Gabriel Madison, Paul Webster, Mirza Mohammed Abdullah Baig & Brandon Marlowe
EEC 525/CIS 660 Final Report
12/05/2018

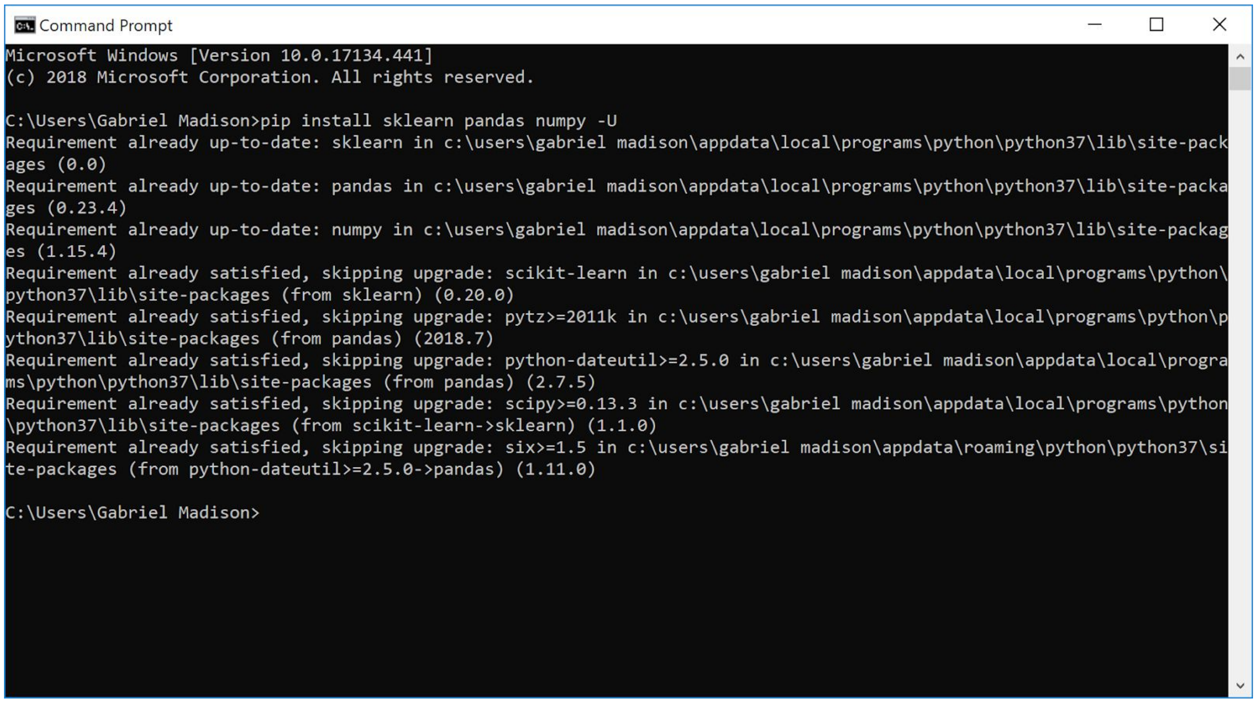
Introduction

For our project we used the Aegean Wi-Fi Intrusion Dataset (AWID). There are two versions of this dataset, one is over 200 million tuples of full data and one is 1.8 million tuples of reduced data. We used the reduced version and we predicted if an attack was one of four types using the k-nearest-neighbors classifier. All of the attack types we used had a distribution that highly favored the non-attack class. Our best results were for the attack “arp” type where we attained over 99% accuracy with over 99% recall. For the other attack types our results were not as good. Below is a report about how we conducted our experiment and the results we obtained.

Software Used

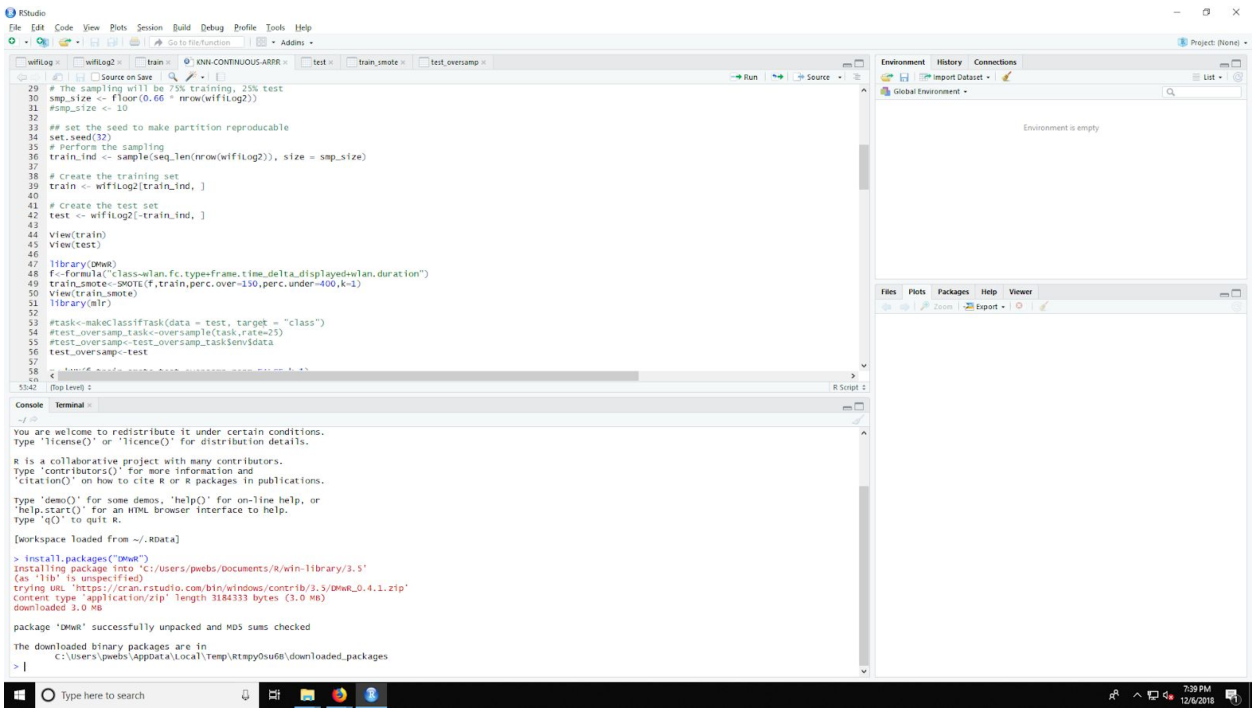
In our project, we used three different programming languages to assist us -- Python, R-Studio, and Microsoft SQL. For Python, we used the latest version available at the time of writing -- Python 3.7.1. In this language, we used the Scikit-Learn, Pandas, and NumPy libraries to help us with our preprocessing steps. Then, in R we used the DMwR library for its SMOTE and K-nearest-neighbors functionality. Finally for Microsoft SQL, we used Server Management Studio to perform queries.

Below is a screenshot of installing Scikit-Learn, Pandas and NumPy into Python:



```
Microsoft Windows [Version 10.0.17134.441]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\Gabriel Madison>pip install sklearn pandas numpy -U
Requirement already up-to-date: sklearn in c:\users\gabriel madison\appdata\local\programs\python\python37\lib\site-pack
ages (0.0)
Requirement already up-to-date: pandas in c:\users\gabriel madison\appdata\local\programs\python\python37\lib\site-packa
ges (0.23.4)
Requirement already up-to-date: numpy in c:\users\gabriel madison\appdata\local\programs\python\python37\lib\site-packag
es (1.15.4)
Requirement already satisfied, skipping upgrade: scikit-learn in c:\users\gabriel madison\appdata\local\programs\python\
python37\lib\site-packages (from sklearn) (0.20.0)
Requirement already satisfied, skipping upgrade: pytz>=2011k in c:\users\gabriel madison\appdata\local\programs\python\p
ython37\lib\site-packages (from pandas) (2018.7)
Requirement already satisfied, skipping upgrade: python-dateutil>=2.5.0 in c:\users\gabriel madison\appdata\local\progra
ms\python\python37\lib\site-packages (from pandas) (2.7.5)
Requirement already satisfied, skipping upgrade: scipy>=0.13.3 in c:\users\gabriel madison\appdata\local\programs\python
\python37\lib\site-packages (from scikit-learn->sklearn) (1.1.0)
Requirement already satisfied, skipping upgrade: six>=1.5 in c:\users\gabriel madison\appdata\roaming\python\python37\si
te-packages (from python-dateutil>=2.5.0->pandas) (1.11.0)

C:\Users\Gabriel Madison>
```

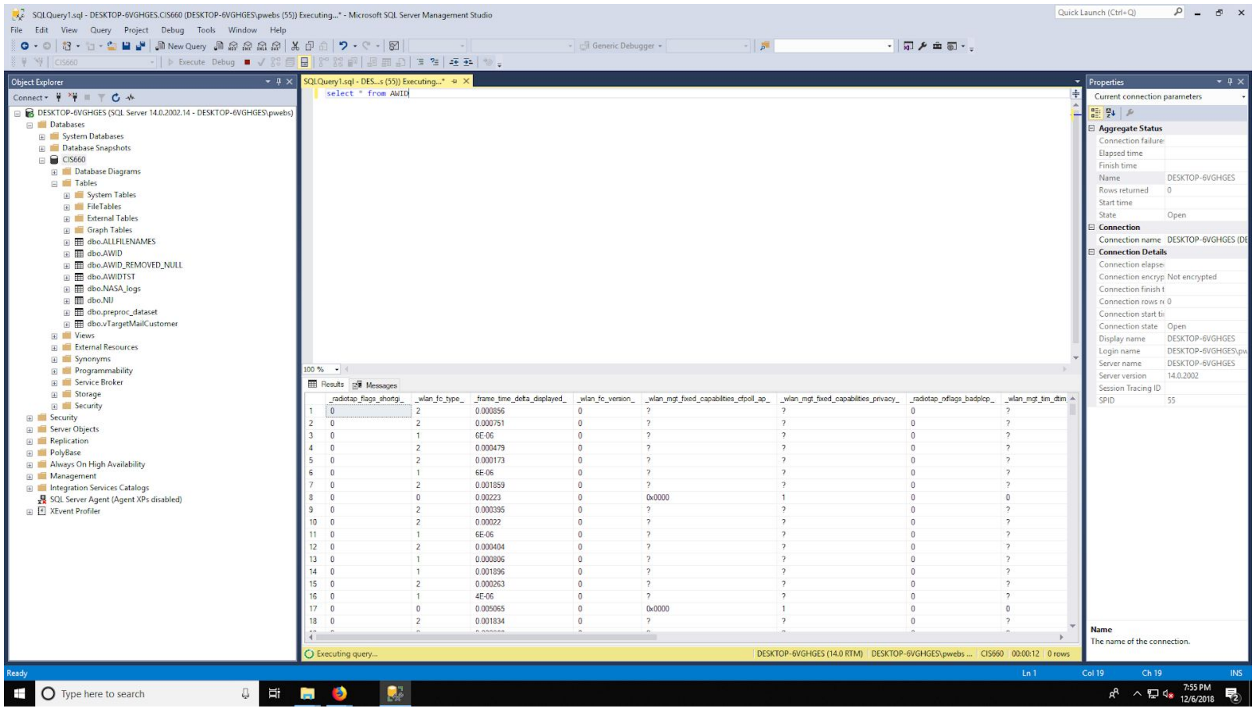
Below is a screenshot of installing the DMwR R library into R-Studio.



```
# The sampling will be 75% training, 25% test
smp_size <- floor(0.66 * nrow(wifiLog2))
#smp_size <- 10

## set the seed to make partition reproducable
set.seed(32)
# Perform the sampling
train_ind <- sample(seq_len(nrow(wifiLog2)), size = smp_size)

# Create the training set
train <- wifiLog2[train_ind, ]

# Create the test set
test <- wifiLog2[-train_ind, ]

View(train)
View(test)

library(DMwR)
f<-formula("class~wlan.fc.type+frame.time_delta_displayed+wlan.duration")
train_smote<-SMOTE(f,train,perc.over=150,perc.under=400,k=1)
View(train_smote)
library(mlr)

#task<-makeClassifTask(data = test, target = "class")
#test_oversamp_task<-oversample(task,rate=25)
#test_oversamp<-test_oversamp_task$env$data
test_oversamp<-test
```

```
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

[Workspace loaded from ~/.RData]

> install.packages("DMwR")
Installing package into 'C:/Users/pwebs/Documents/R/win-library/3.5'
(as 'lib' is unspecified)
trying URL 'https://cran.rstudio.com/bin/windows/contrib/3.5/DMwR_0.4.1.zip'
Content type 'application/zip' length 3184333 bytes (3.0 MB)
downloaded 3.0 MB

package 'DMwR' successfully unpacked and MD5 sums checked

The downloaded binary packages are in
        C:\Users\pwebs\AppData\Local\Temp\RtmpyOsu6B\downloaded_packages
> 
```

To use SQL server we imported the reduced version of our dataset into it after adding the headers to it. Below is a screenshot of the reduced AWID dataset with only the columns posted on the class webpage in Microsoft SQL Server:



SQLQuery1.sql - DESKTOP-6VGHGES.CIS660 (DESKTOP-6VGHGES\pwebs (55)) Executing...* - Microsoft SQL Server Management Studio

```
select * from AWID
```

```
DESKTOP-6VGHGES (SQL Server 14.0.2002.14 - DESKTOP-6VGHGES\pwebs)
  Databases
    System Databases
    Database Snapshots
    CIS660
      Database Diagrams
      Tables
        System Tables
        FileTables
        External Tables
        Graph Tables
        dbo.ALLFILENAMES
        dbo.AWID
        dbo.AWID_REMOVED_NULL
        dbo.AWIDTST
        dbo.NASA_logs
        dbo.NIJ
        dbo.preproc_dataset
        dbo.vTargetMailCustomer
      Views
      External Resources
      Synonyms
      Programmability
      Service Broker
      Storage
      Security
  Security
  Server Objects
  Replication
  PolyBase
  Always On High Availability
  Management
  Integration Services Catalogs
  SQL Server Agent (Agent XPs disabled)
  XEvent Profiler
```

	_radiotap_flags_shortgi_	_wlan_fc_type_	_frame_time_delta_displayed_	_wlan_fc_version_	_wlan_mgt_fixed_capabilities_cfpoll_ap_	_wlan_mgt_fixed_capabilities_privacy_	_radiotap_rxflags_badplcp_	_wlan_mgt_tim_dtim
1	0	2	0.000856	0	?	?	0	?
2	0	2	0.000751	0	?	?	0	?
3	0	1	6E-06	0	?	?	0	?
4	0	2	0.000479	0	?	?	0	?
5	0	2	0.000173	0	?	?	0	?
6	0	1	6E-06	0	?	?	0	?
7	0	2	0.001859	0	?	?	0	?
8	0	0	0.00223	0	0x0000	1	0	0
9	0	2	0.000395	0	?	?	0	?
10	0	2	0.00022	0	?	?	0	?
11	0	1	6E-06	0	?	?	0	?
12	0	2	0.000404	0	?	?	0	?
13	0	1	0.000806	0	?	?	0	?
14	0	1	0.001896	0	?	?	0	?
15	0	2	0.000263	0	?	?	0	?
16	0	1	4E-06	0	?	?	0	?
17	0	0	0.005065	0	0x0000	1	0	0
18	0	2	0.001834	0	?	?	0	?

Executing query... DESKTOP-6VGHGES (14.0 RTM) | DESKTOP-6VGHGES\pwebs ... | CIS660 | 00:00:12 | 0 rows

Properties – Current connection parameters

Aggregate Status	
Connection failures	
Elapsed time	
Finish time	
Name	DESKTOP-6VGHGES
Rows returned	0
Start time	
State	Open
Connection	
Connection name	DESKTOP-6VGHGES (DE
Connection Details	
Connection elapse	
Connection encryp	Not encrypted
Connection finish t	
Connection rows re	0
Connection start ti	
Connection state	Open
Display name	DESKTOP-6VGHGES
Login name	DESKTOP-6VGHGES\pw
Server name	DESKTOP-6VGHGES
Server version	14.0.2002
Session Tracing ID	
SPID	55

Name
The name of the connection.

Ready Ln 1 Col 19 Ch 19 INS

7:55 PM 12/6/2018

Project Goals

Our primary goal for the project was to attain the highest accuracy possible when creating a model that is capable of classifying the 4 attack types we chose. One of the goals that supported this main objective was determining a way to avoid the curse of dimensionality. As our dataset has millions of tuples and over 100 columns, it is easy to become overwhelmed because of its size. The details about how we were able to achieve this task is discussed in greater detail in our preprocessing and data cleaning section.

Attack Types Used

In the reduced version of the AWID dataset, there is class variable with the values *normal*, *arp*, *cafe_latte*, *amok*, *deauthentication*, *authentication_request*, *beacon*, *evil_twin*, *fragmentation*, and *probe_response*. The type *normal* represents the packets that are not an attack. The other types listed represent an attack. For our analysis, we chose to use the k-nearest-neighbors classifier algorithm to predict if a packet was of the type: *arp*, *deauthentication*, *amok* or *authentication_request*.

A brief description of each of the four attack types is as follows. An Address Resolution Protocol (ARP) attack is one in which the malicious actor sends spoofed "address resolution protocol messages onto a local area network" (Wikipedia). In general, the aim of this is to "associate the attacker's MAC address with the IP address of another host, such as the default gateway, causing any traffic meant for that IP address to be sent the attacker instead" (Wikipedia). A *deauthentication attack* is a type of Denial of Service (DoS) in which the attacker uses unprotected deauthentication packets to spoof an entity. The attacker monitors traffic on a network to discover MAC addresses associated with specific clients. Then, a deauthentication message is sent to the access point on behalf of those particular MAC addresses to force a client off the network. Our third attack type is an *authentication request* attack. This is a flooding attack in which the attacker tries to exhaust the access point's resources by overflowing their client association table. The last type of attack is an *amok attack*. Though information on this particular form of cyberattack is limited, we determined that it is another flooding attack on a server, similar to the authentication request.

Major Tasks

During our project, we had three major tasks to accomplish: preprocessing & data cleaning, feature selection, and classification.

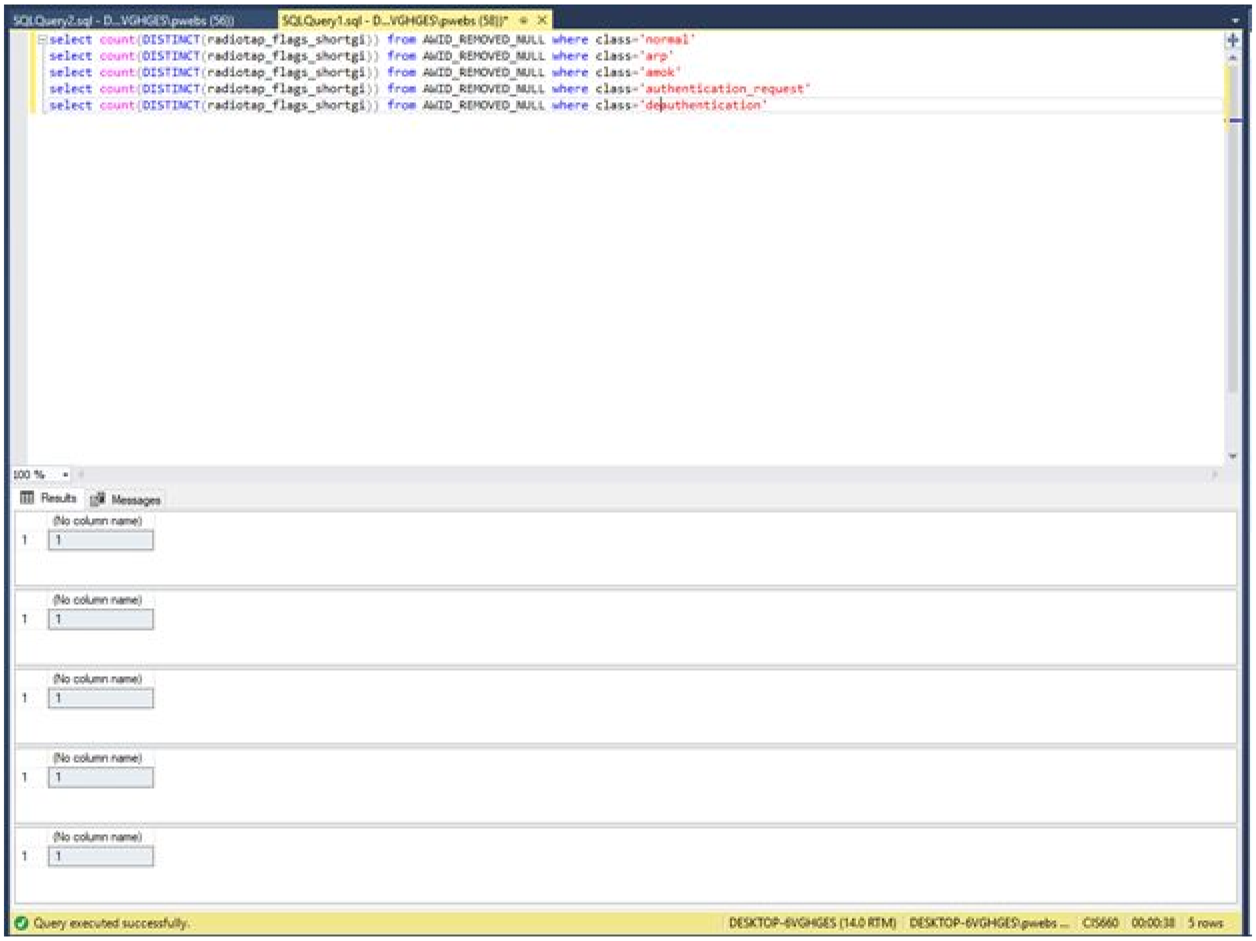
Preprocessing & Data Cleaning

As previously mentioned, we used Python 3.7.1 to perform the majority of our preprocessing and data cleaning on our dataset. To do this, we first downloaded the reduced dataset from the dataset’s host. Then, by using the *read_csv* method from the *pandas* library, we were able to read in a subset of columns that we wanted to work with for our analysis. These 19 desired columns were the following: 2, 5, 45, 62, 64, 65, 68, 71, 74, 75, 88, 91, 92, 105, 106, 110, 116, 120, and 154. After reading in these specific columns, we iterated over a text file containing the attribute names for the columns in our dataset. If the index of our desired columns existed in that list of attribute names, we added it to a list. Then, we were easily able to set the column headers for our minimized dataset.

After we had successfully removed unwanted columns from our larger dataset and gave each column a name, we were ready to proceed with the rest of our preprocessing. To start this process, we replaced the question marks in the dataset with a *NumPy* NaN value. We performed this operation to allow us to use other methods provided in the *pandas* library for removing missing values from our dataset. After replacing these values, we iterated over the columns in our dataset and removed a column if over 60% of its values were NaN (Not A Number). If a large majority of a column in the dataset was NaN, then it was not going to be useful for our later analysis. By performing this step, we removed 7 columns from our already minimized dataset. Then we removed columns that had zero or one unique values. A column had zero unique values if it there were no values provided, but was not filled in with a NaN value. In contrast, a column would have one unique value if it had the same value for every tuple. In both of these situations, this data would not contribute to our future analysis because there was nothing to help distinguish a tuple with a normal class value or one of our four attack types. As a final step in preprocessing and cleaning our dataset, we removed the rows that had at least one NaN value left in it. By doing this, we removed 1972 rows from the dataset. In hindsight, more caution should have been exercised when doing this as these tuples with a NaN value could have helped us further identify anomalies in the dataset. Last, after performing all these steps, we exported the reduced dataset as a CSV file for later use in our analysis. All of the code for performing the operations described in this section are shown in *data_preprocessing_cleaning.py*.

Feature Selection

To select which columns we would use with the classifier we first removed all columns except the columns listed on the course webpage. We then went through them by hand looking for attacks that had null values (?'s in our case) and non-null values for the normal type or vise-versa. None were found so we went on to the next step where we removed many null valued columns and rows (see the preprocessing/data cleaning section). Next we used the following SQL query: *Select Count(Distinct(<COLUMN_NAME>)) from <AWID_DATABASE> where class='<CLASSTYPE>'* where <CLASSTYPE> equaled the normal and all attack classes we used and <COLUMN_NAME> was varied to equal the remaining columns in our data. Before deciding that a column wasn't going to be removed we looked at the data further to make sure that even if all classes had the same number of distinct values that they weren't different distinct values for the attack and the normal type. Below are the results of these queries and our analysis of them.

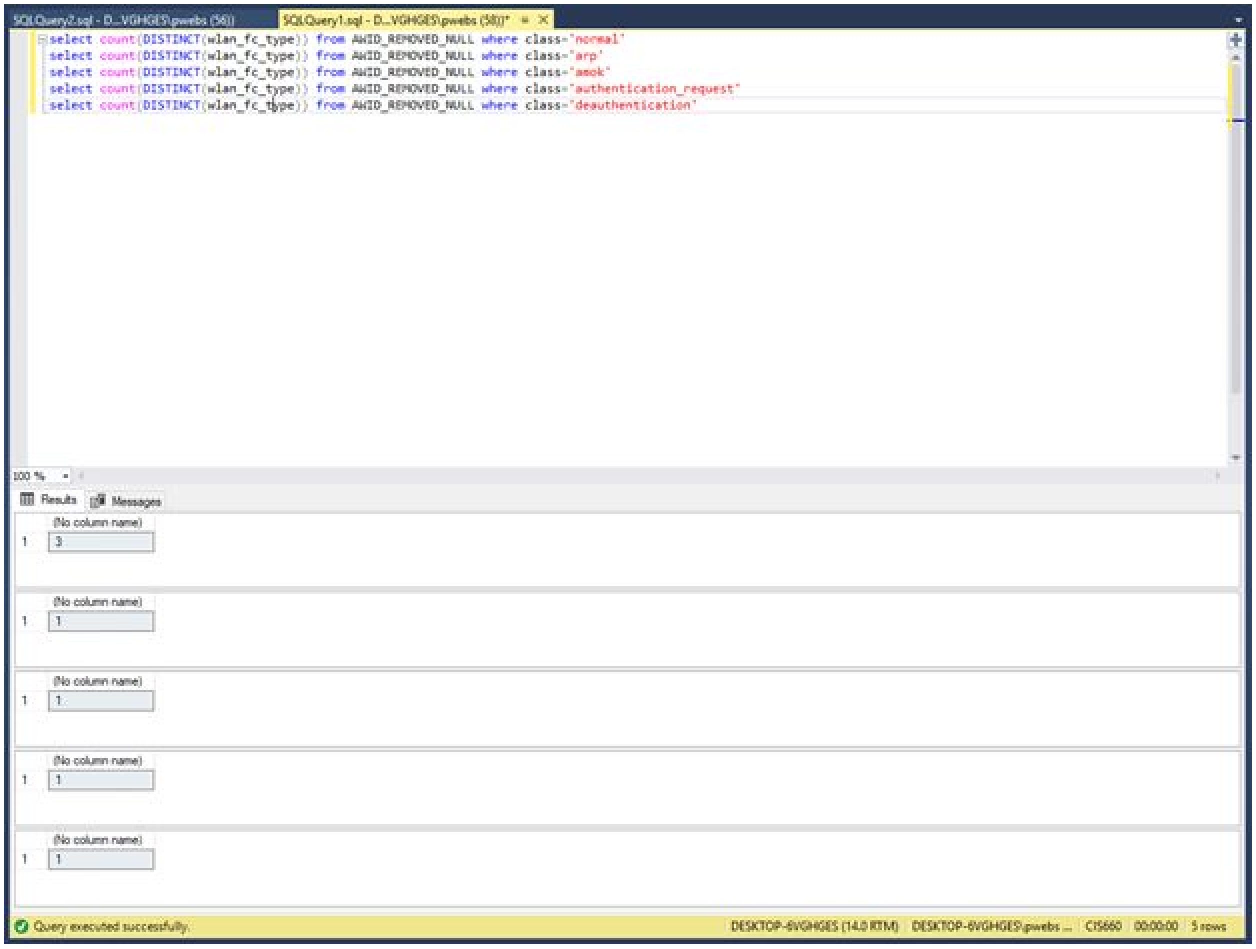

SQLQuery2.sql - D...VGHGES\pwebs (56))
SQLQuery1.sql - D...VGHGES\pwebs (58))*
100 %
Results
Messages

```
select count(DISTINCT(radiotap_flags_shortgi)) from AWID_REMOVED_NULL where class='normal'
select count(DISTINCT(radiotap_flags_shortgi)) from AWID_REMOVED_NULL where class='arp'
select count(DISTINCT(radiotap_flags_shortgi)) from AWID_REMOVED_NULL where class='amok'
select count(DISTINCT(radiotap_flags_shortgi)) from AWID_REMOVED_NULL where class='authentication_request'
select count(DISTINCT(radiotap_flags_shortgi)) from AWID_REMOVED_NULL where class='deauthentication'
```

```
   (No column name)
1  1

   (No column name)
1  1

   (No column name)
1  1

   (No column name)
1  1

   (No column name)
1  1
```

```
Query executed successfully.   DESKTOP-6VGHGES (14.0 RTM) | DESKTOP-6VGHGES\pwebs ... | CIS660 | 00:00:38 | 5 rows
```

We didn’t choose this column (*radiotap_flags_shortgi*) because it had no variation.

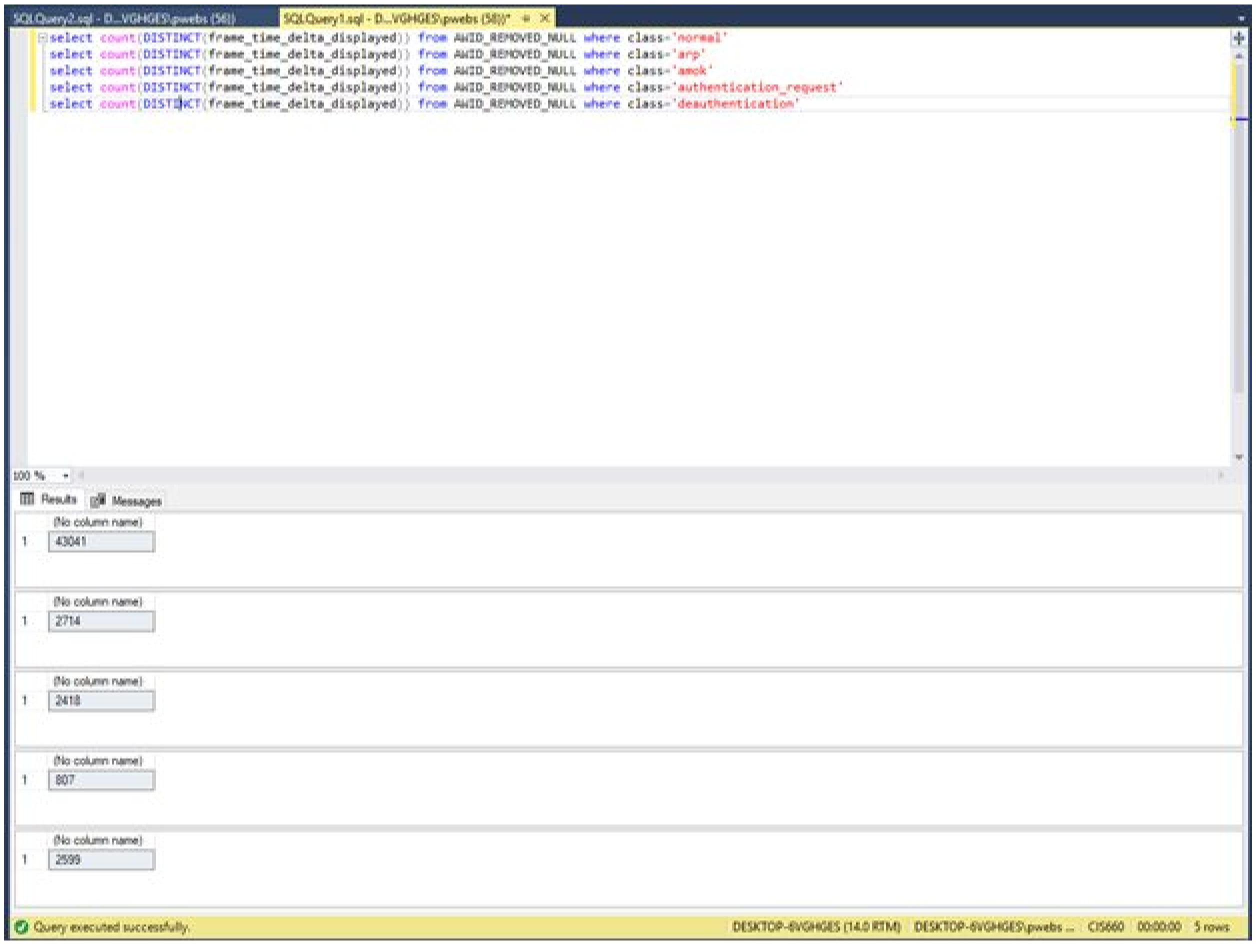


```
select count(DISTINCT(wlan_fc_type)) from AWID_REMOVED_NULL where class='normal'
select count(DISTINCT(wlan_fc_type)) from AWID_REMOVED_NULL where class='arp'
select count(DISTINCT(wlan_fc_type)) from AWID_REMOVED_NULL where class='amok'
select count(DISTINCT(wlan_fc_type)) from AWID_REMOVED_NULL where class='authentication_request'
select count(DISTINCT(wlan_fc_type)) from AWID_REMOVED_NULL where class='deauthentication'
```

```
   (No column name)
1  3

   (No column name)
1  1

   (No column name)
1  1

   (No column name)
1  1

   (No column name)
1  1
```

```
Query executed successfully.    DESKTOP-6VGHGES (14.0 RTM) | DESKTOP-6VGHGES\pwebs ... | CIS660 | 00:00:00 | 5 rows
```

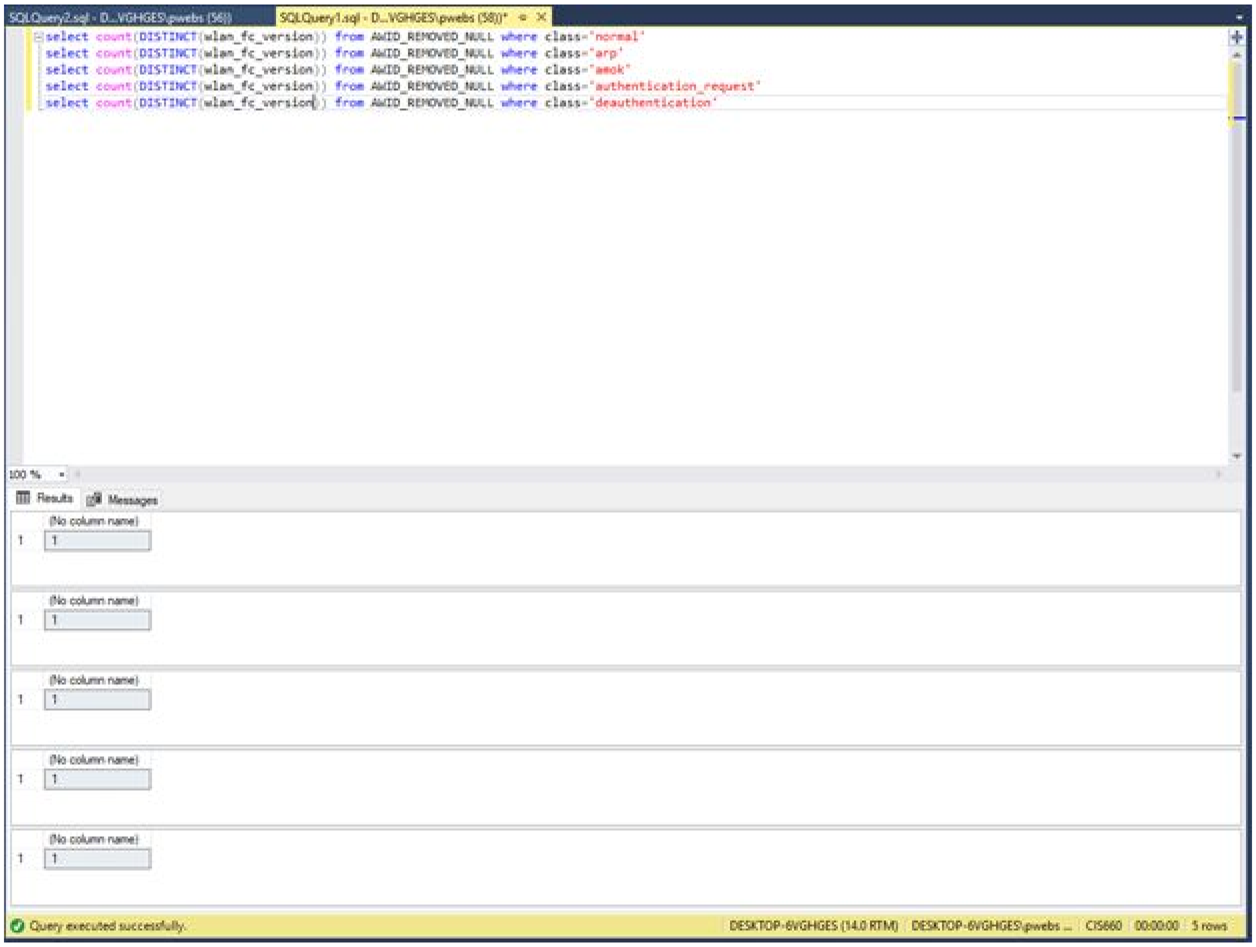
We chose this column (*wlan_fc_type*) because after looking at it further it showed that the *normal* class type varied highly among 3 values while the attack types all had the same values.



```
select count(DISTINCT(frame_time_delta_displayed)) from AWID_REMOVED_NULL where class='normal'
select count(DISTINCT(frame_time_delta_displayed)) from AWID_REMOVED_NULL where class='arp'
select count(DISTINCT(frame_time_delta_displayed)) from AWID_REMOVED_NULL where class='amok'
select count(DISTINCT(frame_time_delta_displayed)) from AWID_REMOVED_NULL where class='authentication_request'
select count(DISTINCT(frame_time_delta_displayed)) from AWID_REMOVED_NULL where class='deauthentication'
```

```
  (No column name)
1 43041

  (No column name)
1 2714

  (No column name)
1 2418

  (No column name)
1 807

  (No column name)
1 2599

Query executed successfully.   DESKTOP-6VGHGES (14.0 RTM) | DESKTOP-6VGHGES\pwebs ... | CIS660 | 00:00:00 | 5 rows
```

We chose this variable (frame_time_delta_displayed) because upon looking further into it we saw that for the attack types all the tuples had very close values while the *normal* type was more spread out.

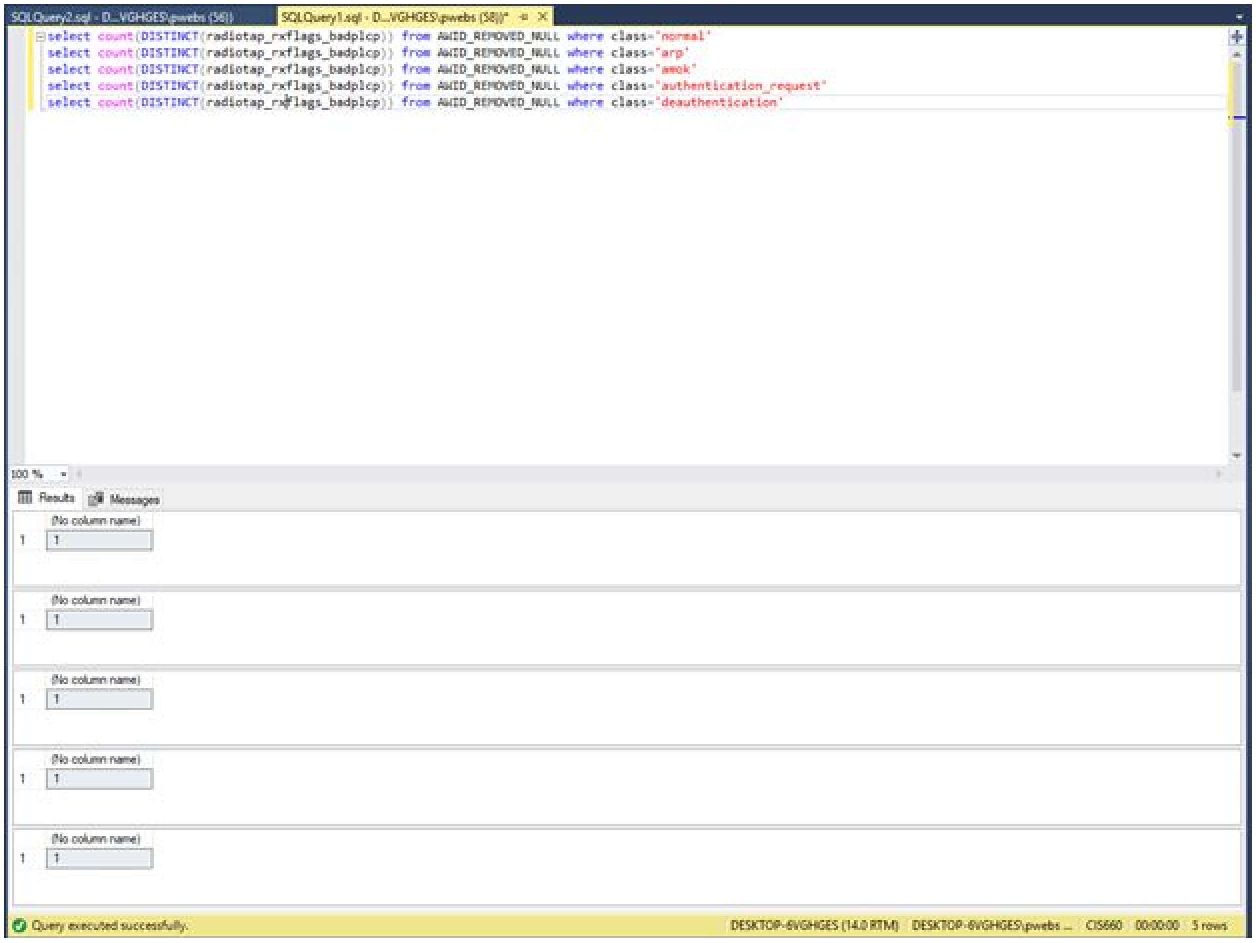

SQLQuery2.sql - D...VGHGES\pwebs (56)) SQLQuery1.sql - D...VGHGES\pwebs (58))*

```
select count(DISTINCT(wlan_fc_version)) from AWID_REMOVED_NULL where class='normal'
select count(DISTINCT(wlan_fc_version)) from AWID_REMOVED_NULL where class='arp'
select count(DISTINCT(wlan_fc_version)) from AWID_REMOVED_NULL where class='amok'
select count(DISTINCT(wlan_fc_version)) from AWID_REMOVED_NULL where class='authentication_request'
select count(DISTINCT(wlan_fc_version)) from AWID_REMOVED_NULL where class='deauthentication'
```

```
   (No column name)
1  1

   (No column name)
1  1

   (No column name)
1  1

   (No column name)
1  1

   (No column name)
1  1
```

Query executed successfully. DESKTOP-6VGHGES (14.0 RTM) DESKTOP-6VGHGES\pwebs ... CIS660 00:00:00 5 rows

We didn’t choose this variable (*wlan_fc_version*) because of lack of variation.



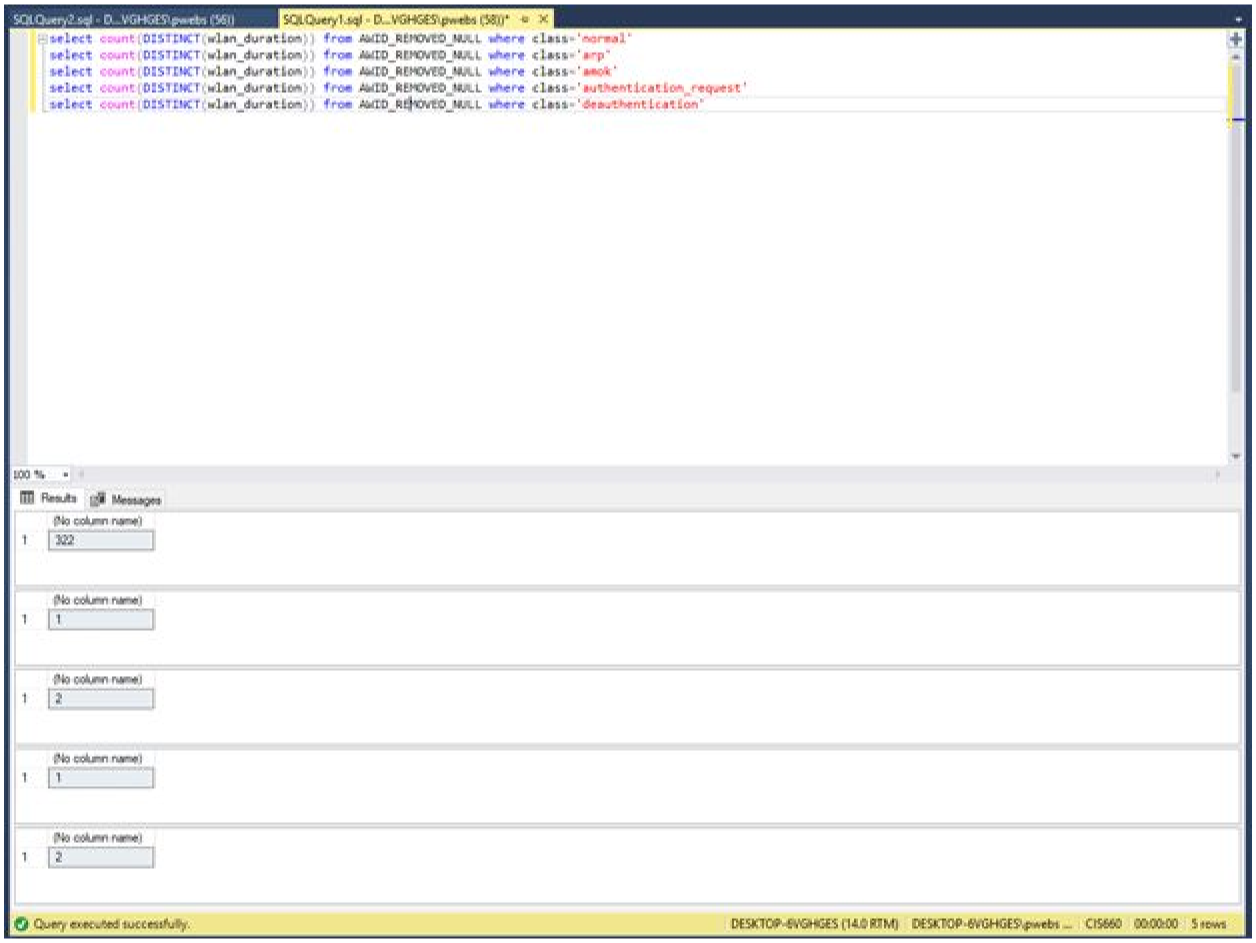
SQLQuery2.sql - D...VGHGES\pwebs (56))   SQLQuery1.sql - D...VGHGES\pwebs (58))*

```sql
select count(DISTINCT(radiotap_rxflags_badplcp)) from AWID_REMOVED_NULL where class='normal'
select count(DISTINCT(radiotap_rxflags_badplcp)) from AWID_REMOVED_NULL where class='arp'
select count(DISTINCT(radiotap_rxflags_badplcp)) from AWID_REMOVED_NULL where class='amok'
select count(DISTINCT(radiotap_rxflags_badplcp)) from AWID_REMOVED_NULL where class='authentication_request'
select count(DISTINCT(radiotap_rxflags_badplcp)) from AWID_REMOVED_NULL where class='deauthentication'
```

```
    (No column name)
1   1

    (No column name)
1   1

    (No column name)
1   1

    (No column name)
1   1

    (No column name)
1   1
```

Query executed successfully.   DESKTOP-6VGHGES (14.0 RTM)   DESKTOP-6VGHGES\pwebs ...   CIS660   00:00:00   5 rows

We didn't choose this variable (*radiotap_rxflags_badplcp*) because of lack of variation.

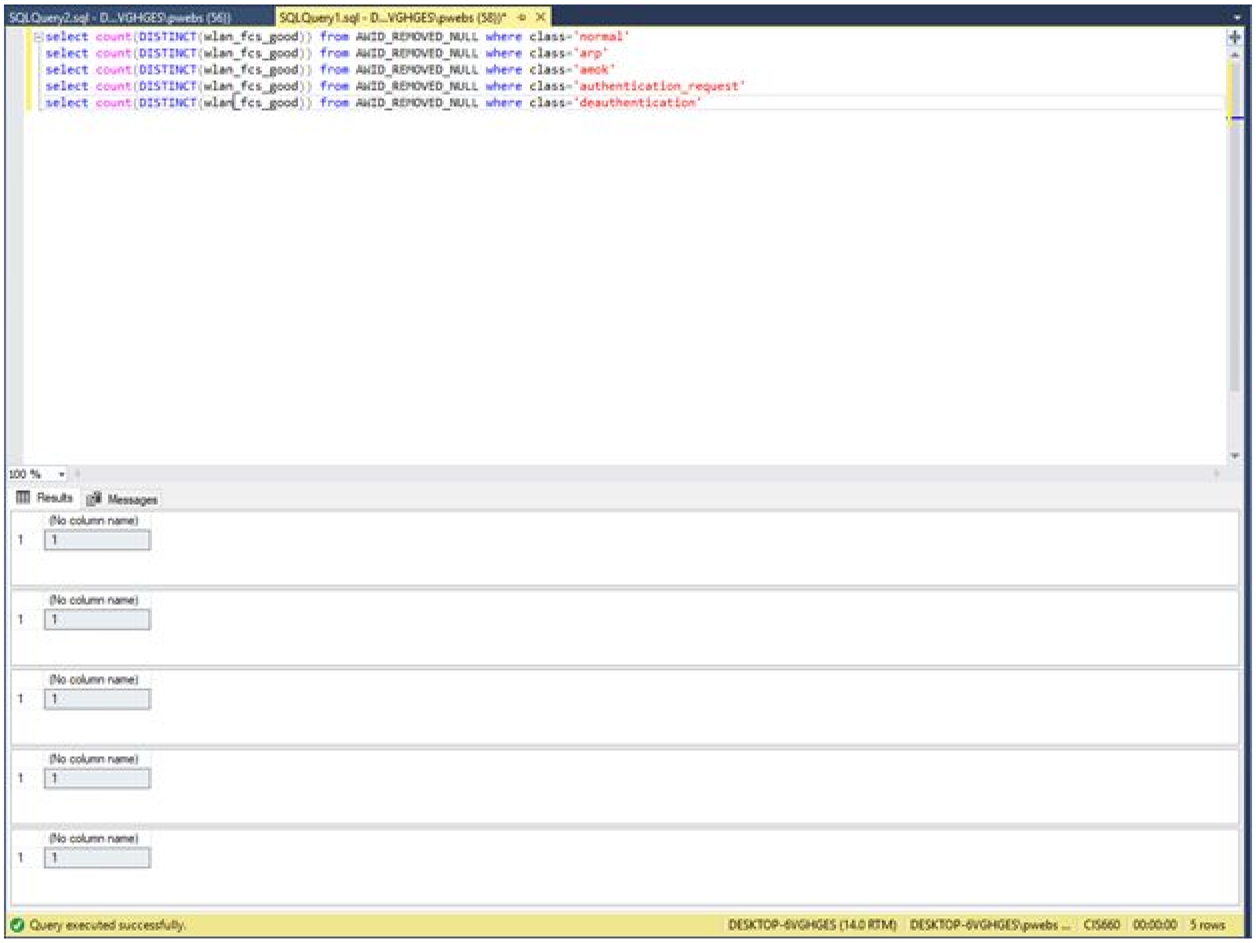


```
select count(DISTINCT(wlan_duration)) from AWID_REMOVED_NULL where class='normal'
select count(DISTINCT(wlan_duration)) from AWID_REMOVED_NULL where class='arp'
select count(DISTINCT(wlan_duration)) from AWID_REMOVED_NULL where class='amok'
select count(DISTINCT(wlan_duration)) from AWID_REMOVED_NULL where class='authentication_request'
select count(DISTINCT(wlan_duration)) from AWID_REMOVED_NULL where class='deauthentication'
```

```
   (No column name)
1  322

   (No column name)
1  1

   (No column name)
1  2

   (No column name)
1  1

   (No column name)
1  2
```

```
Query executed successfully.    DESKTOP-6VGHGES (14.0 RTM)  DESKTOP-6VGHGES\pwebs ...  CIS660  00:00:00  5 rows
```

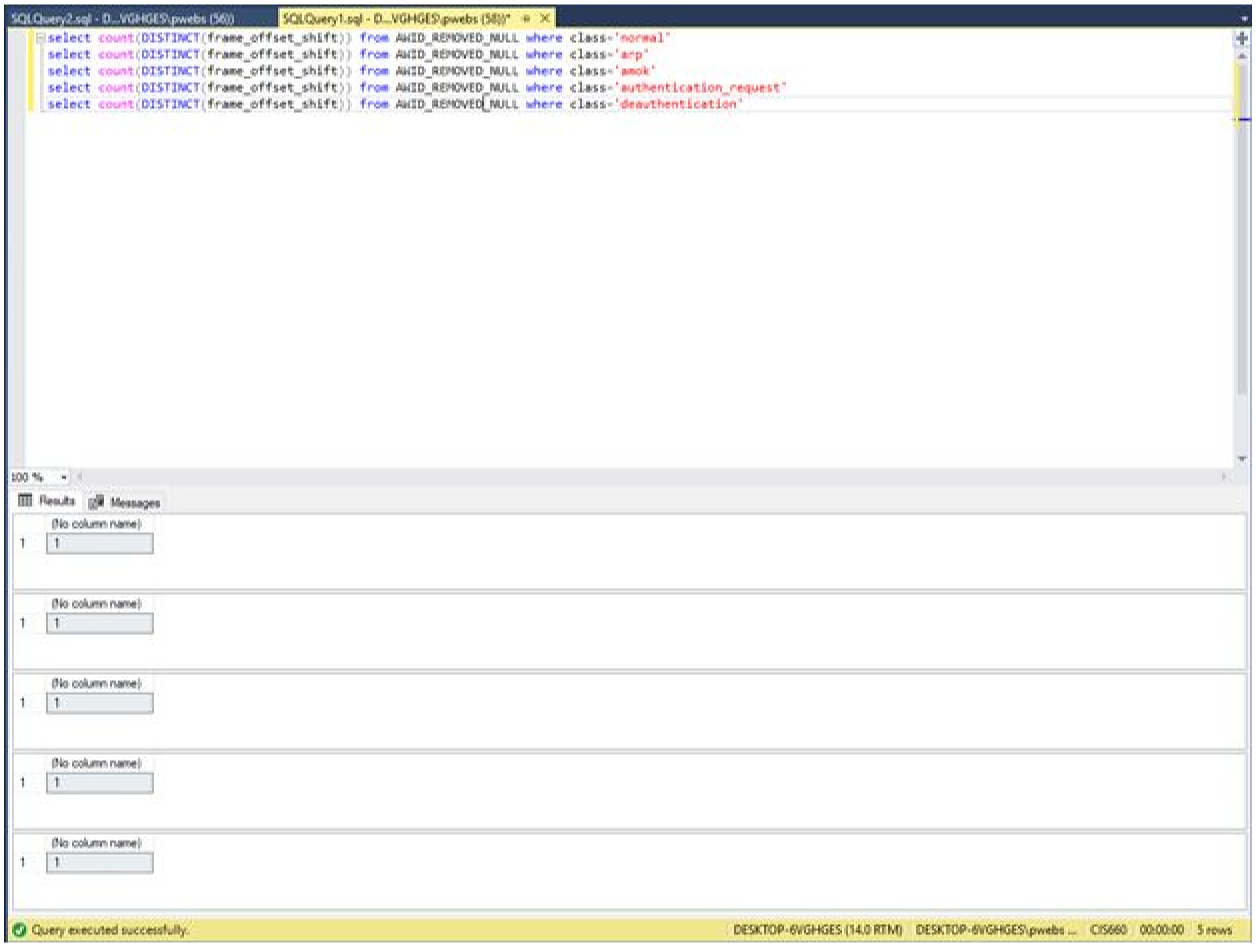
We chose this variable (*wlan_duration*) because there is a lot of variation for type *normal* and barely any for those whose class value is one of the four attack types.



```
SQLQuery2.sql - D...VGHGES\pwebs (56))    SQLQuery1.sql - D...VGHGES\pwebs (58))*

select count(DISTINCT(wlan_fcs_good)) from AWID_REMOVED_NULL where class='normal'
select count(DISTINCT(wlan_fcs_good)) from AWID_REMOVED_NULL where class='arp'
select count(DISTINCT(wlan_fcs_good)) from AWID_REMOVED_NULL where class='amok'
select count(DISTINCT(wlan_fcs_good)) from AWID_REMOVED_NULL where class='authentication_request'
select count(DISTINCT(wlan_fcs_good)) from AWID_REMOVED_NULL where class='deauthentication'
```

```
   (No column name)
1  1

   (No column name)
1  1

   (No column name)
1  1

   (No column name)
1  1

   (No column name)
1  1

Query executed successfully.    DESKTOP-6VGHGES (14.0 RTM)   DESKTOP-6VGHGES\pwebs ...   CIS660   00:00:00   5 rows
```

We chose not to use this variable (*wlan_fcs_good*) because of lack of variation.

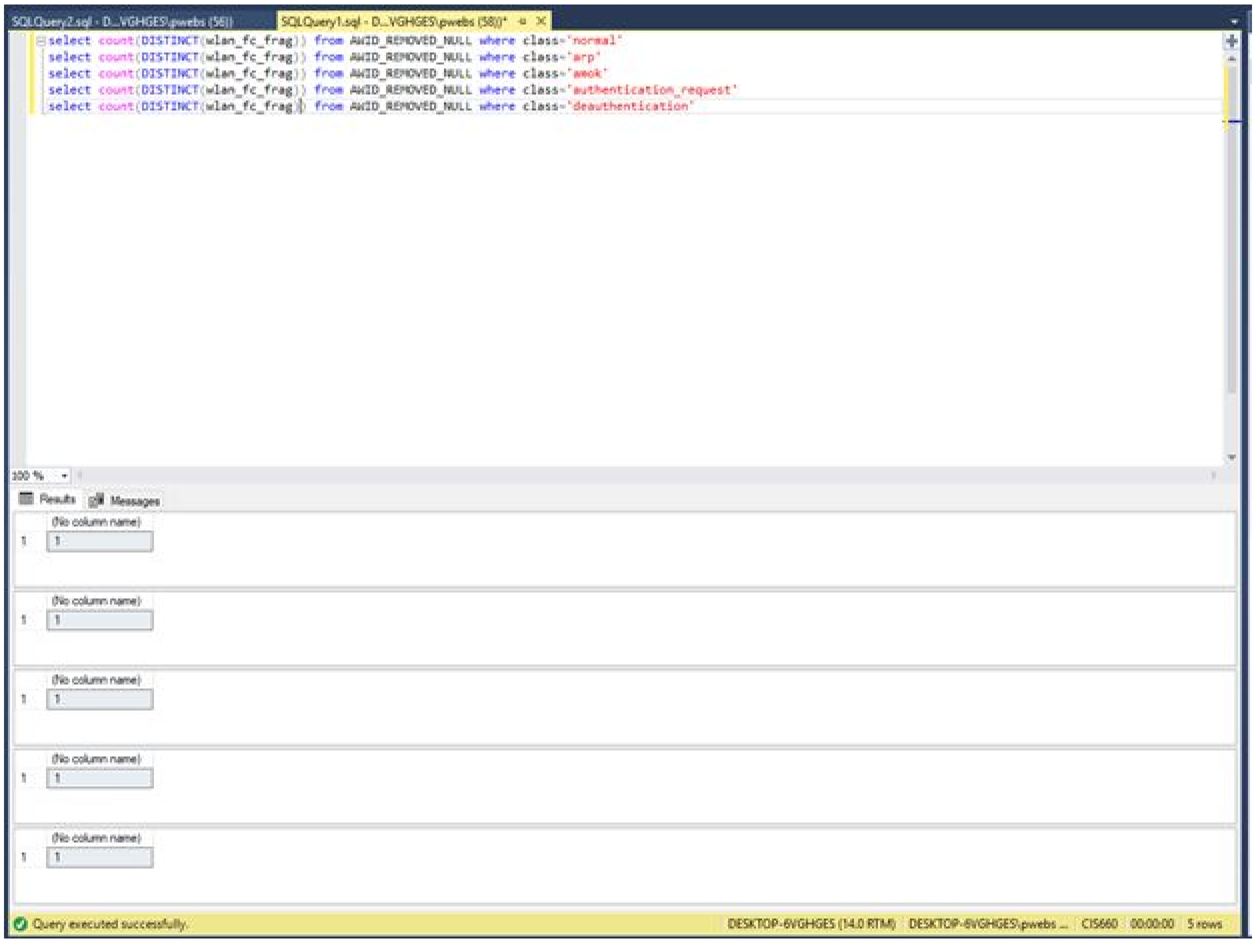


```
select count(DISTINCT(frame_offset_shift)) from AWID_REMOVED_NULL where class='normal'
select count(DISTINCT(frame_offset_shift)) from AWID_REMOVED_NULL where class='arp'
select count(DISTINCT(frame_offset_shift)) from AWID_REMOVED_NULL where class='amok'
select count(DISTINCT(frame_offset_shift)) from AWID_REMOVED_NULL where class='authentication_request'
select count(DISTINCT(frame_offset_shift)) from AWID_REMOVED_NULL where class='deauthentication'
```

```
   (No column name)
1  1

   (No column name)
1  1

   (No column name)
1  1

   (No column name)
1  1

   (No column name)
1  1
```

```
Query executed successfully.    DESKTOP-6VGHGES (14.0 RTM)  DESKTOP-6VGHGES\pwebs ...  CIS660  00:00:00  5 rows
```

We chose to not use this variable (*frame_offset_shift*) because of lack of variation.

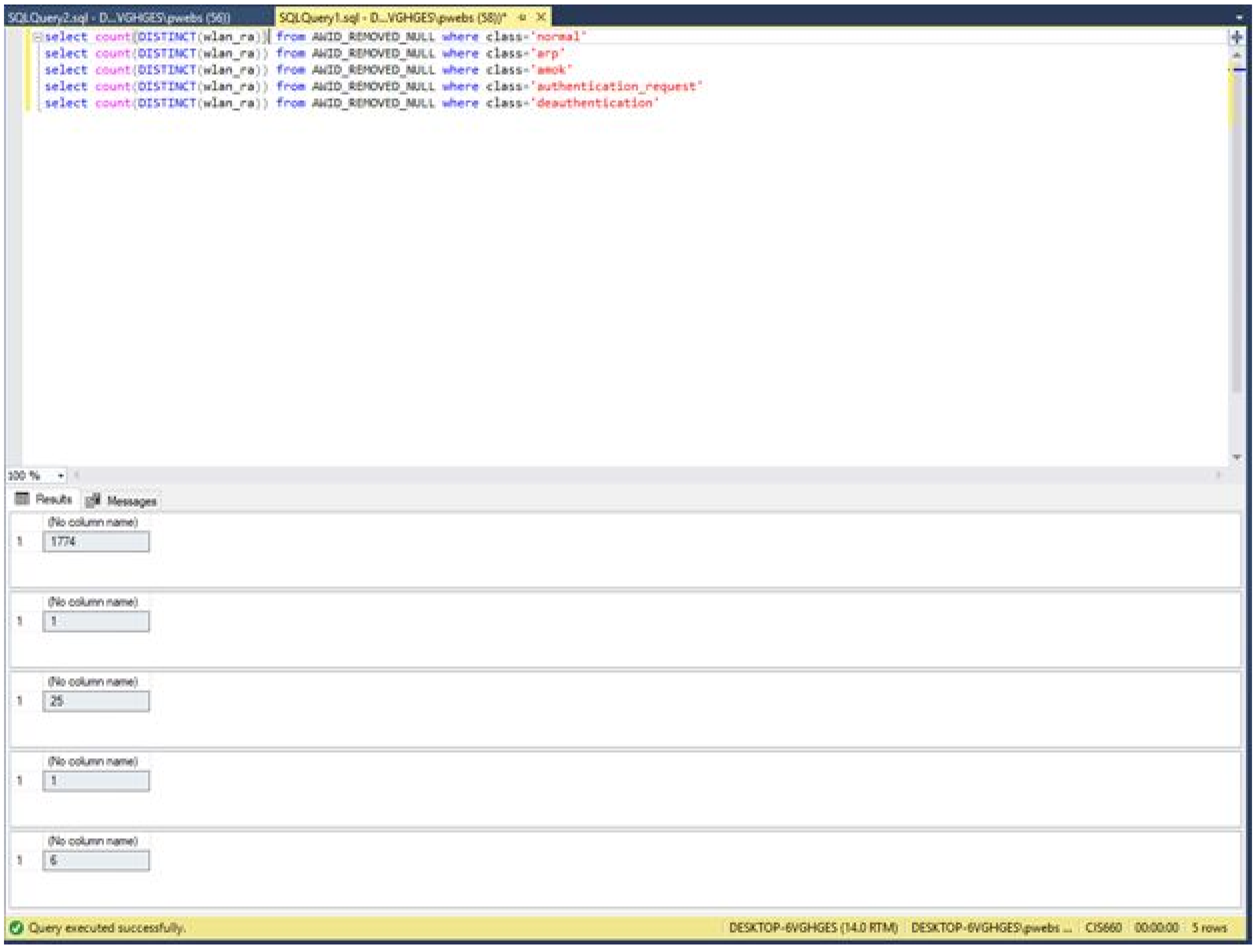


```
select count(DISTINCT(wlan_fc_frag)) from AWID_REMOVED_NULL where class='normal'
select count(DISTINCT(wlan_fc_frag)) from AWID_REMOVED_NULL where class='arp'
select count(DISTINCT(wlan_fc_frag)) from AWID_REMOVED_NULL where class='amok'
select count(DISTINCT(wlan_fc_frag)) from AWID_REMOVED_NULL where class='authentication_request'
select count(DISTINCT(wlan_fc_frag)) from AWID_REMOVED_NULL where class='deauthentication'
```

```
   (No column name)
1  1

   (No column name)
1  1

   (No column name)
1  1

   (No column name)
1  1

   (No column name)
1  1
```

```
Query executed successfully.    DESKTOP-6VGHGES (14.0 RTM)  DESKTOP-6VGHGES\pwebs ...  CIS660  00:00:00  5 rows
```

We chose to not use this variable (*wlan_fc_frag*) because of lack of variation.

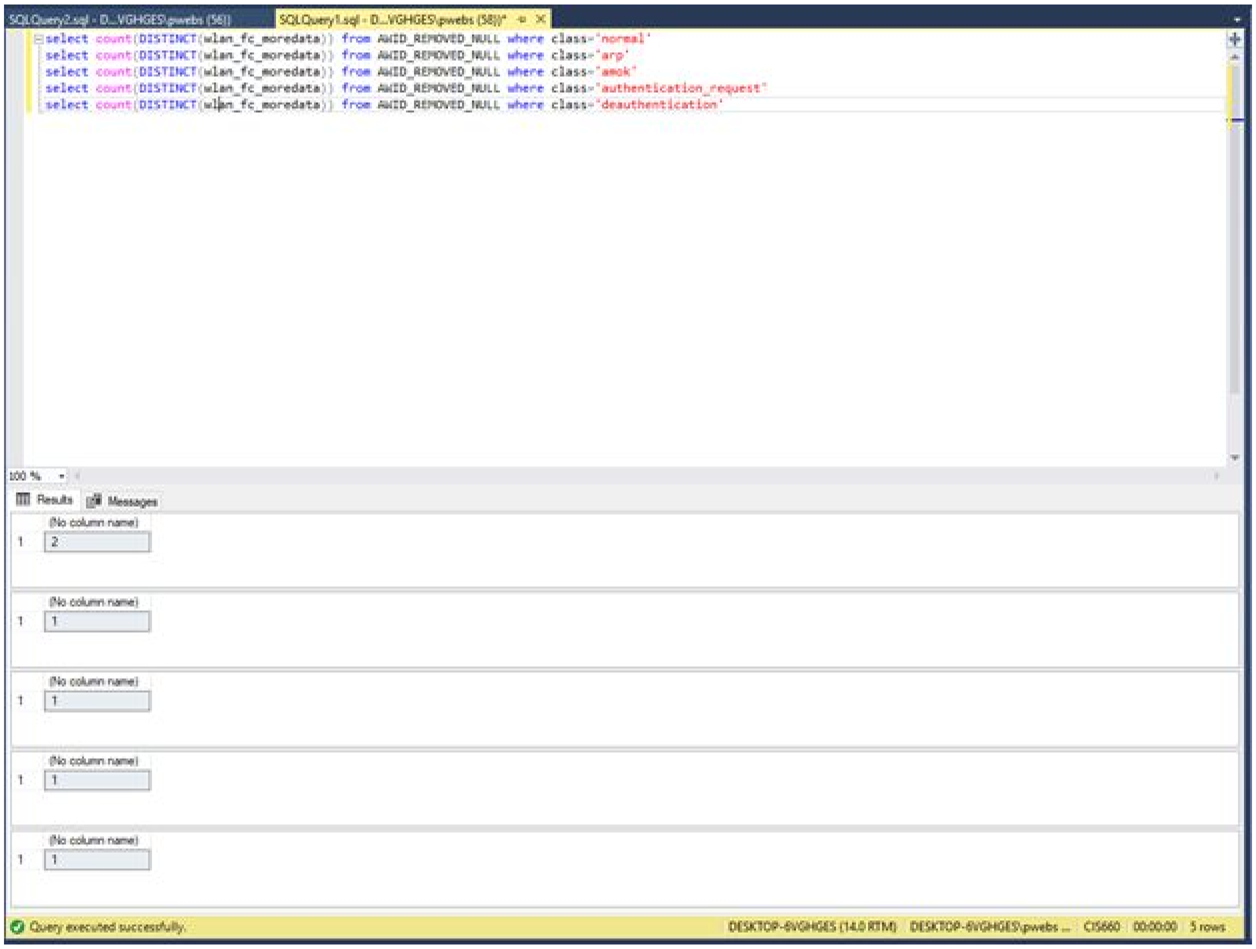


```
select count(DISTINCT(wlan_ra)) from AWID_REMOVED_NULL where class='normal'
select count(DISTINCT(wlan_ra)) from AWID_REMOVED_NULL where class='arp'
select count(DISTINCT(wlan_ra)) from AWID_REMOVED_NULL where class='amok'
select count(DISTINCT(wlan_ra)) from AWID_REMOVED_NULL where class='authentication_request'
select count(DISTINCT(wlan_ra)) from AWID_REMOVED_NULL where class='deauthentication'
```

```
   (No column name)
1  1774

   (No column name)
1  1

   (No column name)
1  25

   (No column name)
1  1

   (No column name)
1  6
```

```
Query executed successfully.   DESKTOP-6VGHGES (14.0 RTM)   DESKTOP-6VGHGES\pwebs ...   CIS660   00:00:00   5 rows
```

After much debate, we decided to not use this variable (*wlan_ra*) even though it has the variation we are looking for because it is nominal and not ordinal and the rest of our variables are continuous. Since SMOTE only works for continuous variables we felt it would be best not to use it otherwise we would probably have to one-hot-encode all of our data and use a different classifier than K-Nearest-Neighbors and not use SMOTE.



```
select count(DISTINCT(wlan_fc_moredata)) from AWID_REMOVED_NULL where class='normal'
select count(DISTINCT(wlan_fc_moredata)) from AWID_REMOVED_NULL where class='arp'
select count(DISTINCT(wlan_fc_moredata)) from AWID_REMOVED_NULL where class='amok'
select count(DISTINCT(wlan_fc_moredata)) from AWID_REMOVED_NULL where class='authentication_request'
select count(DISTINCT(wlan_fc_moredata)) from AWID_REMOVED_NULL where class='deauthentication'
```

```
   (No column name)
1  2

   (No column name)
1  1

   (No column name)
1  1

   (No column name)
1  1

   (No column name)
1  1
```

```
Query executed successfully.    DESKTOP-6VGHGES (14.0 RTM)  DESKTOP-6VGHGES\pwebs ...  CIS660  00:00:00  5 rows
```

We didn’t choose this attribute (*wlan_fc_moredata*) because even though it has the variation we were looking for 99.99% of the normal tuples had the same value as the attack types making it basically useless.

Classification

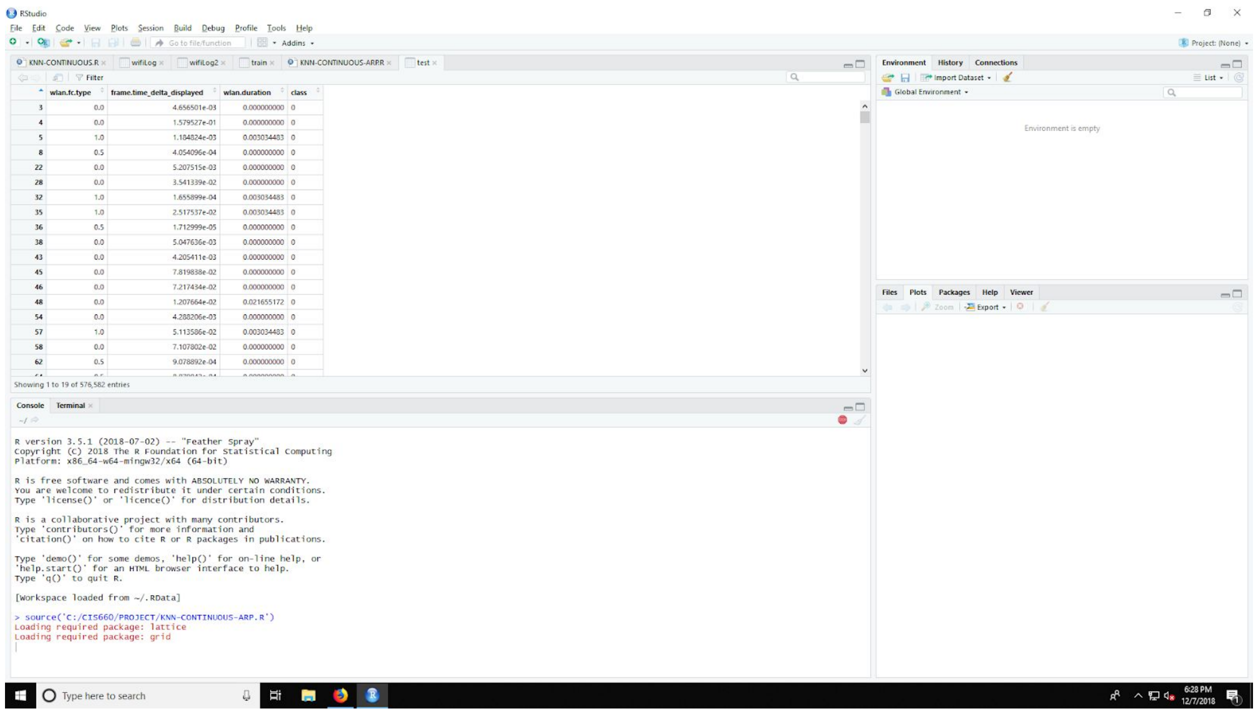
Our classification code is written in R. We chose this because it was already known by the group member that wrote the classification code.

To perform classification we decided early that we would use SMOTE to create synthetic minority class tuples because the dataset is highly biased towards the normal class. We decided on using the K-nearest-neighbor classification algorithm later. We chose the k-nearest-neighbor algorithm because it works well with continuous data and every variable we used to classify was continuous.

To perform the classification we first min-max normalized all of our target attributes to [0,1]. Next, we removed all classes except the normal and the attack class that we were currently predicting. Then, we randomly divided the normalized dataset into 66.6% training and 33.3% test datasets. Next, we fed the training data into the function SMOTE() in R studio. Then, we fed the oversampled resulting training dataset from SMOTE and the test dataset into the function kNN() in R. From the kNN() function we received a vector containing a 0 or 1 depending upon if the attack class was predicted. We used different operands for the SMOTE() and kNN() functions depending upon if we wanted higher recall or higher accuracy.

We attempted cross-validation but due to an error we received (“too many ties”) we were not able to complete it.

Below is a screenshot of our test data for the attack type *arp*:

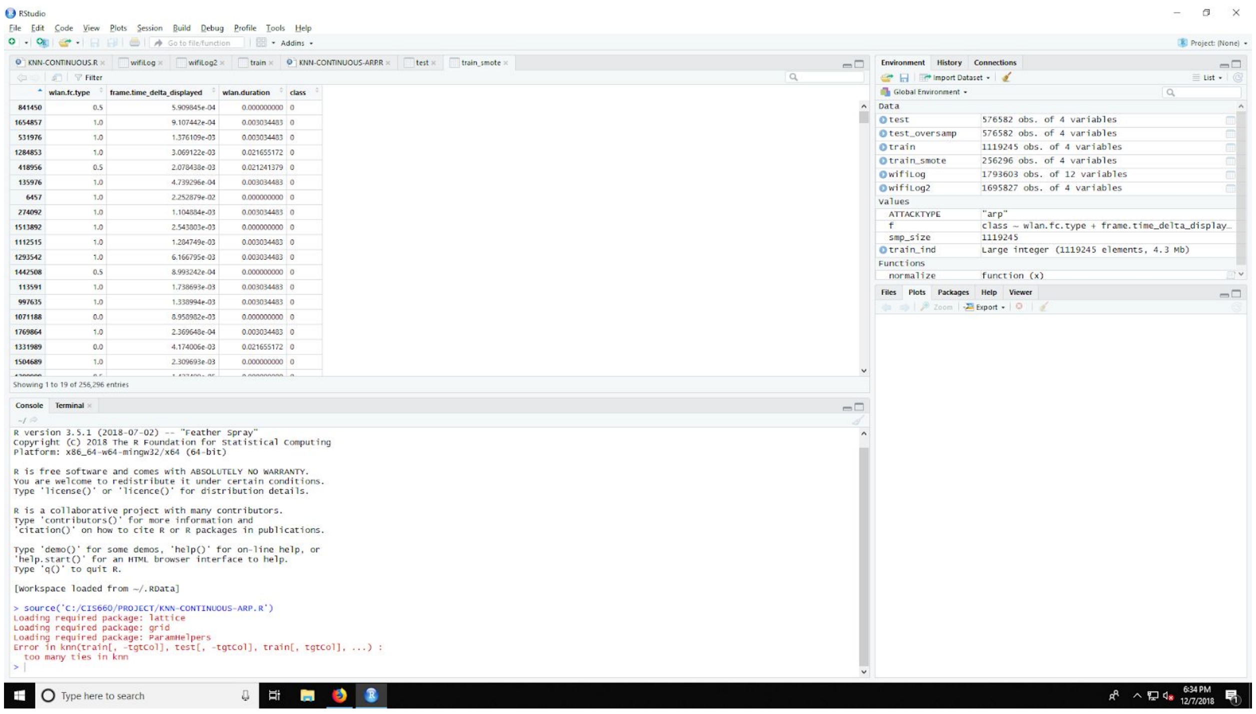


	wlan.fc.type	frame.time_delta_displayed	wlan.duration	class
3	0.0	4.656501e-03	0.000000000	0
4	0.0	1.579527e-01	0.000000000	0
5	1.0	1.184824e-03	0.003034483	0
8	0.5	4.054096e-04	0.000000000	0
22	0.0	5.207515e-03	0.000000000	0
28	0.0	3.541339e-02	0.000000000	0
32	1.0	1.655899e-04	0.003034483	0
35	1.0	2.517537e-02	0.003034483	0
36	0.5	1.712999e-05	0.000000000	0
38	0.0	5.047636e-03	0.000000000	0
43	0.0	4.205411e-03	0.000000000	0
45	0.0	7.819838e-02	0.000000000	0
46	0.0	7.217434e-02	0.000000000	0
48	0.0	1.207664e-02	0.021655172	0
54	0.0	4.288206e-03	0.000000000	0
57	1.0	5.113586e-02	0.003034483	0
58	0.0	7.107802e-02	0.000000000	0
62	0.5	9.078892e-04	0.000000000	0

Showing 1 to 19 of 576,582 entries

```
R version 3.5.1 (2018-07-02) -- "Feather Spray"
Copyright (C) 2018 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

[Workspace loaded from ~/.RData]

> source('C:/CIS660/PROJECT/KNN-CONTINUOUS-ARP.R')
Loading required package: lattice
Loading required package: grid
```

Below is a screenshot of our training data for the attack type *arp* after it had been run through SMOTE:



	wlan.fc.type	frame.time_delta_displayed	wlan.duration	class
841450	0.5	5.909845e-04	0.000000000	0
1654857	1.0	9.107442e-04	0.003034483	0
531976	1.0	1.376109e-03	0.003034483	0
1284853	1.0	3.069122e-03	0.021655172	0
418956	0.5	2.078438e-03	0.021241379	0
135976	1.0	4.739296e-04	0.003034483	0
6457	1.0	2.252879e-02	0.000000000	0
274092	1.0	1.104884e-03	0.003034483	0
1513892	1.0	2.543803e-03	0.000000000	0
1112515	1.0	1.284749e-03	0.003034483	0
1293542	1.0	6.166795e-03	0.003034483	0
1442508	0.5	8.993242e-04	0.000000000	0
113591	1.0	1.738693e-03	0.003034483	0
997635	1.0	1.338994e-03	0.003034483	0
1071188	0.0	8.958982e-03	0.000000000	0
1769864	1.0	2.369648e-04	0.003034483	0
1331989	0.0	4.174006e-03	0.021655172	0
1504689	1.0	2.309693e-03	0.000000000	0

Showing 1 to 19 of 256,296 entries

Data	
test	576582 obs. of 4 variables
test_oversamp	576582 obs. of 4 variables
train	1119245 obs. of 4 variables
train_smote	256296 obs. of 4 variables
wifiLog	1793603 obs. of 12 variables
wifiLog2	1695827 obs. of 4 variables
Values	
ATTACKTYPE	"arp"
f	class ~ wlan.fc.type + frame.time_delta_display…
smp_size	1119245
train_ind	Large integer (1119245 elements, 4.3 Mb)
Functions	
normalize	function (x)

```
R version 3.5.1 (2018-07-02) -- "Feather Spray"
Copyright (C) 2018 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

[Workspace loaded from ~/.RData]

> source('C:/CIS660/PROJECT/KNN-CONTINUOUS-ARP.R')
Loading required package: lattice
Loading required package: grid
Loading required package: ParamHelpers
Error in knn(train[, -tgtCol], test[, -tgtCol], train[, tgtCol], ...) :
  too many ties in knn
> 
```

Results

We performed multiple tests on each of the attack types, and recorded our top two, except for ARP. Our results for ARP were extremely good, which made it very challenging to further improve them. We either ran into errors or achieved similar results each time we attempted to modify the parameters. By tuning the parameters for SMOTE and the KNN classifier we were able to improve our accuracy and precision for each of the other attacks, but at the cost of recall. Several tables displaying the results and parameters used for each of the four attacks can be seen below.

ARP (Test 1) KNN Parameters

- Smote.k = 3
- knn.k = 5
- smote.perc.over = 150
- smote.perc.under = 90

ARP (Test 1) - Anomaly Detection Metrics

False Positives	1,731
True Positives	21,889
True Negatives	552,958
False Negatives	4

Accuracy	99.6990%
Error Rate	0.3009%
Sensitivity	92.6714%
Specificity	99.9992%
Precision	92.6714%
Recall	99.9817%

Amok (Test 1) KNN Parameters

- Smote.k = 3
- knn.k = 5
- smote.perc.over = 150
- smote.perc.under = 90

Amok (Test 1) - Anomaly Detection Metrics

False Positives	42,928
True Positives	10,275
True Negatives	511,451
False Negatives	562

Accuracy	92.3056%
Error Rate	7.6944%
Sensitivity	19.3128%
Specificity	99.8902%
Precision	19.3128%
Recall	94.8140%

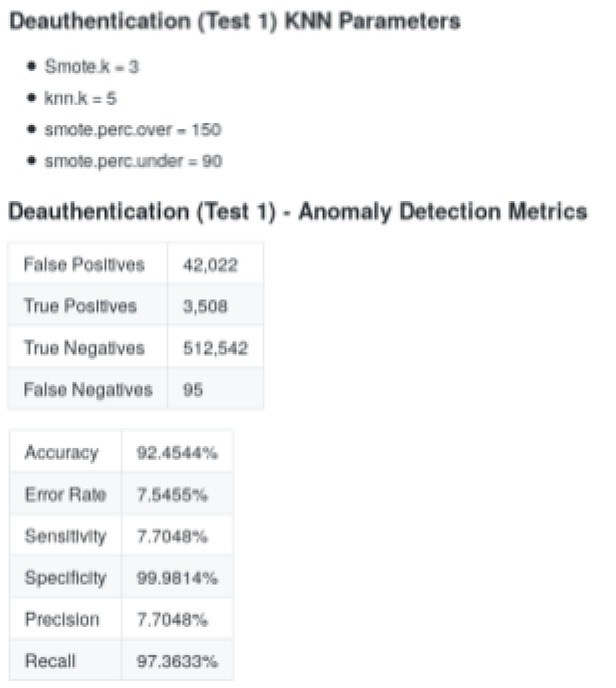
Amok (Test 2) KNN Parameters

- smote.k = 1
- knn.k = 1
- smote.perc.over = 120
- smote.perc.under = 200

Amok (Test 2) - Anomaly Detection Metrics

False Positives	24,473
True Positives	9,738
True Negatives	529,906
False Negatives	1099

Accuracy	95.4757%
Error Rate	4.5242%
Sensitivity	2.8464%
Specificity	99.7930%
Precision	28.4645%
Recall	89.8588%



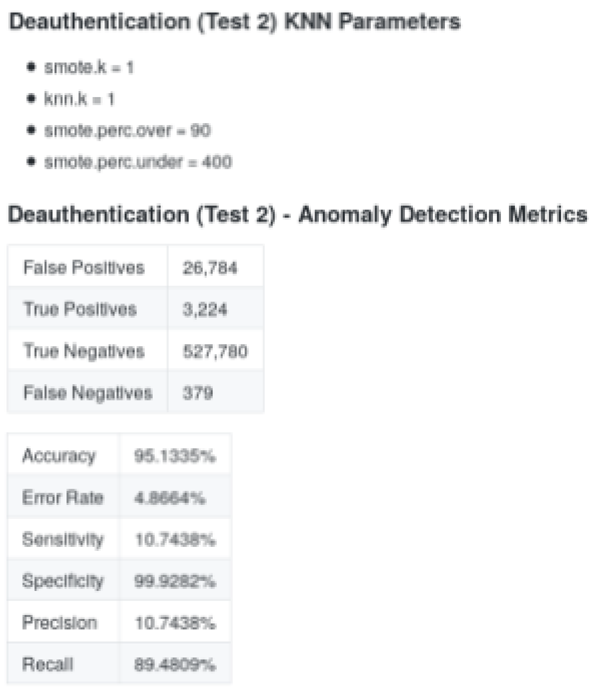
Deauthentication (Test 1) KNN Parameters

- Smote.k = 3
- knn.k = 5
- smote.perc.over = 150
- smote.perc.under = 90

Deauthentication (Test 1) - Anomaly Detection Metrics

False Positives	42,022
True Positives	3,508
True Negatives	512,542
False Negatives	95

Accuracy	92.4544%
Error Rate	7.5455%
Sensitivity	7.7048%
Specificity	99.9814%
Precision	7.7048%
Recall	97.3633%



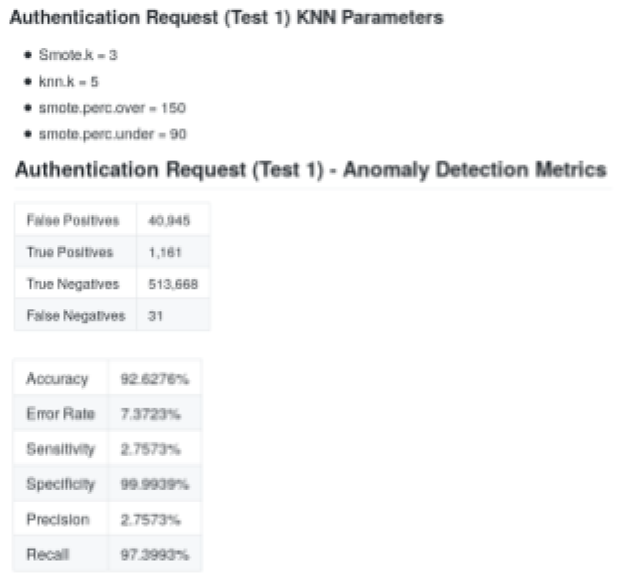
Deauthentication (Test 2) KNN Parameters

- smote.k = 1
- knn.k = 1
- smote.perc.over = 90
- smote.perc.under = 400

Deauthentication (Test 2) - Anomaly Detection Metrics

False Positives	26,784
True Positives	3,224
True Negatives	527,780
False Negatives	379

Accuracy	95.1335%
Error Rate	4.8664%
Sensitivity	10.7438%
Specificity	99.9282%
Precision	10.7438%
Recall	89.4809%



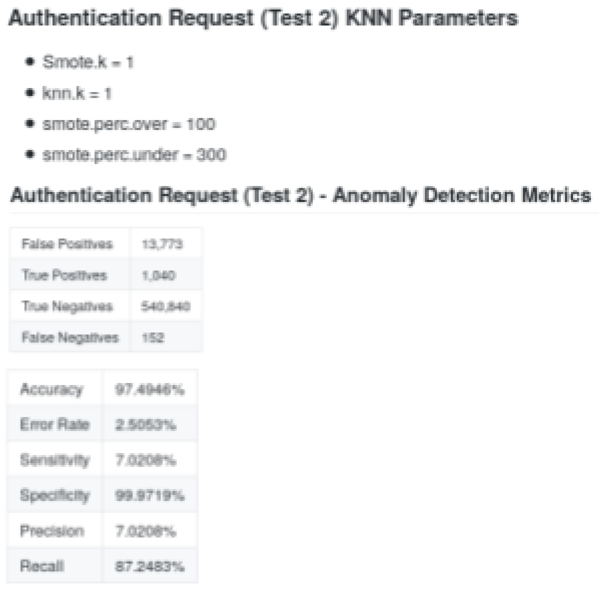
Authentication Request (Test 1) KNN Parameters

- Smote.k = 3
- knn.k = 5
- smote.perc.over = 150
- smote.perc.under = 90

Authentication Request (Test 1) - Anomaly Detection Metrics

False Positives	40,945
True Positives	1,161
True Negatives	513,668
False Negatives	31

Accuracy	92.6276%
Error Rate	7.3723%
Sensitivity	2.7573%
Specificity	99.9939%
Precision	2.7573%
Recall	97.3993%



Authentication Request (Test 2) KNN Parameters

- Smote.k = 1
- knn.k = 1
- smote.perc.over = 100
- smote.perc.under = 300

Authentication Request (Test 2) - Anomaly Detection Metrics

False Positives	13,773
True Positives	1,040
True Negatives	540,840
False Negatives	152

Accuracy	97.4946%
Error Rate	2.5053%
Sensitivity	7.0208%
Specificity	99.9719%
Precision	7.0208%
Recall	87.2483%

We are quite certain the results for ARP were significantly better than the other three attacks due to the fact that some of the variables for ARP had mutually exclusive values, while the same variables for the other attack type shared values with the normal type. For example for ARP the fc_type variable was always 0 and the wlan_duration variable for ARP was always the same while that was not the case for the other attacks.

Future Work

In a future version we would like to try using a classifier with the 3 variables we used plus the *wlan_ra* (MAC address) variable that we didn't use to see how it affects accuracy and recall for the 3 attacks that had lackluster results. To do this we would have to use plain oversampling instead of SMOTE, one-hot-encoding and probably use a different classifier such as neural network which is compatible with one-hot encoding. Additionally, we would like to test a more generalized model that simply determines if an attack occurred. By generalizing the model, the performance would most likely be improved.

Conclusion

In this project we used the k-nearest-neighbor classifier to predict if data from the AWID dataset was normal or an attack. Our results were mixed with either high accuracy and high recall, high accuracy and lower recall or high recall and lower accuracy. Most of our time was spent on deciding how to pre-process the data, deciding which features to use with the classifier and deciding which classifier and oversampling method to use. We learned a lot from the project but if we could do it again we would select more features to classify with.