively run well against KNN.

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC	TT (Sec)
et	Extra Trees Classifier	0.9996	0.9428	0.7646	0.9660	0.8521	0.8519	0.8585	11.051
rf	Random Forest Classifier	0.9995	0.9430	0.7589	0.9606	0.8450	0.8448	0.8521	27.441
lda	Linear Discriminant Analysis	0.9994	0.8868	0.7469	0.8667	0.7989	0.7985	0.8025	0.742
ada	Ada Boost Classifier	0.9992	0.9702	0.6543	0.8163	0.7216	0.7212	0.7280	9.408
lr	Logistic Regression	0.9991	0.9293	0.5980	0.8275	0.6845	0.6841	0.6979	5.570
dt	Decision Tree Classifier	0.9990	0.8569	0.7143	0.7191	0.7146	0.7141	0.7151	2.323
gbc	Gradient Boosting Classifier	0.9990	0.6593	0.5012	0.7980	0.6001	0.5996	0.6230	52.235
ridge	Ridge Classifier	0.9988	0.0000	0.3944	0.8429	0.5299	0.5294	0.5714	0.149
knn	K Neighbors Classifier	0.9984	0.6028	0.0537	0.9667	0.1005	0.1004	0.2204	1.374
svm	SVM - Linear Kernel	0.9982	0.0000	0.0059	0.0286	0.0098	0.0095	0.0126	2.751
nb	Naive Bayes	0.9924	0.9634	0.6184	0.1300	0.2146	0.2124	0.2810	0.144
qda	Quadratic Discriminant Analysis	0.9747	0.9675	0.8481	0.0540	0.1015	0.0987	0.2102	0.717

Here we can see that KNN is getting surpassed by the algorithms: Random Forest Classifier and Extra Trees Classifier which is much like Random Forest. With high Accuracy, AUC, Recall and Precision.

```
              F1   Kappa     MCC  TT (Sec)
et        0.8521  0.8519  0.8585    11.051
rf        0.8450  0.8448  0.8521    27.441
xgboost   0.8473  0.8470  0.8535    16.088
lda       0.7989  0.7985  0.8025     0.742
ada       0.7216  0.7212  0.7280     9.408
lr        0.6845  0.6841  0.6979     5.570
dt        0.7146  0.7141  0.7151     2.323
gbc       0.6001  0.5996  0.6230    52.235
ridge     0.5299  0.5294  0.5714     0.149
knn       0.1005  0.1004  0.2204     1.374
svm       0.0098  0.0095  0.0126     2.751
lightgbm  0.2747  0.2730  0.3074     1.866
nb        0.2146  0.2124  0.2810     0.144
qda       0.1015  0.0987  0.2102     0.717
best
ExtraTreesClassifier(bootstrap=False, ccp_alpha=0.0, class_weight=None,
                     criterion='gini', max_depth=None, max_features='auto',
                     max_leaf_nodes=None, max_samples=None,
                     min_impurity_decrease=0.0, min_impurity_split=None,
                     min_samples_leaf=1, min_samples_split=2,
                     min_weight_fraction_leaf=0.0, n_estimators=100, n_jobs=-1,
                     oob_score=False, random_state=4113, verbose=0,
                     warm_start=False)
Fitting 10 folds for each of 200 candidates, totalling 2000 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 12 concurrent workers.
[Parallel(n_jobs=-1)]: Done  26 tasks      | elapsed:  3.1min
[Parallel(n_jobs=-1)]: Done 176 tasks      | elapsed: 11.8min
```

Here we see an evaluation using the F1 to score the relationship of precision and recall, Kappa magnitude score to measure dichotomous agreement with a score higher than .76. And MCC - Mathew's Correlation Coefficient that measures of .8585. One thing to note though, since we have an imbalanced dataset and F1 score is asymmetric by nature meaning it does not provide similar results if the classes are inverted thus F1 score alone my not be useful as a metric for my project. Another note on the MCC which is a symmetric metric that considers the TP/True Positives, FP/False Positives and FN/False Negatives. It indicates a better score in favor of Extra Trees.

Run: pyCaretTest

```
                              Model  Accuracy     AUC  Recall   Prec.  \
rf             Random Forest Classifier    0.9996  0.9394  0.7981  0.9487
et               Extra Trees Classifier    0.9996  0.9450  0.7954  0.9506
xgboost         Extreme Gradient Boosting    0.9996  0.9802  0.8039  0.9457
lda        Linear Discriminant Analysis    0.9993  0.8948  0.7508  0.8614
dt             Decision Tree Classifier    0.9992  0.8920  0.7844  0.7649
lr                  Logistic Regression    0.9991  0.9409  0.6363  0.8298
ada                Ada Boost Classifier    0.9991  0.9731  0.6840  0.8135
ridge                  Ridge Classifier    0.9989  0.0000  0.4441  0.8465
gbc        Gradient Boosting Classifier    0.9987  0.5349  0.3700  0.7688
knn              K Neighbors Classifier    0.9983  0.6145  0.0645  0.7667
svm                 SVM - Linear Kernel    0.9979  0.0000  0.0833  0.0606
nb                          Naive Bayes    0.9933  0.9676  0.6421  0.1611
lightgbm  Light Gradient Boosting Machine    0.9931  0.6171  0.4393  0.1959
qda       Quadratic Discriminant Analysis    0.9754  0.9703  0.8739  0.0607

              F1   Kappa     MCC  TT (Sec)
rf        0.8635  0.8633  0.8682    26.728
et        0.8638  0.8636  0.8682     9.595
xgboost   0.8672  0.8670  0.8708    14.283
lda       0.7998  0.7995  0.8026     0.680
dt        0.7724  0.7720  0.7731     1.960
lr        0.7156  0.7151  0.7238     4.831
ada       0.7414  0.7410  0.7447     8.291
ridge     0.5807  0.5802  0.6116     0.130
gbc       0.4654  0.4649  0.5102    36.406
knn       0.1175  0.1173  0.2166     1.058
svm       0.0665  0.0660  0.0685     2.422
nb        0.2571  0.2549  0.3190     0.133
lightgbm  0.2551  0.2532  0.2785     1.139
qda       0.1134  0.1105  0.2265     0.701
best
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=None, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=100,
                       n_jobs=-1, oob_score=False, random_state=6202, verbose=0,
                       warm_start=False)
Fitting 10 folds for each of 200 candidates, totalling 2000 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 12 concurrent workers.
```

Here we can see that the two algorithms: Random Forest Classifier and Extra Trees Classifier are neck and neck in accuracy and very comparable per AUC, Recall and Precision, Kappa and MCC. However, the training time for the Extra Trees Classifier is much better at 9.595 seconds vs 26.728 seconds for Random Forest Classifier.

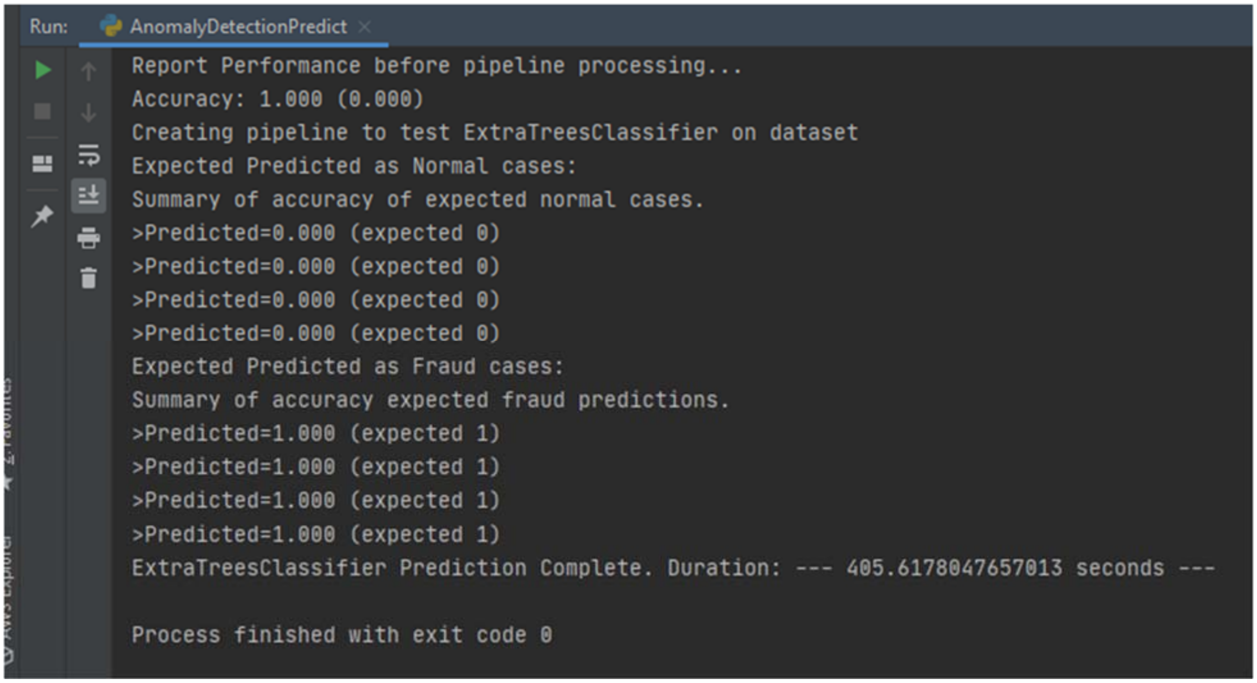
pyCaretTest

```
Fitting 10 folds for each of 200 candidates, totalling 2000 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 12 concurrent workers.
[Parallel(n_jobs=-1)]: Done  26 tasks      | elapsed:  6.6min
[Parallel(n_jobs=-1)]: Done 176 tasks      | elapsed: 20.0min
[Parallel(n_jobs=-1)]: Done 426 tasks      | elapsed: 39.7min
[Parallel(n_jobs=-1)]: Done 776 tasks      | elapsed: 68.2min
[Parallel(n_jobs=-1)]: Done 1226 tasks      | elapsed: 106.9min
[Parallel(n_jobs=-1)]: Done 1776 tasks      | elapsed: 147.2min
[Parallel(n_jobs=-1)]: Done 2000 out of 2000 | elapsed: 167.0min finished
      Accuracy     AUC  Recall   Prec.      F1   Kappa     MCC
0       0.9995  0.9809  0.8889  0.8649  0.8767  0.8765  0.8766
1       0.9996  0.9729  0.8056  0.9667  0.8788  0.8786  0.8823
2       0.9994  0.9780  0.7222  0.9286  0.8125  0.8122  0.8186
3       0.9995  0.9743  0.7778  0.9333  0.8485  0.8482  0.8518
4       0.9996  0.9868  0.8286  0.9355  0.8788  0.8786  0.8802
5       0.9993  0.9781  0.7222  0.8966  0.8000  0.7997  0.8044
6       0.9996  0.9740  0.8056  0.9667  0.8788  0.8786  0.8823
7       0.9995  0.9967  0.7500  0.9643  0.8437  0.8435  0.8502
8       0.9994  0.9790  0.7500  0.9310  0.8308  0.8305  0.8354
9       0.9994  0.9973  0.7500  0.9310  0.8308  0.8305  0.8354
Mean    0.9995  0.9818  0.7801  0.9318  0.8479  0.8477  0.8517
SD      0.0001  0.0085  0.0499  0.0304  0.0280  0.0280  0.0268
ExtraTreesClassifier(bootstrap=False, ccp_alpha=0.0, class_weight=None,
                     criterion='gini', max_depth=None, max_features='auto',
                     max_leaf_nodes=None, max_samples=None,
                     min_impurity_decrease=0.0, min_impurity_split=None,
                     min_samples_leaf=1, min_samples_split=2,
                     min_weight_fraction_leaf=0.0, n_estimators=100,
                     n_jobs=None, oob_score=False, random_state=None, verbose=0,
                     warm_start=False)

Process finished with exit code 0
```

Here my PyCaret evaluation has identified that the best algorithm for my project's model is "Extra Trees Classifier".

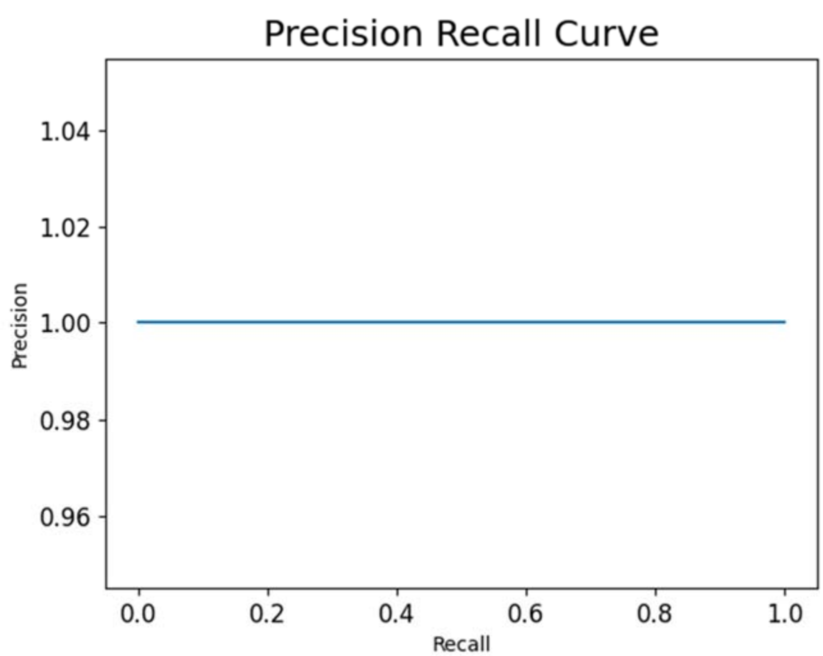
Prediction Pipeline Validation:



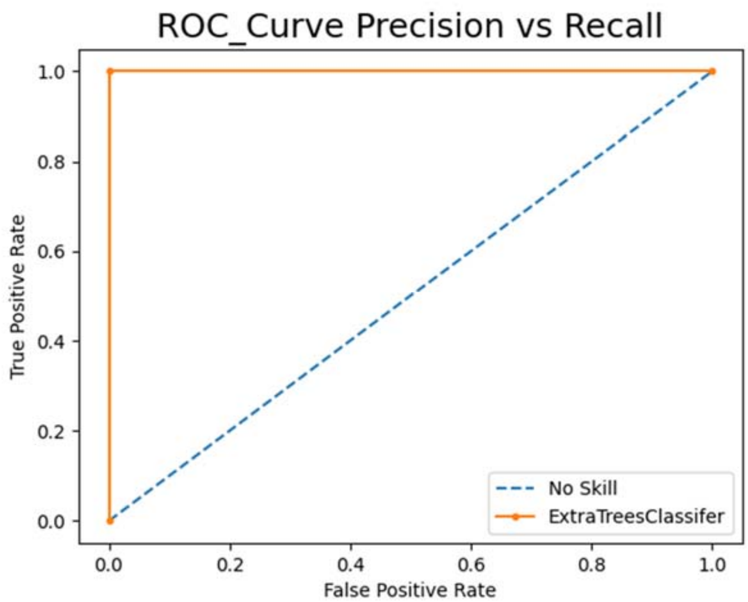
```
Report Performance before pipeline processing...
Accuracy: 1.000 (0.000)
Creating pipeline to test ExtraTreesClassifier on dataset
Expected Predicted as Normal cases:
Summary of accuracy of expected normal cases.
>Predicted=0.000 (expected 0)
>Predicted=0.000 (expected 0)
>Predicted=0.000 (expected 0)
>Predicted=0.000 (expected 0)
Expected Predicted as Fraud cases:
Summary of accuracy expected fraud predictions.
>Predicted=1.000 (expected 1)
>Predicted=1.000 (expected 1)
>Predicted=1.000 (expected 1)
>Predicted=1.000 (expected 1)
ExtraTreesClassifier Prediction Complete. Duration: --- 405.6178047657013 seconds ---

Process finished with exit code 0
```

Here we will create a pipeline to feed in a series of values to simulate the input variables of our model and perform our prediction. Each prediction was completed as expected with the correct result using random test data.

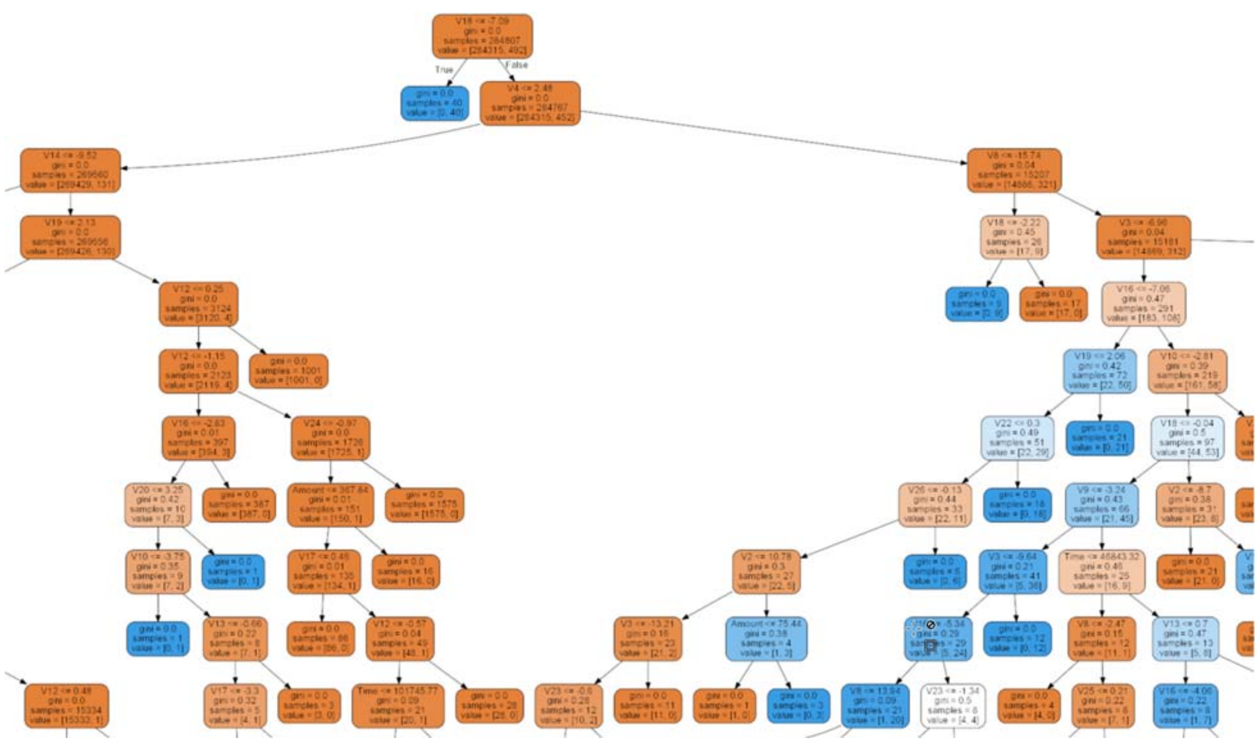


The above Precision Recall Curve plots how well the prediction vs test values matched per TP/True Positives, FP/False Positives and FN/False Negatives. It was measured using Sklearn's average_precision_score metric function therefore a weighted mean of precisions.



The above Precision vs Recall plot shows how well the prediction vs test values matched per TP/True Positives, FP/False Positives and FN/False Negatives using an area over the curve visualization. It uses Sklearn's precision_recall_curve metric function. Since the purpose of my project is to identify false positives i.e. Fraud - which is more important than to false negatives (missing fraudulent transactions), this project is focused more on specificity and therefore precision over sensitivity.

Full Visualization of the decision tree of my model.



Zoomed in view of the center of the decision tree.

Conclusion:

Credit card fraud is an extremely important issue as it devastates a person's credit, violates their identity and privacy of their personal information. Here we learned that we can detect this type of anomaly to be sensitive to the variations in the features of one's credit usage. And build a predictive model that has an accuracy greater than 99%.

Appendixes:

Variables:

Time	Number of seconds between the transactions in the dataset
V1	PCA translated
V2	PCA translated
V3	PCA translated
V4	PCA translated
V5	PCA translated
V6	PCA translated
V7	PCA translated
V8	PCA translated
V9	PCA translated
V10	PCA translated
V11	PCA translated
V12	PCA translated
V13	PCA translated
V14	PCA translated
V15	PCA translated
V16	PCA translated
V17	PCA translated
V18	PCA translated
V19	PCA translated
V20	PCA translated
V21	PCA translated
V22	PCA translated
V23	PCA translated
V24	PCA translated
V25	PCA translated
V26	PCA translated
V27	PCA translated
V28	PCA translated
Amount	Transaction Amount
Class	1 for fraud, 0 for normal

Data Sources:

https://www.kaggle.com/mlg-ulb/creditcardfraud

References:

Sandberg, E.(August 2020). 15 Disturbing Credit Card Fraud Statistics. Retrieved from: https://www.cardrates.com/advice/credit-card-fraud-statistics/

Bessette, C. (June 2020). How serious a Crime Is Credit Card Theft and Fraud? Retrieved from: https://www.nerdwallet.com/article/credit-cards/credit-card-theft-fraud-serious-crime-penalty

Bank, E. (May 2020). My Credit Card was Used Fraudulently (Here's What to Do). Retrieved from: https://www.cardrates.com/advice/my-credit-card-was-used-fraudulently-heres-what-to-do/

Li, S. (July 2019). Anomaly Detection for Dummies. Retrieved from: https://towardsdatascience.com/anomaly-detection-for-dummies-15f148e559c1

Aliyev, V. (October 2020). 3 Simple Outlier/Anomaly Detection Algorithms every Data Scientist needs. Retrieved from: https://towardsdatascience.com/3-simple-outlier-anomaly-detection-algorithms-every-data-scientist-needs-e71b1304a932

Alam, M. (September 2020). Anomaly detection with Local Outlier Factor (LOF). Retrieved from: https://towardsdatascience.com/anomaly-detection-with-local-outlier-factor-lof-d91e41df10f2

Flovik, V. (April 2019). Machine learning for anomaly detection and condition monitoring. Retrieved from: https://towardsdatascience.com/machine-learning-for-anomaly-detection-and-condition-monitoring-d4614e7de770

Sucky, R. (October 2020). A Complete Anomaly Detection Algorithm From Scratch in Python: Step by Step Guide. Retrieved from: https://towardsdatascience.com/a-complete-anomaly-detection-algorithm-from-scratch-in-python-step-by-step-guide-e1daf870336e

Garbade, M. (December 2020). How to use Machine Learning for Anomaly Detection and Conditional Monitoring. Retrieved from: https://www.kdnuggets.com/2020/12/machine-learning-anomaly-detection-conditional-monitoring.html#:~:text=The%20main%20goal%20of%20Anomaly,useful%20in%20understanding%20data%20problems

Vemula, A. (May 2020). Anomaly Detection made simple. Credit card fraud case using PyCaret package. Retrieved from: https://towardsdatascience.com/anomaly-detection-made-simple-70775c914377

Chuprina, R. (February 2021). Credit Card Fraud Detection: Top ML Solutions in 2021. Retrieved from: https://spd.group/machine-learning/credit-card-fraud-detection/

Alam, M. (October 2020). Machine Learning of anomaly detection: Elliptic Envelope. Retrieved from https://towardsdatascience.com/machine-learning-for-anomaly-detection-elliptic-envelope-2c90528df0a6

Alam, M. (September 2020). Anomaly detection with Local Outlier Factor (LOF). Retrieved from: https://towardsdatascience.com/anomaly-detection-with-local-outlier-factor-lof-d91e41df10f2

Young, A. (November 2020). Isolation Forest is the best Anomaly Detection Algorithm for Big Data Right Now. Retrieved from: https://towardsdatascience.com/isolation-forest-is-the-best-anomaly-detection-algorithm-for-big-data-right-now-e1a18ec0f94f

Dawson, C. (November 2018). Outlier Detection with One-Class SVMS. Retrieved from: https://towardsdatascience.com/outlier-detection-with-one-class-svms-5403a1a1878c

Cicchetti, D.V., & Sparrow, S.A. (1981). Developing criteria for establishing interrater reliability of specific items: applications to assessment of adaptive behavior. American Journal of Mental Deficiency, 86(2), 127-137.

Landis, J.R., & Koch, G. G. (1977). The measurement of observer agreement for categorical data. Biometrics, 15-174.

Fleiss, J.L. (1981). Statistical methods for rates and proportions (Second Edition). New York: Wiley.