



Télécom SudPARIS
VAP DANI

Data Science and Network Intelligence

Report

Quality of experience in video streaming

for the exam

Data Science - Theory to practice

Professor

Alessandro Maddaloni

Student

Mahmoud MOHAMED

17 December 2024

Academic Year 2024-2025

Index:

1. **Introduction**
2. **Data Exploration**
3. **Feature Relationships**
4. **Preprocessing Steps**
5. **Model Building and Evaluation**
6. **Model Comparison**
7. **Conclusion**
8. **Project Resources**

1. Introduction

- What is the dataset about?

The dataset originates from the PoQeMoN project, a Platform for Quality Evaluation of Mobile Networks. It has been carefully designed to assess the Quality of Experience for users interacting with YouTube videos in various mobile environments. This extensive dataset includes a broad range of Quality of Experience Influencing Factors, along with subjective Mean Opinion Scores gathered through specialized experiments. The columns are organized as follows:

| category | details of the element |
|--|---|
| Video Parameters (QoA) | Columns from 3 to 10 represent various aspects related to video parameters in the VLC media player. |
| Network Information (QoS) | Columns 11 and 12 pertain to network information |
| Device Characteristics (QoD) | Columns 13 to 15 describe the characteristics of the device, including the model, operating system version, and API level. |
| User Profile (QoU) | Columns 16 to 18 provide information about the user, including gender, age, and level of study. |
| Comments from Users (QoF) | Columns 19 to 22 capture user feedback or comments about the quality of experience, including aspects related to audio and video. |
| Average Opinion Assigned by Testers (MOS) | Column 23 represents the Mean Opinion Score (MOS), which is the average subjective rating given by testers at the end of each video view. |

The columns specific to Quality of Experience (QoE) are grouped into several categories. In fact, this category encompasses the following distinct subcategories:

- QoA - MOS - QoF

- What is the goal of this project?

The project aims to study and implement predictive models that leverage the dataset's various features to predict the Mean Opinion Score (MOS), which reflects users' subjective assessments of their quality of experience while watching videos on mobile devices. To achieve this, I employed the Cross-Industry Standard Process for Data Mining (CRISP-DM) methodology:

- First, I focused on understanding the data and visualizing it.
- Next, I proceeded with the data preparation phase to identify the key features required to predict the MOS value.
- Following that, I moved to the modeling stage, where I applied several models and ultimately selected the most appropriate one.
- Finally, I conducted an evaluation of the models and provided an overall critique of the project.

2. Data Exploration

- Explanation of data details

The dataset consists of 23 columns and 1,543 rows. The columns are categorized into QoE Influence Factors, network information, device characteristics, user profiles, and user feedback. Each column is associated with a specific explanation and value type. The table below provides a detailed breakdown of the information:

| Category | Column | Definition | Possible Values |
|----------|-------------------|---|-----------------|
| - | id | Identifier for each record in the dataset. | - |
| - | user_id | Identifier for the user associated with each record. | - |
| QoA | QoA_VLCresolution | Video resolution parameter from the VLC media player. | (360,240,16) |
| | QoA_VLCbitrate | Video bitrate parameter from the VLC media player. | - |
| | QoA_VLCframerate | Video framerate parameter from the VLC media player. | - |
| | QoA_VLCdropped | Droppedframes parameterfrom the VLC media player. | - |
| | QoA_VLCaudiorate | Audio rate parameter from the VLC media player. | - |

| | | | |
|------------|--------------------|--|--|
| | QoA_VLCAudioloss | Audio loss parameter from the VLC media player. | - |
| | QoA_BUFFERINGcount | Count of buffering events during video playback. | - |
| | QoA_BUFFERINGtime | Total buffering time during video playback. | - |
| QoS | QoS_type | Type of network used for the assessment. | 1: EDGE, 2: UMTS, 3:HSPA, 4: HSPAP, 5: LTE |
| | QoS_operator | Network operator in France for the assessment. | 1: SFR, 2: BOUYEGUES, 3:ORANGE, 4: FREE |
| QoD | QoD_model | Device model characteristics for the assessment. | - |
| | QoD_os-version | Operating system version of the device. | - |
| | QoD_api-level | API level of the device. | - |
| QoU | QoU_sex | Gender of the user | 0: Woman, 1: Man |
| | QoU_age | Age of the user. | - |
| | QoU_study | Level of study of the user | 5: University, 4: Secondary school, 3: College, 2: Primary school, 1: Other |
| QoF | QoF_begin_time | Feedback on the time of the video session. | 1,2,3,4,5 |
| | QoF_shift | Feedback related to any shifts during the video session. | 1,2,3,4,5 |
| | QoF_audio | User feedback aspects. | on |
| | QoF_video | User feedback aspects. | on |
| - | +MOS | Mean Opinion Score given by the user at the end of each video view | 1,2,3,4,5 |

- Data exploration through visualization:

A. Distribution of MOS classes

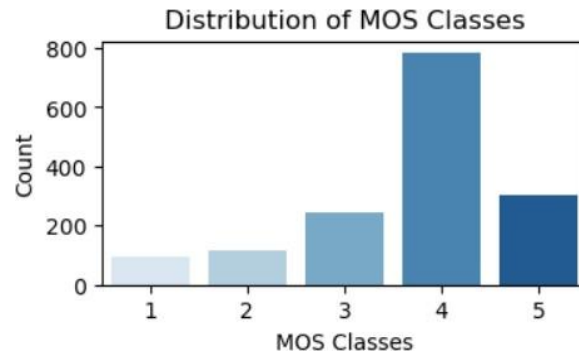


Figure 1 MoS distribution

Figure 1 highlights a notable concentration of positive ratings, predominantly in classes 4 and 5, which together comprise a significant total of 1,087 occurrences. This reflects a considerable proportion of users providing favorable feedback.

Interestingly, Figure 1 shows that the dataset maintains a relatively balanced distribution across all MOS (Mean Opinion Score) classes, without a strong skew towards any particular category. This balance is beneficial for machine learning models, as it ensures a diverse and comprehensive dataset, supporting more effective training and evaluation, and enhancing model performance and generalization.

B. Impact of Video Resolution on MOS :

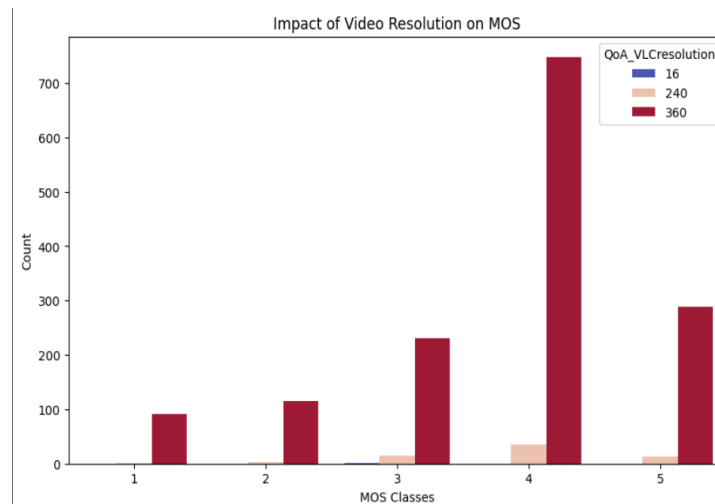


Figure 2 Impact of Video Resolution on MOS

In Figure 2, the histogram reveals a significant peak in MOS at 4, particularly for videos with a 360p resolution. This highlights a consistent trend where users commonly rate videos at this resolution as "Good," indicating a reliable and uniform user experience in this category.

C. Impact of the operator on MOS :

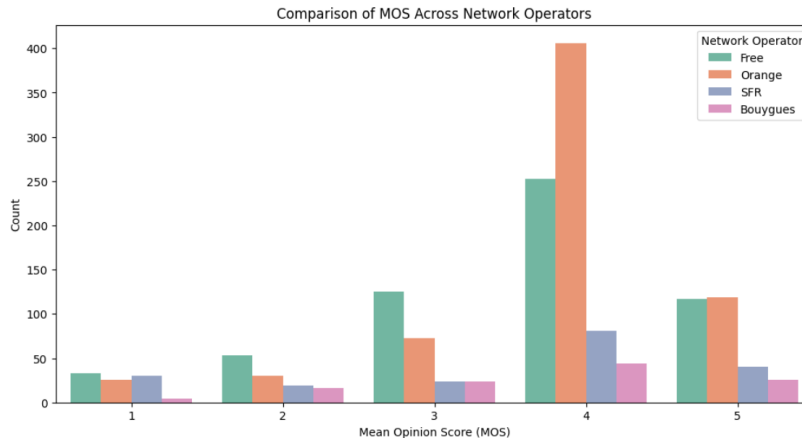


Figure 3 Impact of the operator on MOS

In Figure 3, it is clear that Free and Orange achieve the highest MOS values. Notably, their percentages are significantly higher for higher MOS ratings, with Orange showing an especially strong performance. Moreover, they consistently maintain a greater representation across all MOS values, suggesting an overall superior quality of experience. For MOS=4, Orange stands out prominently with a markedly higher percentage, highlighting its strong presence in this category.

D. Relationship between Network Type and MOS :

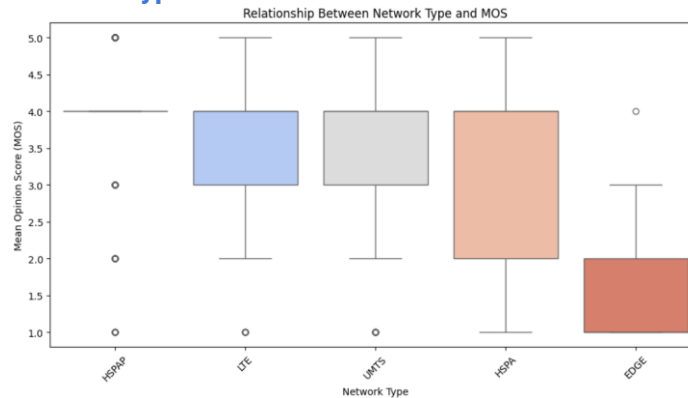


Figure 4 Relationship between Network Type and MOS

In Figure 4, the box plots illustrating MOS across different network types provide valuable insights into the consistency and quality of user experiences. The observed patterns lead to the following conclusions:

HSPAP Network :

- The HSPAP network exhibits consistently high MOS values, indicating a superior and stable user experience.
- There are a few outliers with lower scores, but overall, this network type delivers a highly satisfactory streaming quality.

LTE Network:

- Users of LTE networks experience MOS ratings in the range of moderate to high satisfaction.
- The variability is limited, suggesting consistent and generally good-quality experiences across users.

UMTS Network:

- The UMTS network has similar MOS ratings to LTE, with values in the moderate to good range.
- It demonstrates consistent performance, offering an acceptable level of user experience for most users.

HSPA Network:

- The HSPA network shows a wider range of MOS scores, indicating variability in user experience.
- This variability suggests that while some users might have positive experiences, others encounter issues leading to dissatisfaction.

EDGE Network :

- EDGE networks consistently receive the lowest MOS ratings, reflecting poor user satisfaction.
- The limited range and low scores highlight challenges with network performance, such as slow speeds or instability, which negatively impact the streaming experience.

3. Feature Relationships

Matrix of correlation with MOS:

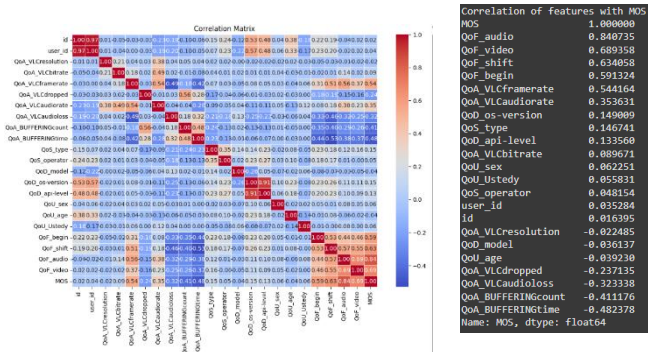


Figure 5 Matrix of correlation with MOS

In Figure 5, the correlation matrix with MOS (Mean Opinion Score) reveals that the features **QoF_audio**, **QoF_video**, **QoF_shift**, **QoAVLCframerate**, and **QoF_begin** show the strongest correlations with the target variable, **MOS**. This suggests that these factors are likely key influences in users' subjective evaluation of their overall experience quality.

- QoF_audio** and **QoF_video** refer to the audio and video quality assessments. The stronger correlation between these two variables indicates that both audio and video quality are significant contributors to users' overall perception of the experience.
- QoF_shift** and **QoF_begin** also demonstrate notable correlations with MOS, suggesting that factors related to video transitions or initial stability may impact users' perceptions of quality.
- QoAVLCframerate** pertains to video quality parameters like frame rate. Its strong correlation with MOS indicates that the frame rate significantly influences the user experience, highlighting its critical role in shaping perceived video quality.

These findings underscore the importance of these variables in enhancing and evaluating the quality of the user experience when watching videos on mobile devices.

4. Preprocessing Steps

i. Check the structure of the dataset

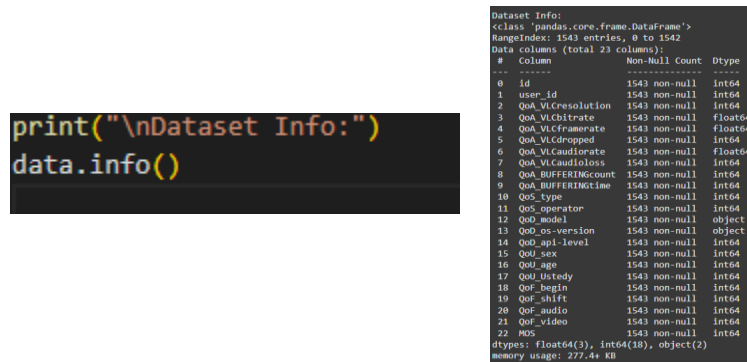


Figure 6 information about dataset

In figure 6 provides detailed information about the structure of a dataset (data) that contains 1543 entries and 23 columns.

ii. Check for missing values

```
print("\nMissing values per column:")
missing_values = data.isnull().sum()
print(missing_values)
```

```
Missing values per column:
id                0
user_id          0
QoA_VLCresolution 0
QoA_VLCbitrate    0
QoA_VLCframerate  0
QoA_VLCdropped    0
QoA_VLCaudiorate  0
QoA_VLCaudioloss  0
QoA_BUFFERINGcount 0
QoA_BUFFERINGtime 0
QoS_type          0
QoS_operator      0
QoD_model         0
QoD_os-version    0
QoD_api-level     0
QoD_sex           0
QoD_age           0
QoD_Ustedy        0
QoF_begin         0
QoF_shift         0
QoF_audio         0
QoF_video         0
MIS              0
dtype: int64
```

Figure 7 missing values

In Figure 7, our dataset does not contain any columns with missing values. As a result, the initial step of handling missing data is unnecessary, as there are no values to remove, generate, or impute.

iii. Check for duplicates

```
# Check for duplicates
duplicate_count = data.duplicated().sum()
print(f"\nNumber of duplicate rows: {duplicate_count}")

Number of duplicate rows: 0
```

Figure 8 duplicate data

In Figure 8, the analysis confirms that there is no duplicate data present in the dataset. This ensures the integrity of the data and eliminates the need for additional preprocessing steps to remove replicated entries, allowing us to proceed with the analysis confidently.

iv. Encode the Data

```
from sklearn.preprocessing import LabelEncoder
# encode categorical variables
data['QoD_model'] = LabelEncoder().fit_transform(data['QoD_model'])
data['QoD_os-version'] = LabelEncoder().fit_transform(data['QoD_os-version'])

print(data.dtypes)

id                int64
user_id          int64
QoA_VLCresolution int64
QoA_VLCbitrate    float64
QoA_VLCframerate  float64
QoA_VLCdropped    int64
QoA_VLCaudiorate  float64
QoA_VLCaudioloss  int64
QoA_BUFFERINGcount int64
QoA_BUFFERINGtime int64
QoS_type          int64
QoS_operator      int64
QoD_model         int64
QoD_os-version    int64
QoD_api-level     int64
QoD_sex           int64
QoD_age           int64
QoD_Ustedy        int64
QoF_begin         int64
QoF_shift         int64
QoF_audio         int64
QoF_video         int64
MIS              int64
dtype: object
```

Figure 9 encoded features

In Figure 9, the encoding process of categorical variables is illustrated. The LabelEncoder was applied to the **QoD_model** and **QoD_os-version** columns to transform them into numerical values. This conversion is a crucial preprocessing step that allows machine learning models to interpret and process these features effectively. By encoding these variables, we ensure the compatibility of the dataset with algorithms that require numerical input, thereby enhancing the analytical and predictive capabilities of the models.

v. Handling outliers :

```
print(data['QoA_VLCdropped'].describe())
```

| | |
|-------|-------------|
| count | 1543.000000 |
| mean | 1.217758 |
| std | 5.618366 |
| min | 0.000000 |
| 25% | 0.000000 |
| 50% | 0.000000 |
| 75% | 1.000000 |
| max | 107.000000 |

Name: QoA_VLCdropped, dtype: float64

```
print(data['QoA_BUFFERINGtime'].describe())
```

| | |
|-------|---------------|
| count | 1543.000000 |
| mean | 6164.184705 |
| std | 15032.225559 |
| min | 683.000000 |
| 25% | 2010.500000 |
| 50% | 2719.000000 |
| 75% | 4067.000000 |
| max | 329271.000000 |

Name: QoA_BUFFERINGtime, dtype: float64

Figure 10 columns with outliers

In Figure 10, the columns **QoA_VLCdropped** and **QoA_BUFFERINGtime** are highlighted as containing outliers. For the **QoA_VLCdropped** column, the disparity between the maximum value and the 75th percentile indicates the presence of extreme values. Similarly, in the **QoA_BUFFERINGtime** column, the substantial difference between the maximum value and the 75th percentile confirms the existence of outliers.

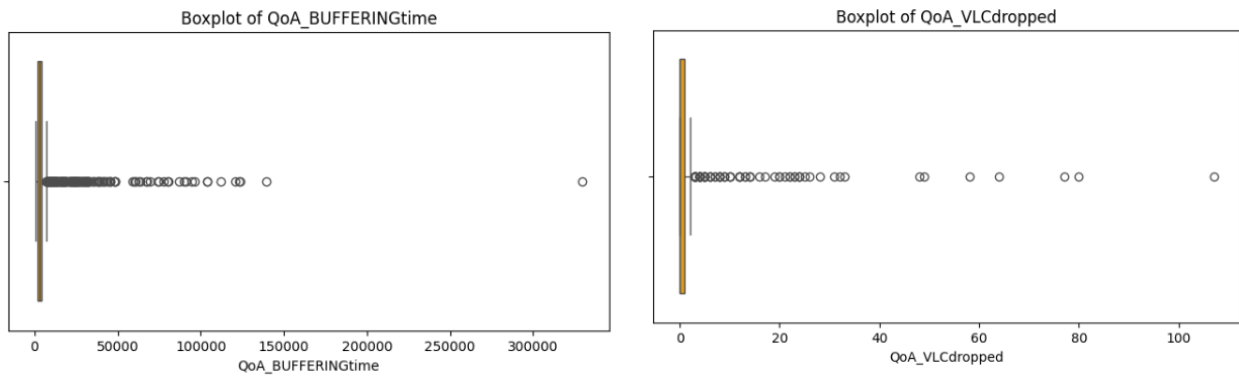


Figure 11 columns before removing the outliers

In Figures 12, 13, These outliers are addressed by setting an upper limit based on the 95th percentile and removing rows with values above this limit to ensure the dataset is more representative and balanced.

```
# Cap outliers using the 95th percentile
threshold = data['QoA_VLCdropped'].quantile(0.97)
data = data[data['QoA_VLCdropped'] < threshold]

print(data['QoA_VLCdropped'].describe())
```

| | |
|-------|-------------|
| count | 1495.000000 |
| mean | 0.480936 |
| std | 0.877812 |
| min | 0.000000 |
| 25% | 0.000000 |
| 50% | 0.000000 |
| 75% | 1.000000 |
| max | 6.000000 |

Name: QoA_VLCdropped, dtype: float64

```
# Cap outliers using the 95th percentile
threshold = data['QoA_BUFFERINGtime'].quantile(0.97)
data = data[data['QoA_BUFFERINGtime'] < threshold]

print(data['QoA_BUFFERINGtime'].describe())
```

| | |
|-------|--------------|
| count | 1450.000000 |
| mean | 3631.406207 |
| std | 3578.269790 |
| min | 683.000000 |
| 25% | 1972.250000 |
| 50% | 2625.500000 |
| 75% | 3724.750000 |
| max | 26877.000000 |

Name: QoA_BUFFERINGtime, dtype: float64

Figure 12 outlier Capping Columns Using the 95th Percentile

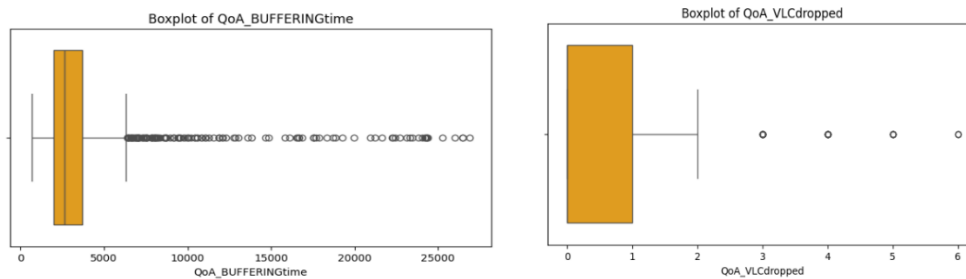


Figure 13 columns after removing the outliers

N.B.: The outliers will not be removed from the dataset to avoid negatively impacting the analysis, as the dataset is relatively small. This decision ensures that sufficient data is retained for meaningful insights. [project data science without.ipynb](#)

vi. [scaling](#) :

in Figure 14, I selected these five features for normalization because I noticed their value ranges and scales varied significantly, which could impact the analysis:

1. **QoA_VLCresolution:** I found that video resolutions differ greatly across videos, leading to a wide range of values. Normalizing this feature makes the data more consistent and easier to compare.
2. **QoA_VLCbitrate:** Since bitrates vary based on video quality and encoding, the values were quite inconsistent. By normalizing this feature, I aim to make them comparable across the dataset.
3. **QoA_VLCframerate:** I observed that frame rates depend on video types and encodings, which resulted in large disparities. Scaling this feature ensures uniformity in my analysis.
4. **QoA_BUFFERINGtime:** Buffering times fluctuated significantly, likely due to varying network conditions. Normalizing this feature helps reduce biases caused by these differences.
5. **QoA_VLCdropped:** Dropped frames varied widely depending on video playback conditions. Normalizing this feature ensures it contributes fairly to my analysis.

By normalizing these features, I aim to address scale differences and ensure the data is balanced and consistent, making my analysis more reliable and accurate.

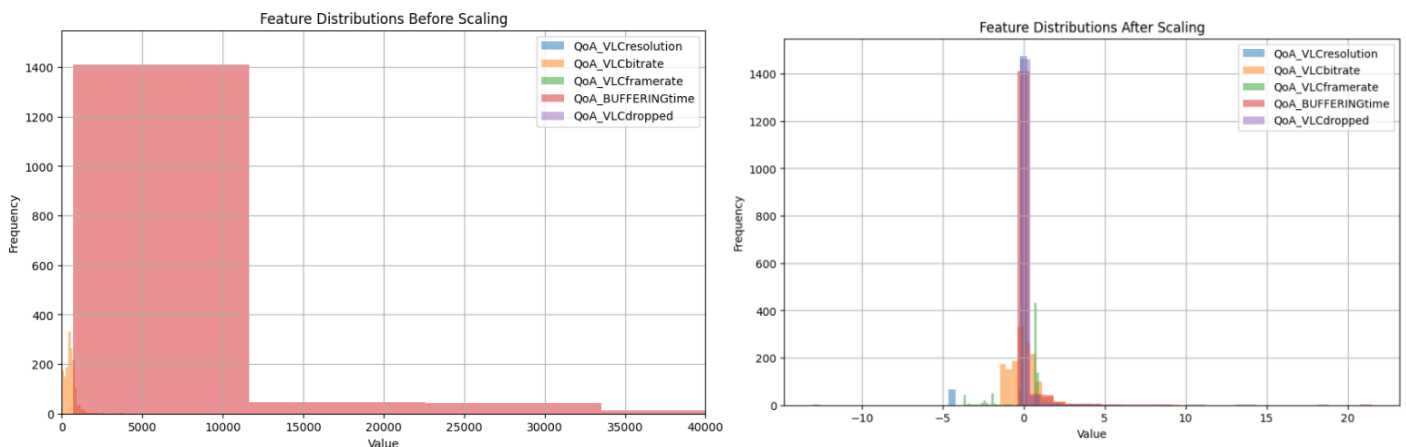


Figure 14 features befor and after scaling

[project data science with outlier.ipynb](#)

5. Model Building and Evaluation

i. [Dropping Irrelevant Features](#)

```
irrelevant_features = ['QoD_model', 'QoD_os-version', 'id', 'user_id', 'QoU_age', 'QoU_sex']
data = data.drop(columns=irrelevant_features)
```

Figure 15 Dropping irrelevant features based on correlation

In Figure 15, before selecting the features for analysis, I identified and dropped irrelevant features that do not contribute significantly to the predictive power of the model. These features included 'QoD_model', 'QoD_os-version', 'id', 'user_id', 'QoU_age', and 'QoU_sex'. These columns either contain redundant information, such as identifiers ('id', 'user_id'), or data that does not directly impact the analysis of the user's video quality experience, such as demographic features ('QoU_age', 'QoU_sex') or technical attributes of the device ('QoD_model', 'QoD_os-version'). By removing these irrelevant features, the dataset is streamlined, allowing for a more focused analysis on the factors that influence the **MOS** (Mean Opinion Score).

ii. Isolating Predictors and Target

```
from collections import Counter

# Handling imbalanced target variable (MOS)
X = data.drop(columns=['MOS'])
y = data['MOS']

# Check class distribution before applying SMOTE (optional)
print("Original class distribution:", Counter(y))

Original class distribution: Counter({4: 784, 5: 302, 3: 246, 2: 118, 1: 93})
```

Figure 16 Features and Target Variable

Features x: After removing irrelevant features all remaining columns are considered as features. These features capture factors that likely influence the user's quality of experience while watching videos on mobile devices.

Target y: The target variable is "MOS" (Mean Opinion Score), representing the user's overall quality of experience, which the model aims to predict.

iii. Splitting the dataset:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, stratify=y, random_state=42)
```

Figure 17 Split the dataset

In Figure 17, the dataset is split into training and test sets using the `train_test_split` function from scikit-learn. The training set, consisting of 80% of the data (`X_train` and `y_train`), is used to train the machine learning model. The test set, which makes up 20% of the data (`X_test` and `y_test`), remains unseen during training and is used to evaluate the model's performance on new, unseen data. The `stratify=y` parameter ensures that the target variable distribution is maintained in both sets, while the `random_state=42` ensures the split is reproducible.

iv. Increasing Training Data with SMOTE

```
smote = SMOTE(random_state=42)
X_train_resampled, y_train_resampled = smote.fit_resample(X_train, y_train)
```

```
Data split completed:
Training set size before SMOTE: 1234 rows
Training set size after SMOTE: 3135 rows
Test set size: 309 rows
```

Figure 18 Increasing Training Data with SMOTE

In Figure 18, the SMOTE (Synthetic Minority Over-sampling Technique) algorithm is used to address the class imbalance problem in the training data. The resampling process generates synthetic examples for the minority class, thereby increasing the amount of data available for model training. This approach helps balance the distribution of target classes and ensures that the model has enough examples to learn from each class. The resampled dataset is then used for model training, improving its ability to generalize to the minority class.

In this part of the report, we explore the training and evaluation of two machine learning models: **Random Forest** and **Gradient Boosting**. Both models are trained and evaluated using the resampled training data, and performance is assessed using cross-validation, as well as classification reports for both the training and test sets.

v. Random Forest Model

```
# Train Random Forest with cross-validation
rf_model = RandomForestClassifier(
    n_estimators=250,
    random_state=42,#42
    max_depth=5, #10
    min_samples_split=10,
    min_samples_leaf=5,
    class_weight='balanced'
)

# Perform cross-validation for Random Forest
rf_cv_scores = cross_val_score(
    rf_model,
    # X_train,
    # y_train,
    X_train_resampled,
    y_train_resampled,
    cv=cv,
    scoring='accuracy'
)

print("\nRandom Forest Cross-Validation Accuracy:")
print(f"Mean Accuracy: {rf_cv_scores.mean():.4f} (+/- {rf_cv_scores.std():.4f})")

# Train and evaluate Random Forest
rf_model.fit(X_train_resampled, y_train_resampled)#X_train, y_train)
rf_pred_train = rf_model.predict(
    X_train_resampled)
    #X_train)
rf_pred_test = rf_model.predict(X_test)
```

Figure 19 Random Forest Model

The Random Forest model is an ensemble learning method that uses multiple decision trees to make predictions. The model is configured with the following parameters:

- `n_estimators=250`: This defines the number of decision trees in the forest.
- `max_depth=5`: This limits the depth of each tree to 5 levels, preventing overfitting.
- `min_samples_split=10` and `min_samples_leaf=5`: These parameters control the minimum number of samples required to split a node or be a leaf, ensuring the trees are not too complex.
- `class_weight='balanced'`: This option adjusts the weights of classes to handle class imbalance, ensuring the model gives equal importance to underrepresented classes.

The model is then evaluated using cross-validation, where the data is split into multiple folds, and the model's accuracy is averaged across these folds. The cross-validation accuracy is printed, along with the mean accuracy and standard deviation. This ensures that the model's performance is stable and not influenced by a specific train-test split.

After cross-validation, the Random Forest model is trained on the full resampled training data (`X_train_resampled` and `y_train_resampled`). The model's predictions are evaluated both on the training set and test set, using the classification report and accuracy score. This helps assess how well the model generalizes to new data and its ability to classify each class correctly.

vi. Gradient Boosting Model

```
# Train Gradient Boosting with cross-validation
gb_model = GradientBoostingClassifier(
    n_estimators=200,          # Reduced number of estimators
    learning_rate=0.01,        # Slower learning rate
    max_depth=3,               # Simpler trees
    subsample=0.8,             # Use only 80% of data for each tree
    min_samples_split=5,       # Larger minimum split size#10
    min_samples_leaf=3,        # Larger leaf size#5
    random_state=42
)

# Perform cross-validation for Gradient Boosting
gb_cv_scores = cross_val_score(
    gb_model,
    # X_train,
    # y_train,
    X_train_resampled,
    y_train_resampled,
    cv=cv,
    scoring='accuracy'
)

print("\nGradient Boosting Cross-Validation Accuracy:")
print(f"Mean Accuracy: {gb_cv_scores.mean():.4f} (+/- {gb_cv_scores.std():.4f})")

# Train and evaluate Gradient Boosting
gb_model.fit(X_train_resampled, y_train_resampled)#X_train, y_train)
gb_pred_train = gb_model.predict(X_train_resampled)#X_train)
gb_pred_test = gb_model.predict(X_test)
```

Figure 20 Gradient Boosting Model

Gradient Boosting is another ensemble method, but it builds trees sequentially, with each tree correcting errors made by the previous one. The model is configured with the following parameters:

- `n_estimators=200`: This defines the number of trees to be built.
- `learning_rate=0.01`: This slows down the learning process, preventing overfitting and allowing for more precise adjustments.
- `max_depth=3`: Each tree is limited to a depth of 3 to avoid overfitting.
- `subsample=0.8`: 80% of the data is used for each tree, making the model more robust by introducing randomness.
- `min_samples_split=5` and `min_samples_leaf=3`: These parameters ensure that each tree is built on a reasonable number of samples, avoiding overly complex or shallow trees.

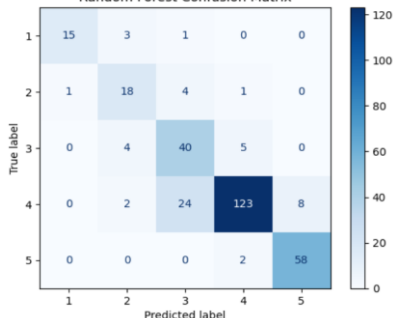
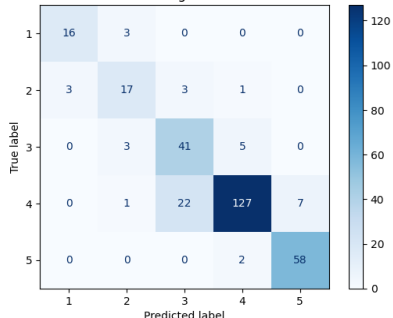
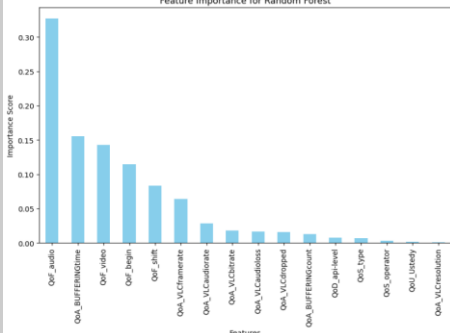
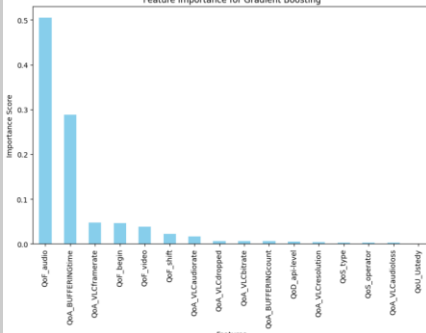
Just like the Random Forest model, Gradient Boosting is evaluated using cross-validation to check its stability and performance. The mean accuracy and standard deviation from the cross-validation process are printed.

After training, the Gradient Boosting model is also evaluated on both the training set and test set. The classification report and accuracy score provide insight into how well the model performs on both seen and unseen data.

6. Model Comparison

In this section, I compare the performance of the Random Forest and Gradient Boosting models based on their evaluation on the training and test sets.

| | Random Forest | Gradient Boosting | Explanation |
|------------------|--|--|--|
| Cross Validation | Random Forest Cross-Validation Accuracy: Mean Accuracy: 0.8313 (+/- 0.0290) | Gradient Boosting Cross-Validation Accuracy: Mean Accuracy: 0.8249 (+/- 0.0281) | Both models were evaluated using cross-validation, with Random Forest achieving a mean accuracy of 0.8313 (+/- 0.0290) and Gradient Boosting achieving a mean accuracy of 0.8249 (+/- 0.0281) . This cross-validation step provides additional assurance that both models perform consistently across different splits of the data, but Random Forest appears to have a slight edge in overall stability. |
| Training | Random Forest Training Set Classification Report: <pre> precision recall f1-score support 1 0.92 0.88 0.90 627 2 0.81 0.89 0.85 627 3 0.82 0.80 0.81 627 4 0.76 0.81 0.79 627 5 0.93 0.84 0.88 627 accuracy 0.85 macro avg 0.85 0.85 0.85 weighted avg 0.85 0.85 0.85 Training Accuracy Score (Random Forest): 0.8456 </pre> | Gradient Boosting Training Set Classification Report: <pre> precision recall f1-score support 1 0.90 0.94 0.92 627 2 0.81 0.87 0.84 627 3 0.82 0.73 0.78 627 4 0.76 0.82 0.79 627 5 0.92 0.85 0.88 627 accuracy 0.84 macro avg 0.84 0.84 0.84 weighted avg 0.84 0.84 0.84 Training Accuracy Score (Gradient Boosting): 0.8408 </pre> | Random Forest model was trained with 250 trees and a maximum depth of 5. On the training set, it achieved an accuracy of 0.85 , indicating that the model was able to learn the patterns in the data effectively. Gradient Boosting model, trained with 200 estimators and a learning rate of 0.01, demonstrated a similar trend. It achieved 0.84 accuracy on the training set, indicating strong performance during training. |
| Testing | Random Forest Test Set Classification Report: <pre> precision recall f1-score support 1 0.94 0.79 0.86 19 2 0.67 0.75 0.71 24 3 0.58 0.82 0.68 49 4 0.94 0.78 0.85 157 5 0.88 0.97 0.92 60 accuracy 0.82 macro avg 0.80 0.82 0.80 weighted avg 0.85 0.82 0.83 Test Accuracy Score (Random Forest): 0.8220 </pre> | Gradient Boosting Test Set Classification Report: <pre> precision recall f1-score support 1 0.84 0.84 0.84 19 2 0.71 0.71 0.71 24 3 0.62 0.84 0.71 49 4 0.94 0.81 0.87 157 5 0.89 0.97 0.93 60 accuracy 0.84 macro avg 0.80 0.83 0.81 weighted avg 0.86 0.84 0.84 Test Accuracy Score (Gradient Boosting): 0.8382 </pre> | Random Forest the accuracy dropped to 0.82 , which suggests some overfitting. Despite this, Random Forest performed well in handling class imbalance due to the <code>class_weight='balanced'</code> parameter, helping it give appropriate attention to the minority classes. Gradient Boosting it performed slightly better than Random Forest with an |

| | | | |
|--------------------|--|---|---|
| | | | <p>accuracy of 0.84. This suggests that Gradient Boosting might be better at generalizing, particularly on unseen data.</p> <p>However, it still exhibited some overfitting, albeit to a lesser degree than Random Forest.</p> |
| Confusion Matrix | <div><p>Random Forest Confusion Matrix</p><p>Gradient Boosting Confusion Matrix</p></div> | <p>-Gradient Boosting demonstrated better generalization, particularly for Class 3 and Class 4, which are larger classes in the dataset.</p> <p>-Random Forest exhibited competitive performance but struggled more with misclassifications in adjacent classes, especially for Class 4.</p> <p>-Both models handled the minority class (Class 1) relatively well, but Gradient Boosting provided slightly more consistent results overall.</p> | |
| Feature Importance | <div><p>Feature Importance for Random Forest</p><p>Feature Importance for Gradient Boosting</p></div> | <p>-Gradient Boosting focuses heavily on QoF_audio and QoA_BUFFERINGtime, making it more reliant on key features for predictions.</p> <p>-Random Forest distributes importance across a broader range of features, considering more factors in its decision-making.</p> <p>-Choose Gradient Boosting for precision with critical features and Random Forest for a more balanced approach.</p> | |

7. Conclusion

This project provides a foundational exploration into predicting the Quality of Experience (QoE) for mobile video streaming. The study successfully evaluated two models, **Random Forest** and **Gradient Boosting**, to predict Mean Opinion Scores (MOS). Among the two, **Gradient Boosting** demonstrated slightly better performance on the test set, showing stronger generalization to unseen data and better handling of majority classes, making it the preferred model for this application. However, **Random Forest** exhibited greater stability and distributed feature importance more broadly, making it a reliable alternative.

Despite these successes, several limitations were identified, including a restricted feature set, the subjective nature of MOS, small dataset size, and challenges in evaluating true model performance. These factors highlight areas for improvement.

Future enhancements could include expanding the dataset to include richer features, such as detailed network conditions or device performance metrics, which may further boost model accuracy. Exploring advanced techniques like deep learning could also uncover intricate patterns in user experience data. In conclusion, this study underscores the importance of QoE in mobile video streaming and demonstrates that **Gradient Boosting** is a promising model for this task while paving the way for further research to develop more accurate and robust predictive models.

8. Project Resources

All the code implementations and related files used in this project, including data preprocessing, model training, evaluation, and visualization, are available in the following link:

[Project Files and Code Repository](#)