Télécom SudParis

Masters in Data Science and Network Intelligence

Data Science – Theory to practice

Project Report:

Improving Mean Opinion Score (MOS) Prediction in Mobile Networks: A Segmentation-Based Machine Learning Approach

Submitted to Professor

Alessandro Maddaloni

By

ALI ALLOUF

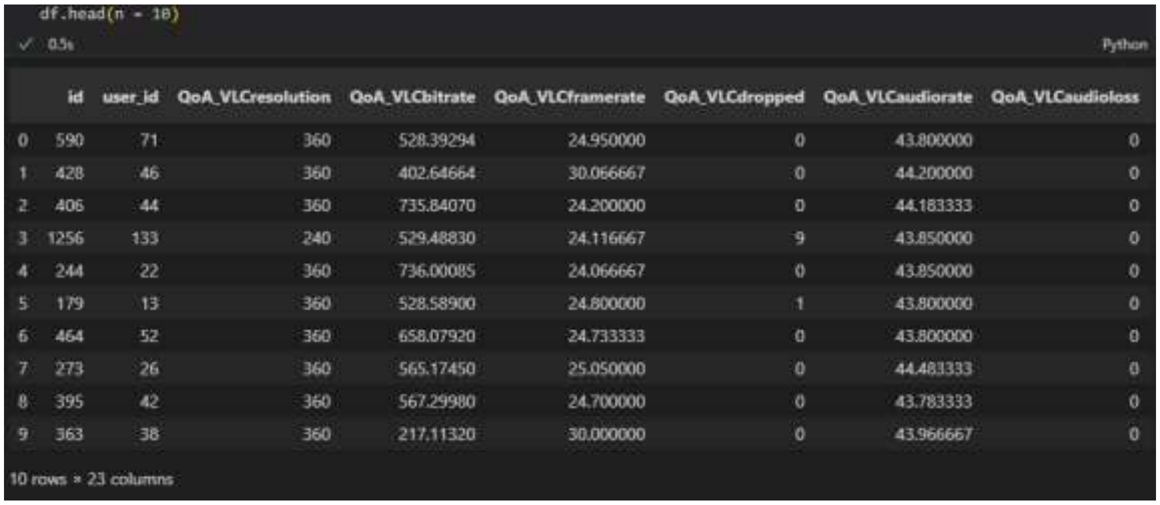
Academic Year 2024-2025

1. Understanding the dataset

I. What is the dataset about:

The Poqemon-QoE-Dataset is a dataset for **<u>Quality of Experience modeling</u>** for video streaming in mobile environments (UMTS, HSPA, LTE …). It contains 1560 samples and covers 23 features (22 QoE Influence Factors and the Mean Opinion Score).

The subjective evaluation campaign utilized a pool of 181 testers to assess the video quality. To capture QoE from a true end-user perspective, the participants possessed no prior experience with video quality assessment experimentation. This ensured that the collected **Mean Opinion Scores (MOS)** accurately reflect the satisfaction of the users.



```
df.head(n = 10)
✓ 0.5s                                                                 Python
```

	id	user_id	QoA_VLCresolution	QoA_VLCbitrate	QoA_VLCframerate	QoA_VLCdropped	QoA_VLCaudiorate	QoA_VLCaudioloss
0	590	71	360	528.39294	24.950000	0	43.800000	0
1	428	46	360	402.64664	30.066667	0	44.200000	0
2	406	44	360	735.84070	24.200000	0	44.183333	0
3	1256	133	240	529.48830	24.116667	9	43.850000	0
4	244	22	360	736.00085	24.066667	0	43.850000	0
5	179	13	360	528.58900	24.800000	1	43.800000	0
6	464	52	360	658.07920	24.733333	0	43.800000	0
7	273	26	360	565.17450	25.050000	0	44.483333	0
8	395	42	360	567.29980	24.700000	0	43.783333	0
9	363	38	360	217.11320	30.000000	0	43.966667	0

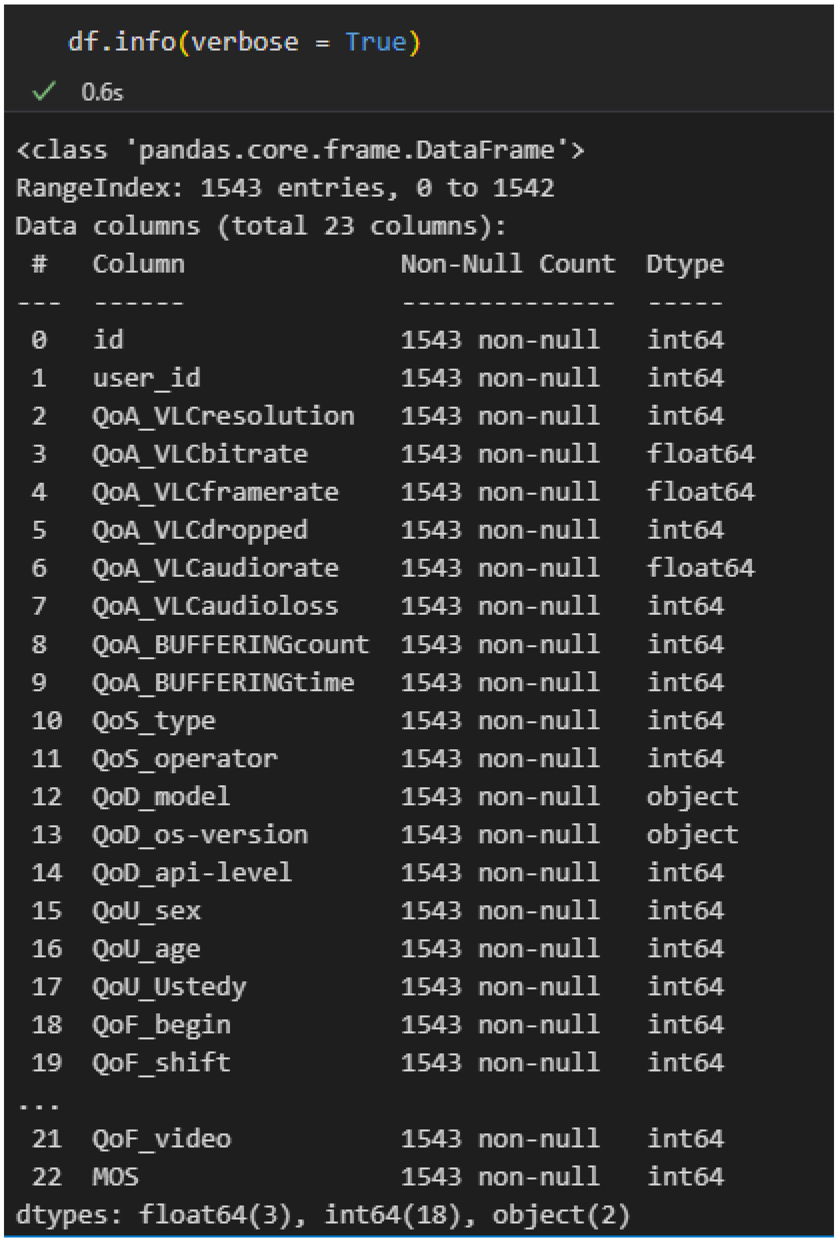
10 rows × 23 columns

II. How has it been used:

The dataset primarily used by Network operators (Orange, SFR, Bouyegues and Free) and in research field by researches for developing QoE prediction models in mobile networks.

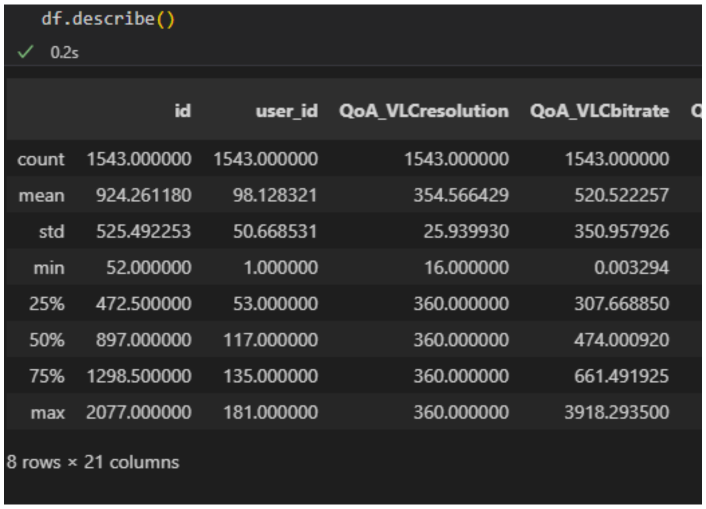
- **Orange Network Study:** The whole project was funded by Orange and a portion of the results used by them to identify the Quality of Service (QoS) indicators required to accurately predict QoE for HTTP YouTube content.

- **The 2015 publication** ("Building a Large Dataset for Model-based QoE Prediction in the Mobile Environment") serves as the foundational paper for the datset. Demonstrating that user experience is a complex function of multiple, interacting technical and contextual variables, not just a single QoS metric.
- The dataset serves as the training and testing ground for algorithms designed to predict the **Mean Opinion Score (MOS)** using the 22 Quality of Experience Influence Factors (QoE IFs).



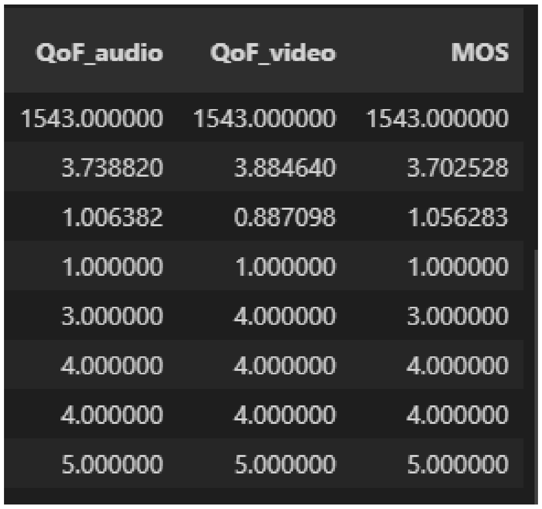
```
df.info(verbose = True)
✓ 0.6s

<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
dtypes: float64(3), int64(18), object(2)
```

2. Data Visualization for Insights



```
df.describe()
✓ 0.2s
```

```
                id      user_id  QoA_VLCresolution  QoA_VLCbitrate  Q
count  1543.000000  1543.000000        1543.000000     1543.000000
mean    924.261180    98.128321         354.566429      520.522257
std     525.492253    50.668531          25.939930      350.957926
min      52.000000     1.000000          16.000000        0.003294
25%     472.500000    53.000000         360.000000      307.668850
50%     897.000000   117.000000         360.000000      474.000920
75%    1298.500000   135.000000         360.000000      661.491925
max    2077.000000   181.000000         360.000000     3918.293500

8 rows × 21 columns
```

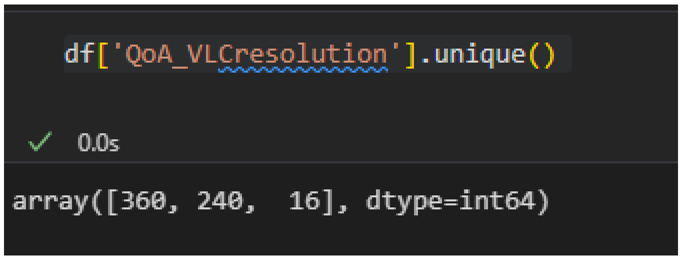


```
   QoF_audio    QoF_video          MOS
1543.000000  1543.000000  1543.000000
   3.738820     3.884640     3.702528
   1.006382     0.887098     1.056283
   1.000000     1.000000     1.000000
   3.000000     4.000000     3.000000
   4.000000     4.000000     4.000000
   4.000000     4.000000     4.000000
   5.000000     5.000000     5.000000
```

Based on descriptive statistics, the crowdsourced video quality assessment data shows a generally positive user experience (median MOS of 4.0).

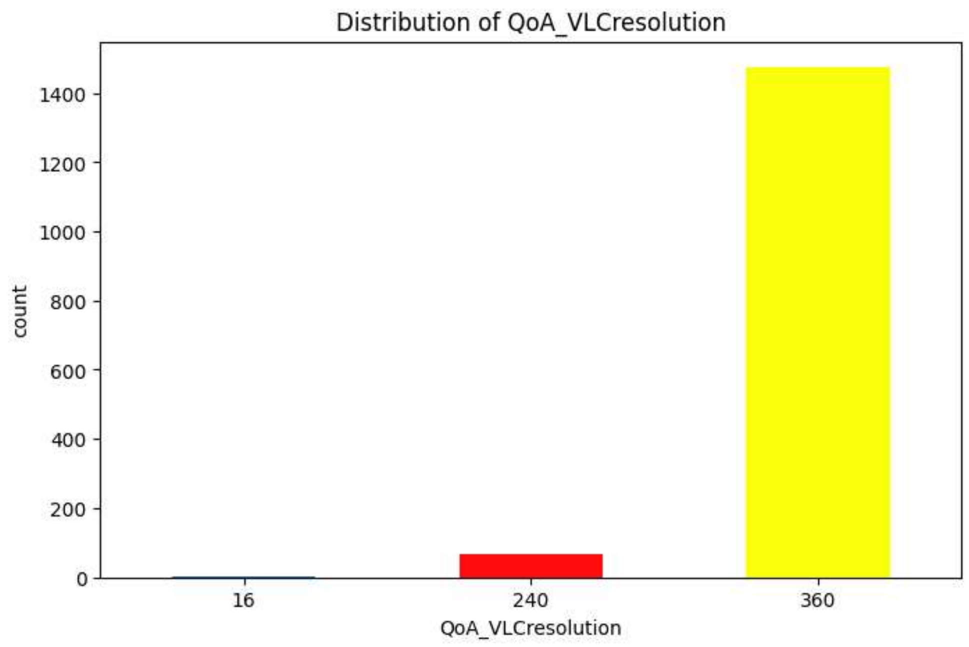
Now, Baes on the README file and observing the csv file, I will select several columns for understanding the user experience: QoA_VLCresolution, QoA_VLcbitrate, QoA_VLCframerate, and QoA_BUFFERINGcount. Analyzing these objective technical features will provide direct insights into the video playback conditions and the challenges users faced during the assessment.

I. Interpreting the QoA_VLCresolution



```
df['QoA_VLCresolution'].unique()
✓ 0.0s
array([360, 240,  16], dtype=int64)
```

The unique values for the resolution are (360*p*), (240*p*), (160*p*). Initial analysis confirms that the user assessments were conducted across three distinct resolution levels.

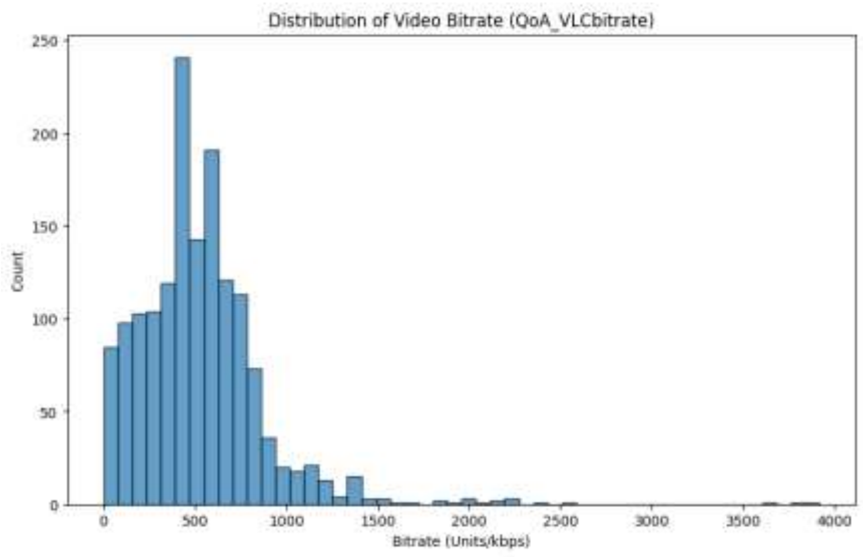


The count plot demonstrates that the 360 resolution is dominant in the dataset. This indicates that the experiment primarily focused on assessing video quality 360p, or that the network conditions most frequently defaulted to this resolution.

Since the resolution, levels are ***distinct categories*** that might not have a linear relationship with MOS (I mean the difference in perceived quality between 360p and 240p might not be the same as the difference between 240p and 160p), I can treat the QoA_VLCresolution as a categorical variable and apply one-hot encoding. This forces the model to treat each resolution as a separate feature.

II. Bitrate Analysis

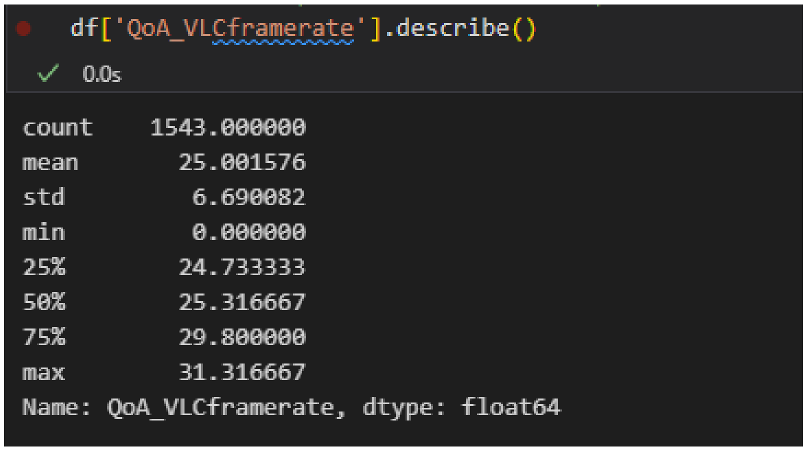
The QoA_VLCbitrate feature represents the effective data rate of the video stream being decoded and played by the user's VLC player at the moment. It is a direct result of the Network Quality of Service (QoS).



The distribution is heavily skewed due to a few extremely high-bitrate outliers. This wide dispersion is crucial, as it captures the broad range of actual network-induced video qualities encountered by users. To mitigate this, applying a $\log(1 + x)$ transformation to compress the high-end values and stabilize the variance, creating a distribution that is more suitable for modeling.

III. Interpreting QoA_VLCframerate

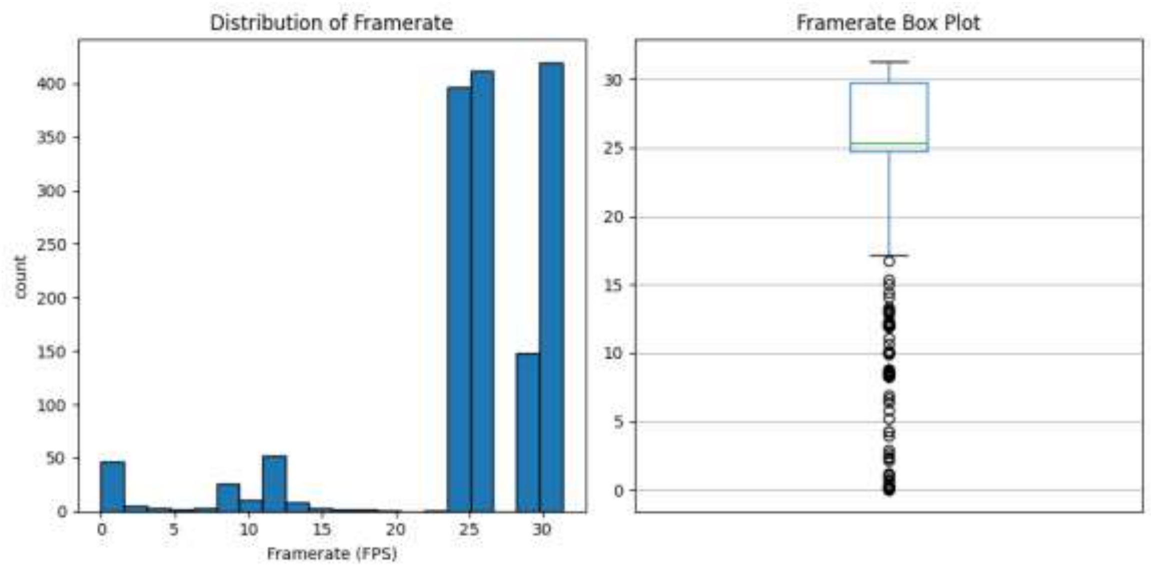
A **stable** framerate is more important than a high framerate. I am going to analyze the central tendency, spread, and relationship with the target variable (MOS).



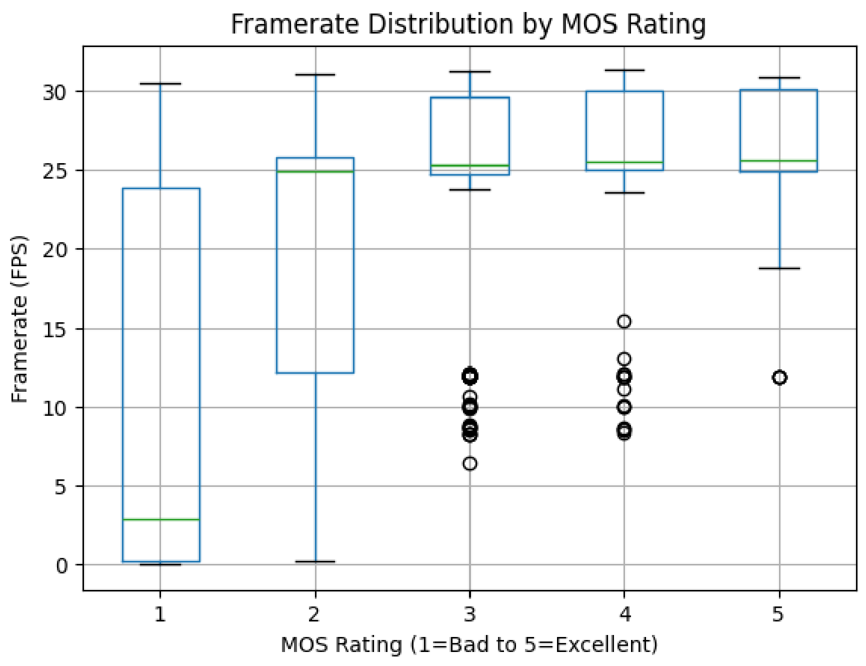
```
df['QoA_VLCframerate'].describe()
```

```
count    1543.000000
mean       25.001576
std         6.690082
min         0.000000
25%        24.733333
50%        25.316667
75%        29.800000
max        31.316667
Name: QoA_VLCframerate, dtype: float64
```

The Standard deviation is high (6.69), the framerate is unstable, which is a major red flag for QoE, and the minimum is zero, some users experienced near-stalled video playback.

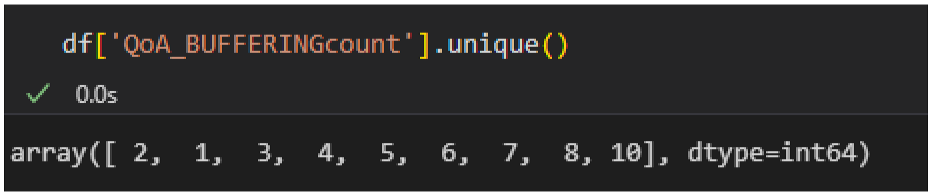


As a final step in interpreting this variable, I will generate a **Box Plot of QoA_VLCframerate against the MOS (Mean Opinion Score)**. This will visually reveal how changes in video smoothness correlate with the users' subjective ratings, allowing me to confirm the variable's predictive strength.



The median framerate increase as the MOS rating goes from 1 to 5.

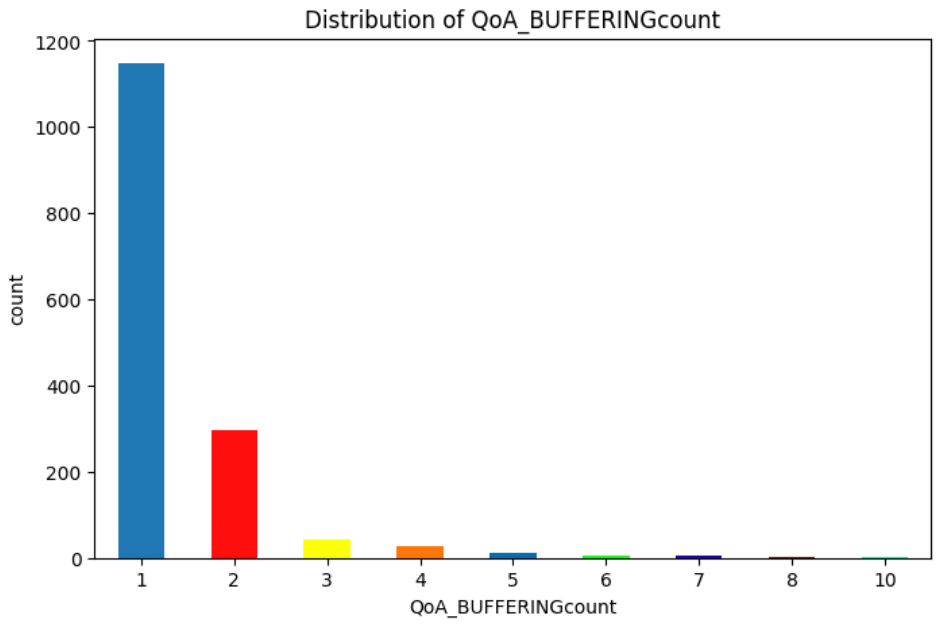
IV. Interpretation of QoA_BUFFERINGcount



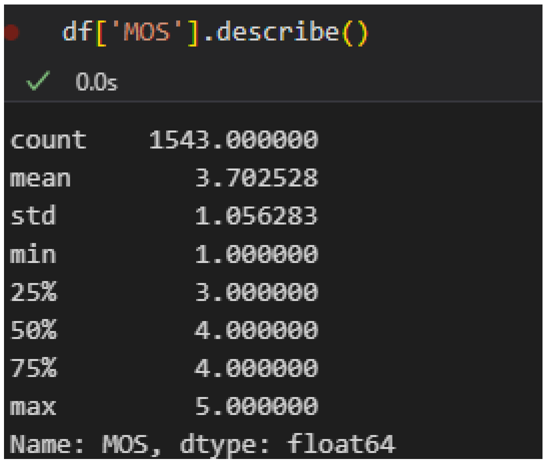
```
df['QoA_BUFFERINGcount'].unique()
✓ 0.0s
array([ 2,  1,  3,  4,  5,  6,  7,  8, 10], dtype=int64)
```

The array shows the unique integer values, this means that across all the observed video streaming sessions, the number of times buffering occurred was one of these values: 1, 2, 3, 4, 5, 6, 7, 8, 10.

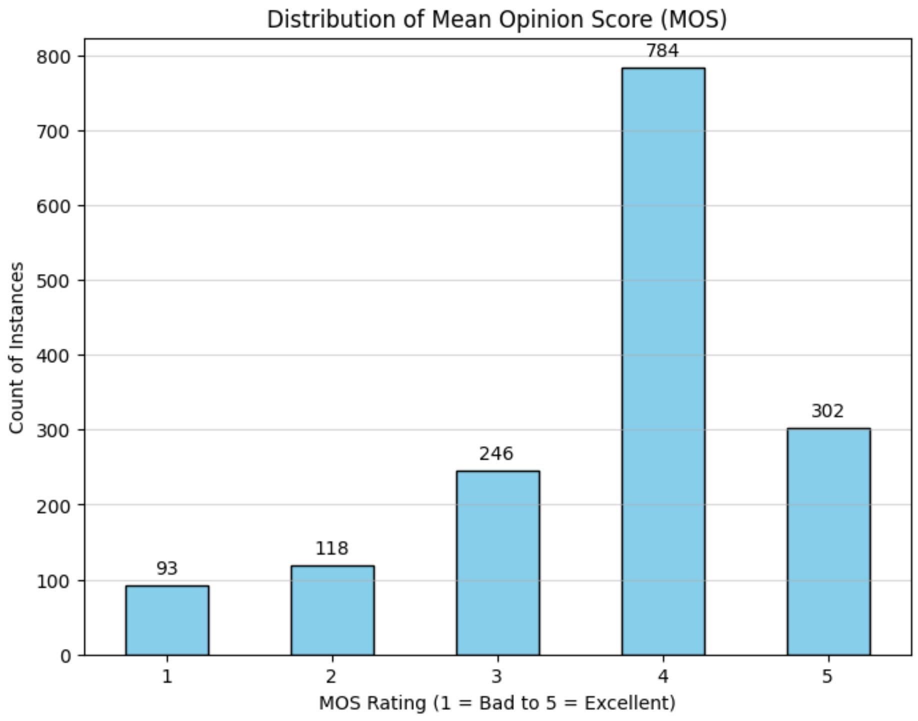
The QoA_BUFFERINGcount represents the total number of times the media player had to pause playback and re-buffer data to fill its buffer queue during the video session.



V. Focusing on the target



```
df['MOS'].describe()
0.0s

count    1543.000000
mean        3.702528
std         1.056283
min         1.000000
25%         3.000000
50%         4.000000
75%         4.000000
max         5.000000
Name: MOS, dtype: float64
```



The bar chart visually confirms that the MOS distribution is skewed toward positive ratings (4 and 5), indicating an imbalance in the target classes.

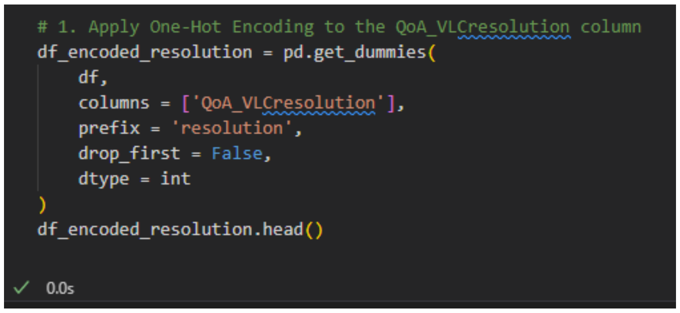
- Dominant Class (MOS = 4): with 784 instances, this is where the model will achieve its highest accuracy.
- Minority Classes (Low QoE): the low scores(MOS = 1, and 2) these classes are the hardest to predict due to low sample size.

We can conclude clearly that to mitigate the class Imbalance I can use techniques like class **weighting** or **oversampling** (SMOTE) during model training to prevent the model from ignoring the low MOS scores. Therefore, a critical element of our analysis will involve comparing the model's performance **with and without the application of the SMOTE (Synthetic Minority Over-sampling Technique)**. This direct comparison will form a key part of the discussion and will determine the optimal strategy for addressing the target variable's class imbalance in the conclusion.

Having visually explored the data's structure and identified potential quality issues, we can now pivot our focus. The following Data Preprocessing steps will transform the raw data into a clean, structured format, specifically by handling categorical variables and normalizing features, in preparation for the machine-learning phase.

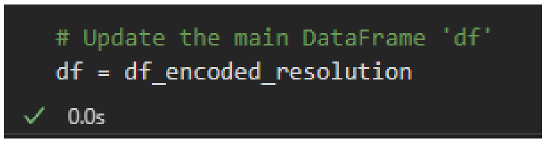
3. Data Preprocessing

I will start with QoA_VLCresolution the numerical values (360p, 240p, 160p) are categories, not continuous measurements. The model should not assume that the difference in QoE between 160p and 240p is the same as 240p and 360p. One Hot Encoding creates separate binary features for each level.



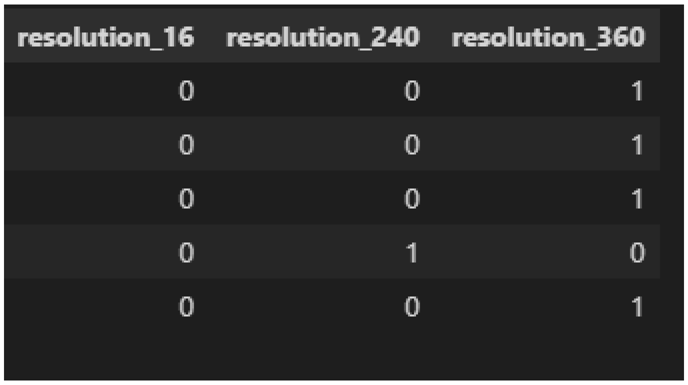
```
# 1. Apply One-Hot Encoding to the QoA_VLCresolution column
df_encoded_resolution = pd.get_dummies(
    df,
    columns = ['QoA_VLCresolution'],
    prefix = 'resolution',
    drop_first = False,
    dtype = int
)
df_encoded_resolution.head()
```

✓ 0.0s



```
# Update the main DataFrame 'df'
df = df_encoded_resolution
```

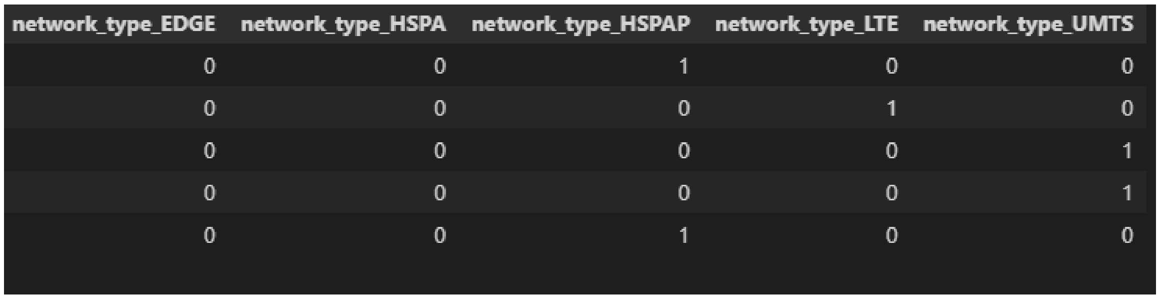
✓ 0.0s



resolution_16	resolution_240	resolution_360
0	0	1
0	0	1
0	0	1
0	1	0
0	0	1

The result of the One-Hot Encoding process for the QoA_VLCresolution column. Since a single observation (row) can only have one resolution at a time, only one of the new columns can be 1 for any given row, while the others must be 0.

Moving to the QoS_type column, this feature uses class identifiers (1: EDGE, 2: UMTS, 3: HSPA, etc.). The numerical values are **nominal**; that is, 4 is not better than 1 in a continuous or ordinal sense, as the **simply represent different netwok technologies**. One-Hot Encoding is used here to avoid creating an artificial ordinal relationship where none exists.

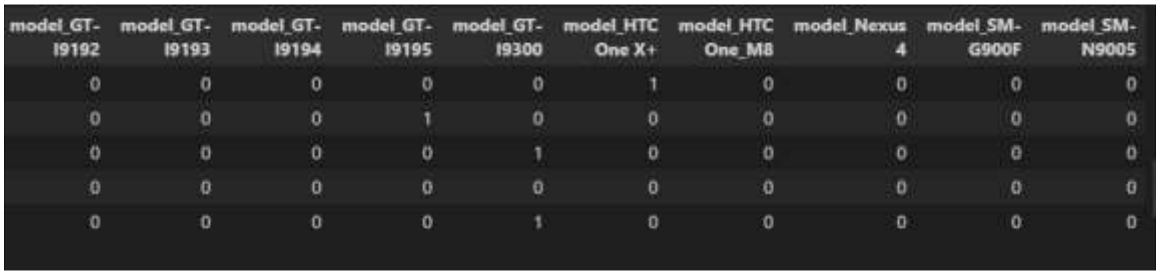


network_type_EDGE	network_type_HSPA	network_type_HSPAP	network_type_LTE	network_type_UMTS
0	0	1	0	0
0	0	0	1	0
0	0	0	0	1
0	0	0	0	1
0	0	1	0	0

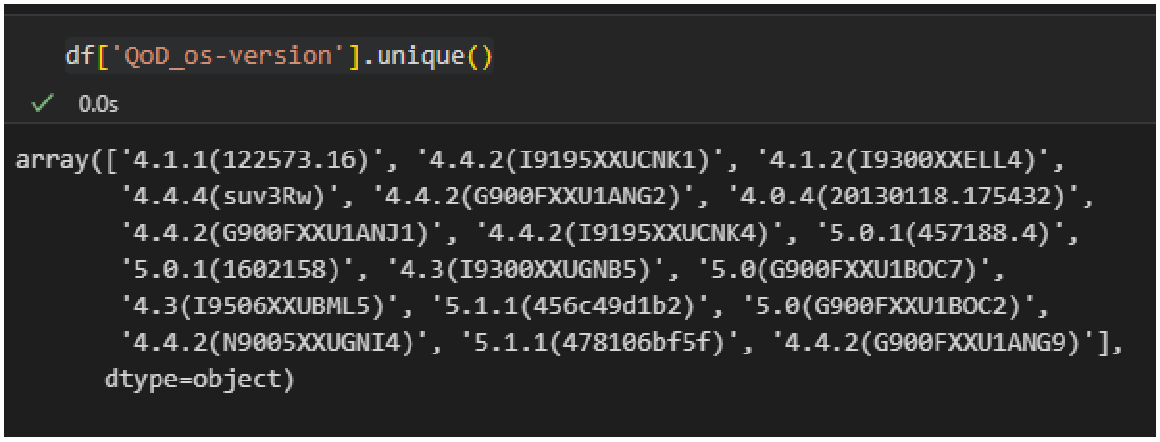
The use of descriptive column names significantly enhances the interpretability of the final predictive model, allowing us to clearly attribute changes in MOS to specific network environments.

Justification for Encoding Device Model

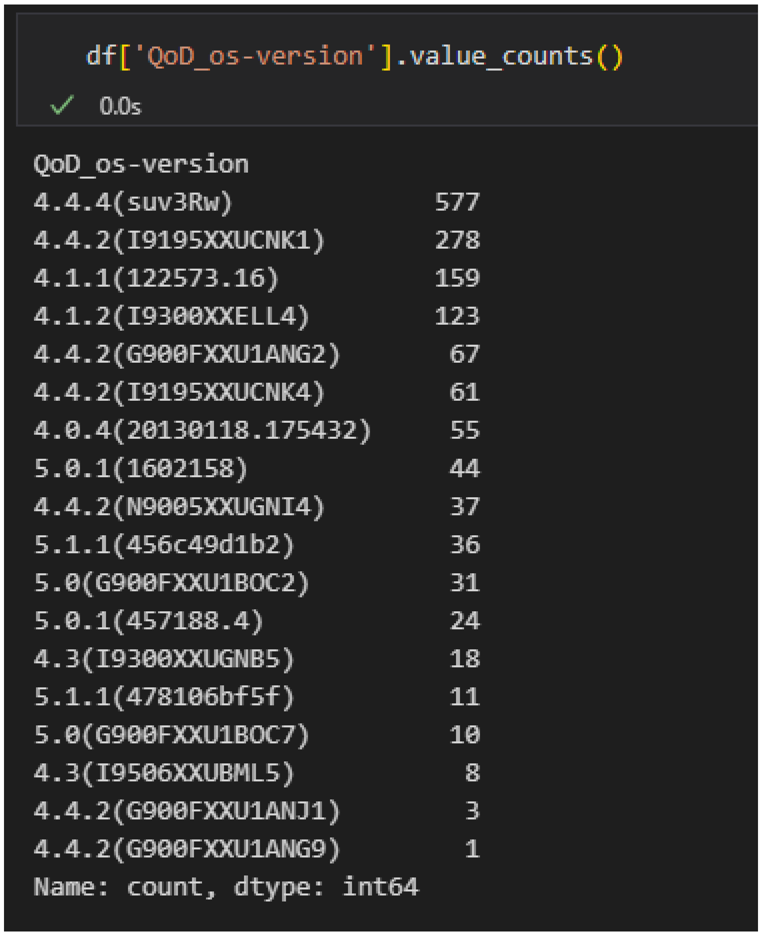
The QoD_model attribute, which identifies the specific device used for testing, was transformed using One-Hot Encoding. This preprocessing step in my opinion is mandatory because the column's values are purely nominal (Categorical), and treating them as numerical would impose a meaningless and artificial ranking. By creating a binary feature for each unique device model, the model can independently asses the contribution of each device to the final MOS.



model_GT-I9192	model_GT-I9193	model_GT-I9194	model_GT-I9195	model_GT-I9300	model_HTC One X+	model_HTC One_M8	model_Nexus 4	model_SM-G900F	model_SM-N9005
0	0	0	0	0	1	0	0	0	0
0	0	0	1	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	0



```
df['QoD_os-version'].unique()
✓ 0.0s

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)
```



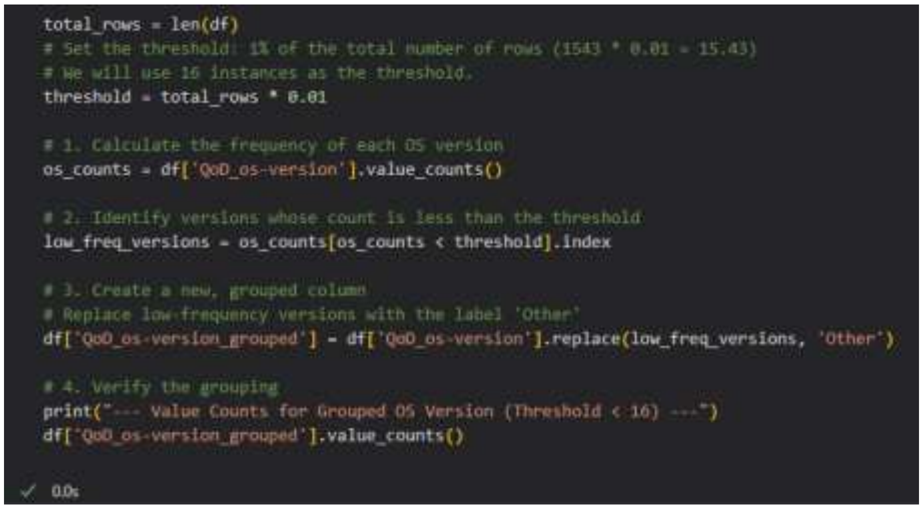
```
df['QoD_os-version'].value_counts()
✓ 0.0s

QoD_os-version
4.4.4(suv3Rw)             577
4.4.2(I9195XXUCNK1)       278
4.1.1(122573.16)          159
4.1.2(I9300XXELL4)        123
4.4.2(G900FXXU1ANG2)       67
4.4.2(I9195XXUCNK4)        61
4.0.4(20130118.175432)     55
5.0.1(1602158)             44
4.4.2(N9005XXUGNI4)        37
5.1.1(456c49d1b2)          36
5.0(G900FXXU1BOC2)         31
5.0.1(457188.4)            24
4.3(I9300XXUGNB5)          18
5.1.1(478106bf5f)          11
5.0(G900FXXU1BOC7)         10
4.3(I9506XXUBML5)           8
4.4.2(G900FXXU1ANJ1)        3
4.4.2(G900FXXU1ANG9)        1
Name: count, dtype: int64
```

The unique values for the QoD_os-version feature show a high degree of cardinality and structural complexity.

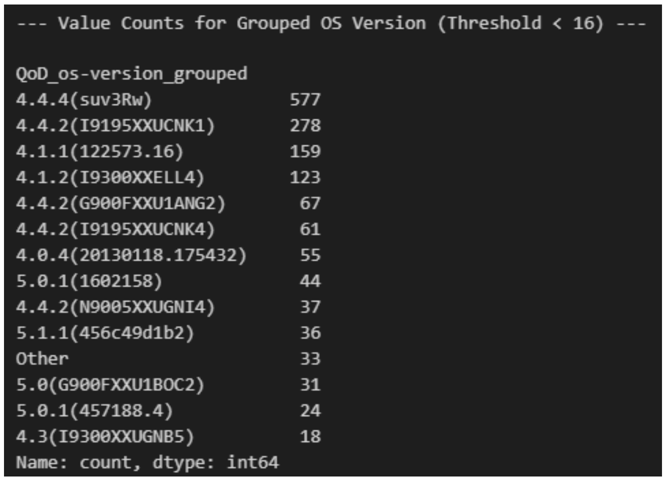
This heterogeneity means the feature is not immediately usable for modeling.

Here I am going to implement a feature grouping strategy.

Defining a threshold based on 1% of the total observations. Any version appearing less than this threshold will be lumped into an 'Other' category. Then I will apply OHE.

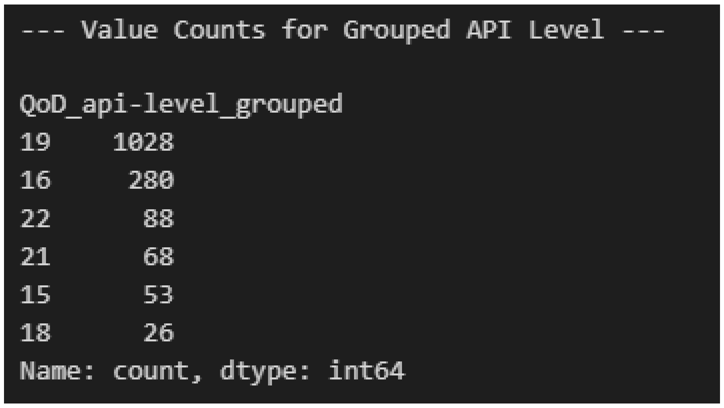


```
total_rows = len(df)
# Set the threshold: 1% of the total number of rows (1543 * 0.01 = 15.43)
# We will use 16 instances as the threshold.
threshold = total_rows * 0.01

# 1. Calculate the frequency of each OS version
os_counts = df['QoD_os-version'].value_counts()

# 2. Identify versions whose count is less than the threshold
low_freq_versions = os_counts[os_counts < threshold].index

# 3. Create a new, grouped column
# Replace low-frequency versions with the label 'Other'
df['QoD_os-version_grouped'] = df['QoD_os-version'].replace(low_freq_versions, 'Other')

# 4. Verify the grouping
print("--- Value Counts for Grouped OS Version (Threshold < 16) ---")
df['QoD_os-version_grouped'].value_counts()

✓ 0.0s
```

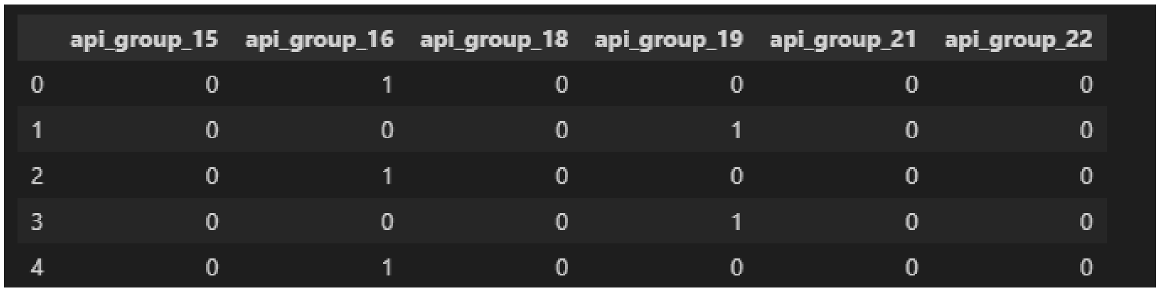


```
--- Value Counts for Grouped OS Version (Threshold < 16) ---

QoD_os-version_grouped
4.4.4(suv3Rw)            577
4.4.2(I9195XXUCNK1)      278
4.1.1(122573.16)         159
4.1.2(I9300XXELL4)       123
4.4.2(G900FXXU1ANG2)      67
4.4.2(I9195XXUCNK4)       61
4.0.4(20130118.175432)    55
5.0.1(1602158)            44
4.4.2(N9005XXUGNI4)       37
5.1.1(456c49d1b2)         36
Other                     33
5.0(G900FXXU1BOC2)        31
5.0.1(457188.4)           24
4.3(I9300XXUGNB5)         18
Name: count, dtype: int64
```

To prevent model overfitting on rare device environments and mitigate the curse of dimensionality, a Feature grouping strategy was implemented. All OS versions with a sample frequency below 1% were consolidated into an ‘Other’ category. The resulting grouped feature was then transformed using One-Hot Encoding, yielding a smaller, more robust set of features.

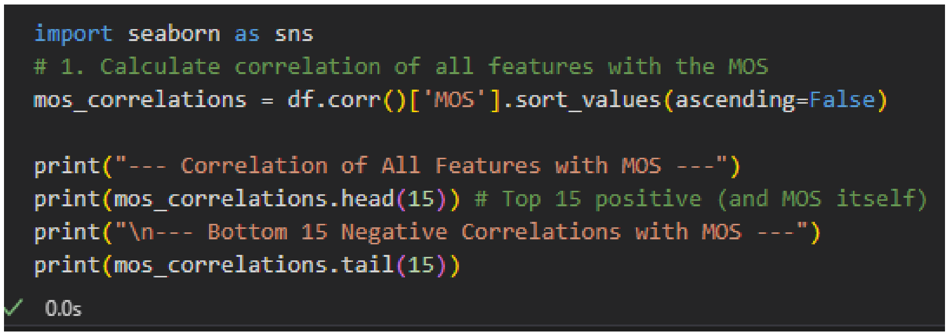


```
--- Value Counts for Grouped API Level ---

QoD_api-level_grouped
19    1028
16     280
22      88
21      68
15      53
18      26
Name: count, dtype: int64
```



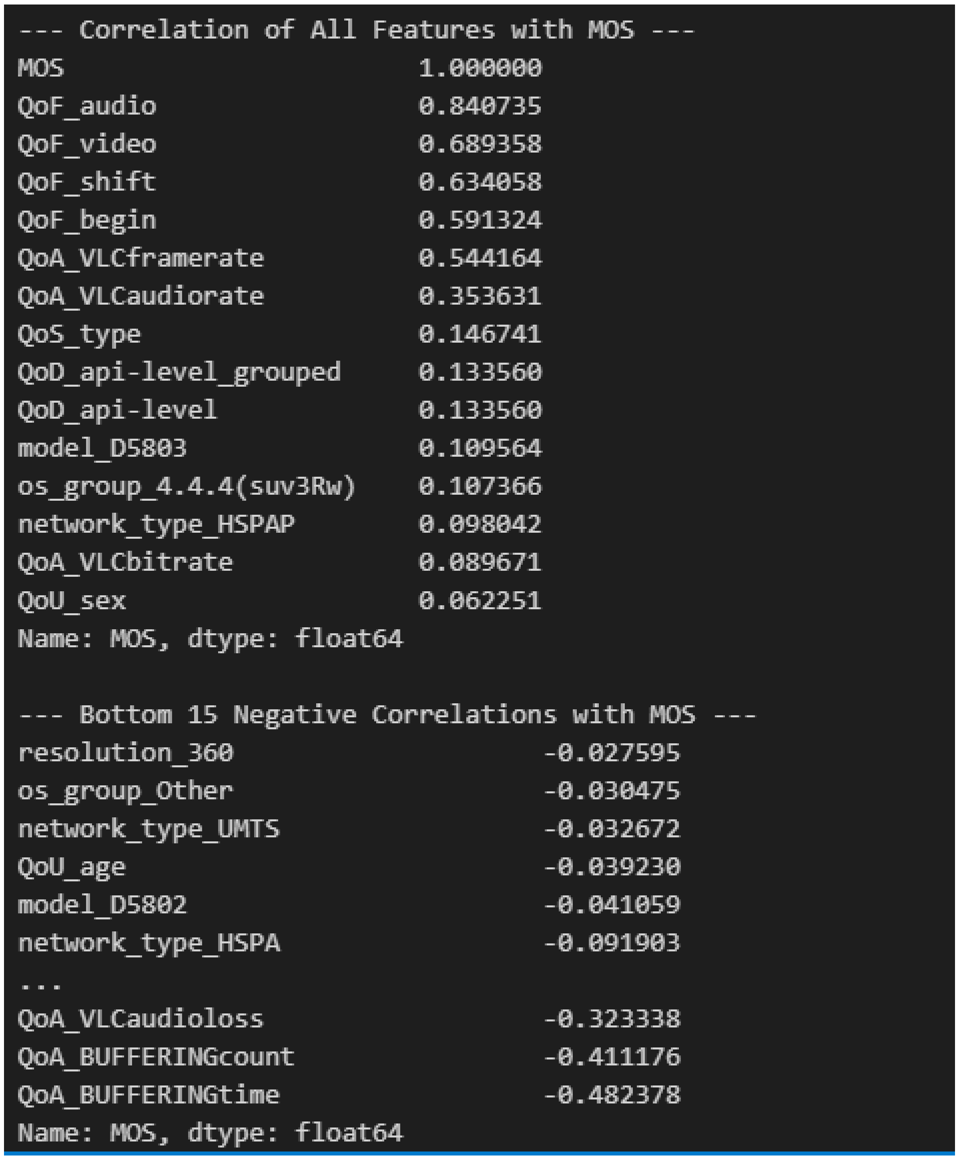
	api_group_15	api_group_16	api_group_18	api_group_19	api_group_21	api_group_22
0	0	1	0	0	0	0
1	0	0	0	1	0	0
2	0	1	0	0	0	0
3	0	0	0	1	0	0
4	0	1	0	0	0	0

As a key step in our analysis, we computed the correlation between each features and the MOS target variable. This allows me to identify and quantify the relationships, highlighting which features are most significant in predicting the outcome.

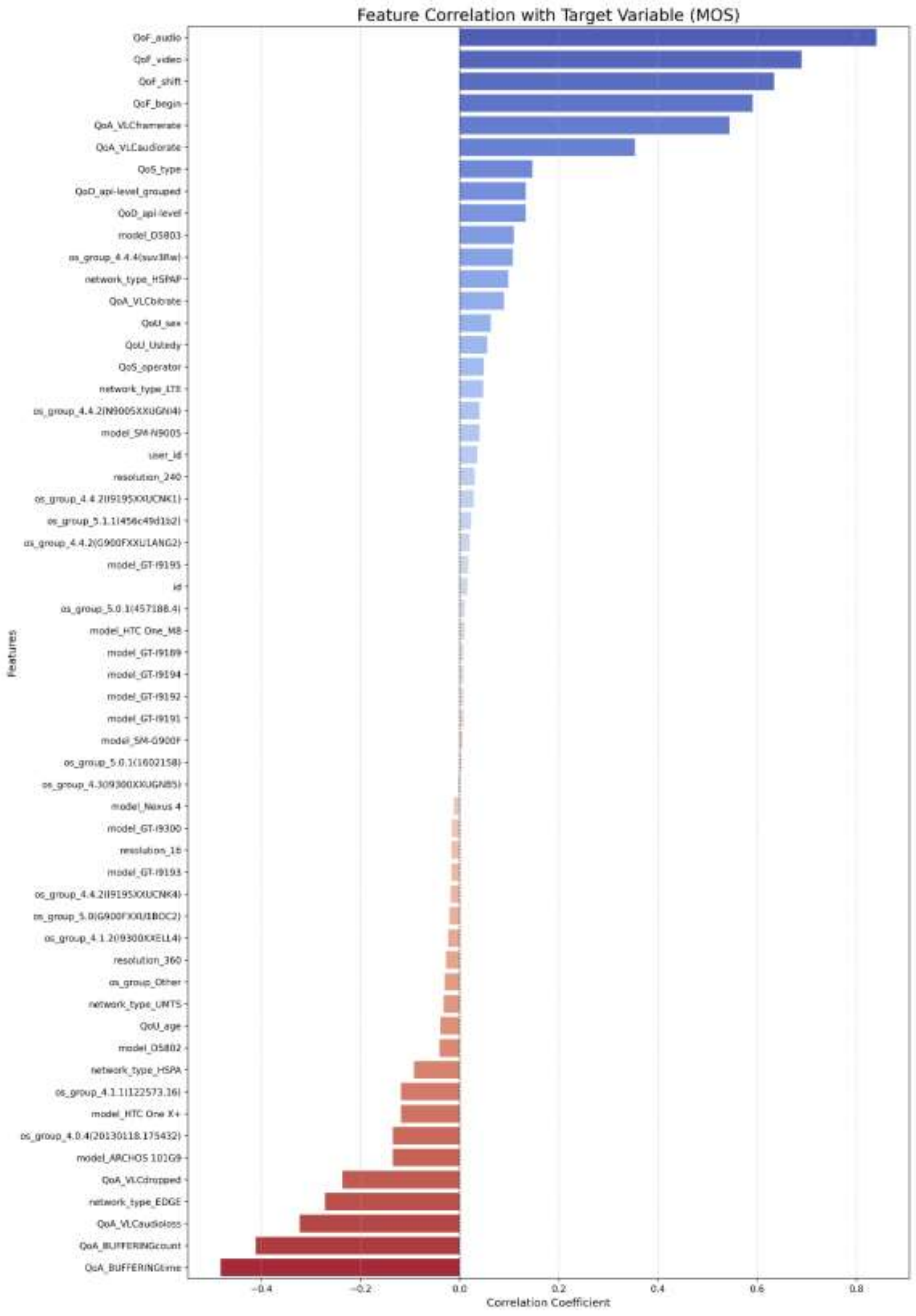


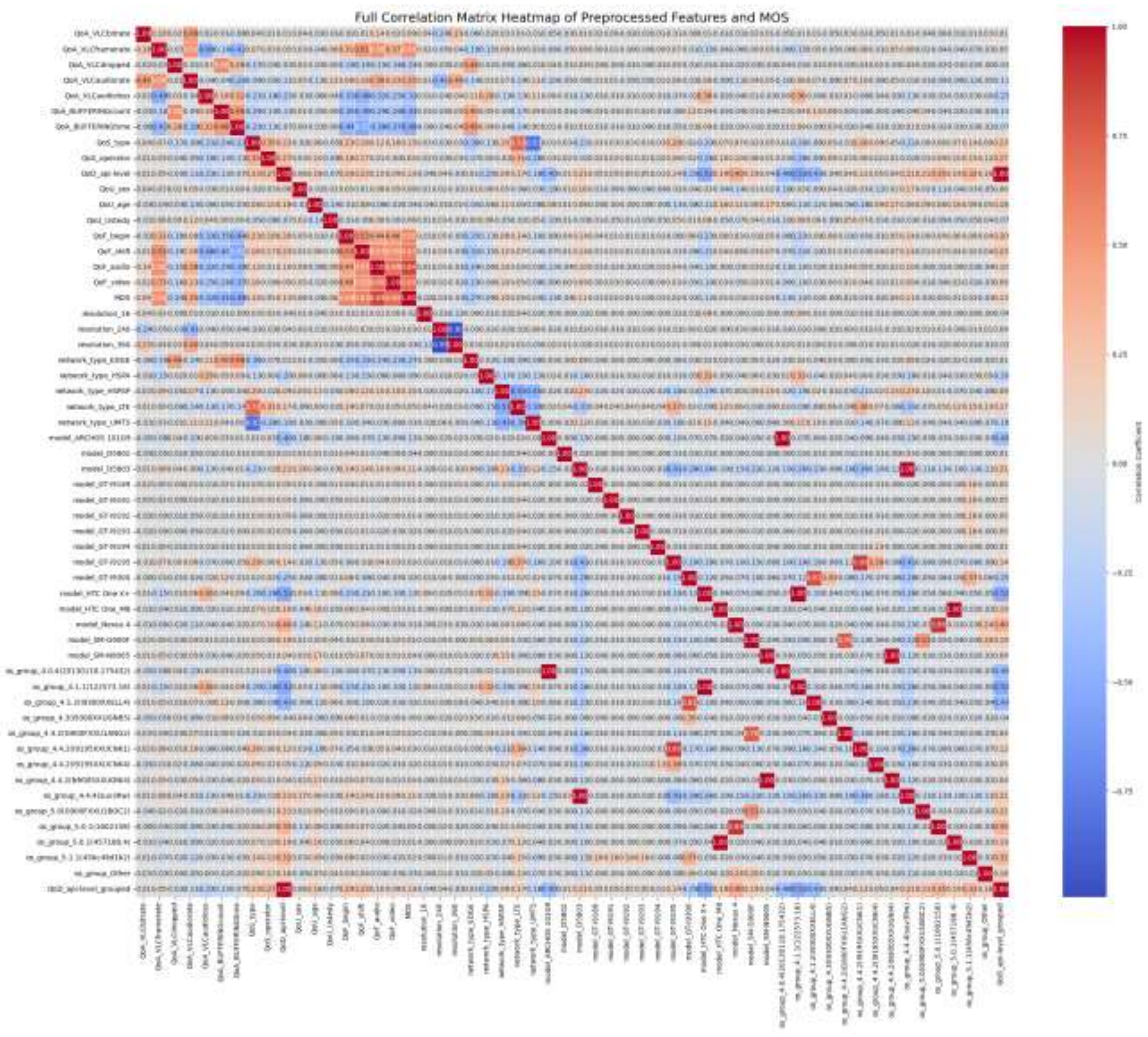
```
import seaborn as sns
# 1. Calculate correlation of all features with the MOS
mos_correlations = df.corr()['MOS'].sort_values(ascending=False)

print("--- Correlation of All Features with MOS ---")
print(mos_correlations.head(15)) # Top 15 positive (and MOS itself)
print("\n--- Bottom 15 Negative Correlations with MOS ---")
print(mos_correlations.tail(15))
```

✓ 0.0s



```
--- Correlation of All Features with MOS ---
MOS                          1.000000
QoF_audio                    0.840735
QoF_video                    0.689358
QoF_shift                    0.634058
QoF_begin                    0.591324
QoA_VLCframerate             0.544164
QoA_VLCaudiorate             0.353631
QoS_type                     0.146741
QoD_api-level_grouped        0.133560
QoD_api-level                0.133560
model_D5803                  0.109564
os_group_4.4.4(suv3Rw)       0.107366
network_type_HSPAP           0.098042
QoA_VLCbitrate               0.089671
QoU_sex                      0.062251
Name: MOS, dtype: float64

--- Bottom 15 Negative Correlations with MOS ---
resolution_360                      -0.027595
os_group_Other                      -0.030475
network_type_UMTS                   -0.032672
QoU_age                             -0.039230
model_D5802                         -0.041059
network_type_HSPA                   -0.091903
...
QoA_VLCaudioloss                    -0.323338
QoA_BUFFERINGcount                  -0.411176
QoA_BUFFERINGtime                   -0.482378
Name: MOS, dtype: float64
```


Feature Correlation with Target Variable (MOS)
QoF_audio
QoF_video
QoF_shift
QoF_begin
QoA_VLCframerate
QoA_VLCaudiorate
QoS_type
QoD_api-level_grouped
QoD_api-level
model_D5803
os_group_4.4.4(suv3llw)
network_type_HSPAP
QoA_VLCbitrate
QoU_sex
QoU_Ustedy
QoS_operator
network_type_LTE
os_group_4.4.2(N9005XXUGNI4)
model_SM-N9005
user_id
resolution_240
os_group_4.4.2(I9195XXUCNK1)
os_group_5.1.1(456c49d1b2)
os_group_4.4.2(G900FXXU1ANG2)
model_GT-I9195
id
os_group_5.0.1(457188.4)
model_HTC One_M8
model_GT-I9189
model_GT-I9194
model_GT-I9192
model_GT-I9191
model_SM-G900F
os_group_5.0.1(1602158)
os_group_4.3(I9300XXUGNB5)
model_Nexus 4
model_GT-I9300
resolution_18
model_GT-I9193
os_group_4.4.2(I9195XXUCNK4)
os_group_5.0(G900FXXU1BOC2)
os_group_4.1.2(I9300XXELL4)
resolution_360
os_group_Other
network_type_UMTS
QoU_age
model_D5802
network_type_HSPA
os_group_4.1.1(122573.16)
model_HTC One X+
os_group_4.0.4(20130118.175432)
model_ARCHOS 101G9
QoA_VLCdropped
network_type_EDGE
QoA_VLCaudioloss
QoA_BUFFERINGcount
QoA_BUFFERINGtime
Features
−0.4
−0.2
0.0
0.2
0.4
0.6
0.8
Correlation Coefficient



The figures provided give us good insights into **which features** to keep and which to drop.

4. Feature Selection

- I will drop the identifiers (user_id, id) which should not influence the target.
- As a next step we must perform a visual inspection of the correlation heatmap of the refined feature set and specifically look for pairs with an absolute correlation of $|r| > 0.8$ or higher.
    - **We will drop these features**: api_group_18 ,os_group_4.1.1(122573.16), os_group_4.4.2(I9195XXUCNK1),

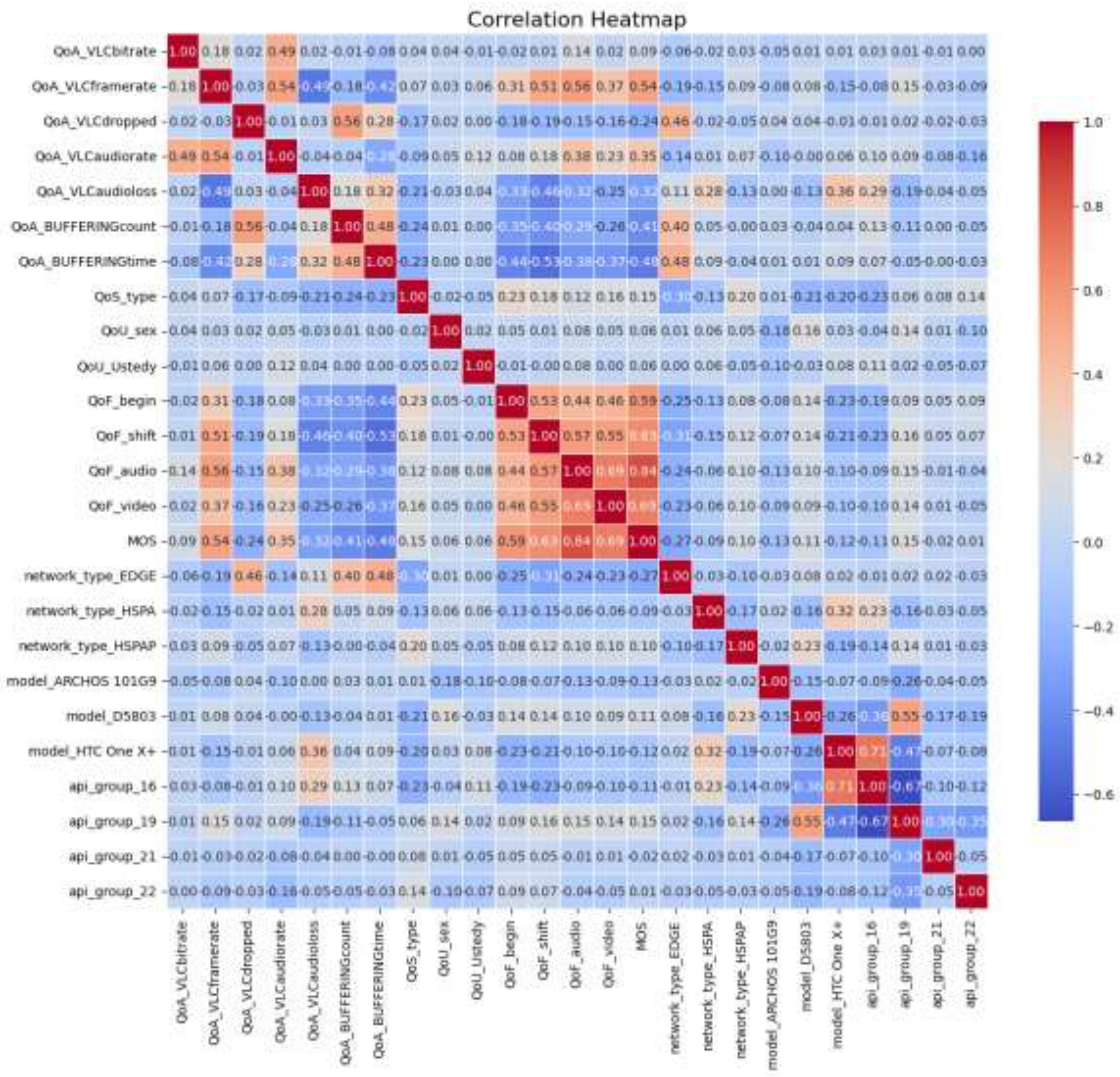
os_group_4.0.4(20130118.175432), os_group_4.4.4(suv3Rw), os_group_5.0.1(457188.4), os_group_5.0.1(1602158), os_group_4.1.2(I9300XXELL4), os_group_4.4.2(N9005XXUGNI4), QoD_api-level_grouped, network_type_UMTS, resolution_360.

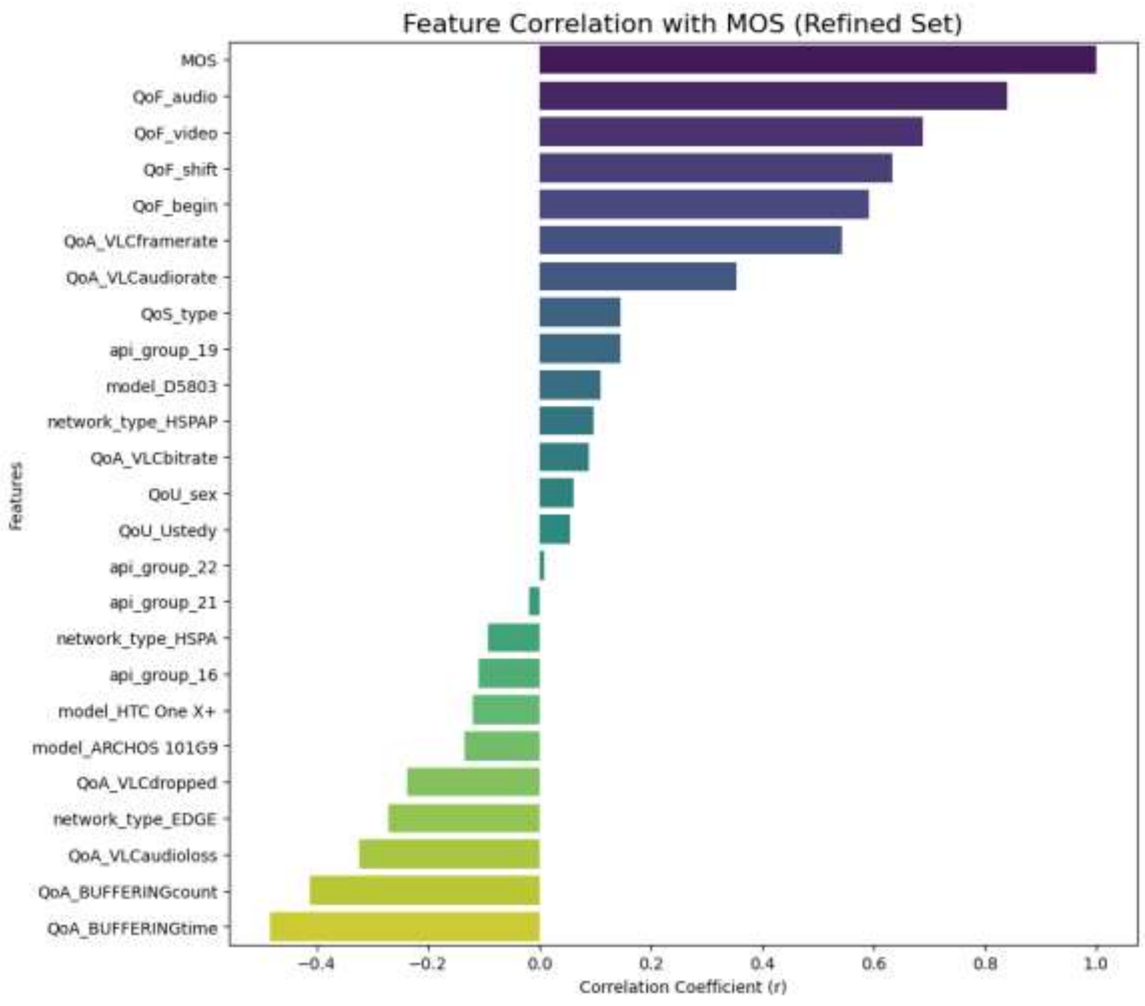
Therefore, until now I dropped 15! Features.

- We must address features that offer little or no predictive value for the target variable, MOS. My criterion: Features were flagged for removal if their absolute correlation with MOS was below a very low threshold (0.05).
    - **We will drop these features**: 'QoS_operator', 'network_type_LTE', 'os_group_4.4.2(N9005XXUGNI4)', 'model_SM-N9005', 'user_id', 'resolution_240', 'os_group_4.4.2(I9195XXUCNK1)', 'os_group_5.1.1(456c49d1b2)', 'os_group_4.4.2(G900FXXU1ANG2)', 'model_GT-I9195', 'id', 'os_group_5.0.1(457188.4)', 'model_HTC One_M8', 'model_GT-I9189', 'model_GT-I9194', 'model_GT-I9192', 'model_GT-I9191', 'model_SM-G900F', 'os_group_5.0.1(1602158)', 'os_group_4.3(I9300XXUGNB5)', 'model_Nexus 4', 'model_GT-I9300', 'resolution_16', 'model_GT-I9193', 'os_group_4.4.2(I9195XXUCNK4)', 'os_group_5.0(G900FXXU1BOC2)', 'os_group_4.1.2(I9300XXELL4)', 'resolution_360', 'os_group_Other', 'network_type_UMTS', 'QoU_age', 'model_D5802'.

In this case **the Low Correlation to** Target I dropped 35! Features.

Following the removal of the extraneous variables, we produced this **high-quality correlation heatmap**.


Correlation Heatmap
QoA_VLCbitrate
QoA_VLCframerate
QoA_VLCdropped
QoA_VLCaudiorate
QoA_VLCaudioloss
QoA_BUFFERINGcount
QoA_BUFFERINGtime
QoS_type
QoU_sex
QoU_Ustedy
QoF_begin
QoF_shift
QoF_audio
QoF_video
MOS
network_type_EDGE
network_type_HSPA
network_type_HSPAP
model_ARCHOS 101G9
model_D5803
model_HTC One X+
api_group_16
api_group_19
api_group_21
api_group_22
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6

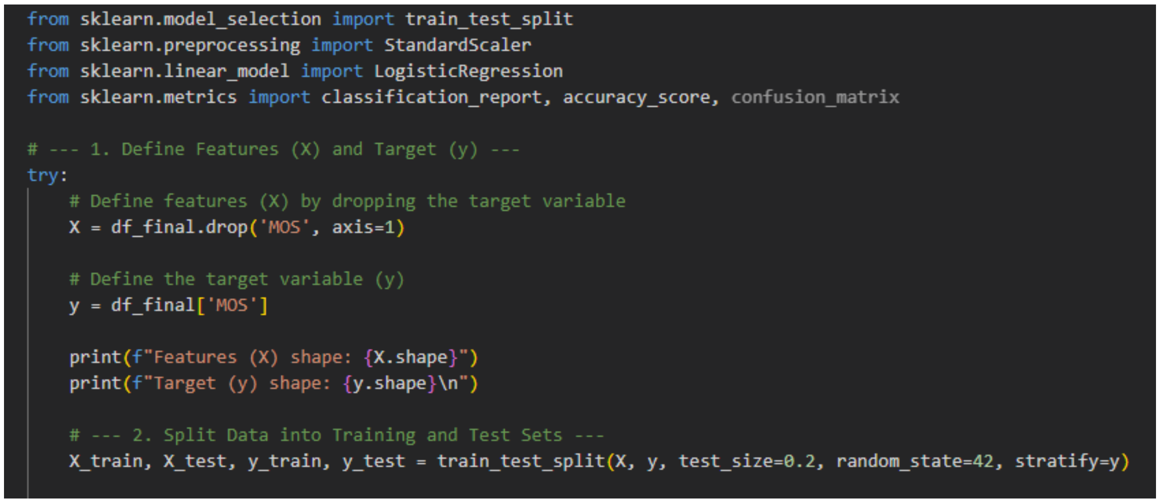


With the selected features, we establish a testing framework to evaluate and compare the performance of different **machine learning models**. This process will specify the analytical approach and the goals for the dataset investigation.

My study aims to predict Video Streaming Quality of Experience (QoE) through the development of machine learning (ML) models for estimating the **Mean Opinion Score (MOS)** based on the selected features. A key objective is to **validate the criticality** of these chosen features as factors affecting QoE.

5. Model Selection and Training Procedure

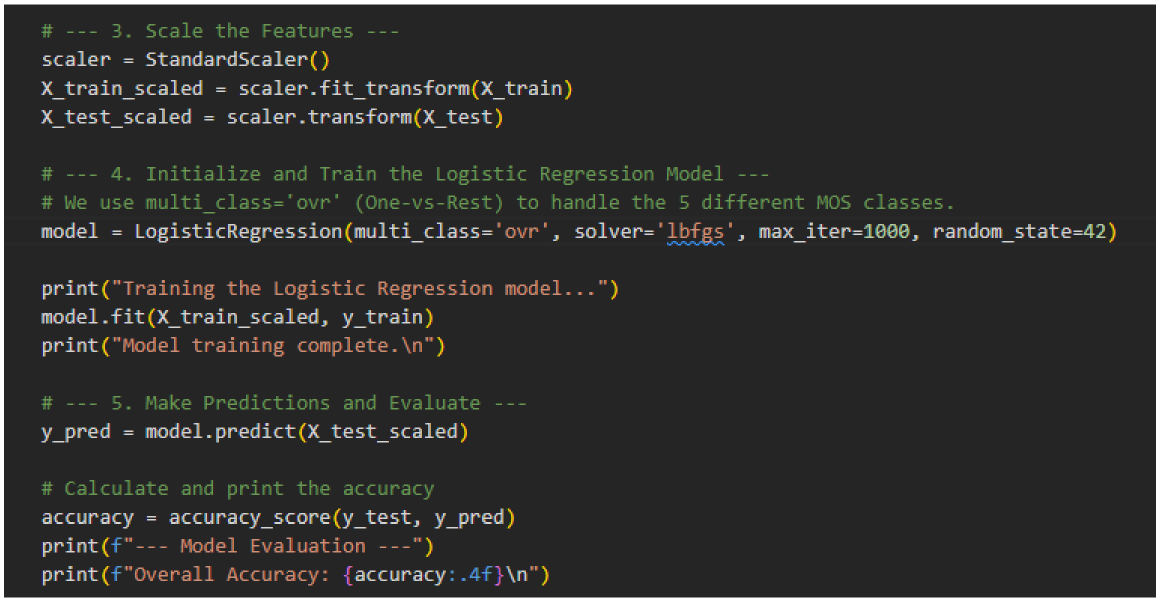
Given the encoding scheme applied to the categorical data, I chose Logistic Regression to serve as the classification model for this task.



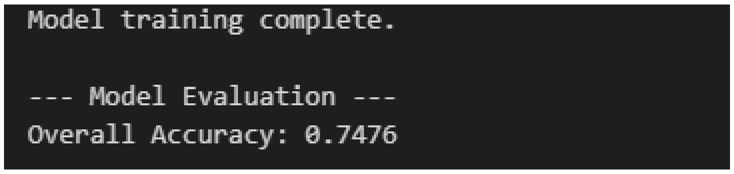
```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, accuracy_score, confusion_matrix

# --- 1. Define Features (X) and Target (y) ---
try:
    # Define features (X) by dropping the target variable
    X = df_final.drop('MOS', axis=1)

    # Define the target variable (y)
    y = df_final['MOS']

    print(f"Features (X) shape: {X.shape}")
    print(f"Target (y) shape: {y.shape}\n")

    # --- 2. Split Data into Training and Test Sets ---
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)
```

Justification why we choose Logistic Regression?

The problem involves predicting “MOS Classes”, which is a clear multiclass classification task. Logistic Regression is a fundamental and robust linear classification algorithm that serves as an excellent starting point.

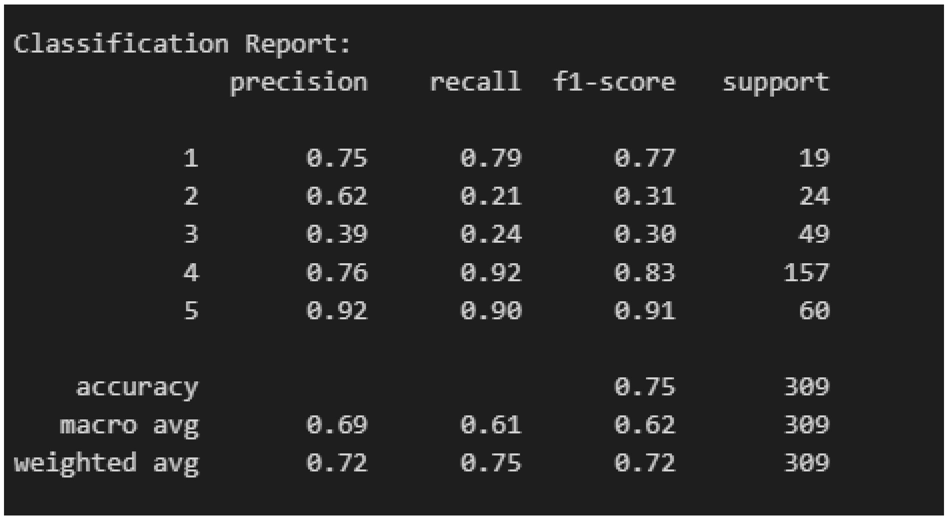


```
    # --- 3. Scale the Features ---
    scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(X_train)
    X_test_scaled = scaler.transform(X_test)

    # --- 4. Initialize and Train the Logistic Regression Model ---
    # We use multi_class='ovr' (One-vs-Rest) to handle the 5 different MOS classes.
    model = LogisticRegression(multi_class='ovr', solver='lbfgs', max_iter=1000, random_state=42)

    print("Training the Logistic Regression model...")
    model.fit(X_train_scaled, y_train)
    print("Model training complete.\n")

    # --- 5. Make Predictions and Evaluate ---
    y_pred = model.predict(X_test_scaled)

    # Calculate and print the accuracy
    accuracy = accuracy_score(y_test, y_pred)
    print(f"--- Model Evaluation ---")
    print(f"Overall Accuracy: {accuracy:.4f}\n")
```



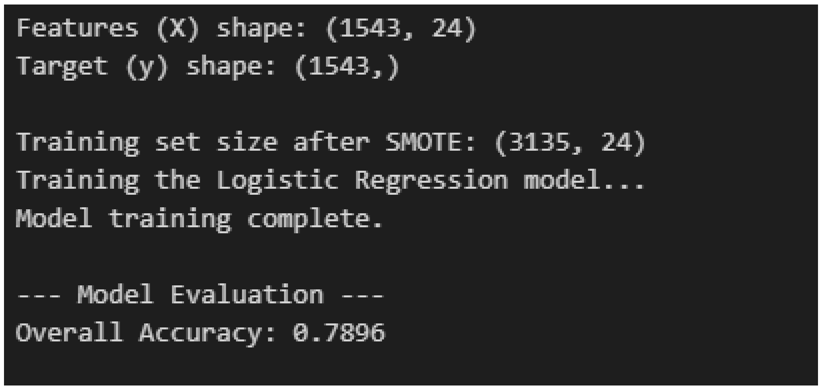
```
Model training complete.

--- Model Evaluation ---
Overall Accuracy: 0.7476
```

The simple Logistic Regression model, which I used as my baseline classifier, achieved a strong accuracy of nearly 75% for predicting Video Streaming Quality of Experience.



```
Classification Report:
              precision    recall  f1-score   support

           1       0.75      0.79      0.77        19
           2       0.62      0.21      0.31        24
           3       0.39      0.24      0.30        49
           4       0.76      0.92      0.83       157
           5       0.92      0.90      0.91        60

    accuracy                           0.75       309
   macro avg       0.69      0.61      0.62       309
weighted avg       0.72      0.75      0.72       309
```

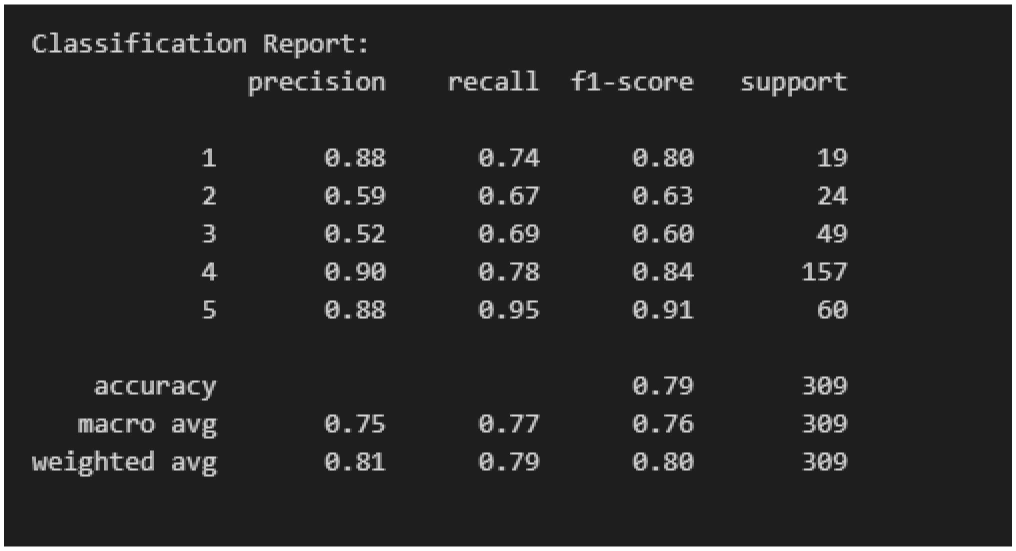
Based on the detailed classification report, the next step involves applying the Synthetic Minority Oversampling Technique (SMOTE) to address the class imbalance (Class 3 and 2). We will then evaluate the model's new accuracy and rigorously analyze the performance changes by constructing a confusion matrix.



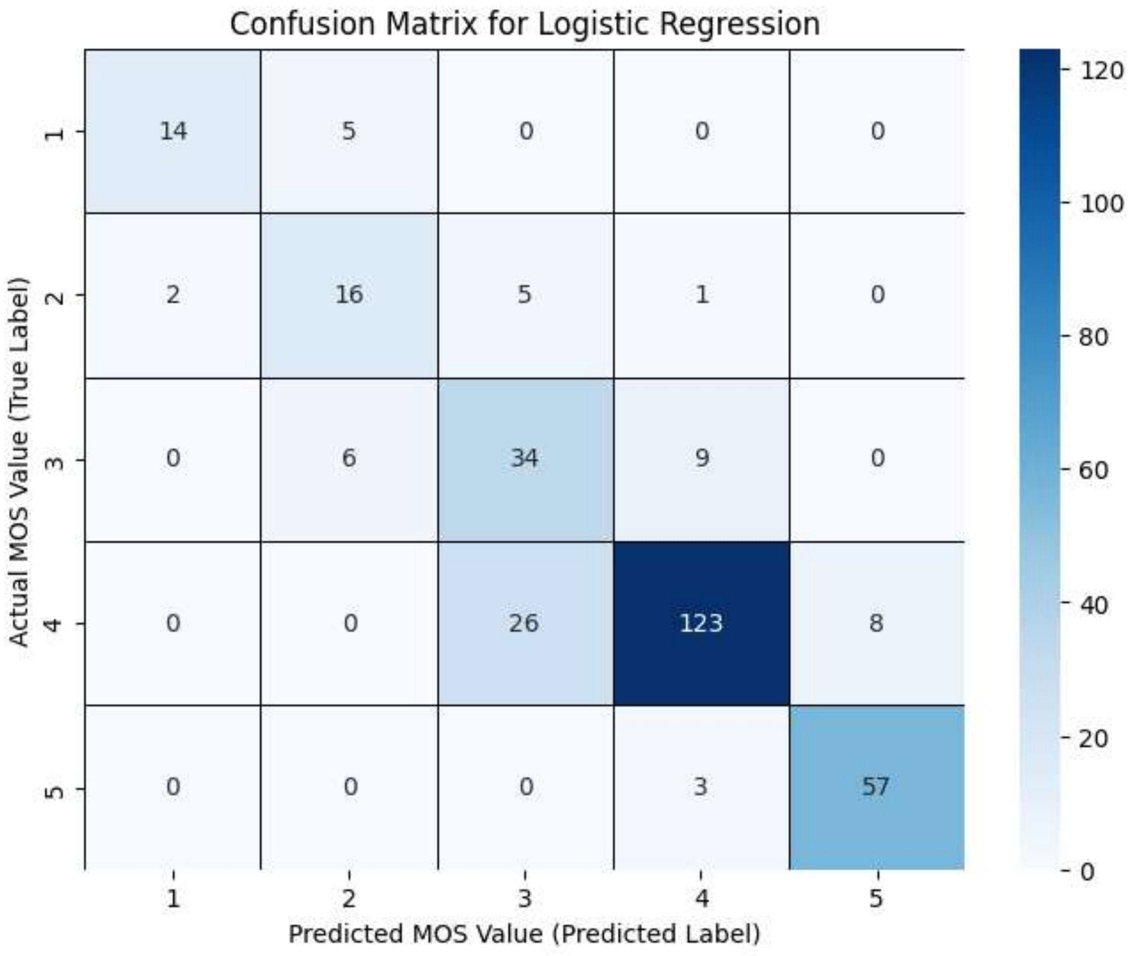
```
Features (X) shape: (1543, 24)
Target (y) shape: (1543,)

Training set size after SMOTE: (3135, 24)
Training the Logistic Regression model...
Model training complete.

--- Model Evaluation ---
Overall Accuracy: 0.7896
```

The introduction of SMOTE was highly effective, significantly improving our model's performance. The accuracy jumped into a strong 0.7896 (**nearly 79%**), which is a clear validation that:

- Our initial feature selection was appropriate.
- Successfully resolving **the class imbalance** issue has resulted in a much more robust and **powerful classification model**.



```
Classification Report:
              precision    recall  f1-score   support

           1       0.88      0.74      0.80        19
           2       0.59      0.67      0.63        24
           3       0.52      0.69      0.60        49
           4       0.90      0.78      0.84       157
           5       0.88      0.95      0.91        60

    accuracy                           0.79       309
   macro avg       0.75      0.77      0.76       309
weighted avg       0.81      0.79      0.80       309
```



By examining the confusion matrix, we can draw conclusions about the model's performance and proceed with a more in-depth explanation of its results.

The number along the **main diagonal** represent the count of correct predictions where the model's prediction matched the actual true label. These are **our True Positive** for each class.

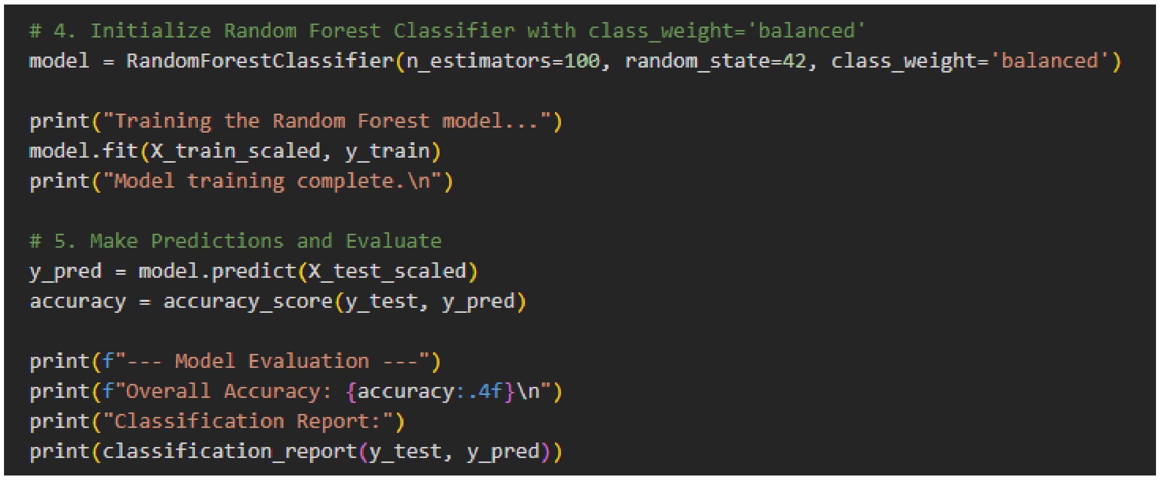
Actual (Row) →	Predicted (Column)	Count	Interpretation
1	**1**	**14**	**14** instances of MOS 1 were correctly predicted as MOS 1.
2	**2**	**16**	**16** instances of MOS 2 were correctly predicted as MOS 2.
3	**3**	**34**	**34** instances of MOS 3 were correctly predicted as MOS 3.
4	**4**	**123**	**123** instances of MOS 4 were correctly predicted as MOS 4.
5	**5**	**57**	**57** instances of MOS 5 were correctly predicted as MOS 5.

The numbers off the diagonal represent incorrect predictions "misclassification".

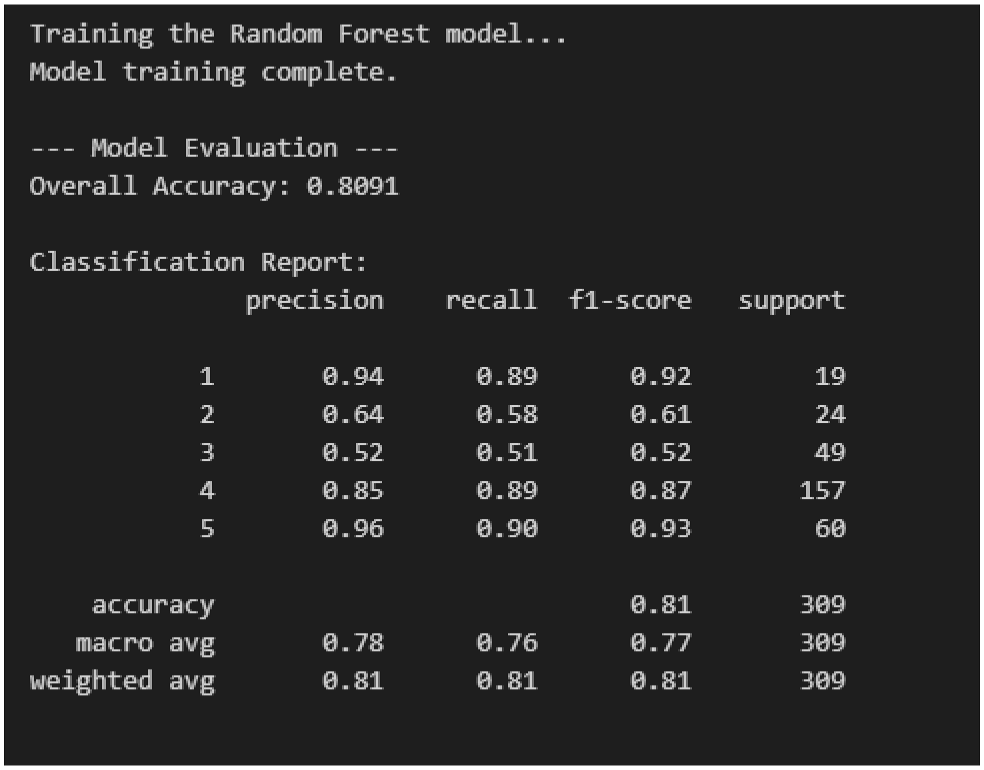
- **Row 4, Column 3 (26):** The model incorrectly predicted **26** instances that were **actually MOS 4** as **MOS 3**. This is a significant error, suggesting the model frequently under-predicts the quality score (MOS 4 —> MOS 3).
- **Row 3, Column 4 (9):** The model incorrectly predicted **9** instances that were **actually MOS 3** as **MOS 4**.
- **Row 2, Column 3 (5):** The model incorrectly predicted **5** instances that were **actually MOS 2** as **MOS 3**.

Our logistic regression model performs best on **classes 4 and 5**, with a high number of correct predictions. The largest number of errors when classifying MOS 4, where 26 of them were confused with MOS 3.

We will now switch classifier and test the Random Forest algorithm. This will allow us to compare its results against our current Logistic Regression performance and determine if a **non-linear**, **ensemble** model can yield further improvements.



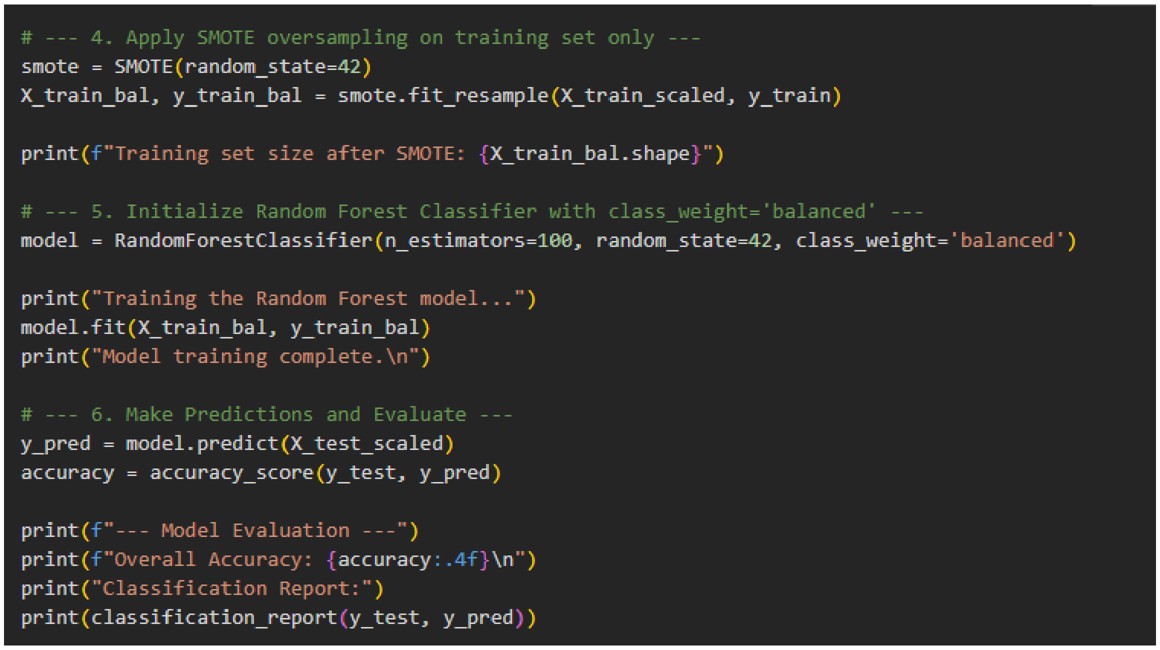
```python
# 4. Initialize Random Forest Classifier with class_weight='balanced'
model = RandomForestClassifier(n_estimators=100, random_state=42, class_weight='balanced')

print("Training the Random Forest model...")
model.fit(X_train_scaled, y_train)
print("Model training complete.\n")

# 5. Make Predictions and Evaluate
y_pred = model.predict(X_test_scaled)
accuracy = accuracy_score(y_test, y_pred)

print(f"--- Model Evaluation ---")
print(f"Overall Accuracy: {accuracy:.4f}\n")
print("Classification Report:")
print(classification_report(y_test, y_pred))
```



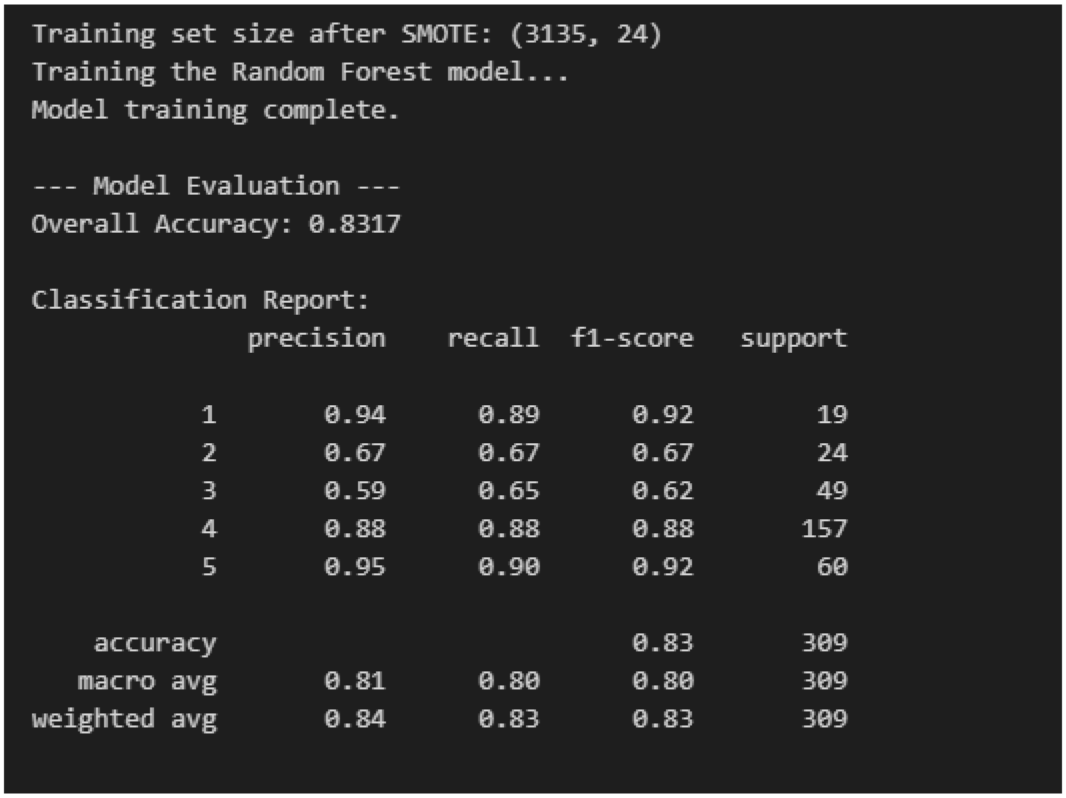
```
Training the Random Forest model...
Model training complete.

--- Model Evaluation ---
Overall Accuracy: 0.8091

Classification Report:
              precision    recall  f1-score   support

           1       0.94      0.89      0.92        19
           2       0.64      0.58      0.61        24
           3       0.52      0.51      0.52        49
           4       0.85      0.89      0.87       157
           5       0.96      0.90      0.93        60

    accuracy                           0.81       309
   macro avg       0.78      0.76      0.77       309
weighted avg       0.81      0.81      0.81       309
```

We can clearly observe a superior result: training the **non-linear classifier** yielded ***better accuracy*** even without needing to reintroduce **SMOTE**.

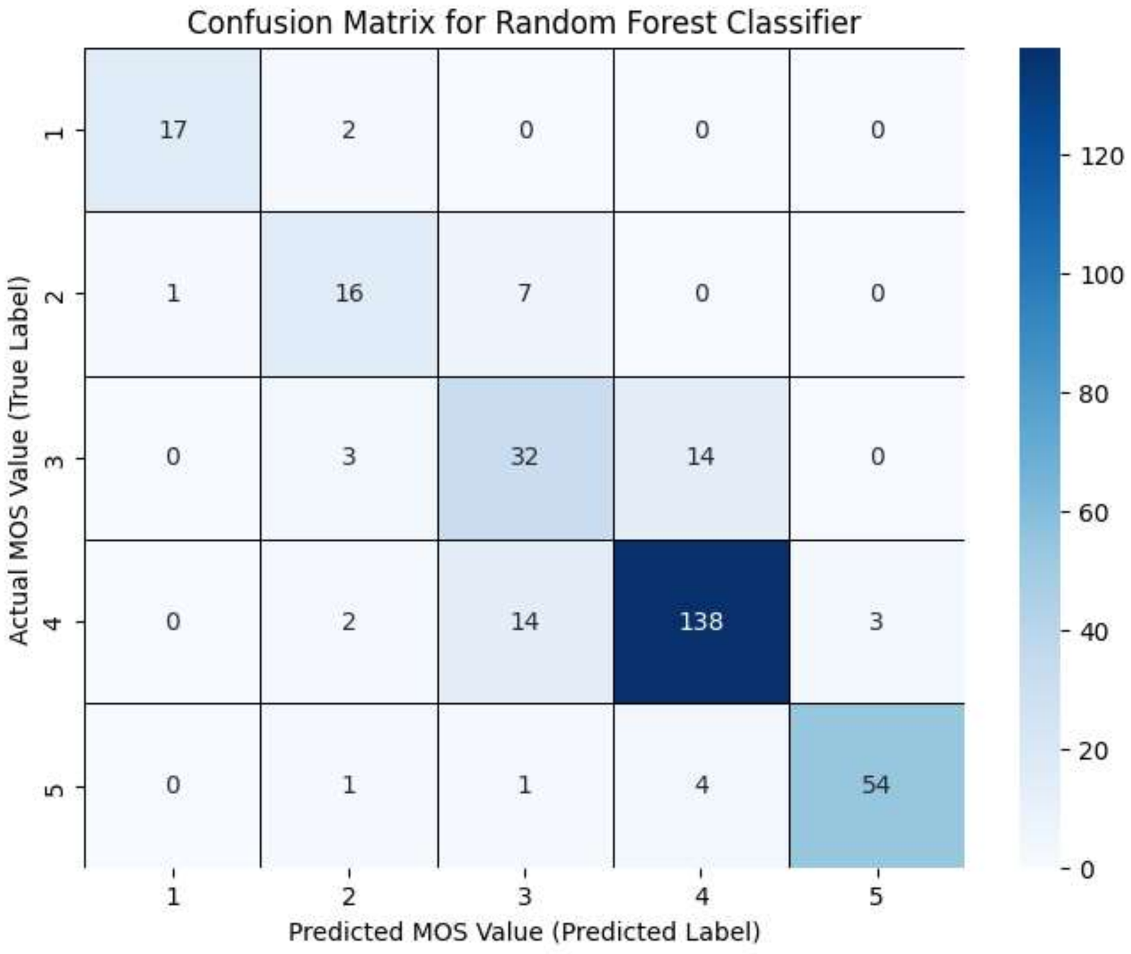
While the non-linear model achieved a great result on it's own, it's worth experimenting: I will still introduce SMOTE to address potential imbalances and observe if that leads to any additional, measurable enhancements.



```python
# --- 4. Apply SMOTE oversampling on training set only ---
smote = SMOTE(random_state=42)
X_train_bal, y_train_bal = smote.fit_resample(X_train_scaled, y_train)

print(f"Training set size after SMOTE: {X_train_bal.shape}")

# --- 5. Initialize Random Forest Classifier with class_weight='balanced' ---
model = RandomForestClassifier(n_estimators=100, random_state=42, class_weight='balanced')

print("Training the Random Forest model...")
model.fit(X_train_bal, y_train_bal)
print("Model training complete.\n")

# --- 6. Make Predictions and Evaluate ---
y_pred = model.predict(X_test_scaled)
accuracy = accuracy_score(y_test, y_pred)

print(f"--- Model Evaluation ---")
print(f"Overall Accuracy: {accuracy:.4f}\n")
print("Classification Report:")
print(classification_report(y_test, y_pred))
```



```
Training set size after SMOTE: (3135, 24)
Training the Random Forest model...
Model training complete.

--- Model Evaluation ---
Overall Accuracy: 0.8317

Classification Report:
              precision    recall  f1-score   support

           1       0.94      0.89      0.92        19
           2       0.67      0.67      0.67        24
           3       0.59      0.65      0.62        49
           4       0.88      0.88      0.88       157
           5       0.95      0.90      0.92        60

    accuracy                           0.83       309
   macro avg       0.81      0.80      0.80       309
weighted avg       0.84      0.83      0.83       309
```

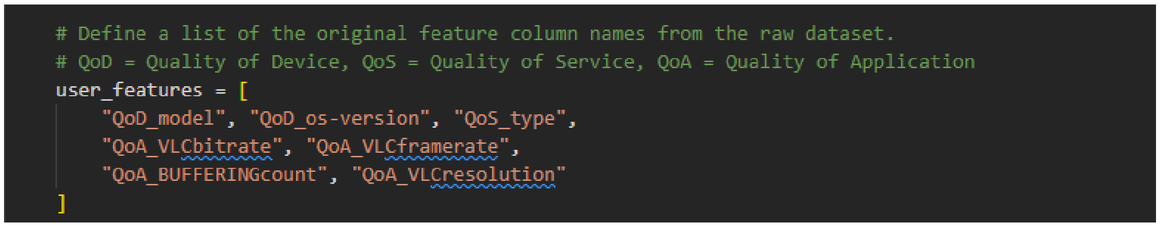


We successfully optimized our performance: the combination of the non-linear classifier and SMOTE boosted the final accuracy to 83%. This wraps up the initial model training and investigation phase precisely as planned.

The next phase of our analysis involves user profiling- a deeper understanding of distinct user types. This in my opinion is a crucial business-level evolution, we are moving from attempting to predict an individual instance of failure toward a more strategic goal of proactively **identifying larger groups ("Can we identify whole segments of at-risk users?")**.

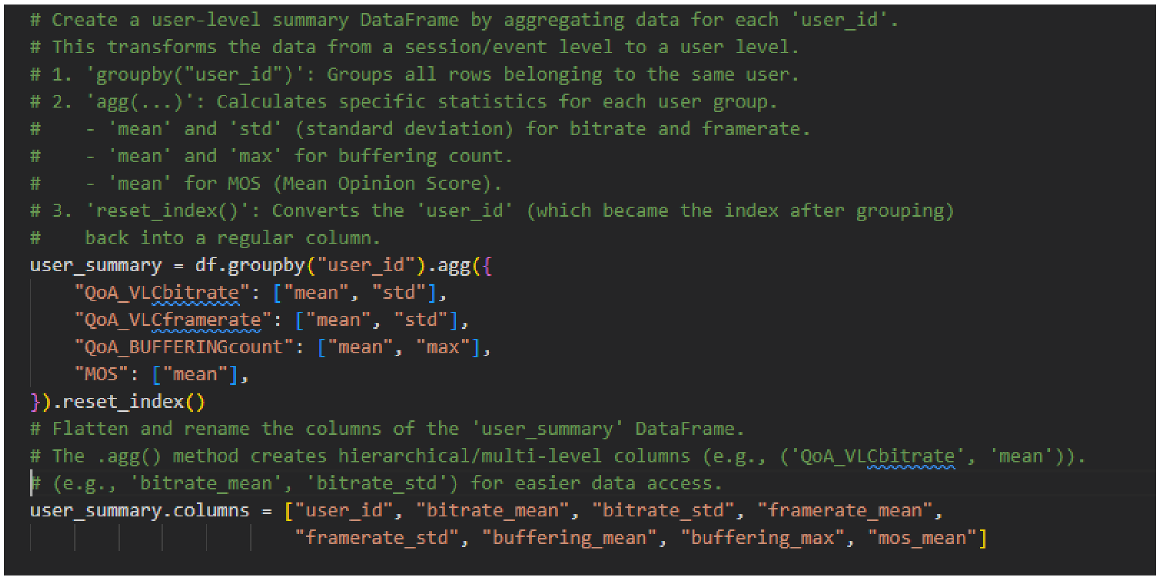
6. User-Centric Modeling

User behavior is diverse, making a single, global Quality of Experience (QoE) model insufficient. Variations include tolerance to **buffering**, consistency of device usage, and **network technology**. To address this, we propose a **user-centric QoE** model: instead of a single model, we cluster users based on shared behavioral, device, or network characteristics.



```
# Define a list of the original feature column names from the raw dataset.
# QoD = Quality of Device, QoS = Quality of Service, QoA = Quality of Application
user_features = [
    "QoD_model", "QoD_os-version", "QoS_type",
    "QoA_VLCbitrate", "QoA_VLCframerate",
    "QoA_BUFFERINGcount", "QoA_VLCresolution"
]
```

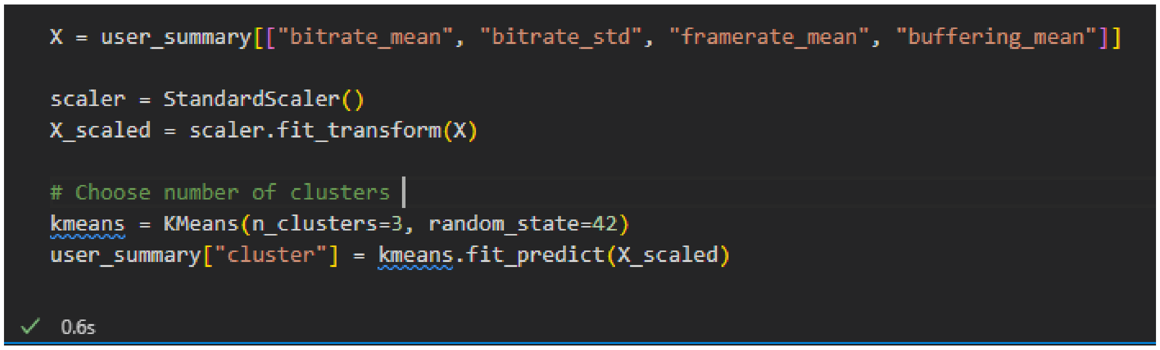
The current data is at the **session level**; each user_id has multiple entires. We have key features like **QoD** (QoD_model, QoD_os-version), **QoS** (QoS_type), and **QoA** (QoA_BUFFERINGcount, QoA_VLCbitrate). To create a **user-centric profile**, we will apply statistical aggregations across all sessions for each user to capture their **typical** or **average** performance.



```
# Create a user-level summary DataFrame by aggregating data for each 'user_id'.
# This transforms the data from a session/event level to a user level.
# 1. 'groupby("user_id")': Groups all rows belonging to the same user.
# 2. 'agg(...)': Calculates specific statistics for each user group.
#    - 'mean' and 'std' (standard deviation) for bitrate and framerate.
#    - 'mean' and 'max' for buffering count.
#    - 'mean' for MOS (Mean Opinion Score).
# 3. 'reset_index()': Converts the 'user_id' (which became the index after grouping)
#    back into a regular column.
user_summary = df.groupby("user_id").agg({
    "QoA_VLCbitrate": ["mean", "std"],
    "QoA_VLCframerate": ["mean", "std"],
    "QoA_BUFFERINGcount": ["mean", "max"],
    "MOS": ["mean"],
}).reset_index()
# Flatten and rename the columns of the 'user_summary' DataFrame.
# The .agg() method creates hierarchical/multi-level columns (e.g., ('QoA_VLCbitrate', 'mean')).
# (e.g., 'bitrate_mean', 'bitrate_std') for easier data access.
user_summary.columns = ["user_id", "bitrate_mean", "bitrate_std", "framerate_mean",
                        "framerate_std", "buffering_mean", "buffering_max", "mos_mean"]
```

I want to capture the essential aspects of users' video streaming experiences. My goal is to summarize each user's typical viewing session with a few interpretable metrics that reflect both quality and consistency. Through ***bitrate_mean*** and ***bitrate_std***, I account for the average video quality delivered and its variability-two factors that I believe are crucial for understanding perceived viewing quality.

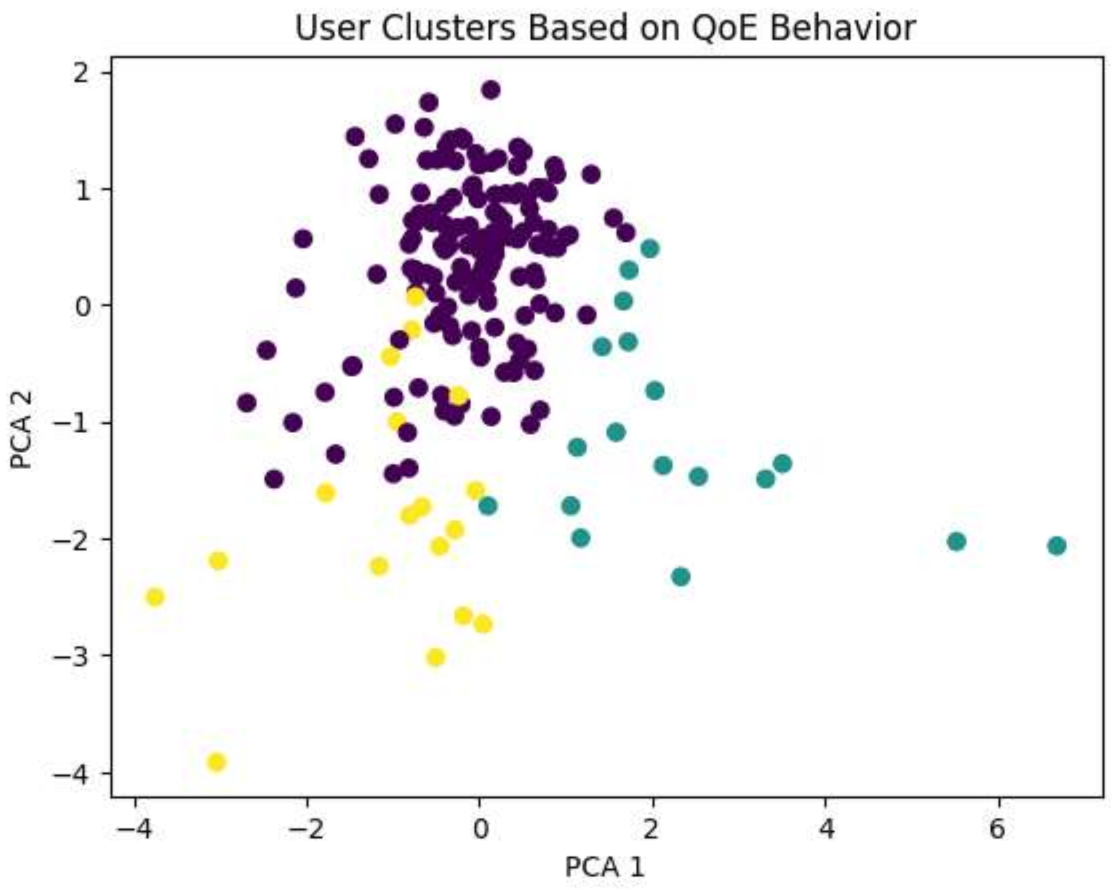
Framerate_mean let us measure how smooth the video typically looks for each user, and ***buffering_mean*** helps us identify how frequently users are facing interruptions or delays.



```
X = user_summary[["bitrate_mean", "bitrate_std", "framerate_mean", "buffering_mean"]]

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Choose number of clusters
kmeans = KMeans(n_clusters=3, random_state=42)
user_summary["cluster"] = kmeans.fit_predict(X_scaled)
```

✓ 0.6s

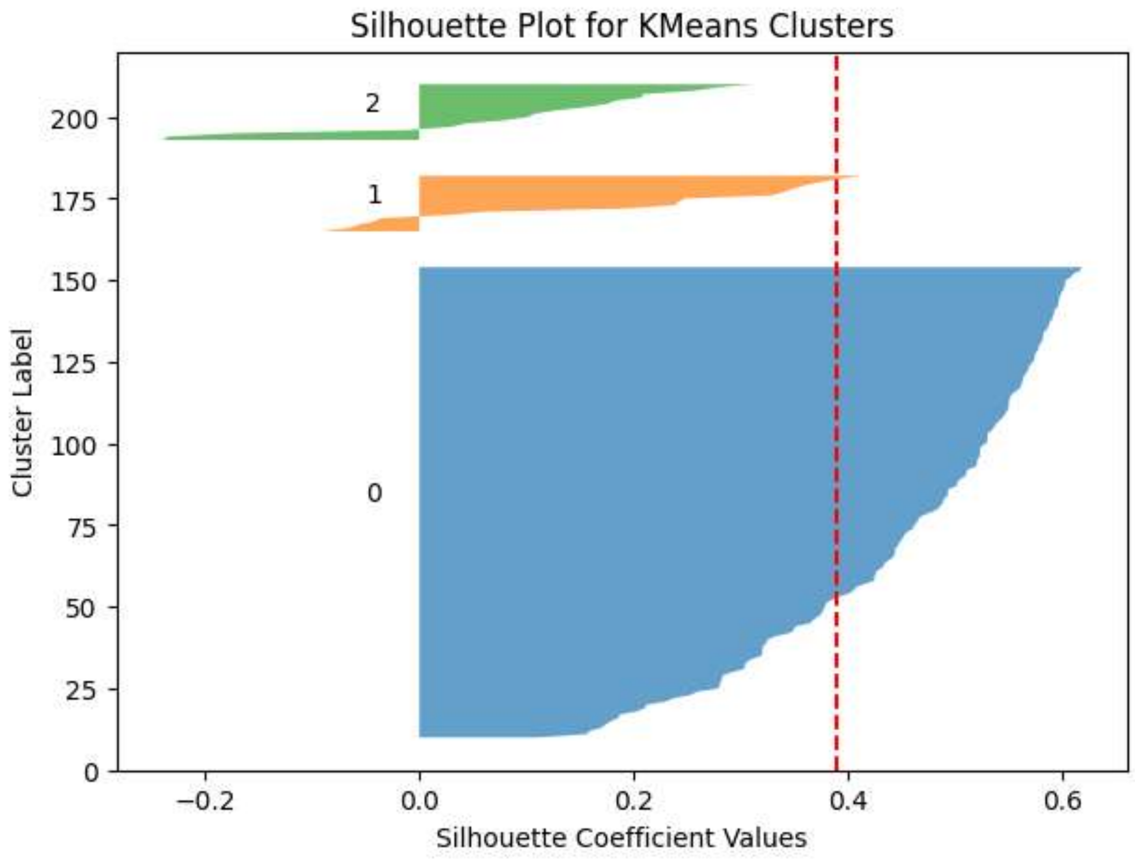
We applied the K-Means algorithm to our scaled data (X_scaled) to partition all users into 3 distinct lusters. To automatically discover natural groupings in the data. I hypothesized that our user base is not uniform; instead, it consists of different segments (high-performance users, low-quality-experience users, unstable connection users).

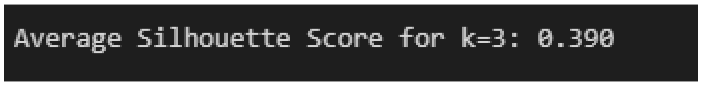


To gain meaningful insights from the high-dimensional Quality of Experience behavior data, we first employed ***Principal Component Analysis (PCA)*** for dimensionality reduction. This step allowed us to transform the original feature subset into a more manageable, **two-dimensional space**.

The color-coding in the plot represents the resulting three distinct user clusters:

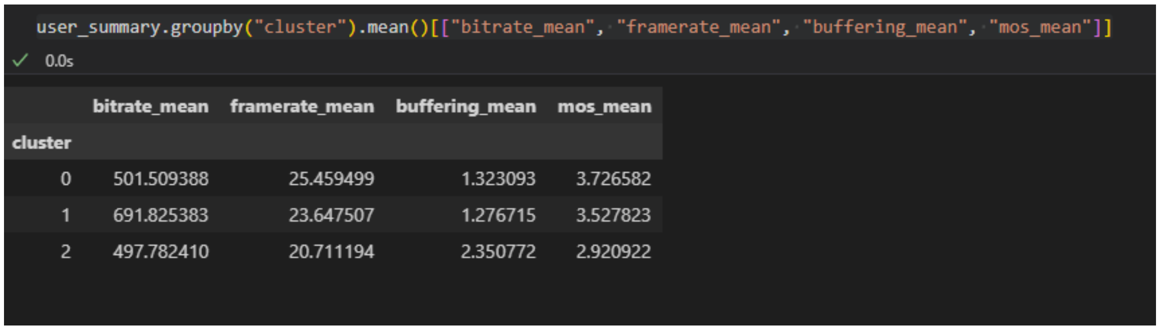
- **Cluster 1 (Dark Purple):** This is the largest group, tightly centered, representing the **"average" user behavior** in terms of QoE characteristics.
- **Cluster 2 (Yellow):** This group is characterized by a significant deviation along the negative axis of PCA 2, suggesting a shared behavior or experience that sets them apart from the main user base.
- **Cluster 3 (Green):** This cluster shows the highest positive values on the PCA 1 axis, indicating a fundamentally different QoE profile that warrants further investigation, potentially highlighting a group facing unique network conditions.





```
Average Silhouette Score for k=3: 0.390
```

Our Average Silhouette Score of **0.390** indicates that the three clusters we found are **not perfectly distinct** or highly cohesive as shown in the figures above.



```
user_summary.groupby("cluster").mean()[["bitrate_mean", "framerate_mean", "buffering_mean", "mos_mean"]]
✓ 0.0s

         bitrate_mean  framerate_mean  buffering_mean  mos_mean
cluster
      0    501.509388       25.459499        1.323093  3.726582
      1    691.825383       23.647507        1.276715  3.527823
      2    497.782410       20.711194        2.350772  2.920922
```

Key Observations and Cluster Profile Analysis:

- **Cluster 1 (Best Quality, High Bitrate):** Has the highest bitrate (691.83), lowest buffering (1.28), and a high MOS (3.53). This represents Premium Users or users with excellent network conditions.
- **Cluster 0 (Mid-Range/Standard):** Has a moderate bitrate (501.51), the highest framerate (25.46), low buffering (1.32), and the highest MOS (3.73). This is the Majority User Group experiencing good service.
- **Cluster 2 (Poor Experience, Low Quality):** Has the lowest framerate (20.71), the highest buffering (2.35), and the lowest MOS (2.92). This represents a Frustrated User Group with significant Quality of Service issues.

Cluster	Interpretation
0	Users with **moderate bitrate**, **stable framerate**, and **few buffering events** → **Good QoE**
1	Users with **higher bitrate** but **less stable framerate** → network throughput is high, but video smoothness may vary
2	Users with **low framerate**, **more buffering**, and **low MOS** → **Poor QoE group**

7. Conclusion

Initial data preparation was conducted to clean the dataset, analyze key Quality of Experience (QoE) influence factors, and prepare the data for subsequent modeling phases. The subjective user satisfaction metric, the **Mean Opinion Score (MOS)**, was used as the ground truth for QoE. a **User-Centric Modeling** approach on the **Poqemon-QoE-Dataset** to reveal the heterogeneity within the video streaming user base.

Future Work

To fully capitalize on the insights gained from the user-centric clustering and to achieve superior prediction accuracy across all user segments, we propose the following critical next step:

Build Separate Models per Cluster

The core recommendation for future work is to transition from a single, global QoE prediction model to **dedicated prediction models for each identified user cluster**. The significant variations in user experience (MOS) and underlying technical factors (bitrate, buffering) among Cluster 0, Cluster 1, and Cluster 2 strongly indicate that a model trained on the data of the **Frustrated User Group (Cluster 2)** will learn fundamentally different feature weights and thresholds than a model trained on the **Premium User Group (Cluster 1)**.

By training three distinct prediction models—one for each segment—we anticipate a substantial increase in the prediction accuracy and utility for each specific user segment. This approach will allow network operators to develop more precise, cluster-specific optimization and resource allocation strategies, ultimately leading to a more consistent and optimized QoE for the entire user base.