

Outputs:

1)

Now, let us try to explore the data and try to understand what data tells us.

```
In [3]: df.shape
```

```
Out[3]: (90099, 27)
```

In our dataset there are **90099** rows and **27** cols.

```
In [4]: # just see how our dataset looks like
df.head()
```

```
Out[4]:
```

	click	C1	banner_pos	site_id	site_domain	site_category	app_id	app_domain	app_category	device_id	device_is	C17	C18	C19	C20	C21	month
0	False	1005	1	830e63d1	5ba89e43	8026772b	ecad2389	7801e6b8	07d7822	a89d214a	9d2e8233	2528	0	38	100075	201	10
1	True	1005	1	e151e045	7a081013	8026772b	ecad2389	7801e6b8	07d7822	a89d214a	9d2e8233	1834	2	38	-1	18	10
2	False	1005	0	43c0809a	d8232f1e	280056bd	ecad2389	7801e6b8	07d7822	a89d214a	9d2e8233	2552	3	187	100002	22	10
3	False	1002	0	0a94453	248e438f	58e219e0	ecad2389	7801e6b8	07d7822	6a87881	9b8e83da	2478	3	187	100074	33	10
4	True	1005	0	1ba0719a	0945787f	280056bd	ecad2389	7801e6b8	07d7822	a89d214a	9d2e8233	1722	0	38	-1	18	10

5 rows x 27 columns

We can see different types of cols and what type of values that col poses.

If we observe, there is a **click** col which has boolean value.
But our last column is **month** which is a numerical value which tells us when 1 is there it is true user clicks on Ads, otherwise it is 0.
Actually, both **click** col and **month** col are telling us same, one is enough.

2)

So, we don't need **click** just drop it from our data set.

```
In [5]: # drop click col
df.drop(['click'], axis=1, inplace=True)
```

```
In [6]: df
```

```
Out[6]:
```

	C1	banner_pos	site_id	site_domain	site_category	app_id	app_domain	app_category	device_id	device_is	C17	C18	C19	C20	C21	month
0	1005	1	830e63d1	5ba89e43	8026772b	ecad2389	7801e6b8	07d7822	a89d214a	9d2e8233	2528	0	38	100075	201	10
1	1005	1	e151e045	7a081013	8026772b	ecad2389	7801e6b8	07d7822	a89d214a	9d2e8233	1834	2	38	-1	18	10
2	1005	0	43c0809a	d8232f1e	280056bd	ecad2389	7801e6b8	07d7822	a89d214a	9d2e8233	2552	3	187	100002	22	10
3	1002	0	0a94453	248e438f	58e219e0	ecad2389	7801e6b8	07d7822	6a87881	9b8e83da	2478	3	187	100074	33	10
4	1005	0	1ba0719a	0945787f	280056bd	ecad2389	7801e6b8	07d7822	a89d214a	9d2e8233	1722	0	38	-1	18	10

90099 rows x 26 columns

See **click** col is dropped.

```
In [7]: # just see what all columns are there.
```

3)

The screenshot shows a Jupyter Notebook interface with the following code and output:

```
In [7]: # just see what all columns are there.
df.columns
```

```
Out[7]: Index(['C1', 'banner_pos', 'site_id', 'site_domain', 'site_category', 'app_id',
             'app_domain', 'app_category', 'device_id', 'device_ip', 'device_model',
             'device_type', 'device_com_type', 'C14', 'C15', 'C16', 'C17', 'C18',
             'C19', 'C20', 'C21', 'month', 'dayofweek', 'day', 'hour', 'y'],
             dtype=object)
```

```
In [8]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 99999 entries, 0 to 99998
Data columns (total 26 columns):
#   Column              Non-Null Count  Dtype
---  -
0   C1                   99999 non-null  int64
1   banner_pos          99999 non-null  int64
2   site_id             99999 non-null  object
3   site_domain         99999 non-null  object
4   site_category       99999 non-null  object
5   app_id              99999 non-null  object
6   app_domain          99999 non-null  object
7   app_category        99999 non-null  object
8   device_id           99999 non-null  object
9   device_ip           99999 non-null  object
10  device_model        99999 non-null  object
11  device_type         99999 non-null  int64
12  device_com_type     99999 non-null  int64
13  C14                 99999 non-null  int64
14  C15                 99999 non-null  int64
15  C16                 99999 non-null  int64
16  C17                 99999 non-null  int64
17  C18                 99999 non-null  int64
```

4)

The screenshot shows the same Jupyter Notebook interface after cleaning the data. The code and output are as follows:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 99999 entries, 0 to 99998
Data columns (total 26 columns):
#   Column              Non-Null Count  Dtype
---  -
0   C1                   99999 non-null  int64
1   banner_pos          99999 non-null  int64
2   site_id             99999 non-null  object
3   site_domain         99999 non-null  object
4   site_category       99999 non-null  object
5   app_id              99999 non-null  object
6   app_domain          99999 non-null  object
7   app_category        99999 non-null  object
8   device_id           99999 non-null  object
9   device_ip           99999 non-null  object
10  device_model        99999 non-null  object
11  device_type         99999 non-null  int64
12  device_com_type     99999 non-null  int64
13  C14                 99999 non-null  int64
14  C15                 99999 non-null  int64
15  C16                 99999 non-null  int64
16  C17                 99999 non-null  int64
17  C18                 99999 non-null  int64
18  C19                 99999 non-null  int64
19  C20                 99999 non-null  int64
20  C21                 99999 non-null  int64
21  month               99999 non-null  int64
22  dayofweek           99999 non-null  int64
23  day                 99999 non-null  int64
24  hour                99999 non-null  int64
25  y                   99999 non-null  int64
dtypes: int64(17), object(9)
memory usage: 19.6+ MB
```

5)

```
In [9]: # to check some statistics of the numerical data we have.
df.describe()
```

Out[9]:

	C1	banner_pos	device_type	device_conn_type	C14	C15	C16	C17	C18	C19
count	99999.000000	99999.000000	99999.000000	99999.000000	99999.000000	99999.000000	99999.000000	99999.000000	99999.000000	99999.000000
mean	1004.957550	0.289943	1.01629	0.329453	18957.231732	318.834548	60.278743	2114.613036	1.439904	227.234932
std	1.091916	0.505707	0.53110	0.854573	4944.919482	21.510752	47.713436	607.489442	1.326824	351.472385
min	1001.000000	0.000000	0.00000	0.000000	375.000000	120.000000	20.000000	112.000000	0.000000	33.000000
25%	1005.000000	0.000000	1.00000	0.000000	18920.000000	320.000000	50.000000	1863.000000	0.000000	35.000000
50%	1005.000000	0.000000	1.00000	0.000000	20346.000000	320.000000	50.000000	2323.000000	2.000000	38.000000
75%	1005.000000	1.000000	1.00000	0.000000	21893.000000	320.000000	50.000000	2526.000000	3.000000	171.000000
max	1012.000000	7.000000	5.00000	5.000000	24043.000000	1024.000000	1024.000000	2757.000000	3.000000	1839.000000

6)

jupyter Click_On_Ads Last Checkpoint: 5 minutes ago (autosaved)

We do some data analysis now.

```
In [18]: # to check missing values
df.isnull().sum()
```

Out[18]:

C1	0
banner_pos	0
site_id	0
site_domain	0
site_category	0
app_id	0
app_domain	0
app_category	0
device_id	0
device_ip	0
device_model	0
device_type	0
device_conn_type	0
C14	0
C15	0
C16	0
C17	0
C18	0
C19	0
C20	0
C21	0
month	0
dayofweek	0
day	0
hour	0
Y	0

No missing values.

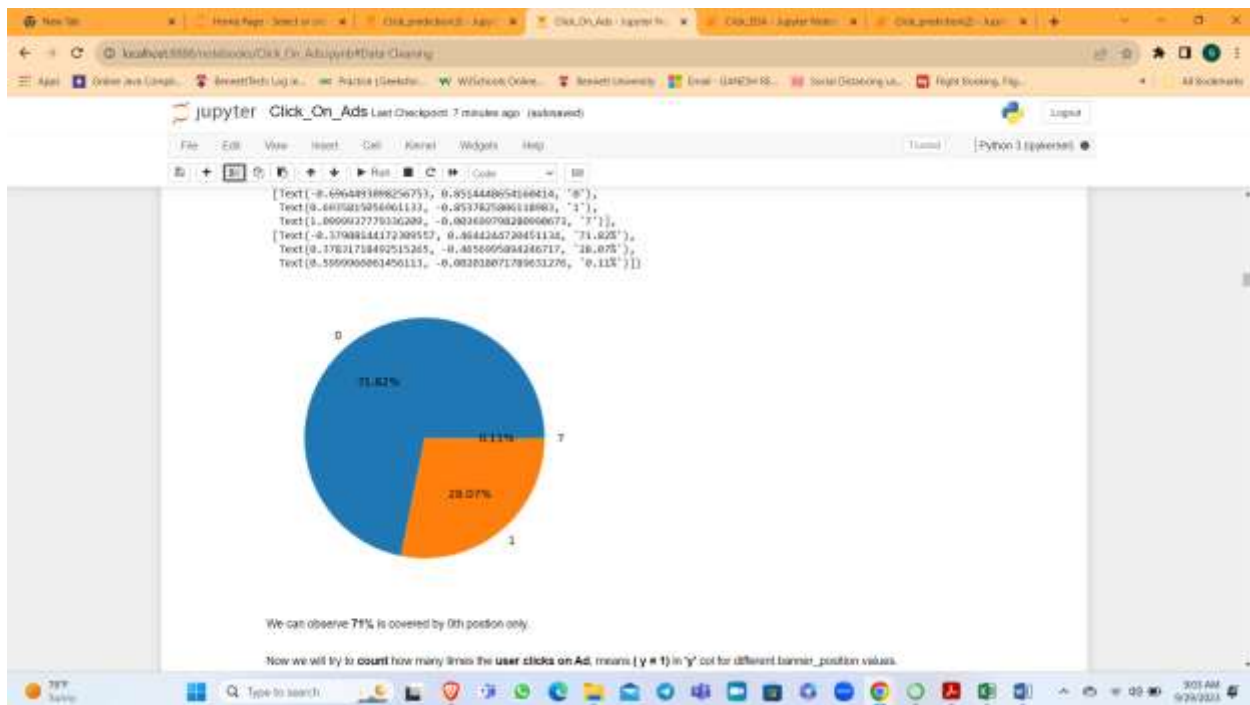
7)

Now we will explore some **relations** between our **cols** and our **target variable(y)**

```
In [12]: # We check relation of banner position with respect to y
df.banner_pos.value_counts()
```

```
Out[12]: 0    71778
         1    28052
         7     107
         2     30
         4     19
         5      9
         3      4
         Name: banner_pos, dtype: int64
```

8)



9)

Now we will try to **count** how many times the **user clicks on Ad**, means { **y = 1** } in 'y' col for different banner_position values.

```
In [34]: # Create a pivot table to count occurrences of '1' in 'y' for each unique value in 'banner_pos'
pivot_table = pd.pivot_table(df, index='banner_pos', values='y', aggfunc='sum', fill_value=0)
print(pivot_table)
```

banner_pos	y
0	11865
1	5074
2	5
3	1
4	1
5	0
7	34

10)

```
In [15]: df.groupby(['dayofweek', 'day', 'hour'])
```

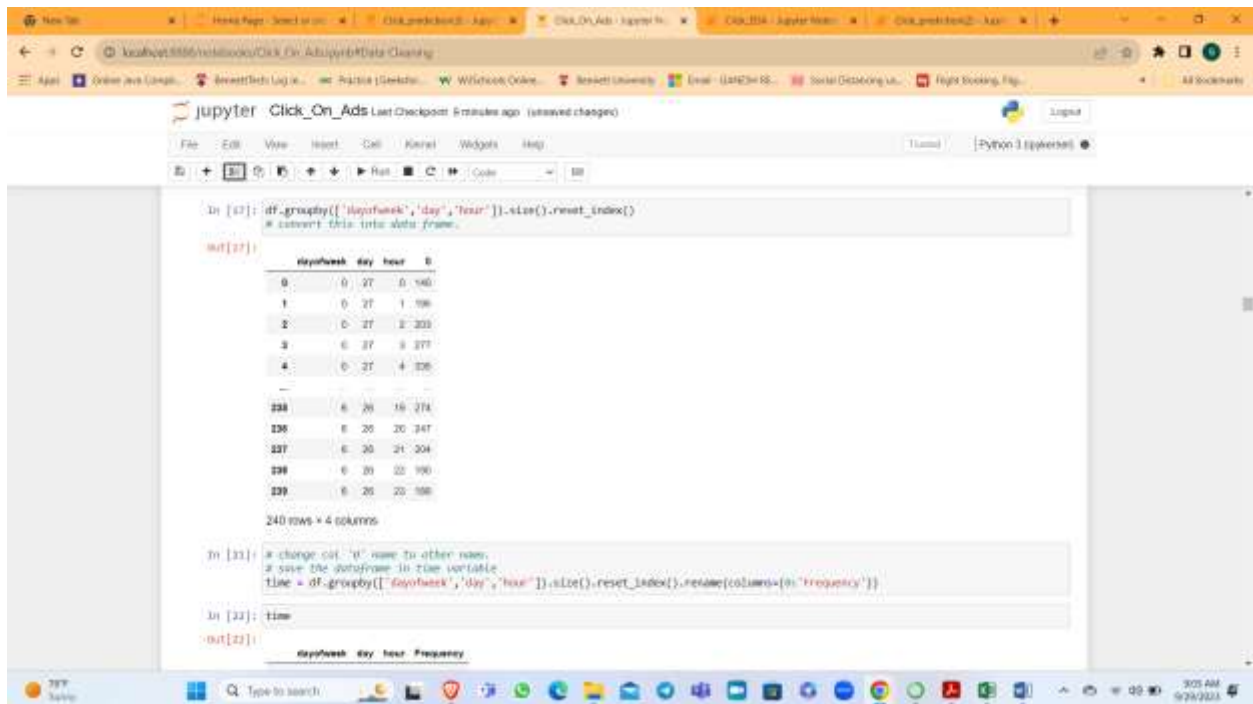
```
Out[15]: <pandas.core.groupby.generic.DataFrameGroupBy object at 0x000002BA5664C110>
```

```
In [16]: df.groupby(['dayofweek', 'day', 'hour']).size()
```

```
Out[16]: dayofweek  day  hour
0          27    0      140
          1      195
          2      203
          3      277
          4      335
...
6          26   19      274
          20      247
          21      204
          22      190
          23      188
Length: 240, dtype: int64
```

Here it shows how many times (dayofweek,day,hour) combination repeated in our dataset. There are **240 combinations**.

11)



A screenshot of a Jupyter Notebook interface. The browser tabs at the top include 'New Tab', 'Hiring Page - Search on...', 'Click prediction2 - App...', 'Click_On_Ads - Jupyter No...', 'Click_BDA - Jupyter Note...', and 'Click prediction2 - App...'. The address bar shows 'localhost:8888/notebooks/Click_On_Ads/jupyterDataCleaning'. The notebook title is 'Click_On_Ads Last Checkpoint: 5 minutes ago (unsaved changes)'. The menu bar includes 'File', 'Edit', 'View', 'Insert', 'Cell', 'Kernel', 'Widgets', and 'Help'. The toolbar shows icons for running, saving, and other actions. The code cell contains the following Python code:

```
In [27]: df.groupby(['dayofweek', 'day', 'hour']).size().reset_index()
# convert this into data frame.

Out[27]:
```

	dayofweek	day	hour	0
0	0	27	0	140
1	0	27	1	195
2	0	27	2	203
3	0	27	3	277
4	0	27	4	325
...				
225	6	28	19	274
226	6	28	20	247
227	6	28	21	304
228	6	28	22	190
229	6	28	23	188

240 rows x 4 columns

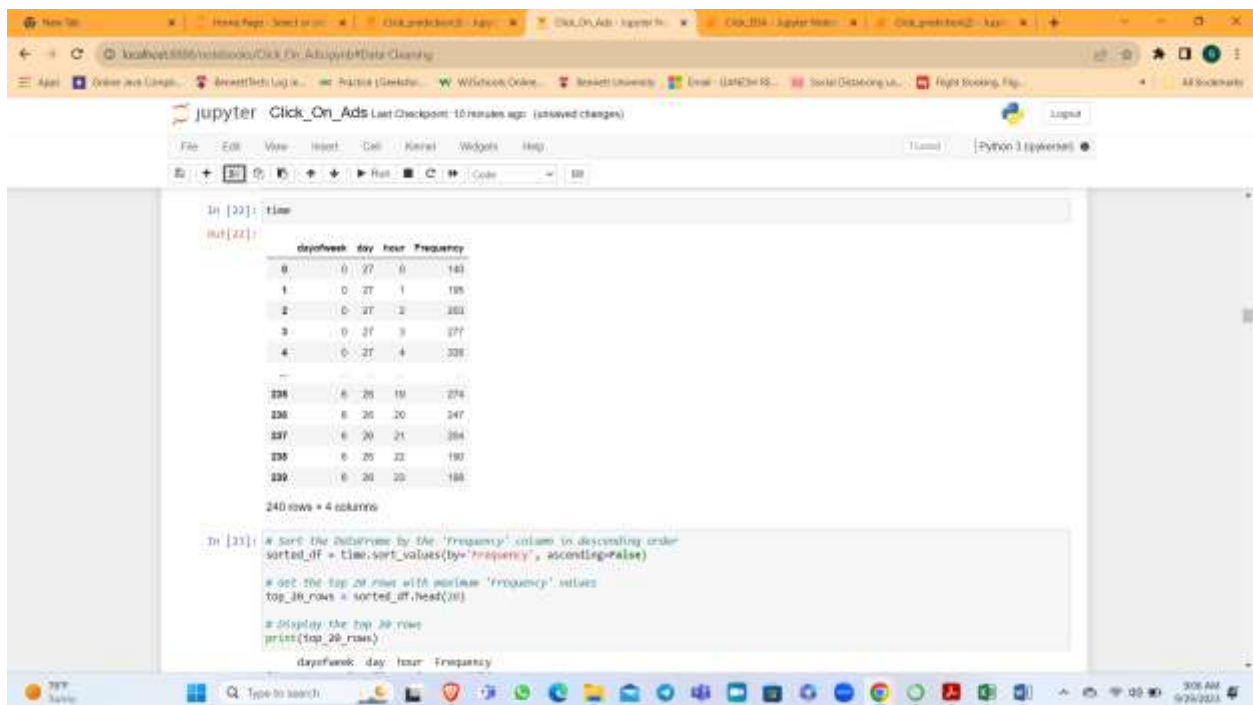
```
In [31]: # change col. '0' name to other name.
# save the dataframe in time variable
time = df.groupby(['dayofweek', 'day', 'hour']).size().reset_index().rename(columns={'0': 'frequency'})

In [32]: time

Out[32]:
```

	dayofweek	day	hour	frequency
--	-----------	-----	------	-----------

12)



A screenshot of a Jupyter Notebook interface, continuing from the previous step. The browser tabs and address bar are the same. The notebook title is 'Click_On_Ads Last Checkpoint: 10 minutes ago (unsaved changes)'. The code cell contains the following Python code:

```
In [32]: time

Out[32]:
```

	dayofweek	day	hour	frequency
0	0	27	0	140
1	0	27	1	195
2	0	27	2	203
3	0	27	3	277
4	0	27	4	325
...				
225	6	28	19	274
226	6	28	20	247
227	6	28	21	304
228	6	28	22	190
229	6	28	23	188

240 rows x 4 columns

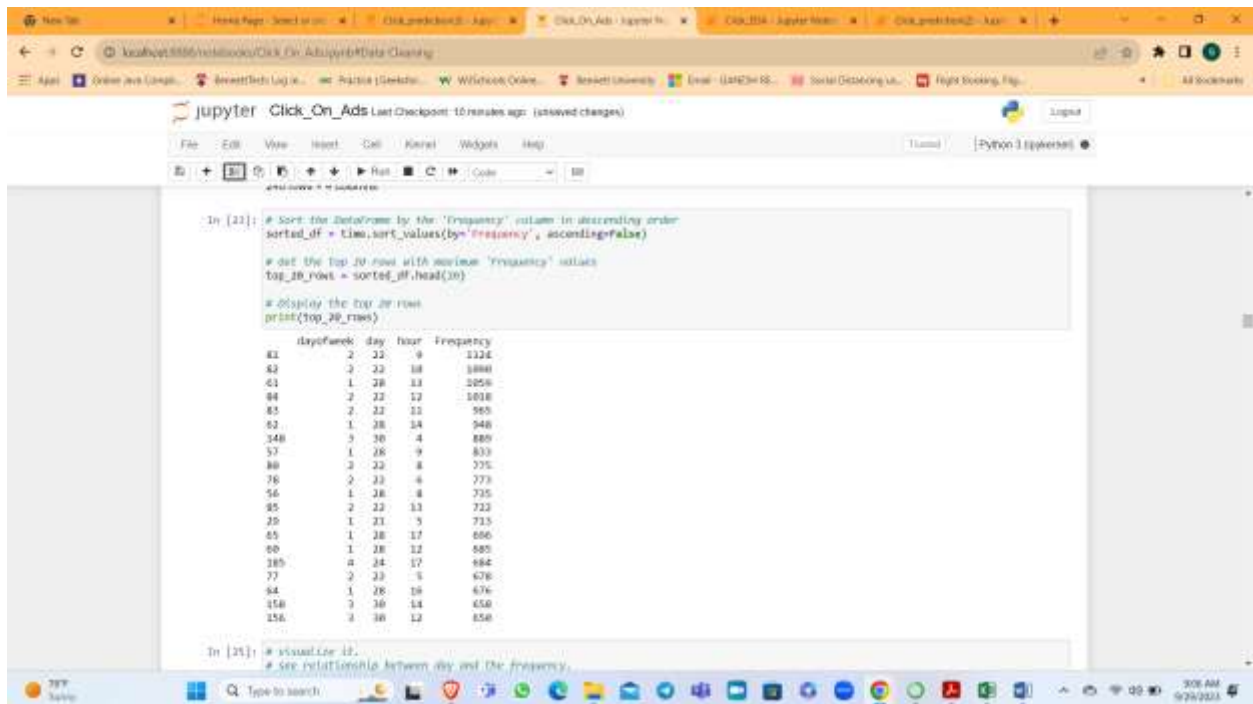
```
In [33]: # sort the dataframe by the 'frequency' column in descending order
sorted_df = time.sort_values(by='frequency', ascending=False)

# get the top 20 rows with maximum 'frequency' values
top_20_rows = sorted_df.head(20)

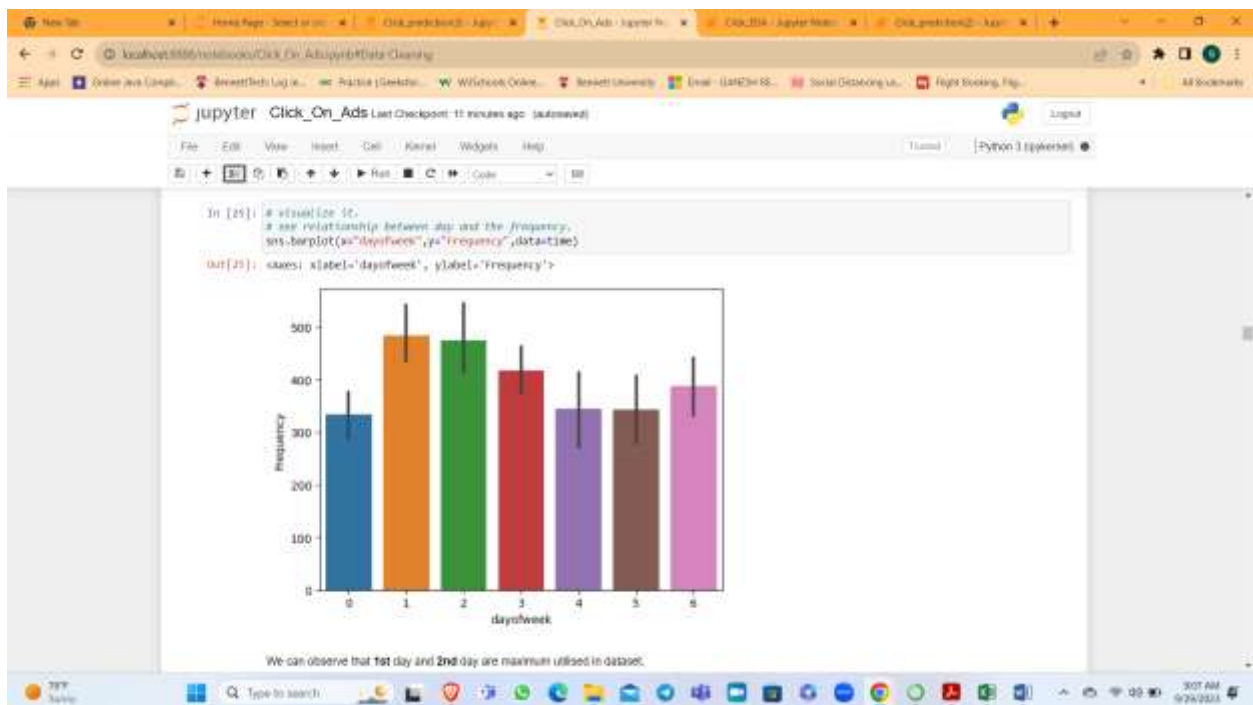
# display the top 20 rows
print(top_20_rows)
```

	dayofweek	day	hour	frequency
--	-----------	-----	------	-----------

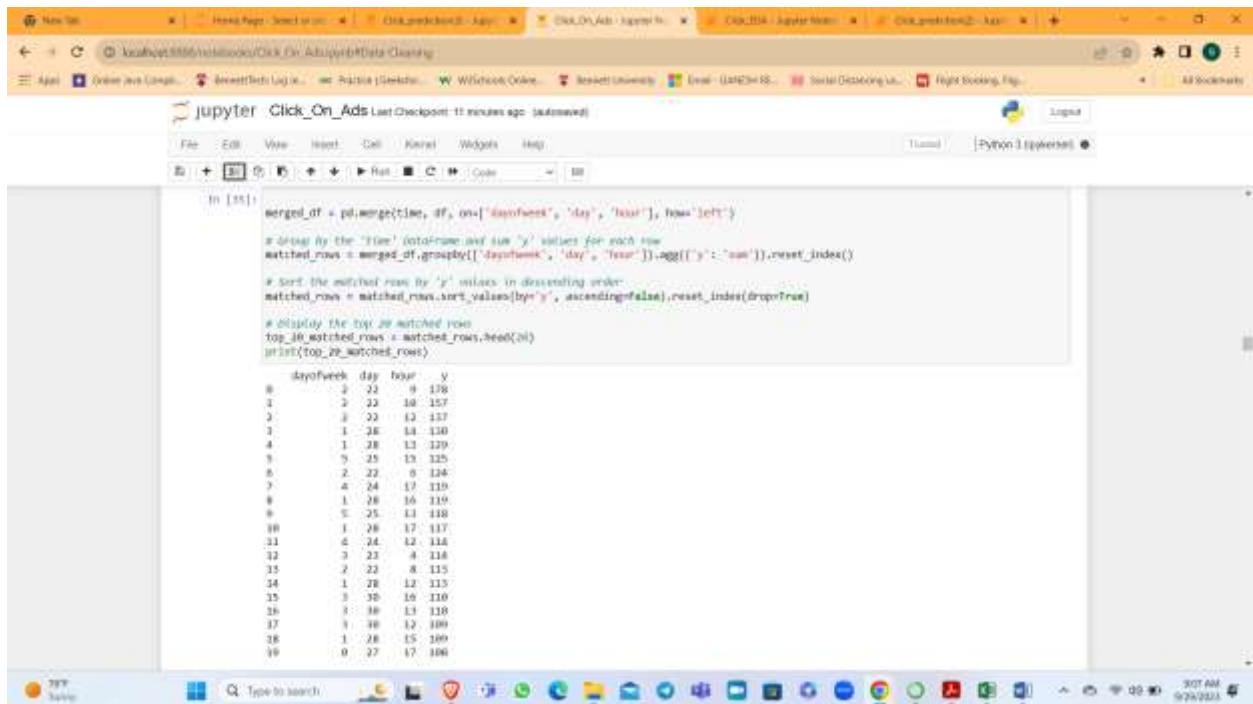
13)



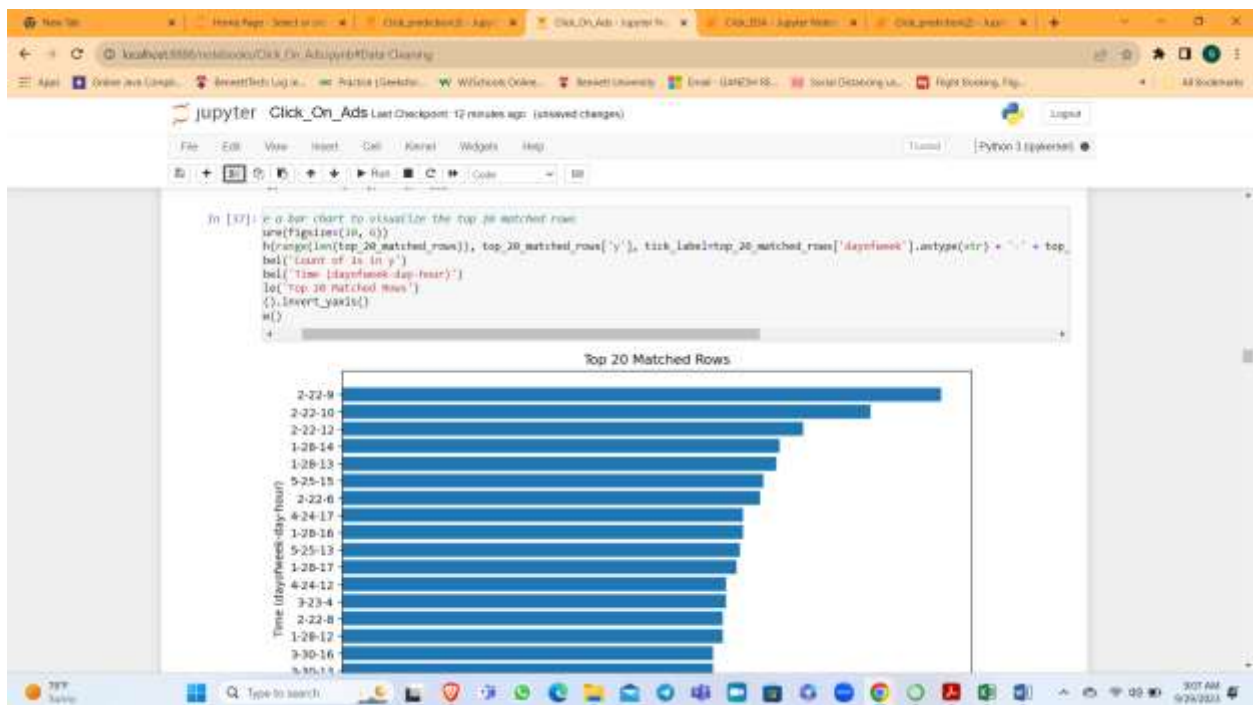
14)



15)



16)



17)

The screenshot shows a Jupyter Notebook interface with the following code and output:

```

In [28]: df.groupby(['site_id', 'site_domain', 'site_category'])
Out[28]: pandas.core.groupby.generic.DataFrameGroupby object at 0x0000200548200000

In [29]: df.groupby(['site_id', 'site_domain', 'site_category']).size()
Out[29]:
site_id  site_domain  site_category
00255f04  240a9324      50e210e0      2
003c103d  00fac8ed      50e210e0      1
005c367d  c4e183de      50e210e0      1
00fa4807  5575e79a      50e210e0      1
00f7c6fc  c4e183de      50e210e0      1
--
77cb2b3a  fef0ac7d4      50e210e0      1
c4e183de      50e210e0      1
c4e183de      50e210e0      6
f0d18ae8      50e210e0      10
f0f7772b  f0f7772b      1
Length: 2108, dtype: int64

There are 2108 combinations.

In [30]: # create site dataframe
site = df.groupby(['site_id', 'site_domain', 'site_category']).size().reset_index().rename(columns={'size': 'site_frequency'})

In [31]: site
Out[31]:
   site_id  site_domain  site_category  site_frequency
0  00255f04  240a9324      50e210e0              2
1  003c103d  00fac8ed      50e210e0              1

```

18)

The screenshot shows a Jupyter Notebook interface with the following code and output:

```

In [30]: # create site dataframe
site = df.groupby(['site_id', 'site_domain', 'site_category']).size().reset_index().rename(columns={'size': 'site_frequency'})

In [31]: site
Out[31]:
   site_id  site_domain  site_category  site_frequency
0  00255f04  240a9324      50e210e0              2
1  003c103d  00fac8ed      50e210e0              1
2  005c367d  c4e183de      50e210e0              1
3  00fa4807  5575e79a      50e210e0              1
4  00f7c6fc  c4e183de      50e210e0              1
--
2103  f0cb2b3a  fef0ac7d4      50e210e0              1
2104  f0cb2b3a  c4e183de      50e210e0              1
2105  f0cb2b3a  c4e183de      50e210e0              6
2106  f0cb2b3a  f0f7772b      50e210e0              10
2107  f0f7772b  f0f7772b      1
2108 rows x 4 columns

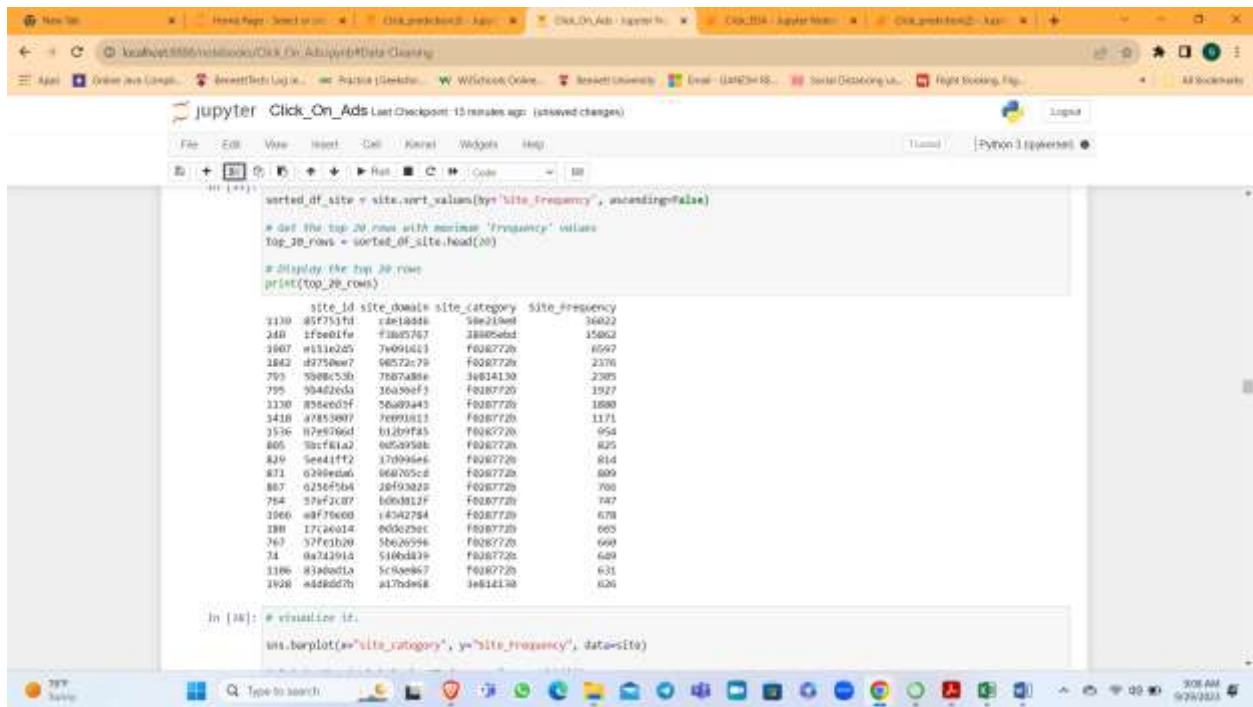
In [32]: sorted_df_site = site.sort_values(by='site_frequency', ascending=False)

# Get the top 20 rows with maximum 'frequency' values
top_20_rows = sorted_df_site.head(20)

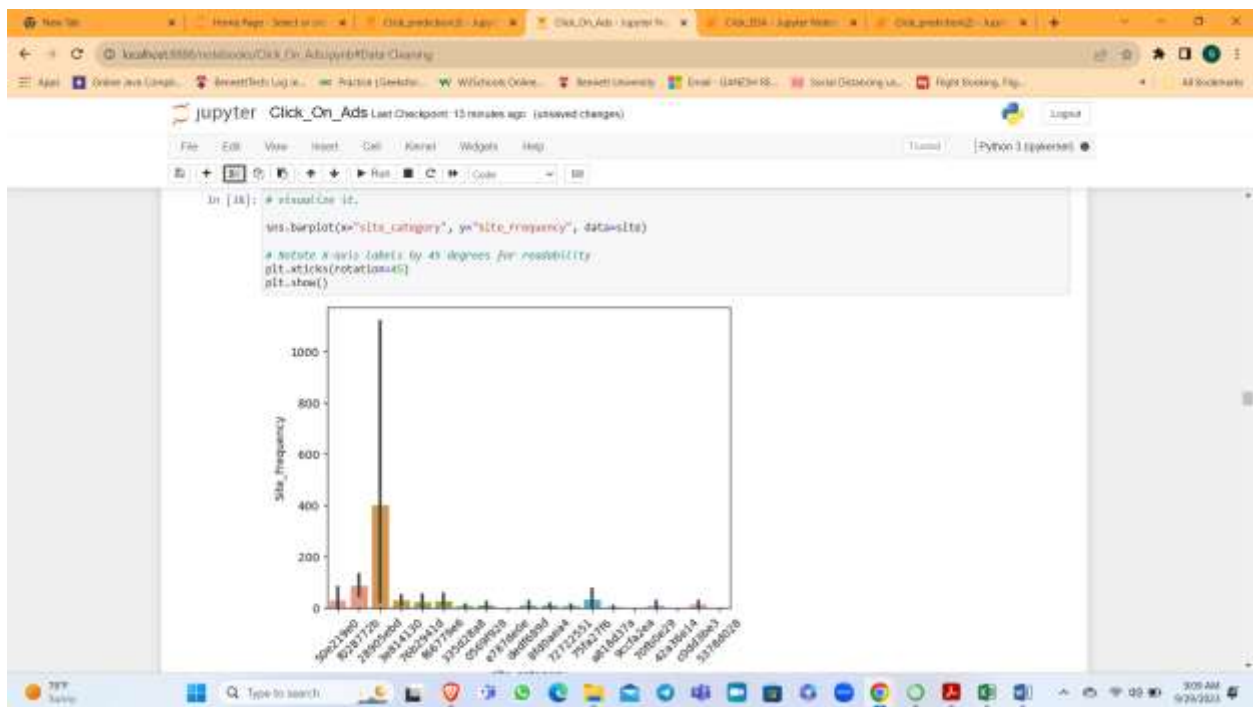
# Display the top 20 rows

```

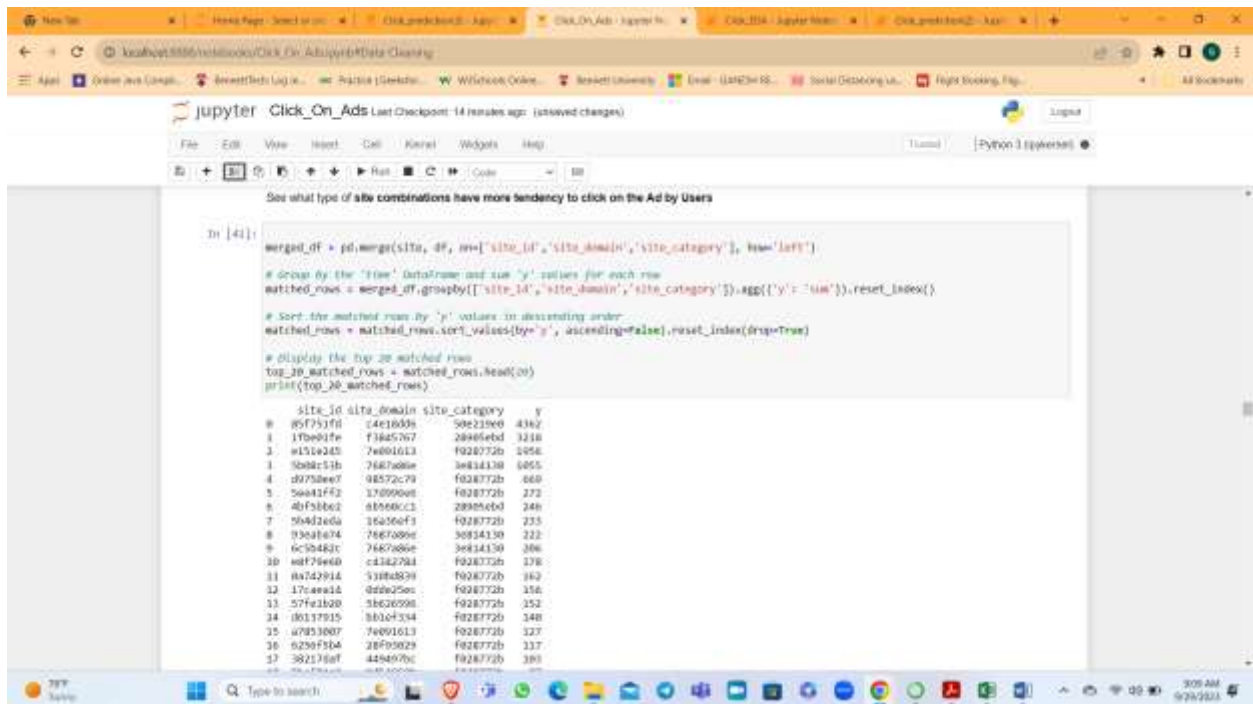
19)



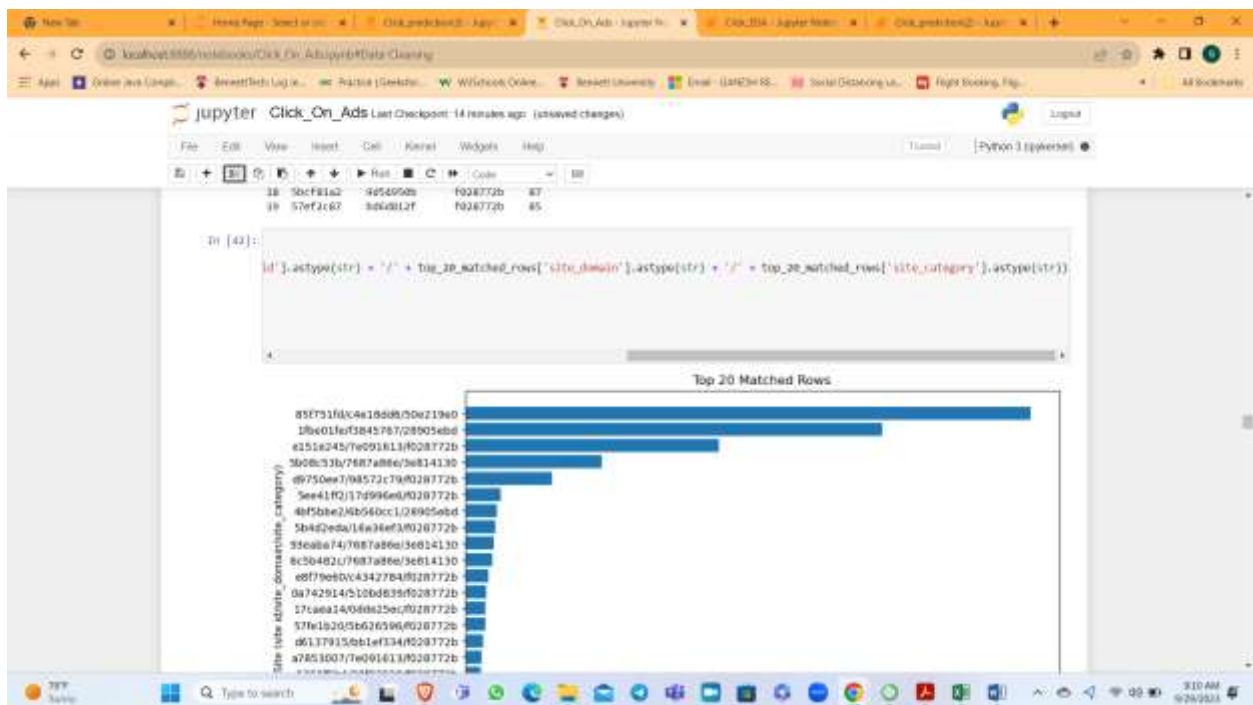
20)



21)



22)



23)

The screenshot shows a Jupyter Notebook interface with the following code and output:

```
In [43]: create_app_dataframe
app = df.groupby(['app_id', 'app_domain', 'app_category']).size().reset_index().rename(columns={'n': 'App_Frequency'})

In [44]: app
Out[44]:
```

	app_id	app_domain	app_category	App_Frequency
0	003ae84	234797a	02181f8	3
1	0079b0a	7801e8d	02181f8	1
2	00848ac	234797a	0a3a649	1
3	00a4e91	08a648e	02181f8	1
4	00a6a27	7801e8d	07d7922	1
...
1390	0e1f10a	7801e8d	02181f8	1
1391	0a108c3	d5a564a	02181f8	2
1392	06a48a	234797a	02181f8	1
1393	06a322b	234797a	02181f8	3
1394	0a9f60	7801e8d	02181f8	82

```
1395 rows x 4 columns

In [45]: sorted_df_site = app.sort_values(by='App_Frequency', ascending=False)

# Get the top 20 rows with maximum "frequency" values
top_20_rows = sorted_df_site.head(20)
```

24)

The screenshot shows the same Jupyter Notebook interface with the following code and output:

```
In [45]: sorted_df_site = app.sort_values(by='App_Frequency', ascending=False)

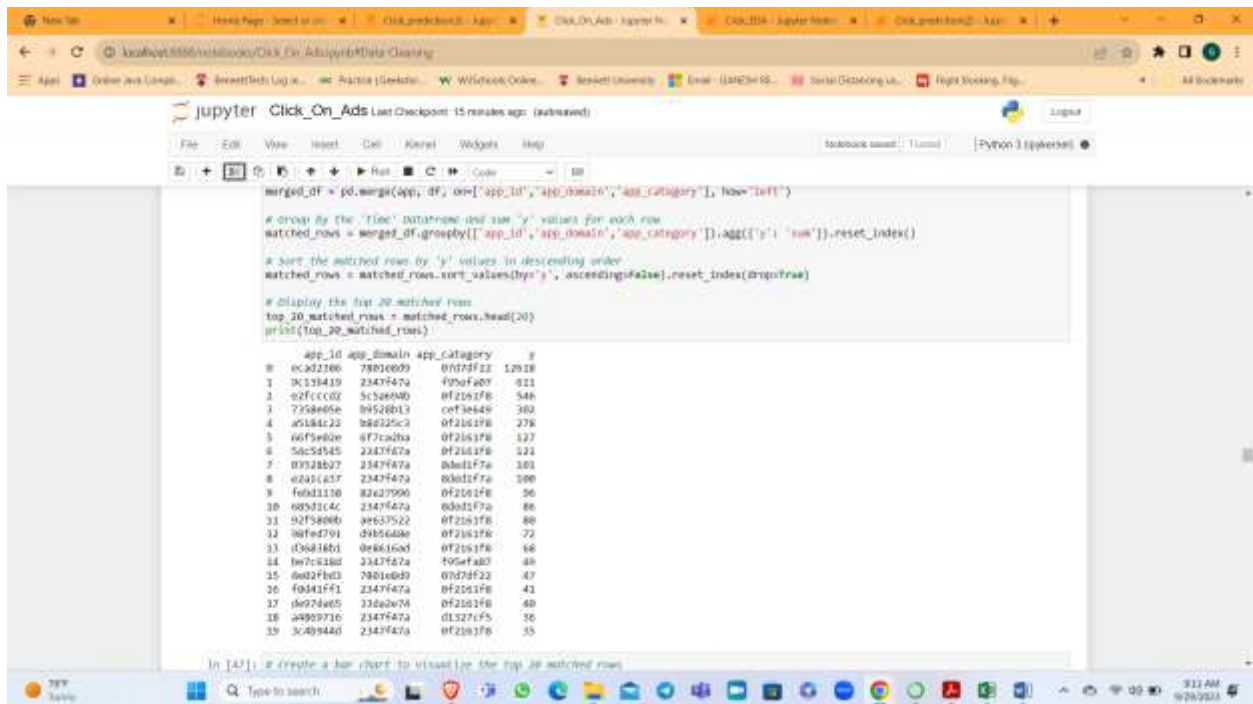
# Get the top 20 rows with maximum "frequency" values
top_20_rows = sorted_df_site.head(20)

# Display the top 20 rows
print(top_20_rows)
```

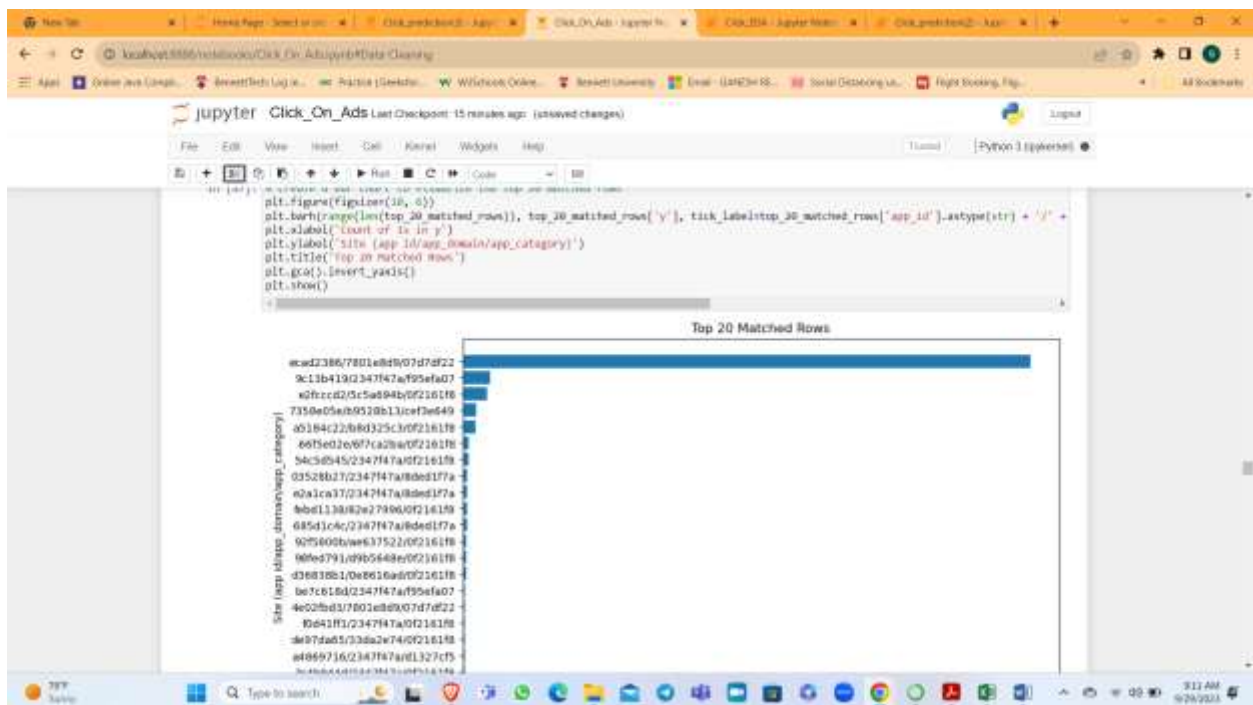
	app_id	app_domain	app_category	App_Frequency
1255	0ca62380	7801e8d	07d7922	61977
761	02f5800b	a637522	0f23b1f8	3885
1197	02fccc02	5c5a094b	0f23b1f8	2849
1349	febd1138	62e27996	0f23b1f8	1888
868	9c118419	2347f47a	f9aefae7	1867
599	7358e05e	b9528b13	cef3ae49	1521
848	4f18dc22	08a275c1	0f23b1f8	1196
1189	036838b1	0a661a6d	0f23b1f8	1124
447	54c50540	2347f47a	0f23b1f8	953
549	085d1c4c	2347f47a	0a6d1f7a	949
1274	f9641f41	2347f47a	0f23b1f8	799
21	05528827	2347f47a	0a6d1f7a	779
1231	a9718628	0f23a7a9	cef3ae49	716
1195	e31c8377	2347f47a	0a6d1f7a	679
426	51ee88da	aefc06bd	0f23b1f8	555
792	08fed791	085658a	0f23b1f8	544
542	06f5e02a	0f7ca28a	0f23b1f8	543
23	03a88c2f	7801e8d	0f23b1f8	520
707	7320c707	2347f47a	0a6d1f7a	492
1385	f5343761	0a661a6d	0f23b1f8	465

```
In [46]:
```

25)



26)



27)

The screenshot shows a Jupyter Notebook titled "Click_On_Advertising" with the following content:

Data Cleaning and Outlier treatment.

As there are **no missing values**, just we find **duplicate rows** and remove it.

```
In [48]: # Remove duplicate rows
df = df.drop_duplicates()
```

```
In [49]: df.shape
Out[49]: (98541, 26)
```

We can observe some rows has been **dropped**

Outlier treatment for numerical values.

We will use **IQR technique** for outlier detection.

```
In [50]: # Before applying the outlier technique, just see the range of values.
# If range is small, no need to apply outlier treatment.
df['C1'].value_counts()
```

```
Out[50]:
3805    91213
3802    1469
3810    2263
3812     253
3807      93
3801      21
3808       9
Name: C1, dtype: int64
```

28)

The screenshot shows a Jupyter Notebook titled "Click_On_Advertising" with the following content:

```
In [51]: df['device_type'].value_counts()
Out[51]:
1    91589
0     5489
4     1928
5       235
Name: device_type, dtype: int64
```

Here also, small range of values.

```
In [52]: df['device_com_type'].value_counts()
Out[52]:
0     95872
2      7932
3      5422
5        235
Name: device_com_type, dtype: int64
```

```
In [53]: df['C14'].value_counts()
Out[53]:
4867    2302
23813   2172
21189   1906
21101   1696
19772   1621
...
29419     1
21705     1
25506     1
8685      1
16152     1
Name: C14, Length: 1722, dtype: int64
```


29)

The screenshot shows a Jupyter Notebook titled "Click_On_Ads" with the following code and output:

```

In [53]: df['C14'].value_counts()
Out[53]:
4647    2302
21611   2172
21189   1980
21391   1890
29772   1821
      ...
19418     1
21785     1
15586     1
9803      1
36153     1
Name: C14, Length: 1722, dtype: int64

```

Now we can see large range of values, so we can do outlier treatment.

In IQR:

1. calculate Q1 and Q3
2. calculate lower_limit and upper_limit: $lower_limit = Q1 - (1.5 \times IQR)$
 $upper_limit = Q3 + (1.5 \times IQR)$

```

In [54]: Q1 = df.C14.quantile(0.25)
          Q3 = df.C14.quantile(0.75)
          Q1, Q3
Out[54]: (16620.0, 21802.0)

In [55]: IQR = Q3 - Q1
          IQR

```

30)

The screenshot shows a Jupyter Notebook titled "Click_On_Ads" with the following code and output:

```

In [54]: Q1 = df.C14.quantile(0.25)
          Q3 = df.C14.quantile(0.75)
          Q1, Q3
Out[54]: (16620.0, 21802.0)

In [55]: IQR = Q3 - Q1
          IQR
Out[55]: 4973.0

In [56]: lower_limit = Q1 - 1.5*IQR
          upper_limit = Q3 + 1.5*IQR
          lower_limit, upper_limit
Out[56]: (9466.5, 29352.5)

In [57]: df[(df.C14 < lower_limit) | (df.C14 > upper_limit)]
Out[57]:

```

	C1	banner_pos	site_id	site_domain	site_category	app_id	app_domain	app_category	device_id	device_is	...	C17	C18	C19	C20	C21
48	1005	0	8873193	ele13a08	55a219a0	7384a05e	da528a13	ca5ef446	55a73f11	55a8a187		178	3	1207	100146	16
81	1005	1	a151a245	7a081613	8c26772b	ecae2380	7801eab8	57a78d22	a992144	e5ea453c		423	2	38	100146	32
88	1002	0	c340c1d4	3a081613	55a219a0	ecae2380	7801eab8	57a78d22	55a78d22	55a78d22		572	2	38	-1	32
88	1005	1	a151a245	7a081613	8c26772b	ecae2380	7801eab8	57a78d22	a992144	a4a2f015		423	2	38	100146	32
87	1005	0	8873193	ele13a08	55a219a0	ecae2380	7801eab8	57a78d22	a992144	883d4151		761	3	115	100077	23
...
89018	1005	0	8873193	ele13a08	55a219a0	7384a05e	da528a13	ca5ef446	55a73f11	55a8a187		178	3	1207	100146	16
89037	1005	1	a151a245	7a081613	8c26772b	ecae2380	7801eab8	57a78d22	a992144	a4a2f015		423	2	38	100146	32

31)

The screenshot shows a Jupyter Notebook interface with the following code and output:

```
In [57]: df[(df.C14>lower_limit)](df.C14>upper_limit)]
Out[57]:
```

	C1	banner_pos	site_id	site_domain	site_category	app_id	app_domain	app_category	device_id	device_is	...	C17	C18	C19	C20	C21
48	1005	0	3807199	eliv1808	55621960	7356e05e	06028e13	ce5e6e9	55a7d71f	55a6a187		176	0	1007	100146	16
81	1005	1	w15w245	7e091013	820772b	ecad2389	7801e6b9	07d7852	ab9214a	ef5a653c		423	2	38	100146	32
98	1002	0	c040c1a0	3e591475	55621960	ecad2389	7801e6b9	07d7852	8ba556e	55a6c24		172	2	38	-1	32
99	1005	1	w15w245	7e091013	820772b	ecad2389	7801e6b9	07d7852	ab9214a	4463615		423	2	38	100146	32
87	1005	0	8f6a1b4	5c6c0907	820772b	ecad2389	7801e6b9	07d7852	ab9214a	80324151		751	0	115	100077	23
...																
89916	1005	0	3807199	c4e18a08	55621960	7356e05e	06028e13	ce5e6e9	55a7d71f	55a6a187		176	0	1007	100146	16
89927	1005	1	w15w245	7e091013	820772b	ecad2389	7801e6b9	07d7852	ab9214a	7ad3145		423	2	38	100146	32
89968	1005	0	6208584	2669204	820772b	ecad2389	7801e6b9	07d7852	ab9214a	71a323a8		672	2	38	-1	32
89978	1005	1	w15w245	7e091013	820772b	ecad2389	7801e6b9	07d7852	ab9214a	83034219		671	2	38	100046	32
89982	1005	0	3807199	c4e18a08	55621960	7356e05e	06028e13	ce5e6e9	55a7d71f	55a6a187		176	0	1007	100146	16

8171 rows x 20 columns

See these are the outliers there are total 8171 outliers in 99999 rows, with respect to C14 col

```
In [58]: # If you want you can remove these outliers from our dataframe.
# store it in new dataframe as df_new.
df_no_outlier = df[(df.C14>lower_limit)&(df.C14<upper_limit)]
df_no_outlier
Out[58]:
```

32)

The screenshot shows a Jupyter Notebook interface with the following code and output:

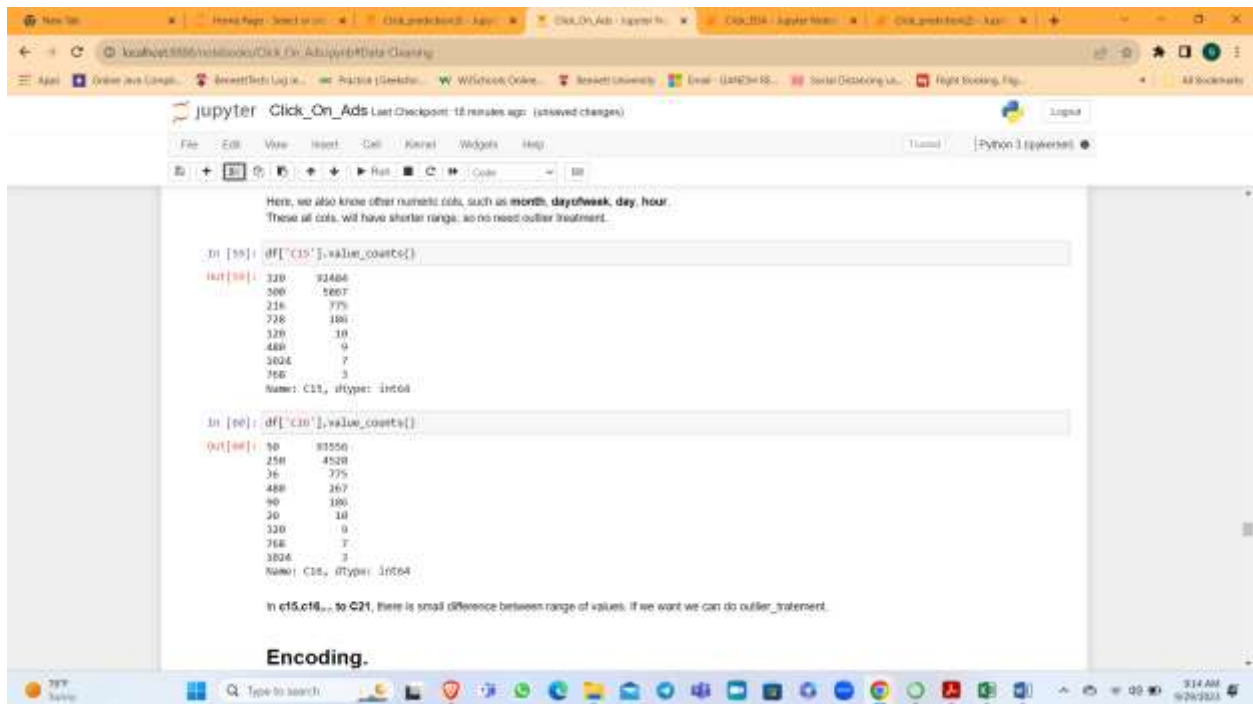
```
In [58]: # If you want you can remove these outliers from our dataframe.
# store it in new dataframe as df_new.
df_no_outlier = df[(df.C14>lower_limit)&(df.C14<upper_limit)]
df_no_outlier
Out[58]:
```

	C1	banner_pos	site_id	site_domain	site_category	app_id	app_domain	app_category	device_id	device_is	...	C17	C18	C19	C20	C21
0	1005	1	856a628	88a8e43	820772b	ecad2389	7801e6b9	07d7852	ab9214a	8620833		2620	0	38	100076	201
1	1005	1	w15w245	7e091013	820772b	ecad2389	7801e6b9	07d7852	ab9214a	5a194e9		1534	2	38	-1	16
2	1005	0	43c295a	c06211e	28056e6	ecad2389	7801e6b9	07d7852	ab9214a	ab6a84c		2652	2	107	100009	20
3	1002	0	0a094462	246e439	55621960	ecad2389	7801e6b9	07d7852	0a67980	88f623a0		2470	0	167	100074	23
4	1005	0	7ba211e	c043787	28056e6	ecad2389	7801e6b9	07d7852	ab9214a	7a6e38a		1722	0	38	-1	76
...																
89994	1005	0	7294eaf	883a09d	3e014100	ecad2389	7801e6b9	07d7852	ab9214a	703653a5		1973	3	38	100146	23
89995	1005	0	7ba211e	c043787	28056e6	ecad2389	7801e6b9	07d7852	ab9214a	70117c0c		2545	0	167	100004	201
89996	1005	0	3807199	c4e18a08	55621960	7356e05e	06028e13	ce5e6e9	55a7d71f	55a6a187		176	0	1007	100146	16
89997	1005	0	80c6c53c	7507a08a	3e014100	ecad2389	7801e6b9	07d7852	ab9214a	4015ea13		2295	2	38	100091	23
89998	1005	0	7ba211e	c043787	28056e6	ecad2389	7801e6b9	07d7852	ab9214a	594e603a		2817	0	38	-1	81

81170 rows x 20 columns

Here, we also know other numeric cols, such as month, dayofweek, day, hour. These all cols, will have shorter range, so no need outlier treatment.

33)



Here, we also know other numeric cols, such as **month, dayofweek, day, hour**. These all cols. will have shorter range, so no need outlier treatment.

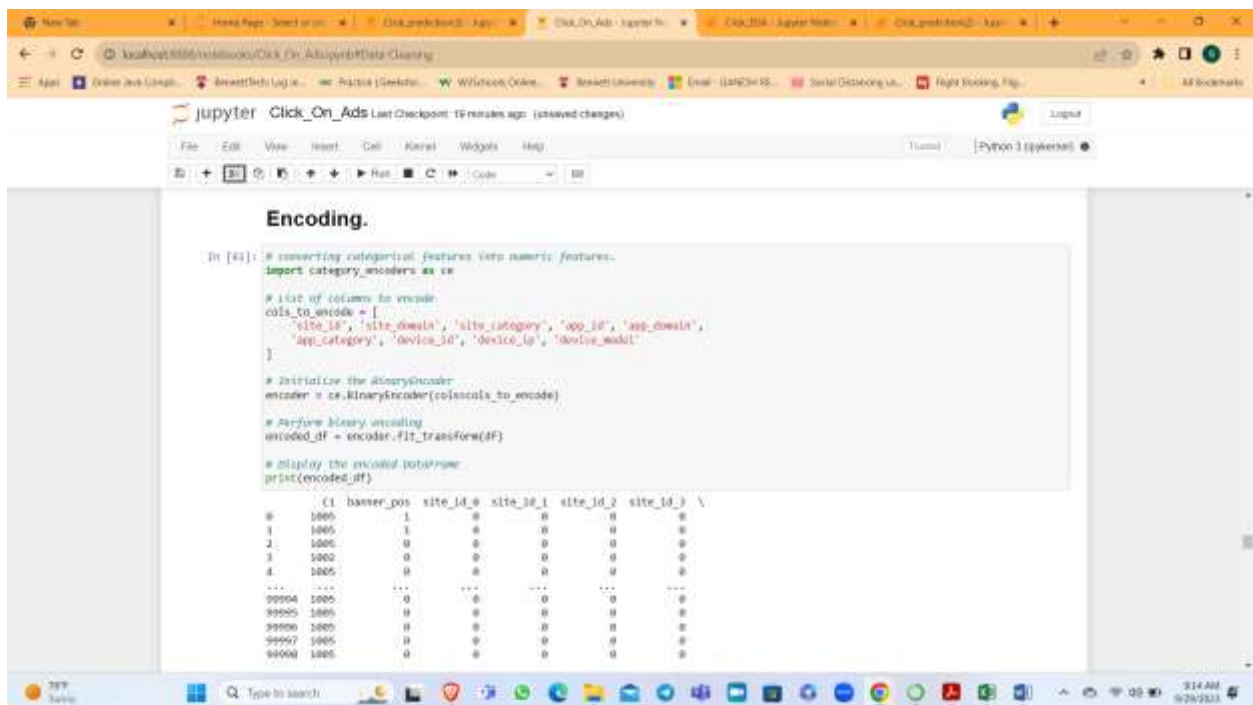
```
In [55]: df['c15'].value_counts()
Out[55]:
320    32486
300     5607
216     775
728     186
320       9
488        7
3024        3
768         0
Name: c15, dtype: int64
```

```
In [56]: df['c20'].value_counts()
Out[56]:
30     83556
258     4528
36     775
488     267
90      186
20       16
320        9
768        7
3024        3
Name: c20, dtype: int64
```

In c15,c16... to c21, there is small difference between range of values. If we want we can do outlier treatment.

Encoding.

34)



Encoding.

```
In [41]: # converting categorical features into numeric features.
import category_encoders as ce

# list of columns to encode
cols_to_encode = [
    'site_id', 'site_domain', 'site_category', 'app_id', 'app_domain',
    'app_category', 'device_id', 'device_id', 'device_model'
]

# Initialize the OneHotEncoder
encoder = ce.OneHotEncoder(cols=cols_to_encode)

# Perform binary encoding
encoded_df = encoder.fit_transform(df)

# display the encoded dataframe
print(encoded_df)
```

```
(1 banner_pos site_id_0 site_id_1 site_id_2 site_id_3) \
0 1000 1 0 0 0
1 1000 1 0 0 0
2 1000 0 0 0 0
3 1000 0 0 0 0
4 1000 0 0 0 0
...
99994 1000 0 0 0 0
99995 1000 0 0 0 0
99996 1000 0 0 0 0
99997 1000 0 0 0 0
99998 1000 0 0 0 0
```

The screenshot shows a Jupyter Notebook with the following content:

```

import pandas as pd
df = pd.read_csv('Click_On_Ads.csv')

df

```

Output of the first cell:

	site_id_4	site_id_5	site_id_6	site_id_7	...	C17	C18	C19
0	0	0	0	0	...	2528	0	39
1	0	0	0	0	...	1934	2	39
2	0	0	0	0	...	3552	3	167
3	0	0	0	0	...	2476	1	167
4	0	0	0	0	...	1722	0	35
...
99996	0	1	1	0	...	1071	3	39
99995	0	0	0	0	...	2545	0	167
99990	0	0	0	1	...	2443	2	39
99997	0	0	0	1	...	2295	2	35
99998	0	0	0	0	...	2617	0	35

```

df = df.drop([C10, C11, C12], axis=1)

df

```

Output of the second cell:

	C20	C21	month	dayofweek	day	hour	y
0	100075	223	10	1	28	14	0
1	-1	16	10	2	22	19	1
2	100082	23	10	3	23	18	0
3	100074	23	10	3	22	19	0
4	-1	79	10	1	21	0	1
...
99994	100146	23	10	3	23	15	0
99995	100004	223	10	2	29	0	0
99996	-1	23	10	2	22	15	1
99997	100081	23	10	5	25	0	1
99998	-1	51	10	1	28	1	0

[99999 rows x 111 columns]

The screenshot shows a Jupyter Notebook with the following code and output:

```
# Define the desired column order
desired_order = [
    'i', 'banner_pos', 'site_id', 'site_domain', 'site_category',
    'app_id', 'app_domain', 'app_category', 'device_id', 'C14',
    'C18', 'C19', 'C23', 'C21', 'month',
    'dayofweek', 'day', 'hour', 'MVDOH', 'y'
]

# Reorder the columns based on the desired order
new_df = new_df[desired_order]
```

In [84]: new_df

Out[84]:

i	site_domain	site_category	app_id	app_domain	app_category	device_id	C14	C18	C19	C23	C21	month	dayofweek	day	hour	MVDOH	y
34	64d36d3	020772b	ecad2388	7801ebd5	07d7d02	ab9214a	22680	0	38	100275	231	10	1	26	14	100214	0
45	7a3b9113	020772b	ecad2388	7801ebd5	07d7d02	ab9214a	17637	2	39	-1	19	10	2	22	19	100219	1
1e	d26311fe	189054cd	ecad2388	7801ebd5	07d7d02	ab9214a	22105	0	187	100200	23	10	3	23	18	100218	0
13	248e438	15a219cd	ecad2388	7801ebd5	07d7d02	0a0338d	21581	0	167	100074	23	10	2	23	18	100218	0
1e	03d43767	189054cd	ecad2388	7801ebd5	07d7d02	ab9214a	15700	0	38	-1	19	10	1	21	8	100208	1
0f	663a9be	3e3d4300	ecad2388	7801ebd5	07d7d02	ab9214a	17236	0	38	100148	23	10	5	23	16	100216	0
1e	03d43767	189054cd	ecad2388	7801ebd5	07d7d02	ab9214a	22261	0	187	100064	221	10	2	29	0	100900	0
0f	04e18a8b	59d119cd	735d65e	04526a1b	0af5e49	07cd8dd	21276	2	39	-1	31	10	3	23	15	100215	1
1e	7951a58e	3e3d4300	ecad2388	7801ebd5	07d7d02	ab9214a	20893	2	38	100061	23	10	6	25	8	100308	1
1e	03d43767	189054cd	ecad2388	7801ebd5	07d7d02	ab9214a	22677	0	38	-1	19	10	1	26	1	100201	0

37)

The screenshot shows a Jupyter Notebook interface with the following code and output:

```
In [45]:
# Write the frequency of each 'app_category' value and store it in a dictionary
app_freq = new_df['app_category'].value_counts().to_dict()

# Create a new column 'frequency_app_category' by mapping the 'app_category' values to their frequencies
new_df.insert(new_df.columns.get_loc('app_category') + 1, 'frequency_app_category', new_df['app_category'].map(app_freq))

out[45]:
```

app_id	app_domain	app_category	frequency_app_category	device_id	C18	C19	C20	C21	month	dayofweek	day	hour	MMDDHH	y	
020772b	ecad2380	7801a6d9	07d7e022	04902	ad8014e	0	39	100015	221	10	1	20	14	100814	0
020772b	ecad2380	7801a6d9	07d7e022	04900	ad8014e	2	34	1	10	10	2	22	19	100219	1
28905d4	ecad2380	7801a6d9	07d7e022	04902	ad8014e	1	167	100202	23	10	3	21	16	100316	0
00a219a0	ecad2380	7801a6d9	07d7e022	04902	9fa5798e	1	167	100074	23	10	2	22	19	100219	0
29905d4	ecad2380	7801a6d9	07d7e022	04902	ad8014e	0	35	1	79	10	1	21	9	100109	1
3a014100	ecad2380	7801a6d9	07d7e022	04902	ad8014e	2	38	100146	23	10	2	20	15	100315	0
28905d4	ecad2380	7801a6d9	07d7e022	04902	ad8014e	0	167	100064	221	10	2	29	8	100908	0
00a219a0	738ba05a	19528a13	1a73e949	4388	027c8e0d	1	38	1	23	10	2	22	16	100216	1
3a014100	ecad2380	7801a6d9	07d7e022	04902	ad8014e	1	38	100094	23	10	5	25	8	100508	1
28905d4	ecad2380	7801a6d9	07d7e022	04902	ad8014e	0	36	1	61	10	1	20	1	100001	0

See another col added.

38)

The screenshot shows a Jupyter Notebook interface with the following code and output:

```
In [46]:
# Now, we can do this for 'site_domain' also.

# calculate the frequency of each 'site_domain' value and store it in a dictionary
site_domain_freq = new_df['site_domain'].value_counts().to_dict()

# Create a new column 'frequency_site_domain' by mapping the 'site_domain' values to their frequencies
new_df.insert(new_df.columns.get_loc('site_domain') + 1, 'frequency_site_domain', new_df['site_domain'].map(site_domain_freq))

out[46]:
```

C1	banner_pos	site_id	site_domain	frequency_site_domain	site_category	app_id	app_domain	app_category	frequency_app_category	C18
0	1005	1	950e4031	0a08a42	1874	03d772b	ecad2380	7801a6d9	07d7e022	04902
1	1005	1	a151d446	7a08b013	8150	03d772b	ecad2380	7801a6d9	07d7e022	04902
2	1005	0	a5c2903a	0262c14	794	29905d4	ecad2380	7801a6d9	07d7e022	04902
3	1002	0	0a944152	048439f	0	00a219a0	ecad2380	7801a6d9	07d7e022	04902
4	1005	0	1fa0714e	0343757	15944	28905d4	ecad2380	7801a6d9	07d7e022	04902
88884	1005	0	7244a07	863a08e	160	3a014100	ecad2380	7801a6d9	07d7e022	04902
88888	1005	0	1fa0714e	0343757	15944	28905d4	ecad2380	7801a6d9	07d7e022	04902
88888	1005	0	8873194	1a618a08	26070	00a219a0	738ba05a	19528a13	1a73e949	4388
88887	1005	0	6006b03b	18d7a01e	0219	3a014100	ecad2380	7801a6d9	07d7e022	04902
88888	1005	0	1fa0714e	0343757	15944	28905d4	ecad2380	7801a6d9	07d7e022	04902

88941 rows x 25 columns

The screenshot shows a Jupyter Notebook environment with the following content:

```

In [47]: X = encoded_df.iloc[:,1:] # except last column
         y = encoded_df['y'] # target variable

In [48]: X.head()

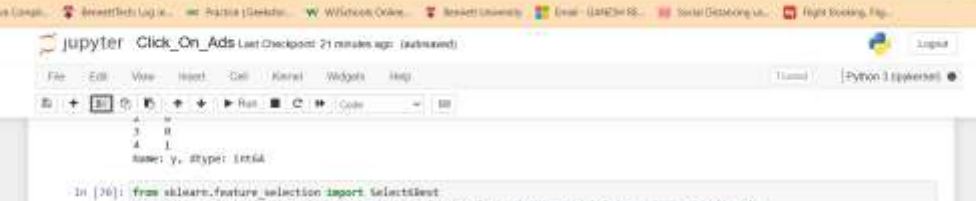
Out[48]:
   id  banner_pos  site_id_0  site_id_1  site_id_2  site_id_3  site_id_4  site_id_5  site_id_6  site_id_7  ...  C16  C17  C18  C19  C20  C21  month  dayofw
0  1005         1         1         0         0         0         0         0         0         0         ...    81  2528    0   38  100078  221    31
1  1005         1         1         0         0         0         0         0         0         0         ...    81  1894    2   38    1   16    30
2  1006         0         0         0         0         0         0         0         0         0         ...    90  2642    3  167  100000    23    30
3  1002         0         0         0         0         0         0         0         0         0         ...    83  2476    3  167  100074    23    30
4  1005         0         0         0         0         0         0         0         0         0         ...    81  1723    0   36    1   79    16

5 rows * 170 columns

In [48]: y.head()

Out[48]:
0    0
1    1
2    0
3    0
4    1
Name: y, dtype: int64

```



The screenshot shows a Jupyter Notebook titled "Click_On_Ads" with a checkpoint from 21 minutes ago. The interface includes a top bar with file explorer, search, and navigation icons. The notebook has a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help) and a toolbar with icons for running, saving, and other actions. The code cell is numbered 36 and contains the following Python code:

```
In [36]: from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import mutual_info_classif # Import Mutual Information score function

# create a selectkbest instance with mutual information and fit it to your data
k_best_ml = SelectKBest(score_func=mutual_info_classif, k=60) # Assign 'k' to the number of features.
k_best_ml.fit(X, y)

# get the indices of selected features (columns)
selected_indices_ml = k_best_ml.get_support(indices=True)

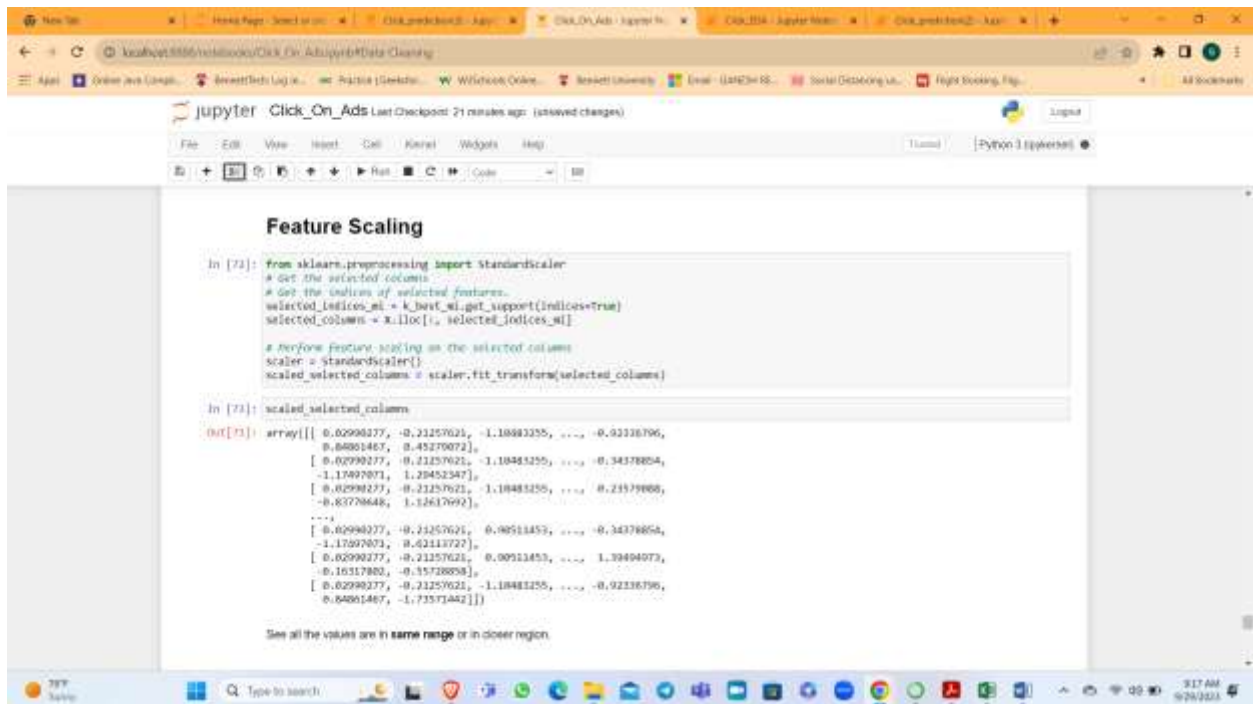
# get the names of selected columns
selected_column_names_ml = [X.columns[i] for i in selected_indices_ml]

# Print the names of selected columns
print("Selected column names (Mutual Information):", selected_column_names_ml)

selected_column_names (Mutual Information): ['ci', 'site_id_1', 'site_id_2', 'site_id_3', 'site_id_4', 'site_id_10', 'site_domain_1', 'site_domain_9', 'site_domain_10', 'site_category_1', 'site_category_3', 'site_category_4', 'app_id_0', 'app_id_8', 'app_id_10', 'app_domain_1', 'app_domain_5', 'app_category_1', 'app_category_4', 'device_id_5', 'device_id_14', 'device_ip_1', 'device_ip_2', 'device_ip_3', 'device_ip_4', 'device_ip_5', 'device_ip_6', 'device_ip_7', 'device_ip_8', 'device_ip_9', 'device_ip_10', 'device_ip_11', 'device_ip_12', 'device_ip_13', 'device_ip_14', 'device_ip_15', 'device_ip_16', 'device_ip_17', 'device_model_1', 'device_model_3', 'device_model_5', 'device_model_6', 'device_model_7', 'device_model_8', 'device_model_9', 'device_model_10', 'device_model_11', 'device_model_13', 'device_type', 'device_com_type', 'c14', 'c15', 'c16', 'c17', 'c18', 'c20', 'c21', 'month', 'day', 'hour']
```

Below the code, there is a text explanation: "See here for [feature_selection](#) I used `mutual_info_classif` as the score function, for Mutual Information-based feature selection. And I also given `k=60`, because after doing binary encoding we get 110 cols. In that I want to use atleast half of the columns for the **model training**. so I used `k = 60`."

41)



The screenshot shows a Jupyter Notebook interface with the title 'Click_On_Ads'. The notebook is running on a remote server, as indicated by the URL in the browser. The code in the notebook is as follows:

```

In [72]: from sklearn.preprocessing import StandardScaler
# Get the selected columns
# Get the indices of selected features
selected_indices_ml = k_best_ml.get_support(indices=True)
selected_columns = X.iloc[:, selected_indices_ml]

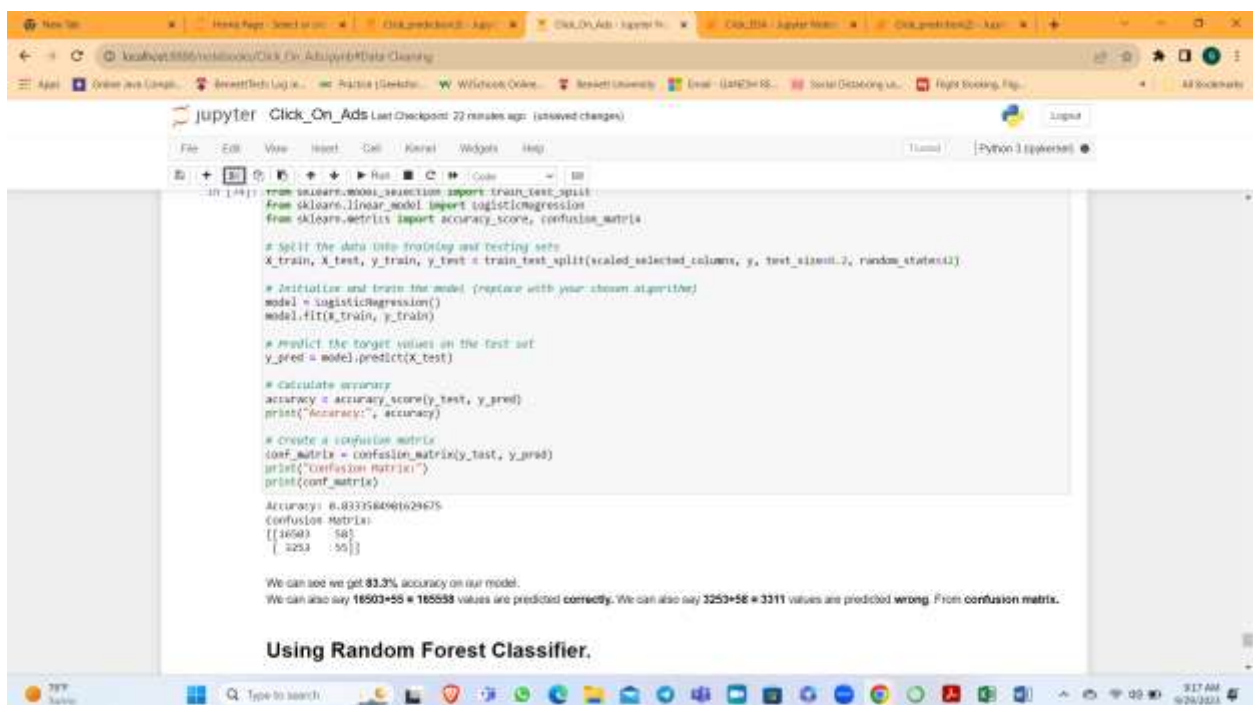
# perform feature scaling on the selected columns
scaler = StandardScaler()
scaled_selected_columns = scaler.fit_transform(selected_columns)

In [73]: scaled_selected_columns
Out[73]: array([[ 0.02990377, -0.21257021, -1.18983255, ..., -0.92336796,
  0.64061467,  0.45270072],
 [ 0.02990377, -0.21257021, -1.18983255, ..., -0.34378804,
 -1.17097071,  1.29452347],
 [ 0.02990377, -0.21257021, -1.18983255, ...,  0.23579968,
 -0.83798648,  1.12617692],
 ...,
 [ 0.02990377, -0.21257021,  0.90513453, ..., -0.34378804,
 -1.17097071,  0.62117727],
 [ 0.02990377, -0.21257021,  0.90513453, ...,  1.39494073,
 -0.16317802, -0.15728898],
 [ 0.02990377, -0.21257021, -1.18983255, ..., -0.92336796,
  0.64061467, -1.73571442]])

```

Below the output, a note states: "See all the values are in same range or in closer region."

42)



The screenshot shows a Jupyter Notebook interface with the title 'Click_On_Ads'. The notebook is running on a remote server, as indicated by the URL in the browser. The code in the notebook is as follows:

```

In [74]: from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix

# split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(scaled_selected_columns, y, test_size=0.2, random_state=1)

# Initialize and train the model (replace with your chosen algorithm)
model = LogisticRegression()
model.fit(X_train, y_train)

# Predict the target values on the test set
y_pred = model.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

# Create a confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(conf_matrix)

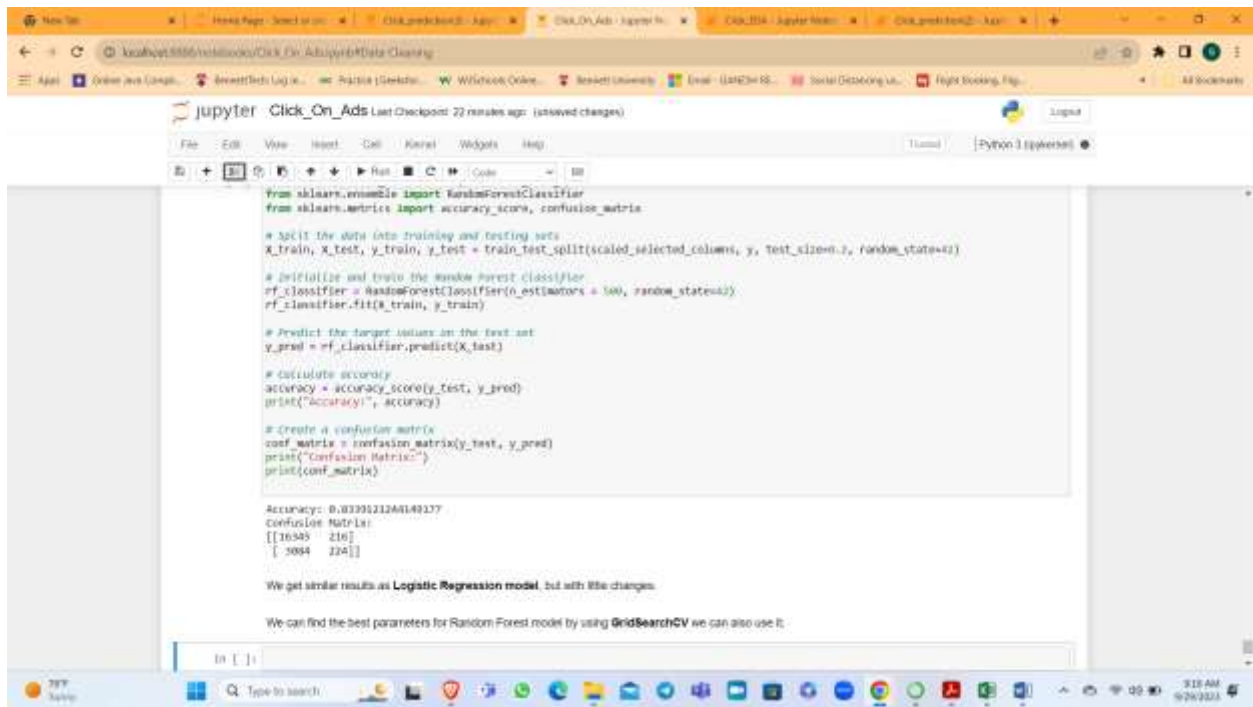
Accuracy: 0.833584961629675
Confusion Matrix:
[[16503  56]
 [ 3253  55]]

```

Below the output, a note states: "We can see we get 83.3% accuracy on our model. We can also say 16503+55 = 16558 values are predicted correctly. We can also say 3253+56 = 3311 values are predicted wrong. From confusion matrix."

Using Random Forest Classifier.

43)



The screenshot shows a Jupyter Notebook titled "Click_On_Ads" with a last checkpoint 22 minutes ago. The notebook is running on a Python 3 (ipykernel) environment. The code in the cell uses sklearn to import RandomForestClassifier, accuracy_score, and confusion_matrix. It splits the data into training and testing sets using train_test_split, initializes and trains a RandomForestClassifier with 100 estimators, predicts on the test set, and calculates the accuracy and confusion matrix. The output shows an accuracy of 0.8339121264149137 and a confusion matrix of [[16345 216], [3084 224]]. Below the code, there are two lines of text: "We get similar results as Logistic Regression model, but with little changes." and "We can find the best parameters for Random Forest model by using GridSearchCV we can also use it."

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(scaled_selected_columns, y, test_size=0.2, random_state=42)

# Initialize and train the Random Forest classifier
rf_classifier = RandomForestClassifier(n_estimators = 100, random_state=42)
rf_classifier.fit(X_train, y_train)

# Predict the target values on the test set
y_pred = rf_classifier.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

# Create a confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(conf_matrix)

Accuracy: 0.8339121264149137
Confusion Matrix:
[[16345  216]
 [ 3084  224]]

We get similar results as Logistic Regression model, but with little changes.

We can find the best parameters for Random Forest model by using GridSearchCV we can also use it.
```

In []: