Datahacks 2021

Team Wild Rifters (Justin Liang, Jonathan Lo, Eric Wang)

Step 1: Data cleaning / Pre-processing

In [1]:
```
# Import data cleaning packages
import pandas as pd
```

In [2]:
```
# Pandas read_csv function to import dataframes
bitcoin_train = pd.read_csv('Datasets/bitcoin_train.csv')
bitcoin_test = pd.read_csv('Datasets/bitcoin_test.csv')
```

In [3]:
```
# Check the first 5 rows of the bitcoin_train dataframe
bitcoin_train.head()
```

Out[3]:

	Unnamed: 0	address	year	day	length	weight	count	looped
0	0	1BpvJgUs7UprQu9z8fLsP7pFvFcCscHRCV	2011	287	2	0.250000	1	0
1	1	1EnSeTPjMxZm9X9iQDYmMUDoLQQ3ouDN6F	2015	77	0	1.000000	1	0
2	2	1mwkhYHeoqGBkVW84yFpYCSqRDt5TWSBQ	2011	164	52	0.000977	23	0
3	3	19XUCsxgpHZGXKLgVMpdoyZqcFdeM3pGeE	2014	86	144	0.000001	1555	1152
4	4	14Ef6MGSYLEbigo55CpPBGEGSGYwwB7xhY	2015	261	6	0.250000	1	0

In [4]:
```
# Check the details of the dataframe
bitcoin_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2333357 entries, 0 to 2333356
Data columns (total 11 columns):
 #   Column      Dtype  
---  ------      -----  
 0   Unnamed: 0  int64  
 1   address     object 
 2   year        int64  
 3   day         int64  
 4   length      int64  
 5   weight      float64
 6   count       int64  
 7   looped      int64  
 8   neighbors   int64  
 9   income      float64
 10  label       object 
dtypes: float64(2), int64(7), object(2)
memory usage: 195.8+ MB
```

Using the info function, we can get a general idea of the dataframe dtypes as well as the total number of columns. All the dtypes look correct with no conversions required, but the column Unnamed: 0 appears to be the row index carried over from the csv file. Since the pandas dataframe already has it's own built in index, we can drop that column.

Edit: In order to match with the bitcoin_test.csv file we were provided, we did not drop the Unnamed: 0 column from our training data as we were not allowed to modify/clean the bitcoin_test.csv file.

In [5]:
```
# Drop the row index column
# bitcoin_train = bitcoin_train.drop('Unnamed: 0', axis=1)
```

For future analysis, we can establish the variable type of each feature.

- address: qualitative
- year: quantitative
- day: quantitative
- length: quantitative
- weight: quantitative
- count: quantitative
- looped: quantitative
- income: quantitative
- label: qualitative

We also notice that `year` and `day` represent the transaction date, so we can combine the two to derive a datetime `date` feature for our dataframe.

In [6]:
```
bitcoin_train['date'] = pd.to_datetime(bitcoin_train['year'] * 1000 + bitcoin_train['da
```

It may be beneficial to sort our dataframe chronologically, so let's do that now.

In [7]:
```
bitcoin_train = bitcoin_train.sort_values('date')
```

In [8]:
```
bitcoin_train
```

Out[8]:

	Unnamed: 0	address	year	day	length	weight	count	lo
2143902	2143902	12ytBU5EHDDEz4iudKVpLtvegQZFy1Fzdx	2011	1	38	9.536743e-07	1	
1053133	1053133	1HSELaheFmPw1nk6RUU4ovXLbhGP79FbpJ	2011	1	36	1.907349e-06	1	
210861	210861	124TQXntz7akAYCrUxDjLuVg7fLGTJqcgt	2011	1	6	6.250000e-02	1	
1491177	1491177	1FhMt2iTqJdwupYbgAHzZiRj4CGGZBjAdu	2011	1	8	8.750000e-01	4	
270322	270322	1H8bG6rjcEoeuus9RiR2VcYkbMF5QDeTqp	2011	1	58	9.313226e-10	1	

	Unnamed: 0	address	year	day	length	weight	count	lo
...	...	...	...	...	...	...	...	
142263	142263	3B1Rt9AvBaG87ncHAwYR4FE33BZTrDH9cr	2018	330	144	9.095650e-02	4142	
1365766	1365766	369CeByY5HyCMiy8ZUAo6rTAzHqZnZz2XE	2018	330	0	5.000000e-01	1	
7481	7481	3BMEXnk3r1suDs8jWSRRcEAXAEn7CZxcgm	2018	330	2	3.750000e-01	1	
1663186	1663186	18cvuNsgvhb3jhHDS14fgvcb2dWiXtuLkv	2018	330	14	2.000000e-02	1	
449720	449720	3NYRNV7qReWZGdCrzonypcaeNDTpbWfvgX	2018	330	6	5.000000e-01	1	

2333357 rows × 12 columns

In [9]:

```
# Check for null values in each feature
bitcoin_train.isnull().sum()
```

Out[9]:

```
Unnamed: 0    0
address       0
year          0
day           0
length        0
weight        0
count         0
looped        0
neighbors     0
income        0
label         0
date          0
dtype: int64
```

Although the dataframe doesn't contain any null values, there may be placeholder values that skew the data incorrectly. Let's loop through each column and check their unique values for any odd values.

In [10]:

```
# Check unique values of each column to identify any outliers, mispellings, or discrepe
for col in bitcoin_train.columns:
    print(col + ' values:')
    print((bitcoin_train[col].unique()))
```

```
Unnamed: 0 values:
[2143902 1053133  210861 ...    7481 1663186  449720]
address values:
['12ytBU5EHDDEz4iudKVpLtvegQZFy1Fzdx' '1HSELaheFmPw1nk6RUU4ovXLbhGP79FbpJ'
 '124TQXntz7akAYCrUxDjLuVg7fLGTJqcgt' ...
 '3BMEXnk3r1suDs8jWSRRcEAXAEn7CZxcgm' '18cvuNsgvhb3jhHDS14fgvcb2dWiXtuLkv'
 '3NYRNV7qReWZGdCrzonypcaeNDTpbWfvgX']
year values:
[2011 2012 2013 2014 2015 2016 2017 2018]
day values:
[  1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18
```

```
  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36
  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54
  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72
  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90
  91  92  93  94  95  96  97  98  99 100 101 102 103 104 105 106 107 108
 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126
 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144
 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162
 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180
 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198
 199 200 201 202 203 204 205 206 207 208 209 210 211 212 213 214 215 216
 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233 234
 235 236 237 238 239 240 241 242 243 244 245 246 247 248 249 250 251 252
 253 254 255 256 257 258 259 260 261 262 263 264 265 266 267 268 269 270
 271 272 273 274 275 276 277 278 279 280 281 282 283 284 285 286 287 288
 289 290 291 292 293 294 295 296 297 298 299 300 301 302 303 304 305 306
 307 308 309 310 311 312 313 314 315 316 317 318 319 320 321 322 323 324
 325 326 327 328 329 330 331 332 333 334 335 336 337 338 339 340 341 342
 343 344 345 346 347 348 349 350 351 352 353 354 355 356 357 358 359 360
 361 362 363 364 365]
length values:
[ 38  36   6   8  58  14 112  52  56  20   4  16   0   2  18 100  30  26
  42 118  50  12  54  40  28  48  46 114  22  64 122  34  74  32 132 110
  62  24  78  10  84  82 108  44 116  66  70  90  98  96  94  92  60 136
  68 102 106 126 140 142 104 128 130 134 120  76 138 124  72  86  88  80
 144]
weight values:
[9.53674316e-07 1.90734863e-06 6.25000000e-02 ... 3.26711616e-04
 2.00088947e+00 9.09565024e-02]
count values:
[    1     4    20 ... 12161 12008 10659]
looped values:
[   0   20    1 ... 8016 7168 8173]
neighbors values:
[    2     1     4     5    56     3     6    27    11     8     7    10
     9    14    13    16    12    15    17    44    28    18    20   146
    22    40    34    21    23    19    35    25    30    32    29    54
    55    24    81    74    79    37    50    68    77    82    57    53
    62    65    59    38    49    51    60    26    45   100    43    31
    41    47    36    48   327    39   104    91    84    90    58    75
   525    33    80    67    64    46    61    66    63   130    76    83
    87  1463  1552  1422  1536  1770  1603    52  1728    69    42    72
    96  1565  1518  1586  1529   122   106   103    78  1553  1287  1593
  1396  1311  1319    88  1406   158   193   152   420   281  1426   277
   214   286    98   174   189   197   211   212   226  1230   350   370
   329   603   528   479   447   507   402   405   478   423   112   475
   477   485   352  1262   468    94   482   483   449   317   410   844
   414   314   721   471    85   533   118   632  1002   629   556   622
   673   641   201   199  1203   522   510   481   535   151   575   384
   144   494   263   656  1046   228  1006   817   669   727   633   505
   602   110   216   713   346   302   337   246   255   323   267   714
   472   347   906   365   251   264   266   119   141   140   134   143
   105   223   202   219   204   218   160   196   238   195   153   162
   276   184   180   382   349   250   185   171   167   148   235   230
   271   292   239   268   128   155   172   176   285   275   343   392
   364   417    89   372   356   411   333   338   354   351   439   419
  2111   375   200   188   198   974   363   324   240   190   490   440
  1774   341   433   450   524   342   186   403   305   291  1149   295
   257   416   387   308   298   135   319   326   120   150   156   496
   227   422   759   169   723   451   339   297  1303   728  3776  1470
   444   252   480   220   868   159   984  1445   394   301   334   127
   391   322   316   274   109   306   289   348   312   113   269   139
   309   206   340   111   446   147   474   445   108   114   360  1543
   173   344   762   467   393   123   518   425   441   273  1402   367
   187   580   249   245   278   237   124   332   310   256   502   311
```

```
   191   320   117   368   236   233  2012   371   973   261   777   161
   258   126   203   207   613   178    70   470   453   554   442   102
   254   290   213   209   288   377   526   270  1324   328  1576   994
   208   137    73   287   272   300   570   626   374   381   361   222
   210    95   157   248   434   116   546  1270   664   400   366   427
   390  1785   536  1392   304   389   491   466   435   408   610   597
   456   573   154   513   636   129   259   229   175   241   383   407
   192    97   730   488   412   452   459   429   895   903    99   458
   542   486   166  1288   662   437   687   778   385   476   401   599
   549    71   590   232   415   362   398   182   836   404    86   592
   587   724   872   299   355   386   676   624   555   517  1156   586
   101   497   690   746   720   715   770   771  1047  1292   966   142
   693   543  1378   749   696   833   572   719  1071   704   755   705
  2502   530   625   661  1512  1061   853   700   905   163   799   734
   133   810   283   910   739   681   837   896  1022  2089   244   901
   804   231   839   818   754  1163   779  1119   938  1121   642    92
   851   260   438   547   865   205   516   225   177   635   527   279
   379  1343   115   776   750  2494   992   647   593   616   672   808
   567   132   380   376   561   523   685   640   630   436   511   679
   598   595   790   562  1001   489   512   551  1451   280   136   170
   145   215   234   121   318   321   388   594   265   315   335   253
   194   165   657    93   369   217  1308  1231   179   164   850   131
   331   457   396   824   589  1094   224   431   609   336   107  3080
   243   965   866   614  1078   149  1809  1194   772   785  1488   168
   996   125   707   492   579  1830   828   138   615   668  1703   648
  1301   876   464   812   813   282  1139   600  1236   552   487   877
  1010  1153   919  1120   569  1232   604  1005   712   540  3541   409
  3575   665  1293   307   293  1196  1253   688  1389  1655  1143   835
   498   769   634  2475   399   247  1656  1360   455   495  1192  2264
   578   761   619   565   506   963  7589   358  1190  1233   242   652
   819   262 11746   378  1549   294   430   620   968   418 12920   881
   469   930]
income values:
[4.62000000e+10 5.11020000e+10 1.88900000e+09 ... 1.72257777e+08
 2.26934296e+08 6.70477700e+07]
label values:
['white' 'CryptoLocker' 'Razy' 'NoobCrypt' 'CryptoWall' 'GlobeImposter'
 'DMALocker' 'CryptoTorLocker2015' 'SamSam' 'Locky' 'Cerber' 'KeRanger'
 'EDA2' 'Jigsaw' 'XTPLocker' 'CryptXXX' 'JigSaw' 'DMALockerv3' 'APT'
 'Globe' 'Globev3' 'Sam' 'ComradeCircle' 'Flyper' 'VenusLocker'
 'XLockerv5.0' 'CryptConsole' 'WannaCry' 'XLocker']
date values:
['2011-01-01T00:00:00.000000000' '2011-01-02T00:00:00.000000000'
 '2011-01-03T00:00:00.000000000' ... '2018-11-24T00:00:00.000000000'
 '2018-11-25T00:00:00.000000000' '2018-11-26T00:00:00.000000000']
```

In [11]:

```
# Use value counts to identify distribution of labels
bitcoin_train['label'].value_counts()
```

Out[11]:

```
white                  2300268
CryptoWall                9872
CryptoLocker              7422
Cerber                    7381
Locky                     5320
CryptXXX                  1933
NoobCrypt                  388
DMALockerv3                290
DMALocker                  210
CryptoTorLocker2015         47
SamSam                      45
GlobeImposter               36
Globev3                     28
WannaCry                    24
Globe                       22
```

```
Razy                    13
APT                      8
KeRanger                 8
CryptConsole             7
XTPLocker                7
Flyper                   7
VenusLocker              6
JigSaw                   4
XLockerv5.0              4
EDA2                     3
ComradeCircle            1
Jigsaw                   1
XLocker                  1
Sam                      1
Name: label, dtype: int64
```

In [12]:

```
bitcoin_train['label'].value_counts().iloc[:4]
```

Out[12]:

```
white          2300268
CryptoWall        9872
CryptoLocker      7422
Cerber            7381
Name: label, dtype: int64
```

All the values seem correct, and we can use the value_counts function to tentatively identify that the top 3 ransom labels are CryptoWall (9,872 transactions), CryptoLocker (7,422 transactions), and Cerber (7,381 transactions).

With that, our data is sufficiently cleaned, and we can proceed with the next step of Data Visualization.

Step 2: Data Visualization

In [13]:

```
# Import Data Viz packages
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

%matplotlib inline
```

Since the visualization of white labels is so dense due to the greater volume of data, we exclude it for some basic visualizations to get a better idea of how the different ransomwares are related to each other. We also exclude outliers because they don't have a large effect on how the primary relationships are visualized.

In [14]:

```
excl_white = bitcoin_train[bitcoin_train['label'] != 'white']
excl_weight_outliers = excl_white[np.abs(excl_white.weight-excl_white.weight.mean()) <=
```

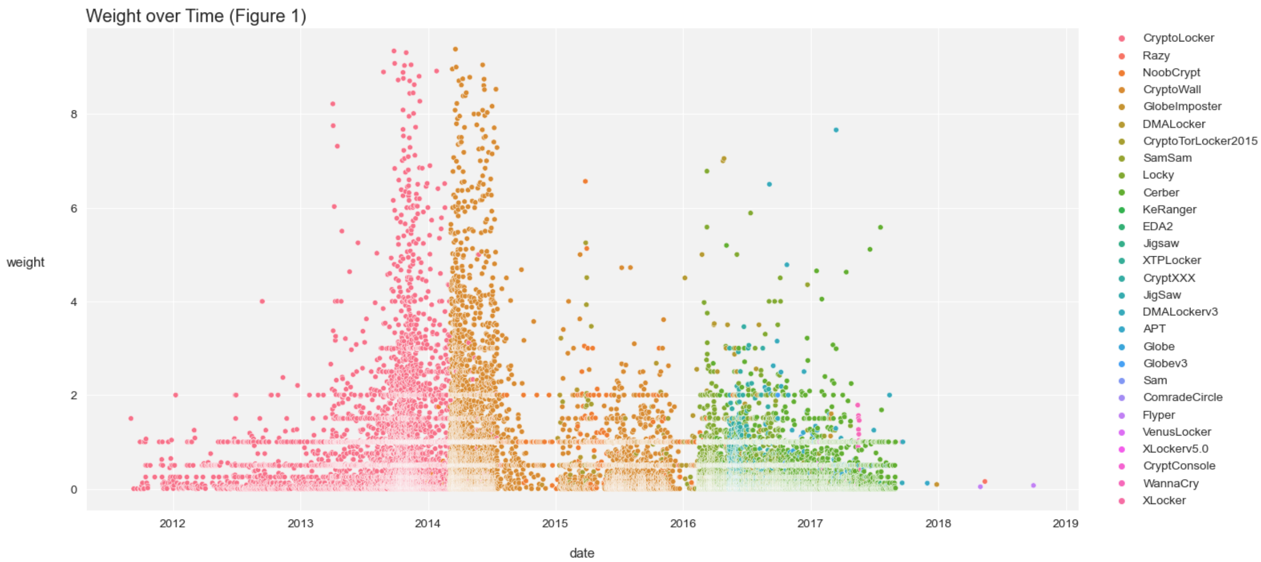
In [15]:

```
plt.figure(figsize = (20,10))
sns.set(font_scale = 1.23)

fig = sns.scatterplot(data = excl_weight_outliers, x = 'date', y = 'weight', hue = 'lab
fig.patch.set_facecolor('#f2f2f2')
fig.patch.set_edgecolor('black')
fig.set_ylabel('weight', fontsize=15, rotation=0)
```

```
fig.set_xlabel('date', fontsize=15, rotation=0)
fig.set_title('Weight over Time (Figure 1)', fontsize=20, loc='left')
fig.yaxis.labelpad = 50
fig.xaxis.labelpad = 20
plt.legend(facecolor = 'white', edgecolor = 'white', bbox_to_anchor = (1.02, 1.01), loc
```

Out[15]:

```
<matplotlib.legend.Legend at 0x1743d191bc8>
```



In [16]:

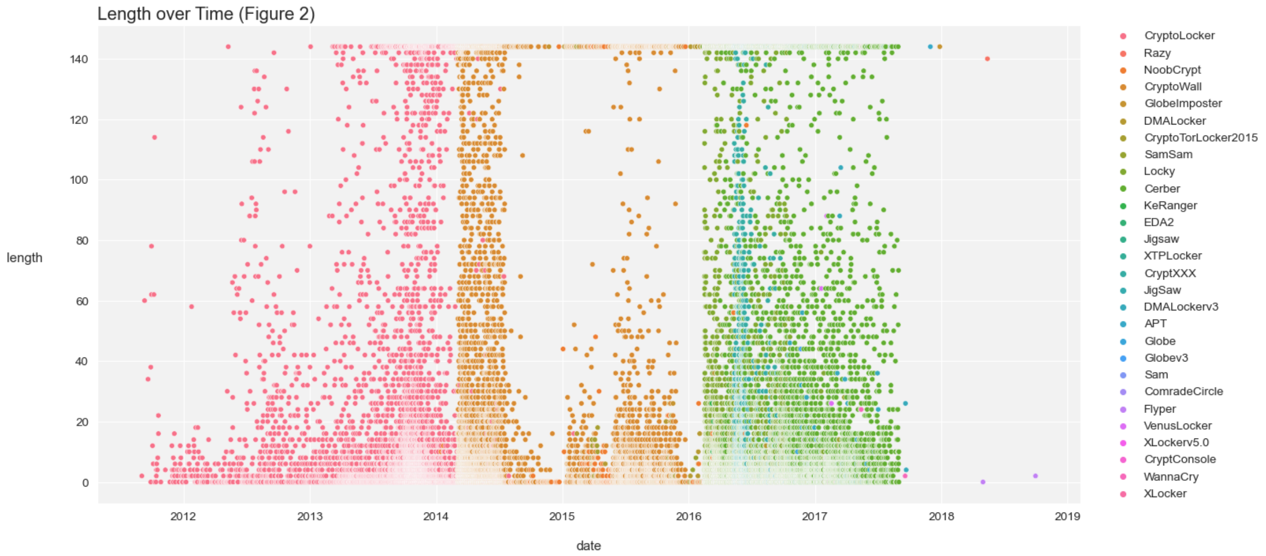
```
excl_length_outliers = excl_white[np.abs(excl_white.length-excl_white.length.mean()) <=
```

In [17]:

```
plt.figure(figsize = (20,10))
sns.set(font_scale = 1.23)

fig = sns.scatterplot(data = excl_weight_outliers, x = 'date', y = 'length', hue = 'lab
fig.patch.set_facecolor('#f2f2f2')
fig.patch.set_edgecolor('black')
fig.set_ylabel('length', fontsize=15, rotation=0)
fig.set_xlabel('date', fontsize=15, rotation=0)
fig.set_title('Length over Time (Figure 2)', fontsize=20, loc='left')
fig.yaxis.labelpad = 50
fig.xaxis.labelpad = 20
plt.legend(facecolor = 'white', edgecolor = 'white', bbox_to_anchor = (1.02, 1.01), loc
```

Out[17]:

```
<matplotlib.legend.Legend at 0x17440c95d88>
```



In [18]:

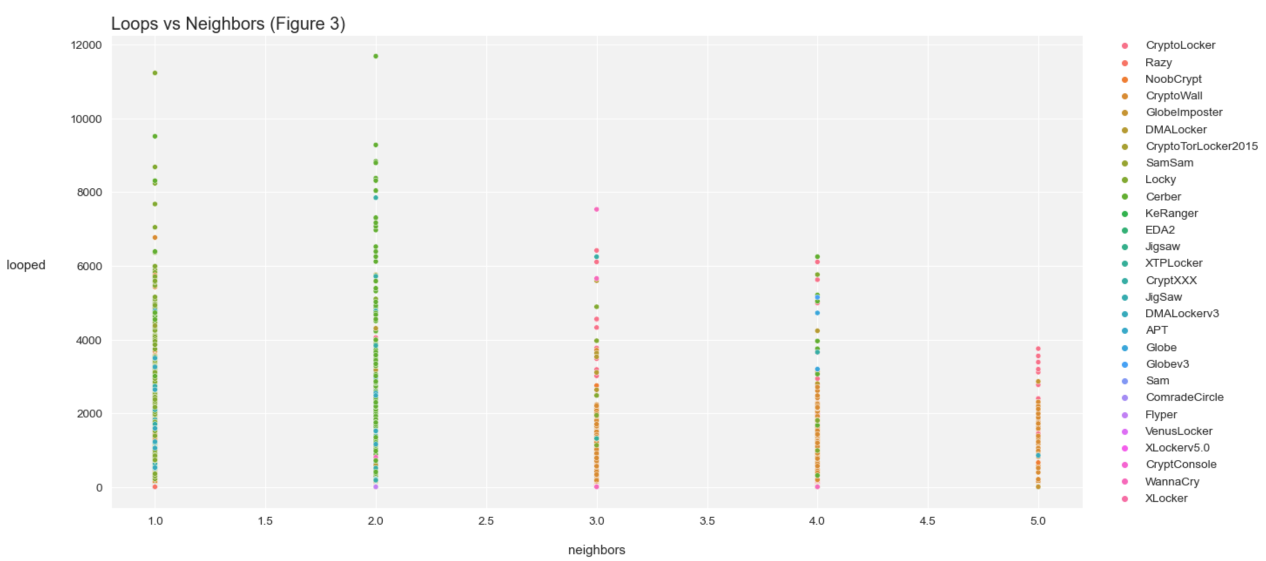
```
excl_white = bitcoin_train[bitcoin_train['label'] != 'white']
excl_neighbors_outliers = excl_white[np.abs(excl_white.neighbors-excl_white.neighbors.m
excl_n_l_outliers = excl_neighbors_outliers[np.abs(excl_neighbors_outliers.neighbors-ex
```

In [19]:

```
plt.figure(figsize = (20,10))
sns.set(font_scale = 1.23)

fig = sns.scatterplot(data = excl_n_l_outliers, x = 'neighbors', y = 'looped', hue = 'l
fig.patch.set_facecolor('#f2f2f2')
fig.patch.set_edgecolor('black')
fig.set_ylabel('looped', fontsize=15, rotation=0)
fig.set_xlabel('neighbors', fontsize=15, rotation=0)
fig.set_title('Loops vs Neighbors (Figure 3)', fontsize=20, loc='left')
fig.yaxis.labelpad = 50
fig.xaxis.labelpad = 20
plt.legend(facecolor = 'white', edgecolor = 'white', bbox_to_anchor = (1.02, 1.01), loc
```

Out[19]:

```
<matplotlib.legend.Legend at 0x1744130fc08>
```

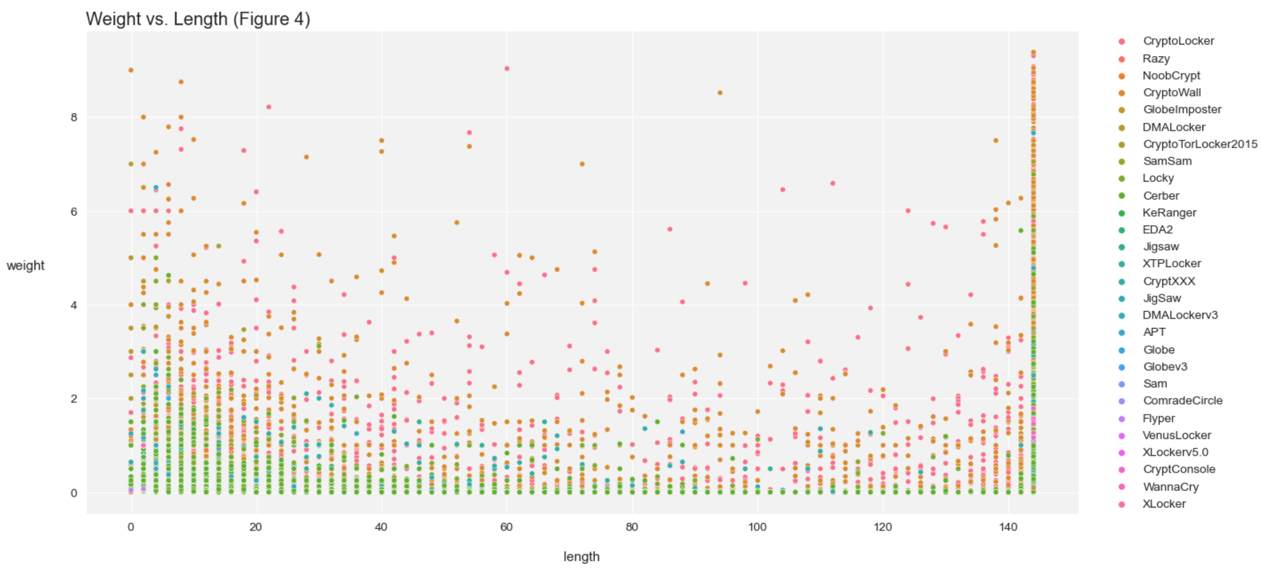


In [20]:

```
excl_length_outliers = excl_white[np.abs(excl_white.length-excl_white.length.mean()) <=
excl_l_w_outliers = excl_length_outliers[np.abs(excl_length_outliers.weight-excl_length
```

In [21]:
```
plt.figure(figsize = (20,10))
sns.set(font_scale = 1.23)

fig = sns.scatterplot(data = excl_l_w_outliers, x = 'length', y = 'weight', hue = 'labe
fig.patch.set_facecolor('#f2f2f2')
fig.patch.set_edgecolor('black')
fig.set_ylabel('weight', fontsize=15, rotation=0)
fig.set_xlabel('length', fontsize=15, rotation=0)
fig.set_title('Weight vs. Length (Figure 4)', fontsize=20, loc='left')
fig.yaxis.labelpad = 50
fig.xaxis.labelpad = 20
plt.legend(facecolor = 'white', edgecolor = 'white', bbox_to_anchor = (1.02, 1.01), loc
```

Out[21]: `<matplotlib.legend.Legend at 0x174414f3dc8>`

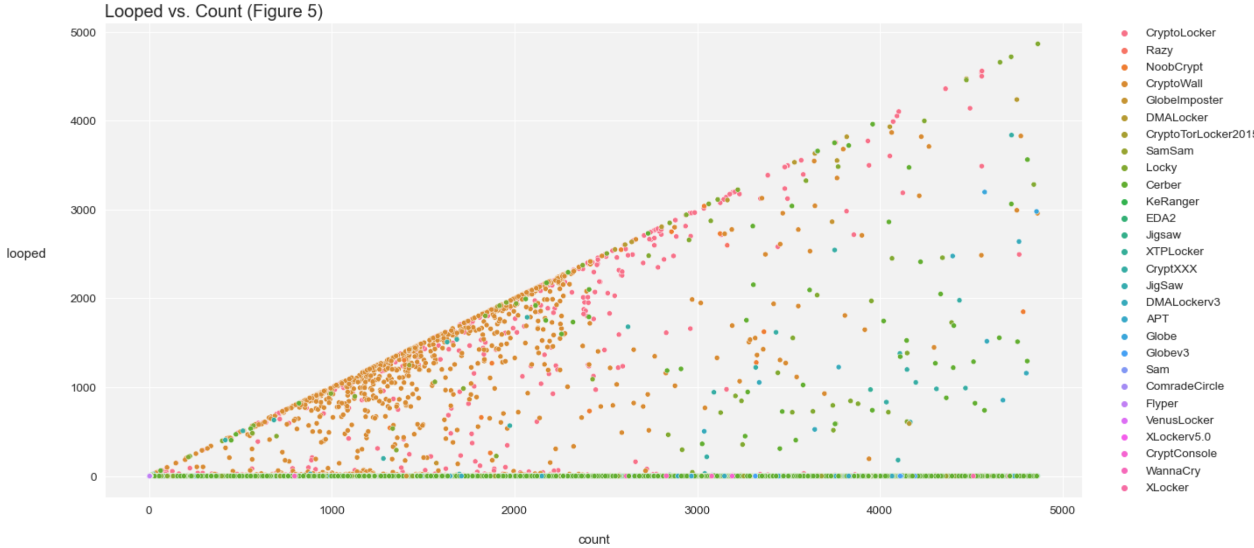


In [22]:
```
excl_count_outliers = excl_white[np.abs(excl_white['count']-excl_white['count'].mean())
excl_c_l_outliers = excl_count_outliers[np.abs(excl_white['looped']-excl_white['looped'
```

```
C:\Users\Eric Wang\anaconda3\lib\site-packages\ipykernel_launcher.py:2: UserWarning: Boo
lean Series key will be reindexed to match DataFrame index.
```

In [23]:
```
plt.figure(figsize = (20,10))
sns.set(font_scale = 1.23)

fig = sns.scatterplot(data = excl_count_outliers, x = 'count', y = 'looped', hue = 'lab
fig.patch.set_facecolor('#f2f2f2')
fig.patch.set_edgecolor('black')
fig.set_ylabel('looped', fontsize=15, rotation=0)
fig.set_xlabel('count', fontsize=15, rotation=0)
fig.set_title('Looped vs. Count (Figure 5)', fontsize=20, loc='left')
fig.yaxis.labelpad = 50
fig.xaxis.labelpad = 20
plt.legend(facecolor = 'white', edgecolor = 'white', bbox_to_anchor = (1.02, 1.01), loc
```

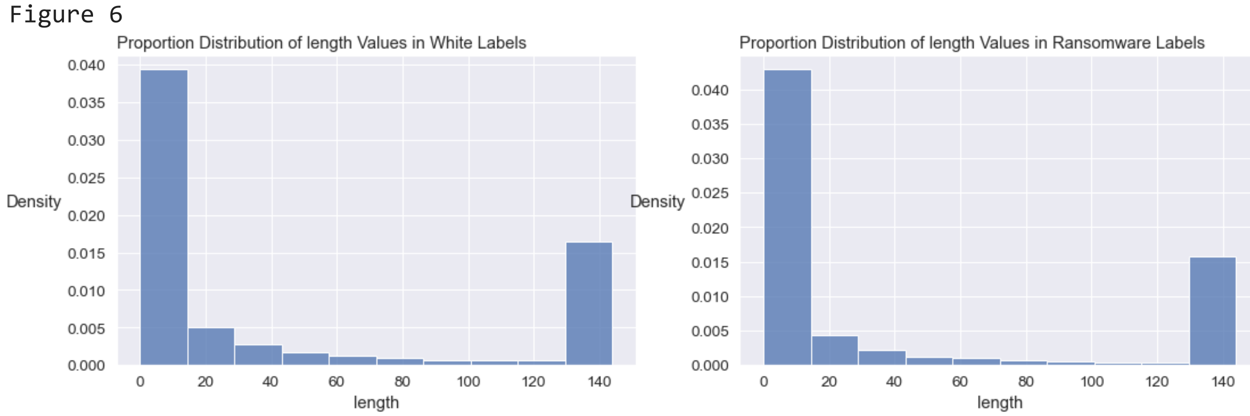
Out[23]: `<matplotlib.legend.Legend at 0x17442e35e88>`

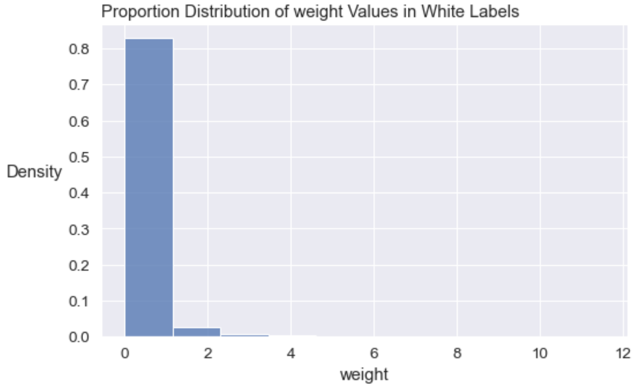


In [24]:

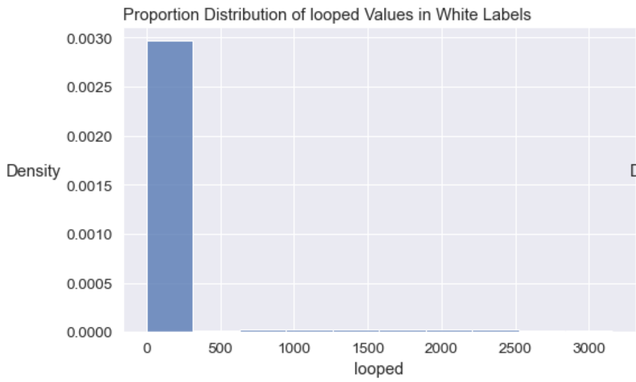
```
print('Figure 6')
for i in ['length', 'weight', 'looped', 'neighbors', 'income', 'count']:

    only_white = bitcoin_train[bitcoin_train['label'] == 'white']
    only_ransomware = bitcoin_train[bitcoin_train['label'] != 'white']

    excl_white_outliers = only_white[np.abs(only_white[i]-only_white[i].mean()) <= (3*o
    excl_ransomware_outliers = only_ransomware[np.abs(only_ransomware[i]-only_ransomwar

    fig, axes = plt.subplots(1, 2)
    fig.set_size_inches(18, 5)

    sns.histplot(excl_white_outliers, bins=10, stat='density', ax = axes[0])
    sns.histplot(excl_ransomware_outliers, bins=10, stat='density')


    axes[0].set_title('Proportion Distribution of ' + i + ' Values in White Labels', lo
    axes[1].set_title('Proportion Distribution of ' + i + ' Values in Ransomware Labels

    for axes in axes:
        axes.set_ylabel(axes.get_ylabel(), rotation=0, horizontalalignment='right')
```

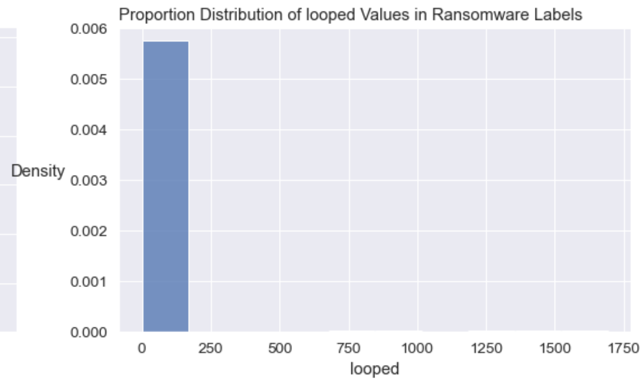
Figure 6

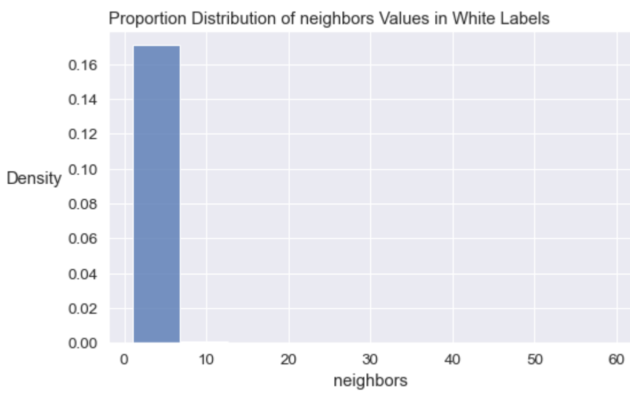


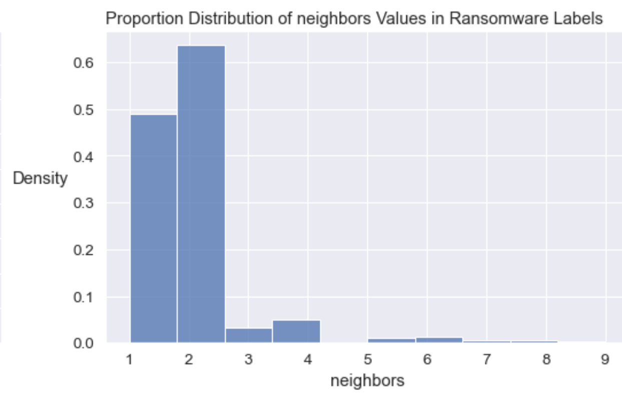

Proportion Distribution of weight Values in White Labels
Density
weight

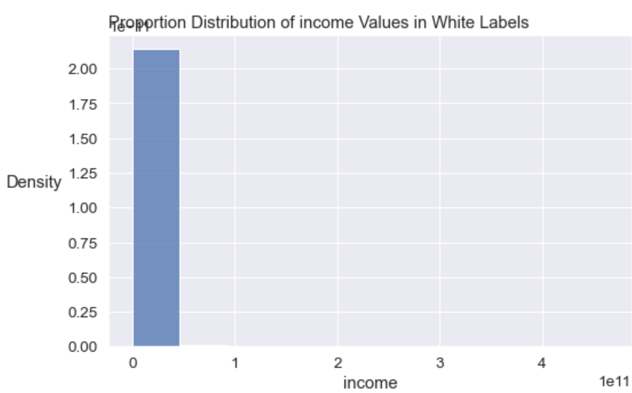

Proportion Distribution of weight Values in Ransomware Labels
Density
weight

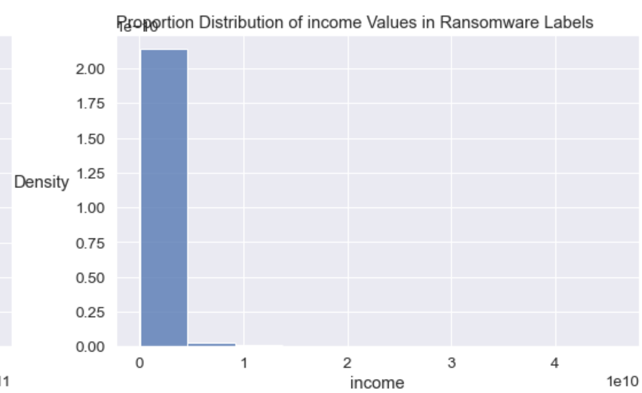

Proportion Distribution of looped Values in White Labels
Density
looped

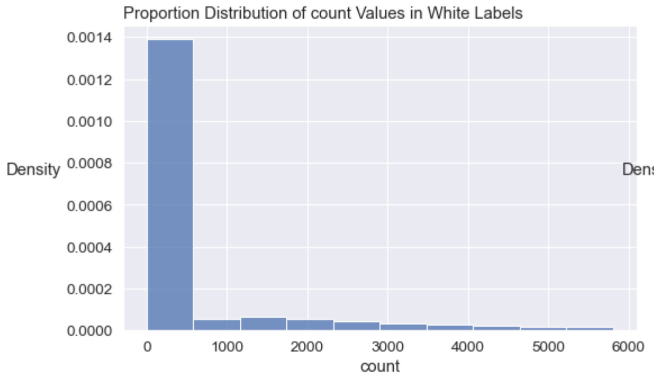

Proportion Distribution of looped Values in Ransomware Labels
Density
looped

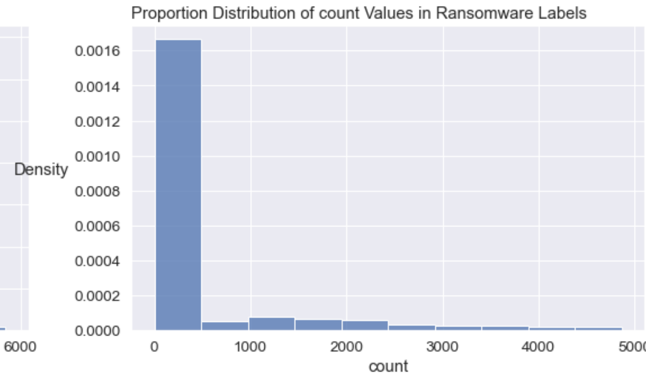

Proportion Distribution of neighbors Values in White Labels
Density
neighbors


Proportion Distribution of neighbors Values in Ransomware Labels
Density
neighbors


Proportion Distribution of income Values in White Labels
Density
income
1e11


Proportion Distribution of income Values in Ransomware Labels
Density
income
1e10





Plotting the ransomware label weights (excluding white labels and outliers) over time in figures 1 and 2 reveals a few interesting patterns.

1. There are somewhat overlapping but distinct windows of time during which specific ransomware transactions were made.
2. In previous years (2011 - 2014), we can see more distinct cutoffs when only one or two ransomwares are used at the same time, but later years (2015 - 2019) have a much greater variety of ransomwares used.
3. There are odd groupings around even intervals of weight. We can see horizontal line patterns at approximately `weight` = {0.25, 0.5, 1, 1.5, 2, and 3}. This is likely some kind of rounding process that occurred during the data collection process where values close to these specific values are defaulted to the nearest value.

More observations can be gleaned from the features' relationships with each other in figures 3, 4, and 5.

1. It appears as though there is some form of distinction between the number of neighbors specific ransomwares have in their transactions (shown in figure 4).
2. The count value seems to define the maximum looped value (shown in figure 5). This could be because larger bitcoin transactions require greater personal info/data security through extensive looping.

Figure 6 gives some insight on how the histogram distributions compare between white labels and ransomware labels across all features. Overall, the white label and ransomware label distributions appear similar for most features. It can be noted that the white label sample size is much greater than the number of ransomware label samples. This results in the white labels having a greater spread of values, even after outliers have been removed.

Step 3: Hypothesis/Experimental Testing

Since we only have a general idea of each features' relationship with `label` , we want to conduct some hypothesis tests to identify statistically significant feature trends that can be used in a machine learning model. We'll conduct a hypothesis test for measures of center and spread such as mean, min, max, etc. on our quantitative data (`length` , `weight` , `count` , `looped` , `neighbors` , `date` , and `label`).

Note: we excluded year, day, and date since they aren't measures of the actual transaction properties.

For our first hypothesis test, we define the hypotheses as follows:

H_0: The average weight of white labels equal the average weight of ransomware labels.

H_a: The average weight of white labels does not equal the average weight of ransomware labels.

α: 0.05

Since we are comparing two groups in a sample, we conduct an independent samples t-test. This will involve calculating pooled standard deviation in order to calculate the test statistic.

In [25]:
```
# Import Hypothesis testing packages
import numpy as np
from scipy import stats
```

In [26]:
```
white_only = bitcoin_train[bitcoin_train['label'] == 'white']
white_weight = white_only['weight'].mean()
white_count = white_only.shape[0]
white_std = white_only['weight'].std()
white_weight
```

Out[26]: 0.5444412201516179

In [27]:
```
ransomware_only = bitcoin_train[bitcoin_train['label'] != 'white']
ransomware_weight = ransomware_only['weight'].mean()
ransomware_count = ransomware_only.shape[0]
ransomware_std = ransomware_only['weight'].std()
ransomware_weight
```

Out[27]: 0.6288455501721444

In [28]:
```
pooled_variance = (((white_count - 1) * (white_std ** 2)) + ((ransomware_count - 1) * (
std_error = np.sqrt(pooled_variance) * np.sqrt((1 / white_count) + (1 / ransomware_coun
t_s = (white_weight - ransomware_weight) / std_error
t_s
```

Out[28]: -4.1733257275897175

The test statistic we calculate is t = -4.1733, and we can compare it to the corresponding t-value (based on degrees of freedom) through the scipy t.cdf function.

In [29]:
```
deg_of_f = white_count + ransomware_count - 2
deg_of_f
```

Out[29]: 2333355

In [30]:
```
# * 2 because of two-tailed test
p = (1 - stats.t.cdf(abs(t_s), deg_of_f)) * 2
p
```

Out[30]: 3.0019614845944176e-05

Because our p-value is 3.002e-05 which is less than the significance level of 0.05, we are able to reject the null hypothesis. There is a statistically significant difference between average white label weight and average ransomware label weight. We can also repeat this process on all the features, but due to time constraints, we are not able to complete additional hypothesis tests just yet.

For our second hypothesis test, we define the hypotheses as follows:

H_0: The average number of neighbors of white labels equal the average number of neighbors of ransomware labels.

H_a: The average number of neighbors of white labels is less than the average number of neighbors of ransomware labels.

α: 0.05

Since we are comparing two groups in a sample, we conduct an independent samples t-test. This will involve calculating pooled standard deviation in order to calculate the test statistic. We also want a significance level of 0.05.

In [31]:

```
white_only = bitcoin_train[bitcoin_train['label'] == 'white']
white_neighbors = white_only['neighbors'].mean()
white_count = white_only.shape[0]
white_std = white_only['neighbors'].std()
white_neighbors
```

Out[31]: 2.215241876163995

In [32]:

```
ransomware_only = bitcoin_train[bitcoin_train['label'] != 'white']
ransomware_neighbors = ransomware_only['neighbors'].mean()
ransomware_count = ransomware_only.shape[0]
ransomware_std = ransomware_only['neighbors'].std()
ransomware_neighbors
```

Out[32]: 2.0684819728610715

In [33]:

```
pooled_variance = (((white_count - 1) * (white_std ** 2)) + ((ransomware_count - 1) * (
std_error = np.sqrt(pooled_variance) * np.sqrt((1 / white_count) + (1 / ransomware_coun
t_s = (white_neighbors - ransomware_neighbors) / std_error
t_s
```

Out[33]: 1.4037608018347385

In [34]:

```
deg_of_f = white_count + ransomware_count - 2
deg_of_f
```

Out[34]: 2333355

In [35]:

```
# * 1 because of single-tailed test
p = (1 - stats.t.cdf(abs(t_s), deg_of_f))
```

```
p
```

Out[35]:
```
0.08019511155581949
```

Our calculated p-value of 0.0802 is greater than our significance level of 0.05. Therefore, we fail to reject the null hypothesis and conclude that the average number of neighbors of ransomware transactions is not greater than the average number of neighbors of white transactions.

Step 4: Classification of Ransomware

In [36]:
```
# Import machine learning packages
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.naive_bayes import BernoulliNB
```

Edit: In order to match the bitcoin_test.csv file, we are dropping the previously added date feature. This is because our machine learning model needs to be fit to a dataset with the same number of features as the one it does predictions on (bitcoin_test.csv).

In [37]:
```
train_labels = bitcoin_train['label']
train_features = bitcoin_train.drop(['label', 'date'], axis=1)
```

In [38]:
```
# Creating the pipeline

# Discerning categorical features
categorical_features = ['address']
categorical_transformer = Pipeline(steps=[
    ('onehot', OneHotEncoder(handle_unknown='ignore'))])

# Discern numeric features
numeric_features = ['length', 'weight', 'count', 'looped', 'neighbors', 'income']
numeric_transformer = Pipeline(steps=[
    ('stdscaler', StandardScaler())])

# Use the columntransformer to preprocess
base_preprocessor = ColumnTransformer(
    transformers=[('cat', categorical_transformer, categorical_features),
                  ('num', numeric_transformer, numeric_features)], remainder="drop")

baseline_pl = Pipeline(steps=[('preprocessor', base_preprocessor),
                       ('classifier', BernoulliNB())])
```

In [39]:
```
baseX_train, baseX_test, basey_train, basey_test = train_test_split(train_features, tra
reals = basey_test.to_list()
```

In [40]:
```
preds = baseline_pl.fit(baseX_train, basey_train).predict(baseX_test)
```

In [41]:
```
print(accuracy_score(preds, reals))
```

```
0.985846590324682
```

In [42]:
```
test_preds = baseline_pl.fit(train_features, train_labels).predict(bitcoin_test)
```

In [43]:
```
bitcoin_test['predicted_label'] = test_preds
```

In [44]:
```
bitcoin_test.head()
```

Out[44]:

	Unnamed: 0	address	year	day	length	weight	count	looped	nei
0	0	16r8CxcVCypUFzvHHZYttyiZtMaGnJn3te	2014	49	0	1.0000	1	0	
1	1	12EK9jUdG3heM7AF6Abyp38yuNMHN4dcq1	2017	265	36	0.0625	1	0	
2	2	16xUAFderxZwbEp9yuz4FdPnMVxTQntcwN	2017	44	0	1.0000	1	0	
3	3	1JvUt1UUDey7JY7WYHNTBSUNuhq1Vkbdfd	2013	264	4	0.1875	2	0	
4	4	138BLKDpeNyKdHnrLT6hZMW119sD4PZJ6D	2014	348	48	1.0000	1	0	

In [45]:
```
bitcoin_test.to_csv('predictions.csv', chunksize=1000)
```

In [ ]: