Summary:

1. Data auditing and creation of derived variables for analysis
2. Correlation analysis
3. Network Traffic Reclassification Using IANA Repository
4. RFM-based customer segmentation modelling

Data Audit

In [ ]:

```
# import libraries
import numpy as np
import pandas as pd
```

In [ ]:

```
sample_df = pd.read_csv('18.csv')[:-3]
```

In [ ]:

```
sample_df.tail()
```

Out[ ]:

	ts	te	td	sa	da	sp	dp	pr	flg	fwd	...	mpls8	mpls9	mpls10	cl
674917	2/8/22 18:54	2/8/22 18:54	0.01	[redacted]		52298	19210	TCP	CEUAP.S.	0	...	0-0-0	0-0-0	0-0-0	0
674918	2/8/22 18:47	2/8/22 18:47	0.01			57084	443	UDP	....	0	...	0-0-0	0-0-0	0-0-0	0
674919	2/8/22 18:59	2/8/22 18:59	0.98			16993	53	UDP	....	0	...	0-0-0	0-0-0	0-0-0	0
674920	2/8/22 18:40	2/8/22 18:40	0.04			53543	1136	TCP	...A.RS.	0	...	0-0-0	0-0-0	0-0-0	0
674921	2/8/22 18:43	2/8/22 18:43	0.00			0	0	ICMP	....	0	...	0-0-0	0-0-0	0-0-0	0

5 rows × 48 columns

In [ ]:

```
sample_df.columns
```

Out[ ]:

```
Index(['ts', 'te', 'td', 'sa', 'da', 'sp', 'dp', 'pr', 'flg', 'fwd', 'stos',
       'ipkt', 'ibyt', 'opkt', 'obyt', 'in', 'out', 'sas', 'das', 'smk', 'dmk',
       'dtos', 'dir', 'nh', 'nhb', 'svln', 'dvln', 'ismc', 'odmc', 'idmc',
       'osmc', 'mpls1', 'mpls2', 'mpls3', 'mpls4', 'mpls5', 'mpls6', 'mpls7',
       'mpls8', 'mpls9', 'mpls10', 'cl', 'sl', 'al', 'ra', 'eng', 'exid',
       'tr'],
      dtype='object')
```

In [ ]:

```
# drop columns
sample_df.drop(columns=['fwd','sas','smk','dmk','das','dtos','nh','nhb','mpls1','mpls2','mpls2','mpls3','m
```

In [ ]:

```
sample_df.head()
```

Out[ ]:

	ts	te	td	sa	da	sp	dp	pr	flg	ipkt	...	obyt	in	out	dir	svln	dvln
0	2/8/22 18:54	2/8/22 18:54	0.00	[redacted]		52478	6379	TCP	......S.	1	...	0	12	15	0	0	0
1	2/8/22 18:42	2/8/22 18:42	0.08	[redacted]		50841	443	UDP	....	10	...	2886	15728709	16	0	0	0
2	2/8/22 18:41	2/8/22 18:41	0.00	[redacted]		34247	9002	TCP	......S.	1	...	0	12	15	0	0	0
3	2/8/22 18:37	2/8/22 18:37	0.00	[redacted]		57303	5060	UDP	....	1	...	576	12	15	0	0	0
4	2/8/22 18:43	2/8/22 18:43	0.02	[redacted]		32518	443	TCP	...A..S.	11	...	7504	52	12	0	0	0

5 rows × 22 columns

In [ ]:

```
# datatypes
sample_df.dtypes
```

Out[ ]:

```
ts        object
te        object
td       float64
sa        object
da        object
sp         int64
dp         int64
pr        object
flg       object
ipkt       int64
ibyt       int64
opkt       int64
obyt       int64
in         int64
out        int64
dir        int64
svln       int64
dvln       int64
ismc      object
odmc      object
idmc      object
osmc      object
dtype: object
```

In [ ]:

```
# convert ts, te, tr to datetime format, sp and dp to object
sample_df['ts'] = pd.to_datetime(sample_df['ts'])
sample_df['te'] = pd.to_datetime(sample_df['te'])
sample_df['sp'] = sample_df['sp'].astype(object)
sample_df['dp'] = sample_df['dp'].astype(object)
sample_df['obyt']=sample_df['obyt'].astype(int)
sample_df['ibyt']=sample_df['ibyt'].astype(int)
sample_df['opkt']=sample_df['opkt'].astype(int)
sample_df['ipkt']=sample_df['ipkt'].astype(int)
sample_df['td']=sample_df['td'].astype(float)
sample_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 674922 entries, 0 to 674921
Data columns (total 22 columns):
 #   Column  Non-Null Count   Dtype
---  ------  --------------   -----
 0   ts      674922 non-null  datetime64[ns]
 1   te      674922 non-null  datetime64[ns]
 2   td      674922 non-null  float64
 3   sa      674922 non-null  object
 4   da      674922 non-null  object
 5   sp      674922 non-null  object
 6   dp      674922 non-null  object
 7   pr      674922 non-null  object
 8   flg     674922 non-null  object
 9   ipkt    674922 non-null  int64
 10  ibyt    674922 non-null  int64
 11  opkt    674922 non-null  int64
 12  obyt    674922 non-null  int64
 13  in      674922 non-null  int64
 14  out     674922 non-null  int64
 15  dir     674922 non-null  int64
 16  svln    674922 non-null  int64
 17  dvln    674922 non-null  int64
 18  ismc    674922 non-null  object
 19  odmc    674922 non-null  object
 20  idmc    674922 non-null  object
 21  osmc    674922 non-null  object
dtypes: datetime64[ns](2), float64(1), int64(9), object(10)
memory usage: 113.3+ MB
```

In [ ]:

```
# checking for null
sample_df.isnull().values.any()
```

Out[ ]:

```
False
```

In [ ]:

```
# checking for duplicated values
sample_df.duplicated().values.any()
```

Out[ ]:

```
False
```

In [ ]:

```
sample_df.head()
```

Out[ ]:

	ts	te	td	sa	da	sp	dp	pr	flg	ipkt	...	obyt	in	out	dir	svln
0	2022-02-08 18:54:00	2022-02-08 18:54:00	0.00	[redacted]	[redacted]	52478	6379	TCP	......S.	1	...	0	12	15	0	0
1	2022-02-08 18:42:00	2022-02-08 18:42:00	0.08	[redacted]	[redacted]	50841	443	UDP	....	10	...	2886	15728709	16	0	0
2	2022-02-08 18:41:00	2022-02-08 18:41:00	0.00	[redacted]	[redacted]	34247	9002	TCP	......S.	1	...	0	12	15	0	0
3	2022-02-08 18:37:00	2022-02-08 18:37:00	0.00	[redacted]	[redacted]	57303	5060	UDP	....	1	...	576	12	15	0	0
4	2022-02-08 18:43:00	2022-02-08 18:43:00	0.02	[redacted]	[redacted]	32518	443	TCP	...A..S.	11	...	7504	52	12	0	0

5 rows × 22 columns

In [ ]:

```
# create new fields for start time and start date
sample_df['start_date'] = [d.date() for d in sample_df['ts']]
sample_df['start_time'] = [d.time() for d in sample_df['ts']]
```

In [ ]:

```
sample_df.head()
```

Out[ ]:

	ts	te	td	sa	da	sp	dp	pr	flg	ipkt	...	out	dir	svln	dvln	
0	2022-02-08 18:54:00	2022-02-08 18:54:00	0.00	[redacted]	[redacted]	52478	6379	TCP	......S.	1	...	15	0	0	0	[redacted]
1	2022-02-08 18:42:00	2022-02-08 18:42:00	0.08	[redacted]	[redacted]	50841	443	UDP	....	10	...	16	0	0	0	[redacted]
2	2022-02-08 18:41:00	2022-02-08 18:41:00	0.00	[redacted]	[redacted]	34247	9002	TCP	......S.	1	...	15	0	0	0	[redacted]
3	2022-02-08 18:37:00	2022-02-08 18:37:00	0.00	[redacted]	[redacted]	57303	5060	UDP	....	1	...	15	0	0	0	[redacted]
4	2022-02-08 18:43:00	2022-02-08 18:43:00	0.02	[redacted]	[redacted]	32518	443	TCP	...A..S.	11	...	12	0	0	0	[redacted]

5 rows × 24 columns

In [ ]:

```
# create new fields for end time and end date
sample_df['end_date'] = [d.date() for d in sample_df['te']]
sample_df['end_time'] = [d.time() for d in sample_df['te']]
```

In [ ]:

```
sample_df.head()
```

Out[ ]:

	ts	te	td	sa	da	sp	dp	pr	flg	ipkt	...	svln	dvln	ismc	
0	2022-02-08 18:54:00	2022-02-08 18:54:00	0.00	[redacted]		52478	6379	TCP	......S.	1	...	0	0	[redacted]	9
1	2022-02-08 18:42:00	2022-02-08 18:42:00	0.08	[redacted]		50841	443	UDP	....	10	...	0	0	[redacted]	8c
2	2022-02-08 18:41:00	2022-02-08 18:41:00	0.00	[redacted]		34247	9002	TCP	......S.	1	...	0	0	[redacted]	9
3	2022-02-08 18:37:00	2022-02-08 18:37:00	0.00	[redacted]		57303	5060	UDP	....	1	...	0	0	[redacted]	5
4	2022-02-08 18:43:00	2022-02-08 18:43:00	0.02	[redacted]		32518	443	TCP	...A..S.	11	...	0	0	[redacted]	b

5 rows × 26 columns

Derived Variables for Analysis

In [ ]:

```
new_df = sample_df.copy()
```

In [ ]:

```
new_df.head()
```

Out[ ]:

	ts	te	td	sa	da	sp	dp	pr	flg	ipkt	...	svln	dvln	ismc
0	2022-02-08 18:54:00	2022-02-08 18:54:00	0.00	[redacted]	[redacted]	52478	6379	TCP	......S.	1	...	0	0	[redacted]
1	2022-02-08 18:42:00	2022-02-08 18:42:00	0.08	[redacted]	[redacted]	50841	443	UDP	....	10	...	0	0	[redacted]
2	2022-02-08 18:41:00	2022-02-08 18:41:00	0.00	[redacted]	[redacted]	34247	9002	TCP	......S.	1	...	0	0	[redacted]
3	2022-02-08 18:37:00	2022-02-08 18:37:00	0.00	[redacted]	[redacted]	57303	5060	UDP	....	1	...	0	0	[redacted]
4	2022-02-08 18:43:00	2022-02-08 18:43:00	0.02	[redacted]	[redacted]	32518	443	TCP	...A..S.	11	...	0	0	[redacted]

5 rows × 26 columns

In [ ]:

```python
# creating new calculated field: total bytes = ibyt + obyt sent over the network
new_df["total_bytes"] = new_df["ibyt"] + new_df["obyt"]
new_df.head()
```

Out[ ]:

	ts	te	td	sa	da	sp	dp	pr	flg	ipkt	...	dvln	ismc
0	2022-02-08 18:54:00	2022-02-08 18:54:00	0.00	[redacted]		52478	6379	TCP	......S.	1	...	[redacted]	
1	2022-02-08 18:42:00	2022-02-08 18:42:00	0.08			50841	443	UDP	....	10	...		
2	2022-02-08 18:41:00	2022-02-08 18:41:00	0.00			34247	9002	TCP	......S.	1	...		
3	2022-02-08 18:37:00	2022-02-08 18:37:00	0.00			57303	5060	UDP	....	1	...		
4	2022-02-08 18:43:00	2022-02-08 18:43:00	0.02			32518	443	TCP	...A..S.	11	...		

5 rows × 27 columns

In [ ]:

```
# create new derived variable: byte_per_sec
new_df["byte_per_sec"] = new_df['total_bytes']/new_df['td']
new_df.head()
```

Out[ ]:

	ts	te	td	sa	da	sp	dp	pr	flg	ipkt	...	ismc	od
0	2022-02-08 18:54:00	2022-02-08 18:54:00	0.00	[redacted]		52478	6379	TCP	......S.	1	...	[redacted]	
1	2022-02-08 18:42:00	2022-02-08 18:42:00	0.08			50841	443	UDP	....	10	...		
2	2022-02-08 18:41:00	2022-02-08 18:41:00	0.00			34247	9002	TCP	......S.	1	...		
3	2022-02-08 18:37:00	2022-02-08 18:37:00	0.00			57303	5060	UDP	....	1	...		
4	2022-02-08 18:43:00	2022-02-08 18:43:00	0.02			32518	443	TCP	...A..S.	11	...		

5 rows × 28 columns

In [ ]:

```
# create new derived variable: out_byte_per_opkt
new_df["out_byte_per_opkt"] = new_df['obyt']/new_df['opkt']
new_df.head()
```

Out[ ]:

	ts	te	td	sa	da	sp	dp	pr	flg	ipkt	...	odmc	id
0	2022-02-08 18:54:00	2022-02-08 18:54:00	0.00	[redacted]		52478	6379	TCP	......S.	1	...	[redacted]	
1	2022-02-08 18:42:00	2022-02-08 18:42:00	0.08	[redacted]		50841	443	UDP	....	10	...	[redacted]	
2	2022-02-08 18:41:00	2022-02-08 18:41:00	0.00	[redacted]		34247	9002	TCP	......S.	1	...	[redacted]	
3	2022-02-08 18:37:00	2022-02-08 18:37:00	0.00	[redacted]		57303	5060	UDP	....	1	...	[redacted]	
4	2022-02-08 18:43:00	2022-02-08 18:43:00	0.02	[redacted]		32518	443	TCP	...A..S.	11	...	[redacted]	

5 rows × 29 columns

In [ ]:

```
# create new derived variable: in_byte_per_ipkt
new_df["in_byte_per_ipkt"] = new_df['ibyt']/new_df['ipkt']
new_df.head()
```

Out[ ]:

	ts	te	td	sa	da	sp	dp	pr	flg	ipkt	...	idmc	os
0	2022-02-08 18:54:00	2022-02-08 18:54:00	0.00	[redacted]		52478	6379	TCP	......S.	1	...	[redacted]	
1	2022-02-08 18:42:00	2022-02-08 18:42:00	0.08			50841	443	UDP	....	10	...		
2	2022-02-08 18:41:00	2022-02-08 18:41:00	0.00			34247	9002	TCP	......S.	1	...		
3	2022-02-08 18:37:00	2022-02-08 18:37:00	0.00			57303	5060	UDP	....	1	...		
4	2022-02-08 18:43:00	2022-02-08 18:43:00	0.02			32518	443	TCP	...A..S.	11	...		

5 rows × 30 columns

In [ ]:

```
# create new derived variable: pkt_per_sec
new_df["byte_per_sec"] = (new_df['ipkt']+new_df['opkt'])/new_df['td']
new_df.head()
```

Out[ ]:

	ts	te	td	sa	da	sp	dp	pr	flg	ipkt	...	idmc	os
0	2022-02-08 18:54:00	2022-02-08 18:54:00	0.00	[redacted]		52478	6379	TCP	......S.	1	...	[redacted]	
1	2022-02-08 18:42:00	2022-02-08 18:42:00	0.08			50841	443	UDP	....	10	...		
2	2022-02-08 18:41:00	2022-02-08 18:41:00	0.00			34247	9002	TCP	......S.	1	...		
3	2022-02-08 18:37:00	2022-02-08 18:37:00	0.00			57303	5060	UDP	....	1	...		
4	2022-02-08 18:43:00	2022-02-08 18:43:00	0.02			32518	443	TCP	...A..S.	11	...		

5 rows × 30 columns

In [ ]:

```
# create new derived variable: byte_delivery_ratio
new_df["byte_delivery_ratio"] = new_df['ibyt']/new_df['obyt']
new_df.head()
```

Out[ ]:

	ts	te	td	sa	da	sp	dp	pr	flg	ipkt	...	osmc	start_date	sta
0	2022-02-08 18:54:00	2022-02-08 18:54:00	0.00	[redacted]		52478	6379	TCP	......S.	1	...	[redacted]	2022-02-08	
1	2022-02-08 18:42:00	2022-02-08 18:42:00	0.08			50841	443	UDP	....	10	...		2022-02-08	
2	2022-02-08 18:41:00	2022-02-08 18:41:00	0.00			34247	9002	TCP	......S.	1	...		2022-02-08	
3	2022-02-08 18:37:00	2022-02-08 18:37:00	0.00			57303	5060	UDP	....	1	...		2022-02-08	
4	2022-02-08 18:43:00	2022-02-08 18:43:00	0.02			32518	443	TCP	...A..S.	11	...		2022-02-08	

5 rows × 31 columns

In [ ]:

```
# create new derived variable: pkt_delivery_ratio
new_df["pkt_delivery_datio"] = new_df['ipkt']/new_df['opkt']
new_df.head()
```

Out[ ]:

	ts	te	td	sa	da	sp	dp	pr	flg	ipkt	...	start_date	start_time	end_date
0	2022-02-08 18:54:00	2022-02-08 18:54:00	0.00	[redacted]	[redacted]	52478	6379	TCP	......S.	1	...	2022-02-08	18:54:00	2022-02-08
1	2022-02-08 18:42:00	2022-02-08 18:42:00	0.08	[redacted]	[redacted]	50841	443	UDP	....	10	...	2022-02-08	18:42:00	2022-02-08
2	2022-02-08 18:41:00	2022-02-08 18:41:00	0.00	[redacted]	[redacted]	34247	9002	TCP	......S.	1	...	2022-02-08	18:41:00	2022-02-08
3	2022-02-08 18:37:00	2022-02-08 18:37:00	0.00	[redacted]	[redacted]	57303	5060	UDP	....	1	...	2022-02-08	18:37:00	2022-02-08
4	2022-02-08 18:43:00	2022-02-08 18:43:00	0.02	[redacted]	[redacted]	32518	443	TCP	...A..S.	11	...	2022-02-08	18:43:00	2022-02-08

5 rows × 32 columns

Network Traffic Reclassification Using IANA Data

In [ ]:

```
from pandas_datareader import wb
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
from pandas.api.types import is_string_dtype, is_numeric_dtype
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
```

In [ ]:

```python
# load traffic types using ports and protocols from IANA website
traffic_df = pd.read_csv('https://www.iana.org/assignments/service-names-port-numbers/service-names-port-n
traffic_df.sample(20)
```

Out[ ]:

	Service Name	Port Number	Transport Protocol	Description	Assignee	Contact	Registration Date	Modification Date	Refere
8385	kar2ouche	4661	udp	Kar2ouche Peer location service	[Andy_Krouwel]	[Andy_Krouwel]	NaN	NaN	
162	deos	76	tcp	Distributed External Object Store	[Robert_Ullmann]	[Robert_Ullmann]	NaN	NaN	
13173	candp	42508	udp	Computer Associates network discovery protocol	[Jon_Press]	[Jon_Press]	2005-09	NaN	
11561	sec-pc2fax-srv	9402	tcp	Samsung PC2FAX for Network Server	[HyeongBae_Yu]	[HyeongBae_Yu]	2008-07-31	NaN	
12713	icl-twobase3	25002	tcp	icl-twobase3	[J_A_Sever]	[J_A_Sever]	NaN	NaN	
9276	NaN	5466-5469	NaN	Unassigned	NaN	NaN	NaN	NaN	
5071	qip-audup	2765	tcp	qip-audup	[Mike_Morgan]	[Mike_Morgan]	NaN	NaN	
1117	nqs	607	udp	nqs	[Bill_Schiefelbein]	[Bill_Schiefelbein]	NaN	NaN	
1251	vpps-qua	672	udp	VPPS-QUA	NaN	NaN	NaN	NaN	
5056	cnrp	2757	udp	CNRP	[Jacob_Ulmert]	[Jacob_Ulmert]	NaN	NaN	
8222	saris	4442	udp	Saris	NaN	NaN	NaN	NaN	
7970	spdm	4194	tcp	Security Protocol and Data Model	[Intel_Corporation]	[Eduardo_Cabre]	2022-01-10	NaN	

2796	cert-initiator	1639	udp	cert-initiator	NaN	NaN	NaN	NaN	
5423	sm-pas-2	2939	udp	SM-PAS-2	NaN	NaN	NaN	NaN	
13809	jukebox	NaN	tcp	Jukebox Request Service	[Gary_Giebler_2]	[Gary_Giebler_2]	2011-10-18	NaN	
14121	slpda	NaN	udp	Remote Service Discovery in the Service Location	NaN	NaN	NaN	NaN	[RFC38
10710	pnet-enc	7798	tcp	Propel Encoder port	[Leif_Hedstrom]	[Leif_Hedstrom]	2002-04	NaN	
368	vmnet	175	udp	VMNET	[Christopher_Tengi]	[Christopher_Tengi]	NaN	NaN	
8659	eq-office-4942	4942	tcp	Equitrac Office	[Tom_Haapanen_2]	[Tom_Haapanen_2]	2007-07-11	NaN	
2758	faxportwinport	1620	udp	faxportwinport	[Chris_Wells]	[Chris_Wells]	NaN	NaN	

In [ ]:

```
# create new columns with concatenated traffic code and traffic type used for indexing and matching
traffic_df['d_traffic_code'] = traffic_df['Port Number'].astype(str) + traffic_df['Transport Protocol']
traffic_df['s_traffic_code'] = traffic_df['Port Number'].astype(str) + traffic_df['Transport Protocol']
traffic_df.sample(20)
```

Out[ ]:

	Service Name	Port Number	Transport Protocol	Description	Assignee	Contact	Registration Date	Modification Date	Re
9838	dt-mgmtsvc	6325	tcp	Double-Take Management Service	[Carbonite_Inc]	[James_Wilkinson]	2012-06-06	2019-08-23	
7981	eims-admin	4199	udp	EIMS ADMIN	[Glenn_Anderson]	[Glenn_Anderson]	NaN	NaN	
156	netrjs-3	73	tcp	Remote Job Service	NaN	NaN	NaN	NaN	
7047	gw-call-port	3745	tcp	GWRTC Call Port	[Felisa_Ares]	[Felisa_Ares]	2003-04	NaN	

9798	tl1-raw-ssl	6251	tcp	TL1 Raw Over SSL/TLS	[Jim_Humphreys]	[Jim_Humphreys]	2008-01-29	NaN
5580	broker-service	3014	udp	Broker Service IANA assigned this well-formed ...	[Dale_Bethers]	[Dale_Bethers]	NaN	NaN
5064	dicom-iscl	2761	udp	DICOM ISCL	NaN	NaN	NaN	NaN
2537	fujitsu-dtc	1513	udp	Fujitsu Systems Business of America, Inc	NaN	NaN	NaN	NaN
11159	canon-bjnp3	8613	tcp	Canon BJNP Port 3	[Atsushi_Nakamura]	[Atsushi_Nakamura]	2003-11	NaN
7479	gvcp	3956	udp	GigE Vision Control	[Eric_Carey]	[Eric_Carey]	2005-08	NaN
10434	oma-rlp	7273	udp	OMA Roaming Location	[Larry_A_Young]	[Larry_A_Young]	2005-08	NaN
3326	fjicl-tep-b	1902	udp	Fujitsu ICL Terminal Emulator Program B	[Bob_Lyon]	[Bob_Lyon]	NaN	NaN
6072	hicp	3250	udp	HMS hicp port	[Joel_Palsson]	[Joel_Palsson]	2002-02	NaN
12711	icl-twobase2	25001	tcp	icl-twobase2	[J_A_Sever]	[J_A_Sever]	NaN	NaN
5767	itu-bicc-stc	3097	sctp	ITU-T Q.1902.1/Q.2150.3	[Greg_Sidebottom]	[Greg_Sidebottom]	NaN	NaN
13025	rt-helper	35006	tcp	ReadyTech Helper Service	[ReadyTech_Corporation]	[Kevin_Woodward]	2013-09-13	NaN
14189	teamlist	NaN	NaN	ARTIS Team Task	[ARTIS_Software]	[ARTIS_Software]	NaN	NaN
3936	hpocbus	2206	tcp	HP OpenCall bus	[Jerome_Forissier]	[Jerome_Forissier]	2005-12	NaN
267	nxedit	126	tcp	NXEdit	[Don_Payette]	[Don_Payette]	NaN	NaN

10201	NaN	6832-6840	NaN	Unassigned	NaN	NaN	NaN	NaN

In [ ]:

```python
# creating new columns for destination and source traffic type to call traffic types in morewave sample fi
traffic_df['s_traffic_type'] = traffic_df['Service Name']
traffic_df['d_traffic_type'] = traffic_df['Service Name']
traffic_df.sample(20)
```

Out[ ]:

	Service Name	Port Number	Transport Protocol	Description	Assignee	Contact	Registration Date	Modification Date
4368	rmtserver	2416	tcp	RMT Server	[Yvon_Marineau]	[Yvon_Marineau]	NaN	NaN
9085	hacl-probe	5303	tcp	HA cluster probing	[Eric_Soderberg_2] [Edward_Yim]	[Eric_Soderberg_2] [Edward_Yim]	NaN	NaN
8789	ita-manager	5052	tcp	ITA Manager	[Don_Merrell]	[Don_Merrell]	NaN	NaN
7130	upstriggervsw	3786	udp	VSW Upstrigger port	[Mark_Tim_Junghanns]	[Mark_Tim_Junghanns]	2003-07	NaN
9274	netops-broker	5465	tcp	NETOPS-BROKER	[John_R_Deuel]	[John_R_Deuel]	NaN	NaN
4829	travsoft-ipx-t	2644	udp	Travsoft IPX Tunnel	[Jack_Wilson]	[Jack_Wilson]	NaN	NaN
10165	adi-gxp-srvprt	6769	udp	ADInstruments GxP Server	[Mathew_Pitchforth]	[Mathew_Pitchforth]	2005-08	NaN
4167	siebel-ns	2320	udp	Siebel NS	[Gilberto_Arnaiz]	[Gilberto_Arnaiz]	NaN	NaN
2841	netview-aix-1	1661	tcp	netview-aix-1	NaN	NaN	NaN	NaN
6660	apcupsd	3551	udp	Apcupsd Information Port	[Riccardo_Facchetti]	[Riccardo_Facchetti]	2002-07	NaN
899	siam	498	udp	siam	[Philippe_Gilbert]	[Philippe_Gilbert]	NaN	NaN
241	ident	113	tcp	NaN	NaN	NaN	NaN	NaN

4017	hao	2245	tcp	HaO	[Panic_Ride]	[Panic_Ride]	NaN	NaN
7829	nuauth	4129	udp	NuFW authentication protocol	[Eric_Leblond]	[Eric_Leblond]	2007-06	NaN
396	qft	189	tcp	Queued File Transport	[Wayne_Schroeder]	[Wayne_Schroeder]	NaN	NaN
3030	oracle-em2	1754	tcp	oracle-em2	[Bob_Purvy]	[Bob_Purvy]	NaN	NaN
8702	hfcs-manager	4999	udp	HFSQL Client/Server Database Engine Manager	[PC_SOFT]	[Jerome_AERTS_2]	2006-03-02	2014-02-02
6834	ehp-backup	3638	tcp	EHP Backup Protocol	[Ed_Fair]	[Ed_Fair]	2002-11	NaN
3953	rpi	2214	udp	RDQ Protocol Interface	[Les_Mather]	[Les_Mather]	2005-12	NaN
1351	flexlm	744	udp	Flexible License Manager	[Matt_Christiano]	[Matt_Christiano]	NaN	NaN

In [ ]:

```
# create new dataframe for destination and source traffic
straffic_df = traffic_df[['s_traffic_code', 's_traffic_type']]
dtraffic_df = traffic_df[['d_traffic_code', 'd_traffic_type']]
```

In [ ]:

```
# create new columns in new_df for traffic code using port and protocols and assigning data type
new_df['d_traffic_code'] = new_df['dp'].astype(str) + new_df['pr']
new_df['s_traffic_code'] = new_df['sp'].astype(str) + new_df['pr']
new_df['s_traffic_code'] = new_df['s_traffic_code'].str.lower()
new_df['d_traffic_code'] = new_df['d_traffic_code'].str.lower()
new_df.sample(20)
```

Out[ ]:

	ts	te	td	sa	da	sp	dp	pr	flg	ipkt	...	end_date	end_time
364976	2022-02-08 18:57:00	2022-02-08 18:57:00	0.00	[redacted]		50872	6379	TCP	......S.	1	...	2022-02-08	18:57:00

77271	2022-02-08 18:56:00	2022-02-08 18:59:00	189.96	[redacted]	58273	443	TCP	...AP.S.	19	...	2022-02-08	18:59:00
673796	2022-02-08 18:53:00	2022-02-08 18:53:00	0.00		25939	53	UDP	....	1	...	2022-02-08	18:53:00
448681	2022-02-08 18:54:00	2022-02-08 18:54:00	0.00		50463	443	TCP	...A..S.	13	...	2022-02-08	18:54:00
622525	2022-02-08 18:57:00	2022-02-08 19:00:00	153.04		37149	161	UDP	....	2	...	2022-02-08	19:00:00
562377	2022-02-08 18:45:00	2022-02-08 18:45:00	0.00		52564	53	UDP	....	1	...	2022-02-08	18:45:00
151565	2022-02-08 18:56:00	2022-02-08 18:56:00	2.65		38670	3482	UDP	....	30	...	2022-02-08	18:56:00
626845	2022-02-08 18:45:00	2022-02-08 18:45:00	0.00		46391	53	UDP	....	1	...	2022-02-08	18:45:00
112654	2022-02-08 18:58:00	2022-02-08 18:58:00	0.00		20641	12507	TCP	......S.	1	...	2022-02-08	18:58:00
159908	2022-02-08 18:50:00	2022-02-08 18:50:00	0.00		11562	53	UDP	....	1	...	2022-02-08	18:50:00
6300	2022-02-08 18:39:00	2022-02-08 18:39:00	0.02		63773	443	UDP	....	31	...	2022-02-08	18:39:00
517643	2022-02-08 18:59:00	2022-02-08 18:59:00	0.00		35529	53	UDP	....	1	...	2022-02-08	18:59:00
322043	2022-02-08 18:56:00	2022-02-08 18:56:00	0.00		53615	445	TCP	......S.	1	...	2022-02-08	18:56:00

186806	2022-02-08 18:52:00	2022-02-08 18:52:00	0.02	[redacted]	47721	443	TCP	...A..S.	23	...	2022-02-08	18:52:00
398234	2022-02-08 18:58:00	2022-02-08 18:58:00	0.00		18006	9119	TCP	......S.	1	...	2022-02-08	18:58:00
568161	2022-02-08 18:37:00	2022-02-08 19:00:00	1359.97		60680	443	UDP	....	692	...	2022-02-08	19:00:00
233199	2022-02-08 18:57:00	2022-02-08 18:57:00	0.05		23788	574	TCP	...A..S.	5	...	2022-02-08	18:57:00
89104	2022-02-08 18:48:00	2022-02-08 18:48:00	0.01		24161	443	TCP	...A..S.	10	...	2022-02-08	18:48:00
426760	2022-02-08 18:40:00	2022-02-08 18:40:00	12.24		58686	37143	TCP	......S.	4	...	2022-02-08	18:40:00
285217	2022-02-08 18:38:00	2022-02-08 18:38:00	0.01		39116	53	UDP	....	1	...	2022-02-08	18:38:00

20 rows × 34 columns

In [ ]:

```
# creating new columns for source and destination traffic types by matching traffic codes in traffic_df da
new_df.insert(2,'s_traffic_type', new_df['s_traffic_code'].map(straffic_df.drop_duplicates('s_traffic_code
new_df.insert(3,'d_traffic_type', new_df['d_traffic_code'].map(dtraffic_df.drop_duplicates('d_traffic_code

new_df.sample(20)
```

Out[ ]:

	ts	te	s_traffic_type	d_traffic_type	td	sa	da	sp	dp	pr	...	end
590188	2022-02-08 18:47:00	2022-02-08 18:47:00	NaN	NaN	12.18	[redacted]		58686	7126	TCP	...	202
489476	2022-02-08	2022-02-08	NaN	domain	0.00			7098	53	UDP	...	202

	18:59:00	18:59:00									
266414	2022-02-08 18:39:00	2022-02-08 18:39:00	NaN	domain	0.25	[redacted]	41803	53	UDP	...	202
233981	2022-02-08 18:42:00	2022-02-08 18:42:00	NaN	https	0.00	[redacted]	50329	443	TCP	...	202
27634	2022-02-08 18:50:00	2022-02-08 18:50:00	NaN	domain	0.02	[redacted]	50082	53	UDP	...	202
296293	2022-02-08 19:00:00	2022-02-08 19:00:00	NaN	domain	0.00	[redacted]	59969	53	UDP	...	202
633400	2022-02-08 18:38:00	2022-02-08 18:38:00	NaN	NaN	0.00	[redacted]	56182	52295	TCP	...	202
248078	2022-02-08 18:43:00	2022-02-08 18:44:00	NaN	https	66.00	[redacted]	15739	443	TCP	...	202
264	2022-02-08 18:58:00	2022-02-08 18:58:00	NaN	ident	0.62	[redacted]	47512	113	TCP	...	202
151311	2022-02-08 18:43:00	2022-02-08 18:43:00	NaN	http-alt	5.23	[redacted]	57497	8080	TCP	...	202
219228	2022-02-08 18:50:00	2022-02-08 18:50:00	NaN	EtherNet-IP-2	0.00	[redacted]	60734	44818	TCP	...	202
429939	2022-02-08 18:56:00	2022-02-08 18:56:00	NaN	NaN	0.00	[redacted]	50180	8017	TCP	...	202
16668	2022-02-08 18:35:00	2022-02-08 18:59:00	NaN	NaN	1476.32	[redacted]	52690	8050	TCP	...	202
	2022-	2022-									202

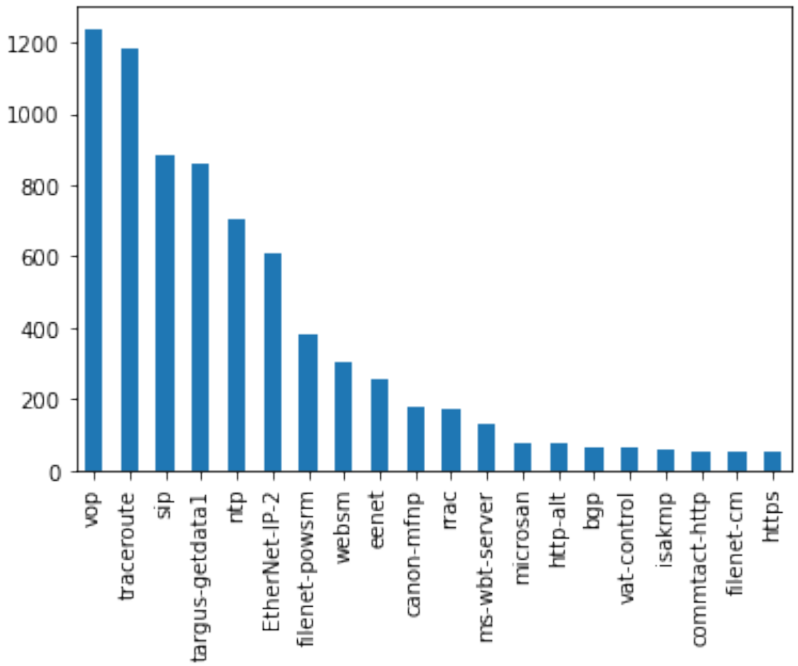
168789	02-08 18:55:00	02-08 18:55:00	NaN	domain	0.00	[redacted]	5788	53	UDP	...	
46634	2022-02-08 18:52:00	2022-02-08 18:52:00	NaN	ntp	0.00	[redacted]	42903	123	UDP	...	202
339458	2022-02-08 18:57:00	2022-02-08 18:57:00	NaN	NaN	12.22	[redacted]	58686	8679	TCP	...	202
436199	2022-02-08 18:42:00	2022-02-08 18:42:00	NaN	snmp	0.00	[redacted]	51107	161	UDP	...	202
20366	2022-02-08 18:49:00	2022-02-08 18:49:00	NaN	https	24.01	[redacted]	57195	443	TCP	...	202
138297	2022-02-08 19:00:00	2022-02-08 19:00:00	NaN	http-alt	0.00	[redacted]	58268	8080	TCP	...	202
53504	2022-02-08 18:47:00	2022-02-08 18:47:00	NaN	NaN	12.17	[redacted]	58686	29507	TCP	...	202

20 rows × 36 columns

In [ ]:

```
# visualising top 20 source traffic types
new_df['s_traffic_type'].value_counts()[:20].plot(kind = 'bar')
```

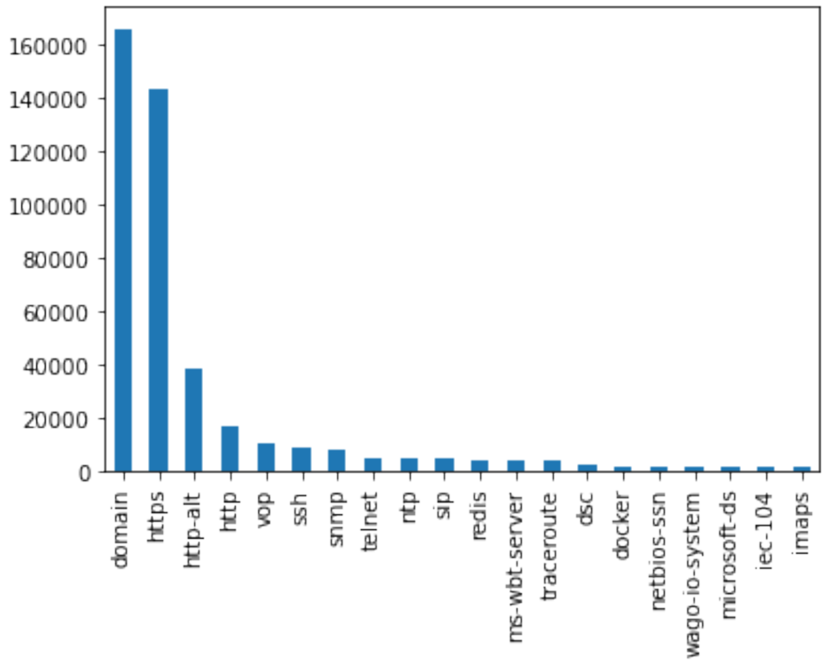
Out[ ]: <AxesSubplot:>



In [ ]:

```
# visualising top 20 destination traffic types
new_df['d_traffic_type'].value_counts()[:20].plot(kind = 'bar')
```

Out[ ]: <AxesSubplot:>



In [ ]:

```
# export analytical file for visualization
new_df.to_csv('new_sample.csv')
```

In [ ]:

```
```

TOTAL BYTES BY UNIQUE IP ADDRESSES

In [ ]:

```
# total bytes by unique clients; source + desitination bytes
new_df.groupby('sa').sum().total_bytes
```

Out[ ]:
```
sa
[redacted]    3680878
                4136
               14532
                1136
                 308
               ...
               11448
               19200
                3772
                  56
                 504
Name: total_bytes, Length: 17071, dtype: int64
```

In [ ]:
```python
totalbytes_df = new_df.groupby('sa', as_index = False)['total_bytes'].sum()
```

In [ ]:
```python
totalbytes_df
```

Out[ ]:

	sa	total_bytes
0	[redacted]	3680878
1		4136
2		14532
3		1136
4		308
...		...
17066		11448
17067		19200
17068		3772
17069		56
17070		504

17071 rows × 2 columns

A 0.0. 0.0 address indicates the client isn't connected to a TCP/IP network, and a device may give itself a 0.0. 0.0 address when it is offline.

1.1. 1.1 is a public DNS resolver operated by Cloudflare that offers a fast and private way to browse the Internet. Unlike most DNS resolvers, 1.1. 1.1 does not sell user data to advertisers.

TOTAL MAC by IP Address

In [ ]:

```
# mac by source IP address; number of macs under each unique IP address
new_df.groupby('sa').count().ismc
```

Out[ ]:

```
sa
[redacted]
     3
     9
     6
     2
     5
    ..
     1
     1
     1
     1
     1
Name: ismc, Length: 17071, dtype: int64
```

In [ ]:

```
sa_df = new_df.groupby('sa').count().ismc
```

In [ ]:

```
# each sa IP address is a unique client, total we have 433 clients using Jame's internet service
sa_df.nunique()
```

Out[ ]:

```
432
```

In [ ]:

```
mac_df = new_df.groupby('sa', as_index = False)['ismc'].count()
```

In [ ]:

```
mac_df.head()
```

Out[ ]:

	sa	ismc
0	[redacted]	3
1		9
2		6
3		2
4		5

In [ ]:

```
mac_df.sort_values(by='ismc', ascending=False)
```

Out[ ]:

	sa	ismc
16648	[redacted]	39205
10057		28481
7171		26740
10098		23105
10022		18818
...		...
2602		1
2601		1
10294		1
2600		1
17070		1

17071 rows × 2 columns

Protoccols and Flags

In [ ]:

```
pr_df = new_df[['sa', 'pr', 'flg', 'total_bytes']].copy()
```

In [ ]:

```
pr_df.head()
```

Out[ ]:

	sa	pr	flg	total_bytes
0	[redacted]	TCP	......S.	60
1	[redacted]	UDP	....	5860
2	[redacted]	TCP	......S.	44
3	[redacted]	UDP	....	1289
4	[redacted]	TCP	...A..S.	11568

In [ ]:

```
# number of unique flags
pr_df.nunique().flg
```

Out[ ]:

```
173
```

In [ ]:

```
# number of unique protoccols
pr_df.nunique().pr
```

Out[ ]:

```
10
```

In [ ]:

```
groupedpr_df = pr_df.groupby('pr', as_index=True).agg({'sa': 'count', 'total_bytes': 'sum'})
groupedpr_df.head(10)
```

Out[ ]:

pr	sa	total_bytes
ESP	72	613211996
GRE	565	1102549077
ICMP	34301	25869330
ICMP6	7	1064
IGMP	1	384
IPIP	2	1317040
IPv6	2	61688
TCP	420727	37698610358
UDP	219210	127129854086
VRRP	35	1629088

In [ ]:

```
# export for visualization
groupedpr_df.to_csv('protoccol.csv')
```

In [ ]:

```
groupedpr_df2 = pr_df.groupby(['pr', 'flg']).agg({'sa': 'count', 'total_bytes': 'sum'})
groupedpr_df2.sample(50)
```

Out[ ]:

pr	flg	sa	total_bytes
TCP	C..A..S.	167	289688
	C.UA..SF	163	165888
	..UA.R.F	1	108
	.EUAP.S.	189	261409
	...A..SF	12894	3230629928

	....PRS.	159	269032
	CE...RSF	166	396240
	..UAPR.F	3	11624
	CEUA..SF	154	339532
	.E.A.RS.	168	288760
	..UAPR..	1	468
	CE...RS.	160	287152
	CE.AP.SF	369	79205646
IPIP	**........**	2	1317040
TCP	**C.....SF**	145	240220
	C.UA.RSF	155	344256
	.EUA..S.	158	152104
	C..AP.S.	177	424525
	...A.R.F	104	38090
	CEU.PRS.	178	333992
	C.UAPR..	2	14607
	.EUA.RS.	177	321552
	CEU.....	1	272
	.EU.PRS.	177	313068
	.E.A..S.	156	286252
	..UA.RS.	180	266615
	....P.S.	150	153148
	..UA..S.	183	251236
	C.UA.RS.	169	238939
	....P.SF	190	326400
	C...PRSF	168	193464

CEU.PRSF	152	325900
.EUA.RSF	162	363207
.E.A...F	1	4439
......S.	137113	13968450
CEUAP...	1	7582
...APR.F	116	341947
..U..RSF	141	226080
.E.AP...	1	2319
C....RS.	168	210676
..UA..SF	3365	3073204
CE.AP.S.	1889	176857737
..U..RS.	156	273992
CEU...SF	168	281088
CEUA.RS.	186	291229
.E..PRSF	149	298520
.EU..RS.	161	253192
...AP.S.	14979	7060109851
.EU.P.S.	174	221740
C.UAPRSF	171	566486

In [ ]:

```
# export for visualization
groupedpr_df2.to_csv('prflg.csv')
```

Drill Down: IP, td, MAC, Total Bytes

In [ ]:

```
new_df.head()
```

Out[ ]:

	ts	te	s_traffic_type	d_traffic_type	td	sa	da	sp	dp	pr	...	end_date	end_
0	2022-02-08 18:54:00	2022-02-08 18:54:00	NaN	redis	0.00	[redacted]		52478	6379	TCP	...	2022-02-08	18:5
1	2022-02-08 18:42:00	2022-02-08 18:42:00	NaN	https	0.08			50841	443	UDP	...	2022-02-08	18:4
2	2022-02-08 18:41:00	2022-02-08 18:41:00	NaN	dynamid	0.00			34247	9002	TCP	...	2022-02-08	18:
3	2022-02-08 18:37:00	2022-02-08 18:37:00	NaN	sip	0.00			57303	5060	UDP	...	2022-02-08	18:
4	2022-02-08 18:43:00	2022-02-08 18:43:00	NaN	https	0.02			32518	443	TCP	...	2022-02-08	18:4

5 rows × 36 columns

In [ ]:

```
# new dataframe
df2 = new_df[['sa', 'total_bytes', 'td', 'ismc']].copy()
```

In [ ]:

```
df2.head()
```

Out[ ]:

	sa	total_bytes	td	ismc
0	[redacted]	60	0.00	[redacted]
1		5860	0.08	
2		44	0.00	
3		1289	0.00	
4		11568	0.02	

In [ ]:

```
df2.dtypes
```

Out[ ]:

```
sa              object
total_bytes      int64
td             float64
ismc            object
dtype: object
```

In [ ]:

```
# new aggregated dataframe
df3 = df2.groupby('sa', as_index=True).agg({'total_bytes': 'sum', 'td':'sum', 'ismc': 'count'})
```

In [ ]:

```
df3.head()
```

Out[ ]:

	total_bytes	td	ismc
sa			
[redacted]	3680878	4214.06	3
	4136	1368.40	9
	14532	3840.02	6
	1136	3.05	2
	308	673.00	5

In [ ]:

```
# renaming columns
df3.rename(columns = {'td':'total_td', 'ismc':'count_ismc'}, inplace = True)
```

In [ ]:

```
df3.head()
```

Out[ ]:

	total_bytes	total_td	count_ismc
sa			
[redacted]	3680878	4214.06	3
	4136	1368.40	9
	14532	3840.02	6
	1136	3.05	2
	308	673.00	5

In [ ]:

```
# average bytes per mac in a IP
df3['avg_bytes_per_mac'] = df3['total_bytes']/df3['count_ismc']
```

In [ ]:

```
df3.head()
```

Out[ ]:

	total_bytes	total_td	count_ismc	avg_bytes_per_mac
sa				
[redacted]	3680878	4214.06	3	1.226959e+06
	4136	1368.40	9	4.595556e+02
	14532	3840.02	6	2.422000e+03
	1136	3.05	2	5.680000e+02
	308	673.00	5	6.160000e+01

In [ ]:
```
df3.dtypes
```

Out[ ]:
```
total_bytes             int64
total_td              float64
count_ismc              int64
avg_bytes_per_mac     float64
dtype: object
```

In [ ]:
```
df3['bytes_per_second']=df3['total_bytes']/df3['total_td']
```

In [ ]:
```
df3.sample(50)
```

Out[ ]:

	total_bytes	total_td	count_ismc	avg_bytes_per_mac	bytes_per_second
sa					
[redacted]	231	0.36	1	231.000000	6.416667e+02
	213	0.40	1	213.000000	5.325000e+02
	180	0.00	3	60.000000	inf
	116	3.00	1	116.000000	3.866667e+01
	1735	0.01	10	173.500000	1.735000e+05
	120	0.00	3	40.000000	inf
	108	0.99	1	108.000000	1.090909e+02
	665522941	55397.64	1509	441035.746190	1.201356e+04
	429	0.12	2	214.500000	3.575000e+03
	54119	74.87	15	3607.933333	7.228396e+02
	108	3.00	1	108.000000	3.600000e+01
	3696	8.00	9	410.666667	4.620000e+02
	668	3.01	11	60.727273	2.219269e+02
	3756	36.98	19	197.684211	1.015684e+02

[redacted]	116	0.00	1	116.000000	inf
	2437	1.03	7	348.142857	2.366019e+03
	129	0.00	1	129.000000	inf
	3340	0.00	47	71.063830	inf
	104	0.00	2	52.000000	inf
	46500	1533.98	340	136.764706	3.031330e+01
	4788	0.03	1	4788.000000	1.596000e+05
	40	0.00	1	40.000000	inf
	44	0.00	1	44.000000	inf
	176	0.00	2	88.000000	inf
	629	0.79	3	209.666667	7.962025e+02
	2400	49.09	10	240.000000	4.888979e+01
	75224	42.72	9	8358.222222	1.760861e+03
	30	0.00	1	30.000000	inf
	52	0.00	1	52.000000	inf
	282	0.00	2	141.000000	inf
	216	0.00	4	54.000000	inf
	80	0.03	1	80.000000	2.666667e+03
	910	0.05	6	151.666667	1.820000e+04
	40	0.00	1	40.000000	inf
	1312	48.88	18	72.888889	2.684124e+01
	33417	1186.97	1	33417.000000	2.815320e+01
	358	0.05	2	179.000000	7.160000e+03
	1506	1.36	7	215.142857	1.107353e+03
	260	5.99	3	86.666667	4.340568e+01
	15792	0.01	3	5264.000000	1.579200e+06

[redacted]	286	0.98	1	286.000000	2.918367e+02
	680	81.23	9	75.555556	8.371291e+00
	3062	5.97	44	69.590909	5.128978e+02
	40	0.00	1	40.000000	inf
	3512	114.03	14	250.857143	3.079891e+01
	29027	23.75	7	4146.714286	1.222189e+03
	140	0.01	1	140.000000	1.400000e+04
	33964	79.59	6	5660.666667	4.267370e+02
	7224	22.94	2	3612.000000	3.149085e+02
	40	0.00	1	40.000000	inf

In [ ]:

```
# export processed data for visualization
df3.to_csv('data.csv')
```

In [ ]:

```
df3.sort_values(by=['total_bytes'], ascending=False, inplace = True)
df3.sample(50)
```

Out[ ]:

	total_bytes	total_td	count_ismc	avg_bytes_per_mac	bytes_per_second
sa					
[redacted]	120	0.00	2	6.000000e+01	inf
	396	0.00	11	3.600000e+01	inf
	156	0.00	1	1.560000e+02	inf
	256	0.00	4	6.400000e+01	inf
	220	0.00	5	4.400000e+01	inf
	120	3.00	1	1.200000e+02	4.000000e+01
	438784	275.20	7	6.268343e+04	1.594419e+03

[redacted]	844	48.88	9	9.377778e+01	1.726678e+01
[redacted]	129	0.00	1	1.290000e+02	inf
[redacted]	40	0.00	1	4.000000e+01	inf
[redacted]	668	3.00	15	4.453333e+01	2.226667e+02
[redacted]	222	0.26	1	2.220000e+02	8.538462e+02
[redacted]	1692	0.00	3	5.640000e+02	inf
[redacted]	3348	83.04	12	2.790000e+02	4.031792e+01
[redacted]	30	0.00	1	3.000000e+01	inf
[redacted]	5966	12.30	73	8.172603e+01	4.850407e+02
[redacted]	6677	10.53	1	6.677000e+03	6.340931e+02
[redacted]	60	0.00	1	6.000000e+01	inf
[redacted]	5024	61.04	26	1.932308e+02	8.230668e+01
[redacted]	33045	3607.91	8	4.130625e+03	9.159042e+00
[redacted]	2580	0.00	43	6.000000e+01	inf
[redacted]	98070	99.34	27	3.632222e+03	9.872156e+02
[redacted]	48	0.00	1	4.800000e+01	inf
[redacted]	40	0.00	1	4.000000e+01	inf
[redacted]	345	0.05	2	1.725000e+02	6.900000e+03
[redacted]	52088559	2814.16	2	2.604428e+07	1.850945e+04
[redacted]	44	0.00	1	4.400000e+01	inf
[redacted]	1561736	319.29	5	3.123472e+05	4.891278e+03
[redacted]	100	0.00	1	1.000000e+02	inf
[redacted]	40	0.00	1	4.000000e+01	inf
[redacted]	360	31.51	1	3.600000e+02	1.142494e+01
[redacted]	129	0.00	1	1.290000e+02	inf
[redacted]	44	0.00	1	4.400000e+01	inf

[redacted]	680	81.23	9	7.555556e+01	8.371291e+00
	45294	2806.02	2	2.264700e+04	1.614172e+01
	80	0.00	2	4.000000e+01	inf
	160	2.99	2	8.000000e+01	5.351171e+01
	460	2.93	1	4.600000e+02	1.569966e+02
	1643	48.97	16	1.026875e+02	3.355115e+01
	744	0.01	16	4.650000e+01	7.440000e+04
	40	0.00	1	4.000000e+01	inf
	44	0.00	1	4.400000e+01	inf
	2084	0.00	31	6.722581e+01	inf
	112108	92.60	1	1.121080e+05	1.210670e+03
	936	36.62	14	6.685714e+01	2.555980e+01
	591	0.03	4	1.477500e+02	1.970000e+04
	184	0.00	1	1.840000e+02	inf
	77688	127.60	772	1.006321e+02	6.088401e+02
	441712	9631.98	54	8.179852e+03	4.585890e+01
	1163	24.45	10	1.163000e+02	4.756646e+01

In [ ]:

```
df3.describe()
```

```
/opt/anaconda3/lib/python3.8/site-packages/numpy/lib/function_base.py:3961: RuntimeWarning: invalid value
encountered in subtract
  diff_b_a = subtract(b, a)
```

Out[ ]:

	total_bytes	total_td	count_ismc	avg_bytes_per_mac	bytes_per_second
count	1.707100e+04	1.707100e+04	17071.000000	1.707100e+04	1.707100e+04
mean	9.757665e+06	5.079986e+03	39.536172	3.309739e+06	inf
std	2.201133e+08	2.890629e+05	619.693841	8.619605e+07	NaN
min	2.800000e+01	0.000000e+00	1.000000	2.800000e+01	6.090258e-02
25%	1.150000e+02	0.000000e+00	1.000000	5.600000e+01	1.004909e+02
50%	3.350000e+02	5.000000e-02	2.000000	1.329457e+02	1.240000e+04
75%	1.509500e+03	1.801000e+01	6.000000	2.730000e+02	NaN
max	2.053199e+10	2.643703e+07	39205.000000	6.591538e+09	inf

Correlation

Using new sample dataframe (df3) total bytes, duration, count of ismc, and average bytes per ismc

In [ ]:

```
import numpy as np
import seaborn as sns
import matplotlib.pylab as plt
corr = df3.corr()
corr
```

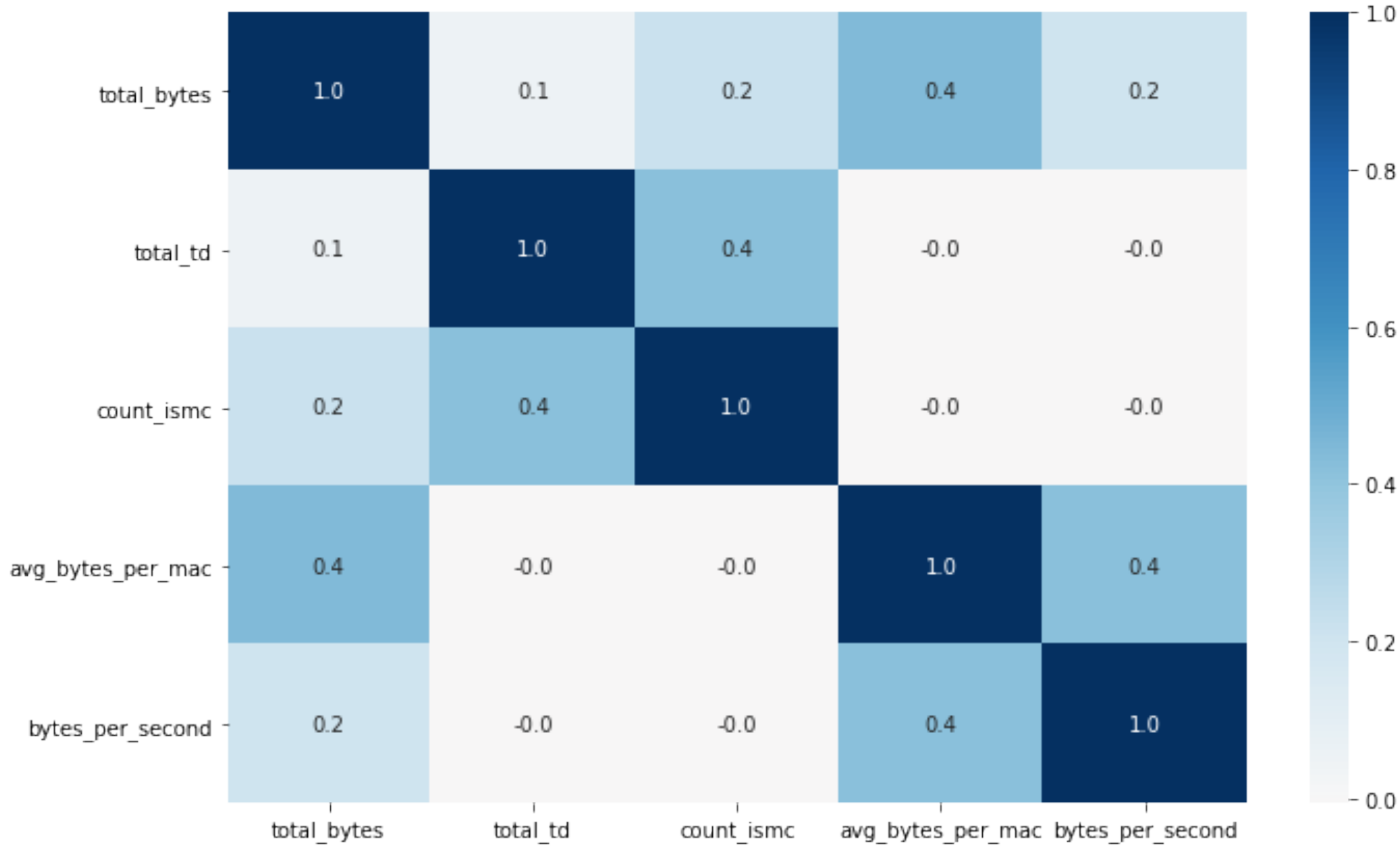
Out[ ]:

	total_bytes	total_td	count_ismc	avg_bytes_per_mac	bytes_per_second
total_bytes	1.000000	0.053371	0.224845	0.441871	0.210322
total_td	0.053371	1.000000	0.419648	-0.000434	-0.001953
count_ismc	0.224845	0.419648	1.000000	-0.002267	-0.005277
avg_bytes_per_mac	0.441871	-0.000434	-0.002267	1.000000	0.422262
bytes_per_second	0.210322	-0.001953	-0.005277	0.422262	1.000000

In [ ]:

```
#heatmap using seaborn
#If the correlation between variables if greater than 0.7 we can say that the two variables are highly cor
#From the above table, the pairs of highly correlated variables are:
fig, ax = plt.subplots()
fig.set_size_inches(11, 7)
sns.heatmap(corr, annot=True, fmt=".1f", cmap="RdBu", center=0, ax=ax)
```

Out[ ]: <AxesSubplot:>



For Additional Visualization: Distribution of Total Bytes

In [ ]:

```
df_dist = df2
df_dist.head()
```

Out[ ]:

	sa	total_bytes	td	ismc
0	[redacted]	60	0.00	c4:ad:34:51:33:93
1		5860	0.08	00:00:00:00:00:00
2		44	0.00	c4:ad:34:51:33:93
3		1289	0.00	c4:ad:34:51:33:93
4		11568	0.02	00:00:5e:00:01:0b

In [ ]:

```
df_dist2 = df_dist.groupby(['ismc', 'sa']).agg({'total_bytes': 'sum'})
df_dist2.sample(30)
```

Out[ ]:

		total_bytes
ismc	sa	
[redacted]		52
		40
		6840
		843
		1334
		1176
		84
		40
		277
		40
		222

[redacted]

	44
	480
	40
	932
	120
	40
	6398
	1389997
	902278887
	1892
	33480
	12108
	40
	1218
	291
	908
	108
	180
	44

In [ ]:

```
# export processed data for visualization
df_dist2.to_csv('dist2.csv')
```

Segmentation

In [ ]:

```
!pip install squarify
```

```
Requirement already satisfied: squarify in /opt/anaconda3/lib/python3.8/site-packages (0.4.3)
WARNING: You are using pip version 22.0.3; however, version 22.0.4 is available.
You should consider upgrading via the '/opt/anaconda3/bin/python -m pip install --upgrade pip' command.
```

In [ ]:

```
#Import libraries
%matplotlib inline
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import squarify
```

Segmentation Modeling

In [ ]:

```
# Frequency = count of ts occurrences
# Usage = Total bytes in and out
# MAC = count of ismc


#Create Segmentation Modelling scores for each customer
Scores = new_df.groupby('sa').agg({'ismc': lambda x: len(set(x)), 'ts': lambda x: len(set(x)), 'total_byte

#Convert Invoice Date into type int
Scores['ts'] = Scores['ts'].astype(int)

#Rename column names to Usage, Frequency and MAC
Scores.rename(columns={'ts': 'Frequency',
                        'total_bytes': 'Usage',
                        'ismc': 'MAC'}, inplace=True)

Scores.reset_index().head()
```

Out[ ]:

	sa	MAC	Frequency	Usage
0	[redacted]	2	2	3680878
1		2	5	4136
2		3	4	14532
3		1	2	1136
4		1	5	308

In [ ]:

```
#Descriptive Statistics (Usage)
Scores.describe()
```

Out[ ]:

	MAC	Frequency	Usage
count	17071.000000	17071.000000	1.707100e+04
mean	1.259329	3.903755	9.757665e+06
std	0.730094	5.616179	2.201133e+08
min	1.000000	1.000000	2.800000e+01
25%	1.000000	1.000000	1.150000e+02
50%	1.000000	1.000000	3.350000e+02
75%	1.000000	4.000000	1.509500e+03
max	10.000000	27.000000	2.053199e+10

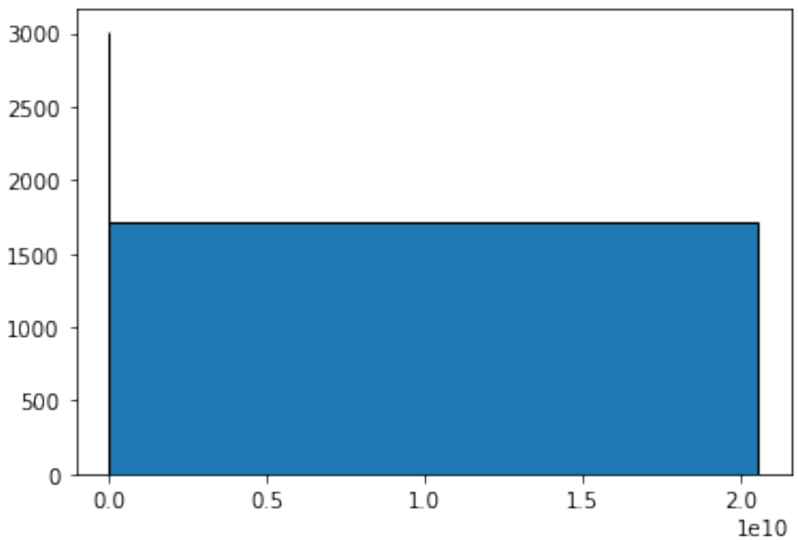
In [ ]:

```
#define function to calculate equal-frequency bins
x = Scores['Usage']

def equalObs(x, nbin):
    nlen = len(x)
    return np.interp(np.linspace(0, nlen, nbin + 1),
                     np.arange(nlen),
                     np.sort(x))

#create histogram with equal-frequency bins
n, bins, patches = plt.hist(x, equalObs(x, 10), edgecolor='black')
plt.show()

#display bin boundaries and frequency per bin
bins, n
```



Out[ ]:

```
(array([2.80000000e+01, 4.00000000e+01, 8.00000000e+01, 1.32000000e+02,
        2.07000000e+02, 3.35000000e+02, 5.64000000e+02, 1.18070000e+03,
        2.92320000e+03, 1.57852000e+04, 2.05319945e+10]),
 array([ 118., 3012., 1952., 1731., 1721., 1670., 1746., 1707., 1707.,
        1707.]))
```

In [ ]:

```
df = pd.DataFrame(data = n)
print(df)
df.to_csv('usage_freq.csv')
```

```
        0
0   118.0
1  3012.0
2  1952.0
3  1731.0
4  1721.0
5  1670.0
6  1746.0
7  1707.0
8  1707.0
9  1707.0
```

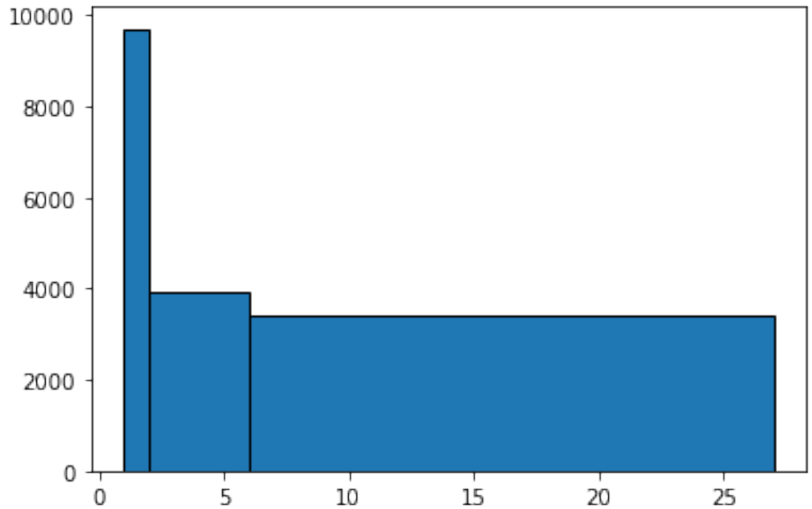
In [ ]:

```
#define function to calculate equal-frequency bins
x = Scores['Frequency']

def equalObs(x, nbin):
    nlen = len(x)
    return np.interp(np.linspace(0, nlen, nbin + 1),
                     np.arange(nlen),
                     np.sort(x))

#create histogram with equal-frequency bins
n, bins, patches = plt.hist(x, equalObs(x, 5), edgecolor='black')
plt.show()

#display bin boundaries and frequency per bin
bins, n
```



Out[ ]:
```
(array([ 1.,  1.,  1.,  2.,  6., 27.]),
 array([   0.,     0., 9703., 3945., 3423.]))
```

In [ ]:
```
df = pd.DataFrame(data = bins)

print(df)
df.to_csv('frequency_freq.csv')
```

```
      0
0   1.0
1   1.0
2   1.0
3   2.0
4   6.0
5  27.0
```

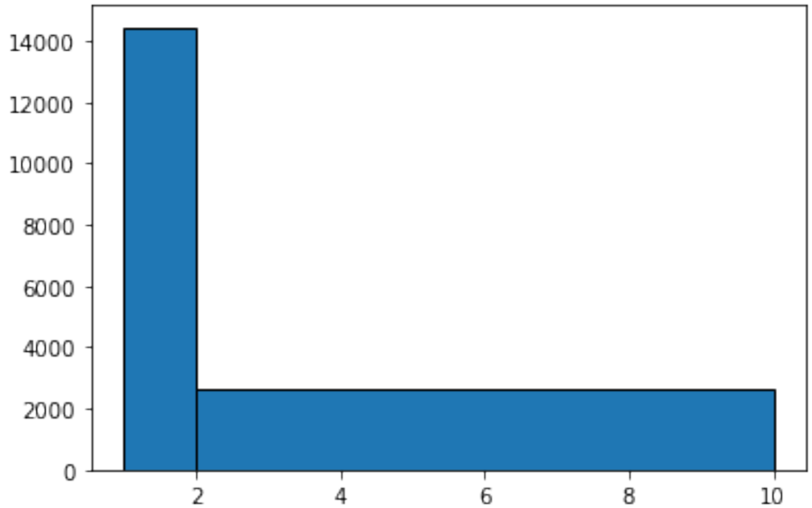
In [ ]:

```
#define function to calculate equal-frequency bins
x = Scores['MAC']

def equalObs(x, nbin):
    nlen = len(x)
    return np.interp(np.linspace(0, nlen, nbin + 1),
                     np.arange(nlen),
                     np.sort(x))

#create histogram with equal-frequency bins
n, bins, patches = plt.hist(x, equalObs(x, 10), edgecolor='black')
plt.show()

#display bin boundaries and frequency per bin
bins, n
```



Out[ ]:

```
(array([ 1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  2., 10.]),
 array([    0.,     0.,     0.,     0.,     0.,     0.,     0.,     0.,
        14445.,  2626.]))
```

In [ ]:

```
df = pd.DataFrame(data = bins)

print(df)
df.to_csv('mac_freq.csv')
```

```
       0
0    1.0
1    1.0
2    1.0
3    1.0
4    1.0
5    1.0
6    1.0
7    1.0
8    1.0
9    2.0
10  10.0
```

In [ ]:

```
#Split into four segments using quantiles
quantiles = Scores.quantile(q=[0.25,0.5,0.75])
quantiles = quantiles.to_dict()
```

In [ ]:

```
quantiles
```

Out[ ]:

```
{'MAC': {0.25: 1.0, 0.5: 1.0, 0.75: 1.0},
 'Frequency': {0.25: 1.0, 0.5: 1.0, 0.75: 4.0},
 'Usage': {0.25: 115.0, 0.5: 335.0, 0.75: 1509.5}}
```

In [ ]:

```
# Functions to create Usage, Frequency and MAC segments
def UFMScoring(x,p,d):
    if x <= d[p][0.25]:
        return 1
    elif x <= d[p][0.50]:
        return 2
    elif x <= d[p][0.75]:
        return 3
    else:
        return 4
```

In [ ]:

```
#Calculate and Add U, F and M segment value columns in the existing dataset to show U, F and M segment val
Scores['U'] = Scores['Usage'].apply(UFMScoring, args=('Usage',quantiles,))
Scores['F'] = Scores['Frequency'].apply(UFMScoring, args=('Frequency',quantiles,))
Scores['M'] = Scores['MAC'].apply(UFMScoring, args=('MAC',quantiles,))
Scores.head()
```

Out[ ]:

sa	MAC	Frequency	Usage	U	F	M
[redacted]	2	2	3680878	4	3	4
	2	5	4136	4	4	4
	3	4	14532	4	3	4
	1	2	1136	3	3	1
	1	5	308	2	4	1

In [ ]:

```
#Calculate and Add UFMGroup value column showing combined concatenated score of UFM
Scores['UFMGroup'] = Scores.U.map(str) + Scores.F.map(str) + Scores.M.map(str)

#Calculate and Add UFMScore value column showing total sum of UFMGroup values
Scores['UFMScore'] = Scores[['U', 'F', 'M']].sum(axis = 1)
Scores.head()
```

Out[ ]:

sa	MAC	Frequency	Usage	U	F	M	UFMGroup	UFMScore
[redacted]	2	2	3680878	4	3	4	434	11
	2	5	4136	4	4	4	444	12
	3	4	14532	4	3	4	434	11
	1	2	1136	3	3	1	331	7
	1	5	308	2	4	1	241	7

In [ ]:

```
#Assign Value Level to each customer
Value_Level = ['No Value', 'Low', 'Medium', 'High']
Score_cuts = pd.qcut(Scores.UFMScore, q = 4, labels = Value_Level)
Scores['UFM_Value_Level'] = Score_cuts.values
Scores.reset_index().head()
```

Out[ ]:

	sa	MAC	Frequency	Usage	U	F	M	UFMGroup	UFMScore	UFM_Value_Level
0	[redacted]	2	2	3680878	4	3	4	434	11	High
1	[redacted]	2	5	4136	4	4	4	444	12	High
2	[redacted]	3	4	14532	4	3	4	434	11	High
3	[redacted]	1	2	1136	3	3	1	331	7	Medium
4	[redacted]	1	5	308	2	4	1	241	7	Medium

In [ ]:

```
#Validate the data for UFMGroup = 111
Scores[Scores['UFMGroup']=='111'].sort_values('Usage', ascending=False).reset_index().head(10)
```

Out[ ]:

	sa	MAC	Frequency	Usage	U	F	M	UFMGroup	UFMScore	UFM_Value_Level
0	[redacted]	1	1	114	1	1	1	111	3	No Value
1	[redacted]	1	1	113	1	1	1	111	3	No Value
2	[redacted]	1	1	113	1	1	1	111	3	No Value
3	[redacted]	1	1	112	1	1	1	111	3	No Value
4	[redacted]	1	1	112	1	1	1	111	3	No Value
5	[redacted]	1	1	112	1	1	1	111	3	No Value
6	[redacted]	1	1	112	1	1	1	111	3	No Value
7	[redacted]	1	1	112	1	1	1	111	3	No Value
8	[redacted]	1	1	112	1	1	1	111	3	No Value
9	[redacted]	1	1	112	1	1	1	111	3	No Value

In [ ]:

```
# Define ufm_level function
def ufm_level(df):
    if df['UFMScore'] >= 10:
        return 'Require Upgrade'
    elif ((df['UFMScore'] >= 7) and (df['UFMScore'] < 10)):
        return 'Potential Sales'
    elif ((df['UFMScore'] >= 4) and (df['UFMScore'] < 7)):
        return 'Needs Attention'
    else:
        return 'Possible Customer Loss'

# Create a new variable UFM_Level
Scores['UFM_Level'] = Scores.apply(ufm_level, axis=1)
# Print the header with top 5 rows to the console
Scores.head()
```

Out[ ]:

	MAC	Frequency	Usage	U	F	M	UFMGroup	UFMScore	UFM_Value_Level	UFM_Level
sa										
[redacted]	2	2	3680878	4	3	4	434	11	High	Require Upgrade
	2	5	4136	4	4	4	444	12	High	Require Upgrade
	3	4	14532	4	3	4	434	11	High	Require Upgrade
	1	2	1136	3	3	1	331	7	Medium	Potential Sales
	1	5	308	2	4	1	241	7	Medium	Potential Sales

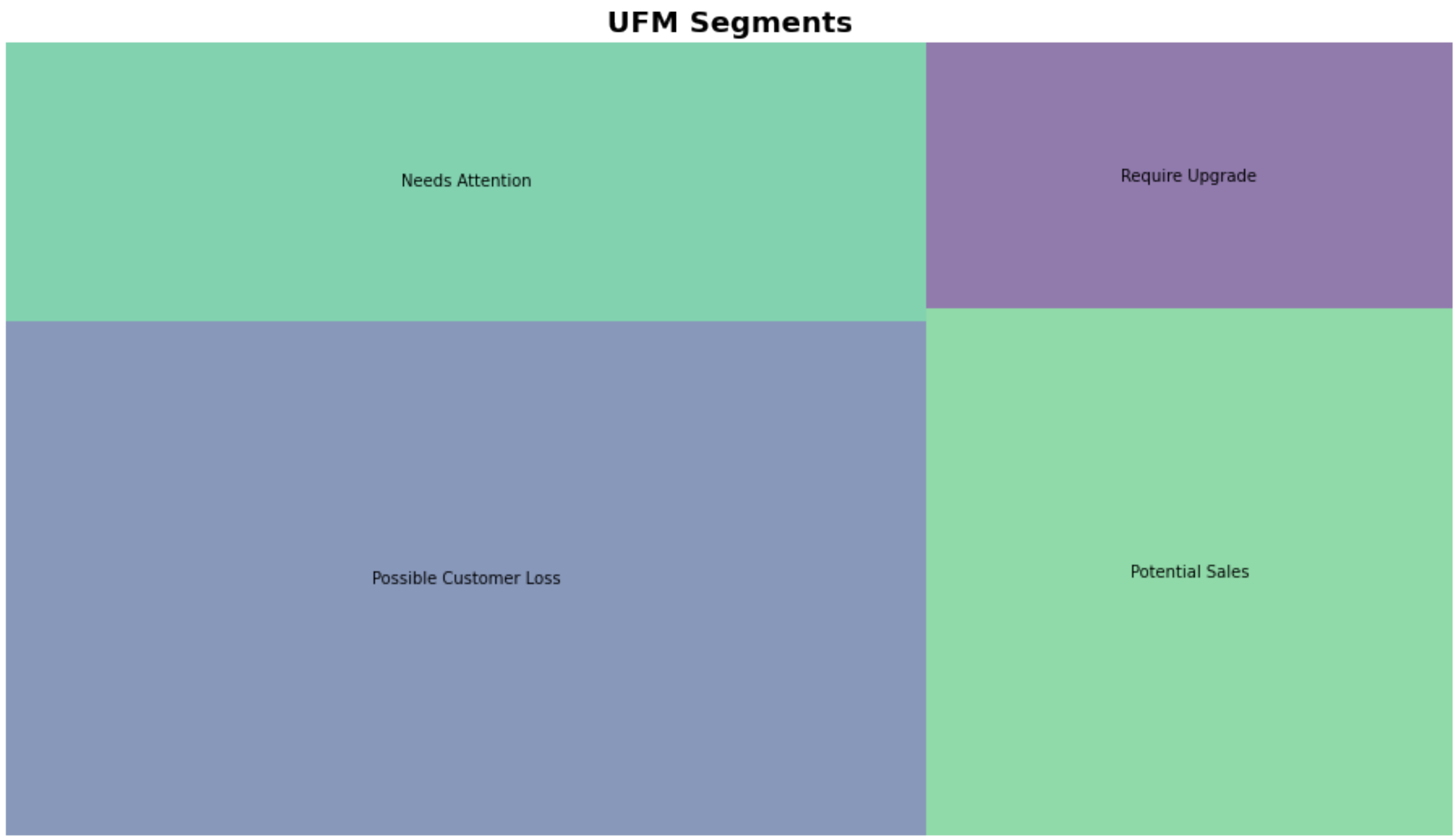
In [ ]:

```
# Calculate average values for each UFM_Level, and return a size of each segment
ufm_level_agg = Scores.groupby('UFM_Level').agg({
    'Usage': 'mean',
    'Frequency': 'mean',
    'MAC': ['mean', 'count']
}).round(1)
# Print the aggregated dataset
print(ufm_level_agg)
```

```
                           Usage Frequency  MAC      
                            mean      mean mean count
UFM_Level                                            
Needs Attention        6663759.4       1.3  1.0  7039
Possible Customer Loss      52.8       1.0  1.0  3816
Potential Sales        7319169.4       7.3  1.1  4121
Require Upgrade       42722865.4      11.3  2.8  2095
```

In [ ]:

```
ufm_level_agg.columns = ['UsageMean','FrequencyMean','MACMean', 'Count']
#Create our plot and resize it.
fig = plt.gcf()
ax = fig.add_subplot()
fig.set_size_inches(16, 9)
squarify.plot(sizes=ufm_level_agg['Count'],
              label=['Possible Customer Loss',
                     'Needs Attention',
                     'Potential Sales',
                     'Require Upgrade'], alpha=.6 )
plt.title("UFM Segments",fontsize=18,fontweight="bold")
plt.axis('off')
plt.show()
```



In [ ]:
```
Scores.to_csv('segments.csv')
```

In [ ]: