Alessandro Maddaloni

Part 1 : Introduction to Dataset

What is the dataset about?

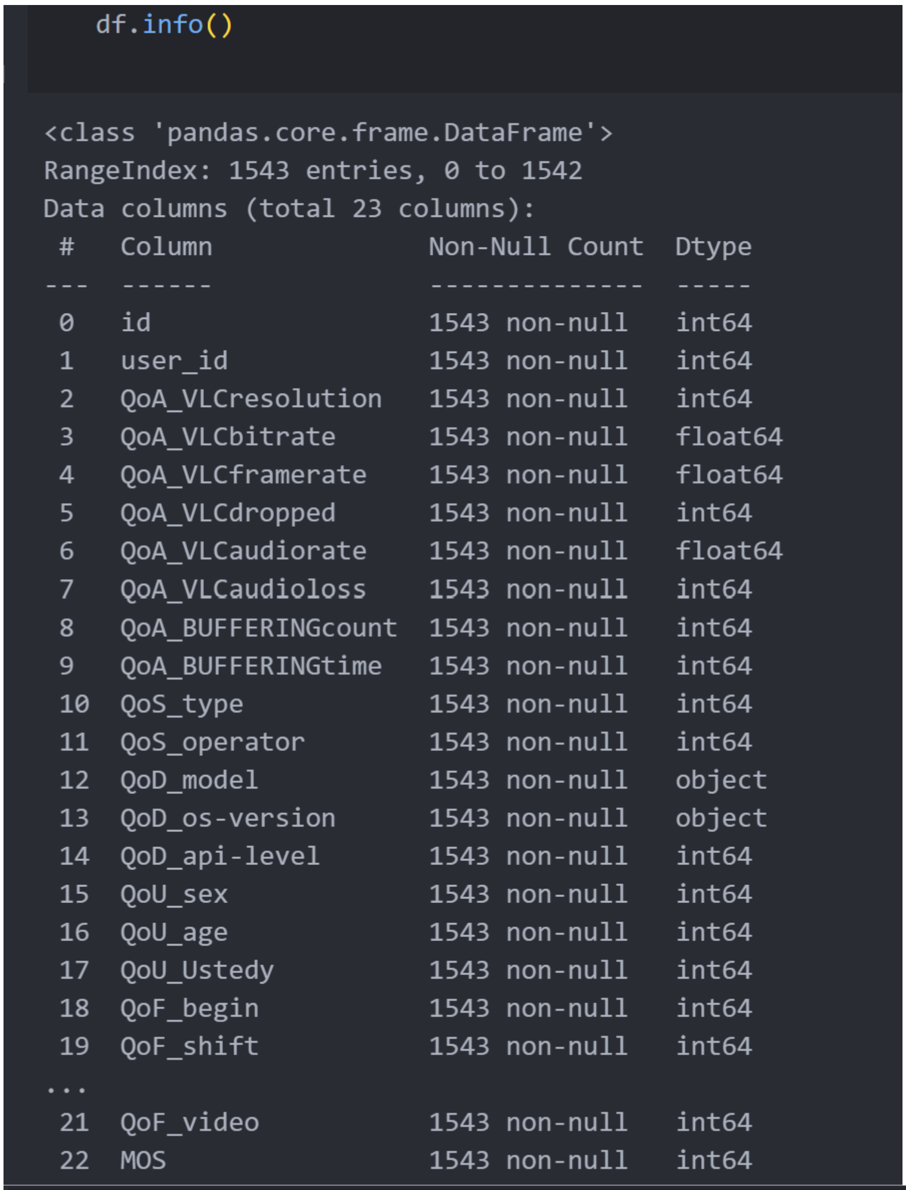
1. It is a dataset collected to study Quality of Experience (QoE) of mobile video streaming from a user perspective.
2. 181 users viewed videos on mobile devices over cellular networks and rated the quality on a 1-5 Mean Opinion Score (MOS).
3. It contains 1560 samples with 29 features covering video quality, network conditions, device details, user profile, etc.
4. The goal was to quantify the impact of various "Quality of Experience Influence Factors" on the perceived quality of experience.

How has it been used?

1. Used by Orange telecom to correlate network Quality of Service to video streaming QoE.
2. Used in a research paper to demonstrate the methodology for crowdsourced QoE evaluation.
3. Can be used to build models to predict video streaming QoE from measurable factors.
4. Provides real-world data to analyze interactions between the many factors impacting QoE.
5. Allows benchmarking QoE across different network types, video parameters, devices, etc.

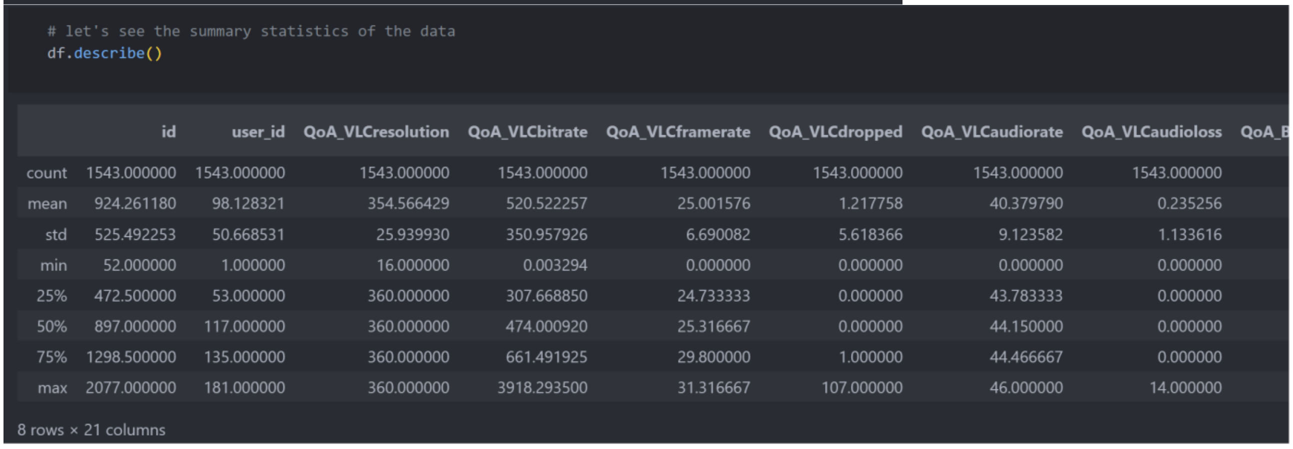
In summary, it is a rich dataset focused on modeling and predicting mobile video streaming quality of experience based on crowdsourced subjective evaluations.

Submitted by: Pyae Sone Kyaw (DANI)



```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1543 entries, 0 to 1542
Data columns (total 23 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   id                  1543 non-null   int64
 1   user_id             1543 non-null   int64
 2   QoA_VLCresolution   1543 non-null   int64
 3   QoA_VLCbitrate      1543 non-null   float64
 4   QoA_VLCframerate    1543 non-null   float64
 5   QoA_VLCdropped      1543 non-null   int64
 6   QoA_VLCaudiorate    1543 non-null   float64
 7   QoA_VLCaudioloss    1543 non-null   int64
 8   QoA_BUFFERINGcount  1543 non-null   int64
 9   QoA_BUFFERINGtime   1543 non-null   int64
 10  QoS_type            1543 non-null   int64
 11  QoS_operator        1543 non-null   int64
 12  QoD_model           1543 non-null   object
 13  QoD_os-version      1543 non-null   object
 14  QoD_api-level       1543 non-null   int64
 15  QoU_sex             1543 non-null   int64
 16  QoU_age             1543 non-null   int64
 17  QoU_Ustedy          1543 non-null   int64
 18  QoF_begin           1543 non-null   int64
 19  QoF_shift           1543 non-null   int64
...
 21  QoF_video           1543 non-null   int64
 22  MOS                 1543 non-null   int64
```



```
# let's see the summary statistics of the data
df.describe()
```

	id	user_id	QoA_VLCresolution	QoA_VLCbitrate	QoA_VLCframerate	QoA_VLCdropped	QoA_VLCaudiorate	QoA_VLCaudioloss	QoA_B
count	1543.000000	1543.000000	1543.000000	1543.000000	1543.000000	1543.000000	1543.000000	1543.000000	
mean	924.261180	98.128321	354.566429	520.522257	25.001576	1.217758	40.379790	0.235256	
std	525.492253	50.668531	25.939930	350.957926	6.690082	5.618366	9.123582	1.133616	
min	52.000000	1.000000	16.000000	0.003294	0.000000	0.000000	0.000000	0.000000	
25%	472.500000	53.000000	360.000000	307.668850	24.733333	0.000000	43.783333	0.000000	
50%	897.000000	117.000000	360.000000	474.000920	25.316667	0.000000	44.150000	0.000000	
75%	1298.500000	135.000000	360.000000	661.491925	29.800000	1.000000	44.466667	0.000000	
max	2077.000000	181.000000	360.000000	3918.293500	31.316667	107.000000	46.000000	14.000000	

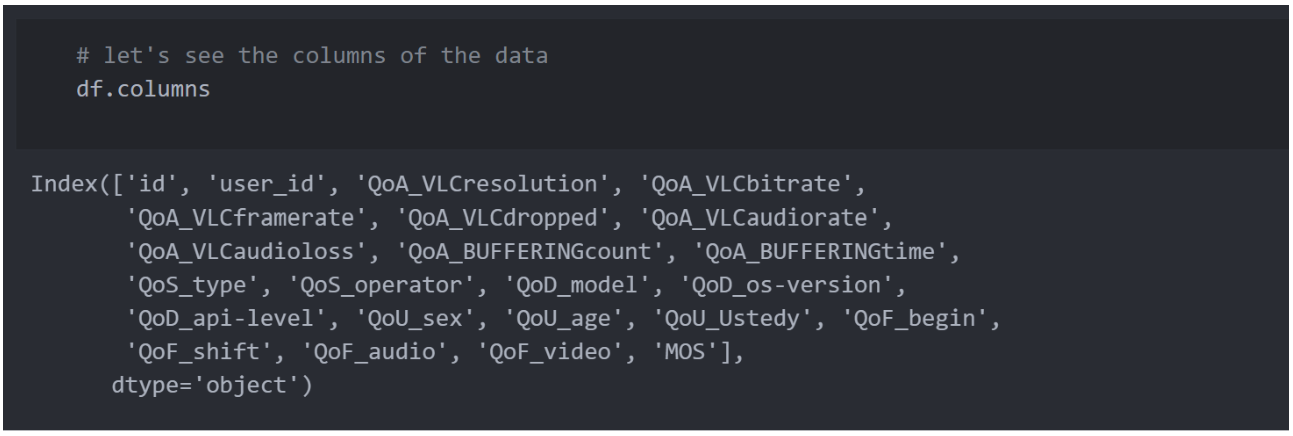
8 rows × 21 columns

Based on these basic understanding of the entire dataset by seeing the summary statistics of the dataset ,info and unique values, we can draw a conclusion that it’s a well-rounded dataset with good distribution and consistent values .

Part 2: Data Exploration

We will characterize and understand our dataset via data exploration. But, I will not report everything I found: I will only report what I think is useful or interesting to observe.

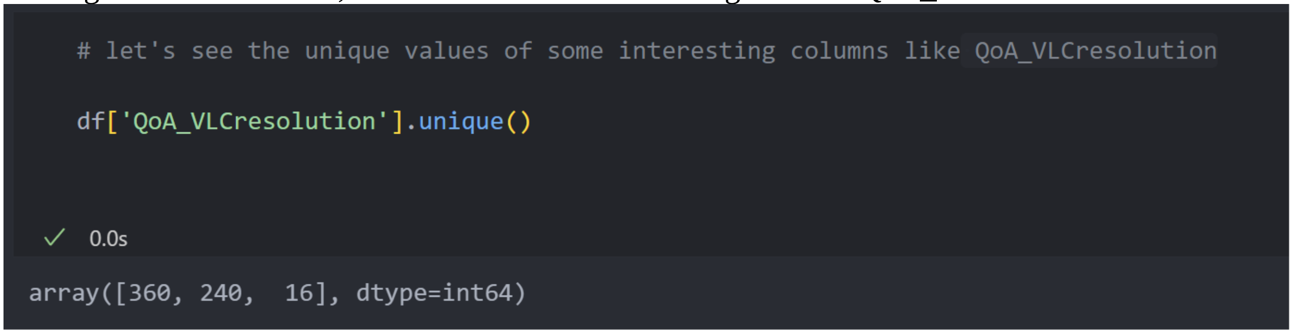
Even though we can learn about most of the ambiguous columns and features from Readme.md file, we can still try our best to understanding some of the columns and the values better.



```
# let's see the columns of the data
df.columns
```

```
Index(['id', 'user_id', 'QoA_VLCresolution', 'QoA_VLCbitrate',
       'QoA_VLCframerate', 'QoA_VLCdropped', 'QoA_VLCaudiorate',
       'QoA_VLCaudioloss', 'QoA_BUFFERINGcount', 'QoA_BUFFERINGtime',
       'QoS_type', 'QoS_operator', 'QoD_model', 'QoD_os-version',
       'QoD_api-level', 'QoU_sex', 'QoU_age', 'QoU_Ustedy', 'QoF_begin',
       'QoF_shift', 'QoF_audio', 'QoF_video', 'MOS'],
      dtype='object')
```

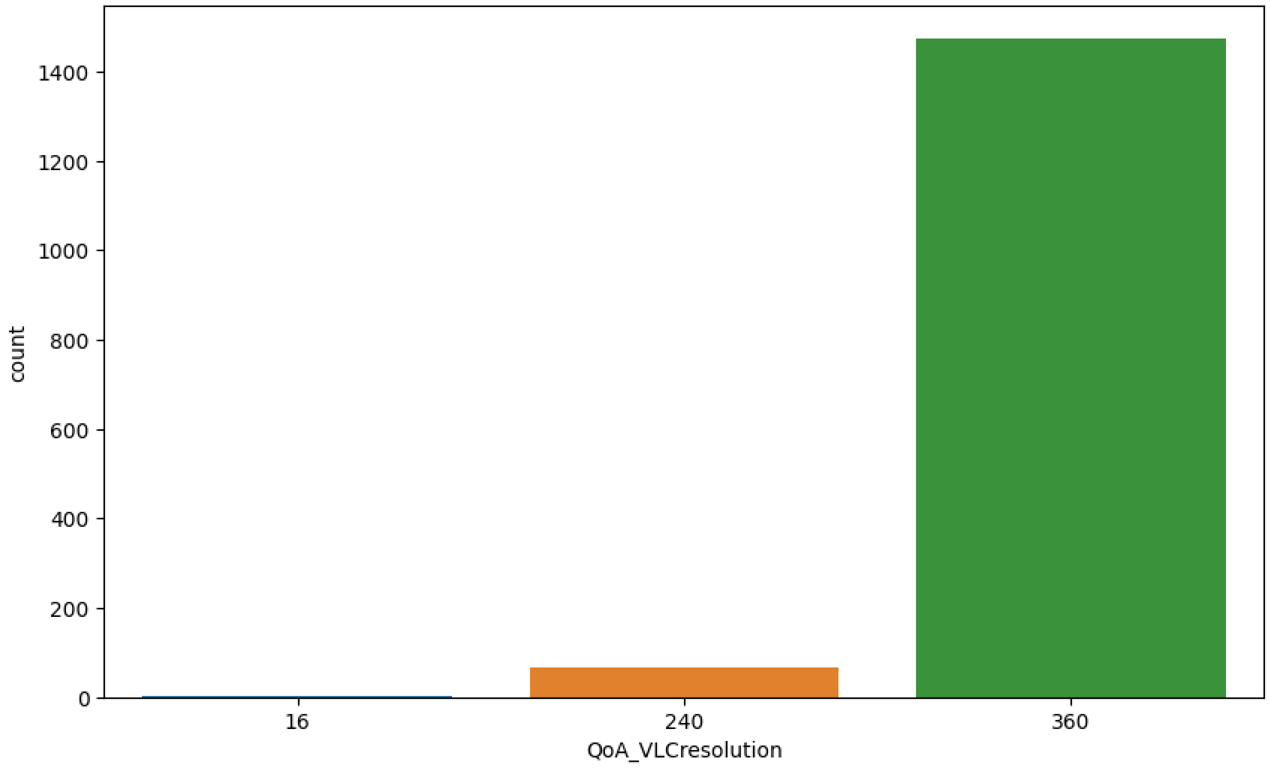
Among all these columns , first of all the most interesting to me is QoA_VLCresolution.



```
# let's see the unique values of some interesting columns like QoA_VLCresolution

df['QoA_VLCresolution'].unique()
```

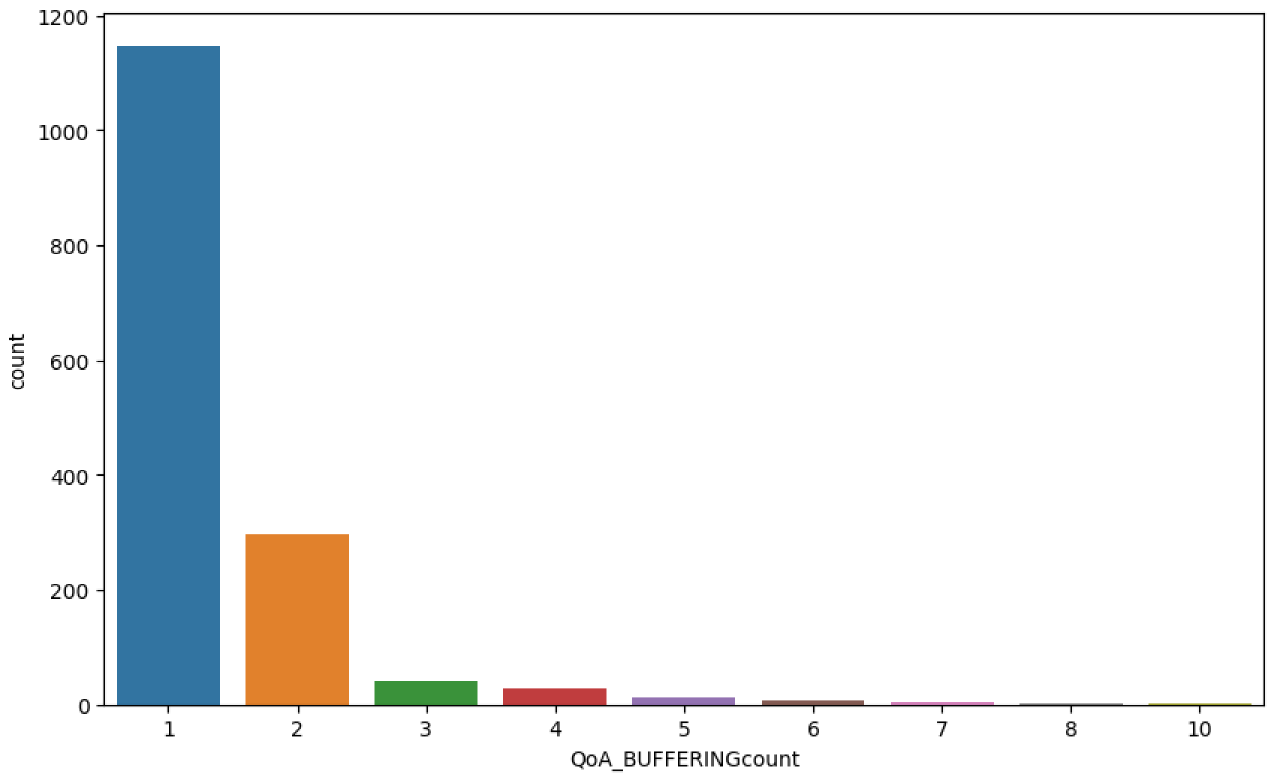
✓ 0.0s

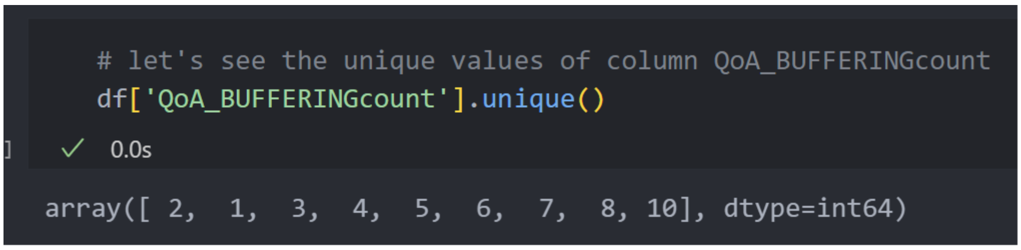
```
array([360, 240,  16], dtype=int64)
```



There are only three unique values in the column QoA_VLCresolution which means that there are only three different resolutions in which the videos are being played. This means that the videos are being played in 160p, 240p and 360p resolutions. This feature is a categorical feature.and it might be useful to convert it into a numerical feature by assigning a number to each resolution.

Moving to another one , we have QoA_BufferingCount with a distribution:





```
# let's see the unique values of column QoA_BUFFERINGcount
df['QoA_BUFFERINGcount'].unique()
✓ 0.0s
array([ 2,  1,  3,  4,  5,  6,  7,  8, 10], dtype=int64)
```

Based on the context that this array [2, 1, 3, 4, 5, 6, 7, 8, 10] represents the unique values for the feature QoA_BufferingCount in the dataset, this indicates:

QoA_BufferingCount refers to the number of buffering events that occurred during the video streaming session. Buffering happens when the video playback stops temporarily to load more content.

The numbers in the array indicate the actual count of buffering events observed in different samples in the dataset.

For example:

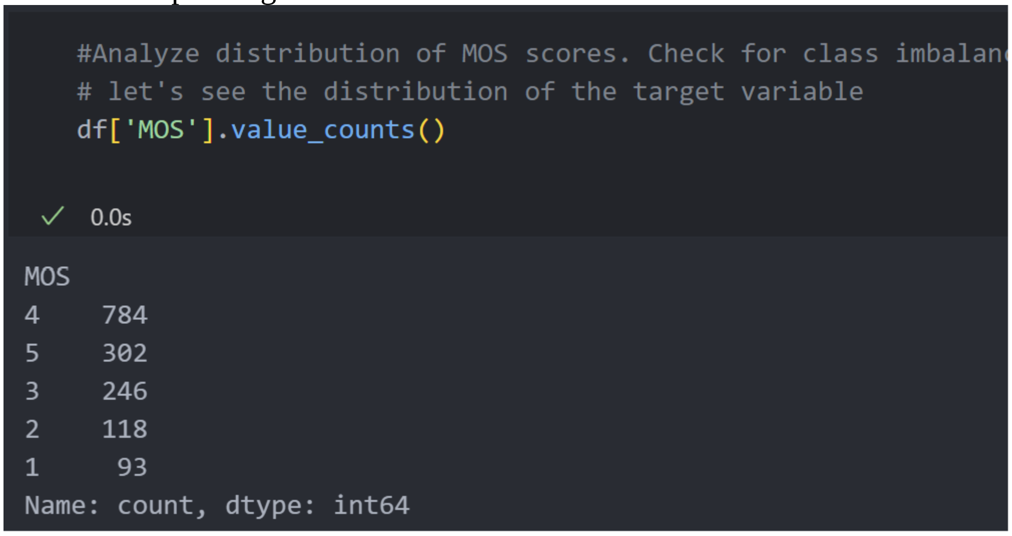
2 - Some video sessions had 2 buffering events
1 - Some had 1 buffering event
3 - Some had 3 buffering events
and so on...

10 - The maximum number of buffering events in a video session was 10 which rarely happens.

So in summary:
- This array shows the distribution of the buffering count values across video sessions in the dataset samples.
- It ranges from minimum 1 to maximum 10 buffering events per video session.

Getting such a distribution gives a sense of the variability and range of this particular factor that affects quality of experience. Analysing factors like buffering count and their correlation with Mean Opinion Score can provide insights into mobile video streaming quality. (let's dive into more columns later on based on our objectives)

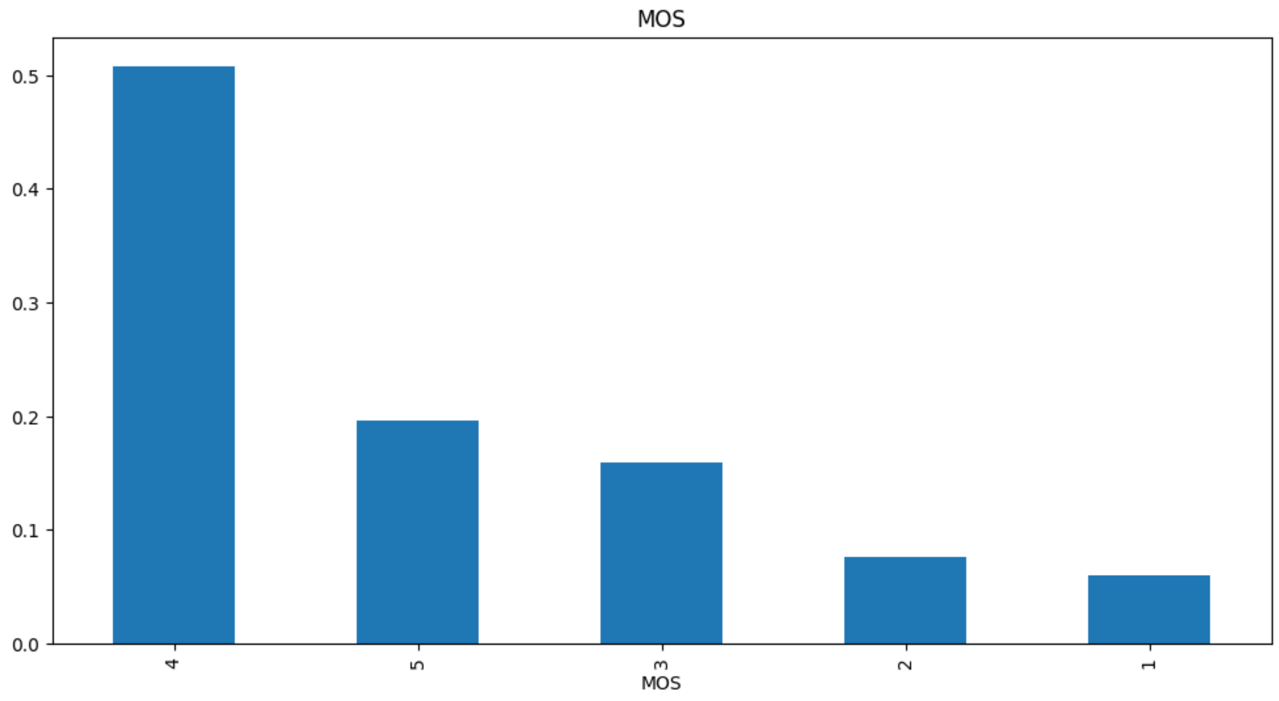
Now, let's move on to some preliminary analysis on our target variable `MOS' with its unique values corresponding to each classes can be seen as:



```
#Analyze distribution of MOS scores. Check for class imbalan
# let's see the distribution of the target variable
df['MOS'].value_counts()
```

✓ 0.0s

```
MOS
4    784
5    302
3    246
2    118
1     93
Name: count, dtype: int64
```



Here are my thoughts on the class distribution above:

1. MOS scores are fairly evenly distributed across the different classes from Bad (MOS 1) to Excellent (MOS 5) quality. This is good to have representations of all perception levels.

2. The dataset is still imbalanced with many more samples for the higher quality scores:
   - Highest number of samples in Good (MOS 4) class
   - Least number of samples in Bad (MOS 1) class

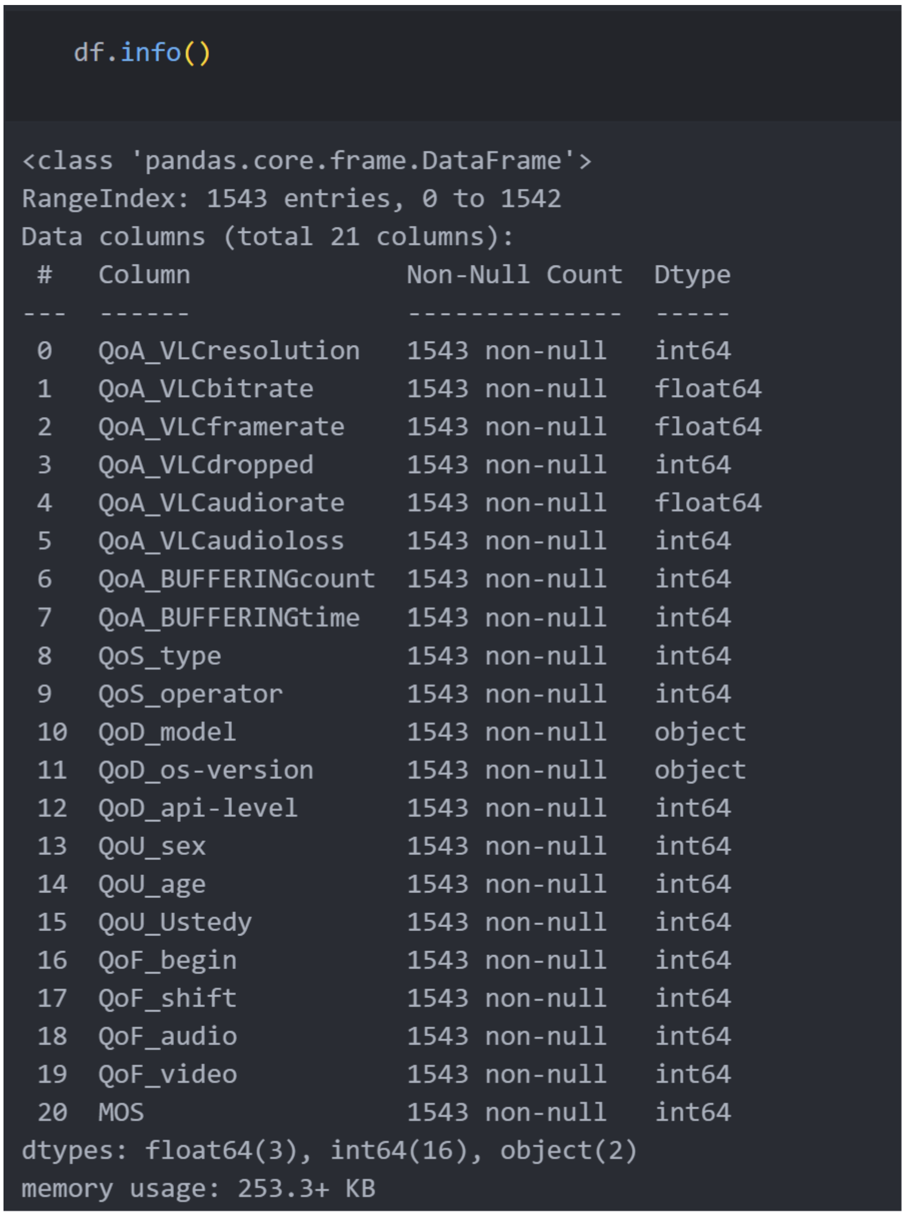
3. The imbalance ratio seems moderate but not highly skewed. Highest to lowest class ratio is:
   - 784/93 = 8.4x (MOS 4 / MOS 1)

4. We could consider oversampling minority classes or undersampling majority classes as part of data preprocessing to balance it further.

In summary, there is class imbalance present but not severely. I could use some sampling strategies to mitigate it so machine learning models do not get biased towards predicting only the dominant classes. We need sufficient samples across all classes for robust video QoE modeling. (This will be one of our experiments: ablation studies : the effects of SMOTE or without SMOTE on our target variable).

Now , it's time to do Data Preprocessing and Feature Selection!

Let's see our datasets and its cloumns' datatypes again by calling df.info() :

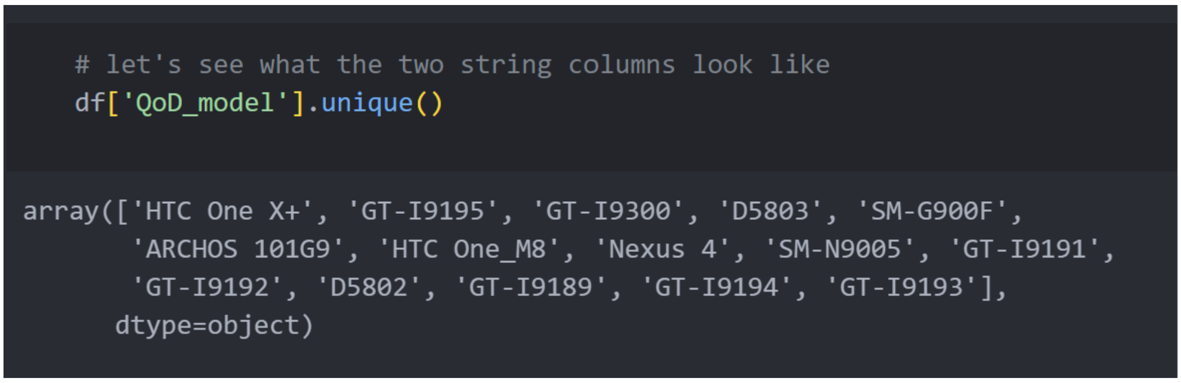


```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1543 entries, 0 to 1542
Data columns (total 21 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   QoA_VLCresolution   1543 non-null   int64  
 1   QoA_VLCbitrate      1543 non-null   float64
 2   QoA_VLCframerate    1543 non-null   float64
 3   QoA_VLCdropped      1543 non-null   int64  
 4   QoA_VLCaudiorate    1543 non-null   float64
 5   QoA_VLCaudioloss    1543 non-null   int64  
 6   QoA_BUFFERINGcount  1543 non-null   int64  
 7   QoA_BUFFERINGtime   1543 non-null   int64  
 8   QoS_type            1543 non-null   int64  
 9   QoS_operator        1543 non-null   int64  
 10  QoD_model           1543 non-null   object 
 11  QoD_os-version      1543 non-null   object 
 12  QoD_api-level       1543 non-null   int64  
 13  QoU_sex             1543 non-null   int64  
 14  QoU_age             1543 non-null   int64  
 15  QoU_Ustedy          1543 non-null   int64  
 16  QoF_begin           1543 non-null   int64  
 17  QoF_shift           1543 non-null   int64  
 18  QoF_audio           1543 non-null   int64  
 19  QoF_video           1543 non-null   int64  
 20  MOS                 1543 non-null   int64  
dtypes: float64(3), int64(16), object(2)
memory usage: 253.3+ KB
```

As we can see , now we will have to do some data preprocessing to make the data ready for modelling ; starting with the categorical features (we will have to convert them to numerical features) so that we can finally use them in our models or even run a correlation analysis on them to determine their features importance on our target variable ‘MOS’.

The first string column being QoD-model , its unique values are :



```
# let's see what the two string columns look like
df['QoD_model'].unique()
```

```
array(['HTC One X+', 'GT-I9195', 'GT-I9300', 'D5803', 'SM-G900F',
       'ARCHOS 101G9', 'HTC One_M8', 'Nexus 4', 'SM-N9005', 'GT-I9191',
       'GT-I9192', 'D5802', 'GT-I9189', 'GT-I9194', 'GT-I9193'],
      dtype=object)
```

The attribute `QoD_model` refers to the specific model of mobile device used during the video streaming sessions as per the dataset documentation.

The array of values we have called indicates there were 15 unique mobile device models that were used for crowdsourced testing of video Quality of Experience (QoE) by the 181 users.

Some observations:

- Many models seem to be Samsung Galaxy phones (GT-XXXX) and Sony Xperia phones (DXXXX)
- Mix of high-end (Galaxy S, HTC One), mid-range (Xperia) and low-end (Archos) phones
- Different OS platforms - Android (Samsung, Sony, Nexus), Windows (Archos)
- Range of phone form factors - Smartphones & tablets (Archos 101 tablet)

Having such diversity of devices is useful to evaluate if mobile device characteristics have an impact on perceived video streaming quality.

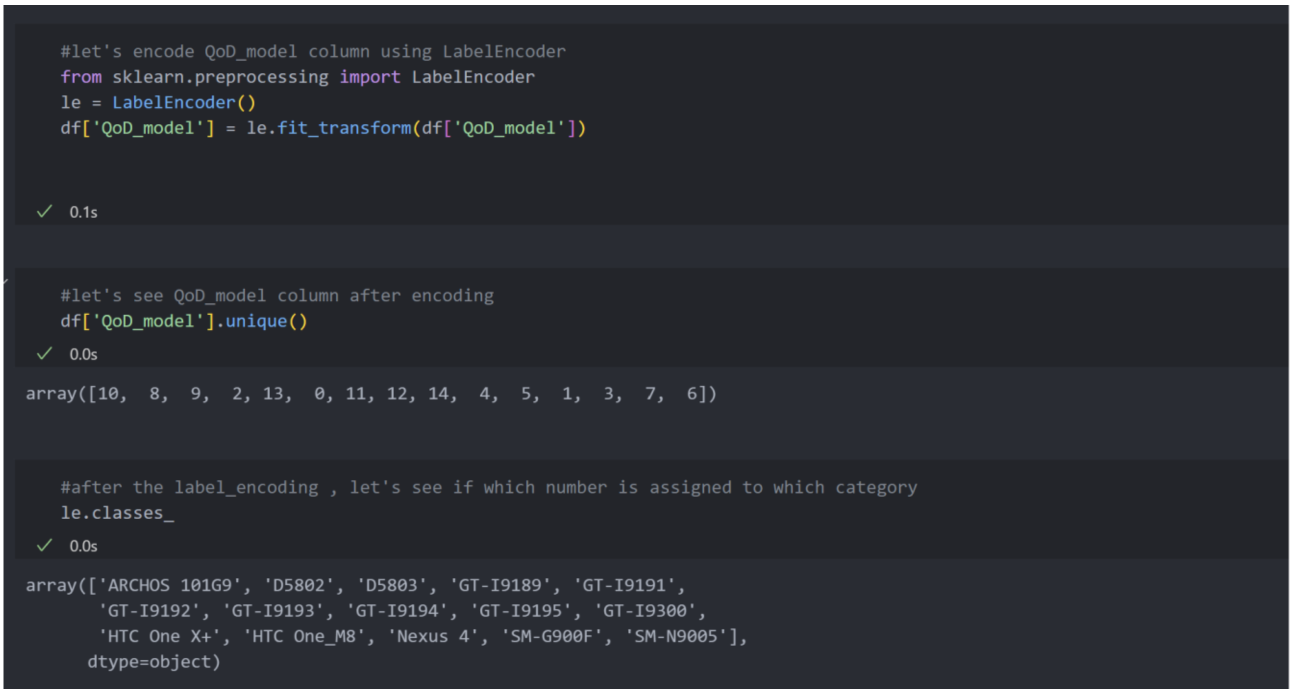
As part of data preprocessing, we could:

- Assign device performance tiers based on their specs (high, mid, low end)
- Group by OS platforms
- Create variables for form factor (phone vs tablet)

These additional attributes derived from QoD_model can help correlate device quality with video QoE scores given by users. They may explain some variability in MOS ratings.

In summary, the QoD_model attribute provides a rich set of device information that can be used to analyze the impact of mobile device characteristics on video streaming quality. So, surely it is a useful attribute to have in the dataset.

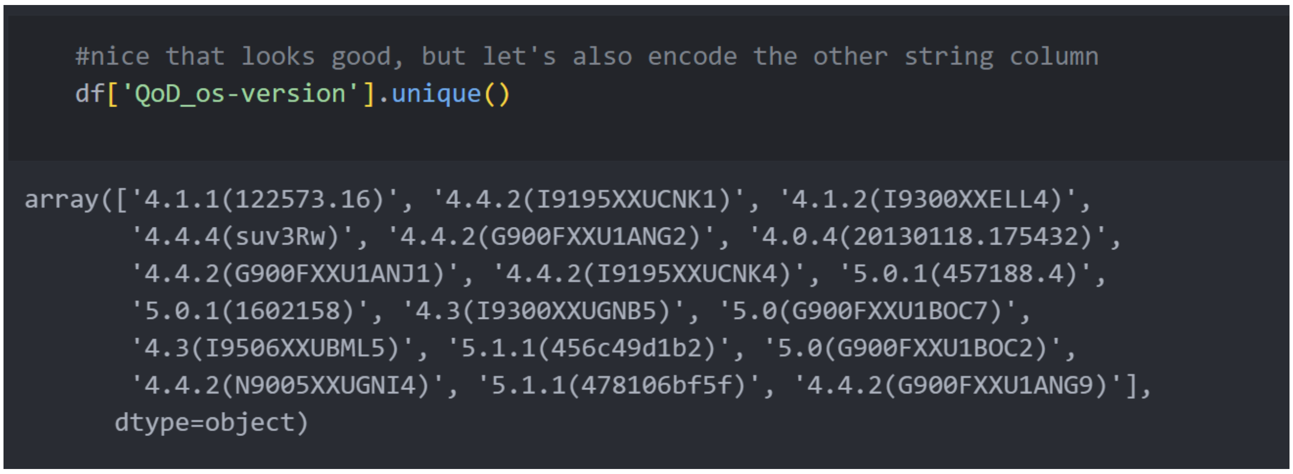
For us to turn the object type of the column `QoD_model` into a numerical type, we can use the `LabelEncoder` from the `sklearn.preprocessing` module.



```
#let's encode QoD_model column using LabelEncoder
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
df['QoD_model'] = le.fit_transform(df['QoD_model'])

✓ 0.1s

#let's see QoD_model column after encoding
df['QoD_model'].unique()
✓ 0.0s
array([10,  8,  9,  2, 13,  0, 11, 12, 14,  4,  5,  1,  3,  7,  6])

#after the label_encoding , let's see if which number is assigned to which category
le.classes_
✓ 0.0s
array(['ARCHOS 101G9', 'D5802', 'D5803', 'GT-I9189', 'GT-I9191',
       'GT-I9192', 'GT-I9193', 'GT-I9194', 'GT-I9195', 'GT-I9300',
       'HTC One X+', 'HTC One_M8', 'Nexus 4', 'SM-G900F', 'SM-N9005'],
      dtype=object)
```

As simple as that , it’s done easily. But , the other string/objects columns get a little bit challenging to encode. But,I could do it accordingly.

Another categorical feature that we will have to convert to numerical is the `QoD_os-version` feature.



```
#nice that looks good, but let's also encode the other string column
df['QoD_os-version'].unique()

array(['4.1.1(122573.16)', '4.4.2(I9195XXUCNK1)', '4.1.2(I9300XXELL4)',
       '4.4.4(suv3Rw)', '4.4.2(G900FXXU1ANG2)', '4.0.4(20130118.175432)',
       '4.4.2(G900FXXU1ANJ1)', '4.4.2(I9195XXUCNK4)', '5.0.1(457188.4)',
       '5.0.1(1602158)', '4.3(I9300XXUGNB5)', '5.0(G900FXXU1BOC7)',
       '4.3(I9506XXUBML5)', '5.1.1(456c49d1b2)', '5.0(G900FXXU1BOC2)',
       '4.4.2(N9005XXUGNI4)', '5.1.1(478106bf5f)', '4.4.2(G900FXXU1ANG9)'],
      dtype=object)
```

The `QoD_os-version` attribute refers to the operating system version installed on the mobile devices used for video streaming in the crowdsourced test.

The array of values indicates there were 18 unique OS versions across the different phones and tablets used.

A few key observations:

- Most are Android OS versions given the convention of the strings. Versions range from older 4.0 to newer 5.1.
- There are some device manufacturer/model specific firmware versions like Samsung's XXELL4 firmware.
- The OS versions correspond to the phones we saw under QoD_model like Galaxy S5, Xperia, etc.
- Some OS versions have patch levels also specified like 4.1.1 or security patch dates.

Such diversity of OS versions and their evolution over time can impact factors like:

- Device performance - processing, graphics, battery life
- Default app behaviors and settings
- Underlying driver and framework changes

These differences could influence quality perceptions during video streaming across user devices.

For modeling, we need to preprocess this data to extract the core OS version (4.1, 5.0) and possibly normalize the patch numbering. Categorization of versions by age may also help correlate with MOS.

Since there are 18 unique values, we have got to be careful while encoding this column.

I would follow " the traditional group them together and encode tactics" To preprocess the OS version data:

1. Extract Core OS Version
   - For each version string, extract the main OS version by parsing the number before the first decimal point.
   - So "5.1.1(478106bf5f)" becomes "5.1"

2. Bin Versions
   - Group versions into bins based on release timeframe:
     - Older than 4.4 -> "Old"
     - 4.4 to 5.0 -> "Intermediate"
     - Above 5.0 -> "New"

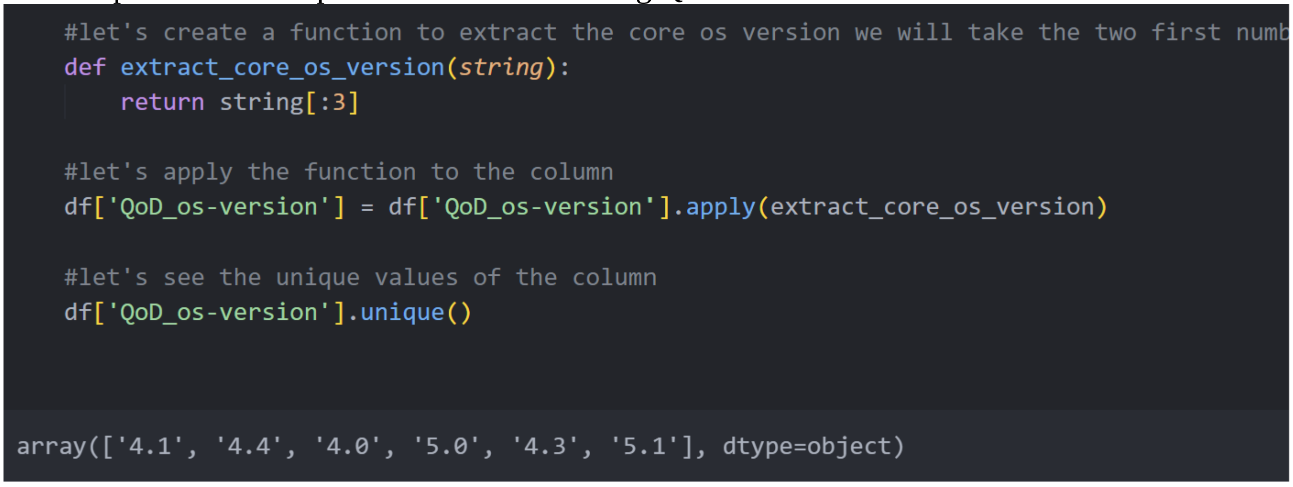
3. Label Encode
   - Assign numeric labels to these bins:
     - Old = 0
     - Intermediate = 1
     - New = 2

4. Use Encoded Column
   - Include this new numeric OS version encoding in prediction models.

Key Benefits:

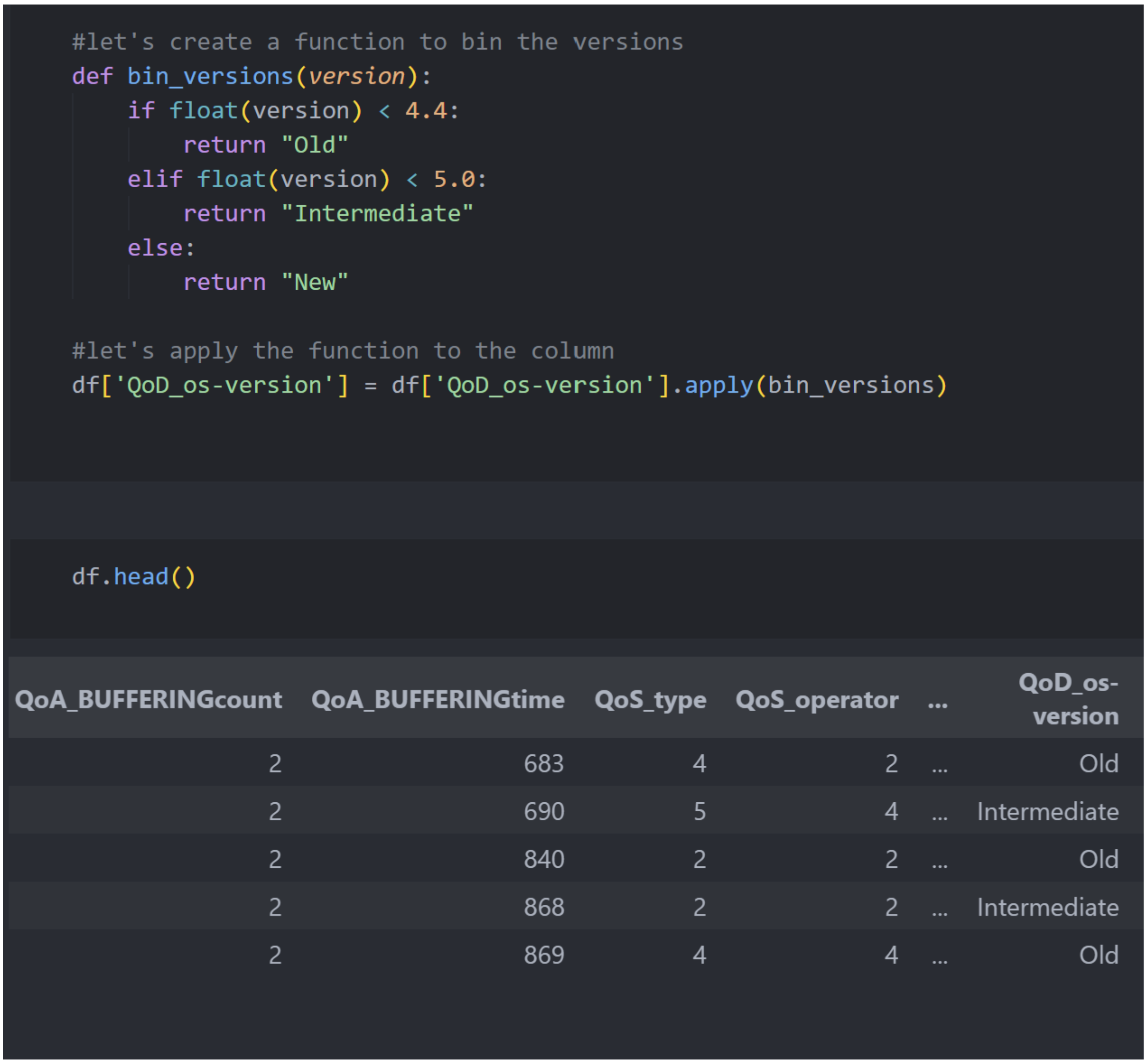
- Reduces sparsity of many unique but similar versions
- Generalizes OS based on age rather than minor differences
- Simplifies modeling and identifying trends

Additionally, we can extract specific metadata like security patch date where available. That may show impact of device updates on video streaming QoE.



```
#let's create a function to extract the core os version we will take the two first numb
def extract_core_os_version(string):
    return string[:3]

#let's apply the function to the column
df['QoD_os-version'] = df['QoD_os-version'].apply(extract_core_os_version)

#let's see the unique values of the column
df['QoD_os-version'].unique()
```

```
array(['4.1', '4.4', '4.0', '5.0', '4.3', '5.1'], dtype=object)
```

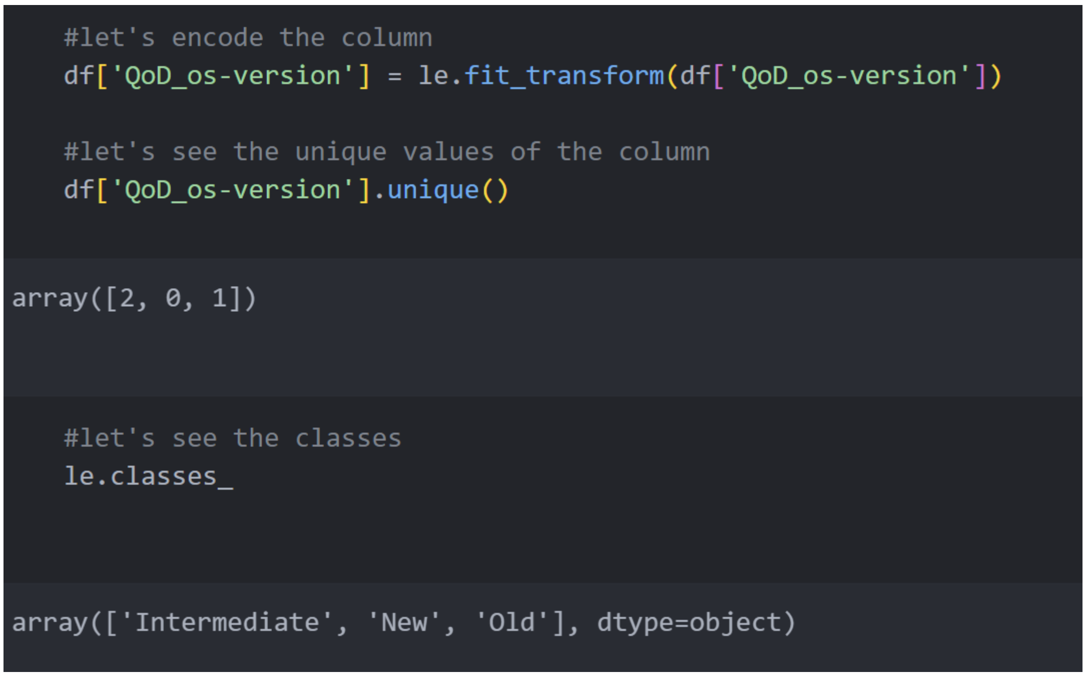


```python
#let's create a function to bin the versions
def bin_versions(version):
    if float(version) < 4.4:
        return "Old"
    elif float(version) < 5.0:
        return "Intermediate"
    else:
        return "New"

#let's apply the function to the column
df['QoD_os-version'] = df['QoD_os-version'].apply(bin_versions)
```

```python
df.head()
```

QoA_BUFFERINGcount	QoA_BUFFERINGtime	QoS_type	QoS_operator	...	QoD_os-version
2	683	4	2	...	Old
2	690	5	4	...	Intermediate
2	840	2	2	...	Old
2	868	2	2	...	Intermediate
2	869	4	4	...	Old

Now, we will have to encode these new columns again to numbers 0,1 and 2.



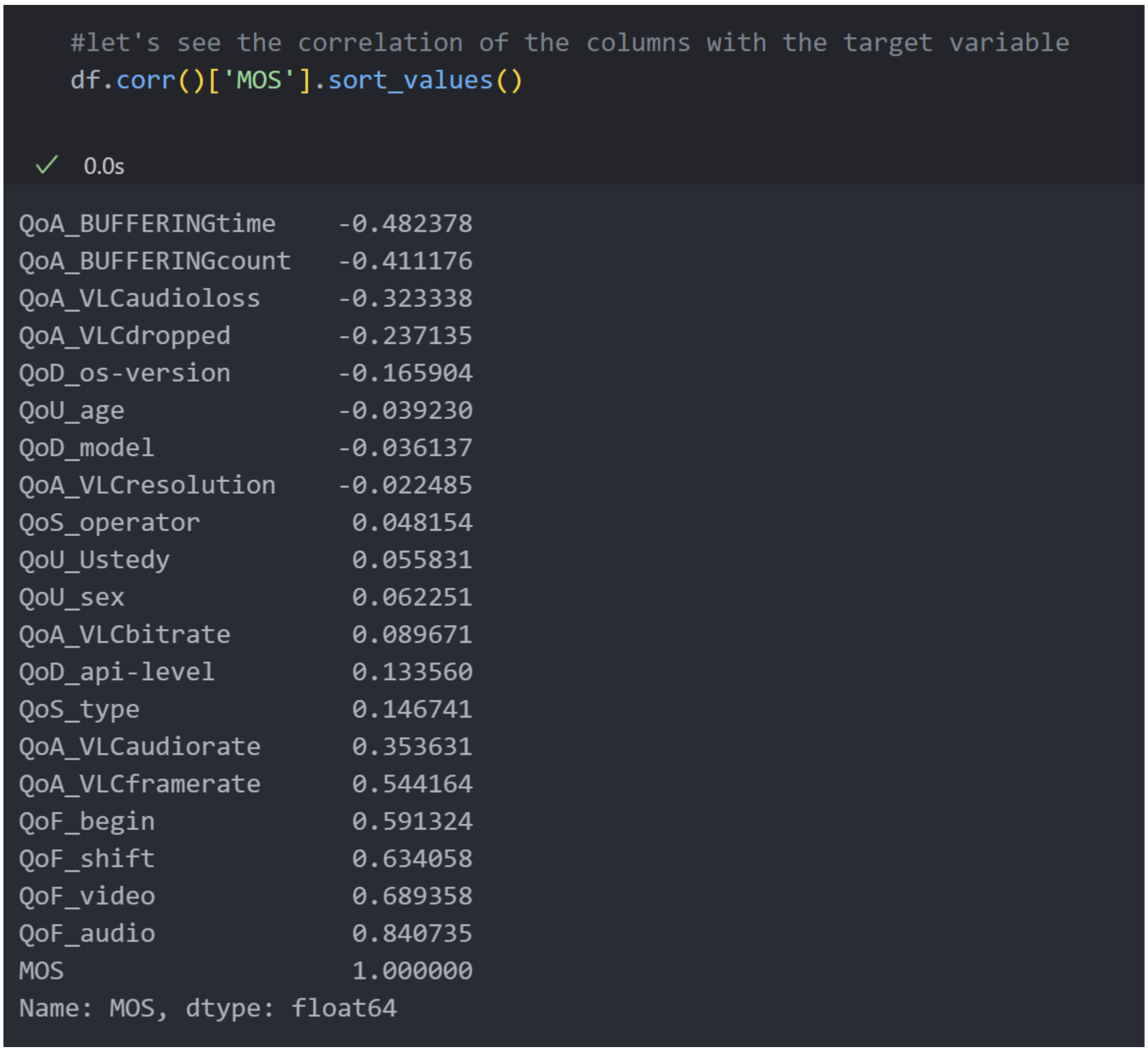
```
#let's encode the column
df['QoD_os-version'] = le.fit_transform(df['QoD_os-version'])

#let's see the unique values of the column
df['QoD_os-version'].unique()
```

```
array([2, 0, 1])
```

```
#let's see the classes
le.classes_
```

```
array(['Intermediate', 'New', 'Old'], dtype=object)
```

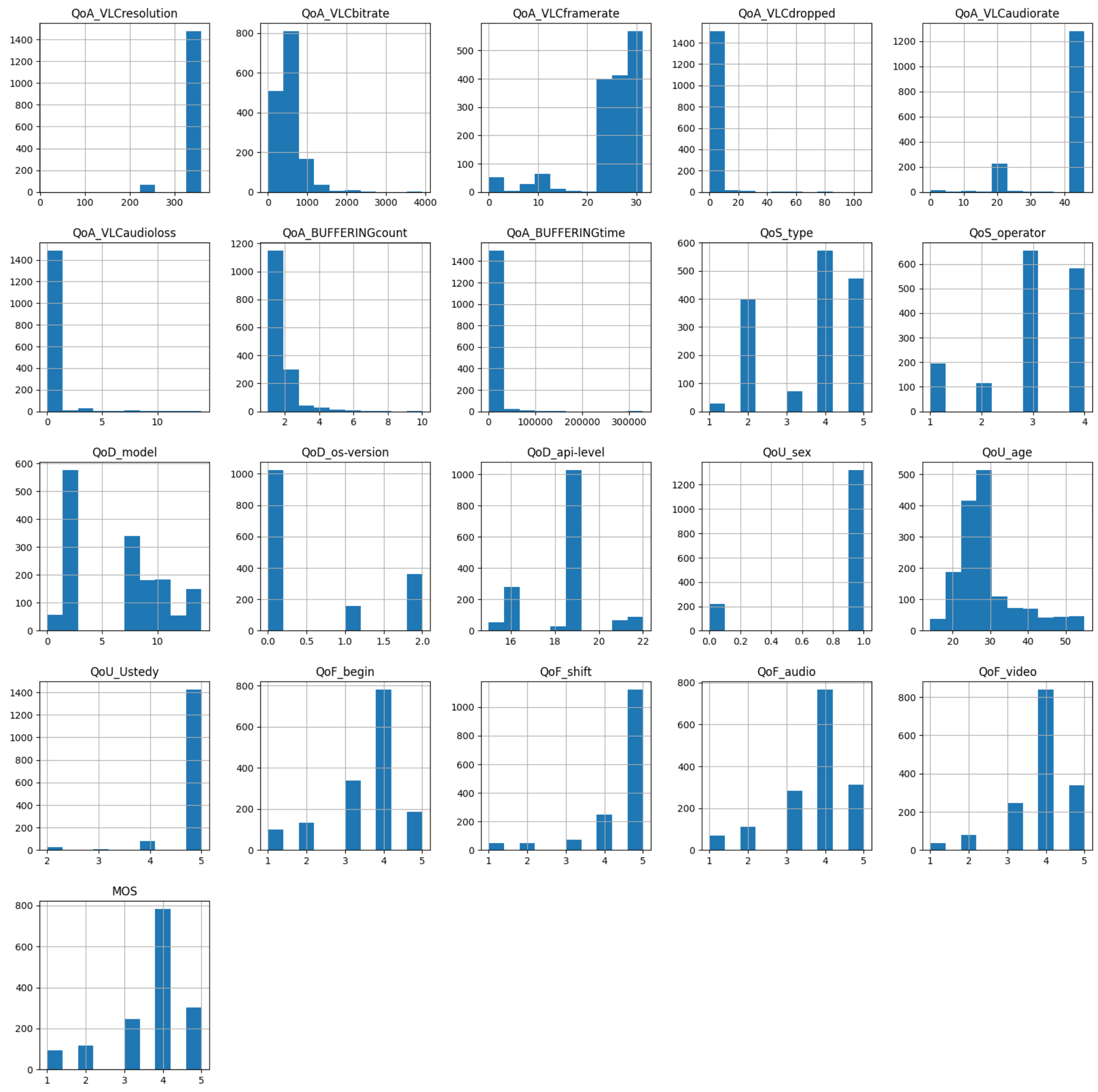
Finally we can calculate the correlation between the features and the target variable `MOS` to see which features are more important than others.

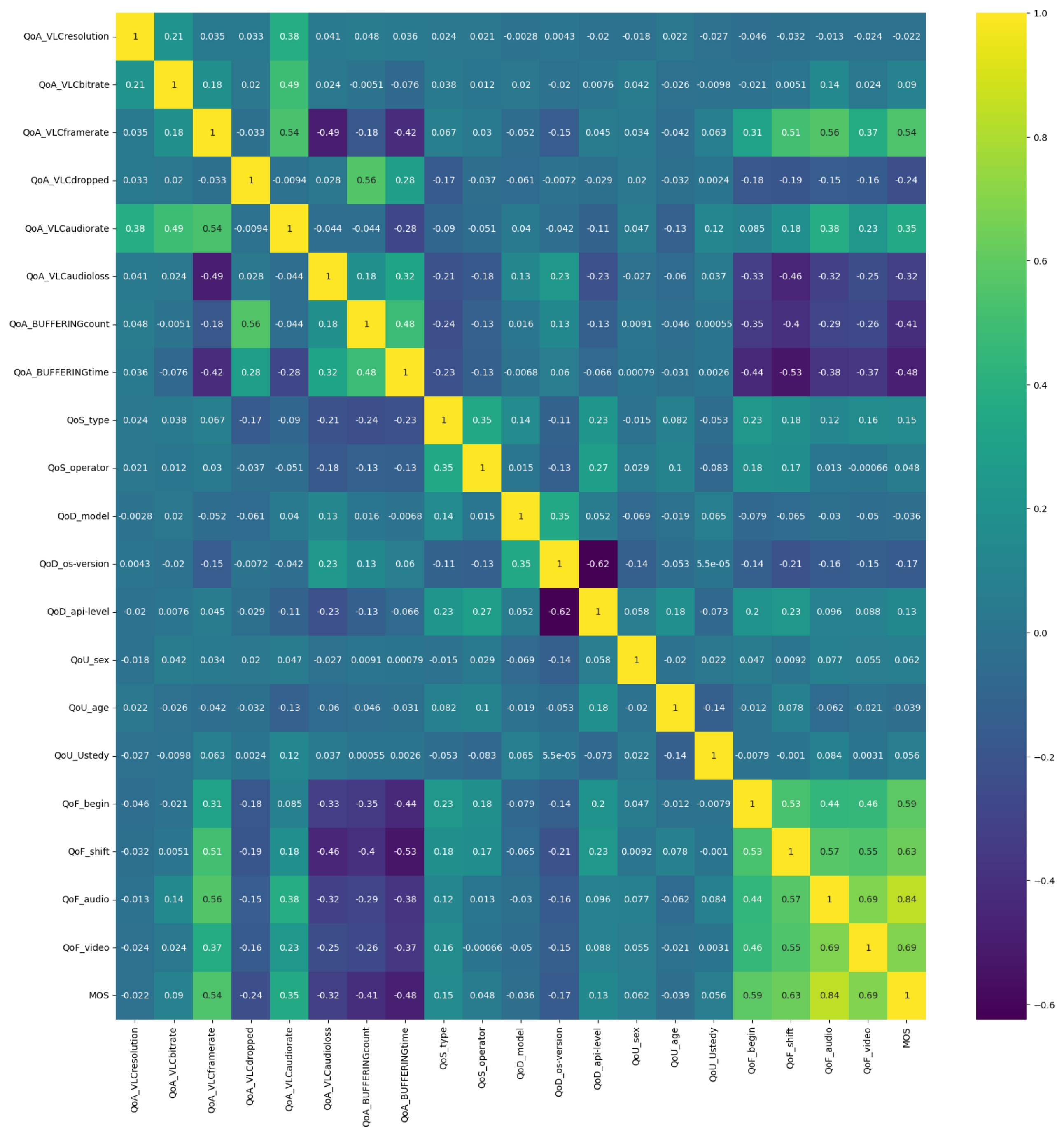


```
#let's see the correlation of the columns with the target variable
df.corr()['MOS'].sort_values()
```

✓ 0.0s

```
QoA_BUFFERINGtime    -0.482378
QoA_BUFFERINGcount   -0.411176
QoA_VLCaudioloss     -0.323338
QoA_VLCdropped       -0.237135
QoD_os-version       -0.165904
QoU_age              -0.039230
QoD_model            -0.036137
QoA_VLCresolution    -0.022485
QoS_operator          0.048154
QoU_Ustedy            0.055831
QoU_sex               0.062251
QoA_VLCbitrate        0.089671
QoD_api-level         0.133560
QoS_type              0.146741
QoA_VLCaudiorate      0.353631
QoA_VLCframerate      0.544164
QoF_begin             0.591324
QoF_shift             0.634058
QoF_video             0.689358
QoF_audio             0.840735
MOS                   1.000000
Name: MOS, dtype: float64
```


QoA_VLCresolution
QoA_VLCbitrate
QoA_VLCframerate
QoA_VLCdropped
QoA_VLCaudiorate
QoA_VLCaudioloss
QoA_BUFFERINGcount
QoA_BUFFERINGtime
QoS_type
QoS_operator
QoD_model
QoD_os-version
QoD_api-level
QoU_sex
QoU_age
QoU_Ustedy
QoF_begin
QoF_shift
QoF_audio
QoF_video
MOS



Based on these three graphs that I have come up, we will move on to Feature Selection:

we will Drop these (correlation < 0.1):

- QoU_age
- QoD_model
- QoA_resolution

- QoS_operator
- QoU_study
- QoU_sex

Additionally, we will drop these negatively correlated ones:

- QoA_BUFFERINGtime
- QoA_BUFFERINGcount
- QoA_VLCaudioloss
- QoA_VLCdropped
- QoD_os-version

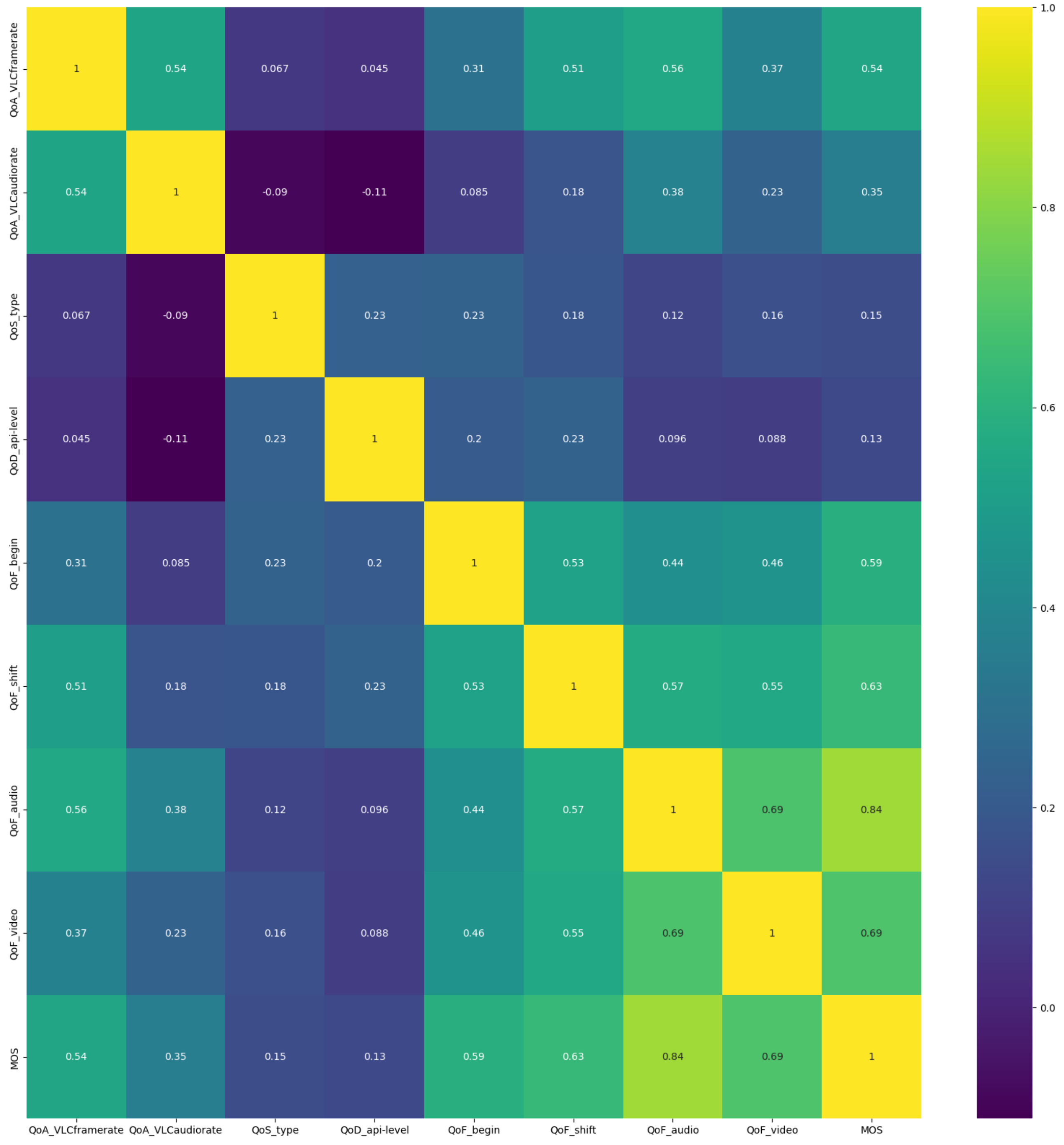
So in total 11 features to drop.

Key reasons:

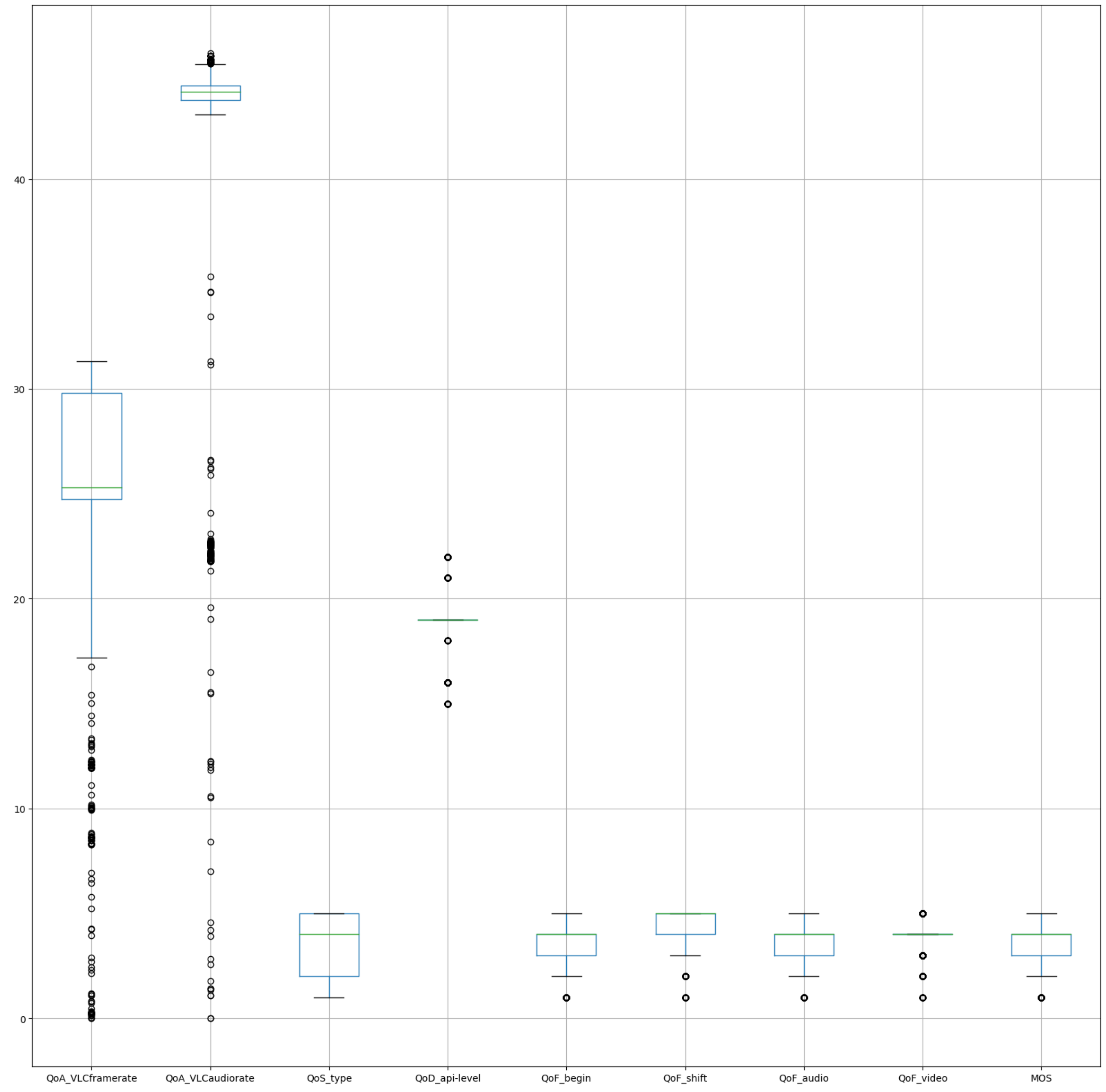
- we will Focus modeling on relevant and predictive features
- we will Remove redundant features
- we will Simplify model building

The remaining positively correlated features cover video quality, network, device and user feedback related factors.

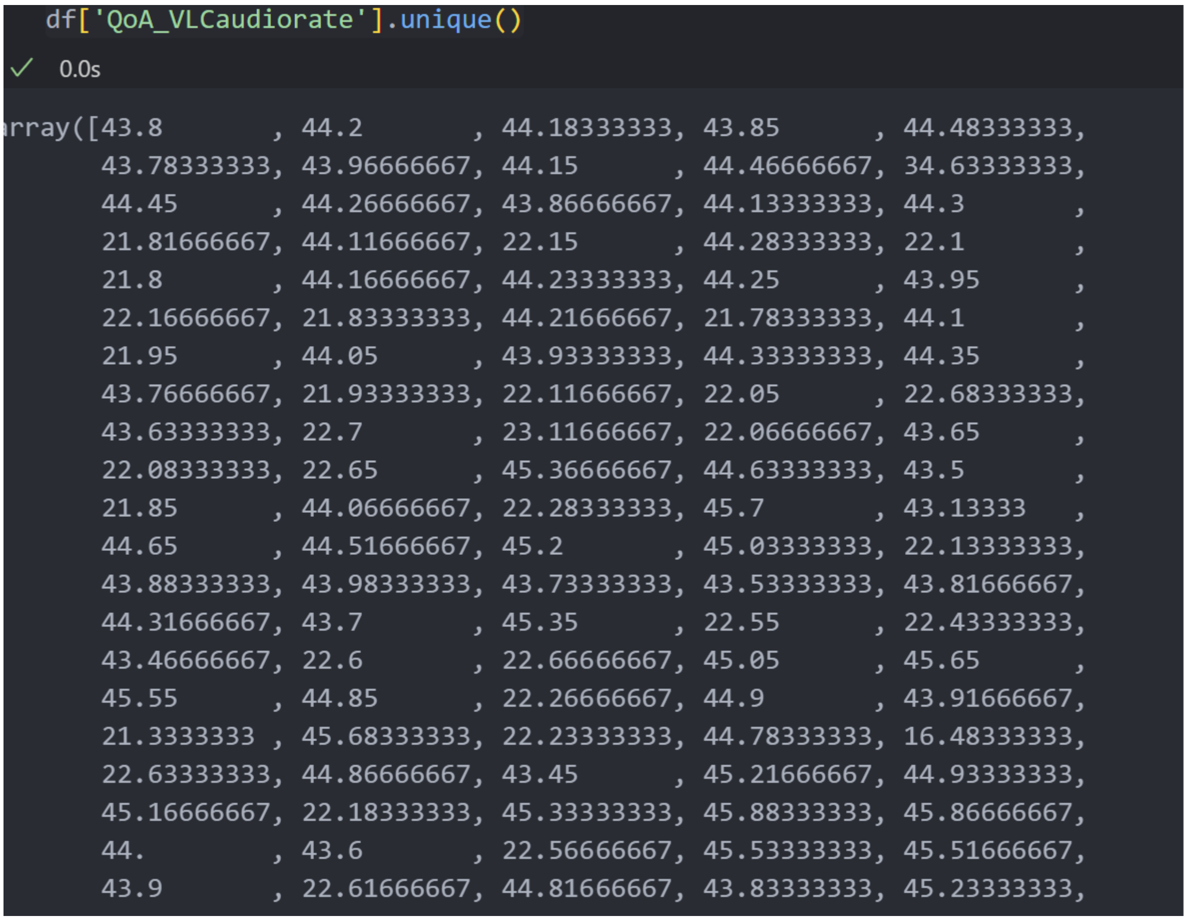
After we have dropped all of them, we have these healthy and beautiful correlation heatmap :



When called the boxplot for our features :



We can see that there are some anomalies in the data like QoA_VLCbitrate and QoD_api-level



```
df['QoA_VLCaudiorate'].unique()
```

✓ 0.0s

```
rray([43.8       , 44.2       , 44.18333333, 43.85      , 44.48333333,
      43.78333333, 43.96666667, 44.15      , 44.46666667, 34.63333333,
      44.45      , 44.26666667, 43.86666667, 44.13333333, 44.3       ,
      21.81666667, 44.11666667, 22.15      , 44.28333333, 22.1       ,
      21.8       , 44.16666667, 44.23333333, 44.25      , 43.95      ,
      22.16666667, 21.83333333, 44.21666667, 21.78333333, 44.1       ,
      21.95      , 44.05      , 43.93333333, 44.33333333, 44.35      ,
      43.76666667, 21.93333333, 22.11666667, 22.05      , 22.68333333,
      43.63333333, 22.7       , 23.11666667, 22.06666667, 43.65      ,
      22.08333333, 22.65      , 45.36666667, 44.63333333, 43.5       ,
      21.85      , 44.06666667, 22.28333333, 45.7       , 43.13333   ,
      44.65      , 44.51666667, 45.2       , 45.03333333, 22.13333333,
      43.88333333, 43.98333333, 43.73333333, 43.53333333, 43.81666667,
      44.31666667, 43.7       , 45.35      , 22.55      , 22.43333333,
      43.46666667, 22.6       , 22.66666667, 45.05      , 45.65      ,
      45.55      , 44.85      , 22.26666667, 44.9       , 43.91666667,
      21.3333333 , 45.68333333, 22.23333333, 44.78333333, 16.48333333,
      22.63333333, 44.86666667, 43.45      , 45.21666667, 44.93333333,
      45.16666667, 22.18333333, 45.33333333, 45.88333333, 45.86666667,
      44.        , 43.6       , 22.56666667, 45.53333333, 45.51666667,
      43.9       , 22.61666667, 44.81666667, 43.83333333, 45.23333333,
```

The `QoA_VLCaudiorate` column provides the audio bitrate information for the videos streamed during the QoE test sessions, based on my understanding from the dataset documentation READme.md.

Some key points about this feature I can share:

- Audio bitrate indicates the number of bits processed per second for encoding the audio track of the videos
- Higher audio bitrates can enable better sound quality, multi-channel audio etc.

Looking at the wide range of unique values we observed for this column, it indicates:

- The test videos had audio tracks with very different bitrate encoding levels
- This could be because various source videos with diverse audio quality were selected
- Bitrates likely adapted to network conditions - hence increased variability

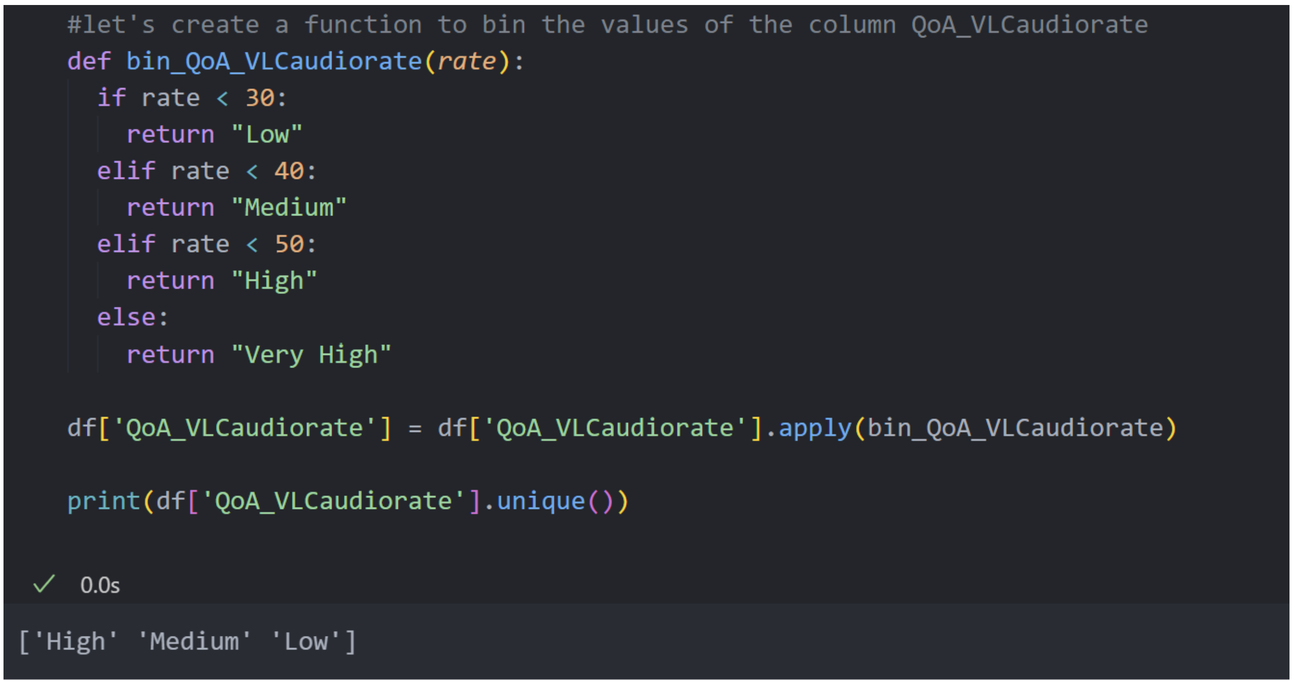
While audio influences overall user experience, research shows video parameters predominantly drive perceived quality.

However, analyzing audio bitrate changes and correlations to MOS can still provide useful insights e.g:

- User sensitivity to audio quality
- Tradeoffs between audio and video bitrates
- Audio robustness during adaptation triggers

So, It’s time to do some feature engineering. We will create a new column called QoA_VLCaudiorate_bins and bin the values of the column QoA_VLCaudiorate

- 0-1000 -> "Low"
- 1000-2000 -> "Medium"
- 2000-3000 -> "High"
- 3000-4000 -> "Very High"
- 4000-5000 -> "Ultra High"

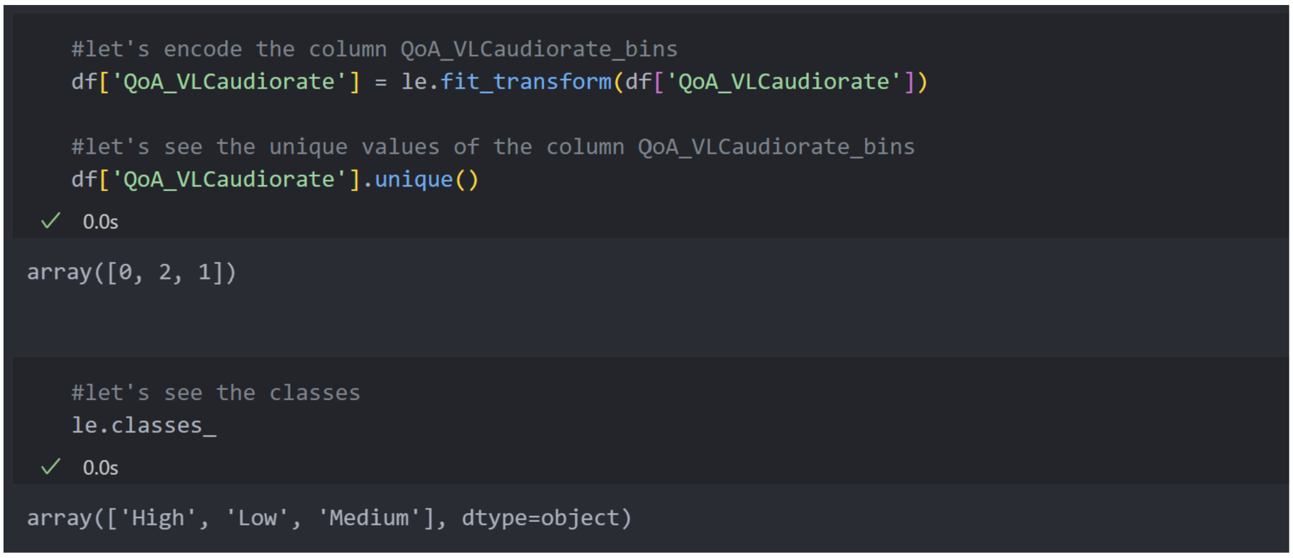


```
#let's create a function to bin the values of the column QoA_VLCaudiorate
def bin_QoA_VLCaudiorate(rate):
  if rate < 30:
    return "Low"
  elif rate < 40:
    return "Medium"
  elif rate < 50:
    return "High"
  else:
    return "Very High"

df['QoA_VLCaudiorate'] = df['QoA_VLCaudiorate'].apply(bin_QoA_VLCaudiorate)

print(df['QoA_VLCaudiorate'].unique())
```

✓ 0.0s

```
['High' 'Medium' 'Low']
```

Now let’s encode .



```
#let's encode the column QoA_VLCaudiorate_bins
df['QoA_VLCaudiorate'] = le.fit_transform(df['QoA_VLCaudiorate'])

#let's see the unique values of the column QoA_VLCaudiorate_bins
df['QoA_VLCaudiorate'].unique()
```

✓ 0.0s

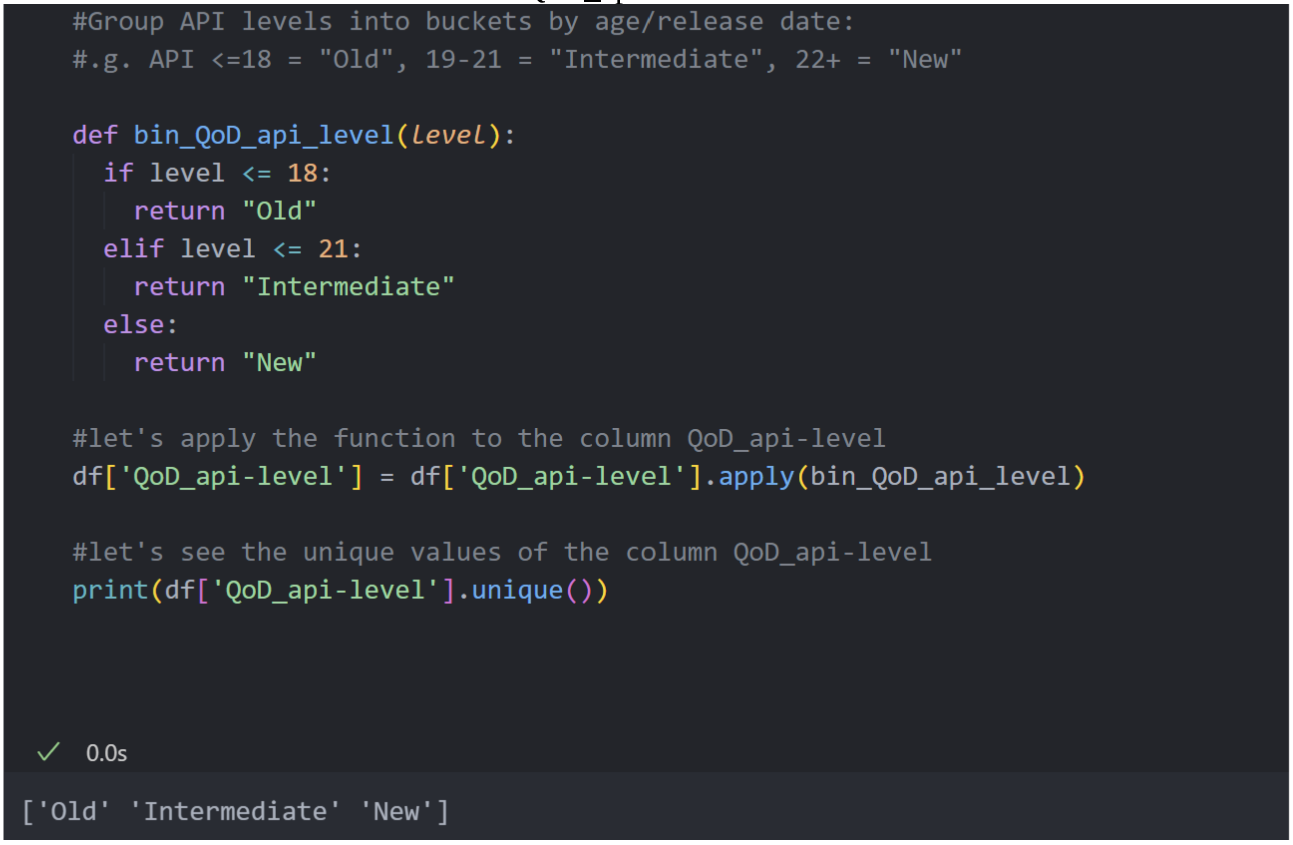
```
array([0, 2, 1])
```

```
#let's see the classes
le.classes_
```

✓ 0.0s

```
array(['High', 'Low', 'Medium'], dtype=object)
```

This is lovely : all that mess of a distribution is now much more organized and unified.
We will have to do the same for column QoD_api-level.

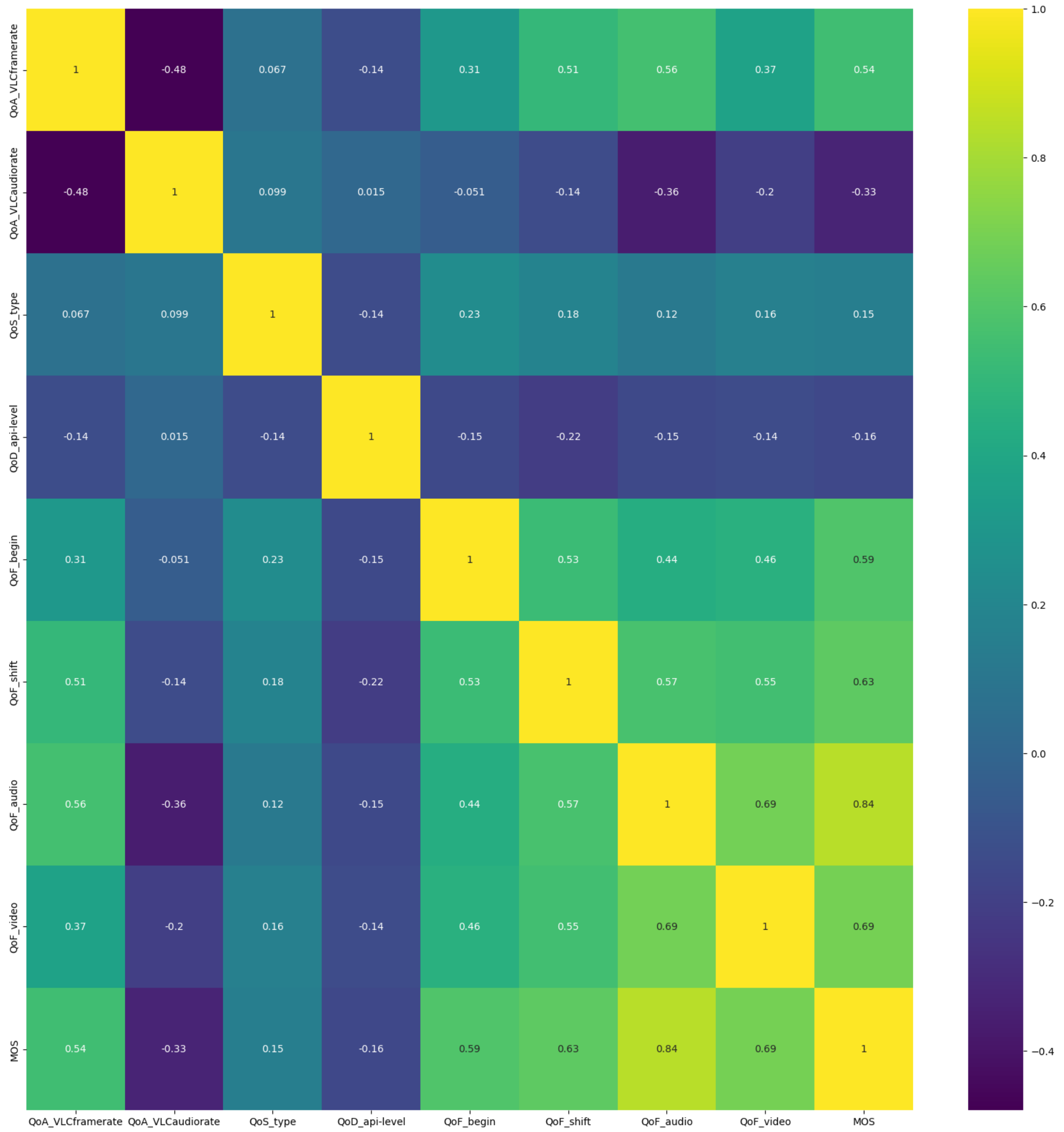


```
#Group API levels into buckets by age/release date:
#.g. API <=18 = "Old", 19-21 = "Intermediate", 22+ = "New"

def bin_QoD_api_level(level):
    if level <= 18:
        return "Old"
    elif level <= 21:
        return "Intermediate"
    else:
        return "New"

#let's apply the function to the column QoD_api-level
df['QoD_api-level'] = df['QoD_api-level'].apply(bin_QoD_api_level)

#let's see the unique values of the column QoD_api-level
print(df['QoD_api-level'].unique())
```

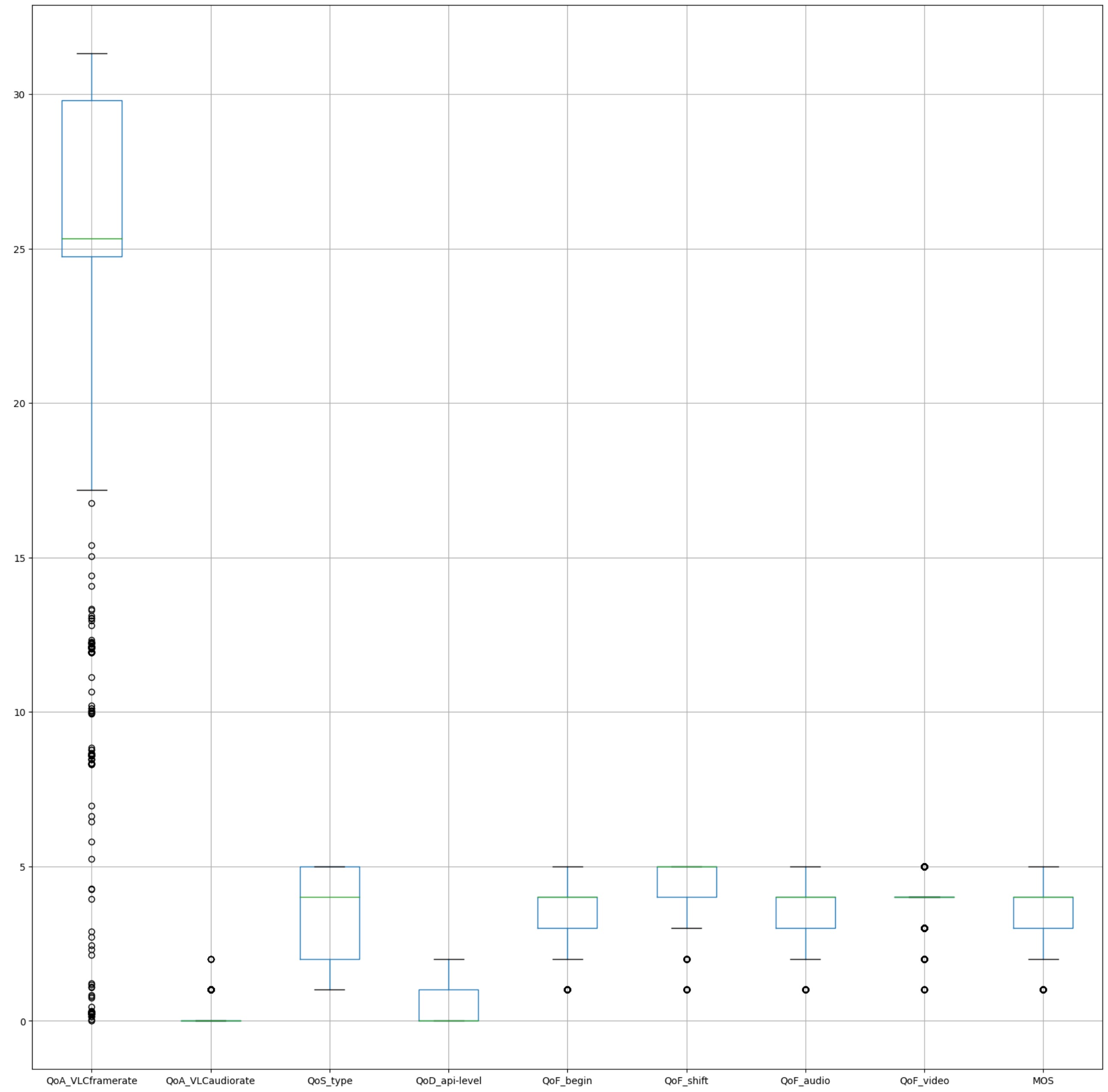
✓ 0.0s

```
['Old' 'Intermediate' 'New']
```

Finally, we have ourselves our permanent features even though after much of Data Engineering, our two columns suffer a loss in correlation matrix to ‘MOS’.

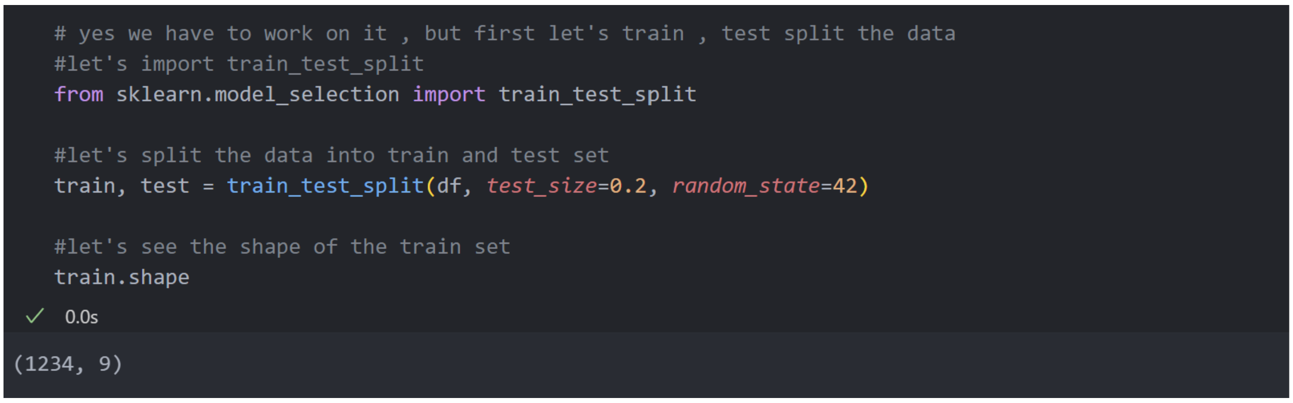


But, we won’t drop the QoA_VLCaudiorate and QOD_api-level features since it is highly correlated with MOS before and after the transformation since it is a categorical feature and we have already encoded it using the LabelEncoder. We will keep it for now and see how it performs in our models.



But, based on this new boxplot, we can say that the two distributions have tremendously improved a lot!

Finally we now have a good dataset for us to train, test and validate our models on.



```
# yes we have to work on it , but first let's train , test split the data
#let's import train_test_split
from sklearn.model_selection import train_test_split

#let's split the data into train and test set
train, test = train_test_split(df, test_size=0.2, random_state=42)

#let's see the shape of the train set
train.shape
✓ 0.0s
(1234, 9)
```

Moving on to our models and algorithms:
our main goals and potential experiments

Some potential goals I would pursue are:

1. Predict Video Streaming QoE
- Build machine learning models to predict Mean Opinion Score (MOS) based on the various influence factors
- Identify the most important factors impacting QoE and their thresholds
- Evaluate different ML algorithms and compare performance

2. Analyze Impact of Factors on QoE (already done above)
- Statistical analysis to quantify the correlation and effect size of each factor (video bitrate, buffering, etc.) on MOS
- Identify interactions between factors that influence QoE

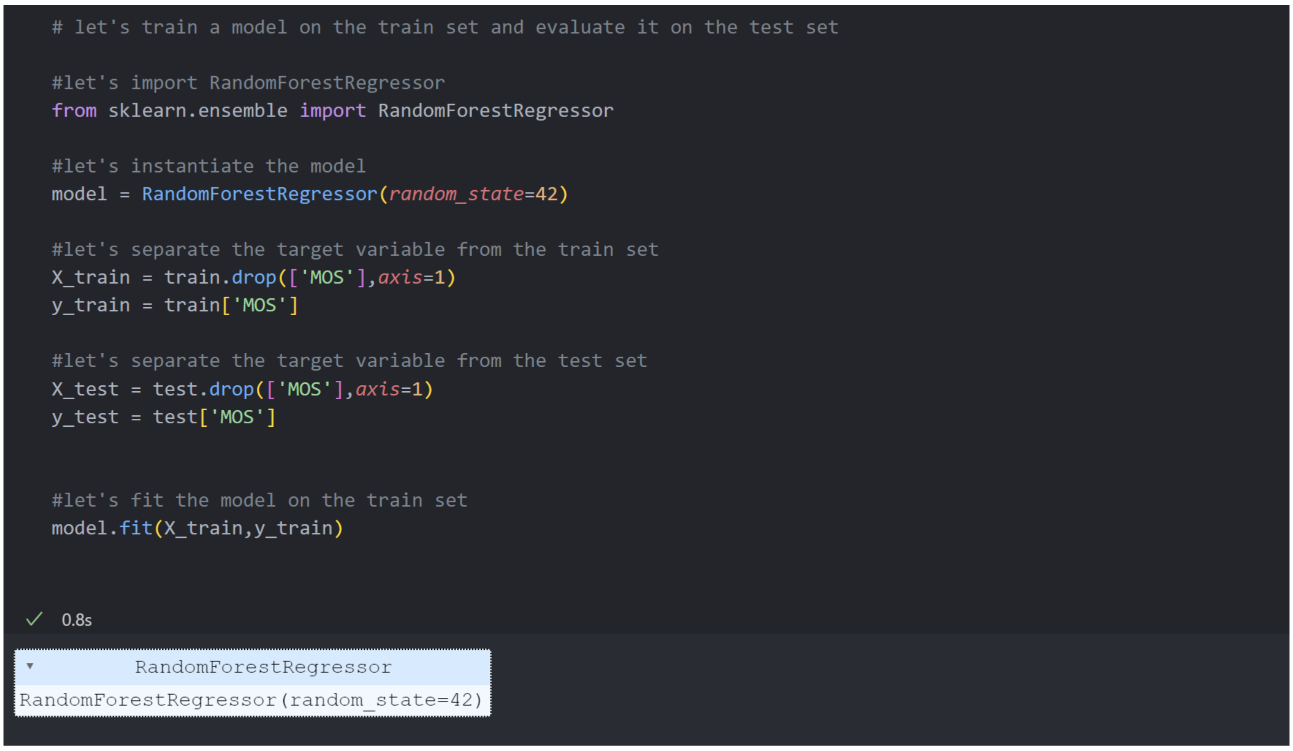
3. Segment Users by QoE
- Apply clustering techniques to group users based on their QoE patterns
- Develop user profiles most likely to perceive poor/good QoE

---

1. Predict Video Streaming QoE

Starting with our Primary Goal , *Predict Video Streaming QoE*

I choose Random Forests Regressor as my first Regression model on my train_data.

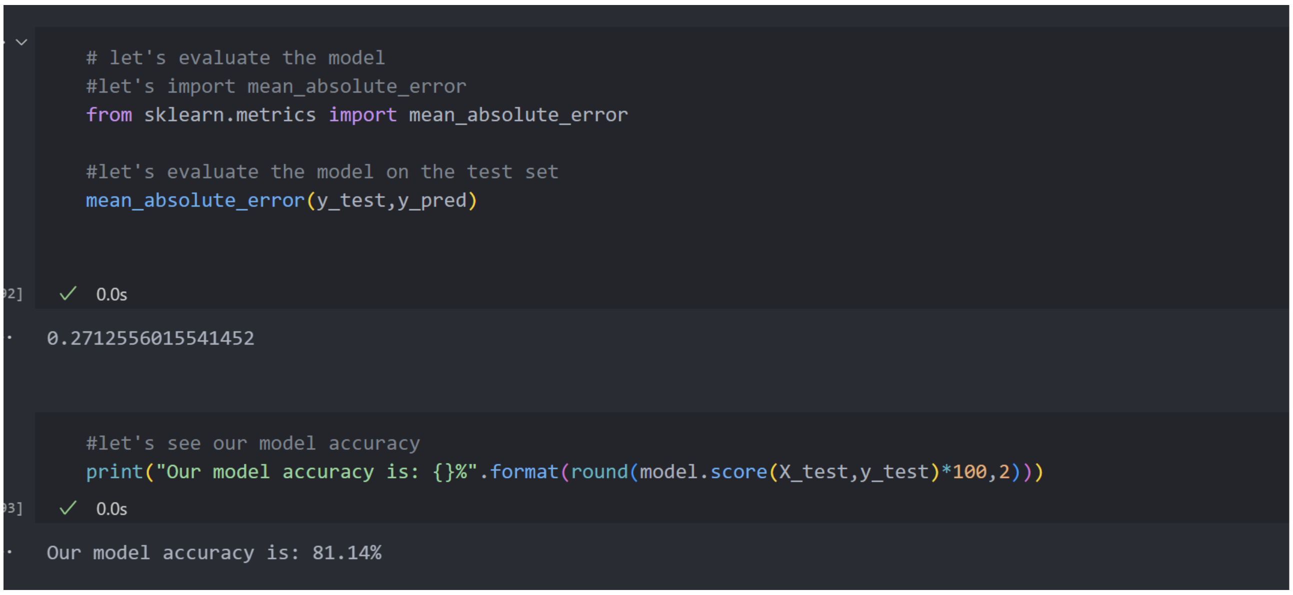


```
# let's train a model on the train set and evaluate it on the test set

#let's import RandomForestRegressor
from sklearn.ensemble import RandomForestRegressor

#let's instantiate the model
model = RandomForestRegressor(random_state=42)

#let's separate the target variable from the train set
X_train = train.drop(['MOS'],axis=1)
y_train = train['MOS']

#let's separate the target variable from the test set
X_test = test.drop(['MOS'],axis=1)
y_test = test['MOS']


#let's fit the model on the train set
model.fit(X_train,y_train)
```

```
✓ 0.8s
▾           RandomForestRegressor
RandomForestRegressor(random_state=42)
```

Why Random Forests Regressor? Because it is a powerful ensemble learning algorithm that can handle both categorical and numerical features. It is also robust to outliers and can handle non-linear relationships between features and target variable. It is also easy to interpret and explain to stakeholders. It is a good starting point for modeling video streaming QoE.

Since our dataset having outliers and somewhat skewed distributions, we will use the Random Forest Regressor algorithm to train our model on the dataset and see how it performs on the dataset as it is as our first Regression model.



```
# let's evaluate the model
#let's import mean_absolute_error
from sklearn.metrics import mean_absolute_error

#let's evaluate the model on the test set
mean_absolute_error(y_test,y_pred)
```

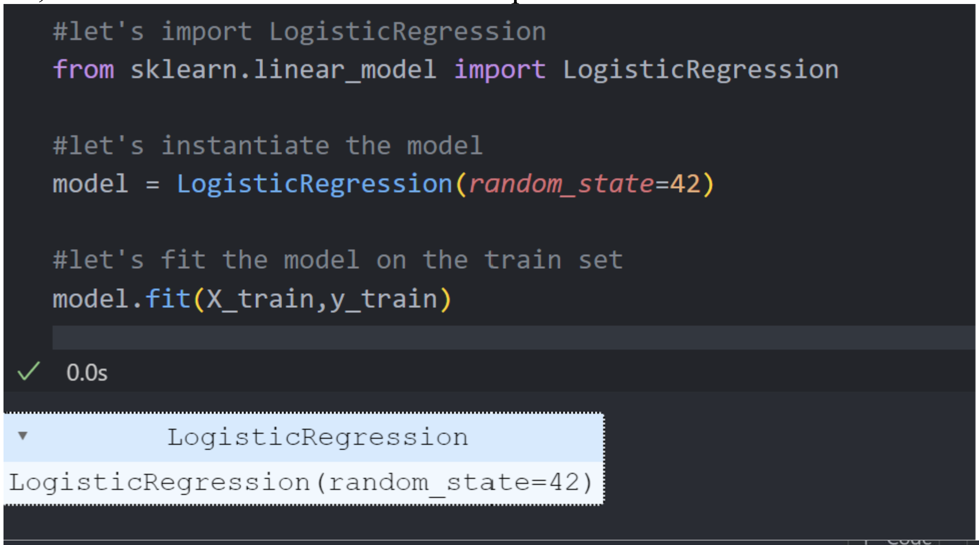
```
2]  ✓ 0.0s
0.2712556015541452
```

```
#let's see our model accuracy
print("Our model accuracy is: {}%".format(round(model.score(X_test,y_test)*100,2)))
```

```
3]  ✓ 0.0s
Our model accuracy is: 81.14%
```

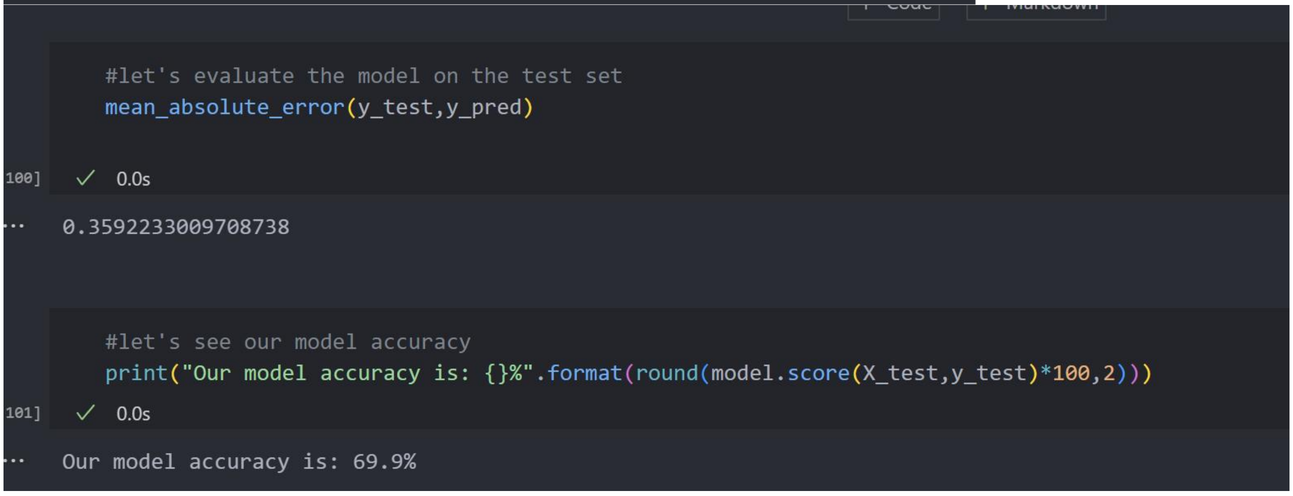
With a simple Random Forest Regressor model, accuracy of 81.14% is good enough. We can get a baseline performance for predicting video streaming QoE based on the various influence factors. We can then compare this with more complex models to see if they provide better performance.

But, note that our target variable is not continuous values! They are classes form 0-4
So, it's obvious that it's a classification problem. We should start out with logistics regression.



```
#let's import LogisticRegression
from sklearn.linear_model import LogisticRegression

#let's instantiate the model
model = LogisticRegression(random_state=42)

#let's fit the model on the train set
model.fit(X_train,y_train)
```

```
✓ 0.0s

▾ LogisticRegression
LogisticRegression(random_state=42)
```



```
#let's evaluate the model on the test set
mean_absolute_error(y_test,y_pred)
```

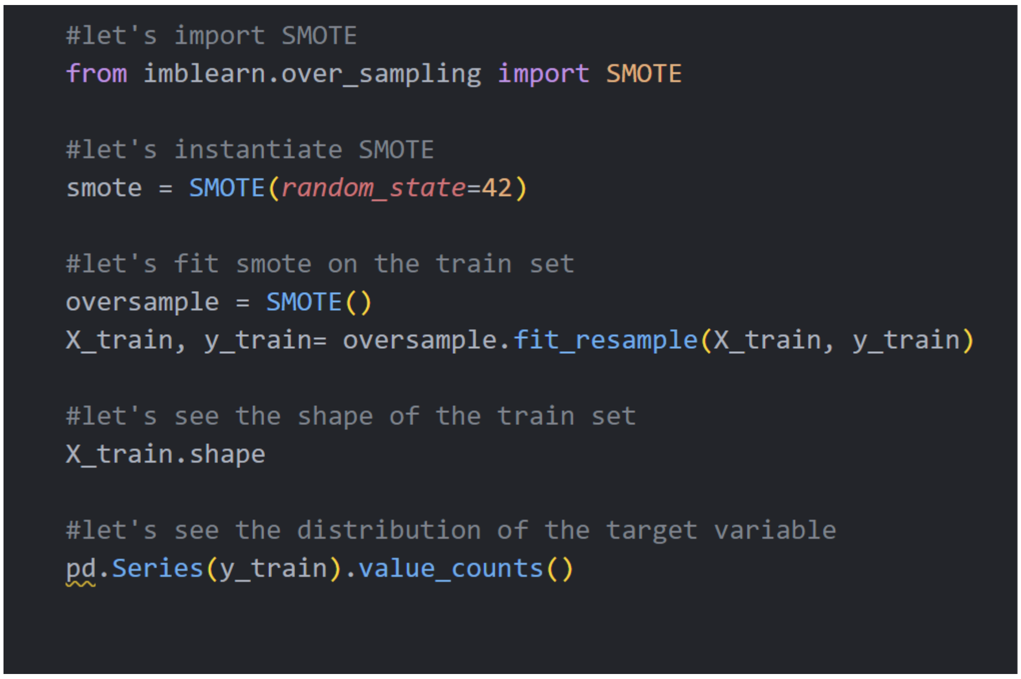
```
[100] ✓ 0.0s

··· 0.3592233009708738
```

```
#let's see our model accuracy
print("Our model accuracy is: {}%".format(round(model.score(X_test,y_test)*100,2)))
```

```
[101] ✓ 0.0s

··· Our model accuracy is: 69.9%
```

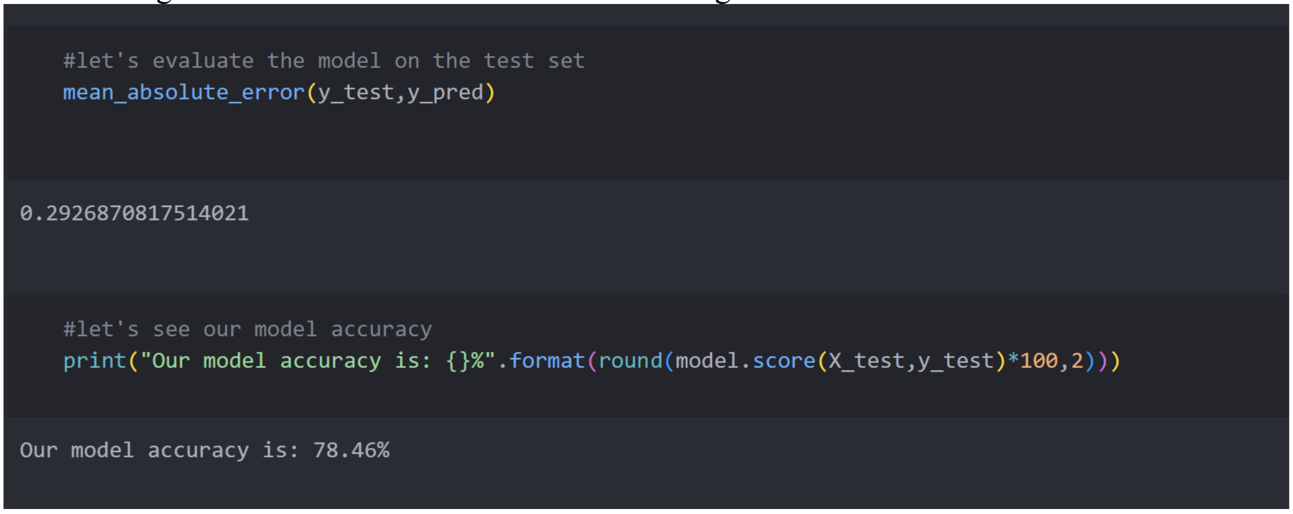
You can see that our model accuracy has decreased to 69.9 which is perfect(our model isn't overfitting or underfitting like others) ! Now we don't even have to use SMOTE(downsample or upsample our target classes) to balance the classes!

But,it's worth a shot! We will simply use SMOTE to upsample all target variables.



```
#let's import SMOTE
from imblearn.over_sampling import SMOTE

#let's instantiate SMOTE
smote = SMOTE(random_state=42)

#let's fit smote on the train set
oversample = SMOTE()
X_train, y_train= oversample.fit_resample(X_train, y_train)

#let's see the shape of the train set
X_train.shape

#let's see the distribution of the target variable
pd.Series(y_train).value_counts()
```

Let’s train again on our first model : Random Forest Regressor !



```
#let's evaluate the model on the test set
mean_absolute_error(y_test,y_pred)
```

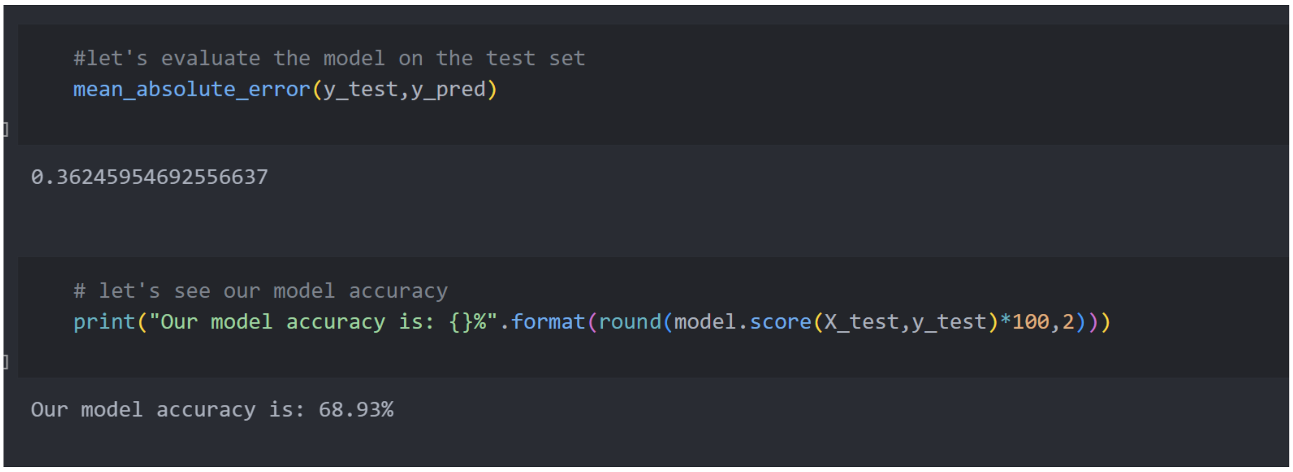
```
0.2926870817514021
```

```
#let's see our model accuracy
print("Our model accuracy is: {}%".format(round(model.score(X_test,y_test)*100,2)))
```

```
Our model accuracy is: 78.46%
```

See! The accuracy has dropped down to 78.46% which is good! It’s a sign that our model without SMOTE was a bit overfitting.

How about logistics Regression?



```
#let's evaluate the model on the test set
mean_absolute_error(y_test,y_pred)
```

```
0.36245954692556637
```

```
# let's see our model accuracy
print("Our model accuracy is: {}%".format(round(model.score(X_test,y_test)*100,2)))
```

```
Our model accuracy is: 68.93%
```

As I mentioned above from the Beginning, since the class imbalance is not that big, even though we balanced the classes using SMOTE, the accuracy of the model has decreased which is good but, not that much in difference! But, it’s good to learn the impact of fixing the class imbalance on the accuracy of the model and learning how to deal with it!

---

3. Segment Users by QoE

User segmentation by quality of experience (QoE) patterns is a valuable analysis technique for this dataset.
Here is how we should get started:

1. Feature Selection
   - Identify columns relating to user feedback, perceptions (QoF_*)
   - Add relevant params like video quality, stall events etc.

2. Clustering
   - Apply K-Means clustering to group users based on selected features
   - Determine optimal clusters (K) through elbow plot method

3. Analyze Clusters
   - Profile each cluster segregated by QoE patterns
     e.g. Cluster 1 - High quality, low stalls

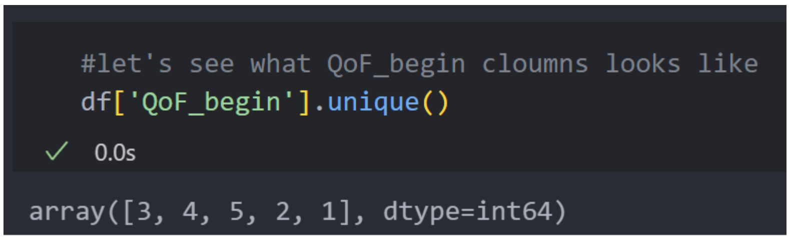
   - Compare cluster-level statistics for distribution of ratings, events faced etc.

4. Validate Clusters
   - Check if cluster assignments correlate to overall MOS groups
   - For example, a "bad QoE" cluster should have lower MOS scores

Key Outcomes:
- User subgroups based on their subjective QoE evaluations

- Tailored enhancements to address issues faced per each profile
- Targeted marketing and promotions to specific user segments
- Better understanding of user behavior and preferences
- Insights into factors impacting QoE for different user groups

1. Feature Selection
   - Identify columns relating to user feedback, perceptions (QoF_*)



```
#let's see what QoF_begin cloumns looks like
df['QoF_begin'].unique()
```

✓ 0.0s

```
array([3, 4, 5, 2, 1], dtype=int64)
```

The `QoF_begin` column in the dataset refers to the user feedback rating on the video playback start time, based on the documentation.

The unique values array `[3, 4, 5, 2, 1]` indicates:

- These are the 5 rating levels provided by the users for the playback start time
- It ranges from 1 (bad) to 5 (excellent)

Specific interpretations:

- Rating 5 - Playback started instantly
- Rating 3 - Some minor delay at start
- Rating 1 - Excessive delay before video played

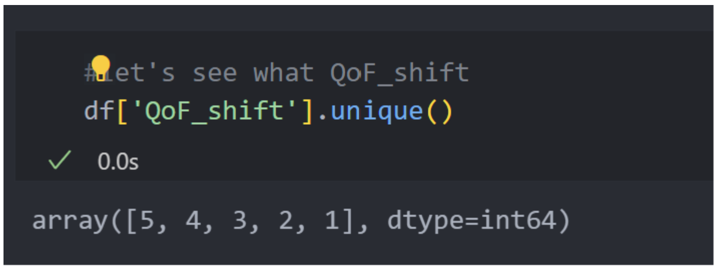
A few reasons for the distribution across ratings that we observe:

- Network conditions impact initial buffering delay
- Device performance affects loading time
- Video complexity influences readiness

Since first impressions are important, analyzing user feedback on startup time and its correlation to overall QoE can provide insights into:

- Optimizing initial buffers for instant playback
- Reducing playback latency on slower devices
- Selecting bitrates based on device and network

How about QoF_shift column?



```
#Let's see what QoF_shift
df['QoF_shift'].unique()
✓ 0.0s
array([5, 4, 3, 2, 1], dtype=int64)
```

The `QoF_shift` column in the dataset refers to the user feedback rating on video quality shifts during the streaming session, as per my understanding.

The unique values array [5, 4, 3, 2, 1] indicates:

- Users had the option to rate quality shifts on a scale of 1 (Bad) to 5 (Excellent)
- It covers the full range of possible ratings

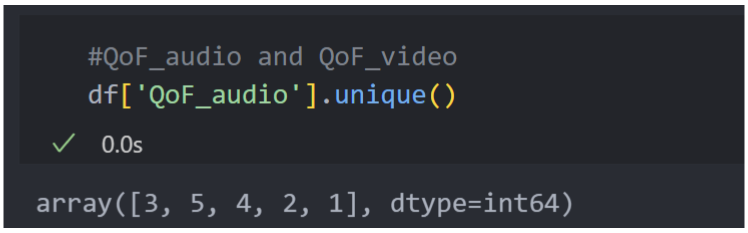
Interpretation of ratings:

- 5 - No perceived quality shifts
- 3 - Some distracting shifts
- 1 - Frequent, jarring quality switches

Some reasons for variability in user ratings:

- Adaptive bitrate switching causes shifts between qualities
- Network throughput changes trigger rate adaptations
- Device display and hardware constrain renditions

The rest two QoF_cloumns are QoF_Audio and Video with unique values like:



```
#QoF_audio and QoF_video
df['QoF_audio'].unique()
✓ 0.0s
array([3, 5, 4, 2, 1], dtype=int64)
```

The `QoF_audio` column captures the user feedback rating for audio quality during the video streaming session.

The unique rating values [3, 5, 4, 2, 1] indicate:

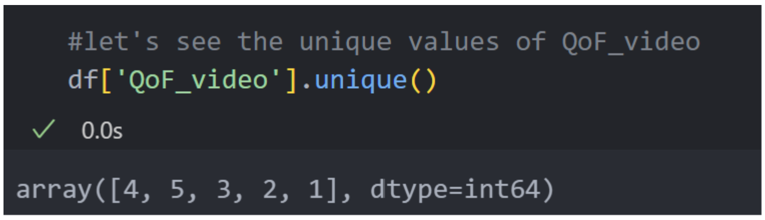
- Users rated audio quality on scale of 1 (Bad) to 5 (Excellent)
- Full range of rating options provided

Interpretation:

- 5 - Excellent audio quality
- 3 - Acceptable audio but some issues
- 1 - Inaudible or largely corrupted audio

Potential factors affecting audio ratings:

- Encoding bitrates and codecs used
- Packet loss causing audio distortion
- Hardware/software audio rendering
- Environment/context noise levels



```
#let's see the unique values of QoF_video
df['QoF_video'].unique()
```

```
✓ 0.0s
array([4, 5, 3, 2, 1], dtype=int64)
```

The `QoF_video` column in the dataset represents the user feedback rating specifically for the video quality aspects during the streaming session.

The unique rating values:
[4, 5, 3, 2, 1]

This indicates:

- Users rated video quality on a scale of 1 (Bad) to 5 (Excellent)
- Full spectrum of possible ratings provided

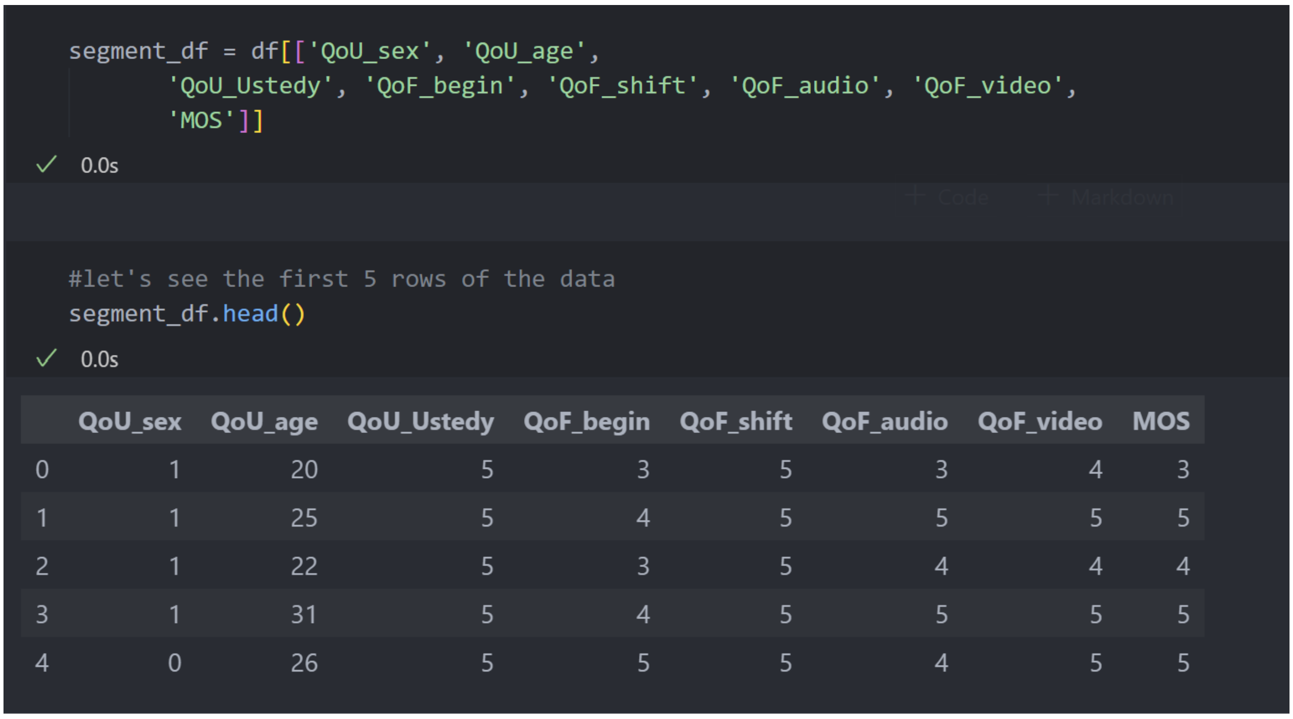
Interpretation:

- 5 - Excellent video quality
- 3 - Acceptable but noticeable issues
- 1 - Extremely poor, unusable video

Potential factors affecting variations in video quality ratings:

- Video resolution and encoding bitrates
- Frame drops due to network issues
- Device display characteristics and hardware constraints
- Playback stalls and adaptiveness

Since we have decided to only use QoU and QoF cloumns in relation to MOS , let's create our segment data frame!

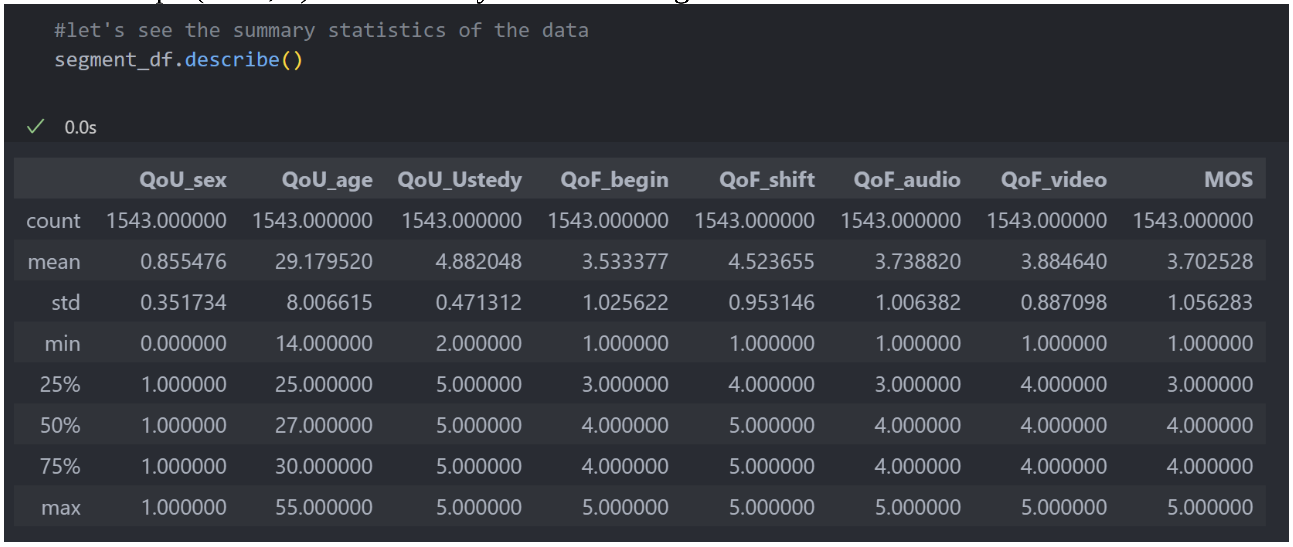


```
segment_df = df[['QoU_sex', 'QoU_age',
        'QoU_Ustedy', 'QoF_begin', 'QoF_shift', 'QoF_audio', 'QoF_video',
        'MOS']]
```
✓ 0.0s

```
#let's see the first 5 rows of the data
segment_df.head()
```
✓ 0.0s

	QoU_sex	QoU_age	QoU_Ustedy	QoF_begin	QoF_shift	QoF_audio	QoF_video	MOS
0	1	20	5	3	5	3	4	3
1	1	25	5	4	5	5	5	5
2	1	22	5	3	5	4	4	4
3	1	31	5	4	5	5	5	5
4	0	26	5	5	5	4	5	5

With its shape (1543, 8) and summary statistics being :

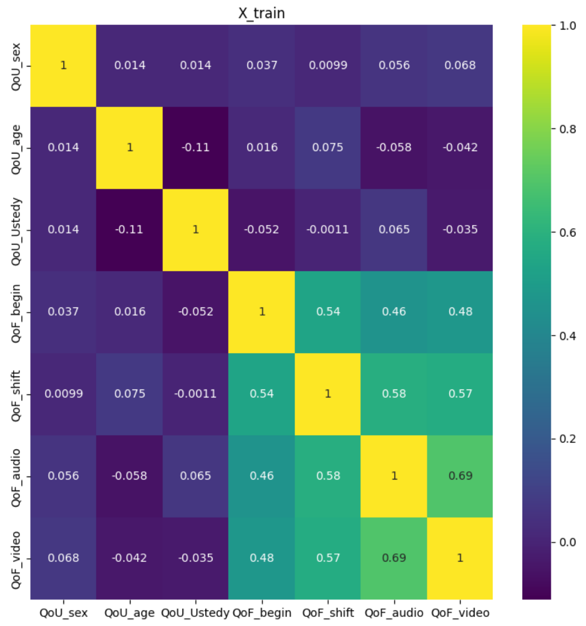


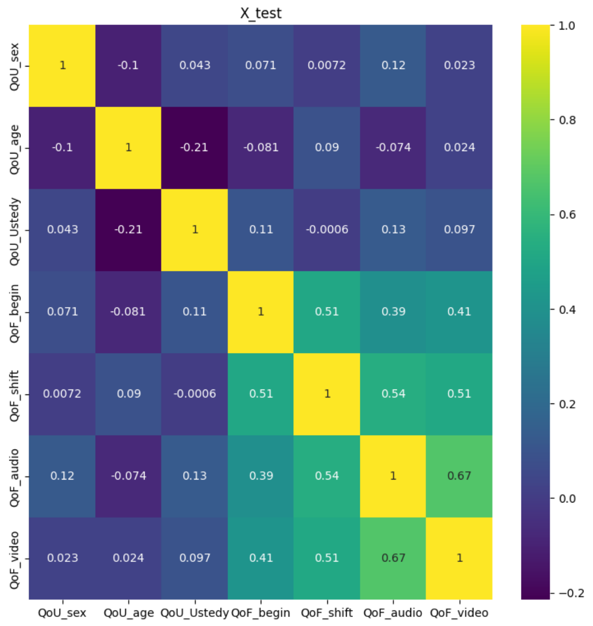
```
#let's see the summary statistics of the data
segment_df.describe()
```
✓ 0.0s

	QoU_sex	QoU_age	QoU_Ustedy	QoF_begin	QoF_shift	QoF_audio	QoF_video	MOS
count	1543.000000	1543.000000	1543.000000	1543.000000	1543.000000	1543.000000	1543.000000	1543.000000
mean	0.855476	29.179520	4.882048	3.533377	4.523655	3.738820	3.884640	3.702528
std	0.351734	8.006615	0.471312	1.025622	0.953146	1.006382	0.887098	1.056283
min	0.000000	14.000000	2.000000	1.000000	1.000000	1.000000	1.000000	1.000000
25%	1.000000	25.000000	5.000000	3.000000	4.000000	3.000000	4.000000	3.000000
50%	1.000000	27.000000	5.000000	4.000000	5.000000	4.000000	4.000000	4.000000
75%	1.000000	30.000000	5.000000	4.000000	5.000000	4.000000	4.000000	4.000000
max	1.000000	55.000000	5.000000	5.000000	5.000000	5.000000	5.000000	5.000000

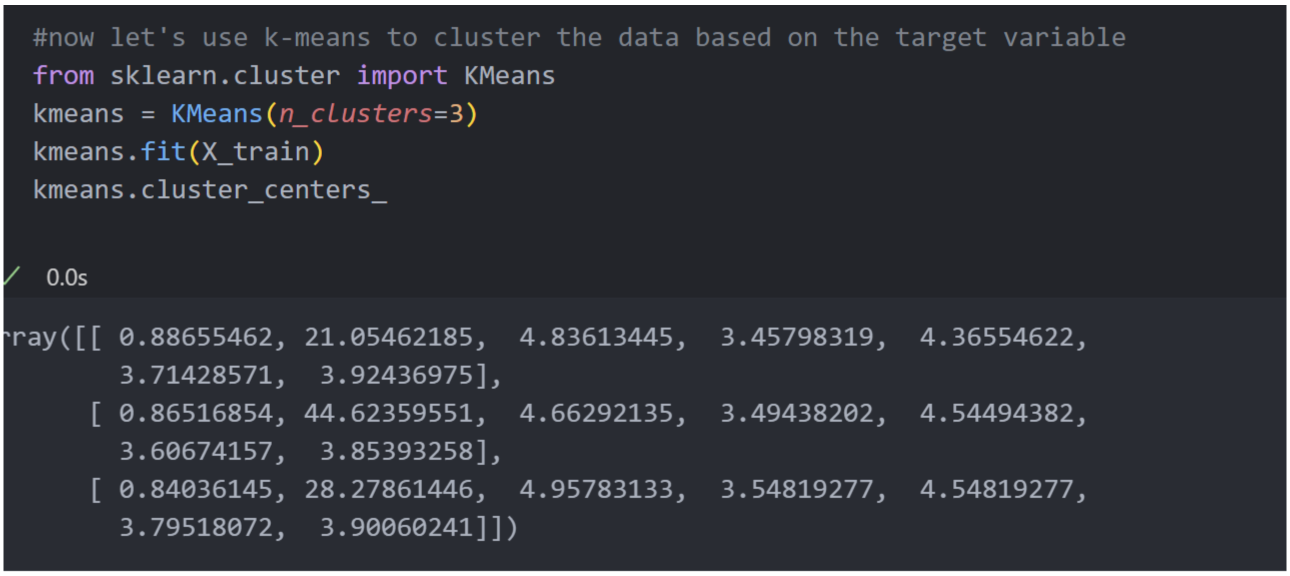
Our segment dataset is looking good and ready to model.

After train,test,split ,we can see the correlation matrix (heatmaps) of train and test sets corresponding to their target variable MOS ! In Train set there are some low correlations but in test set, everything seems positive which is good to me! It won’t overfit!





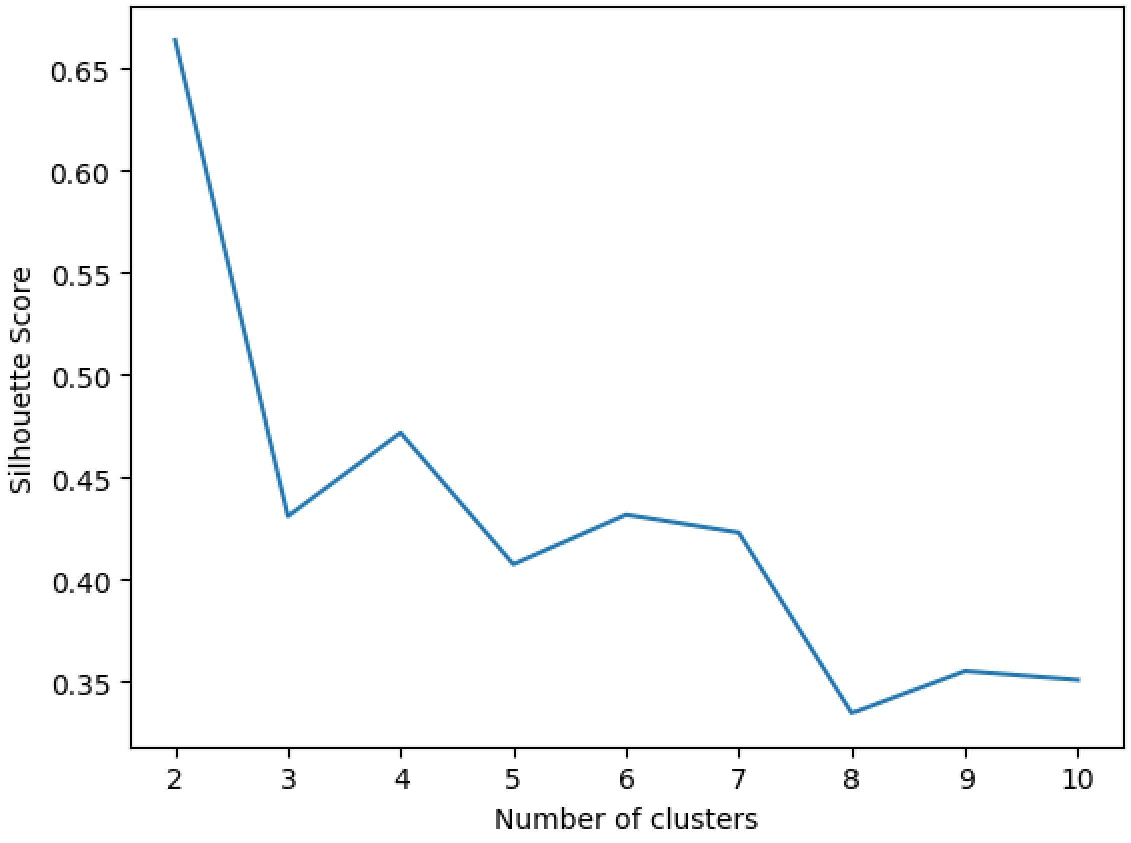
Let's start our clustering with k=3 and then we will use elbow-method and figure out their silhouette scores!



```
#now let's use k-means to cluster the data based on the target variable
from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters=3)
kmeans.fit(X_train)
kmeans.cluster_centers_
```

✓ 0.0s

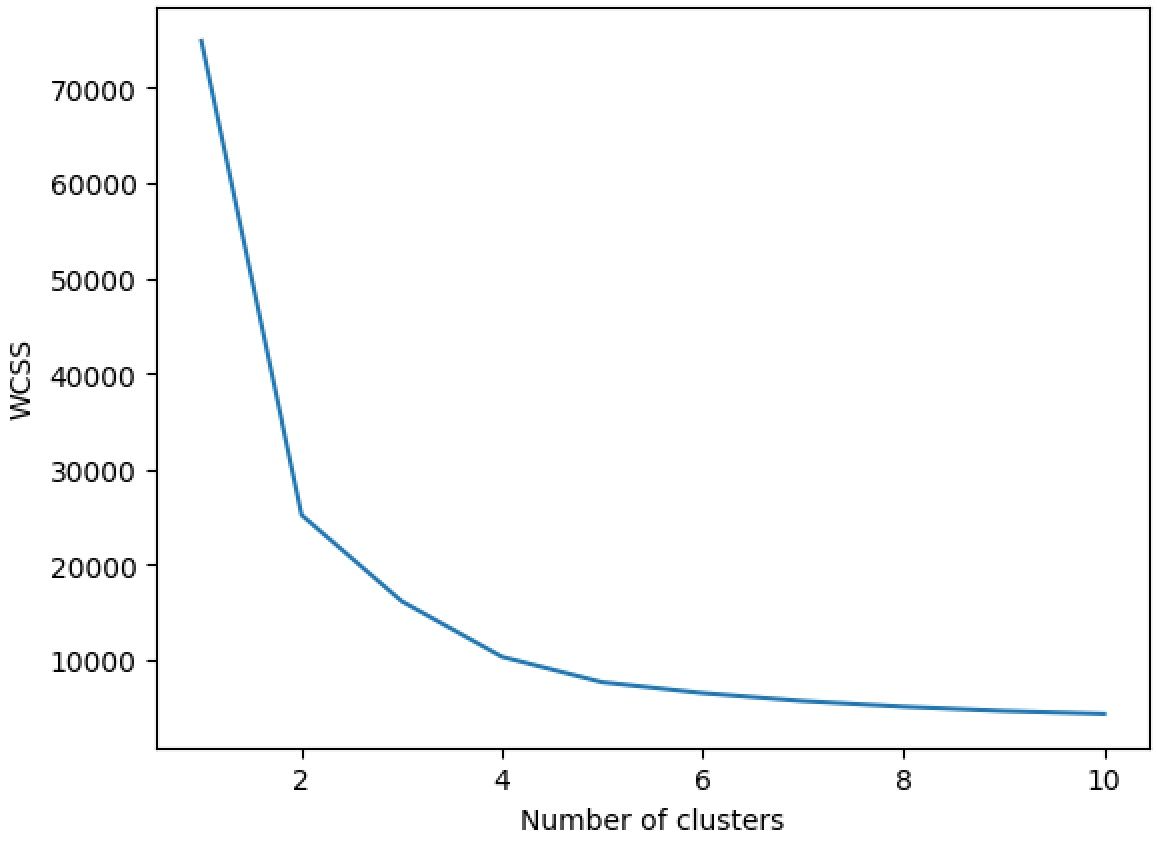
```
ray([[ 0.88655462, 21.05462185,  4.83613445,  3.45798319,  4.36554622,
        3.71428571,  3.92436975],
      [ 0.86516854, 44.62359551,  4.66292135,  3.49438202,  4.54494382,
        3.60674157,  3.85393258],
      [ 0.84036145, 28.27861446,  4.95783133,  3.54819277,  4.54819277,
        3.79518072,  3.90060241]])
```



With this figure corresponding to their scores being :

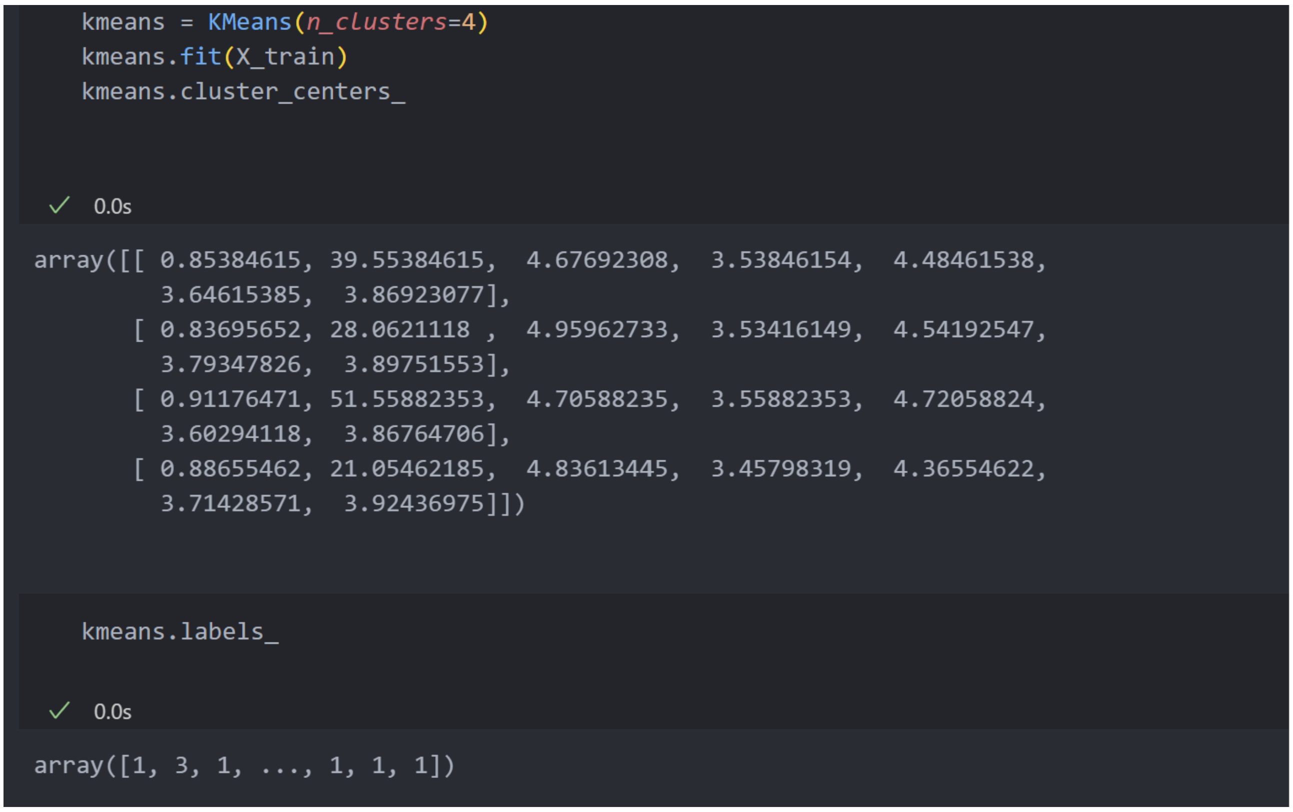
```
Silhouette score for 2 clusters is 0.6642583167790189
Silhouette score for 3 clusters is 0.43110792810785215
Silhouette score for 4 clusters is 0.47203228064797614
Silhouette score for 5 clusters is 0.4075243708801069
Silhouette score for 6 clusters is 0.43183641918172877
Silhouette score for 7 clusters is 0.4230519377705377
Silhouette score for 8 clusters is 0.3346462672557333
Silhouette score for 9 clusters is 0.3551563560684741
Silhouette score for 10 clusters is 0.35082594938983064
```

Since the silhouette score of clusters 4 is the highest before it starts to drop, we can say that the optimal number of clusters is 4.



Due to these figures and scores, I have decided that our optimal number of clusters with respect to the silhouette score and elbow method is 4.

So, let’s fit our model !



```
kmeans = KMeans(n_clusters=4)
kmeans.fit(X_train)
kmeans.cluster_centers_
```

✓ 0.0s

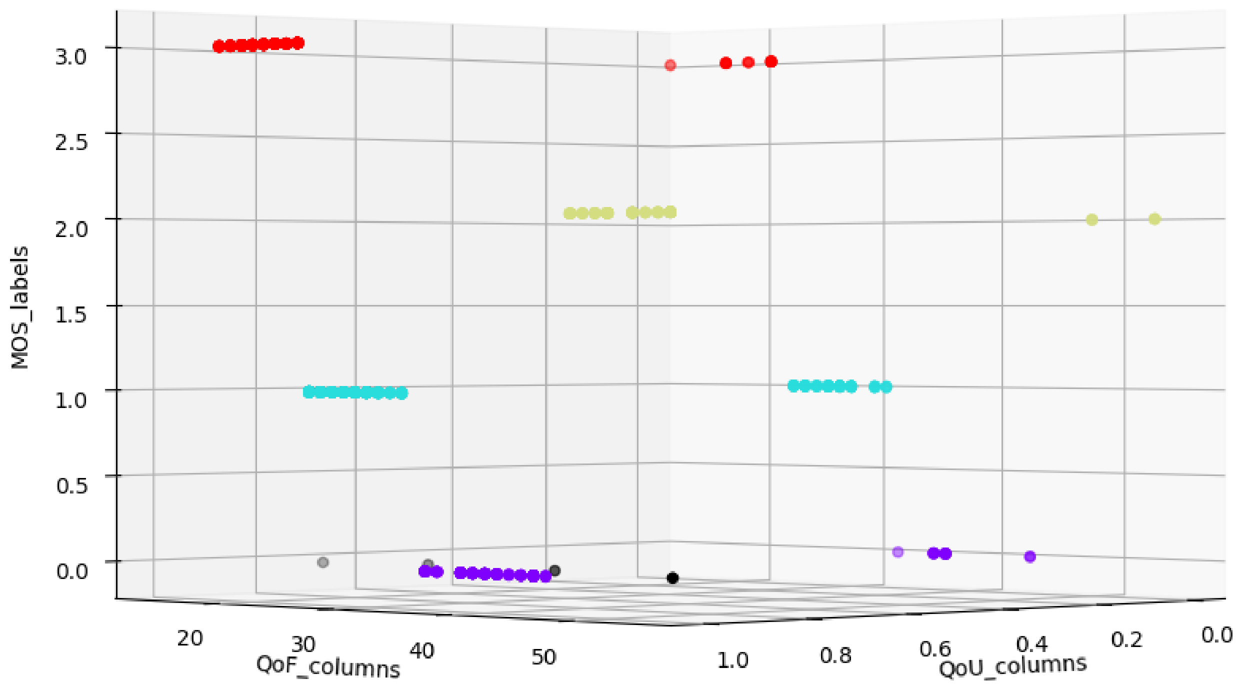
```
array([[ 0.85384615, 39.55384615,  4.67692308,  3.53846154,  4.48461538,
         3.64615385,  3.86923077],
       [ 0.83695652, 28.0621118 ,  4.95962733,  3.53416149,  4.54192547,
         3.79347826,  3.89751553],
       [ 0.91176471, 51.55882353,  4.70588235,  3.55882353,  4.72058824,
         3.60294118,  3.86764706],
       [ 0.88655462, 21.05462185,  4.83613445,  3.45798319,  4.36554622,
         3.71428571,  3.92436975]])
```

```
kmeans.labels_
```

✓ 0.0s

```
array([1, 3, 1, ..., 1, 1, 1])
```

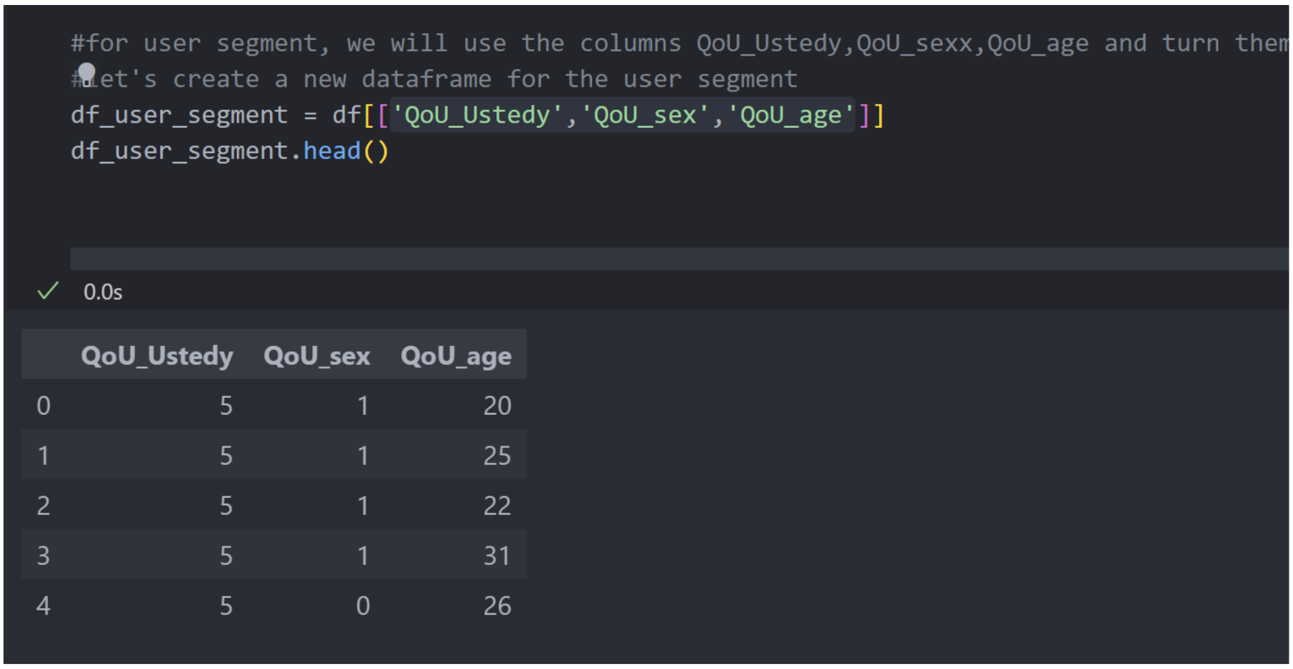
With the final output looks something like this : (K=4)



That’s not very conclusive or moving result ! But, it’s a starting point!

How about we try only the user segmentations ? (Later we could combine it with QOS again)

User_Segmentations_Only_Kmeans



```
#for user segment, we will use the columns QoU_Ustedy,QoU_sexx,QoU_age and turn them
#Let's create a new dataframe for the user segment
df_user_segment = df[['QoU_Ustedy','QoU_sex','QoU_age']]
df_user_segment.head()
```

✓ 0.0s

	QoU_Ustedy	QoU_sex	QoU_age
0	5	1	20
1	5	1	25
2	5	1	22
3	5	1	31
4	5	0	26

These three columns make up the users and we have to come up with users groups labels for us to later apply on MOS and other related tasks!

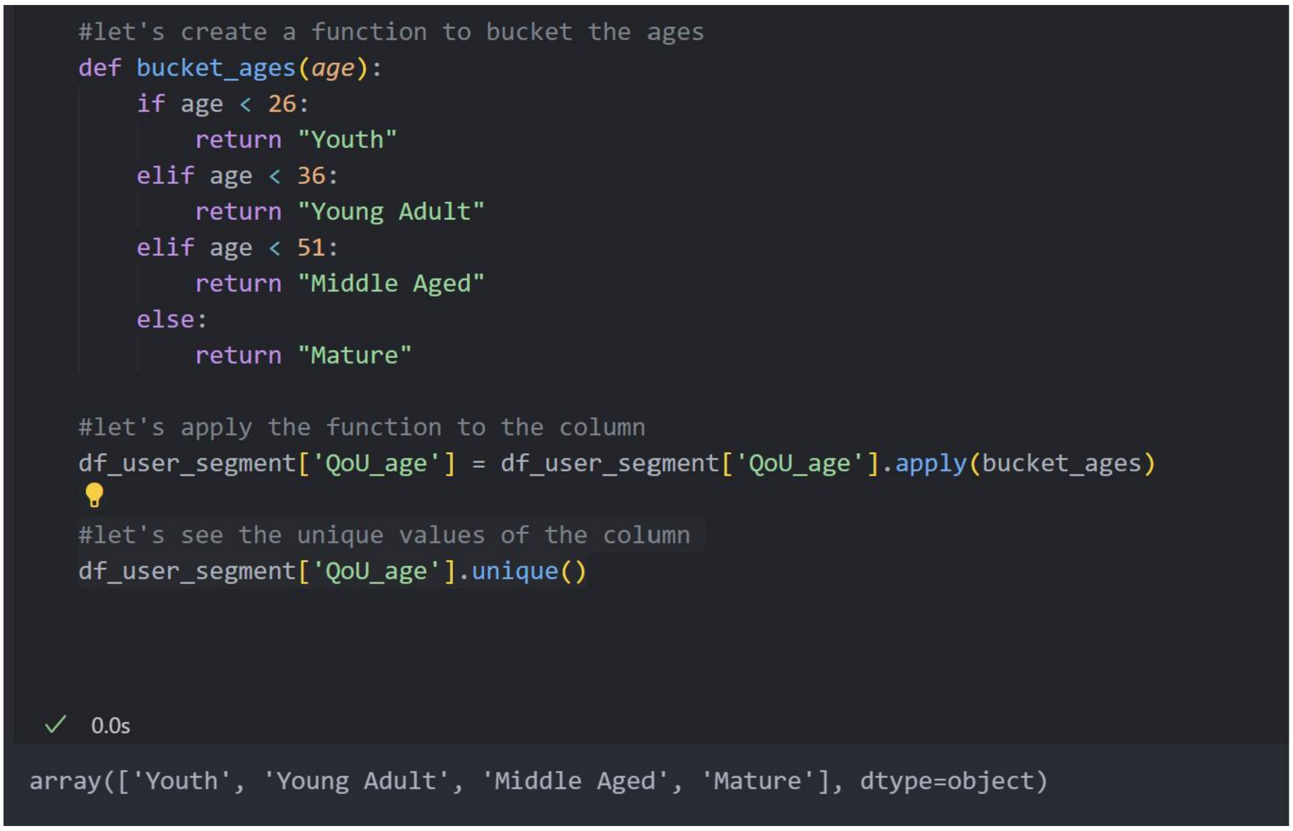
Since there are 3 separate columns related to users - education, gender and age.

To perform user segmentation through clustering, we need to consolidate them into representative user groups.

Here is one approach to achieve that:

1. Encode Categorical Columns -which we have already done !
2. Encode gender and education level into numeric formats
which we have already done !

Derive User Type
Bucket age groups (teenagers, youth, middle-aged etc)
Combine above 3 columns into a categorical user type column e.g. "Female College-aged User"
This is necessary to cluster users based on their profiles.

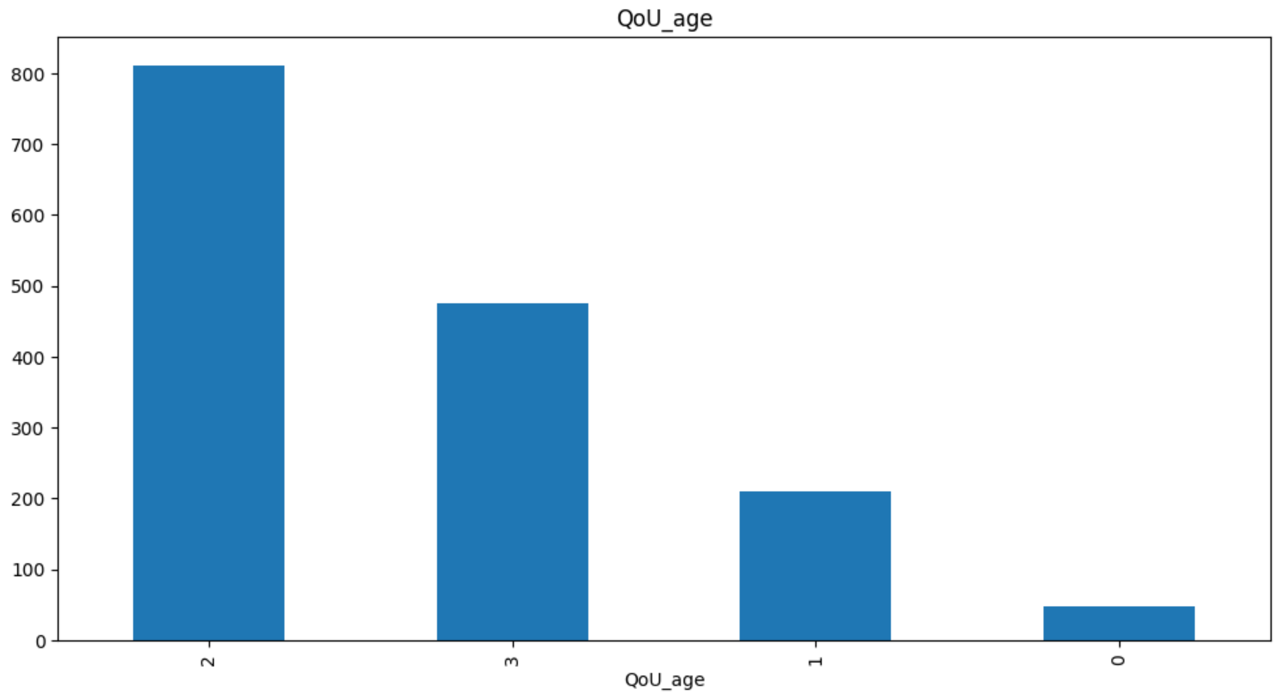


```
#let's create a function to bucket the ages
def bucket_ages(age):
    if age < 26:
        return "Youth"
    elif age < 36:
        return "Young Adult"
    elif age < 51:
        return "Middle Aged"
    else:
        return "Mature"

#let's apply the function to the column
df_user_segment['QoU_age'] = df_user_segment['QoU_age'].apply(bucket_ages)

#let's see the unique values of the column
df_user_segment['QoU_age'].unique()
```

```
✓ 0.0s
```

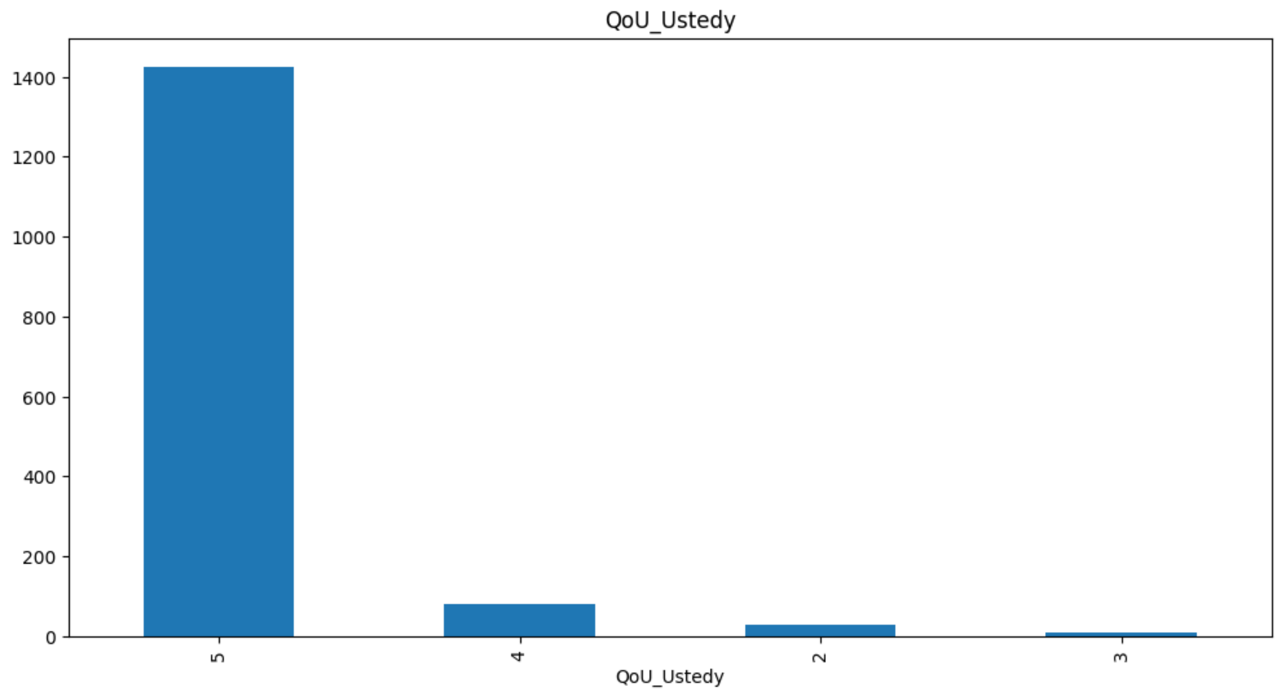
```
array(['Youth', 'Young Adult', 'Middle Aged', 'Mature'], dtype=object)
```

After following the usual and much more complicated Data-Preprocessing/feature engineering we have :

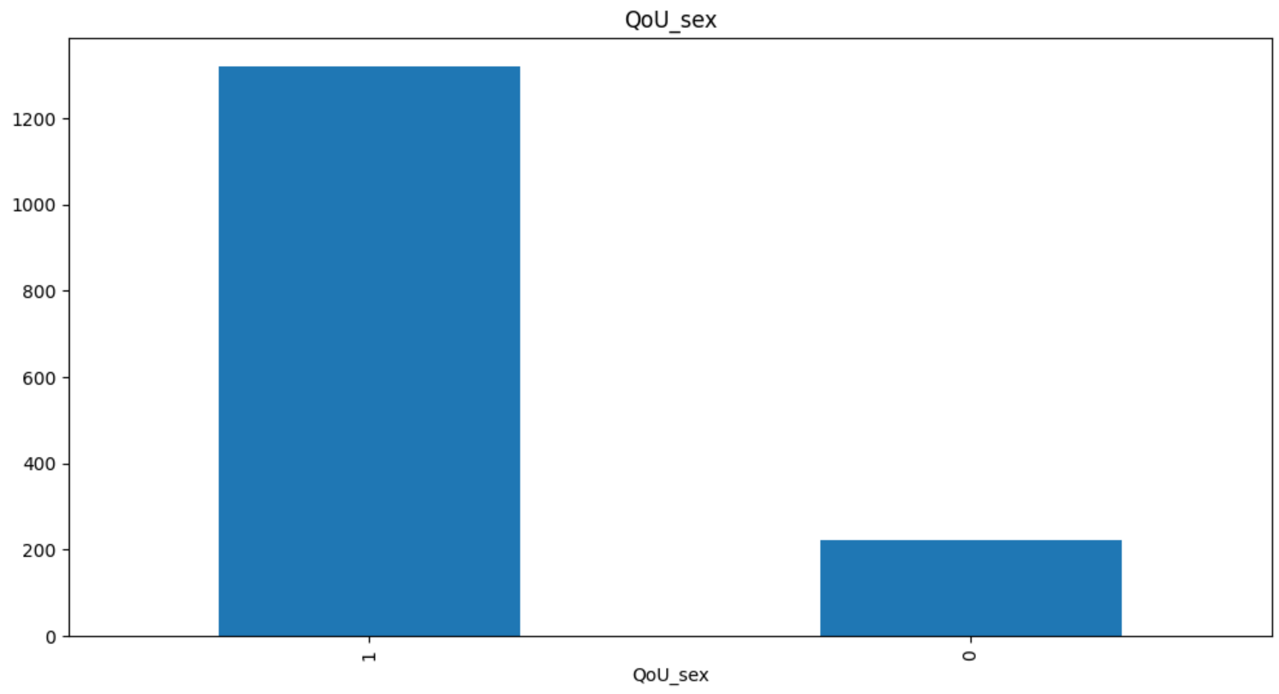


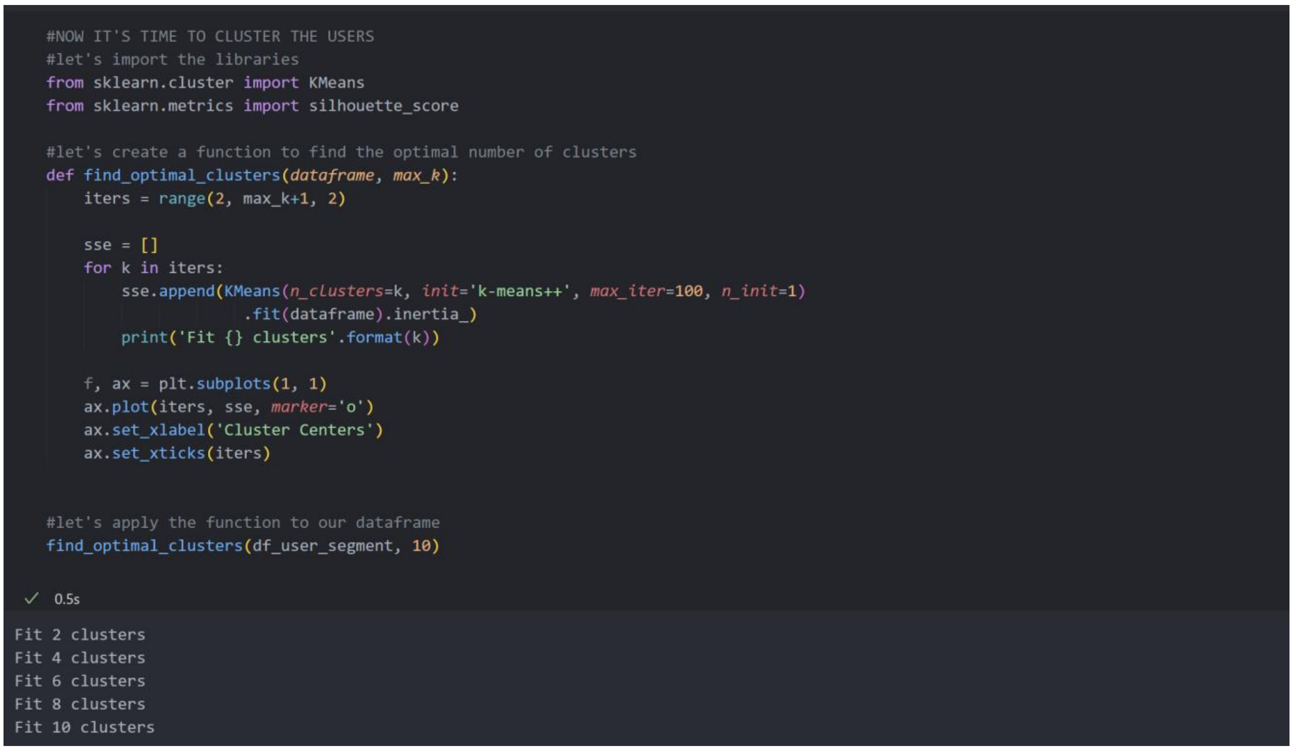
middle-age, mature, young adult, youth accordingly!

The same goes for Education of users!



Now that we have 4x4 features , we will drop "man" dominant column QoU_sex feature for simplification !



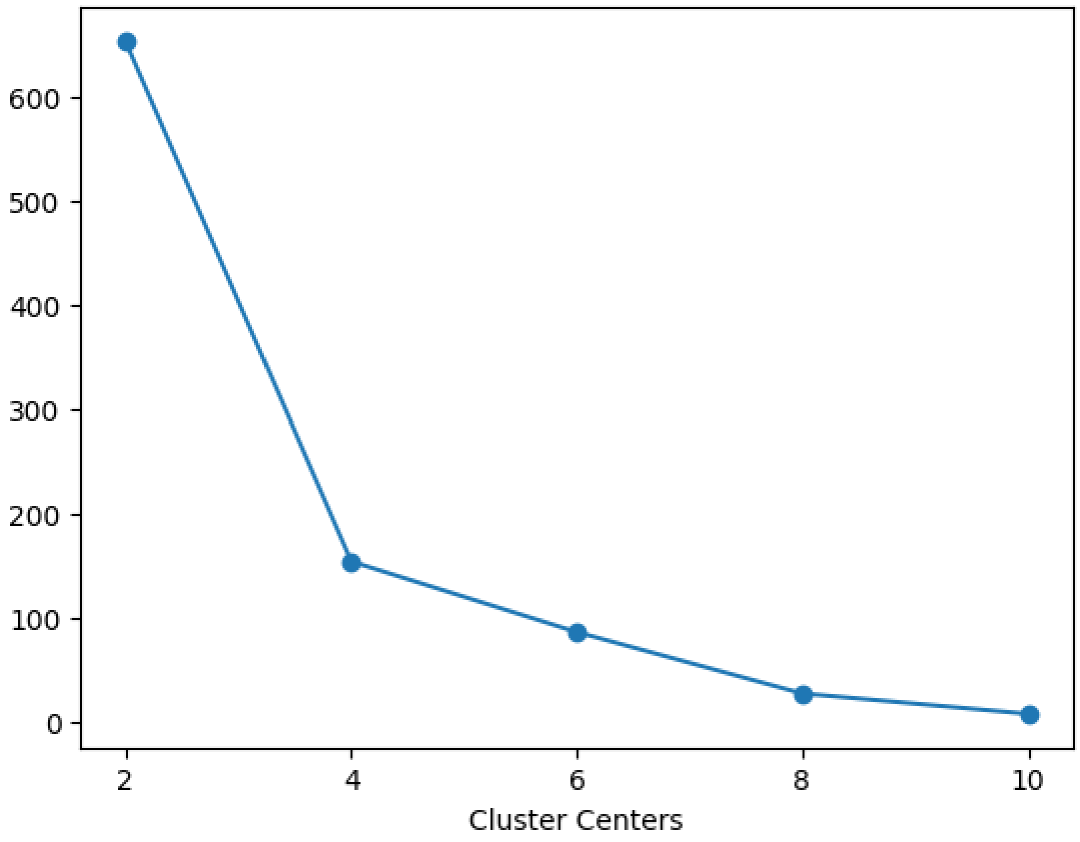


```
#NOW IT'S TIME TO CLUSTER THE USERS
#let's import the libraries
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score

#let's create a function to find the optimal number of clusters
def find_optimal_clusters(dataframe, max_k):
    iters = range(2, max_k+1, 2)

    sse = []
    for k in iters:
        sse.append(KMeans(n_clusters=k, init='k-means++', max_iter=100, n_init=1)
                    .fit(dataframe).inertia_)
        print('Fit {} clusters'.format(k))

    f, ax = plt.subplots(1, 1)
    ax.plot(iters, sse, marker='o')
    ax.set_xlabel('Cluster Centers')
    ax.set_xticks(iters)


#let's apply the function to our dataframe
find_optimal_clusters(df_user_segment, 10)
```

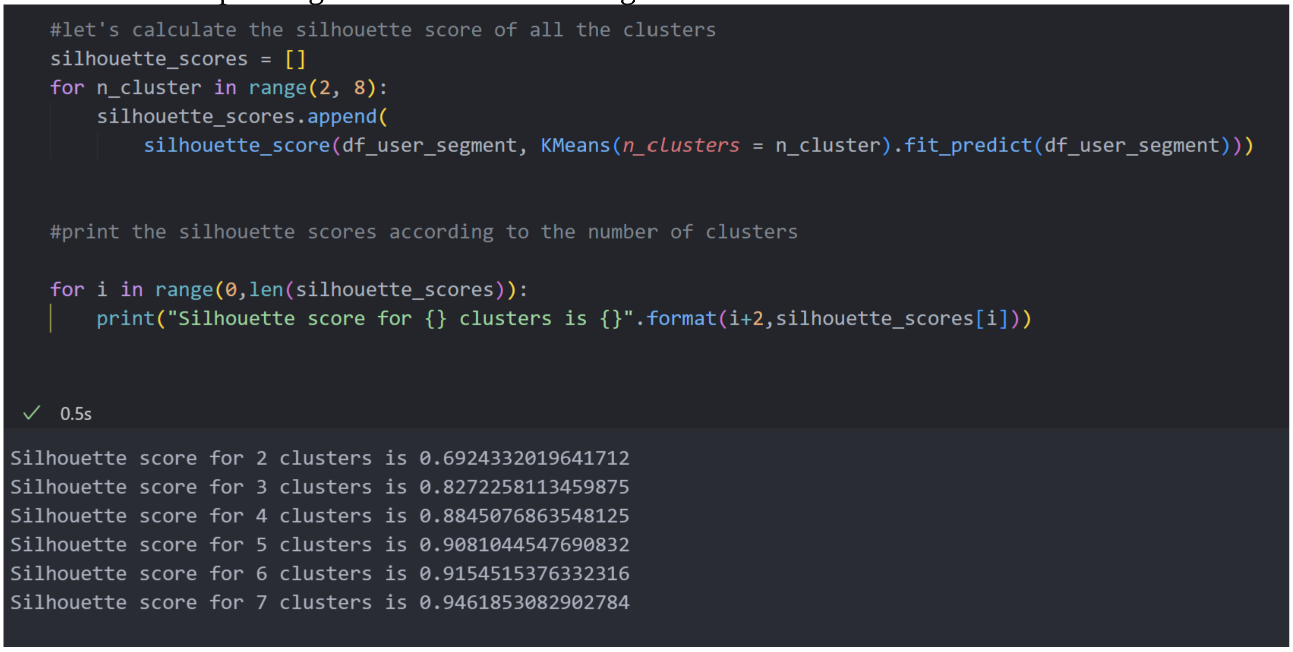
✓ 0.5s

```
Fit 2 clusters
Fit 4 clusters
Fit 6 clusters
Fit 8 clusters
Fit 10 clusters
```

By using Elbow method, we can cluster our user-segments and find the optimal cluster centres out of all the possible clusters!



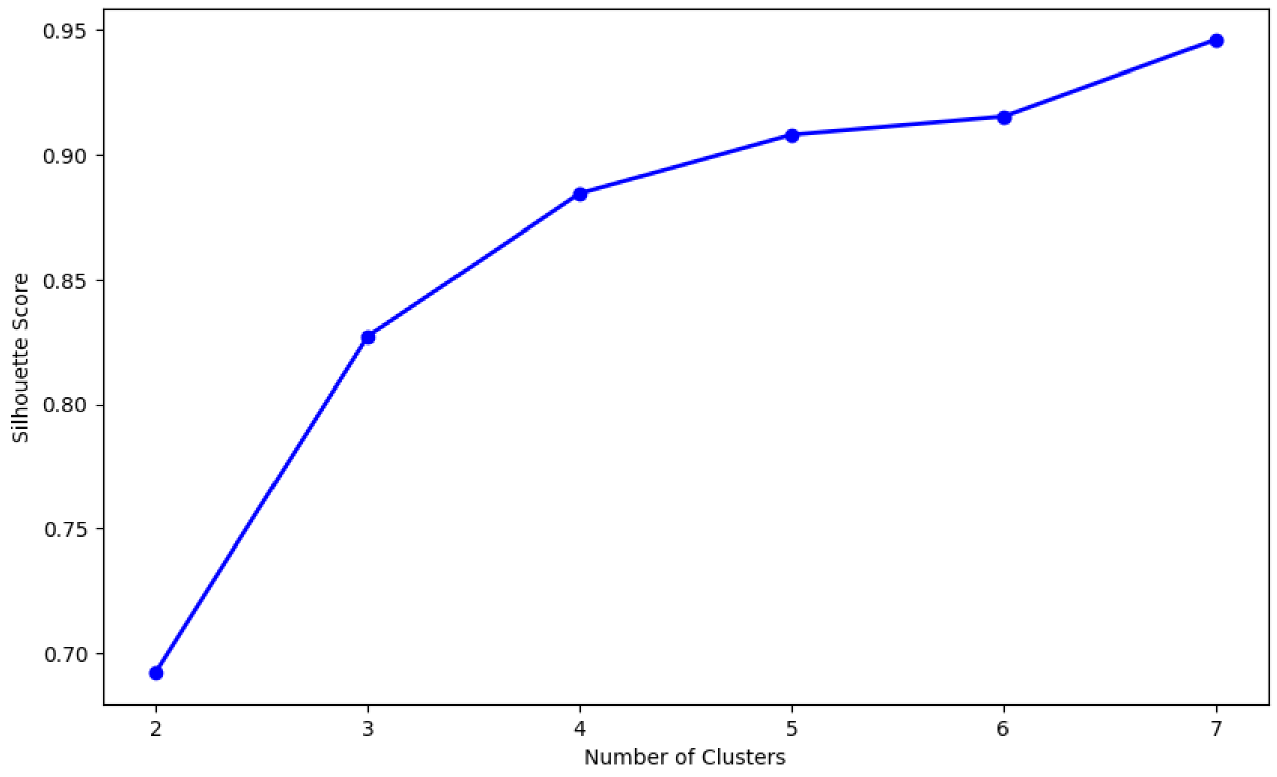
With their corresponding silhouettes scores being:



```
#let's calculate the silhouette score of all the clusters
silhouette_scores = []
for n_cluster in range(2, 8):
    silhouette_scores.append(
        silhouette_score(df_user_segment, KMeans(n_clusters = n_cluster).fit_predict(df_user_segment)))


#print the silhouette scores according to the number of clusters

for i in range(0,len(silhouette_scores)):
    print("Silhouette score for {} clusters is {}".format(i+2,silhouette_scores[i]))
```

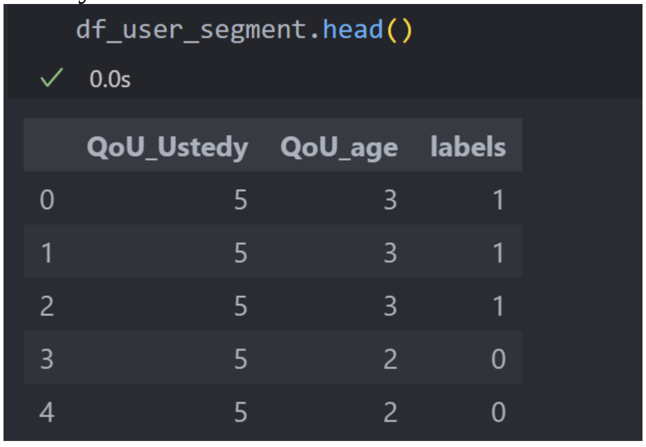
✓ 0.5s

```
Silhouette score for 2 clusters is 0.6924332019641712
Silhouette score for 3 clusters is 0.8272258113459875
Silhouette score for 4 clusters is 0.8845076863548125
Silhouette score for 5 clusters is 0.9081044547690832
Silhouette score for 6 clusters is 0.9154515376332316
Silhouette score for 7 clusters is 0.9461853082902784
```



However, we will choose 4 clusters since it is the elbow point and also the silhouette score is the highest, right in the middle for 4 clusters!

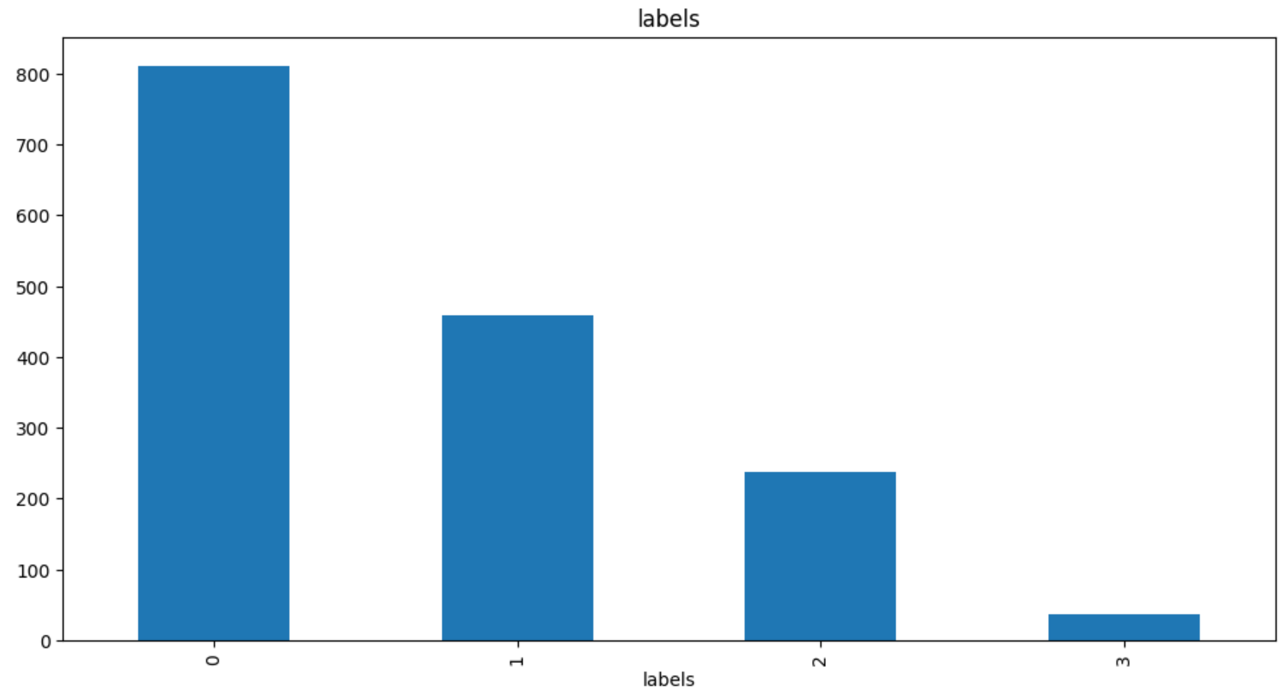
Finally we have our labels!



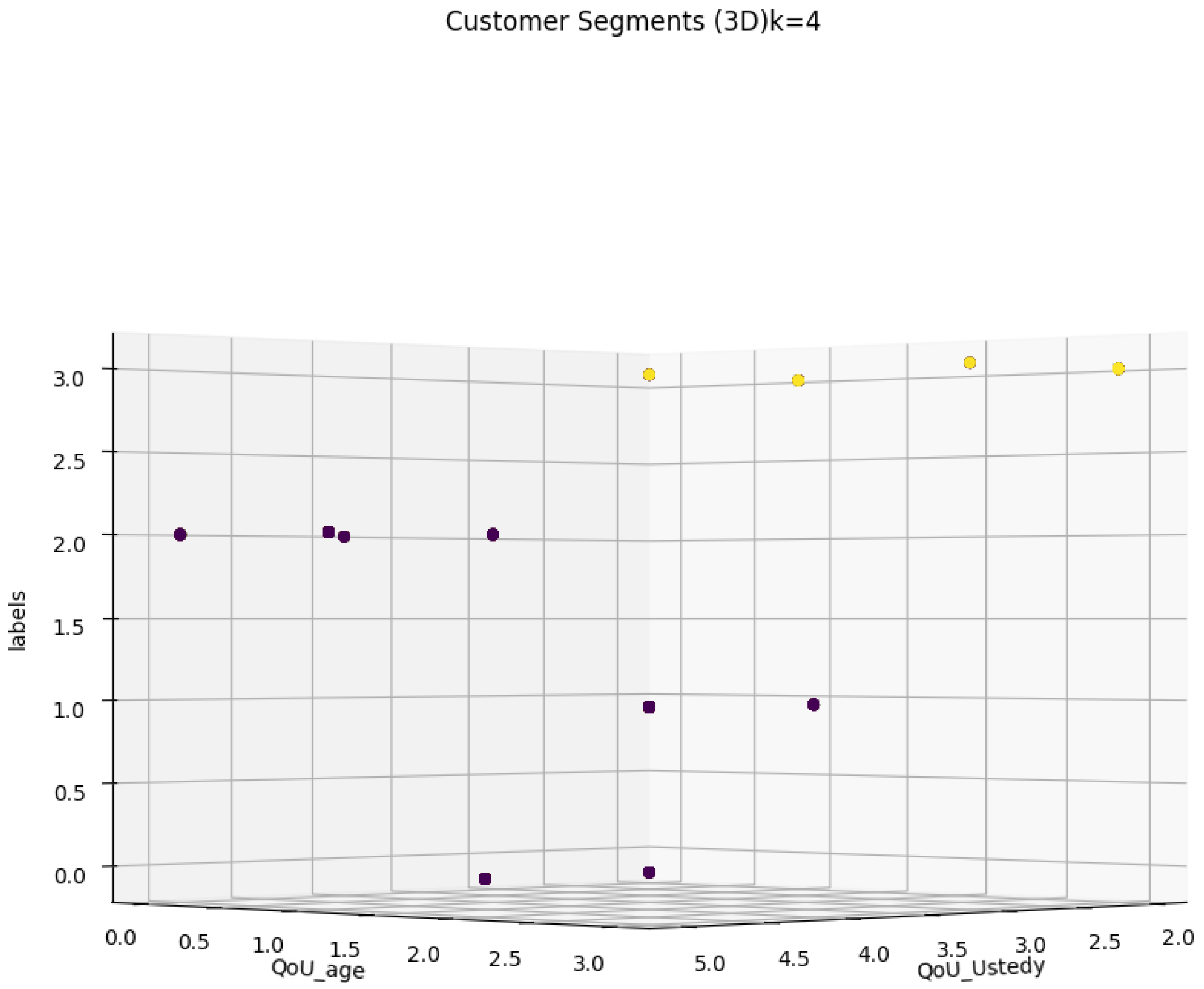
```
df_user_segment.head()
✓ 0.0s
```

	QoU_Ustedy	QoU_age	labels
0	5	3	1
1	5	3	1
2	5	3	1
3	5	2	0
4	5	2	0

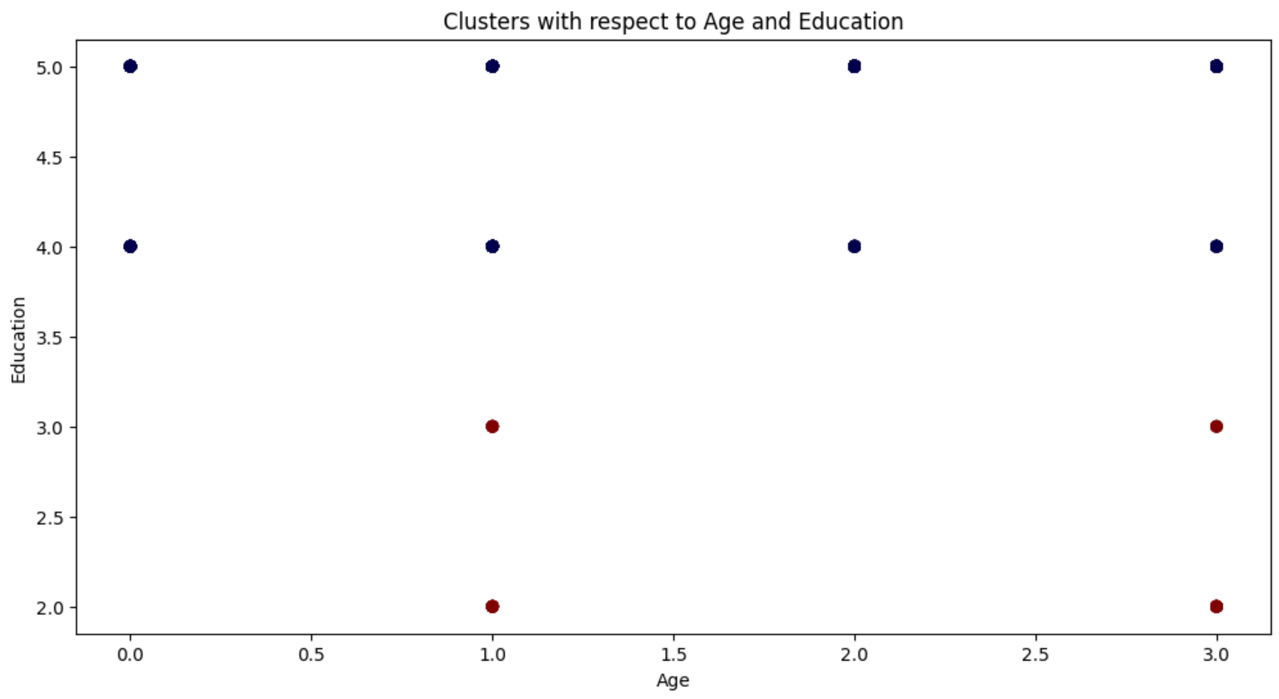
With its distribution being:



To visualize the whole user-segmentation further we could use 3d projection like:



Or in 2-D space, we can have something like :



This is not that bad.

Future Work

"In addition to categorizing users by attributes like education level, gender and age, an impactful future work would be to perform user segmentation specifically based on patterns in quality of experience (QoE) like we have done in the first place (combination of two would prove better visualization). This could be achieved with more time .

All in all, these QoE-based user persona can help diagnose issues faced by certain segments more severely. They also open possibilities for targeted tuning of delivery platforms and content encoding tailored to known reasonable expectations of each profile. Such personalized enhancements can maximize overall user base satisfaction and quality of experience."

Link to my code repository:

soneeee22000/Quality-of-Experience-QoE-for-mobile-networks-services-DANI-Project: The dataset Poqemon_QoE_Dataset has been used in research papers to predict QoE from network QoS metrics, and to study interactions between the many influence factors. I will also work on the same dataset to apply the right machine learning algorithm to predict Video_Streaming and so many other tasks... (github.com)

Submitted by: Pyae Sone Kyaw (DANI)