# Data analysis techniques - individual project

# Carmel Gafa

## Some background - what we are talking about

The `Arousal–Valence Model` is a visual framework that organizes emotions based on arousal (ranging from calm to agitated) and valence (ranging from negative to positive). By pinpointing one's position within this matrix, individuals can better understand their emotional states, enhancing both emotional and bodily awareness. This tool is particularly beneficial for neurodivergent individuals who may face challenges in identifying and regulating emotions, as it bridges the gap between physical sensations and emotional recognition.

Visual aids like the `Arousal–Valence Model` are instrumental in improving emotional intelligence, especially for those with neurodivergent traits such as ADHD or autism. These individuals often experience heightened levels of alexithymia, making it difficult to label and process emotions. By utilizing visual tools, they can more effectively recognize patterns in their emotional experiences and identify triggers that lead to specific responses. This approach not only aids in emotional identification but also supports the development of coping strategies, ultimately fostering better emotional regulation and well-being.

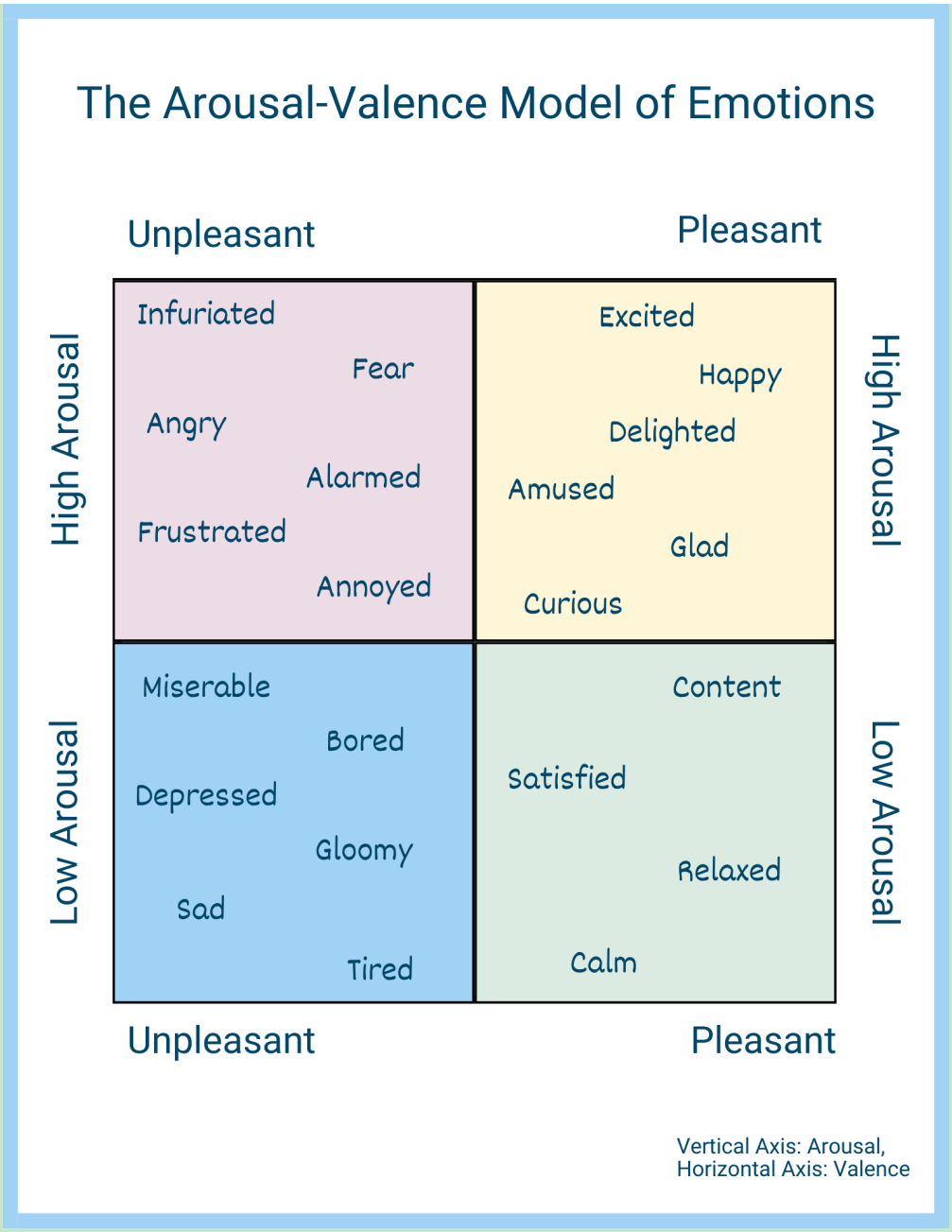



Image taken from [1]

## Mel frequency cepstral co-efficients

One of the most widely used features in speech processing is the Mel Frequency Cepstral Coefficients (MFCCs).

MFCCs represent the short-term power spectrum of an audio signal and are designed to mimic how humans perceive sound. The key idea is to capture the spectral envelope of speech — essentially how the vocal tract is shaped during speech production — which plays a crucial role in conveying emotion. The MFCC extraction pipeline is as follows:

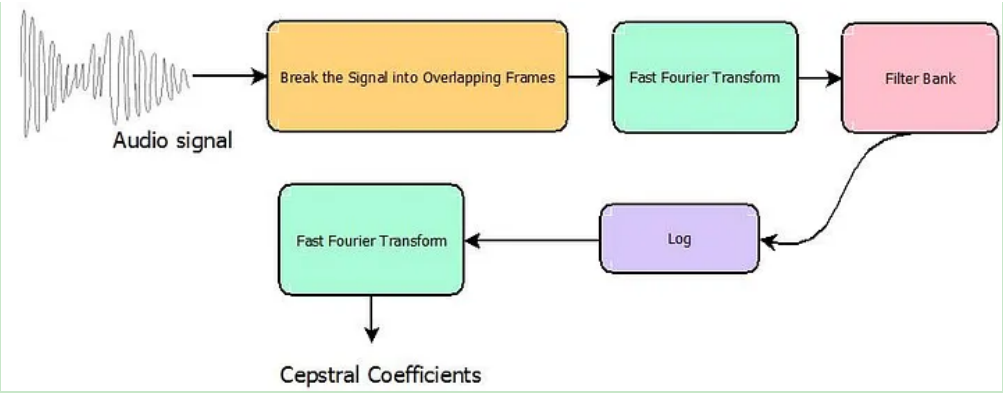



Image taken from [2]

We can see that the pipeline consists of a number of stages:

- Framing & Windowing: The audio signal is split into short overlapping frames (e.g. 25 ms) to capture stationarity.
- Fast Fourier Transform: Converts each frame from the time domain to the frequency domain.
- Mel Filter Bank: Applies a set of filters spaced according to the Mel scale, which better reflects human auditory perception.
- Log Energy: Emphasizes low-energy components and simulates how humans perceive loudness.
- Discrete Cosine Transform: Used to decorrelate and compress the filterbank energies into a small number of coefficients — the MFCCs.

MFCC are important because they

- Capture the timbre and articulation of speech — features that are sensitive to emotional variation.
- They are particularly informative when a speaker is:
  - Speaking fast or loud (high arousal)
  - Speaking in a soft or relaxed tone (low arousal)
  - Using a positive or negative tone (affecting valence)

MFCCs do not explicitly model prosodic features such as:

- Pitch (F0)
- Intensity (energy)
- Speech rate or rhythm

These features can also be strong indicators of emotion, so MFCCs are sometimes combined with other features such as pitch and intensity.

The dataset includes 130 acoustic features extracted using the openSMILE toolkit, commonly used for speech analysis. These features capture multiple sound characteristics of the speech signal, such as tone, intensity, and rhythm. The 130 features include:

- MFCCs
  - Static coefficients: `mfcc_sma[1]` to `mfcc_sma[14]`

    - Dynamic (delta) coefficients: `mfcc_sma_de[1]` to `mfcc_sma_de[14]`
- Spectral descriptors
    - Includes spectral centroid, roll-off, entropy, variance, skewness, and slope — describing tone and sharpness.
- Prosodic features:
    - Includes pitch (F0), jitter, shimmer, and voicing probability — related to intonation and vocal stability.
- Energy-based features:
    - Includes RMS energy, log harmonic-to-noise ratio (logHNR), and zero-crossing rate — reflecting loudness and clarity.
- Filterbank energies:
    - `audSpec_Rfilt_sma[0–25]` and their delta derivatives — capturing frequency-band-specific energy profiles.
- Psychoacoustic measures:
    - Includes sharpness, harmonicity, and auditory spectrum length — modeling how humans perceive sound.

Each feature ends with `_amean`, indicating it is the mean value computed over a 3-second analysis window, after being extracted frame-by-frame (~25 ms windows with 10 ms hop).

---

**Sources**

1. https://neurodivergentinsights.com/arousal-valence-model/
2. https://medium.com/prathena/the-dummys-guide-to-mfcc-aceab2450fd
3. https://stackoverflow.com/questions/13810873/mfcc-13-coefficients
4. https://opus.bibliothek.uni-augsburg.de/opus4/frontdoor/deliver/index/docId/77173/file/77173.pdf

In [3]:
```
# imports

import pandas as pd
import numpy as np
import os
import scipy.stats as stats
import matplotlib.pyplot as plt
import seaborn as sns
```

## Preamble - Loading the Data and Initial Inspection to Get Acquainted with the Data

The data is uploaded and inspected in this stage. The following steps are carried out:

- The data is loaded into a pandas dataframe.

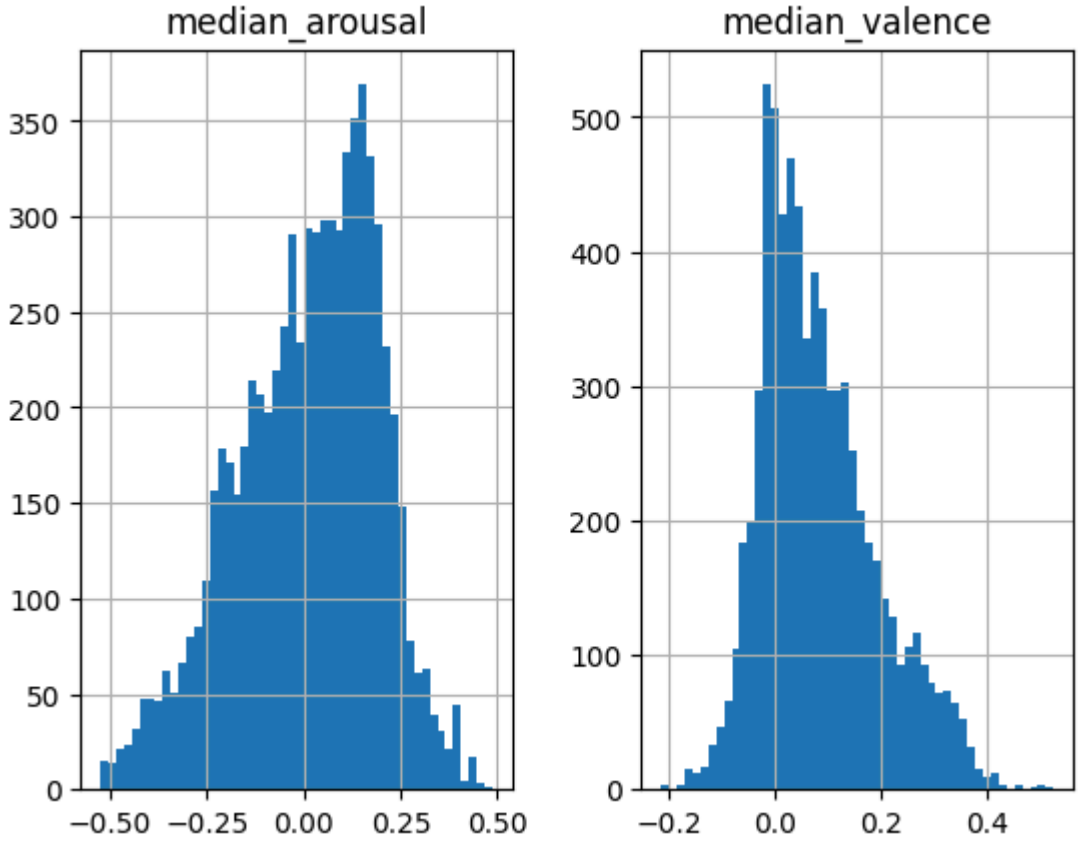
- A summary of the data is created and saved in a file. It includes:
  - Feature name
  - Non-null count. It was noted that **all records** contained 7238 non-null values.
  - Data type. It was observed that the `participant` feature is an integer, while all other features are floats. The response variables are also floats.
- An additional summary is created, showing the 10 features with the highest number of non-null values. This serves as a second check to ensure that no missing records are present in the dataset.

## An Initial Look at the Target Values

- The distributions of the two response variables, `median_arousal` and `median_valence`, are examined. It is noted that the distribution of `median_arousal` is skewed to the left, while the distribution of `median_valence` is skewed to the right.
- The descriptive statistics of the two response variables are analyzed. It is noted that for
  - `median_arousal`:
    - the mean is 0.010609,
    - the skewness is -0.424667, indicating that the distribution is skewed to the left,
    - the kurtosis is -0.260799, indicating that the distribution is slightly platykurtic, meaning it is flatter than a normal distribution.
  - `median_valence`:
    - the mean is 0.083961,
    - the skewness is 0.698025, indicating that the distribution is skewed to the right,
    - the kurtosis is 0.192967, indicating that the distribution is slightly leptokurtic, meaning it is more peaked than a normal distribution.
- A scatter plot of `median_arousal` vs `median_valence` is created to identify the relationship between the two predictors. It is observed that there is a somewhat positive trend between the predictors, although it may not be linear.

In [4]:
```
data_folder = 'data/'
data_filename = 'project_data.csv'

data_path = os.path.join(data_folder, data_filename)
df_raw = pd.read_csv(data_path, sep=',', header=0, encoding='utf-8')

# print(df_raw.head())


df_summary = pd.DataFrame({
    'Feature': df_raw.columns,
    'Non-Null Count': df_raw.notnull().sum().values,
    'Data Type': df_raw.dtypes.values
})
```

```
# just to be certain about non nulls
df_raw.isnull().sum().sort_values(ascending=False).head(10)


### An initial look at the target values
### ------------------------------------

df_raw[['median_arousal', 'median_valence']].hist(bins=50)
plt.show()

print('desccriptive statistics of arousal and valence:')
print('-----------------------------------------------')
print(df_raw[['median_arousal', 'median_valence']].describe())
print('skewness of arousal and valence:')
print('--------------------------------')
print(df_raw[['median_arousal', 'median_valence']].skew())
print('kurtosis of arousal and valence:')
print('--------------------------------')
print(df_raw[['median_arousal', 'median_valence']].kurt())


### Histograms of predictors

for col in ['median_arousal', 'median_valence']:
    sns.histplot(df_raw[col], kde=True)
    plt.title(f'Histogram + KDE: {col}')
    plt.show()

### relationships between predictors


plt.figure(figsize=(6, 6))
plt.scatter(df_raw['median_arousal'], df_raw['median_valence'], alpha=0.5
plt.xlabel('Median Arousal')
plt.ylabel('Median Valence')
plt.title('Scatter Plot: Arousal vs Valence')
plt.grid(True)
plt.show()
```




desccriptive statistics of arousal and valence:

| | median_arousal | median_valence |
|---|---|---|
| count | 7238.000000 | 7238.000000 |
| mean | 0.010609 | 0.083961 |
| std | 0.182776 | 0.109831 |
| min | −0.526533 | −0.214067 |
| 25% | −0.115183 | 0.001067 |
| 50% | 0.034000 | 0.063900 |
| 75% | 0.149317 | 0.147383 |
| max | 0.487000 | 0.526133 |

skewness of arousal and valence:

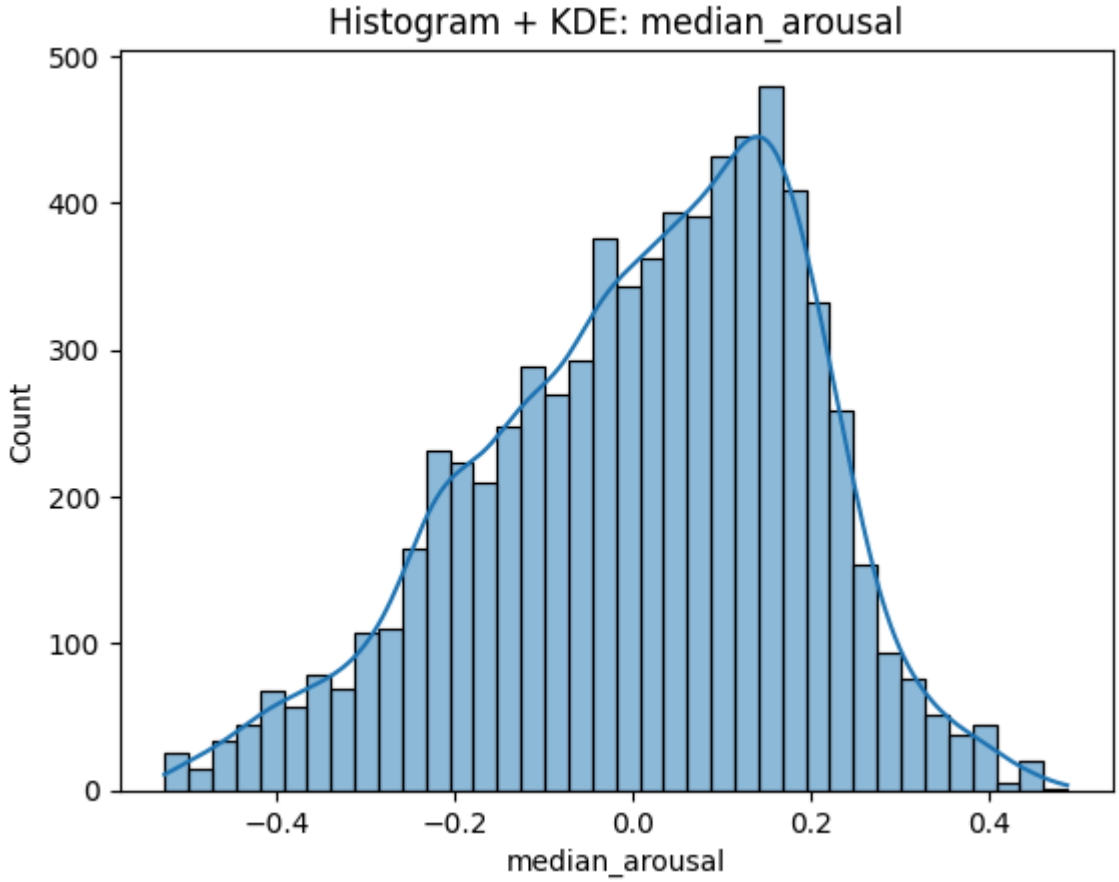
| | |
|---|---|
| median_arousal | −0.424667 |
| median_valence | 0.698025 |

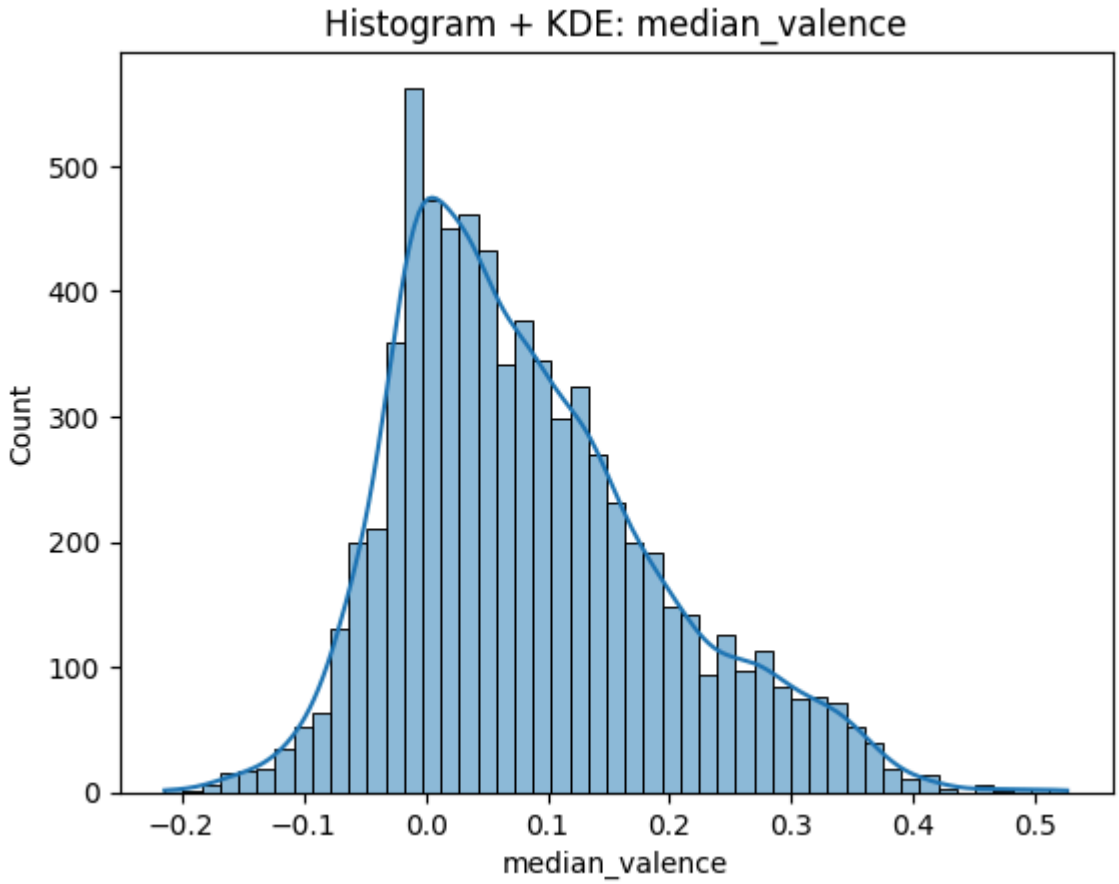
dtype: float64

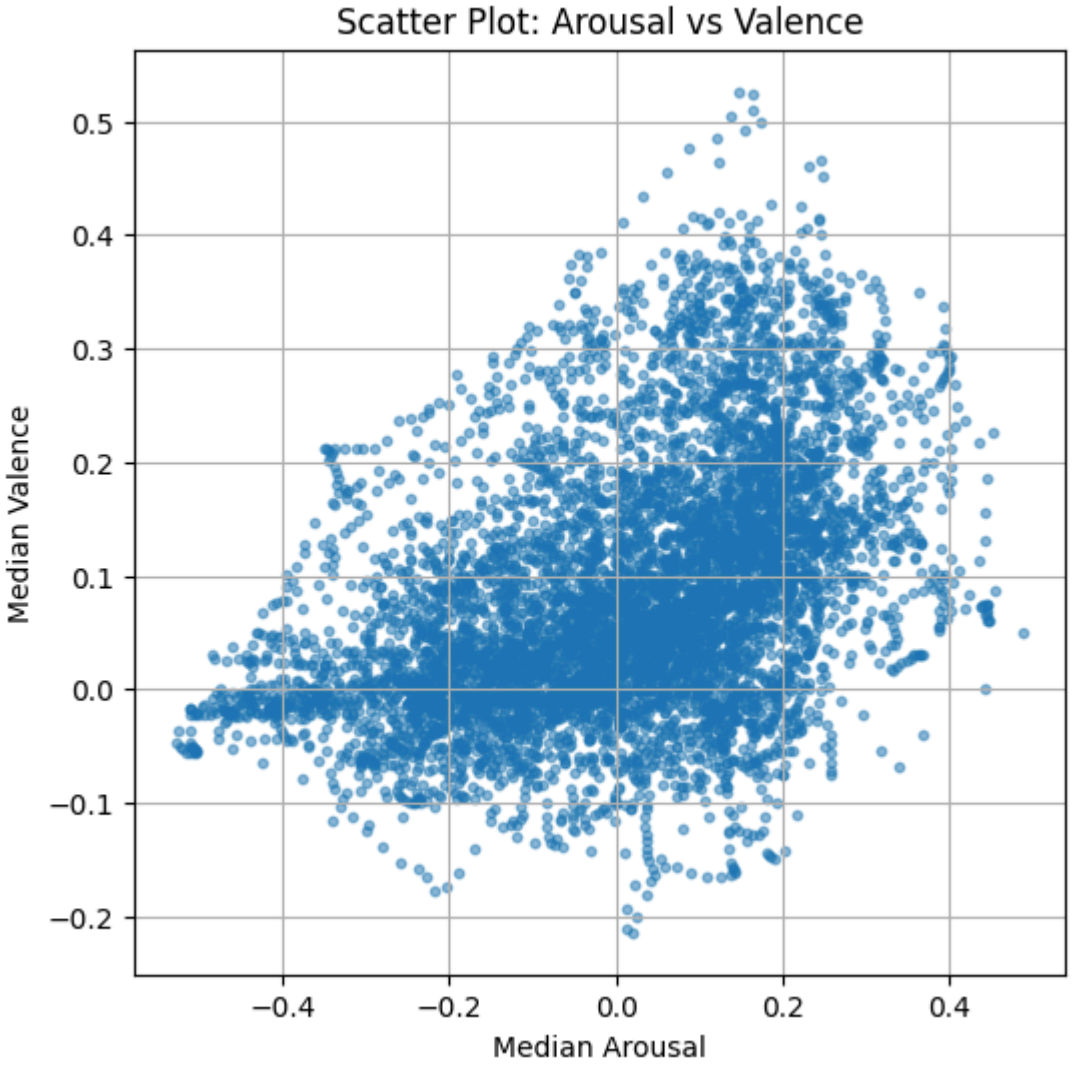
kurtosis of arousal and valence:

| | |
|---|---|
| median_arousal | −0.260799 |
| median_valence | 0.192967 |

dtype: float64


Histogram + KDE: median_arousal
Count
median_arousal
500
400
300
200
100
0
−0.4
−0.2
0.0
0.2
0.4


Histogram + KDE: median_valence
Count
median_valence
500
400
300
200
100
0
−0.2
−0.1
0.0
0.1
0.2
0.3
0.4
0.5




## Task 1

**If you were asked to build a model for predicting arousal and valence, using the provided audio features as explanatory variables:**

### Which performance metrics would you use to evaluate your model's predictions?

As the labels in this exercise, `median_arousal` and `median_valence` are present and are continuous variables, we are dealing with a supervised regression task.

In these cases, the cost function, which aggregates the individual loss functions across the dataset, can be used as it provides a single scalar value representing the model's overall performance on all training samples. It can be calculated as

$$\frac{1}{m}\sum_{i=1}^{m}\mathcal{L}(\hat{y}, y)$$

There are many different cost functions, but in this exercise, the Mean Square Error will be used;

$$MSE = \frac{\sum_{i=1}^{n}(\hat{y}^{(i)} - y^{(i)})^2}{n}$$

Mean square error (MSE) diminishes the effect of negligible residuals and amplifies the larger residuals. MSE is, however, sensitive to outliers, so for this reason, the outliers' situation is first very briefly evaluated using the following code that ranks the number of outliers per feature:

In [5]:
```
from scipy.stats import zscore
z_scores = df_raw.drop(columns=['Participant', 'median_arousal', 'median_
outlier_counts = (z_scores.abs() > 3).sum().sort_values(ascending=False)
print(outlier_counts)
```

```
ComParE13_LLD_25Hz_audSpec_Rfilt_sma_de[25]_amean          198
ComParE13_LLD_25Hz_audSpec_Rfilt_sma_de[24]_amean          194
ComParE13_LLD_25Hz_audSpec_Rfilt_sma_de[22]_amean          189
ComParE13_LLD_25Hz_audSpec_Rfilt_sma_de[23]_amean          186
ComParE13_LLD_25Hz_pcm_Mag_spectralSlope_sma_de_amean      184
                                                          ...
ComParE13_LLD_25Hz_mfcc_sma[12]_amean                       20
ComParE13_LLD_25Hz_mfcc_sma[14]_amean                       20
ComParE13_LLD_25Hz_mfcc_sma_de[1]_amean                     19
ComParE13_LLD_25Hz_audspecRasta_lengthL1norm_sma_amean      17
ComParE13_LLD_25Hz_mfcc_sma[2]_amean                        10
Length: 130, dtype: int64
```

As the proportion of outliers is modest, it is acceptable to proceed with MSE as a primary performance metric. However there are additional nuisances related to outliers, such as the skewness of the distributions of the features. This factor is not considered at this stage, but, if MAE will give inconsistent results because of the outliers an alternative metric like Mean Absolute Error will be used.

$$MAE = \frac{\sum_{i=1}^{n}|\hat{y}_i - y_i|}{n}$$

Mean absolute error is often used where the data being considered contains outliers as it is less sensitive to large deviations when compared to other cost functions like the mean square error.

An additional metric that will be used in this case is the Coefficient of Determination ($R^2$) that measures the proportion of variance in $y$ explained by the model. This metric ranges from $-\infty$ to 1.

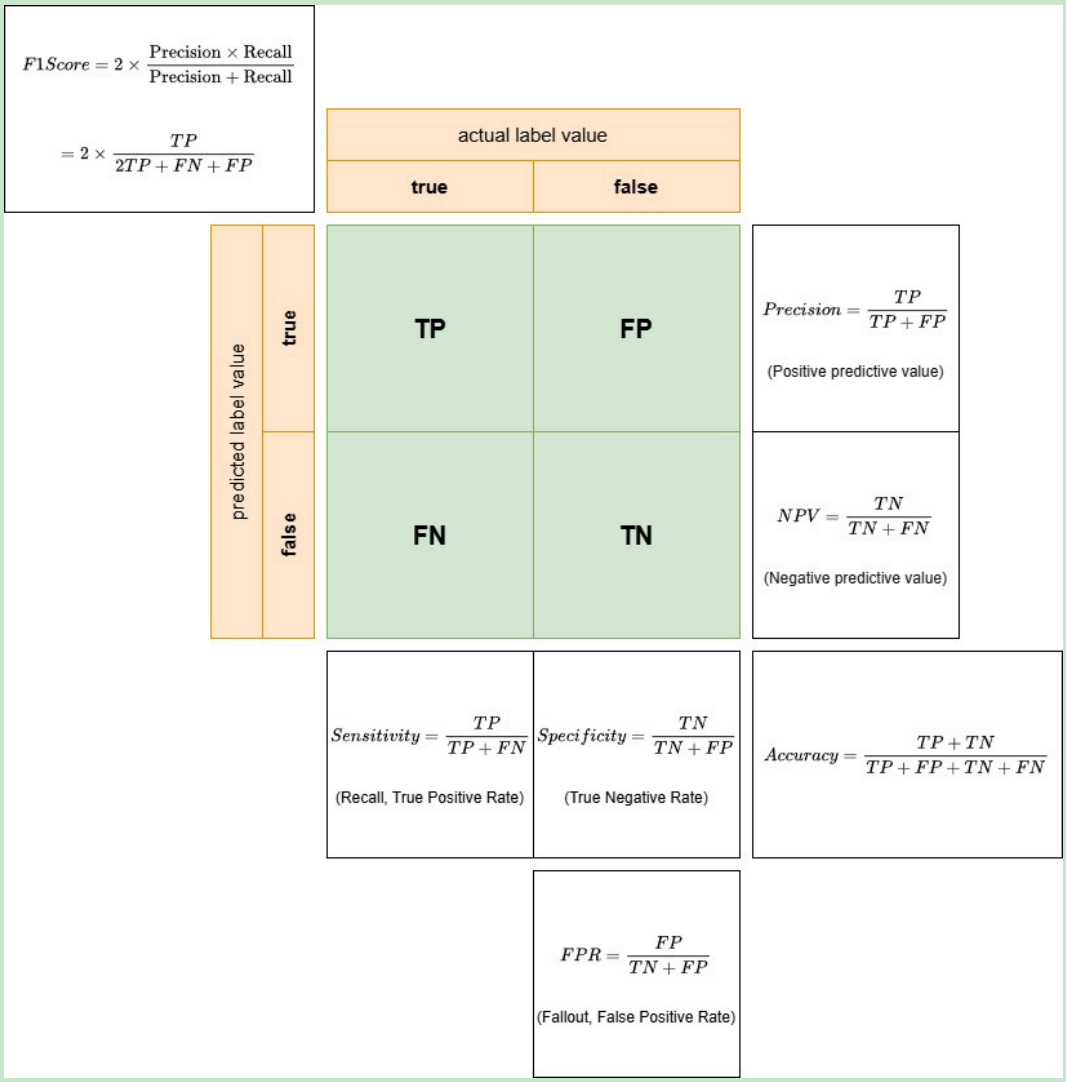
$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$$

- $R^2 = 1$: perfect fit
- $R^2 = 0$: model does no better than mean
- $R^2 < 0$: model is worse than constant predictor

We can apply a system of threshold values to change the system to a classification one, with the simplest case involving a single threshold that will split the label into a high or low, true or false, etc., where such a system is referred to as a binary classifier. In this case, a different system of performance metrics is considered, where the fundamental consideration is the confusion matrix, a grid that identifies the elements that are correctly and incorrectly classified:

| Prediction | Label Value | Outcome | Description |
|---|---|---|---|
| 1 | 1 | True Positive (TP) | The model correctly predicted the positive class. |
| 1 | 0 | False Positive (FP) | The model incorrectly predicted the positive class (a 'false alarm'). |
| 0 | 1 | False Negative (FN) | The model incorrectly predicted the negative class (a 'miss'). |
| 0 | 0 | True Negative (TN) | The model correctly predicted the negative class. |

The image below illustrates a typical confusion matrix and the metrics that emerge from it.




The performance metrics that will be used in this scenario include:

**Precision** is a robust metric used in cases where false positives are a concern, particularly in imbalanced datasets. It focuses on the proportion of True Positives (TP) out of all the samples that were predicted as positive (i.e., both True Positives and False Positives).

$$Precision = \frac{TP}{TP + FP}$$

Precision answers the question: 'Out of all the observations predicted to be positive, how many were really positive?' In other words, it measures the model's ability to avoid false positives.

**Recall** focuses on the positive class and measures how well the model captures all positive instances, regardless of how many false positives it makes. It is imperative when missing positive cases (false negatives) is costly, such as in medical diagnoses (e.g., cancer detection) or fraud detection.

$$Recall = \frac{TP}{TP + FN}$$

Recall answers the question: 'Out of all the actual positive instances, how many did the model correctly identify as positive?'

**The F1-score** is the harmonic mean of precision and recall. It provides a single metric that balances precision and recall, which is useful when there is an uneven class distribution or when false positives and false negatives are essential.

The formula for the F1 score is:

$$F1 = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{1}$$

$$\tag{2}$$

$$= 2 \times \frac{TP}{2TP + FN + FP} \tag{3}$$

## Does the selection of the performance metric depend on the type of the response variables? Explain your reasoning.

This topic was addressed in the previous question, but to summarize, yes, the type of response variable does affect the performance metric.

If the response variable is continuous, such as arousal or valence in their original form, the problem is a regression task. In this case, appropriate performance metrics include:

- Mean Squared Error (MSE): $\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2$ penalizes larger errors more heavily and is useful for optimization.
- Mean Absolute Error (MAE): $\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|\hat{y}_i - y_i$ is less sensitive to outliers and is easier to interpret.
- Coefficient of Determination ($R^2$ Score): $R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$ evaluates how well the model explains the variance in the data.

If the response variable is categorical, such as after binarization ('high' vs 'low') or discretization ('low', 'neutral', 'high'), the problem becomes a classification task. In that case, regression metrics are not appropriate, and instead classification metrics are used, typically derived from the confusion matrix:

- Accuracy: the proportion of correct predictions.
- Precision and Recall: especially important in imbalanced datasets.
- F1 Score: harmonic mean of precision and recall.
- ROC-AUC: useful for evaluating the ranking ability of probabilistic classifiers.
- Therefore, the selection of the performance metric must always align with the type of the response variable, as using an incorrect metric can lead to misleading evaluation of model performance.

## Which validation protocol (e.g., holdout set, k-fold cross-validation, etc.) would you use given that the objective is to build a predictive model able to generalise across participants (i.e., make accurate predictions for unseen participants)? Justify your choice.

Which validation protocol (e.g., holdout set, k-fold cross-validation, etc.) would you use given that the objective is to build a predictive model able to generalise across participants (i.e., make accurate predictions for unseen participants)? Justify your choice.

The challenge in this example's validation protocol selection is that the model has to generalise to unseen participants. The validation protocol must, therefore, prevent any overlap between the data used for training and the data used for evaluation across different participants that will lead to information leakage and overly optimistic performance estimates.

These considerations limit the use of popular validation techniques:

- **Holdout Set**. A simple train/test split may randomly divide samples from the same participant across both sets, resulting in data leakage, where the model indirectly 'sees' the test distribution during training, leading to over-optimistic performance estimates. It also suffers from high variance, especially with a low participant count.

- **Standard K-Fold Cross-Validation** involves randomly partitioning data into $k$ folds without considering participant boundaries. Consequently, it may include data from the same participant in both training and testing folds, violating the independence assumption.
- **Stratified K-Fold** preserves label distribution across folds; however, it does not maintain group integrity. Samples from the same participant may appear in multiple folds, which can lead to data leakage and inaccurate metrics.

Following these considerations, **Group K-Fold Cross-Validation** can be used. Group K-Fold Cross-Validation is a data splitting strategy that ensures all samples from a given group appear entirely in either the training set or the validation set and **never in both**, preserving the independence between training and test data when samples are, **not i.i.d.**, but grouped.

Group K-Fold Cross-Validation is used because in grouped data, **generalisation to unseen groups** (not just unseen samples) is the real objective. Group K-Fold simulates this deployment scenario more faithfully than standard techniques.

## Task 2: Using the provided audio features, build predictive models for arousal and valence:

### Develop a predictive model for each response variable (arousal and valence) using some or all of the provided audio features as explanatory variables.

### Evaluate the implemented models using the metrics and validation protocol you proposed in Task 1.

### Interpret the trained models (if the selected approach allows for interpretation) and the obtained results.

## Task 2 narrative

The initial architecture for this task utilized a scikit-learn pipeline structure, which was later discarded to allow for finer control and investigation of fold-level inconsistencies at all stages of the pipeline.

### Experiment 1 - Linear Regression

#### Input

- **Target Variable (** `y_median_arousal` **)**: the `median_arousal` column from the dataset.

- **Feature Matrix ( `X` )**: all 130 acoustic features extracted from audio signals, excluding `Participant`, `median_arousal`, and `median_valence`.

### Pipeline Components

- **GroupKFold**: with 10 splits was used to perform cross-validation using participants, so to ensure that each fold excluded one participant.
- **Recursive Feature Elimination**: applied to the training set of each fold using a `LinearRegression` estimator to select the top 10 features based on model coefficients.
- **StandardScaler**: applied after feature selection to standardize features to zero mean and unit variance, based solely on the training set.
- **LinearRegression**: trained on the selected and scaled features to model arousal as a continuous output.

### Execution

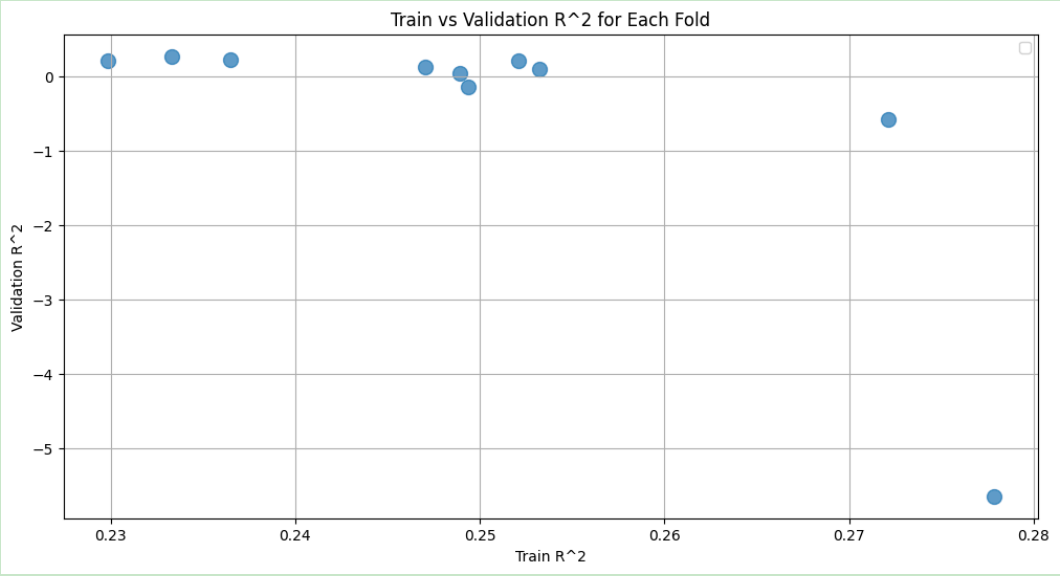
Cross-validation was implemented manually in a loop using `GroupKFold`. For each fold:

- Features were selected using the training data.
- Selected features were scaled.
- The model was trained on the training set and evaluated on the held-out validation set.

### Evaluation Metrics

Model performance was assessed using $R^2$, MSE and MAE that were computed for both training and validation sets to evaluate model fit and generalization.

Following the execution of this pipeline, a scatter plot was generated showing training versus validation $R^2$ values across folds.




The plot shows that the system suffered from both high bias and high variance. The low training $R^2$ values (between 0.23 and 0.28) indicated that the model was

unable to fit the training data well, indicating underfitting and high bias.

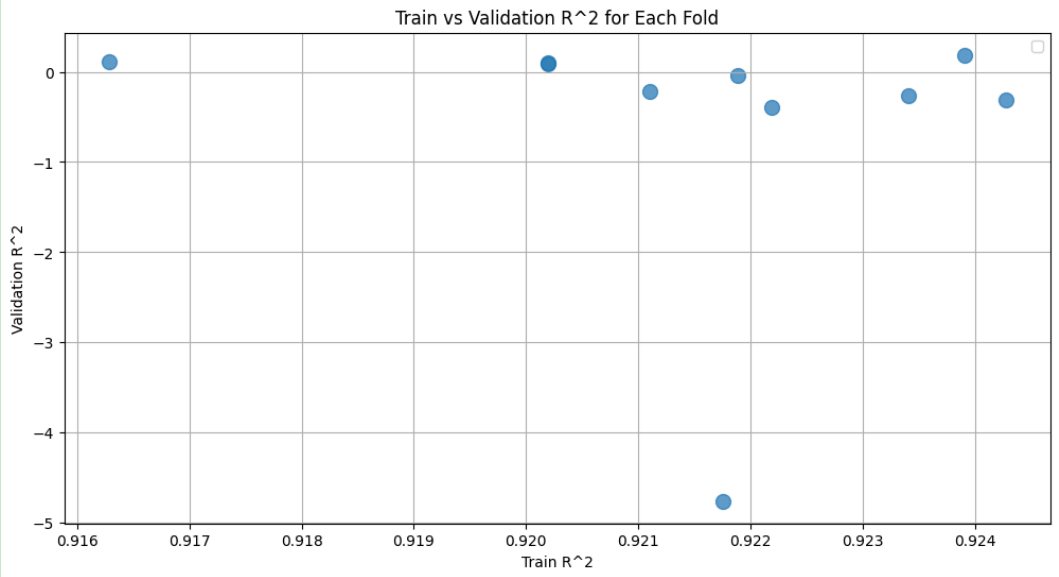
Simultaneously, validation performance was highly variable across folds, with some validation $R^2$ scores dropping below 0 and one fold reaching as low as –5.5. This extreme variability points to high variance.

Due to the issues with underfitting and instability, it was clear that we needed a better model. We chose the Random Forest Regressor because it can capture complex patterns and nonlinear interactions that linear regression can't, thus helping reduce bias in our predictions.

As the Random Forest Regressor performs ensemble averaging across multiple decision trees to produce stable predictions, it should also reduce variance.

## Experiment 2 - Random Forest Regressor and additional measures

The pipeline was subsequently modified by replacing the linear regressor with a Random Forest Regressor while leaving all other components (feature selection, scaling, and cross-validation setup) unchanged. This model modification was carried out to address the underfitting and limited expressiveness observed in the linear model. Under these conditions, the following scatter plot shows the comparison of training versus validation $R^2$ values across folds:




The resulting scatter plot of training versus validation $R^2$ values indicates that the bias problem observed in the linear regression model has been effectively addressed as we started seeing consistently high training $R^2$ values across folds (0.916 to 0.925). The Random Forest Regressor is capable of capturing the underlying relationships in the training data.

However, the plot also reveals that the model exhibits considerable variance across validation folds, with validation $R^2$ values ranging from slightly negative up to approximately 0.1. One fold, in particular, shows a strongly negative validation $R^2$, suggesting that the specific model generalises poorly.

While this variability may initially seem concerning, it is not unexpected in the context of Group K-Fold cross-validation, where each validation fold corresponds to an entirely unseen participant, so it is acceptable for validation $R^2$ values to approach zero or even slightly dip below, especially when the data distribution diverges significantly from the training population.

Thus, while the high training performance confirms that Random Forests solve the underfitting issue (high bias), the persistent fold-to-fold variation reflects a high variance regime.

## Addressing poor performance

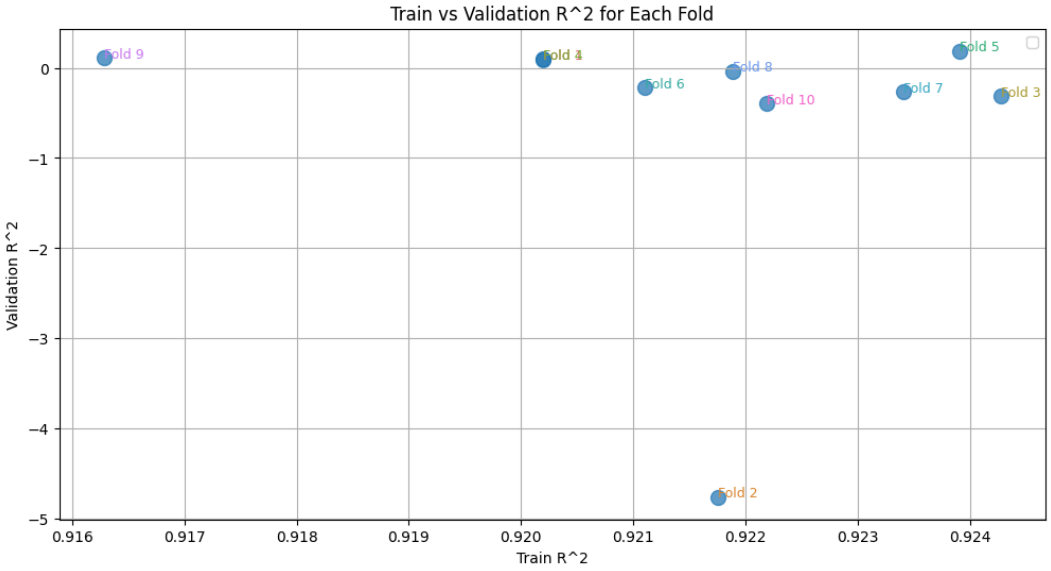
At this stage, a decent model had been developed that addressed most scenarios. However, before reaching complete satisfaction with the model, a solution was needed for the problematic fold, and there was also potential to improve the overall performance of the model.

A couple of techniques were implemented at this stage without objective analysis, an approach which proved to be very inefficient indeed. The underlying assumption at this stage was that the data contained outliers that were causing problems.

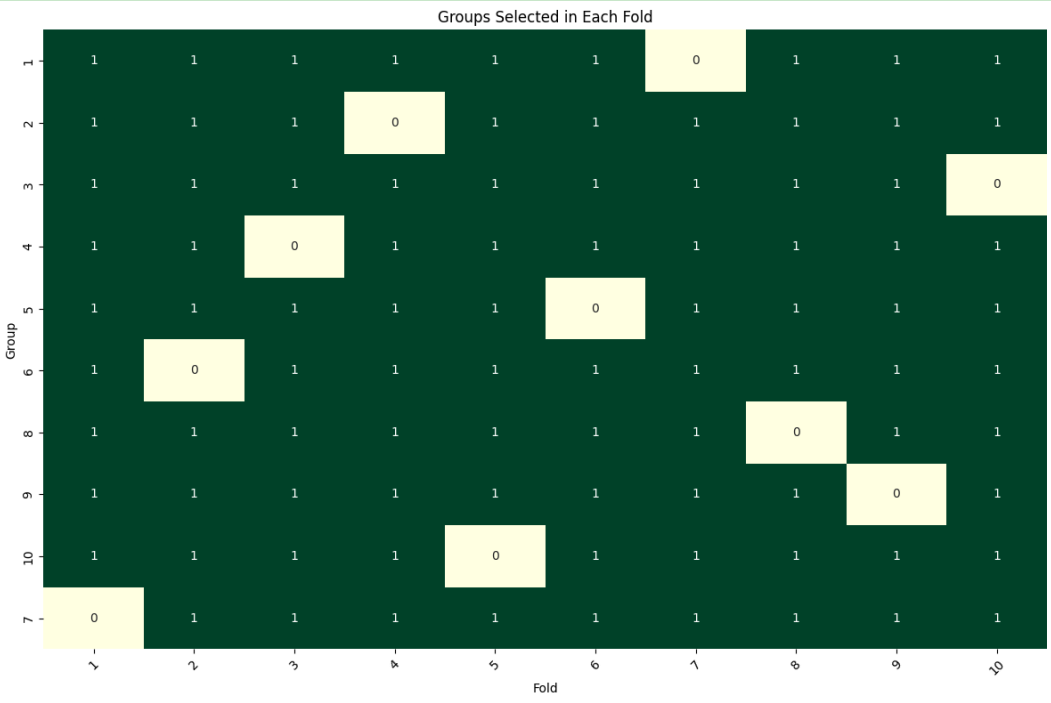
Replaced StandardScaler with RobustScaler to reduce sensitivity to outlier-prone features. There was a slight improvement in fold consistency, but core issues persisted. StandardScalar was then reintroduced. Added two new principal components from PCA on the full feature matrix to the feature set before training to inject global structure and reduce unexplained variance. These features were removed as although there was a visual improvement in variance reduction in some folds, validation $R^2$ was slightly more stable but not universally improved. Removed rows where `y_median_arousal` exceeded $|z|3$, computed per participant to reduce label noise. This step did not substantially improve fold consistency but was retained for completeness' sake.

At this stage, it became clear that randomly searching for a good solution was not effective. It was essential to understand the cause of the validation $R^2$ results, particularly the one that produced a significantly poor outcome.

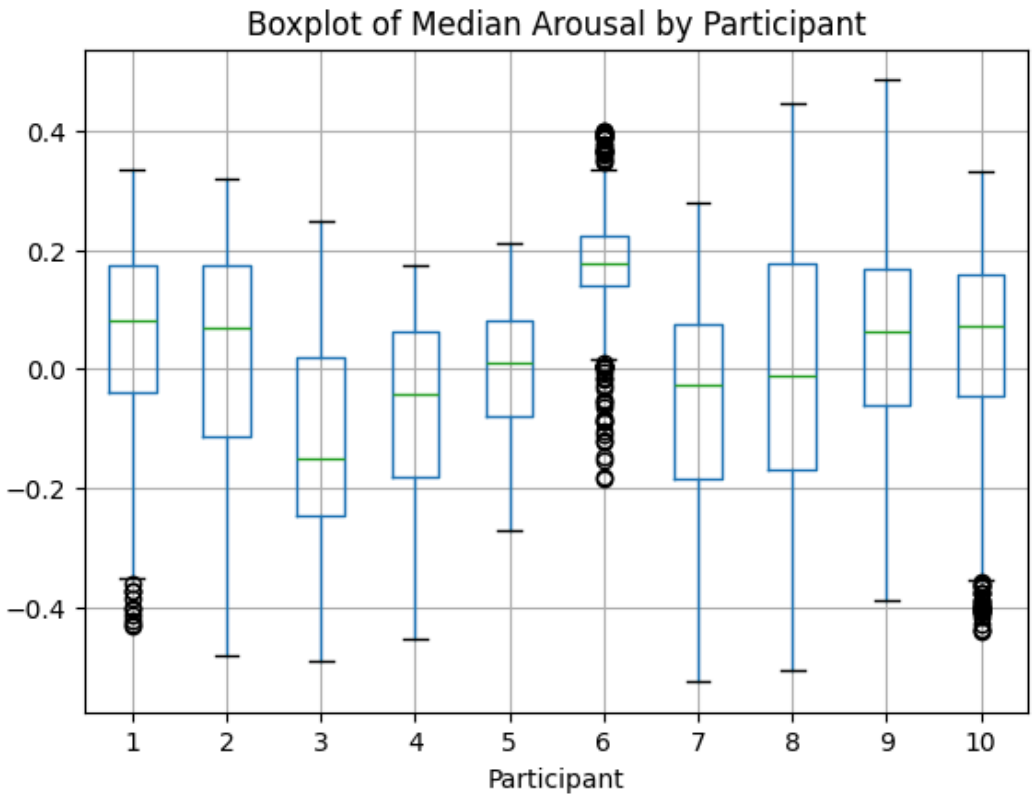
This investigation began with a slight modification to the scatter plot to understand which fold resulted in poor outcomes. As can be seen in the following plot, this was fold number 2.




A thorough understanding of each iteration of the Group K-Fold execution was achieved to identify which participants were held out for validation. In particular, it was essential to determine the held-out participant in Fold-2. This analysis showed that the participant in question was number 6.




A box plot of `y_median_arousal` for every participant was created. The plot revealed numerous outliers in Participant 6, which initially raised questions about outlier elimination, as it had been previously carried out in the pipeline. Upon analysis of Participant 6, it was noticed that its distribution differed from the distribution of the other participants:




- The mean was around 0.2, whereas the distribution of the other participants was around 0
- The distribution was very tight around the mean, whereas the distribution of the other participants was wider

It was, therefore, evident at this stage that the problem with Participant 6 was not due to outliers, albeit the box plot was portraying it as such. However, participant 6 is statistically different from all the others, and therefore, the model's assumptions do not apply to it, leading to a high validation error.

A good technique, given this information, is to normalise the target values of each participant as follows:

$$\text{normalized arousal} = \frac{y - \mu_{participant}}{\sigma_{participant}}$$

Participant-level normalisation will mean that for all participants, the mean of mean_arousal will be zero, and its standard deviation will be 1. The model is, in this way, trained on the relative changes of mean_arousla and not the absolute values. This approach significantly improved the result, as evident in the final code pipeline below.

---

## Additional references

- https://www.youtube.com/watch?v=6dDet0-Drzc

In [6]:
```
# # draw boxplot of median_arousal for each participant
# df_raw.boxplot(column='median_arousal', by='Participant')
# plt.title('Boxplot of Median Arousal by Participant')
# plt.suptitle('')
# plt.show()


# for fold in range(NUMBER_OF_SPLITS):
#     selected_groups = results_df.loc[fold, 'Groups']
#     for group in groups_selected_df.index:
#         if group in selected_groups:
#             groups_selected_df.at[group, fold + 1] = 'yes'
#         else:
#             groups_selected_df.at[group, fold + 1] = 'no'

# groups_selected_df.to_csv('groups_selected.csv')

# # draw groups_selected_df as a heatmap
# plt.figure(figsize=(12, 8))
# sns.heatmap(groups_selected_df.applymap(lambda x: 1 if x == 'yes' else
# plt.title('Groups Selected in Each Fold')
# plt.xlabel('Fold')
# plt.ylabel('Group')
# plt.xticks(rotation=45)
# plt.tight_layout()
# plt.show()
```

In [7]:
```
from sklearn.model_selection import GroupKFold
from sklearn.linear_model import LinearRegression
from sklearn.feature_selection import RFE
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_e
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestRegressor
import pandas as pd
from sklearn.preprocessing import StandardScaler
from scipy.stats import zscore
from sklearn.decomposition import PCA
```

## Outlier handling

Extreme ratings of arousal and valence that are unusually high or low (beyond 3 standard deviations) for each participant are removed. This method helps to avoid bias and ensures that individual differences are considered, which is important in the study of emotions.

In [8]:
```
cleaned_rows = []
z_thresh = 3

for pid in df_raw['Participant'].unique():
    participant_df = df_raw[df_raw['Participant'] == pid].copy()

    # Calculate Z-scores for both arousal and valence
    participant_df['z_arousal'] = zscore(participant_df['median_arousal']
    participant_df['z_valence'] = zscore(participant_df['median_valence']
```

```
    # Keep only rows where both z-scores are within threshold
    participant_df = participant_df[
        (participant_df['z_arousal'].abs() <= z_thresh) &
        (participant_df['z_valence'].abs() <= z_thresh)
    ]

    # Drop the z-score columns
    participant_df = participant_df.drop(columns=['z_arousal', 'z_valence

    cleaned_rows.append(participant_df)

df_clean = pd.concat(cleaned_rows, ignore_index=True)

print(f'Original dataset shape: {df_raw.shape}')
print(f'Cleaned dataset shape: {df_clean.shape}')
```

```
Original dataset shape: (7238, 133)
Cleaned dataset shape: (7175, 133)
```

In [9]:
```
# Not used

# from sklearn.model_selection import train_test_split

# # Extract features, labels, and stratification labels
# X = df_clean.drop(columns=['Participant', 'median_arousal', 'median_val
# y = df_clean[['median_arousal', 'median_valence']]
# stratify_labels = df_clean['Participant']

# # Do a stratified split based on Participant column (row-wise stratific
# X_train_val, X_test, y_train_val, y_test = train_test_split(
#     X, y,
#     test_size=0.2,
#     stratify=stratify_labels,
#     random_state=42
# )

# # Also extract participant column for each split
# groups_train_val = stratify_labels.loc[X_train_val.index]
# groups_test = stratify_labels.loc[X_test.index]

# # Optional: check proportions
# print('Train participant proportions:')
# print(groups_train_val.value_counts(normalize=True).sort_index())

# print('\nTest participant proportions:')
# print(groups_test.value_counts(normalize=True).sort_index())
```

## Feature and target extraction

Key variables are set up by separating the input features (X) target variables (y_median_arousal and y_median_valence) grouping labels (groups).

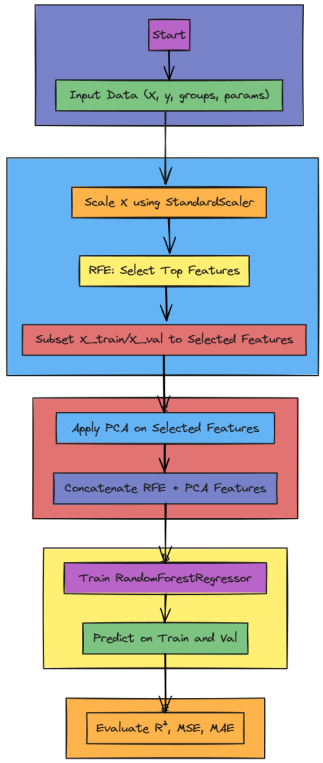
This separation is important for creating models that are fair, reproducible, and can be used with new participants.

In [10]:
```
X = df_clean.drop(columns=['Participant', 'median_arousal', 'median_valen
print(f'Shape of feature matrix X: {X.shape}')

y_median_valence = df_clean['median_valence']

y_median_arousal = df_clean['median_arousal']

groups = df_clean['Participant']

print(f'Shape of target variable median_arousal: {y_median_arousal.shape}
print(f'Shape of target variable median_valence: {y_median_valence.shape}
print(f'Number of unique participants: {len(groups.unique())}')
```

```
Shape of feature matrix X: (7175, 130)
Shape of target variable median_arousal: (7175,)
Shape of target variable median_valence: (7175,)
Number of unique participants: 10
```

# Pipeline setup

This pipeline uses Group K-Fold Cross-Validation integrated with feature selection and dimensionality reduction, as the dataset has observations that are grouped by participant. It ensures that group integrity is preserved during validation to prevent data leakage. The pipeline combines Recursive Feature Elimination to identify the most relevant predictors, Principal Component Analysis to extract key latent patterns, and a Random Forest Regressor to model relationships in the data. Evaluation metrics are collected across all folds, enabling performance tracking and insight into model consistency. The final output is a structured results summary, exported and returned for further analysis or reporting.



1. **Data and Parameters Preparation** The pipeline begins by receiving the feature matrix, target variable, group labels, and configuration parameters, such as the number of cross-validation folds, the number of features to select, the number of PCA components, and optional model hyperparameters.
2. **Group-Based Data Splitting** The dataset is divided into training and validation sets using Group K-Fold cross-validation, ensuring that samples from the same group (participant) are kept together, thereby preventing any leakage between the training and validation sets.
3. **Feature Standardization** All input features are standardized within the training set.
4. **Feature Selection via RFE** Recursive Feature Elimination is used to identify and retain the most predictive features, reducing dimensionality and noise.
5. **Dimensionality Reduction Using PCA** Principal Component Analysis is applied to the selected features to extract a set of orthogonal components that capture the most significant variance in the data. This step helps compress the data while preserving its essential structure.
6. **Feature Set Enrichment** The original selected features are combined with the PCA-derived components to create an enriched input matrix.
7. **Model Training** A Random Forest Regressor is trained on the enriched feature set. This model was selected as it is capable of learning complex, nonlinear patterns.
8. **Prediction Generation** The trained model is used to generate predictions for both the training and validation datasets.
9. **Performance Evaluation** Model performance is evaluated using key regression metrics: $R^2$ score, mean squared error and mean absolute error across both training and validation splits.
10. **Fold-Level Result Recording** For each fold, the pipeline logs the selected features, group identifiers used in training, and all performance metrics. Logging enables fold-wise analysis of model behaviour and consistency. These results are aggregated into a summary table, which is returned as a DataFrame, providing a programmatic interface for further analysis or visualization.

In [11]:
```
def regression_cv_pipeline(
    X: pd.DataFrame,
    y: pd.Series,
    groups: pd.Series,
    splits: int = 10,
    rfe_features : int = 10,
    pca_components : int = 2 ,
    results_path : str = 'results.csv',
    model_params : dict = None):

    gkf = GroupKFold(n_splits=splits)
    results_df = pd.DataFrame(columns=['Fold', 'Groups', 'Selected Featur
    # print('Group K Fold Start...')
    fold_counter = 1
```

```
    selected_features_folds = []

    for train_index_fold, val_index_fold in gkf.split(X, y, groups):

        # normalize y train and val in the fold (moved from outside the l
        y_train = (y.iloc[train_index_fold] - y.iloc[train_index_fold].me
        y_val = (y.iloc[val_index_fold] - y.iloc[val_index_fold].mean())

        # x_train and x_val in the fold
        X_train = X.iloc[train_index_fold]
        X_val = X.iloc[val_index_fold]

        # groups in the train -- should be all participants minus the one
        groups_train = groups.iloc[train_index_fold]

        # scale features for rfe as it uses lin reg
        scaler = StandardScaler()
        X_train_scaled = pd.DataFrame(scaler.fit_transform(X_train), colu
        X_val_scaled = pd.DataFrame(scaler.transform(X_val), columns=X_va

        estimator = LinearRegression()
        rfe_selector = RFE(estimator=estimator, n_features_to_select=rfe_
        rfe_selector.fit(X_train_scaled, y_train)
        selected_features_mask = rfe_selector.support_
        current_selected_features = X_train_scaled.columns[selected_featu
        selected_features_folds.append(current_selected_features)

        # narrow down the x train and val to the selected features
        X_train_selected = X_train[current_selected_features]
        X_val_selected = X_val[current_selected_features]

        # print('shape of X_train_selected:', X_train_selected.shape)
        pca = PCA(n_components=pca_components)
        X_train_pca_features = pca.fit_transform(X_train_scaled)
        X_val_pca_features = pca.transform(X_val_scaled)

        X_train_selected = np.concatenate([X_train_selected.values, X_tra
        X_val_selected = np.concatenate([X_val_selected.values, X_val_pca
        # # print('shape of X_train_selected:', X_train_selected.shape)

        # now we try to fit a a rand forest
        # parameter injection
        if model_params is None:
            model_params = {
                'n_estimators': 100,
                'max_depth': None,
                'min_samples_split': 2,
                'min_samples_leaf': 1,
                'max_features': None
            }

        n_estimators = model_params.get('n_estimators', 100)
        max_depth = model_params.get('max_depth', None)
        min_samples_split = model_params.get('min_samples_split', 2)
        min_samples_leaf = model_params.get('min_samples_leaf', 1)
        max_features = model_params.get('max_features', None)

        # define the model
        model = RandomForestRegressor(
            n_estimators=n_estimators,
```

```
            max_depth=max_depth,
            min_samples_split=min_samples_split,
            min_samples_leaf=min_samples_leaf,
            max_features=max_features,
            random_state=42
        )

        # fit the model on the train set
        model.fit(X_train_selected, y_train)

        # now predict on train and pray
        y_train_pred = model.predict(X_train_selected)

        train_r2 = r2_score(y_train, y_train_pred)
        train_mse = mean_squared_error(y_train, y_train_pred)
        train_mae = mean_absolute_error(y_train, y_train_pred)
        # print(f'  Training R^2: {train_r2:.4f}, MSE: {train_mse:.4f}, M


        # now predict on val and pray
        y_val_pred = model.predict(X_val_selected)
        val_r2 = r2_score(y_val, y_val_pred)
        val_mse = mean_squared_error(y_val, y_val_pred)
        val_mae = mean_absolute_error(y_val, y_val_pred)
        # print(f'  Validation R^2: {val_r2:.4f}, MSE: {val_mse:.4f}, MAE

        # save this fold result in results dataframe
        results_dict = {
            'Fold': fold_counter,
            'Groups': groups_train.unique().tolist(),
            'Selected Features': current_selected_features,
            'Train R^2': train_r2,
            'Train MSE': train_mse,
            'Train MAE': train_mae,
            'Val R^2': val_r2,
            'Val MSE': val_mse,
            'Val MAE': val_mae
        }
        # print(f'  Results for fold {fold_counter}: {results_dict}')
        if fold_counter == 1:
            results_df = pd.DataFrame([results_dict])
        else:
            results_df = pd.concat([results_df, pd.DataFrame([results_dic

        fold_counter += 1

    # save the results dataframe in a file
    results_df.to_csv(results_path, index=False)
    return results_df
```

## Model evaluation

Final Model Evaluation for Arousal and Valence

The results, including $R^2$, MSE, and MAE, are averaged across the folds and exported for reporting purposes. This allows for a comparative assessment of the

model's effectiveness in predicting arousal and valence from audio features. This step serves as the final quantitative validation of your affect modeling approach.

In [12]:
```
NUMBER_OF_SPLITS_AROUSAL = 6
NUMBER_OF_RFE_FEATURES_AROUSAL = 45
param_arousal = {
    'n_estimators': 200,
    'max_depth': 5,
    'min_samples_split': 5,
    'min_samples_leaf': 1,
    'max_features': 'log2',
}
PCA_COMPONENTS_AROUSAL = 30

results_arousal =  regression_cv_pipeline(X, y_median_arousal, groups, pc
results_arousal.to_csv('results/results_arousal.csv', index=False)

mean_train_r2 = results_arousal['Train R^2'].mean()
mean_train_mse = results_arousal['Train MSE'].mean()
mean_train_mae = results_arousal['Train MAE'].mean()

mean_val_r2 = results_arousal['Val R^2'].mean()
mean_val_mse = results_arousal['Val MSE'].mean()
mean_val_mae = results_arousal['Val MAE'].mean()

print(f'Mean Train R^2: {mean_train_r2:.4f}, MSE: {mean_train_mse:.4f}, M
print(f'Mean Val R^2: {mean_val_r2:.4f}, MSE: {mean_val_mse:.4f}, MAE: {m

############################################################

NUMBER_OF_SPLITS_VALANCE = 7
NUMBER_OF_RFE_FEATURES_VALENCE = 15
param_valence = {
    'n_estimators': 300,
    'max_depth': 5,
    'min_samples_split': 2,
    'min_samples_leaf': 5,
    'max_features': 'log2'
}
PCA_COMPONENTS_VALANCE= 20

results_valence = regression_cv_pipeline(X, y_median_valence, groups, pca
results_valence.to_csv('results/results_valence.csv', index=False)

mean_train_r2 = results_valence['Train R^2'].mean()
mean_train_mse = results_valence['Train MSE'].mean()
mean_train_mae = results_valence['Train MAE'].mean()

mean_val_r2 = results_valence['Val R^2'].mean()
mean_val_mse = results_valence['Val MSE'].mean()
mean_val_mae = results_valence['Val MAE'].mean()

print(f'Mean Train R^2: {mean_train_r2:.4f}, MSE: {mean_train_mse:.4f}, M
print(f'Mean Validation R^2: {mean_val_r2:.4f}, MSE: {mean_val_mse:.4f},
```

```
Mean Train R^2: 0.3832, MSE: 0.6167, MAE: 0.6318
Mean Val R^2: 0.2381, MSE: 0.7612, MAE: 0.6925
Mean Train R^2: 0.1579, MSE: 0.8420, MAE: 0.7321
Mean Validation R^2: 0.0325, MSE: 0.9664, MAE: 0.7938
```

## Cross-Validation Metrics Analysis

This function analyzes the results of cross-validation runs in the prediction pipeline. The `analyze_cv_results` function provides summaries and plots to show how well the model performs across different folds of validation. It also gives insights into how stable the feature selection is across these splits.

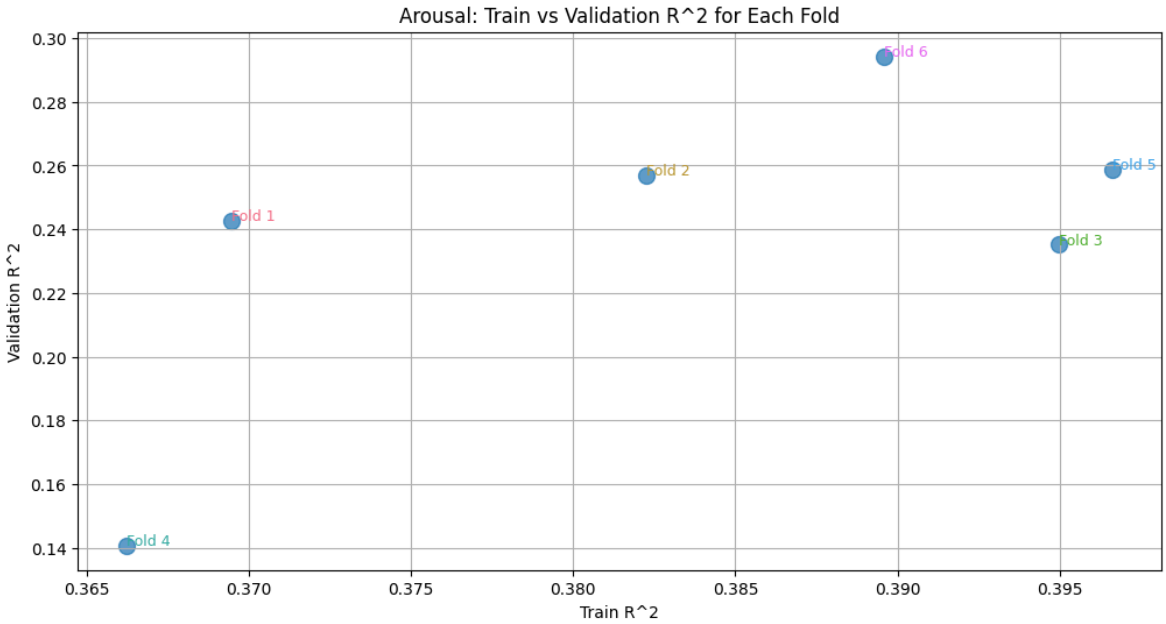
Key Features:

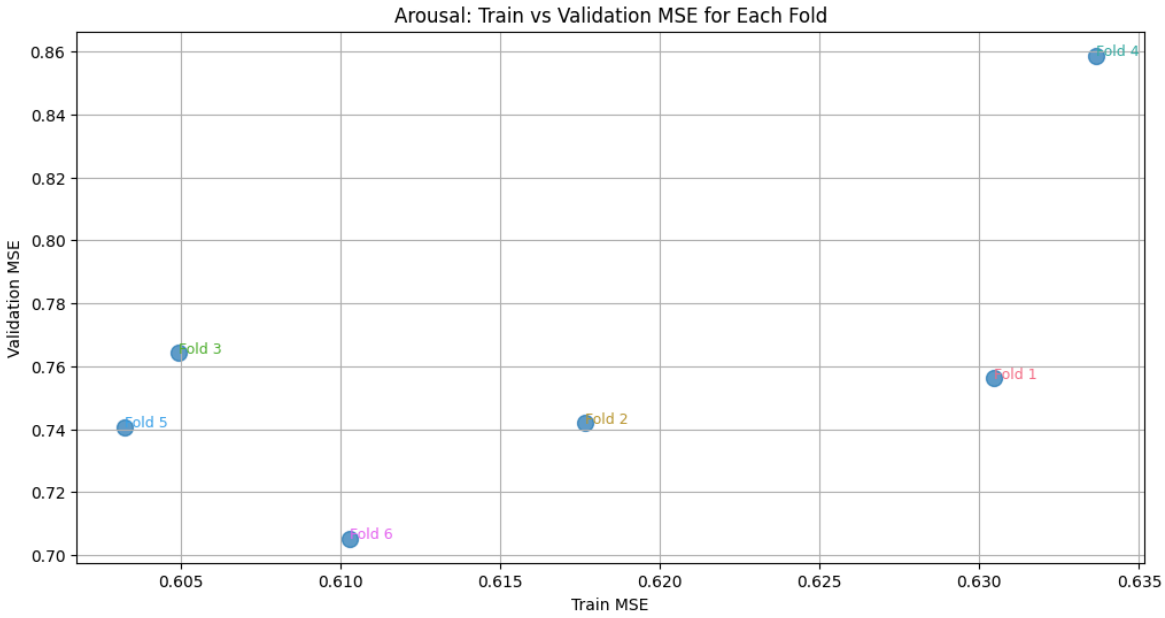
- Metric Aggregation
- Performance Scatter Plots
- Fold-wise Annotation
- Feature Selection Heatmap

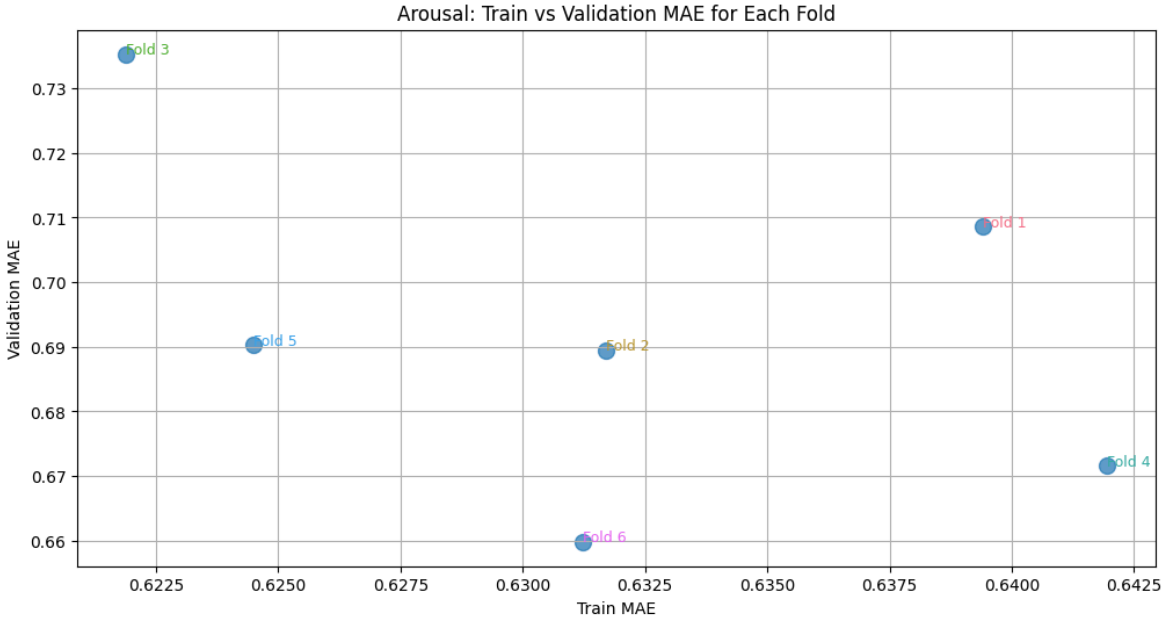
In [13]:
```
import seaborn as sns

def analyze_cv_results(results_df, n_splits, title_prefix='', show_graphs
    '''
    Analyze cross-validation results: print mean metrics, plot train vs v
    and show feature selection heatmap.
    '''
    import matplotlib.pyplot as plt

    mean_train_r2 = results_df['Train R^2'].mean()
    mean_train_mse = results_df['Train MSE'].mean()
    mean_train_mae = results_df['Train MAE'].mean()

    mean_val_r2 = results_df['Val R^2'].mean()
    mean_val_mse = results_df['Val MSE'].mean()
    mean_val_mae = results_df['Val MAE'].mean()


    print(f'Results for {title_prefix}')
    print(f'Mean Train R^2: {mean_train_r2:.4f}, MSE: {mean_train_mse:.4f
    print(f'Mean Validation R^2: {mean_val_r2:.4f}, MSE: {mean_val_mse:.4


    if show_graphs:
        colors = sns.color_palette('husl', n_splits)

        plt.figure(figsize=(12, 6))
        plt.scatter(results_df['Train R^2'], results_df['Val R^2'], alpha
        plt.title(f'{title_prefix}Train vs Validation R^2 for Each Fold')
        plt.xlabel('Train R^2')
        plt.ylabel('Validation R^2')
        for i, row in results_df.iterrows():
            plt.annotate(f"Fold {row['Fold']}", (row['Train R^2'], row['V
        plt.grid(True)
```

```
        plt.show()

        plt.figure(figsize=(12, 6))
        plt.scatter(results_df['Train MSE'], results_df['Val MSE'], alpha
        plt.title(f'{title_prefix}Train vs Validation MSE for Each Fold')
        plt.xlabel('Train MSE')
        plt.ylabel('Validation MSE')
        for i, row in results_df.iterrows():
            plt.annotate(f"Fold {row['Fold']}", (row['Train MSE'], row['V
        plt.grid(True)
        plt.show()

        plt.figure(figsize=(12, 6))
        plt.scatter(results_df['Train MAE'], results_df['Val MAE'], alpha
        plt.title(f'{title_prefix}Train vs Validation MAE for Each Fold')
        plt.xlabel('Train MAE')
        plt.ylabel('Validation MAE')
        for i, row in results_df.iterrows():
            plt.annotate(f"Fold {row['Fold']}", (row['Train MAE'], row['V
        plt.grid(True)
        plt.show()

        # Feature selection heatmap
        selected_features_set = set()
        for fold in range(n_splits):
            selected_features_set.update(results_df.loc[fold, 'Selected F
        selected_features_set = list(selected_features_set)

        features_selected_df = pd.DataFrame(index=selected_features_set,
        for fold in range(n_splits):
            selected_features = results_df.loc[fold, 'Selected Features']
            for feature in selected_features_set:
                features_selected_df.at[feature, fold + 1] = 'yes' if fea

        features_selected_df.to_csv(f'results/{title_prefix}features_sele

        plt.figure(figsize=(12, 8))
        sns.heatmap(features_selected_df.applymap(lambda x: 1 if x == 'ye
        plt.title(f'{title_prefix}Features Selected in Each Fold')
        plt.xlabel('Fold')
        plt.ylabel('Feature')
        plt.xticks(rotation=45)
        plt.tight_layout()
        plt.show()

analyze_cv_results(results_arousal, NUMBER_OF_SPLITS_AROUSAL, title_prefi
analyze_cv_results(results_valence, NUMBER_OF_SPLITS_VALANCE, title_prefi
```

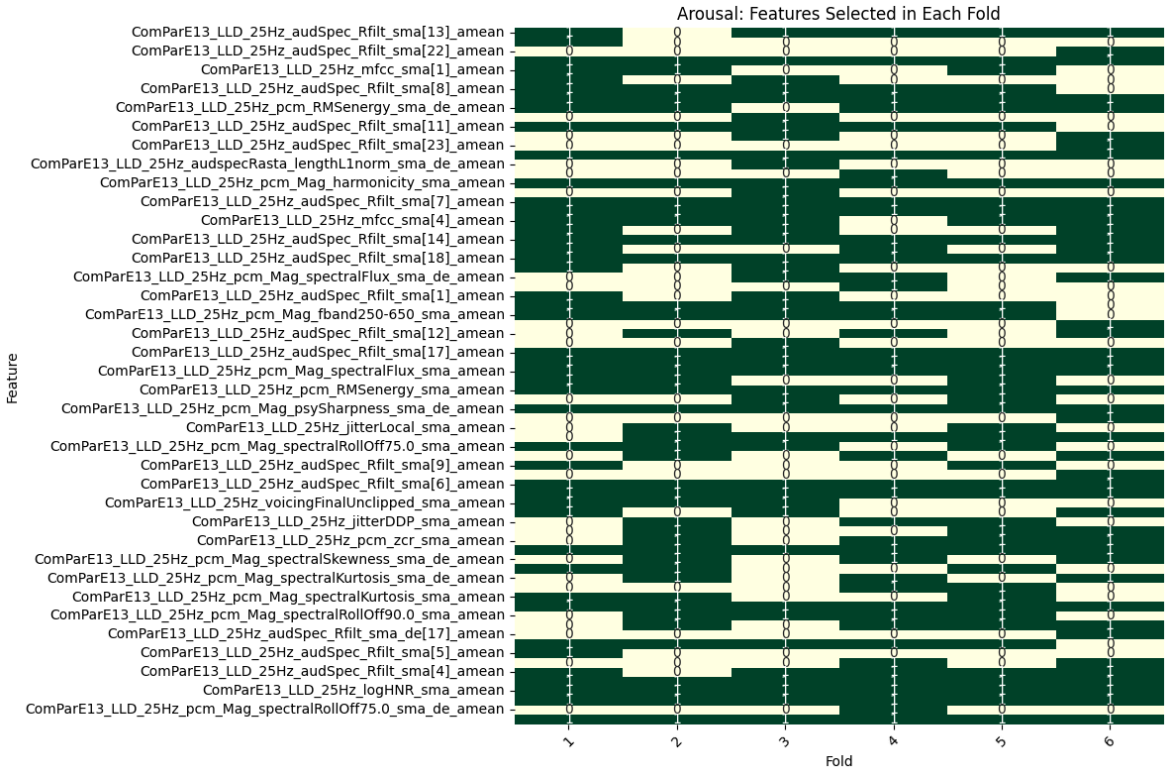
```
Results for Arousal:
Mean Train R^2: 0.3832, MSE: 0.6167, MAE: 0.6318
Mean Validation R^2: 0.2381, MSE: 0.7612, MAE: 0.6925
```


Arousal: Train vs Validation R^2 for Each Fold
Validation R^2
Train R^2
Fold 1
Fold 2
Fold 3
Fold 4
Fold 5
Fold 6

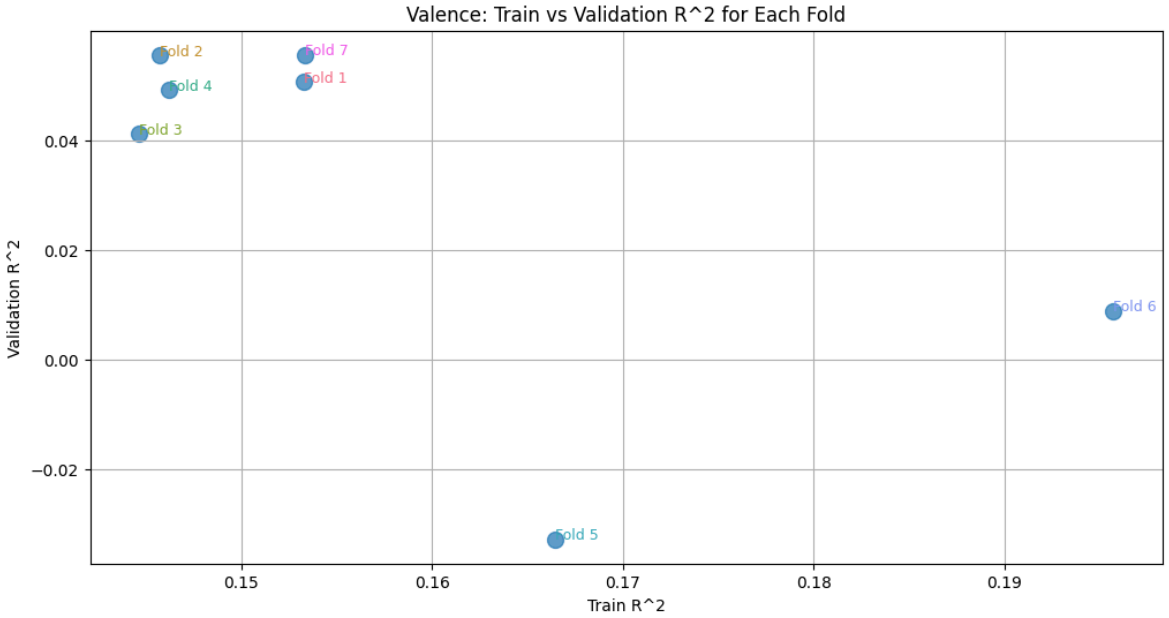

Arousal: Train vs Validation MSE for Each Fold
Validation MSE
Train MSE
Fold 1
Fold 2
Fold 3
Fold 4
Fold 5
Fold 6

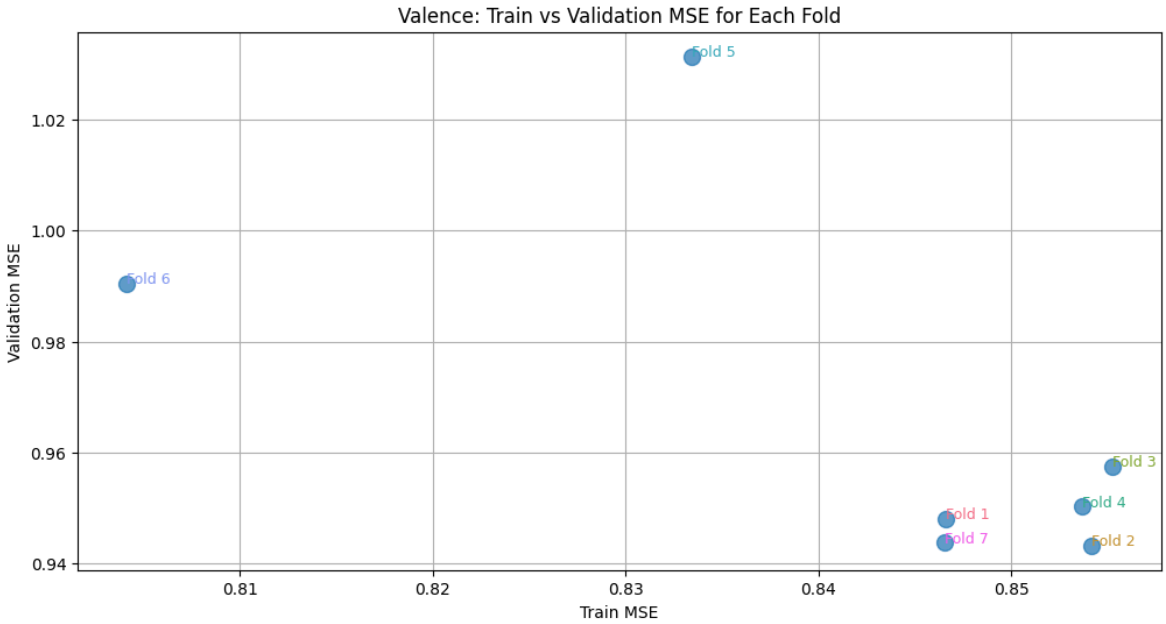

Arousal: Train vs Validation MAE for Each Fold
Validation MAE
Train MAE
Fold 1
Fold 2
Fold 3
Fold 4
Fold 5
Fold 6

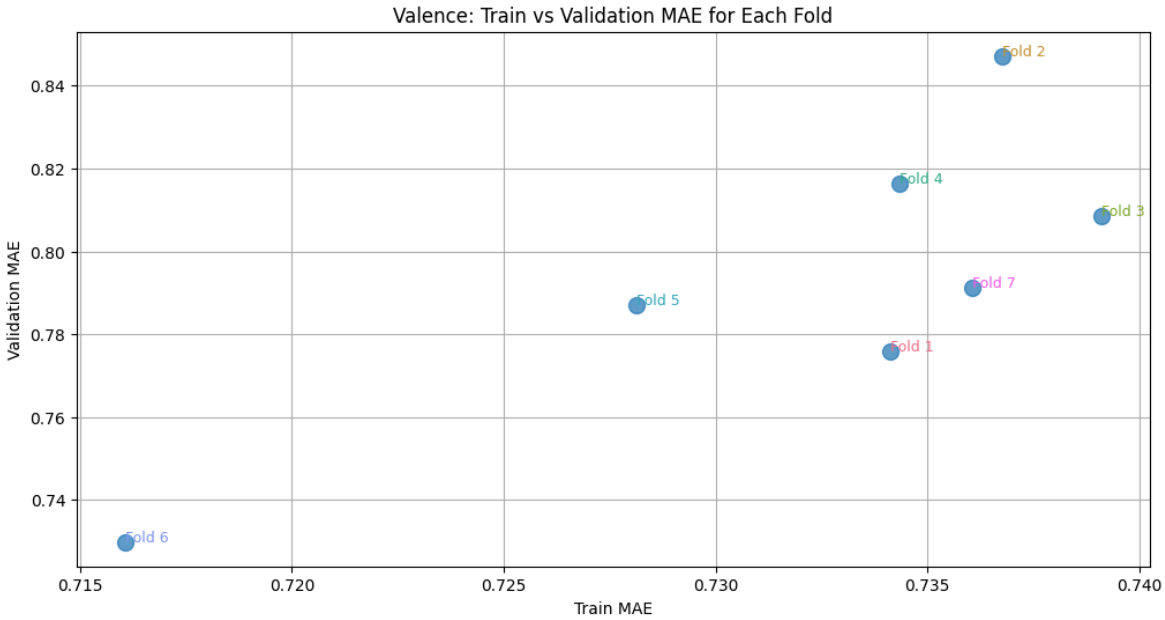
```
C:\Users\carme\AppData\Local\Temp\ipykernel_47612\2780075607.py:72: FutureWarning: DataFrame.applymap has been deprecated. Use DataFrame.map instead.
  sns.heatmap(features_selected_df.applymap(lambda x: 1 if x == 'yes' else 0), cmap='YlGn', annot=True, fmt='d', cbar=False)
```



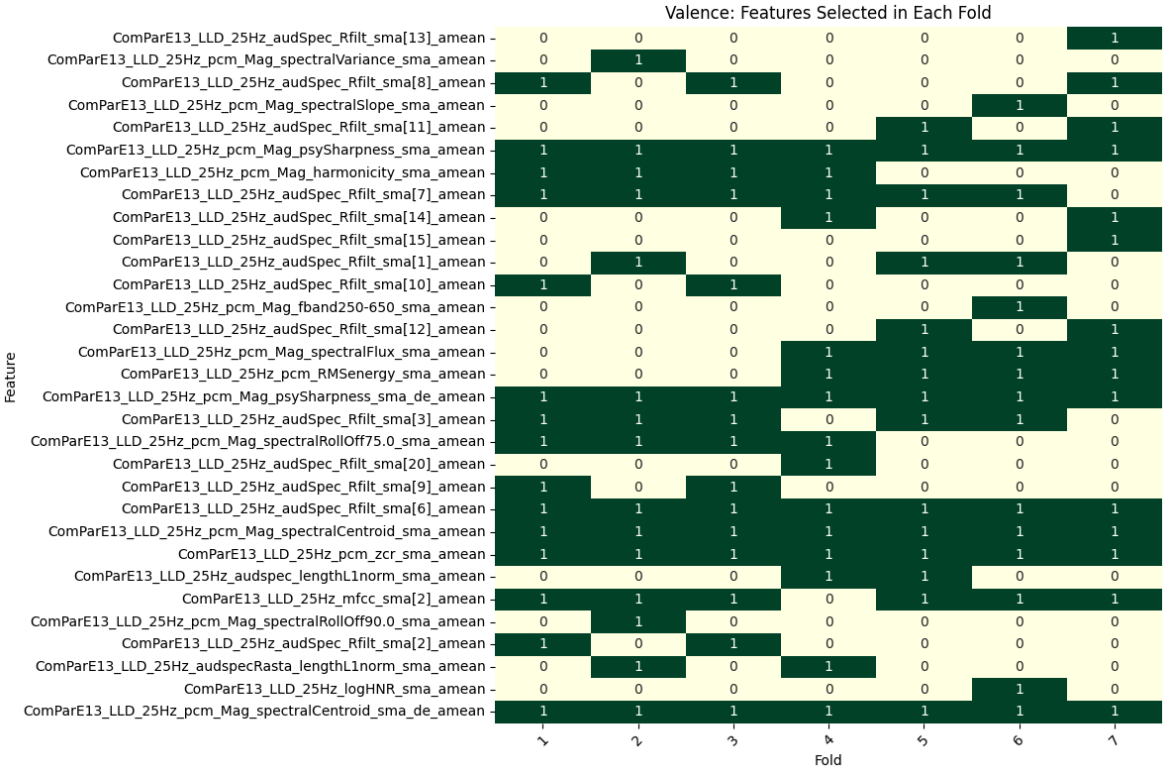

```
Results for Valence:
Mean Train R^2: 0.1579, MSE: 0.8420, MAE: 0.7321
Mean Validation R^2: 0.0325, MSE: 0.9664, MAE: 0.7938
```






```
C:\Users\carme\AppData\Local\Temp\ipykernel_47612\2780075607.py:72: FutureWarning: DataFrame.applymap has been deprecated. Use DataFrame.map instead.
  sns.heatmap(features_selected_df.applymap(lambda x: 1 if x == 'yes' else 0), cmap='YlGn', annot=True, fmt='d', cbar=False)
```




# Evaluation Metrics

The model's performance was assessed using three key regression metrics: $R^2$, Mean Squared Error (MSE), and Mean Absolute Error (MAE). These were computed on both training and validation sets using GroupKFold cross-validation, ensuring validation always occurred on unseen participants. The results are reported separately for arousal and valence prediction.

## Arousal Prediction

$R^2$

- Training ($R^2 = 0.3832$) indicates moderate model fit to the training data, capturing around 38% of the variance in arousal scores. This suggests that the model learns meaningful structure while avoiding overfitting.
- Validation ($R^2 = 0.2381$) reflects relatively strong generalization performance given the difficulty of the task. A $R^2$ above 0.2 in cross-participant affect modeling is considered promising, especially for continuous regression on z-normalized arousal scores.
- The scatter plot of Train vs Validation $R^2$ shows most folds clustering tightly in the range 0.24–0.29, suggesting stable fold-wise generalization.

**Mean Squared Error (MSE)**

- Training MSE = 0.6167, Validation MSE = 0.7612
- The modest gap between training and validation MSEs shows controlled variance and no severe overfitting.

- Visualizations show folds distributed within a tight band, supporting model consistency.

**Mean Absolute Error (MAE)**

- Training MAE = 0.6318, Validation MAE = 0.6925
- With MAE under 0.7 (in z-normalized space), the model achieves reasonable average prediction error, further confirming generalizability.

### Feature Selection Stability

The heatmap of selected features across folds shows many features were consistently selected, indicating that the model relies on a stable set of acoustic indicators to predict arousal.

### Summary

The arousal model demonstrates generalization with a validation $R^2$ of 0.2381, outperforming earlier configurations. The small gap between training and validation metrics reflects a good balance between bias and variance. Feature selection was also stable, reinforcing the reliability of the input space.

## Valence Prediction

$R^2$

- Training ($R^2 = 0.1579$) shows that the model captures only a small amount of the variance in valence scores.
- Validation ($R^2 = 0.0325$) is marginally above zero, indicating near-baseline performance. Despite tuning, the model struggles to generalize to unseen participants for this variable.
- The validation $R^2$ plot shows tight clustering near zero across folds, confirming the absence of consistent predictive patterns.

**Mean Squared Error (MSE)**

- Training MSE = 0.8420, Validation MSE = 0.9664
- The larger gap compared to arousal suggests greater variance, possibly due to label noise or weaker correlation between acoustic features and valence.

**Mean Absolute Error (MAE)**

- Training MAE = 0.7321, Validation MAE = 0.7938
- While not disastrous, these values indicate that the model struggles to reliably predict valence across participants.

### Feature Selection Stability

The valence heatmap shows high variability in selected features across folds, indicating low feature stability, consistent with poor predictive power.

### Summary

Valence prediction remains a challenging task in this setup. The model shows weak generalization, with validation metrics near baseline. These results are consistent with known challenges in modeling valence acoustically, due to its subjective and ambiguous nature.

### Overall Observations

- Arousal prediction is meaningfully above baseline, with fold-stable metrics and stable features.
- Valence prediction performs near baseline with high variance and inconsistent feature selection.

---

### Additional references

- https://stackoverflow.com/questions/54859240/combination-of-all-parameters-for-grid-search
- https://www.nature.com/articles/s43856-024-00468-0

## Components sweep

Thhe following cells conduct a systematic exploration of the number of RFE features, PCA components, and Group K-old splits, to assess their effect on the generalization performance of the arousal and valence regression models. The aim is to identify stable or optimal dimensionalities while holding all other model and feature selection parameters constant.

This process facilitates evidence-based tuning of hyperparameter and helps validate the choices made regarding dimensionality in the final model.

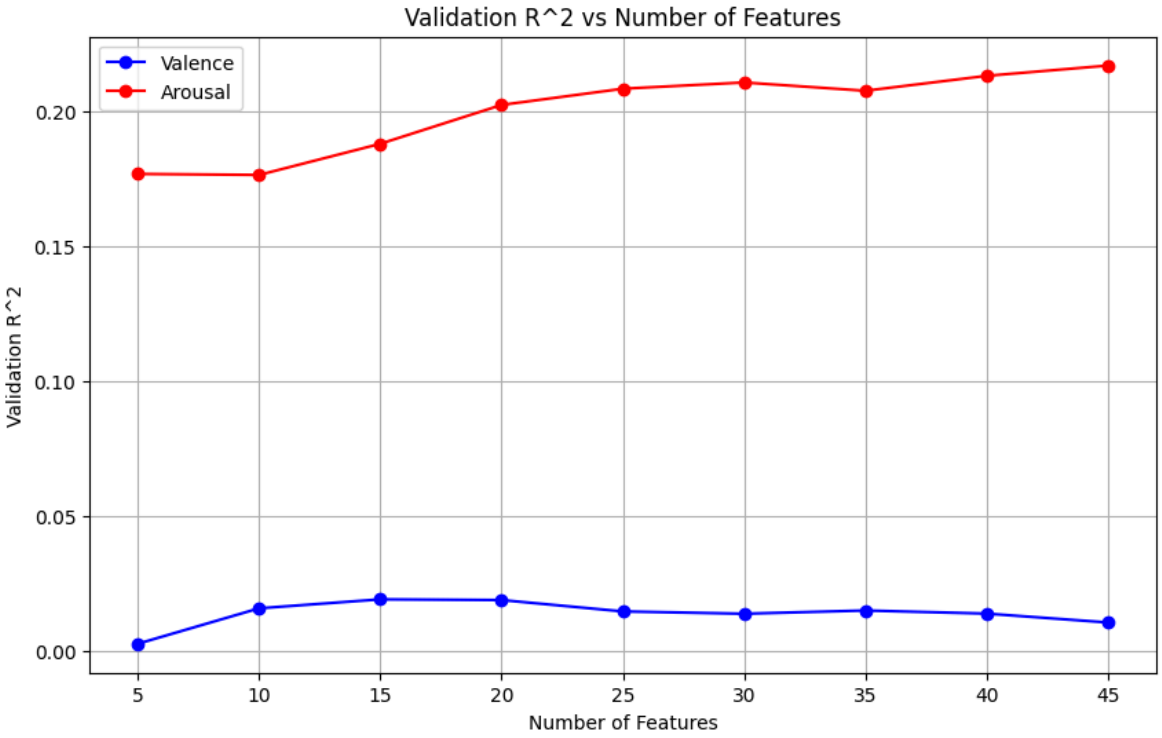
### RFE feature count sweep

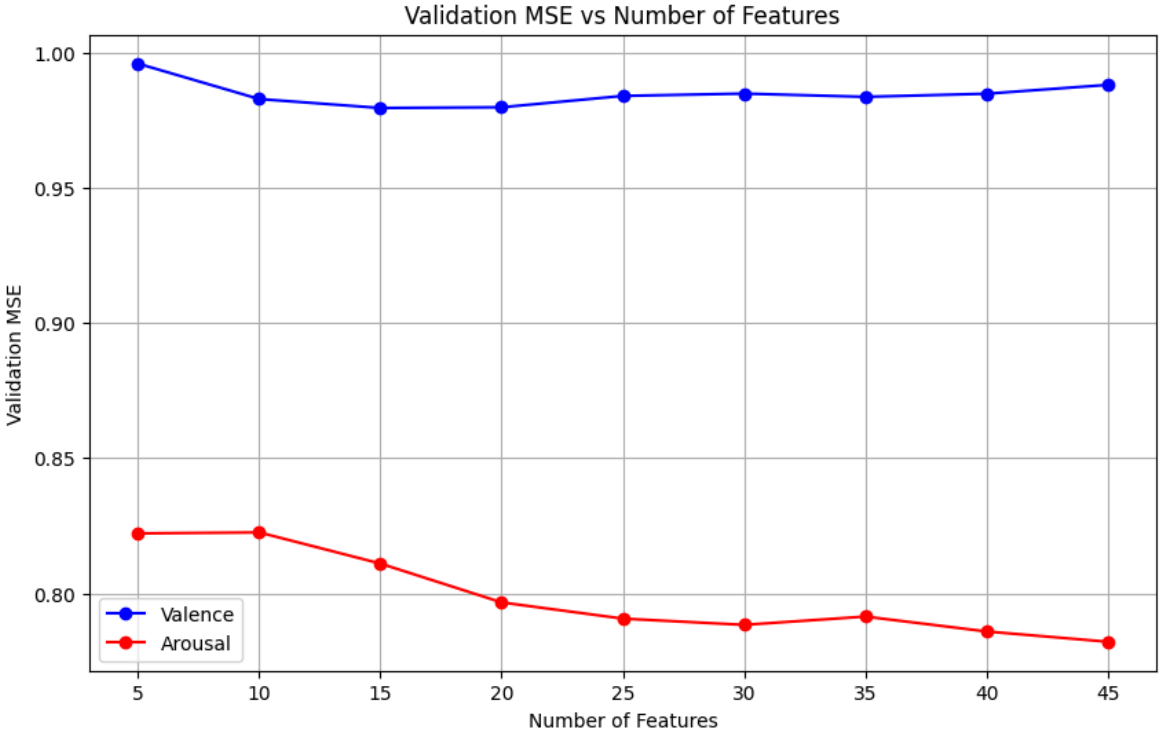
In [14]:
```
NUMBER_OF_SPLITS_AROUSAL = 10

param_arousal = {
    'n_estimators': 200,
    'max_depth': 5,
    'min_samples_split': 5,
    'min_samples_leaf': 1,
    'max_features': 'log2',
}
PCA_COMPONENTS_AROUSAL = 10


NUMBER_OF_SPLITS_VALANCE = 10
```

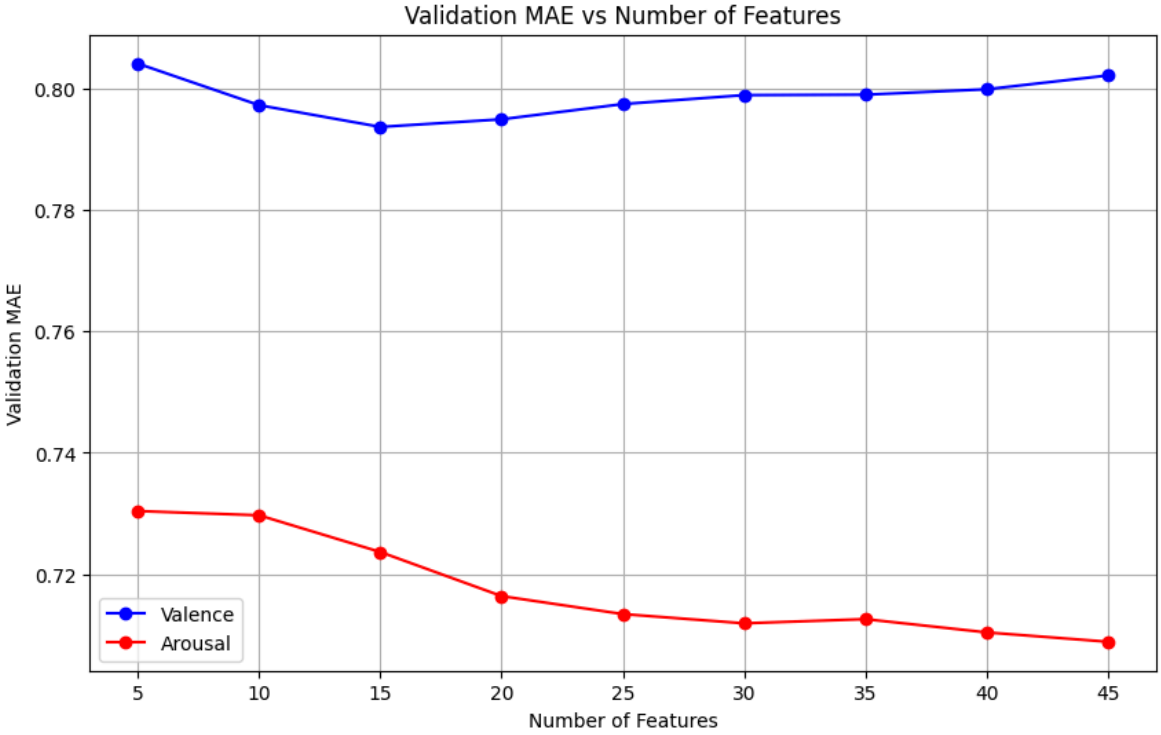
```
param_valence = {
    'n_estimators': 300,
    'max_depth': 5,
    'min_samples_split': 2,
    'min_samples_leaf': 5,
    'max_features': 'log2'
}
PCA_COMPONENTS_VALANCE = 10

r2s_valance_rfe = []
r2s_arousal_rfe = []
mses_valance_rfe = []
mses_arousal_rfe = []
maes_valance_rfe = []
maes_arousal_rfe = []

features_range = np.arange(5, 50, 5)

for features in features_range:

    print(f'Number of Features: {features}')

    results_valance_rfe = regression_cv_pipeline(X, y_median_valence, gro
    r2s_valance_rfe.append(results_valance_rfe['Val R^2'].mean())
    mses_valance_rfe.append(results_valance_rfe['Val MSE'].mean())
    maes_valance_rfe.append(results_valance_rfe['Val MAE'].mean())

    results_arousal_rfe = regression_cv_pipeline(X, y_median_arousal, gro
    r2s_arousal_rfe.append(results_arousal_rfe['Val R^2'].mean())
    mses_arousal_rfe.append(results_arousal_rfe['Val MSE'].mean())
    maes_arousal_rfe.append(results_arousal_rfe['Val MAE'].mean())

plt.figure(figsize=(10, 6))
plt.plot(features_range, r2s_valance_rfe, marker='o', linestyle='-', colo
plt.plot(features_range, r2s_arousal_rfe, marker='o', linestyle='-', colo
plt.xlabel('Number of Features')
plt.ylabel('Validation R^2')
plt.title('Validation R^2 vs Number of Features')
plt.legend()
plt.grid(True)
plt.show()

plt.figure(figsize=(10, 6))
plt.plot(features_range, mses_valance_rfe, marker='o', linestyle='-', col
plt.plot(features_range, mses_arousal_rfe, marker='o', linestyle='-', col
plt.xlabel('Number of Features')
plt.ylabel('Validation MSE')
plt.title('Validation MSE vs Number of Features')
plt.legend()
plt.grid(True)
plt.show()

plt.figure(figsize=(10, 6))
plt.plot(features_range, maes_valance_rfe, marker='o', linestyle='-', col
plt.plot(features_range, maes_arousal_rfe, marker='o', linestyle='-', col
plt.xlabel('Number of Features')
plt.ylabel('Validation MAE')
plt.title('Validation MAE vs Number of Features')
plt.legend()
```

```
plt.grid(True)
plt.show()

print(f'Best RFE Features for Valence: {features_range[r2s_valance_rfe.in
print(f'Best RFE Features for Arousal: {features_range[r2s_arousal_rfe.in
```

```
Number of Features: 5
Number of Features: 10
Number of Features: 15
Number of Features: 20
Number of Features: 25
Number of Features: 30
Number of Features: 35
Number of Features: 40
Number of Features: 45
```





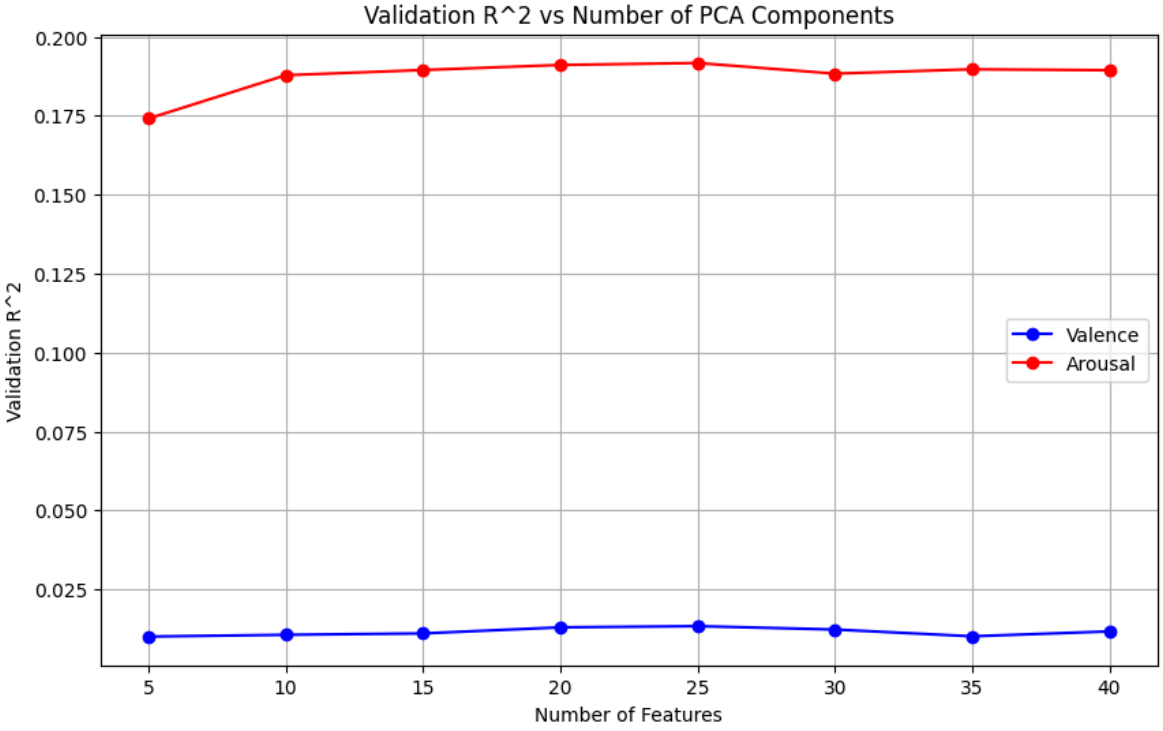

```
Best RFE Features for Valence: 15
Best RFE Features for Arousal: 45
```

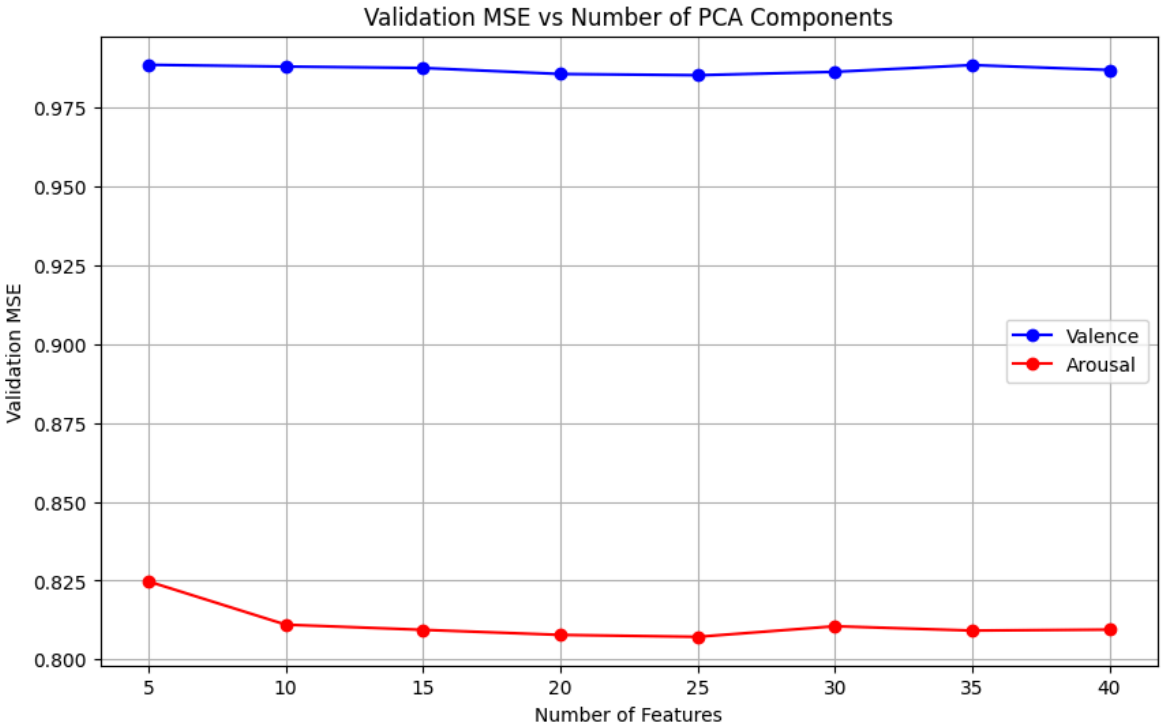
## PCA components sweep

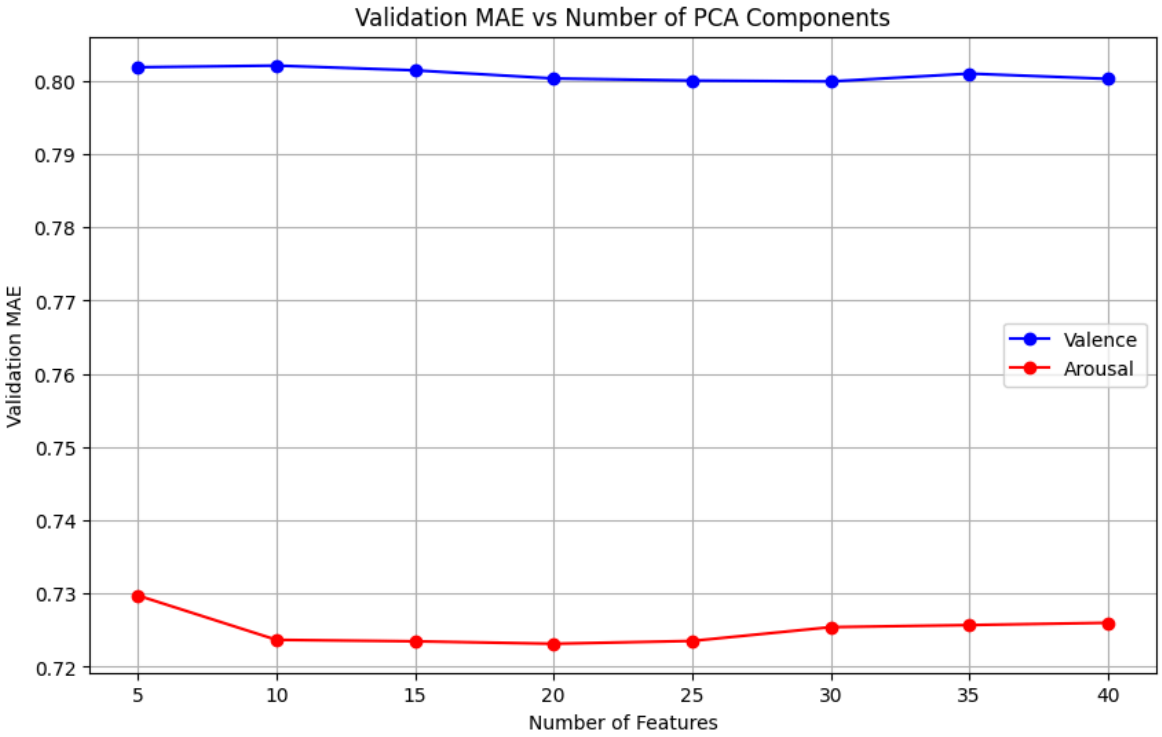
In [15]:
```python
NUMBER_OF_SPLITS_AROUSAL = 10
NUMBER_OF_RFE_FEATURES_AROUSAL = 15
param_arousal = {
    'n_estimators': 200,
    'max_depth': 5,
    'min_samples_split': 5,
    'min_samples_leaf': 1,
    'max_features': 'log2',
}

NUMBER_OF_SPLITS_VALANCE = 10
NUMBER_OF_RFE_FEATURES_VALENCE = 45
param_valence = {
    'n_estimators': 300,
    'max_depth': 5,
    'min_samples_split': 2,
    'min_samples_leaf': 5,
    'max_features': 'log2'
}

r2s_valance_pca = []
r2s_arousal_pca = []
mses_valance_pca = []
mses_arousal_pca = []
maes_valance_pca = []
maes_arousal_pca = []

pca_components_range = np.arange(5, 45, 5)

for pca_components in pca_components_range:
```

```
    print(f'PCA Components: {pca_components}')

    results_valance_pca = regression_cv_pipeline(X, y_median_valence, gro
    r2s_valance_pca.append(results_valance_pca['Val R^2'].mean())
    mses_valance_pca.append(results_valance_pca['Val MSE'].mean())
    maes_valance_pca.append(results_valance_pca['Val MAE'].mean())

    results_arousal_rfe = regression_cv_pipeline(X, y_median_arousal, gro
    r2s_arousal_pca.append(results_arousal_rfe['Val R^2'].mean())
    mses_arousal_pca.append(results_arousal_rfe['Val MSE'].mean())
    maes_arousal_pca.append(results_arousal_rfe['Val MAE'].mean())

plt.figure(figsize=(10, 6))
plt.plot(pca_components_range, r2s_valance_pca, marker='o', linestyle='-'
plt.plot(pca_components_range, r2s_arousal_pca, marker='o', linestyle='-'
plt.xlabel('Number of Features')
plt.ylabel('Validation R^2')
plt.title('Validation R^2 vs Number of PCA Components')
plt.legend()
plt.grid(True)
plt.show()

plt.figure(figsize=(10, 6))
plt.plot(pca_components_range, mses_valance_pca, marker='o', linestyle='-
plt.plot(pca_components_range, mses_arousal_pca, marker='o', linestyle='-
plt.xlabel('Number of Features')
plt.ylabel('Validation MSE')
plt.title('Validation MSE vs Number of PCA Components')
plt.legend()
plt.grid(True)
plt.show()

plt.figure(figsize=(10, 6))
plt.plot(pca_components_range, maes_valance_pca, marker='o', linestyle='-
plt.plot(pca_components_range, maes_arousal_pca, marker='o', linestyle='-
plt.xlabel('Number of Features')
plt.ylabel('Validation MAE')
plt.title('Validation MAE vs Number of PCA Components')
plt.legend()
plt.grid(True)
plt.show()

print(f'Best PCA Components for Valence: {pca_components_range[r2s_valanc
print(f'Best PCA Components for Arousal: {pca_components_range[r2s_arousa
```

```
PCA Components: 5
PCA Components: 10
PCA Components: 15
PCA Components: 20
PCA Components: 25
PCA Components: 30
PCA Components: 35
PCA Components: 40
```


Validation R^2 vs Number of PCA Components
Validation R^2
Number of Features
Valence
Arousal
0.200
0.175
0.150
0.125
0.100
0.075
0.050
0.025
5
10
15
20
25
30
35
40


Validation MSE vs Number of PCA Components
Validation MSE
Number of Features
Valence
Arousal
0.975
0.950
0.925
0.900
0.875
0.850
0.825
0.800
5
10
15
20
25
30
35
40



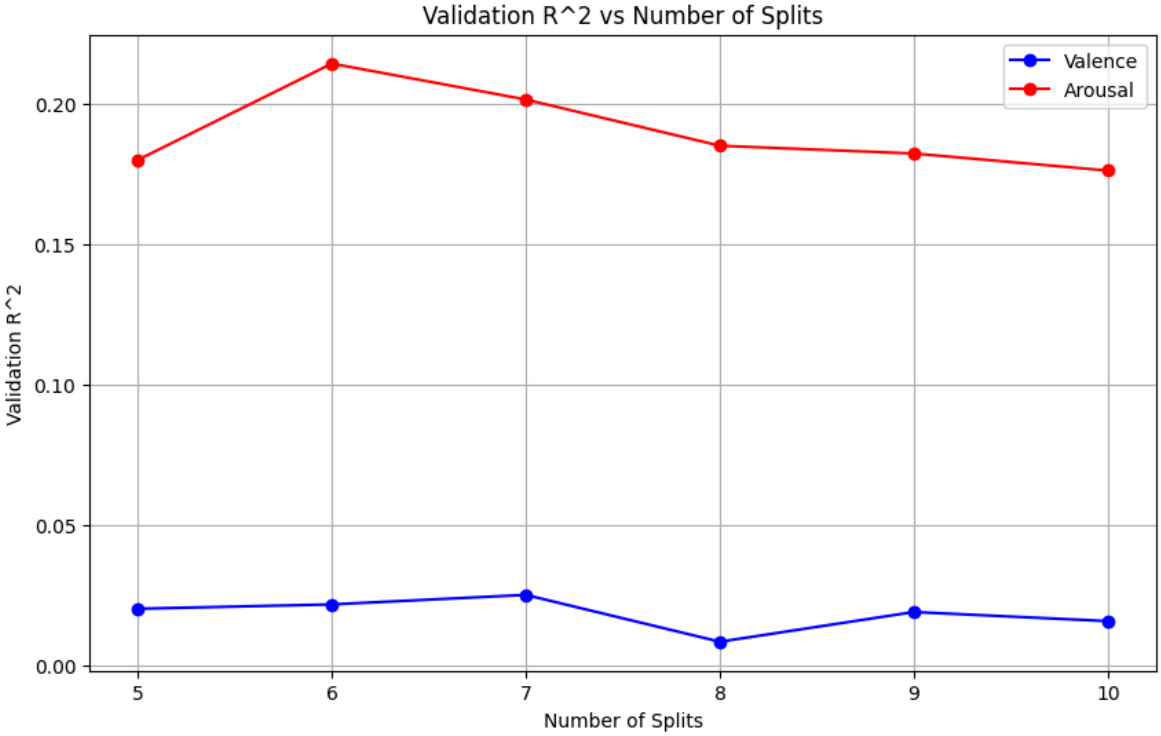

```
Best PCA Components for Valence: 25
Best PCA Components for Arousal: 25
```

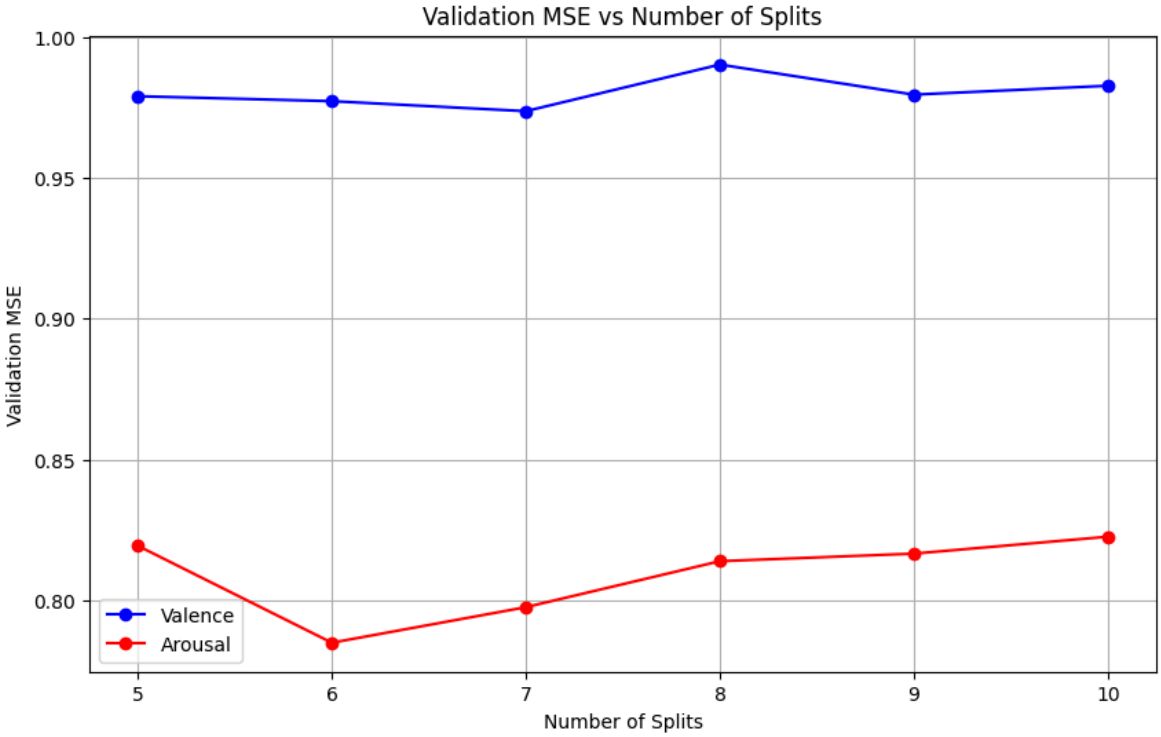
## Splits sweep

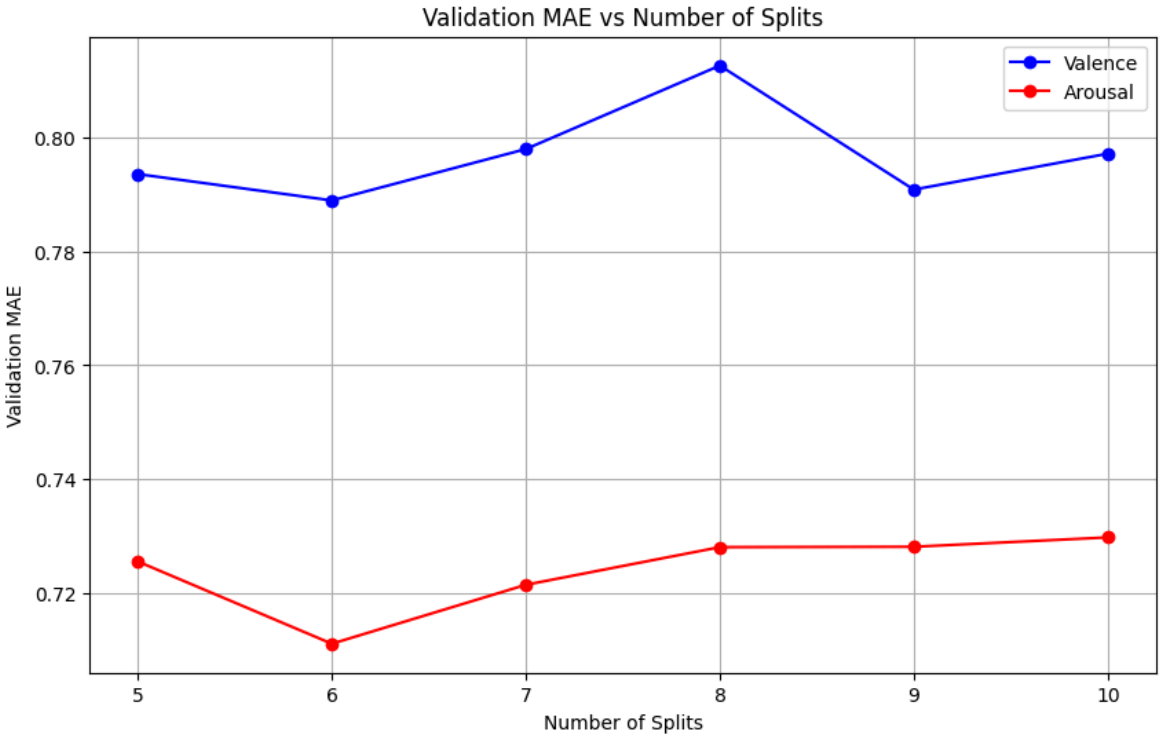
In [16]:
```python
NUMBER_OF_RFE_FEATURES_AROUSAL = 10
param_arousal = {
    'n_estimators': 200,
    'max_depth': 5,
    'min_samples_split': 5,
    'min_samples_leaf': 1,
    'max_features': 'log2',
}
PCA_COMPONENTS_AROUSAL = 10


NUMBER_OF_RFE_FEATURES_VALENCE = 10
param_valence = {
    'n_estimators': 300,
    'max_depth': 5,
    'min_samples_split': 2,
    'min_samples_leaf': 5,
    'max_features': 'log2'
}
PCA_COMPONENTS_VALANCE = 10

r2s_valance_splits = []
r2s_arousal_splits = []
mses_valance_splits = []
mses_arousal_splits = []
maes_valance_splits = []
maes_arousal_splits = []

splits_range = np.arange(5, 11, 1)
```

```
for splits in splits_range:

    print(f'Number of Splits: {splits}')

    results_valance_splits = regression_cv_pipeline(X, y_median_valence,
    r2s_valance_splits.append(results_valance_splits['Val R^2'].mean())
    mses_valance_splits.append(results_valance_splits['Val MSE'].mean())
    maes_valance_splits.append(results_valance_splits['Val MAE'].mean())

    results_arousal_splits = regression_cv_pipeline(X, y_median_arousal,
    r2s_arousal_splits.append(results_arousal_splits['Val R^2'].mean())
    mses_arousal_splits.append(results_arousal_splits['Val MSE'].mean())
    maes_arousal_splits.append(results_arousal_splits['Val MAE'].mean())

plt.figure(figsize=(10, 6))
plt.plot(splits_range, r2s_valance_splits, marker='o', linestyle='-', col
plt.plot(splits_range, r2s_arousal_splits, marker='o', linestyle='-', col
plt.xlabel('Number of Splits')
plt.ylabel('Validation R^2')
plt.title('Validation R^2 vs Number of Splits')
plt.legend()
plt.grid(True)
plt.show()

plt.figure(figsize=(10, 6))
plt.plot(splits_range, mses_valance_splits, marker='o', linestyle='-', co
plt.plot(splits_range, mses_arousal_splits, marker='o', linestyle='-', co
plt.xlabel('Number of Splits')
plt.ylabel('Validation MSE')
plt.title('Validation MSE vs Number of Splits')
plt.legend()
plt.grid(True)
plt.show()

plt.figure(figsize=(10, 6))
plt.plot(splits_range, maes_valance_splits, marker='o', linestyle='-', co
plt.plot(splits_range, maes_arousal_splits, marker='o', linestyle='-', co
plt.xlabel('Number of Splits')
plt.ylabel('Validation MAE')
plt.title('Validation MAE vs Number of Splits')
plt.legend()
plt.grid(True)
plt.show()

print(f'Best Splits for Valence: {splits_range[r2s_valance_splits.index(m
print(f'Best Splits for Arousal: {splits_range[r2s_arousal_splits.index(m
```

```
Number of Splits: 5
Number of Splits: 6
Number of Splits: 7
Number of Splits: 8
Number of Splits: 9
Number of Splits: 10
```


Validation R^2 vs Number of Splits
Valence
Arousal
Validation R^2
0.00
0.05
0.10
0.15
0.20
5
6
7
8
9
10
Number of Splits


Validation MSE vs Number of Splits
Validation MSE
0.80
0.85
0.90
0.95
1.00
Valence
Arousal
5
6
7
8
9
10
Number of Splits




```
Best Splits for Valence: 7
Best Splits for Arousal: 6
```

# Hyperparameter optimization

In [17]:
```
# from itertools import product
# import time
# import random

# NUMBER_TESTS_AROUSAL = 100
# NUMBER_TESTS_VALENCE = 100


# ## arousal hyperparameter tuning
# start_time = time.time()


# df_arousal_hyperparameter_results = pd.DataFrame(columns=['n_estimators

# n_estimators = [50, 100, 200]
# max_depth = [None, 10, 20]
# min_samples_split = [2, 5]
# min_samples_leaf = [1, 2]
# max_features = [None, 'sqrt', 'log2']
# rfe_features = [40, 45, 50]
# pca_components = [20,25,30]
# splits = [5,6,7]

# param_combinations = product(n_estimators, max_depth, min_samples_split
# param_list = list(param_combinations)
# n_combinations = len(param_list)
# count_combinations = 0

# print(f'Total hyperparameter combinations to test: {n_combinations}')
# print(len(param_list))
```

```
# random_combinations = random.sample(param_list, k=NUMBER_OF_SPLITS_AROU

# for param in random_combinations:

#     count_combinations += 1

#     param = {
#         'n_estimators': param[0],
#         'max_depth': param[1],
#         'min_samples_split': param[2],
#         'min_samples_leaf': param[3],
#         'max_features': param[4],
#         'rfe_features': param[5],
#         'pca_components': param[6],
#         'splits': param[7]
#     }

#     results_file_name = f"results/hyperparam/arousal_hyperparameter_res

#     results = regression_cv_pipeline(X, y_median_arousal, groups, split
#     mean_train_r2 = results['Train R^2'].mean()
#     mean_val_r2 = results['Val R^2'].mean()

#     results_dict = {
#         'n_estimators': param['n_estimators'],
#         'max_depth': param['max_depth'],
#         'min_samples_split': param['min_samples_split'],
#         'min_samples_leaf': param['min_samples_leaf'],
#         'max_features': param['max_features'],
#         'rfe_features': param['rfe_features'],
#         'pca_components': param['pca_components'],
#         'splits': param['splits'],
#         'Train R^2': mean_train_r2,
#         'Val R^2': mean_val_r2
#     }

#     print(f'Processing hyperparameter combination {count_combinations}

#     if count_combinations == 1:
#         df_arousal_hyperparameter_results = pd.DataFrame([results_dict]
#     else:
#         df_arousal_hyperparameter_results = pd.concat([df_arousal_hyper

# df_arousal_hyperparameter_results.to_csv('results/arousal_hyperparamete

# end_time_arousal = time.time()


# ## valence hyperparameter tuning

# df_valence_hyperparameter_results = pd.DataFrame(columns=['n_estimators

# n_estimators = [100, 200, 300]
# max_depth = [None, 10, 20]
# min_samples_split = [2, 5, 10]
# min_samples_leaf = [1, 3, 5]
# max_features = [None, 'sqrt', 'log2']
# rfe_features = [10, 15, 20]
# pca_components = [15, 20]
```

```
# splits = [5,6,7]

# param_combinations = product(n_estimators, max_depth, min_samples_split
# param_list = list(param_combinations)
# n_combinations = len(param_list)
# count_combinations = 0

# random_combinations = random.sample(param_list, k=NUMBER_TESTS_VALENCE)

# for param in random_combinations:

#     count_combinations += 1

#     param = {
#         'n_estimators': param[0],
#         'max_depth': param[1],
#         'min_samples_split': param[2],
#         'min_samples_leaf': param[3],
#         'max_features': param[4],
#         'rfe_features': param[5],
#         'pca_components': param[6],
#         'splits': param[7]
#     }

#     results_file_name = f"results/hyperparam/valence_hyperparameter_res

#     results = regression_cv_pipeline(X, y_median_valence, groups, split
#     mean_train_r2 = results['Train R^2'].mean()
#     mean_val_r2 = results['Val R^2'].mean()

#     results_dict = {
#         'n_estimators': param['n_estimators'],
#         'max_depth': param['max_depth'],
#         'min_samples_split': param['min_samples_split'],
#         'min_samples_leaf': param['min_samples_leaf'],
#         'max_features': param['max_features'],
#         'rfe_features': param['rfe_features'],
#         'pca_components': param['pca_components'],
#         'splits': param['splits'],
#         'Train R^2': mean_train_r2,
#         'Val R^2': mean_val_r2
#     }

#     print(f'Processing hyperparameter combination {count_combinations}

#     if count_combinations == 1:
#         df_valence_hyperparameter_results = pd.DataFrame([results_dict]
#     else:
#         df_valence_hyperparameter_results = pd.concat([df_valence_hyper

# df_valence_hyperparameter_results.to_csv('results/valence_hyperparamete

# end_time_valence = time.time()

# print(f'Total time for arousal hyperparameter tuning: {end_time_arousal
# print(f'Total time for valence hyperparameter tuning: {end_time_valence
```

In [18]:
```
df_hyperpar_res_arousal = pd.read_csv('results/arousal_hyperparameter_res
min_r_squared_rows = df_hyperpar_res_arousal.nlargest(6, 'Val R^2')
```

```
print('Rows with Largest R^2 for arousal')
print(min_r_squared_rows)


df_hyperpar_res_valence = pd.read_csv('results/valence_hyperparameter_res
min_r_squared_rows = df_hyperpar_res_valence.nlargest(6, 'Val R^2')
print('Rows with Largest R^2 for valence')
print(min_r_squared_rows)
```

```
Rows with Largest R^2 for arousal
   n_estimators  max_depth  min_samples_split  min_samples_leaf max_featur
es  \
6           200        NaN                  5                 2         sq
rt
4           100        NaN                  2                 1         sq
rt
8           200       20.0                  2                 1         lo
g2
3           100        NaN                  5                 2         lo
g2
5            50        NaN                  5                 2         lo
g2
7           100        NaN                  2                 1         lo
g2

   rfe_features  pca_components  splits  Train R^2   Val R^2
6            45              25       7   0.902221  0.255564
4            45              20       6   0.932309  0.252071
8            40              25       7   0.926192  0.251022
3            40              30       6   0.892287  0.247257
5            40              25       7   0.889930  0.245317
7            50              25       5   0.930905  0.233702
Rows with Largest R^2 for valence
    n_estimators  max_depth  min_samples_split  min_samples_leaf max_featu
res  \
26          300       10.0                 10                 3         s
qrt
9           200       10.0                 10                 1         s
qrt
58          300       10.0                  5                 1         s
qrt
99          300       10.0                  2                 1         l
og2
74          300       20.0                  5                 5         l
og2
67          300       10.0                  5                 3         l
og2

    rfe_features  pca_components  splits  Train R^2   Val R^2
26            20              15       7   0.472071  0.040162
9             20              20       7   0.487624  0.039739
58            20              15       7   0.507406  0.038194
99            20              20       7   0.514930  0.037793
74            20              15       7   0.675368  0.037362
67            15              15       7   0.466181  0.037340
```

# Final model preparation

In [19]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import GroupShuffleSplit
from sklearn.linear_model import LinearRegression
from sklearn.feature_selection import RFE
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_e
from sklearn.decomposition import PCA
from scipy.stats import zscore

def run_final_model(df_raw, target_col='median_valence', rfe_features=30,

    # outlier removal
    df_clean = df_raw.copy()
    z_thresh = 3
    df_clean['z_arousal'] = zscore(df_clean['median_arousal'])
    df_clean['z_valence'] = zscore(df_clean['median_valence'])
    df_clean = df_clean[
        (df_clean['z_arousal'].abs() <= z_thresh) &
        (df_clean['z_valence'].abs() <= z_thresh)
    ]

    df_clean = df_clean.drop(columns=['z_arousal', 'z_valence'])

    print(f'Original dataset shape: {df_raw.shape}')
    print(f'Cleaned dataset shape: {df_clean.shape}')

    # Normalized target
    y = (df_clean[target_col] - df_clean[target_col].mean()) / df_clean[t
    X = df_clean.drop(columns=['Participant', 'median_arousal', 'median_v

    print(X.shape)

    # scale features for rfe as it uses
    scaler = StandardScaler()
    X_scaled = pd.DataFrame(scaler.fit_transform(X), columns=X.columns, i

    estimator = LinearRegression()
    rfe_selector = RFE(estimator=estimator, n_features_to_select=rfe_feat
    rfe_selector.fit(X_scaled, y)
    selected_features_mask = rfe_selector.support_
    current_selected_features = X_scaled.columns[selected_features_mask].

    X_selected = X[current_selected_features]


    if pd.isnull(X_scaled).any().any():
        raise ValueError('X_scaled contains NaN values, which PCA cannot

    #pca
    pca = PCA(n_components=pca_components)
    X_pca = pca.fit_transform(X_selected)

    print(X_pca.shape, X_selected.shape)

    X_selected = np.concatenate([X_selected, X_pca], axis=1)

    # Model configuration
```

```
    if model_params is None:
        model_params = {
            'n_estimators': 100,
            'max_depth': None,
            'min_samples_split': 2,
            'min_samples_leaf': 1,
            'max_features': None
        }

    n_estimators = model_params['n_estimators']
    max_depth = model_params['max_depth']
    min_samples_split = model_params['min_samples_split']
    min_samples_leaf = model_params['min_samples_leaf']
    max_features = model_params['max_features']

    model = RandomForestRegressor(
        n_estimators=n_estimators,
        max_depth=max_depth,
        min_samples_split=min_samples_split,
        min_samples_leaf=min_samples_leaf,
        max_features=max_features,
        random_state=42)
    model.fit(X_selected, y)

    y_pred = model.predict(X_selected)
    r2 = r2_score(y, y_pred)
    mse = mean_squared_error(y, y_pred)
    mae = mean_absolute_error(y, y_pred)

    print(f'Final Model Evaluation for {target_col}:')
    print(f'R^2: {r2:.4f}')
    print(f'MSE: {mse:.4f}')
    print(f'MAE: {mae:.4f}')


run_final_model(
    df_raw,
    target_col='median_arousal',
    pca_components=25,
    rfe_features=45,
    model_params={
        'n_estimators': 200,
        'max_depth': None,
        'min_samples_split': 5,
        'min_samples_leaf': 2,
        'max_features': 'sqrt'
    }
)

run_final_model(
    df_raw,
    target_col='median_valence',
    pca_components=10,
    rfe_features=15,
    model_params={
        'n_estimators': 300,
        'max_depth': 10,
        'min_samples_split': 5,
        'min_samples_leaf': 1,
        'max_features': 'log2'
```

```
    }
)
```

```
Original dataset shape: (7238, 133)
Cleaned dataset shape: (7216, 133)
(7216, 130)
(7216, 25) (7216, 45)
Final Model Evaluation for median_arousal:
R^2: 0.9058
MSE: 0.0942
MAE: 0.2349
Original dataset shape: (7238, 133)
Cleaned dataset shape: (7216, 133)
(7216, 130)
(7216, 10) (7216, 15)
Final Model Evaluation for median_valence:
R^2: 0.4824
MSE: 0.5175
MAE: 0.5695
```

A common approach in affect modelling is arousal/valence binarization. Towards this direction, a threshold value is defined and emotional states with arousal/valence values larger than or equal to that threshold are denoted as “high” arousal/valence, while emotional states with arousal/valence value lower than the threshold are denoted as “low” arousal/valence.

Select and justify appropriate threshold values for binarizing both arousal and valence annotations (the threshold for binarizing arousal should not necessarily be equal to the threshold for binarizing valence).

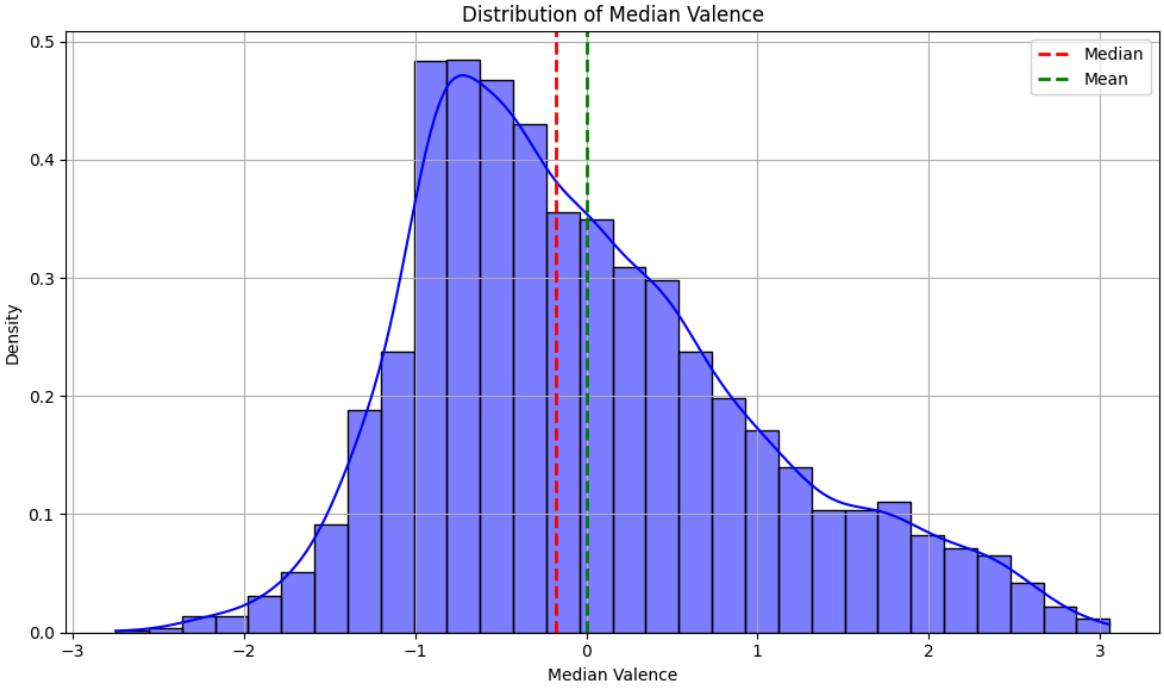
## Threshold Determination

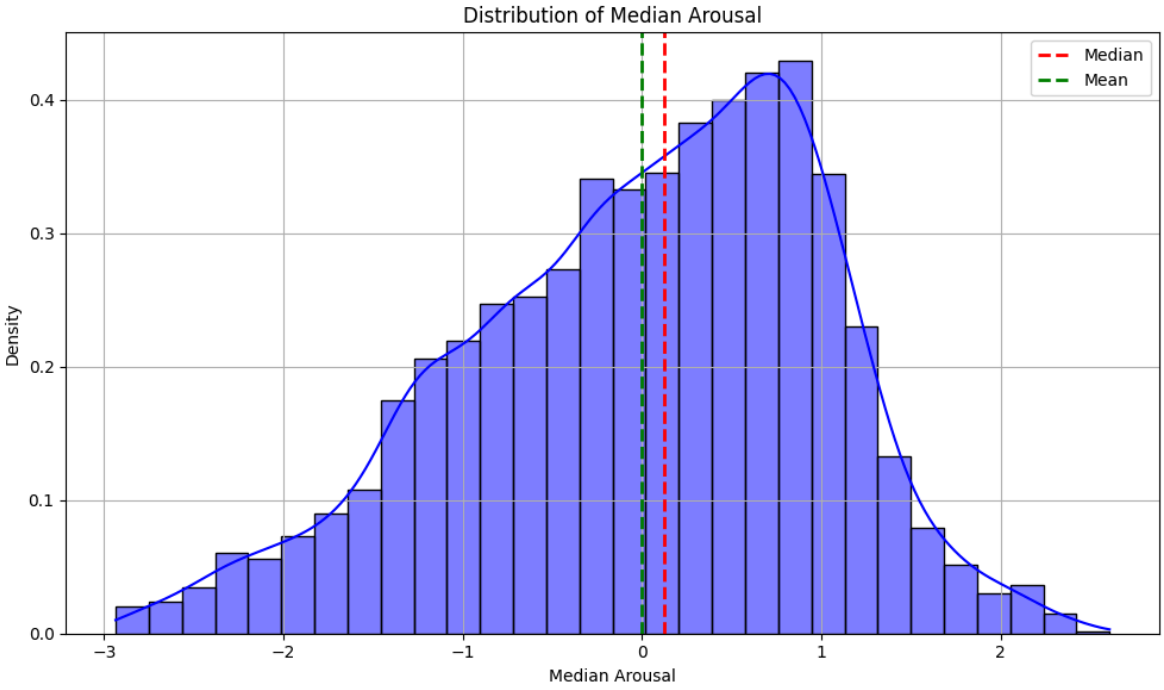
This discussion will start by looking at the median values of arousal and valence

In [20]:
```
df_binary_raw = df_raw.copy()

print(f'Original dataset shape: {df_binary_raw.shape}')
# remove the oputliers
z_thresh = 3
df_binary_raw['z_arousal'] = zscore(df_binary_raw['median_arousal'])
df_binary_raw['z_valence'] = zscore(df_binary_raw['median_valence'])
df_binary_clean = df_binary_raw[
    (df_binary_raw['z_arousal'].abs() <= z_thresh) &
    (df_binary_raw['z_valence'].abs() <= z_thresh)
]

df_binary_clean = df_binary_clean.drop(columns=['z_arousal', 'z_valence']

print(f'Cleaned dataset shape: {df_binary_clean.shape}')
```

```
# norm, z
median_valance_mean = df_binary_clean['median_valence'].mean()
median_valance_std = df_binary_clean['median_valence'].std()
median_arousal_mean = df_binary_clean['median_arousal'].mean()
median_arousal_std = df_binary_clean['median_arousal'].std()

df_binary_clean['median_valence'] = (df_binary_clean['median_valence'] -
df_binary_clean['median_arousal'] = (df_binary_clean['median_arousal'] -

median_valance_median = df_binary_clean['median_valence'].median()
median_valance_mean = df_binary_clean['median_valence'].mean()
median_valance_std = df_binary_clean['median_valence'].std()

median_arousal_median = df_binary_clean['median_arousal'].median()
median_arousal_mean = df_binary_clean['median_arousal'].mean()
median_arousal_std = df_binary_clean['median_arousal'].std()

#check
# print(f'Median Valence: {median_valance_median}')
# print(f'Mean Valence: {median_valance_mean}')
# print(f'Standard Deviation Valence: {median_valance_std}')
# print(f'Median Arousal: {median_arousal_median}')
# print(f'Mean Arousal: {median_arousal_mean}')
# print(f'Standard Deviation Arousal: {median_arousal_std}')

plt.figure(figsize=(10, 6))
sns.histplot(df_binary_clean['median_valence'], bins=30, kde=True, color=
plt.xlabel('Median Valence')
plt.ylabel('Density')
plt.title('Distribution of Median Valence')
plt.axvline(median_valance_median, color='red', linestyle='dashed', linew
plt.axvline(median_valance_mean, color='green', linestyle='dashed', linew
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()

# median_arousal distribution
plt.figure(figsize=(10, 6))
sns.histplot(df_binary_clean['median_arousal'], bins=30, kde=True, color=
plt.xlabel('Median Arousal')
plt.ylabel('Density')
plt.title('Distribution of Median Arousal')
plt.axvline(median_arousal_median, color='red', linestyle='dashed', linew
plt.axvline(median_arousal_mean, color='green', linestyle='dashed', linew
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()

print(f' for median_valence: median={median_valance_median:.4f}, mean={me
print(f' for median_arousal: median={median_arousal_median:.4f}, mean={me
```

```
Original dataset shape: (7238, 133)
Cleaned dataset shape: (7216, 133)
```




```
for median_valence: median=-0.1804, mean=-0.0000
for median_arousal: median=0.1271, mean=0.0000
```

To ensure robustness, threshold determination was computed after removing anomalies across the dataset.

After standardizing the distributions, the statistical properties of `median_valence` and `median_arousal` are analyzed.

For `median_valence`, the distribution has noticeable right-skewed, with a peak just below zero and a long tail extending toward higher values. The mean is positioned to the right of the median, reflecting this skew. Choosing the median (−0.1804) as the threshold ensures a more balanced binarization between the 'low' and 'high' valence classes.

For median_arousal, the distribution is more symmetric but still slightly skewed right. The median (0.1271) and mean are closer than the valance case, indicating a closer-to-normal distribution. In this case, it is interesting to use either the mean or median for thresholding, though again, the median offers a statistically robust and balance-preserving option.

In summary, both thresholds should ideally be based on the medians of the respective distributions, especially for valence, where the skew is more prominent, ensuring an equitable split between the binary categories.

## Binarization

This cell converts the continuous median_valence and median_arousal scores into binary classification labels by applying the thresholds. Values greater than or equal to the threshold are labeled as high, and values below as low.

In [21]:
```
valance_threshold = median_valance_median
arousal_threshold = median_arousal_median

df_binary_clean['valence_binary'] = np.where(df_binary_clean['median_vale
df_binary_clean['arousal_binary'] = np.where(df_binary_clean['median_arou

# check
valence_ones = df_binary_clean['valence_binary'].sum()
print(f'Number of ones in valence_binary: {valence_ones} of {len(df_binar

# check
arousal_ones = df_binary_clean['arousal_binary'].sum()
print(f'Number of ones in arousal_binary: {arousal_ones} of {len(df_binar
```

```
Number of ones in valence_binary: 3610 of 7216
Number of ones in arousal_binary: 3608 of 7216
```

Implement a predictive model for each binarized response variable.

Select appropriate metrics to evaluate the performance of the model in this scenario using the validation protocol you proposed in Task 1.

In [22]:
```
from sklearn.model_selection import StratifiedGroupKFold
from sklearn.linear_model import LogisticRegression
from sklearn.feature_selection import RFE
from sklearn.metrics import confusion_matrix, f1_score, accuracy_score, p
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import VotingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_auc_score
from sklearn.metrics import precision_recall_curve
from sklearn.metrics import roc_curve
```

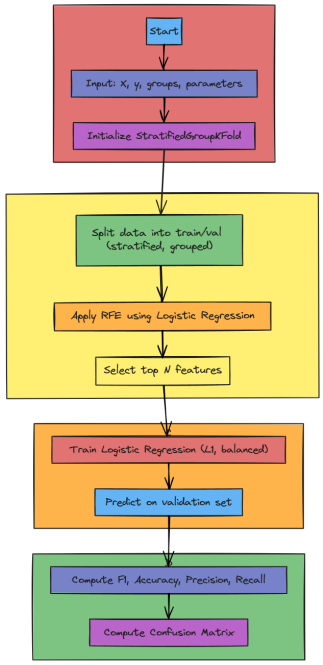
## Preparation of Features and Labels

This cell prepares the input features, binary targets, and group identifiers for the classification models:

In [23]:
```
X_binary = df_binary_clean.drop(columns=['Participant', 'median_arousal',
y_arousal_binary = df_binary_clean['arousal_binary']
y_valence_binary = df_binary_clean['valence_binary']
groups_binary = df_binary_clean['Participant']
```

# Binary Classification Pipeline

This binary classification pipeline is designed to evaluate model performance using




1. **Stratified Group K-Fold Cross-Validation**, ensuring that the distribution of target classes is preserved across folds while preventing data leakage from grouped observations.
2. For each fold, the pipeline first applies **Recursive Feature Elimination** with logistic regression to automatically select a subset of the most predictive features.
3. It then trains a **Logistic Regression model** with L1 regularization and class balancing to handle potentially imbalanced datasets.
4. Predictions are generated for the validation set, and performance is assessed using standard classification metrics including F1 score, accuracy, precision, recall, and a confusion matrix breakdown (true negatives, false positives, false negatives, and true positives).

5. The results from each fold are collected and saved, while all true and predicted labels, as well as predicted probabilities are returned for downstream analysis or visualization.

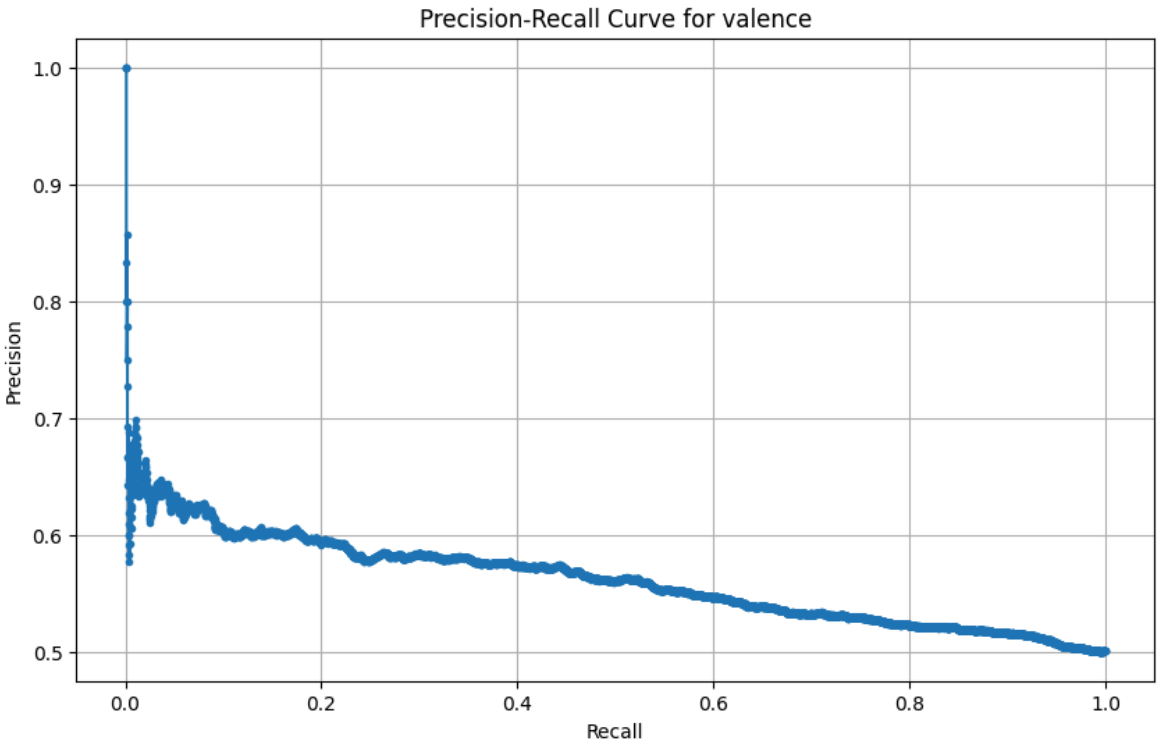
In [24]:
```
def classification_cv_pipeline(X, y, groups, rfe_features=10, splits=10,

    results_df = pd.DataFrame(columns=[
        'Fold', 'F1', 'Accuracy', 'Precision', 'Recall', 'True Negative',
    ])

    sgkf = StratifiedGroupKFold(n_splits=splits, shuffle=True, random_sta

    y_val_all = []
    y_pred_all = []
    y_proba_all = []

    fold = 1
    for train_idx, val_idx in sgkf.split(X, y, groups):
        # print(f'Fold {fold}:')

        X_train, X_val = X.iloc[train_idx], X.iloc[val_idx]
        y_train, y_val = y.iloc[train_idx], y.iloc[val_idx]

        # RFE
        rfe_estimator = LogisticRegression(solver='liblinear', max_iter=1
        rfe = RFE(estimator=rfe_estimator, n_features_to_select=rfe_featu
        rfe.fit(X_train, y_train)
        selected_features = X_train.columns[rfe.support_]
        X_train_selected = X_train[selected_features]
        X_val_selected = X_val[selected_features]


        model = LogisticRegression(solver='liblinear', max_iter=1000,  pe
        # random_forest = RandomForestClassifier(n_estimators=100, class_

        # model = VotingClassifier(
        # estimators=[
        #         ('lr', logistic),
        #         ('rf', random_forest)
        #     ],
        #     voting='hard'  # majority class
        # )


        model.fit(X_train_selected, y_train)
        # Predict
        y_pred = model.predict(X_val_selected)
        y_proba = model.predict_proba(X_val_selected)

        y_val_all.append(y_val)
        y_pred_all.append(y_pred)
        y_proba_all.append(y_proba)


        f1 = f1_score(y_val, y_pred)
        acc = accuracy_score(y_val, y_pred)
        cm = confusion_matrix(y_val, y_pred)
```

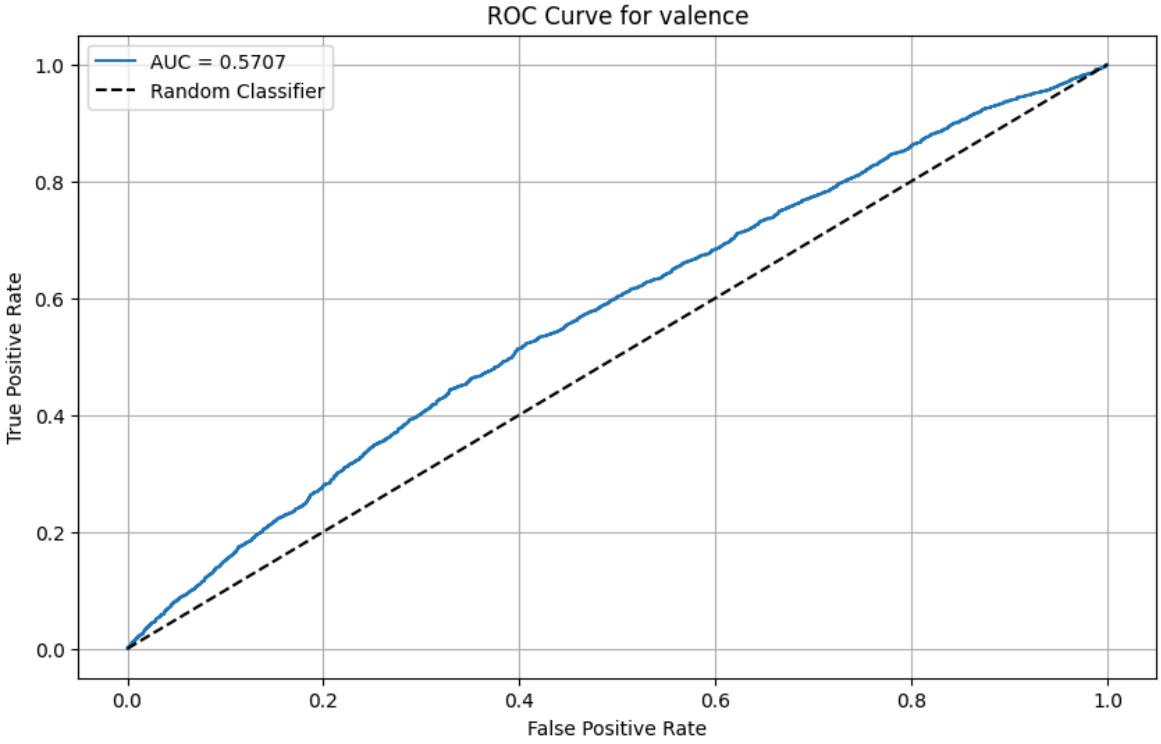
```
        precision = precision_score(y_val, y_pred)
        recall = recall_score(y_val, y_pred)

        results_line = {
            'Fold': fold,
            'F1': f1,
            'Accuracy': acc,
            'Precision': precision,
            'Recall': recall,
            'True Negative': cm[0, 0],
            'False Positive': cm[0, 1],
            'False Negative': cm[1, 0],
            'True Positive': cm[1, 1],
        }

        if fold == 1:
            results_df = pd.DataFrame([results_line])
        else:
            results_df = pd.concat([results_df, pd.DataFrame([results_lin

        fold += 1

    # save the results dataframe in a file
    results_df.to_csv(results_path, index=False)
    return results_df, y_val_all, y_pred_all, y_proba_all
```

# Evaluation

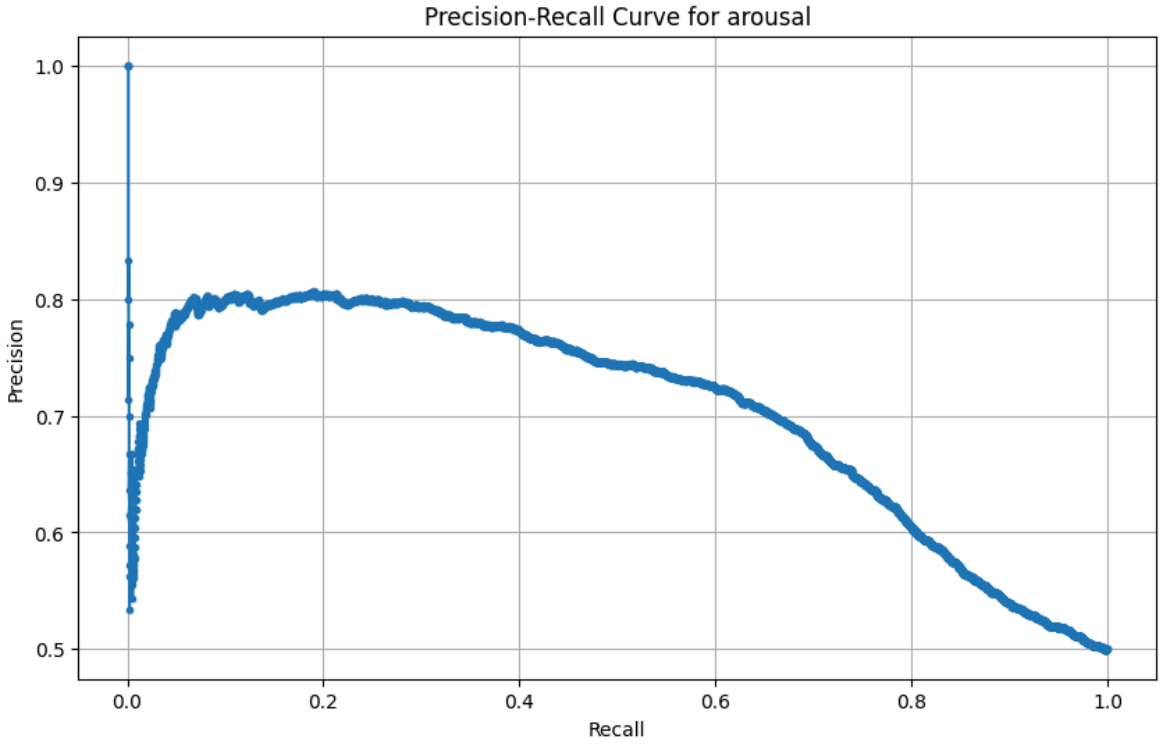
In [25]:
```
def classification_model_evaluation(X, y, groups, name, rfe_features=10,

    results, y_val_all, y_pred_all, y_proba_all = classification_cv_pipel
        X,
        y,
        groups,
        rfe_features=rfe_features,
        splits=splits,
        results_path=results_path
    )

    import numpy as np

    y_val_all_flat = np.concatenate(y_val_all)
    y_proba_all_flat = np.concatenate([proba[:, 1] for proba in y_proba_a

    precision_curve, recall_curve, _ = precision_recall_curve(y_val_all_f
    plt.figure(figsize=(10, 6))
    plt.plot(recall_curve, precision_curve, marker='.')
    plt.title(f'Precision-Recall Curve for {name}')
    plt.xlabel('Recall')
    plt.ylabel('Precision')
    plt.grid(True)
    plt.show()

    roc_auc = roc_auc_score(y_val_all_flat, y_proba_all_flat)
    fpr, tpr, _ = roc_curve(y_val_all_flat, y_proba_all_flat)
```

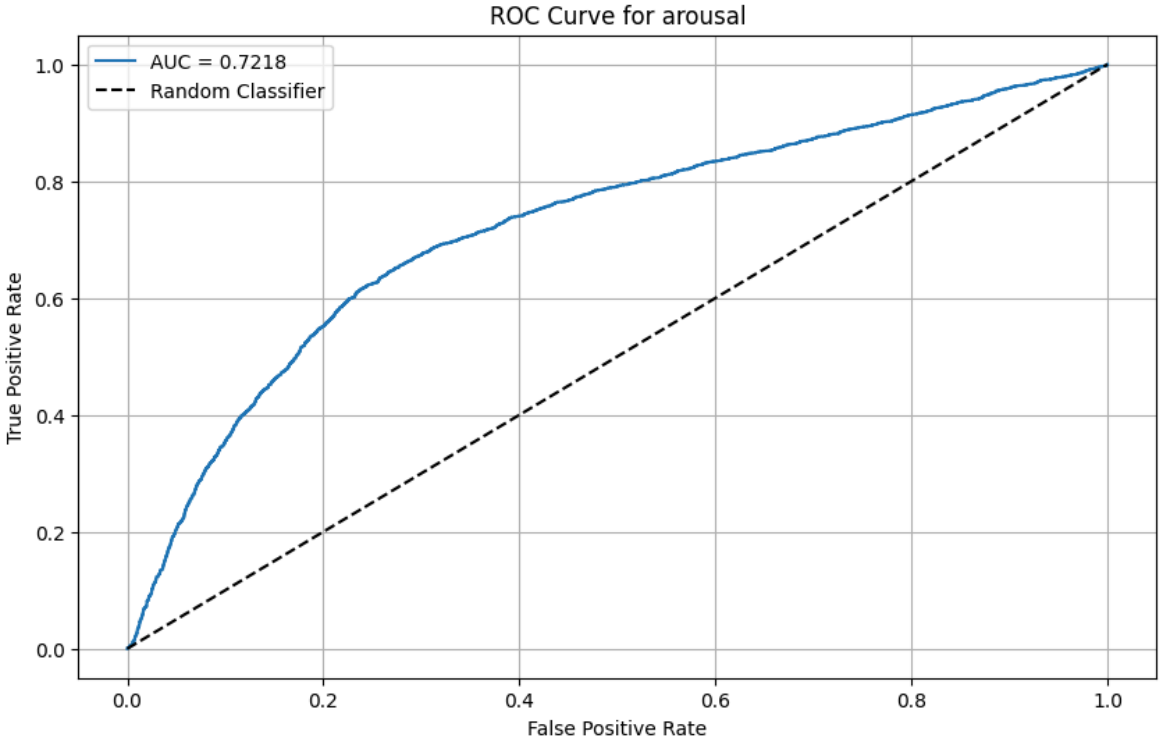
```
    plt.figure(figsize=(10, 6))
    plt.plot(fpr, tpr, label=f'AUC = {roc_auc:.4f}')
    plt.plot([0, 1], [0, 1], 'k--', label='Random Classifier')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title(f'ROC Curve for {name}')
    plt.legend()
    plt.grid(True)
    plt.show()

    avg_f1 = results['F1'].mean()
    avg_acc = results['Accuracy'].mean()
    avg_precision = results['Precision'].mean()
    avg_recall = results['Recall'].mean()
    avg_tn = results['True Negative'].mean()
    avg_fp = results['False Positive'].mean()
    avg_fn = results['False Negative'].mean()
    avg_tp = results['True Positive'].mean()

    print(f'Results for {name}')
    print(f'Average F1: {avg_f1:.4f}')
    print(f'Average Accuracy: {avg_acc:.4f}')
    print(f'Average Precision: {avg_precision:.4f}')
    print(f'Average Recall: {avg_recall:.4f}')
    print(f'Average ROC-AUC: {roc_auc:.4f}')
    print(f'Average True Negative: {avg_tn:.4f}')
    print(f'Average False Positive: {avg_fp:.4f}')
    print(f'Average False Negative: {avg_fn:.4f}')
    print(f'Average True Positive: {avg_tp:.4f}')

classification_model_evaluation(X_binary, y_valence_binary, groups_binary
classification_model_evaluation(X_binary, y_arousal_binary, groups_binary
```






```
Results for valence
Average F1: 0.5069
Average Accuracy: 0.5557
Average Precision: 0.5568
Average Recall: 0.5535
Average ROC-AUC: 0.5707
Average True Negative: 209.0000
Average False Positive: 151.6000
Average False Negative: 169.0000
Average True Positive: 192.0000
```






```
Results for arousal
Average F1: 0.6608
Average Accuracy: 0.6881
Average Precision: 0.6986
Average Recall: 0.6598
Average ROC-AUC: 0.7218
Average True Negative: 265.6000
Average False Positive: 95.2000
Average False Negative: 129.9000
Average True Positive: 230.9000
```

## Classification Evaluation Metrics

The classification performance was evaluated using multiple metrics:

- F1 Score
- Accuracy
- Precision
- Recall
- ROC-AUC
- confusion matrix components

Evaluation was conducted using **StratifiedGroupKFold** to ensure participant-independent validation. Both **precision-recall** and **ROC curves** were plotted to assess discriminative ability and decision confidence, with results reported separately for arousal and valence.

## Arousal Classification

### F1 Score

- F1 = **0.6608** indicates strong balance between precision and recall.

- This shows that the model is effectively identifying high-arousal states while minimizing false predictions.

**Accuracy**

- Accuracy = **0.6881**, well above random baseline (0.5), confirming solid classification performance.

**Precision & Recall**

- Precision = **0.6986**, Recall = **0.6598** show that most predicted positives are correct, and most true positives are successfully detected.

**ROC-AUC**

- AUC = **0.7218**, indicating good class separability.
- The ROC curve shows consistent advantage over random classification, with good early separation.

**PR Curve**

- The precision-recall curve has a high area under the curve, reflecting strong confidence in predictions across thresholds.

**Confusion Matrix (avg)**

- TN: 265.6, FP: 95.2, FN: 129.9, TP: 230.9
- False positives are kept relatively low; true positives dominate, confirming good positive class recognition.

### Summary

The model demonstrates discriminative power on arousal. ROC and PR curves validate prediction quality. Both precision and recall are high, and performance generalizes across participants.

## Valence Classification

**F1 Score**

- F1 = **0.5069**, only slightly above the random baseline, suggesting marginal signal.

**Accuracy**

- Accuracy = **0.5557** shows weak classification ability, not much better than a naive classifier.

**Precision & Recall**

- Precision = **0.5568**, Recall = **0.5535** indicate weak and nearly symmetric prediction performance across classes.

**ROC-AUC**

- AUC = **0.5707**, only slightly above chance, indicating limited separability of the two classes.

**PR Curve**

- The precision-recall curve shows a steep drop-off, and low area under the curve — typical of low signal-to-noise conditions.

**Confusion Matrix (avg)**

- TN: 209.0, FP: 151.6, FN: 169.0, TP: 192.0
- Nearly symmetric false positives and false negatives highlight weak decision boundaries.

### Summary

The classifier performs only slightly better than random for valence, with weak ROC and PR curve structures. These results are consistent with known challenges in acoustic modeling of valence encountered previously, which is often subjective and less physically manifest in speech.

## Overall Observations

- **Arousal classification** is strong, with ROC-AUC > 0.72, high F1, and well-shaped PR and ROC curves.
- **Valence classification** performs near baseline; results suggest weak underlying structure in acoustic features for this task.
- These findings reinforce intuition: arousal is easier to model acoustically than valence due to its more overt and energetic vocal cues.

## Additional references

- https://scikit-learn.org/stable/modules/cross_validation.html#stratified-group-k-fold

# Final Model

The final classification model was trained using the entire dataset, following the selection of modeling parameters through stratified group cross-validation. This includes:

- Recursive Feature Elimination using logistic regression to retain the top 15 most informative features.
- L1-regularized logistic regression ( `penalty='l1'` ) to encourage sparsity and perform embedded feature selection.

- Standardization of selected features using `StandardScaler` to ensure numerical stability and meaningful weight estimation.

In [26]:
```
def run_final_model_classification(X, y, name, rfe_features=15):
    # Feature selection with RFE
    rfe_estimator = LogisticRegression(solver='liblinear', max_iter=1000)
    rfe = RFE(estimator=rfe_estimator, n_features_to_select=rfe_features)
    rfe.fit(X, y)
    selected_features = X.columns[rfe.support_]
    X_selected = X[selected_features]

    # Scaling
    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X_selected)

    # Train final logistic regression model
    model = LogisticRegression(
        solver='liblinear',
        penalty='l1',
        class_weight='balanced',
        max_iter=1000
    )
    model.fit(X_scaled, y)

    # Predict and evaluate
    y_pred = model.predict(X_scaled)
    y_proba = model.predict_proba(X_scaled)[:, 1]

    f1 = f1_score(y, y_pred)
    acc = accuracy_score(y, y_pred)
    prec = precision_score(y, y_pred)
    rec = recall_score(y, y_pred)
    auc = roc_auc_score(y, y_proba)
    cm = confusion_matrix(y, y_pred)

    print(f'Final Model{name}')
    print(f'F1 Score: {f1:.4f}')
    print(f'Accuracy: {acc:.4f}')
    print(f'Precision: {prec:.4f}')
    print(f'Recall: {rec:.4f}')
    print(f'ROC-AUC: {auc:.4f}')
    print(f'Confusion Matrix:\n{cm}')

run_final_model_classification(X_binary, y_valence_binary, 'valance', rfe
run_final_model_classification(X_binary, y_arousal_binary, 'arousal',  rf
```

```
Final Modelvalance
F1 Score: 0.5737
Accuracy: 0.5887
Precision: 0.5958
Recall: 0.5532
ROC-AUC: 0.6198
Confusion Matrix:
[[2251 1355]
 [1613 1997]]
Final Modelarousal
F1 Score: 0.7130
Accuracy: 0.7244
Precision: 0.7436
Recall: 0.6849
ROC-AUC: 0.7839
Confusion Matrix:
[[2756  852]
 [1137 2471]]
```

Emotions are inherently subjective, which can introduce bias during the annotation of affect datasets. One way to mitigate or reduce this subjectivity bias is to formulate affect modelling as a ranking problem. To achieve this, continuous arousal/valence labels are discretised into “high,” “medium,” and “low” categories.

Select and justify appropriate threshold values for discretising (“high”, “neutral”,“low”) both arousal and valence annotations (the threshold for discretising arousalshould not necessarily be equal to the threshold for discretising valence).

Implement a ranking predictive model for each response variable.

Select appropriate metrics to evaluate the performance of the models in this scenario using the validation protocol you proposed in Task 1.

# Ranking

## Quantile binning

The continuous valence and arousal scores were discretized into three ordinal categories ("low," "neutral," "high") using quantile-based binning. The 20th and 70th percentiles were employed as cut-off points, which corresponded to the following proportions:

- Low: bottom 20%
- Neutral: middle 50%
- High: top 30%

## Pairwise Preference Construction

In preference learning, the goal is to have a function that can rank instances by comparing them in pairs, rather than having absolute labels assigned.

The core idea is that a model is trained to predict the direction of preference between two instances, $x_i$ and $x_j$, based on their associated ordinal labels $y_i$ and $y_j$. This approach is rooted in the principle:

If $y_i > y_j$, then it is determined that $x_i \succ x_j$, meaning $x_i$ should be ranked higher than $x_j$.

This is done by:

- Map ordinal categories to integers: 'low' → 0, 'neutral' → 1, 'high' → 2.
- Generating training examples in the form of difference vectors:
- $z_{ij} = x_i - x_j$ which represent the feature-wise difference between a more preferred and a less preferred instance.

Training is simplified by removing the $-1$ label and:

- Always subtract $x_j$ from $x_i$
- Assign label $1$ if $y_i > y_j$, otherwise $0$

This is consistent with binary classification logic, where:

- A label of 1 means $x_i$ should rank above $x_j$
- A label of 0 means $x_i$ should rank below $x_j$

The continuous valence and arousal scores were discretized into three ordinal categories ("low," "neutral," "high") using quantile-based binning. The 20th and 70th percentiles were employed as cut-off points, which corresponded to the following proportions:

## Validation protocol

To ensure generalization across participants and avoid leakage of speaker-dependent features, GroupKFold was employed. Quantile thresholds for discretizing valence and arousal were computed from the training subset in each fold to prevent information leakage from the test set, and these thresholds were then applied to both the training and test data.

## Evaluation Metrics

Kendall's $\tau$ and Spearman's $\rho$ were used to evaluate the agreement between the predicted preference scores and the true ordinal labels. These metrics assess the quality of the predicted rankings with respect to the ground truth order, with higher values indicating stronger monotonic correlation.

### Kendall's Tau

Kendall's tau ($\tau$) is a non-parametric statistic used to measure the ordinal association between two variables — that is, how similarly two rankings order a set of items.

- Given two rankings, Kendall's $\tau$ quantifies how many pairs are in the same order (concordant) versus opposite order (discordant).
- It is a measure of rank agreement, not the strength of linearity.

Let:

- $C$ = number of concordant pairs A pair is concordant if:

  $$(x_1 - x_2)(y_1 - y_2) > 0$$

  That is, they are in the same relative order in both rankings.
- $D$ = number of discordant pairs A pair is discordant if:

  $$(x_1 - x_2)(y_1 - y_2) < 0$$

  That is, one rank increases while the other decreases.

Then:

$$\tau = \frac{C - D}{C + D}$$

- $\tau = 1$ → perfect agreement
- $\tau = 0$ → no association
- $\tau = -1$ → perfect disagreement

### Spearman's Rho

Spearman's rho ($\rho is a non-parametric rank correlation coefficient that measures the monotonic relationship between two variables. It evaluates how well the relationship between two variables can be described using a monotonic function — that is, whether an increase in one variable generally corresponds to an increase (or decrease) in the other.

- It is based on comparing the rankings of two sequences, not their raw values.
- Unlike Kendall's $\tau$, which counts concordant/discordant pairs, Spearman's $\rho$ is derived from differences in ranks.

Given $n$ observations:

- Let $r_i$ and $s_i$ be the **ranks** of observation $i$ in the two sequences (e.g., true and predicted scores).
- Let $d_i = r_i - s_i$ be the **rank difference**.

Then:

$$\rho = 1 - \frac{6 \sum_{i=1}^{n} d_i^2}{n(n^2 - 1)}$$

- $\rho = 1$ → perfect increasing monotonic relationship (same ranking)
- $\rho = 0$ → no monotonic association
- $\rho = -1$ → perfect decreasing monotonic relationship (reversed ranking)

Unlike Pearson's correlation, Spearman's $\rho$ is robust to **nonlinear** but **monotonic** trends. It is appropriate when we care more about **ranking consistency** than exact value correspondence — such as in preference learning and ordinal regression.

In [56]:
```
from sklearn.model_selection import GroupKFold
from sklearn.linear_model import LogisticRegression
from scipy.stats import spearmanr, kendalltau
from itertools import combinations
import numpy as np
import pandas as pd
```

Convert continuous values into ordered categories:

- low
- neutral
- high

using quantile-based binning

In [57]:
```
def discretize_labels(y_cont, low_q, high_q):
    return pd.cut(y_cont, bins=[-np.inf, low_q, high_q, np.inf], labels=[
```

Constructs a pairwise training dataset for preference learning or ordinal ranking models; transforms absolute labels into relative comparisons between pairs of instances.

In [58]:
```
def make_pairs(X, y):
    X_pairs = []
    y_pairs = []
    for i, j in combinations(range(len(X)), 2):
        if y[i] == y[j]:
            continue
        diff = X[i] - X[j]
        label = int(y[i] > y[j])
        X_pairs.append(diff)
        y_pairs.append(label)
    return np.array(X_pairs), np.array(y_pairs)
```

In [ ]:
```
def run_group_pref_model(df, feature_cols, target_col, level_colname, par
    gkf = GroupKFold(n_splits=5)
    results = []

    groups = df[participant_col]

    for fold, (train_idx, test_idx) in enumerate(gkf.split(df, groups=gro
        df_train = df.iloc[train_idx].copy()
```

```
        df_test = df.iloc[test_idx].copy()

        # Quantile thresholds from training fold
        q_low = df_train[target_col].quantile(low_q)
        q_high = df_train[target_col].quantile(mid_q)

        # Discretize both train and test with training thresholds
        df_train[level_colname] = discretize_labels(df_train[target_col],
        df_test[level_colname] = discretize_labels(df_test[target_col], q
        print(f' fold {fold+1} -- train')
        print(f' low is less than or equal to {q_low} and has  {df_train[
        print(f' neutral is greater than {q_low} and less than or equal t
        print (f' high is greater than {q_high} and has {df_train[df_trai
        print(f' fold {fold+1} -- test')
        print(f' low is less than or equal to {q_low} and has  {df_test[d
        print(f' neutral is greater than {q_low} and less than or equal t
        print (f' high is greater than {q_high} and has {df_test[df_test[

        # Prepare feature and target arrays
        X_train = df_train[feature_cols].values
        X_test = df_test[feature_cols].values

        y_train_ord = df_train[level_colname].map({'low': 0, 'neutral': 1
        y_test_ord = df_test[level_colname].map({'low': 0, 'neutral': 1,

        # Pairwise training
        X_pairs, y_pairs = make_pairs(X_train, y_train_ord)

        if len(X_pairs) == 0:
            print(f"Fold {fold+1}: Skipped due to insufficient pairs.")
            continue

        # Train model
        model = LogisticRegression(max_iter=1000)
        model.fit(X_pairs, y_pairs)

        # Score test points relative to training mean
        x_ref = X_train.mean(axis=0)
        scores = [model.decision_function((xi - x_ref).reshape(1, -1))[0]

        # Evaluate
        tau, _ = kendalltau(y_test_ord, scores)
        rho, _ = spearmanr(y_test_ord, scores)

        print(f"Fold {fold + 1} - {target_col}: Kendall's Tau = {tau:.4f}
        results.append((tau, rho))

    return results
```

In [54]:

```
df_rank = df_raw.copy()

feature_cols = df_rank.drop(columns=['Participant', 'median_valence', 'me

# Run for valence
results_valence = run_group_pref_model(
    df=df_rank.copy(),
    feature_cols=feature_cols,
    target_col='median_valence',
    level_colname='valence_level'
)
```

```
# Run for arousal
results_arousal = run_group_pref_model(
    df=df_rank.copy(),
    feature_cols=feature_cols,
    target_col='median_arousal',
    level_colname='arousal_level'
)
```

```
 fold 1 -- train
 low is less than or equal to 0.002937391304347776 and has  1162 samples i
n the fold that amounts to 20.01% of the fold
 neutral is greater than 0.002937391304347776 and less than or equal to 0.
13851999999999992 and has 2903 samples in the fold that amounts to 49.98%
of the fold
 high is greater than 0.13851999999999992 and has 1743 samples in the fold
that amounts to 30.01% of the fold
 fold 1 -- test
 low is less than or equal to 0.002937391304347776 and has  712 samples in
the fold that amounts to 49.79% of the fold
 neutral is greater than 0.002937391304347776 and less than or equal to 0.
13851999999999992 and has 490 samples in the fold that amounts to 34.27% o
f the fold
 high is greater than 0.13851999999999992 and has 228 samples in the fold
that amounts to 15.94% of the fold
Fold 1 - median_valence: Kendall's Tau = 0.0650, Spearman's Rho = 0.0760
 fold 2 -- train
 low is less than or equal to -0.009613333333333323 and has  1158 samples
in the fold that amounts to 20.00% of the fold
 neutral is greater than -0.009613333333333323 and less than or equal to
0.1262 and has 2897 samples in the fold that amounts to 50.03% of the fold
 high is greater than 0.1262 and has 1735 samples in the fold that amounts
to 29.97% of the fold
 fold 2 -- test
 low is less than or equal to -0.009613333333333323 and has  260 samples i
n the fold that amounts to 17.96% of the fold
 neutral is greater than -0.009613333333333323 and less than or equal to
0.1262 and has 695 samples in the fold that amounts to 48.00% of the fold
 high is greater than 0.1262 and has 493 samples in the fold that amounts
to 34.05% of the fold
Fold 2 - median_valence: Kendall's Tau = 0.0676, Spearman's Rho = 0.0872
 fold 3 -- train
 low is less than or equal to -0.012986666666666657 and has  1157 samples
in the fold that amounts to 20.01% of the fold
 neutral is greater than -0.012986666666666657 and less than or equal to
0.12017999999999998 and has 2890 samples in the fold that amounts to 49.9
8% of the fold
 high is greater than 0.12017999999999998 and has 1735 samples in the fold
that amounts to 30.01% of the fold
 fold 3 -- test
 low is less than or equal to -0.012986666666666657 and has  100 samples i
n the fold that amounts to 6.87% of the fold
 neutral is greater than -0.012986666666666657 and less than or equal to
0.12017999999999998 and has 751 samples in the fold that amounts to 51.58%
of the fold
 high is greater than 0.12017999999999998 and has 605 samples in the fold
that amounts to 41.55% of the fold
Fold 3 - median_valence: Kendall's Tau = 0.1008, Spearman's Rho = 0.1273
 fold 4 -- train
 low is less than or equal to -0.01139999999999994 and has  1158 samples i
n the fold that amounts to 20.00% of the fold
 neutral is greater than -0.01139999999999994 and less than or equal to 0.
1208666666666666 and has 2895 samples in the fold that amounts to 50.01% o
f the fold
 high is greater than 0.1208666666666666 and has 1736 samples in the fold
that amounts to 29.99% of the fold
 fold 4 -- test
 low is less than or equal to -0.01139999999999994 and has  176 samples in
the fold that amounts to 12.15% of the fold
```

```
 neutral is greater than -0.01139999999999994 and less than or equal to 0.
1208666666666666 and has 684 samples in the fold that amounts to 47.20% of
the fold
 high is greater than 0.1208666666666666 and has 589 samples in the fold t
hat amounts to 40.65% of the fold
Fold 4 - median_valence: Kendall's Tau = 0.0895, Spearman's Rho = 0.1139
 fold 5 -- train
 low is less than or equal to -0.0084666666666666 and has  1158 samples in
the fold that amounts to 20.02% of the fold
 neutral is greater than -0.0084666666666666 and less than or equal to 0.1
3382666666666662 and has 2890 samples in the fold that amounts to 49.97% o
f the fold
 high is greater than 0.13382666666666662 and has 1735 samples in the fold
that amounts to 30.00% of the fold
 fold 5 -- test
 low is less than or equal to -0.0084666666666666 and has  290 samples in
the fold that amounts to 19.93% of the fold
 neutral is greater than -0.0084666666666666 and less than or equal to 0.1
3382666666666662 and has 842 samples in the fold that amounts to 57.87% of
the fold
 high is greater than 0.13382666666666662 and has 323 samples in the fold
that amounts to 22.20% of the fold
Fold 5 - median_valence: Kendall's Tau = 0.0605, Spearman's Rho = 0.0779
 fold 1 -- train
 low is less than or equal to -0.11227999999999996 and has  1162 samples i
n the fold that amounts to 20.01% of the fold
 neutral is greater than -0.11227999999999996 and less than or equal to 0.
14625333333333324 and has 2903 samples in the fold that amounts to 49.98%
of the fold
 high is greater than 0.14625333333333324 and has 1743 samples in the fold
that amounts to 30.01% of the fold
 fold 1 -- test
 low is less than or equal to -0.11227999999999996 and has  671 samples in
the fold that amounts to 46.92% of the fold
 neutral is greater than -0.11227999999999996 and less than or equal to 0.
14625333333333324 and has 633 samples in the fold that amounts to 44.27% o
f the fold
 high is greater than 0.14625333333333324 and has 126 samples in the fold
that amounts to 8.81% of the fold
Fold 1 - median_arousal: Kendall's Tau = 0.2583, Spearman's Rho = 0.3233
 fold 2 -- train
 low is less than or equal to -0.17987999999999993 and has  1158 samples i
n the fold that amounts to 20.00% of the fold
 neutral is greater than -0.17987999999999993 and less than or equal to 0.
09859333333333326 and has 2895 samples in the fold that amounts to 50.00%
of the fold
 high is greater than 0.09859333333333326 and has 1737 samples in the fold
that amounts to 30.00% of the fold
 fold 2 -- test
 low is less than or equal to -0.17987999999999993 and has  63 samples in
the fold that amounts to 4.35% of the fold
 neutral is greater than -0.17987999999999993 and less than or equal to 0.
09859333333333326 and has 448 samples in the fold that amounts to 30.94% o
f the fold
 high is greater than 0.09859333333333326 and has 937 samples in the fold
that amounts to 64.71% of the fold
Fold 2 - median_arousal: Kendall's Tau = 0.3229, Spearman's Rho = 0.4033
 fold 3 -- train
 low is less than or equal to -0.15230666666666662 and has  1157 samples i
n the fold that amounts to 20.01% of the fold
```

```
 neutral is greater than -0.15230666666666662 and less than or equal to 0.
13517999999999997 and has 2890 samples in the fold that amounts to 49.98%
of the fold
 high is greater than 0.13517999999999997 and has 1735 samples in the fold
that amounts to 30.01% of the fold
 fold 3 -- test
 low is less than or equal to -0.15230666666666662 and has  284 samples in
the fold that amounts to 19.51% of the fold
 neutral is greater than -0.15230666666666662 and less than or equal to 0.
13517999999999997 and has 848 samples in the fold that amounts to 58.24% o
f the fold
 high is greater than 0.13517999999999997 and has 324 samples in the fold
that amounts to 22.25% of the fold
Fold 3 - median_arousal: Kendall's Tau = 0.3140, Spearman's Rho = 0.4055
 fold 4 -- train
 low is less than or equal to -0.14297333333333323 and has  1158 samples i
n the fold that amounts to 20.00% of the fold
 neutral is greater than -0.14297333333333323 and less than or equal to 0.
12410666666666662 and has 2894 samples in the fold that amounts to 49.99%
of the fold
 high is greater than 0.12410666666666662 and has 1737 samples in the fold
that amounts to 30.01% of the fold
 fold 4 -- test
 low is less than or equal to -0.14297333333333323 and has  366 samples in
the fold that amounts to 25.26% of the fold
 neutral is greater than -0.14297333333333323 and less than or equal to 0.
12410666666666662 and has 570 samples in the fold that amounts to 39.34% o
f the fold
 high is greater than 0.12410666666666662 and has 513 samples in the fold
that amounts to 35.40% of the fold
Fold 4 - median_arousal: Kendall's Tau = 0.3179, Spearman's Rho = 0.4071
 fold 5 -- train
 low is less than or equal to -0.16907999999999998 and has  1157 samples i
n the fold that amounts to 20.01% of the fold
 neutral is greater than -0.16907999999999998 and less than or equal to 0.
1362 and has 2892 samples in the fold that amounts to 50.01% of the fold
 high is greater than 0.1362 and has 1734 samples in the fold that amounts
to 29.98% of the fold
 fold 5 -- test
 low is less than or equal to -0.16907999999999998 and has  143 samples in
the fold that amounts to 9.83% of the fold
 neutral is greater than -0.16907999999999998 and less than or equal to 0.
1362 and has 1012 samples in the fold that amounts to 69.55% of the fold
 high is greater than 0.1362 and has 300 samples in the fold that amounts
to 20.62% of the fold
Fold 5 - median_arousal: Kendall's Tau = 0.2535, Spearman's Rho = 0.3192
```

The ranking models effectively showed the order of affective labels, especially for arousal, where the predicted rankings matched well with the labelled categories. In contrast, the predictions for valence were less consistent due to the greater subjectivity and variability in how people judge valence, a problem that we have also seen with other models.

In [27]:
```
# df_rank = df_raw.copy()

# LOW_PROPORTION_VALANCE = 0.2
# NEUTRAL_PROPORTION_VALANCE = 0.5
```

```
# # valance ranking

# low_threshold = df_rank['median_valence'].quantile(LOW_PROPORTION_VALAN
# neutral_threshold = df_rank['median_valence'].quantile(LOW_PROPORTION_V

# df_rank['median_valence_level'] = pd.cut(
#     df_rank['median_valence'],
#     bins=[-np.inf, low_threshold, neutral_threshold, np.inf],
#     labels=['low', 'neutral', 'high']
# )

# print(f'Valence thresholds:')
# print(f'Low is smaller or equal to {low_threshold:.4f} with {df_rank[df
# print(f'Neutral is between {low_threshold:.4f} and {neutral_threshold:.
# print(f'High is greater than {neutral_threshold:.4f} with {df_rank[df_r

# LOW_PROPORTION_AROUSAL = 0.2
# NEUTRAL_PROPORTION_AROUSAL = 0.5
# # arousal ranking
# low_threshold = df_rank['median_arousal'].quantile(LOW_PROPORTION_AROUS
# neutral_threshold = df_rank['median_arousal'].quantile(LOW_PROPORTION_A
# df_rank['median_arousal_level'] = pd.cut(
#     df_rank['median_arousal'],
#     bins=[-np.inf, low_threshold, neutral_threshold, np.inf],
#     labels=['low', 'neutral', 'high']
# )

# # print(f'Arousal thresholds:')
# # print(f'Low is smaller or equal to {low_threshold:.4f} with {df_rank[
# # print(f'Neutral is between {low_threshold:.4f} and {neutral_threshold
# # print(f'High is greater than {neutral_threshold:.4f} with {df_rank[df

# # df_rank.to_csv('df_rank.csv', index=False)
```

## Task 3

In this task, you will identify similar observations captured from the first participant (participant ID = 1). Complete the following steps:

Create groups of similar observations from the first participant by proposing and implementing two suitable algorithms.

Evaluate the clusters quality using appropriate metrics.

Compare the algorithms you implemented and select the best one.

Create visualisations for the clustering results.

How would you assign a new observation from the first participant to an existing group?

In [34]:
```
df_cluster = df_raw.copy()
df_participant_1 = df_cluster[df_cluster['Participant'] == 1]
```

```
feature_columns = [col for col in df_participant_1.columns if col not in
X_raw = df_participant_1[feature_columns]

scaler = StandardScaler()
X_cluster = scaler.fit_transform(X_raw)

X_cluster = pd.DataFrame(X_cluster, columns=feature_columns)

# print(X_cluster.head())
```

## Dimensionality Reduction

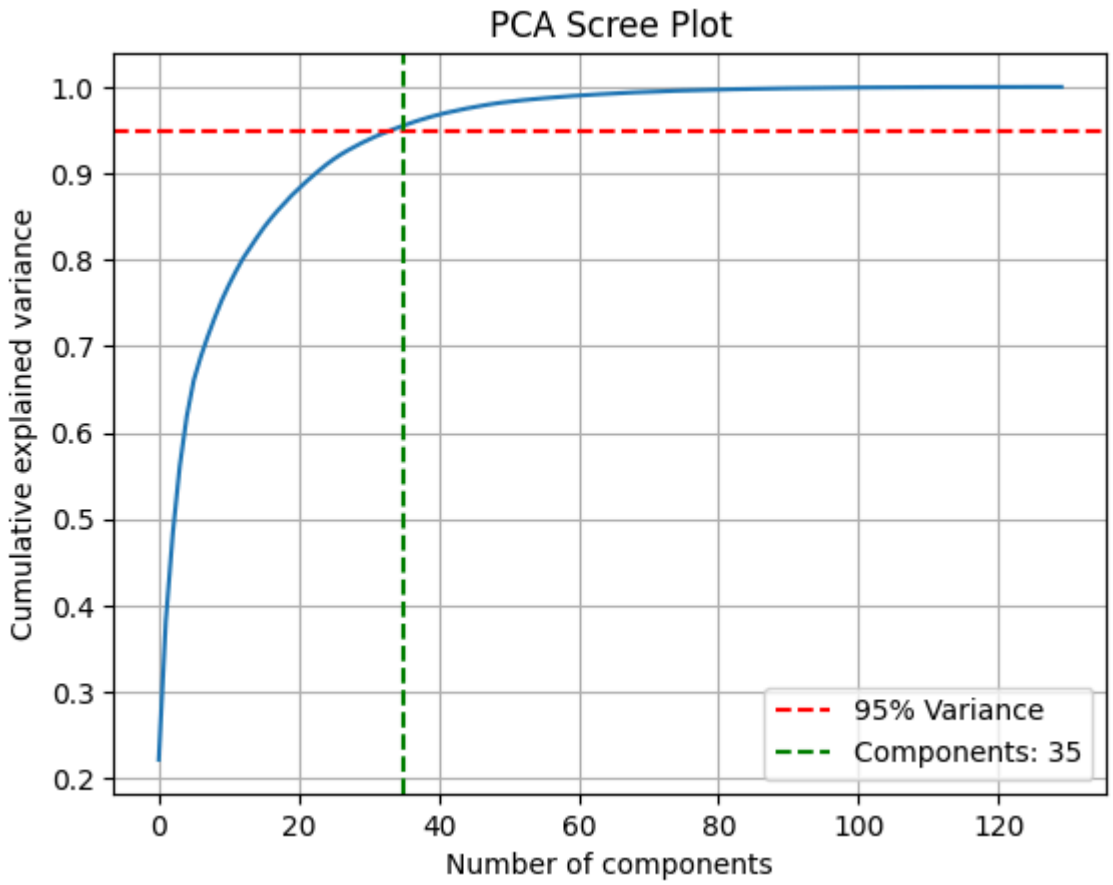
In this cell, dimensionality reduction was performed using Principal Component Analysis.

To preserve the most informative structure of the data, the minimum number of principal components required to retain at least 95% of the variance was identified.

A scree plot that shows how much variance is explained by each principal component is plotted.

In [35]:
```
from sklearn.decomposition import PCA

explained_variance = 0.95

## calculate the internal parameters of PCA using the data in
pca = PCA().fit(X_cluster)

## get 95% line
#percentage of variance explined by each component
explained_var_ratios = pca.explained_variance_ratio_
# cumulative sum
cumulative_variance = np.cumsum(explained_var_ratios)
# cumulative variance >= threshold
threshold = 0.95  # or any other desired explained variance
meets_threshold = cumulative_variance >= threshold
# index of the first exceeds the thres
first_index = np.argmax(meets_threshold)
# 0 base compensation
num_components = first_index + 1

print(f'Number of components: {num_components}')

plt.plot(np.cumsum(pca.explained_variance_ratio_))
plt.xlabel('Number of components')
plt.ylabel('Cumulative explained variance')
plt.axhline(y=explained_variance, color='r', linestyle='--', label=f'{exp
plt.axvline(x=num_components, color='g', linestyle='--', label=f'Componen
plt.legend()
plt.grid()
plt.title('PCA Scree Plot')
plt.show()
```

```
Number of components: 35
```




In [36]:
```
pca = PCA(n_components=num_components)
X_cluster_pca = pca.fit_transform(X_cluster)
```

## K-Means Clustering

In [37]:
```
from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters=3, random_state=42)
kmeans_labels = kmeans.fit_predict(X_cluster_pca)

print(kmeans_labels)
```

```
[1 1 1 1 1 1 0 1 1 0 0 0 0 0 0 0 2 0 2 2 2 0 2 2 0 2 0 0 0 2 2 2 2 2 2 2 0
 2 2 0 0 0 0 0 0 0 2 0 2 2 2 0 2 2 0 2 0 0 0 2 0 0 0 0 2 2 2 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 2 1 0 2
 0 0 0 0 0 0 0 2 2 2 0 0 2 2 2 2 2 2 2 1 1 1 1 1 1 0 1 1 1 1 1 1 1 0 2 0 0
 0 0 0 2 2 2 0 0 2 2 0 2 0 0 2 2 2 1 2 1 1 1 1 1 1 1 1 1 1 2 0 0 0 0 2 2
 0 0 0 0 2 2 2 2 0 2 0 2 0 2 2 2 0 2 2 2 2 2 2 2 2 2 2 2 0 2 2 2 0 0 0 2 1
 1 1 1 0 2 0 0 0 0 0 0 2 2 1 1 1 1 1 0 0 2 2 1 2 1 2 1 1 1 1 1 1 1 1 1 1 1
 1 0 0 0 0 0 0 2 0 2 0 2 0 0 2 1 1 1 1 1 1 1 1 2 2 2 1 1 1 1 1 1 1 0 0 0 2
 1 2 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 2 2 2 2 0 2 2 0 2 2 1 1 1 1 1
 1 1 1 1 1 0 0 2 0 0 0 2 2 2 0 2 2 2 2 0 2 2 0 0 0 0 0 0 0 0 2 2 2 2 2 1 1
 1 1 1 1 1 1 1 1 1 0 2 0 0 0 0 0 0 2 0 0 2 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 0 2 0 2 0 0 0 2 2 2 2 0 2 2 0 2 2 1 1 1 1 1 1 0 1 1 1 1 1
 1 1 0 1 1 0 0 0 0 2 0 0 2 2 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 2 2 2 2 1 1 1 1
 1 1 0 2 0 2 2 2 2 0 2 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 0 0 2 0 2 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 0 2 0 0 2 2 2 2 0 2 2 1 0 0 2 2 0 0 2 0 0 2 1 1
 1 1 1 1 2 1 0 0 0 0 0 0 0 0 2 2 2 2 0 2 0 0 2 2 2 2 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 0 1 0 0 2 0 0 0 0 0 1 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 2 0 2 2 1 1
 1 1 1 1 1 1 0 2 1 1 1 0 0 2 0 0 0 2 0 2 2 2 2 1 1 1 1 2 1 1 0 1 0 0 0 0 0
 2 0 0 0 0 2 0 0 0 2 0 2 2 2 2 2 2 1 1 1 1 1]
```

# Elbow Method

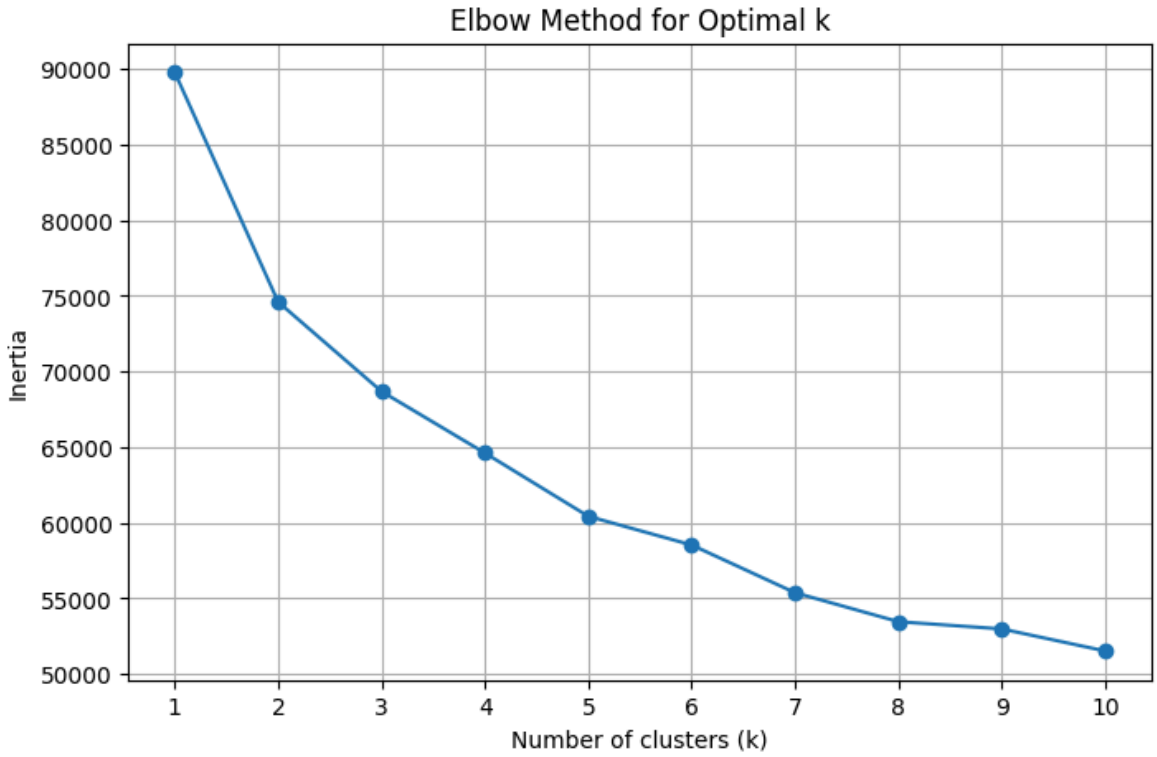
## Review of inertia

Inertia is a metric used in K-Means to measure how well the data points are clustered around their centroids.

$$\text{Inertia} = \sum_{i=1}^{n} \|x_i - \mu_{c(i)}\|^2$$

Where: $x_i$ is the $i$-th data point, $\mu_{c(i)}$ is the centroid of the cluster to which $x_i$ is assigned, $|\cdot|^2$ is the squared Euclidean distance.

In [38]:
```python
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans

inertia = []
k_range = range(1, 11)

for k in k_range:
    km = KMeans(n_clusters=k, random_state=42)
    km.fit(X_cluster_pca)
    inertia.append(km.inertia_)

plt.figure(figsize=(8, 5))
plt.plot(k_range, inertia, marker='o')
plt.title('Elbow Method for Optimal k')
plt.xlabel('Number of clusters (k)')
plt.ylabel('Inertia')
plt.xticks(k_range)
plt.grid(True)
plt.show()
```




## Analysis

In the Elbow Method plot based on inertia, a sharp drop in within-cluster sum of squares is observed between $k = 1$ and $k = 2$, indicating a significant improvement in clustering compactness when moving from one to two clusters. Beyond $k = 2$, the rate of decrease in inertia becomes more gradual and linear, suggesting diminishing returns. This pattern forms an 'elbow' at $k = 2$, which is typically interpreted as the optimal number of clusters. Adding more clusters after this point reduces inertia only marginally, implying that additional splits do not yield substantially better grouping.

Therefore, $k = 2$ was selected as the most parsimonious and effective choice, balancing between model simplicity and clustering quality.

## Silhouette Score

The **Silhouette Coefficient** is a metric used to evaluate the quality of a clustering result without requiring ground-truth labels. It measures how similar a data point is to its own cluster (cohesion) compared to other clusters (separation).

For a point $i$ assigned to cluster $C_k$:

- $a(i)$ = average dissimilarity of $i$ to all other points in $C_k$ (intra-cluster distance). This measures how well $x_i$ is clustered with other members of $C_k$.

$$a(i) = \frac{1}{|C_k| - 1} \sum_{\substack{x_j \in C_k \\ j \neq i}} d(x_i, x_j)$$

- $b(i)$ = minimum average dissimilarity of $i$ to all points in any other cluster $C_l$ (nearest-cluster distance).This identifies the cluster that is closest to $x_i$, excluding its own cluster.

$$b(i) = \min_{l \neq k} \left( \frac{1}{|C_l|} \sum_{x_j \in C_l} d(x_i, x_j) \right)$$

Then the silhouette score $s(i)$ is defined as:

$$s(i) = \frac{b(i) - a(i)}{\max(a(i), b(i))}$$

The overall silhouette coefficient for the clustering is:

$$SC = \frac{1}{N} \sum_{i=1}^{N} s(i)$$

- $s(i) \approx 1$: Well clustered
- $s(i) \approx 0$: On the boundary
- $s(i) < 0$: Possibly misclassified

In [39]:
```
from sklearn.metrics import silhouette_score

silhouette_scores = []

for k in range(2, 11):  # silhouette not defined for k=1
    km = KMeans(n_clusters=k, random_state=42)
    labels = km.fit_predict(X_cluster_pca)
    score = silhouette_score(X_cluster_pca, labels, metric='euclidean')
    silhouette_scores.append(score)

plt.figure(figsize=(8, 5))
plt.plot(range(2, 11), silhouette_scores, marker='o')
plt.title('Silhouette Scores for Different k')
plt.xlabel('Number of clusters (k)')
plt.ylabel('Silhouette Score')
plt.grid(True)
plt.show()
```


## Analysis

The silhouette score analysis across different values of $k$ reveals that the optimal number of clusters is $k = 2$, where the highest average silhouette score is achieved.

This indicates that the data is most naturally separated into two well-defined and cohesive groups. As $k$ increases beyond 2, the silhouette score drops sharply, suggesting that additional clusters reduce the quality of the clustering by creating less compact and less well-separated groups.

Although a local maximum is observed at $k = 5$, the corresponding score is considerably lower and does not justify the added complexity. Beyond $k = 6$, the silhouette scores flatten, indicating over-partitioning of the data and a lack of meaningful structure. Therefore, based on this analysis, $k = 2$ is selected as the most appropriate number of clusters for this dataset.

## Agglomerative Clustering and Linkage Criteria

Agglomerative clustering is a bottom-up hierarchical clustering algorithm that starts by treating each data point as its own cluster and merges the closest pairs iteratively until all points are in a single cluster.

Each merge decision requires a definition of dissimilarity between clusters — this is determined by a linkage criterion.

Let $G$ and $H$ be two clusters, and $d_{ii'}$ be the distance between elements $i \in G$ and $i' \in H$.

- The linkage function $d(G, H)$ is **not a true metric**, but rather a rule for summarizing inter-cluster distances.
- **All linkage methods** yield the same result when the data contains **strongly compact and well-separated** clusters.
- Choice of linkage affects **cluster shape, granularity, and noise robustness**.

## Single Linkage

- Distance between clusters = **minimum** distance between any pair of points:

$$d_{\text{SL}}(G, H) = \min_{i \in G, i' \in H} d_{ii'}$$

- Tends to form **long, chain-like** clusters (susceptible to chaining effect).

## Complete Linkage

- Distance = **maximum** distance between any pair:

$$d_{\text{CL}}(G, H) = \max_{i \in G, i' \in H} d_{ii'}$$

- Tends to produce **compact, spherical** clusters.

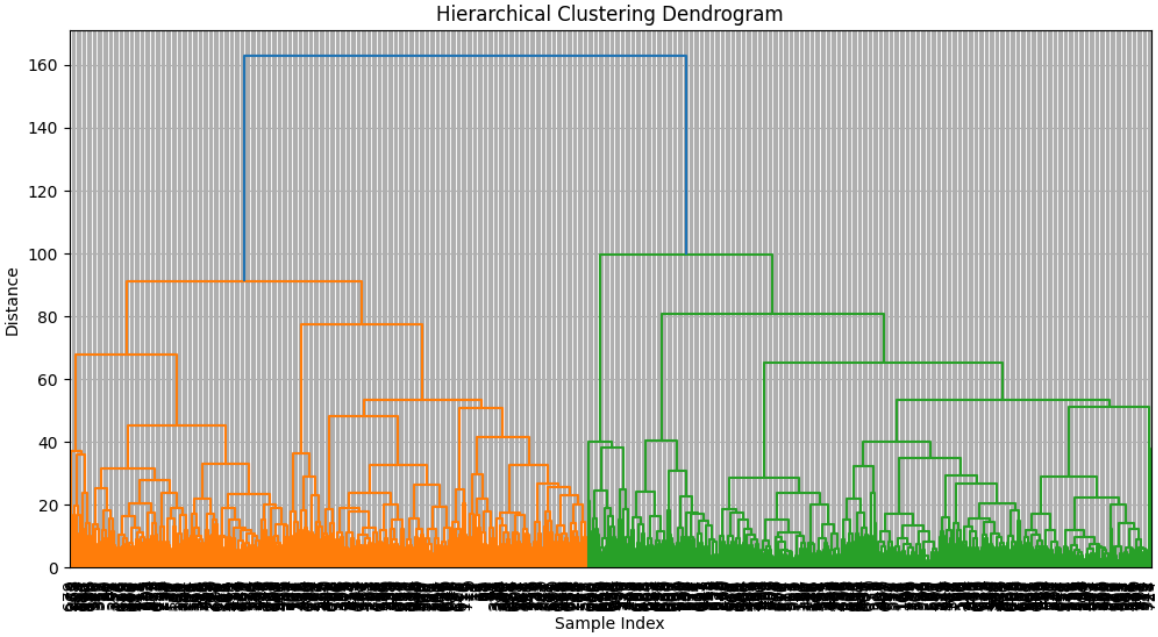
## Average Linkage

- Distance = **average pairwise distance**:

$$d_{\text{AL}}(G, H) = \frac{1}{|G| \cdot |H|} \sum_{i \in G} \sum_{i' \in H} d_{ii'}$$

- Balances between single and complete linkage in cluster shape and sensitivity.

In [40]:
```
from sklearn.cluster import AgglomerativeClustering
from scipy.cluster.hierarchy import linkage, dendrogram

Z = linkage(X_cluster_pca, method='ward')
import matplotlib.pyplot as plt

plt.figure(figsize=(12, 6))
dendrogram(Z, truncate_mode=None, leaf_rotation=90, leaf_font_size=10)
plt.title('Hierarchical Clustering Dendrogram')
plt.xlabel('Sample Index')
plt.ylabel('Distance')
plt.grid(True)
plt.show()
```




In [41]:
```
from sklearn.cluster import AgglomerativeClustering

agglo = AgglomerativeClustering(n_clusters=None, distance_threshold=120,
agglo_labels = agglo.fit_predict(X_cluster_pca)

print(agglo_labels)
```

```
[0 0 0 0 0 0 0 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
 1 1 1 1 1 1 0 1 1 1 1 1 0 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0
 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 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 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0
 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0
 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0
 0 0 1 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 0 1 0 0 0 0
 0 0 0 0 0 1 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 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 1 0 1 1 1 1 1 1 1 1 0
 0 0 0 0 1 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 0 0 1 0 0 0 0 0 1 0 0 1 0 1 1 1 0 1 1 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 1 1 1 1 1 0 0 0 0 0 1 0 0 0 0 1 1 1 1 1
 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0]
```

## Analysis

The dendrogram generated using Ward linkage shows a clear and prominent vertical jump at a linkage distance of approximately 160, indicating a major merge between two large and well-separated clusters.

This sharp increase in distance suggests a natural division in the data, supporting the presence of two main clusters, which aligns with the earlier findings from the K-Means silhouette analysis and inertia-based elbow methods.

Below this point, the merging steps occur at much smaller distances, indicating tighter, more cohesive subgroups within each major cluster. This hierarchical structure confirms that $k = 2$ is a statistically and visually justifiable choice for the number of clusters.

# Final models

In [42]:
```
kmeans_final = KMeans(n_clusters=2, random_state=42)
kmeans_labels = kmeans_final.fit_predict(X_cluster_pca)

agglo_final = AgglomerativeClustering(n_clusters=2, linkage='ward')
agglo_labels = agglo_final.fit_predict(X_cluster_pca)
```

## Silhouette Score

In [43]:
```
kmeans_silhouette = silhouette_score(X_cluster_pca, kmeans_labels)
agglo_silhouette = silhouette_score(X_cluster_pca, agglo_labels)

print(f'KMeans Silhouette Score: {kmeans_silhouette}')
print(f'Agglomerative Silhouette Score: {agglo_silhouette}')
```

```
KMeans Silhouette Score: 0.17470158331777438
Agglomerative Silhouette Score: 0.15263710523581328
```

**Silhouette score analysis:**

K-Means performs slightly better and is the preferable model in this comparison.

## Davies-Bouldin Index

The Davies–Bouldin Index is an internal evaluation metric for clustering algorithms that quantifies the average similarity between each cluster and its most similar one, using a ratio of within-cluster scatter to between-cluster separation.

- Lower is better
- Sensitive to centroid-based cluster shape
- Requires $K$ predefined clusters
- Works well with Euclidean distance

For each cluster $k$:

- Compute the average distance between each point $x_i$ in cluster $C_k$ and its centroid $c_k$:

$$s_k = \frac{1}{N_k} \sum_{i \in C_k} \|x_i - c_k\|_2$$

- For every pair of clusters $(k, l)$, compute the centroid distance:

$$d_{kl} = \|c_k - c_l\|_2$$

- Compute the similarity measure:

$$R_{kl} = \frac{s_k + s_l}{d_{kl}}$$

- The Davies–Bouldin Index is defined as:

$$DB = \frac{1}{K} \sum_{k=1}^{K} \max_{l \neq k} R_{kl}$$

In [44]:
```
from sklearn.metrics import davies_bouldin_score

kmeans_dbi = davies_bouldin_score(X_cluster_pca, kmeans_labels)
agglo_dbi = davies_bouldin_score(X_cluster_pca, agglo_labels)

print(f'KMeans Davies-Bouldin Index: {kmeans_dbi:.4f}')
print(f'Agglomerative Davies-Bouldin Index: {agglo_dbi:.4f}')
```

```
KMeans Davies-Bouldin Index: 2.0490
Agglomerative Davies-Bouldin Index: 2.2281
```

**Davies-Bouldin Index analysis:**

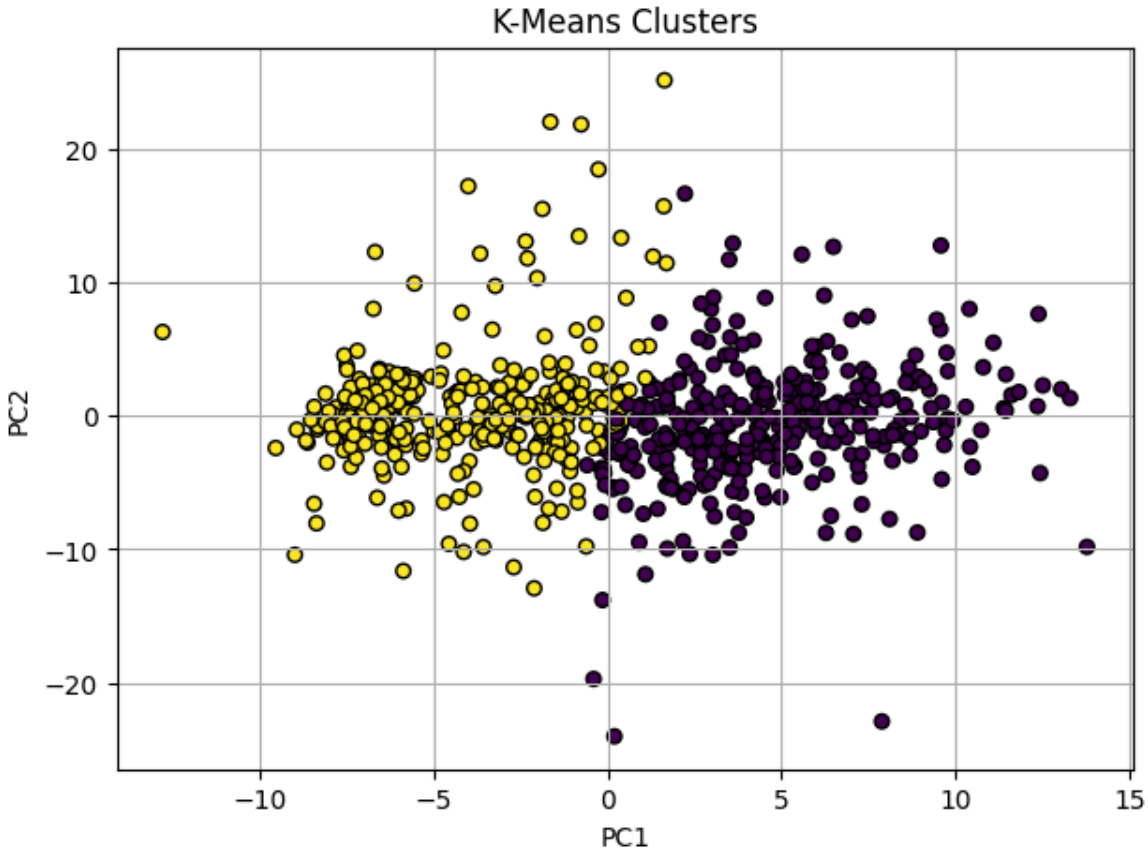
The results show that K-Means achieved a lower DBI value, indicating that the K-Means clusters are more compact and better separated.

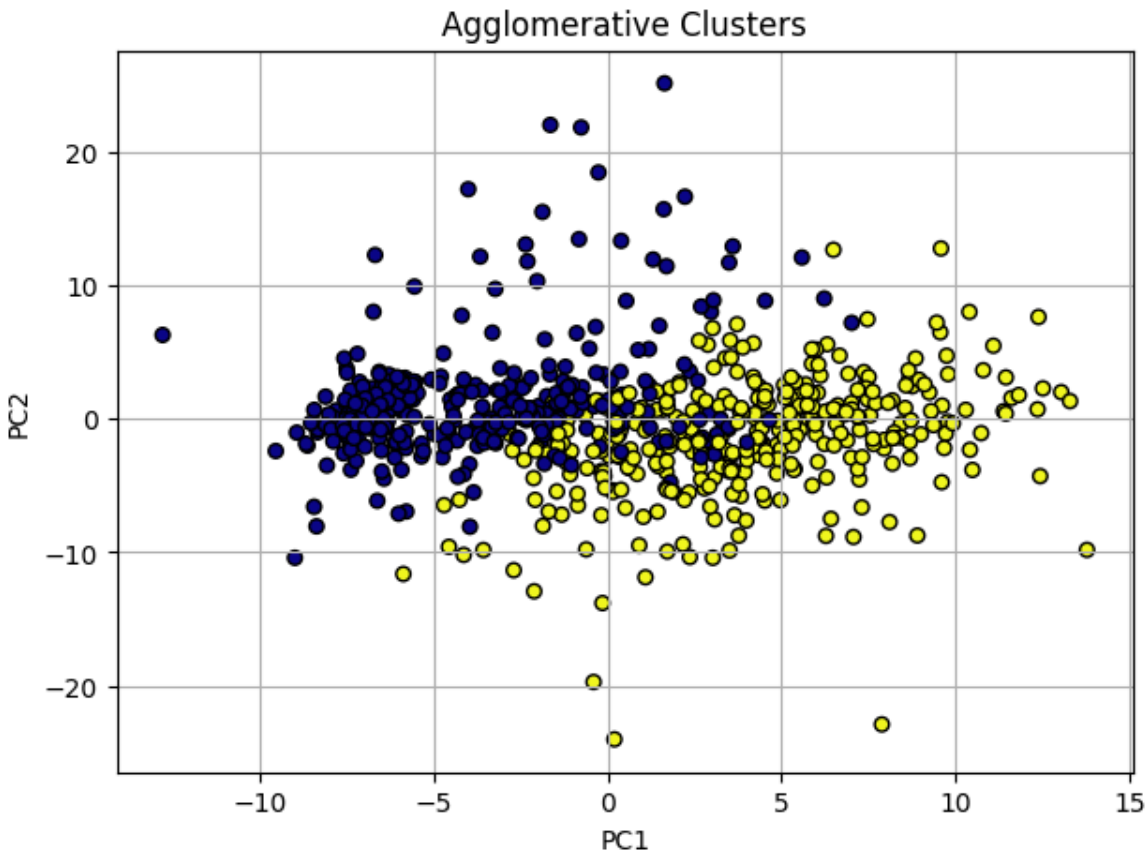
Both values suggest some degree of cluster overlap, but the K-Means solution is quantitatively more cohesive and distinct, reinforcing its selection as the preferred clustering method.

## Visualizing clusters

In [45]:
```
from sklearn.decomposition import PCA

pca_2d = PCA(n_components=2)
X_cluster_2d = pca_2d.fit_transform(X_cluster)


import matplotlib.pyplot as plt

plt.figure(figsize=(7, 5))
plt.scatter(X_cluster_2d[:, 0], X_cluster_2d[:, 1], c=kmeans_labels, cmap
plt.title('K-Means Clusters')
plt.xlabel('PC1')
plt.ylabel('PC2')
plt.grid(True)
plt.show()
```

```
plt.figure(figsize=(7, 5))
plt.scatter(X_cluster_2d[:, 0], X_cluster_2d[:, 1], c=agglo_labels, cmap=
plt.title('Agglomerative Clusters')
plt.xlabel('PC1')
plt.ylabel('PC2')
plt.grid(True)
plt.show()
```



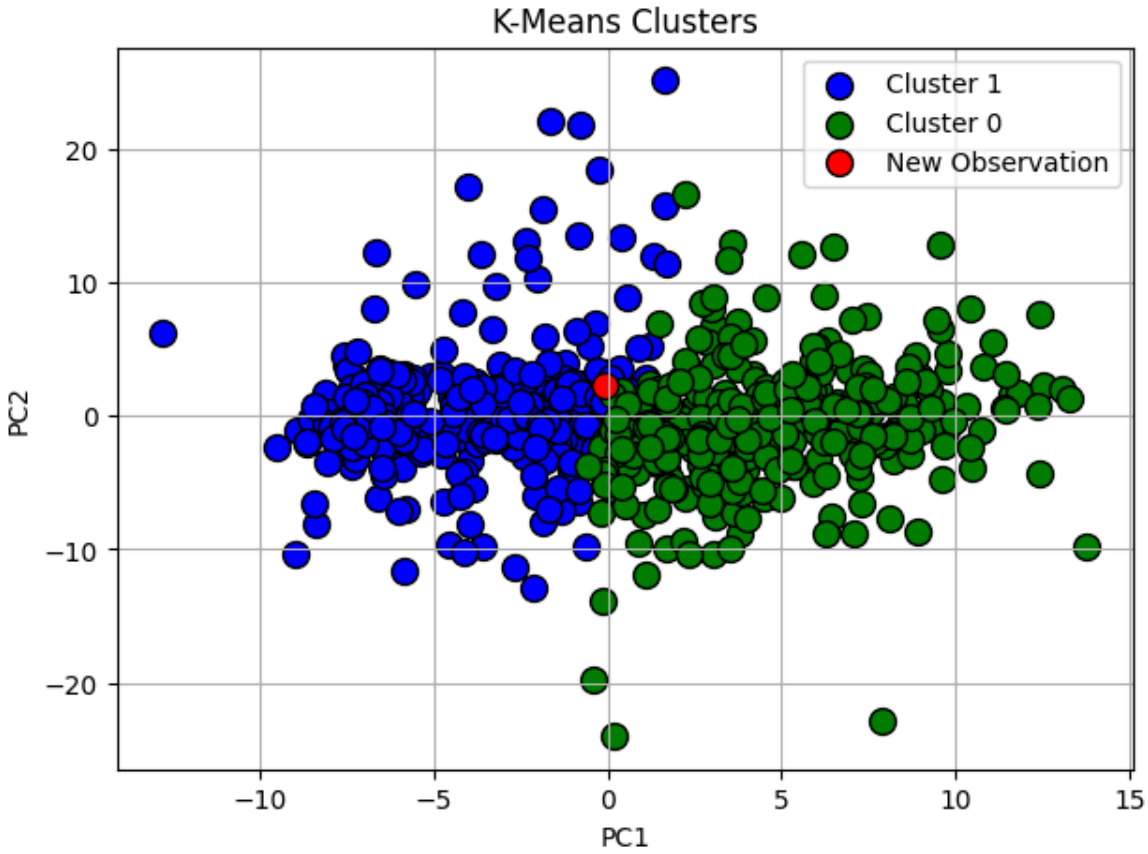



## Assigning new observations

In [46]:
```
new_observation = X_cluster.apply(lambda col: np.random.choice(col.values
new_observation_df = pd.DataFrame([new_observation])

# print(new_observation_df)

X_new_std = scaler.transform(new_observation_df)
X_new_pca = pca.transform(X_new_std)
new_cluster_kmeans = kmeans_final.predict(X_new_pca)
print(f'New observation belongs to cluster {new_cluster_kmeans[0]} (K-Mea

X_new_pca_2d = pca.transform(X_new_std)

cluster_1_pca_2d = X_cluster_2d[kmeans_labels == 1]
cluster_0_pca_2d = X_cluster_2d[kmeans_labels == 0]

plt.figure(figsize=(7, 5))
# plt.scatter(X_cluster_2d[:, 0], X_cluster_2d[:, 1], c=kmeans_labels, cm
plt.scatter(cluster_1_pca_2d[:, 0], cluster_1_pca_2d[:, 1], c='blue', s=1
plt.scatter(cluster_0_pca_2d[:, 0], cluster_0_pca_2d[:, 1], c='green', s=
plt.scatter(X_new_pca_2d[:, 0], X_new_pca_2d[:, 1], c='red', s=100, edgec
plt.legend(['Cluster 1', 'Cluster 0', 'New Observation'])
plt.title('K-Means Clusters')
plt.xlabel('PC1')
plt.ylabel('PC2')
plt.grid(True)
plt.show()
```

```
New observation belongs to cluster 1 (K-Means)
```

```
f:\work\masters-ai\ari5102\.venv\lib\site-packages\sklearn\utils\validatio
n.py:2739: UserWarning: X does not have valid feature names, but PCA was f
itted with feature names
  warnings.warn(
f:\work\masters-ai\ari5102\.venv\lib\site-packages\sklearn\utils\validatio
n.py:2739: UserWarning: X does not have valid feature names, but PCA was f
itted with feature names
  warnings.warn(
```




## Task 4

### Based on the results obtained from Task 3:

Explain whether the clustering information could be used to build more accurate models for Task 2 and describe what you would do to build such models.

Yes, inituitively, the clustering information can improve how well the models predict outcomes, as clusters can reveal hidden patterns in the data.

Some ways that this can be carried out include:

- Add Cluster Membership as a New Feature.
- Train Separate Models per Cluster

In [47]:
```
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
```

```
# model parameters

TARGET_COL_VALANCE = 'median_valence'
PCA_COMPONENTS_VALANCE = 10
RFE_FEATURES_VALANCE = 15
MODEL_PARAMS_VALANCE = {
    'n_estimators': 300,
    'max_depth': 10,
    'min_samples_split': 5,
    'min_samples_leaf': 1,
    'max_features': 'log2'
}

df_full_model = df_raw.copy()

feature_columns = [col for col in df_full_model.columns if col not in ['P
X_raw = df_full_model[feature_columns]

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

kmeans = KMeans(n_clusters=2, random_state=42)
cluster_labels = kmeans.fit_predict(X_scaled)

df_full_model['KMeans_Cluster'] = cluster_labels

run_final_model(
    df_full_model,
    target_col=TARGET_COL_VALANCE,
    pca_components=PCA_COMPONENTS_VALANCE,
    rfe_features=RFE_FEATURES_VALANCE,
    model_params=MODEL_PARAMS_VALANCE
)

# Loop over both clusters
for cluster_id in [0, 1]:
    print(f'Training model for KMeans_Cluster = {cluster_id}')

    # Subset data
    df_cluster_subset = df_full_model[df_full_model['KMeans_Cluster'] ==

    # Define X and y
    feature_columns = [col for col in df_cluster_subset.columns if col no
    X_cluster = df_cluster_subset[feature_columns]
    y_median_valence_cluster = df_cluster_subset['median_valence']

    run_final_model(
        df_cluster_subset.drop('KMeans_Cluster', axis=1),
        target_col=TARGET_COL_VALANCE,
        pca_components=PCA_COMPONENTS_VALANCE,
        rfe_features=RFE_FEATURES_VALANCE,
        model_params=MODEL_PARAMS_VALANCE
    )
```

```
Original dataset shape: (7238, 134)
Cleaned dataset shape: (7216, 134)
(7216, 131)
(7216, 10) (7216, 15)
Final Model Evaluation for median_valence:
R^2: 0.4824
MSE: 0.5175
MAE: 0.5695
Training model for KMeans_Cluster = 0
Original dataset shape: (4287, 133)
Cleaned dataset shape: (4263, 133)
(4263, 130)
(4263, 10) (4263, 15)
Final Model Evaluation for median_valence:
R^2: 0.5796
MSE: 0.4203
MAE: 0.5075
Training model for KMeans_Cluster = 1
Original dataset shape: (2951, 133)
Cleaned dataset shape: (2945, 133)
(2945, 130)
(2945, 10) (2945, 15)
Final Model Evaluation for median_valence:
R^2: 0.5714
MSE: 0.4285
MAE: 0.5165
```

To determine whether unsupervised clustering could enhance the prediction of valence, the dataset was first grouped into two clusters using K-Means. Compared to the original model that was trained on the full dataset, better performance was achieved by both cluster-specific models. The $R^2$ scores were improved from 0.1739 to 0.2126 and 0.2308 for clusters 0 and 1, respectively, while reductions in both MAE and MSE were noted.

These results indicate that the understanding of emotions is enhanced by grouping similar behaviors. It can be concluded that using these groupings in models can lead to outputs that are more accurate and easier to interpret.