Library Imports & Data Loading and Preprocessing

In [1]:
```
# Import necessary libraries
import sqlite3
import pandas as pd
import numpy as np
import statsmodels.api as sm
import warnings
import matplotlib.pyplot as plt
import seaborn as sns
from statsmodels.stats.stattools import durbin_watson

# Ignore specific warnings
warnings.filterwarnings("ignore", category=DeprecationWarning)

# Set Pandas options to avoid wrapping of DataFrame
pd.set_option('display.expand_frame_repr', False)
```

Data Loading and Preprocessing

First, we'll connect to the SQLite database and load the data into a pandas DataFrame.

In [2]:
```
# Connect to the SQLite database
conn = sqlite3.connect('../meeting_cost_calculator.db')

# Write the SQL query
query = """
SELECT
    meeting_id,
    duration,
    attendees,
    salary
FROM
    meeting_submission
"""

# Read the query into a DataFrame
df = pd.read_sql_query(query, conn)

# Close the database connection
conn.close()

# Display the first few rows of the DataFrame
print(df.head())
```

```
   meeting_id  duration  attendees    salary
0      755788      0.75          1  250000.0
1      755788      0.75          1  163808.0
2      653454      0.25          1  250000.0
3      653454      0.25          1  135000.0
4      653454      0.25          1  163808.0
```

Calculate Total Cost and Total Cost per Meeting

Now, we'll add columns for total cost and total cost per meeting.

In [3]:
```
# Calculate the total cost per meeting
df['total_cost'] = df['duration'] * df['attendees'] * df['salary']

# Sum the total cost per meeting_id
df['total_cost_per_meeting'] = df.groupby('meeting_id')['total_cost'].transform('sum')

# Define numeric columns
numeric_columns = ['duration', 'attendees', 'salary', 'total_cost', 'total_cost_per_meeting

# Display the updated DataFrame
print(df.head())
```

```
   meeting_id  duration  attendees    salary  total_cost  total_cost_per_meeting
0      755788      0.75          1  250000.0    187500.0                310356.0
1      755788      0.75          1  163808.0    122856.0                310356.0
2      653454      0.25          1  250000.0     62500.0                168452.0
3      653454      0.25          1  135000.0     33750.0                168452.0
4      653454      0.25          1  163808.0     40952.0                168452.0
```

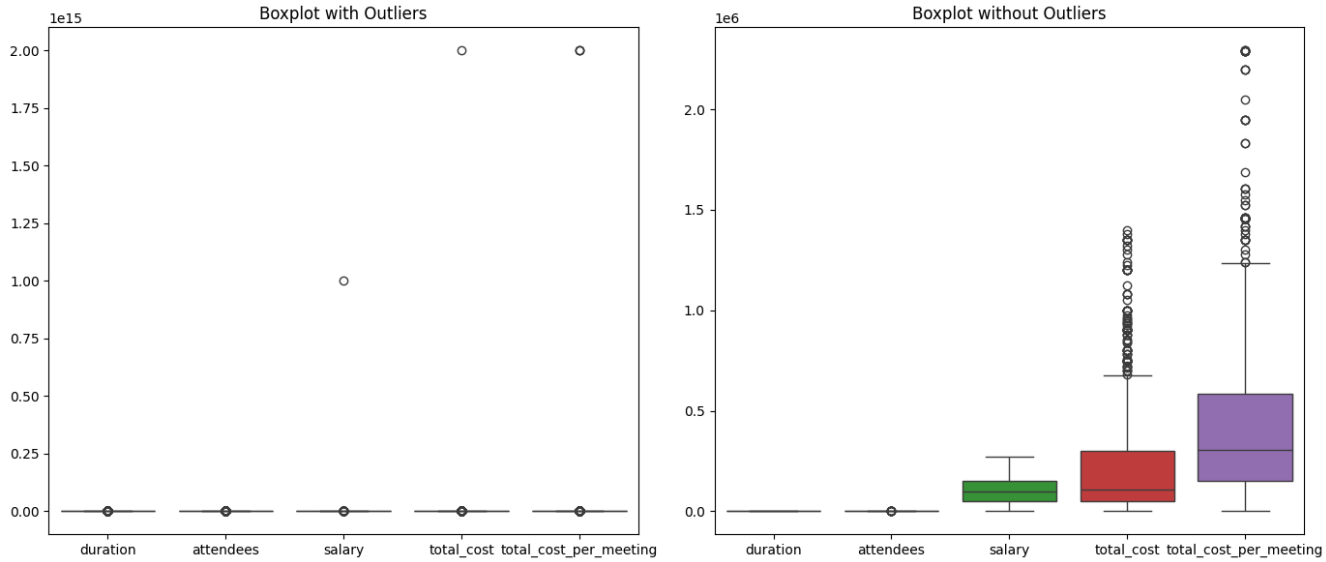
Outlier Detection and Removal

Now, let's identify and remove outliers using the Interquartile Range (IQR) method.

In [4]:
```
# Calculate Q1, Q3, and IQR for numeric columns
Q1 = df[numeric_columns].quantile(0.25)
Q3 = df[numeric_columns].quantile(0.75)
IQR = Q3 - Q1
print("Q1 values:")
print(Q1)
print("\nQ3 values:")
print(Q3)
print("\nIQR values:")
print(IQR)

# Define outliers as points outside 1.5 * IQR
outliers = ((df[numeric_columns] < (Q1 - 1.5 * IQR)) | (df[numeric_columns] > (Q3 + 1.5 * I
print("\nOutliers detected:")
print(outliers)

# Filter out the outliers
df_no_outliers = df.loc[~outliers]
print("\nDataFrame after removing outliers:")
print(df_no_outliers.head())

print(f"\nOriginal dataset size: {len(df)}")
print(f"Dataset size after removing outliers: {len(df_no_outliers)}")

# Print min/max of each data feature before outlier handling
print("\nMin/Max values before outlier handling:")
print(df[numeric_columns].agg(['min', 'max']))

# Print min/max of each data feature after outlier handling
print("\nMin/Max values after outlier handling:")
print(df_no_outliers[numeric_columns].agg(['min', 'max']))

# Visualize the outliers before and after removal
plt.figure(figsize=(14, 6))
```

```
# Boxplot with outliers
plt.subplot(1, 2, 1)
sns.boxplot(data=df[numeric_columns])
plt.title('Boxplot with Outliers')

# Boxplot without outliers
plt.subplot(1, 2, 2)
sns.boxplot(data=df_no_outliers[numeric_columns])
plt.title('Boxplot without Outliers')

plt.tight_layout()
plt.show()
```

```
Q1 values:
duration                       0.5
attendees                      1.0
salary                     60000.0
total_cost                 60000.0
total_cost_per_meeting    195000.0
Name: 0.25, dtype: float64

Q3 values:
duration                       1.0
attendees                      6.0
salary                    150000.0
total_cost                600000.0
total_cost_per_meeting   1060000.0
Name: 0.75, dtype: float64

IQR values:
duration                       0.5
attendees                      5.0
salary                     90000.0
total_cost                540000.0
total_cost_per_meeting    865000.0
dtype: float64

Outliers detected:
0       False
1       False
2       False
3       False
4       False
        ...
1801    False
1802    False
1803    False
1804     True
1805    False
Length: 1806, dtype: bool

DataFrame after removing outliers:
   meeting_id  duration  attendees    salary  total_cost  total_cost_per_meeting
0      755788      0.75          1  250000.0    187500.0                310356.0
1      755788      0.75          1  163808.0    122856.0                310356.0
2      653454      0.25          1  250000.0     62500.0                168452.0
3      653454      0.25          1  135000.0     33750.0                168452.0
4      653454      0.25          1  163808.0     40952.0                168452.0

Original dataset size: 1806
Dataset size after removing outliers: 1339

Min/Max values before outlier handling:
     duration  attendees        salary    total_cost  total_cost_per_meeting
min      0.25          0  0.000000e+00  0.000000e+00            0.000000e+00
max      8.00    1000000  1.000000e+15  2.000000e+15            2.000100e+15

Min/Max values after outlier handling:
     duration  attendees    salary  total_cost  total_cost_per_meeting
min      0.25          0       0.0         0.0                     0.0
max      1.75         13  270000.0   1400000.0               2293750.0
```

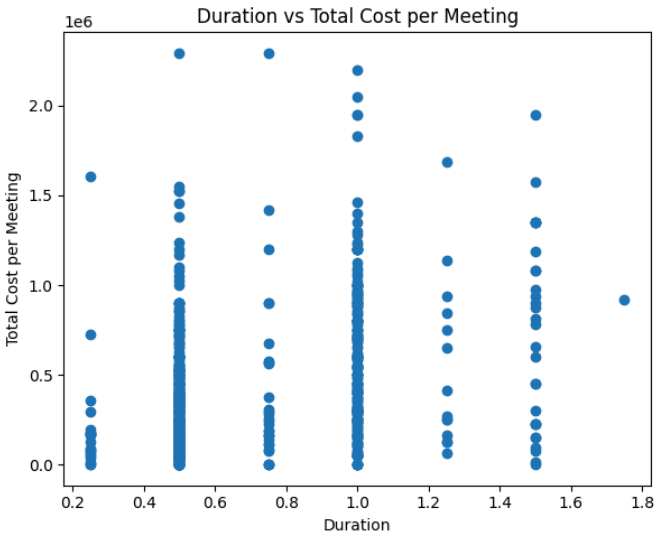


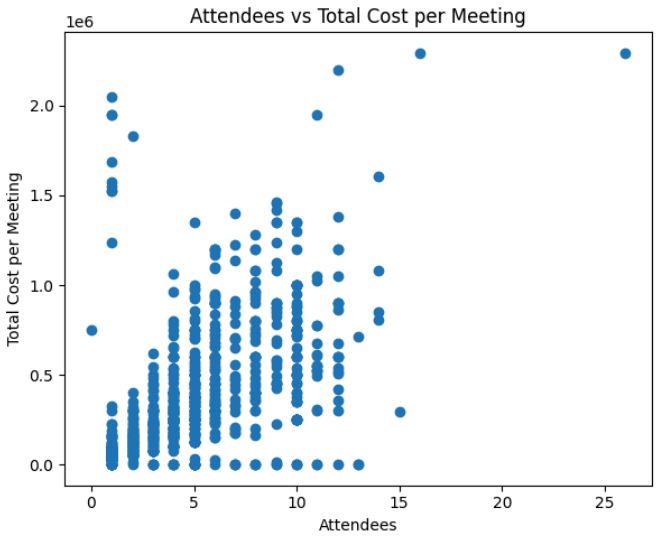
In [5]:
```
# After removing outliers, aggregate the total cost per meeting
df_no_outliers_agg = df_no_outliers.groupby('meeting_id').agg({
    'duration': 'mean',       # Mean of duration for each meeting
    'attendees': 'sum',       # Total attendees for each meeting
    'total_cost_per_meeting': 'first'  # Total cost per meeting (already aggregated)
}).reset_index()
```

Data Health, Integrity and Suitability Checks

Check Linearity

In [6]:
```
# Check linearity with scatterplots
plt.figure(figsize=(12, 5))

# Scatterplot for duration vs total cost per meeting
plt.subplot(1, 2, 1)
plt.scatter(df_no_outliers_agg['duration'], df_no_outliers_agg['total_cost_per_meeting'])
plt.title('Duration vs Total Cost per Meeting')
plt.xlabel('Duration')
plt.ylabel('Total Cost per Meeting')

# Scatterplot for attendees vs total cost per meeting
plt.subplot(1, 2, 2)
plt.scatter(df_no_outliers_agg['attendees'], df_no_outliers_agg['total_cost_per_meeting'])
plt.title('Attendees vs Total Cost per Meeting')
plt.xlabel('Attendees')
plt.ylabel('Total Cost per Meeting')

plt.tight_layout()
plt.show()
```

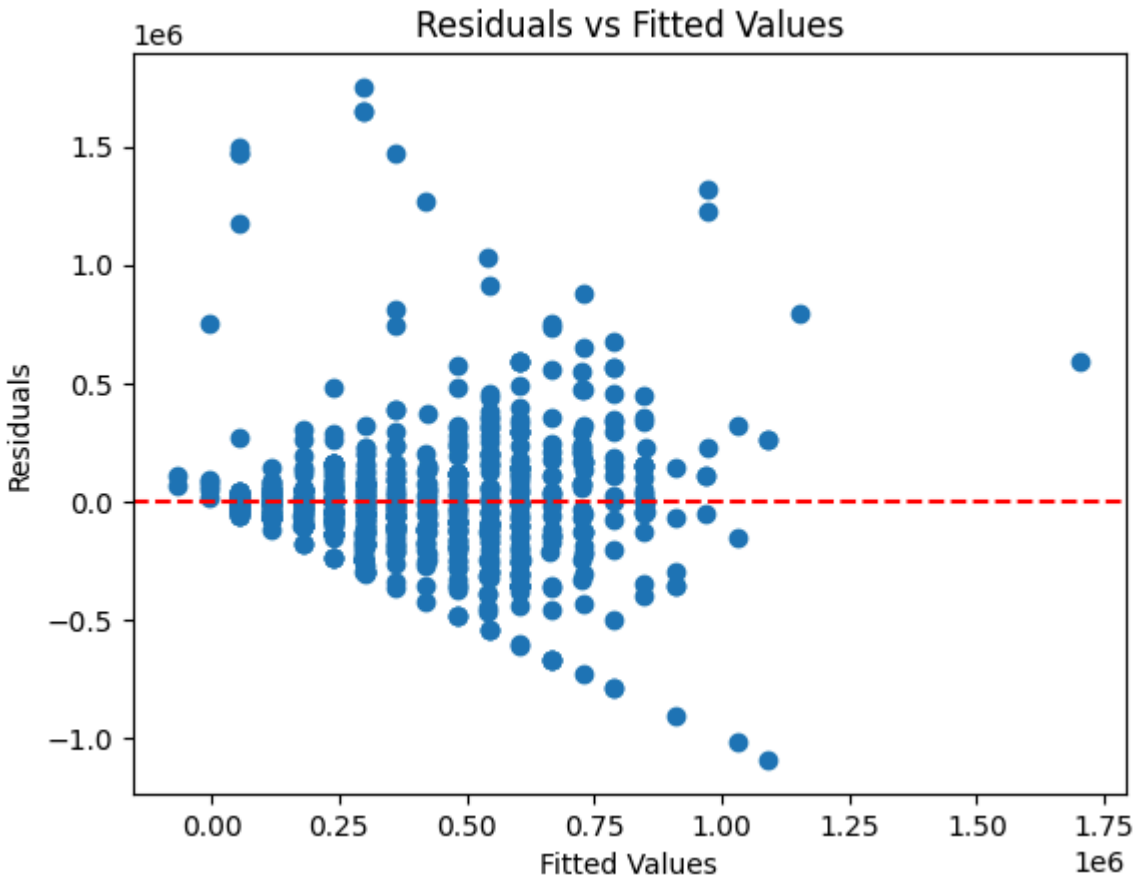




Check for Homoscedasticity

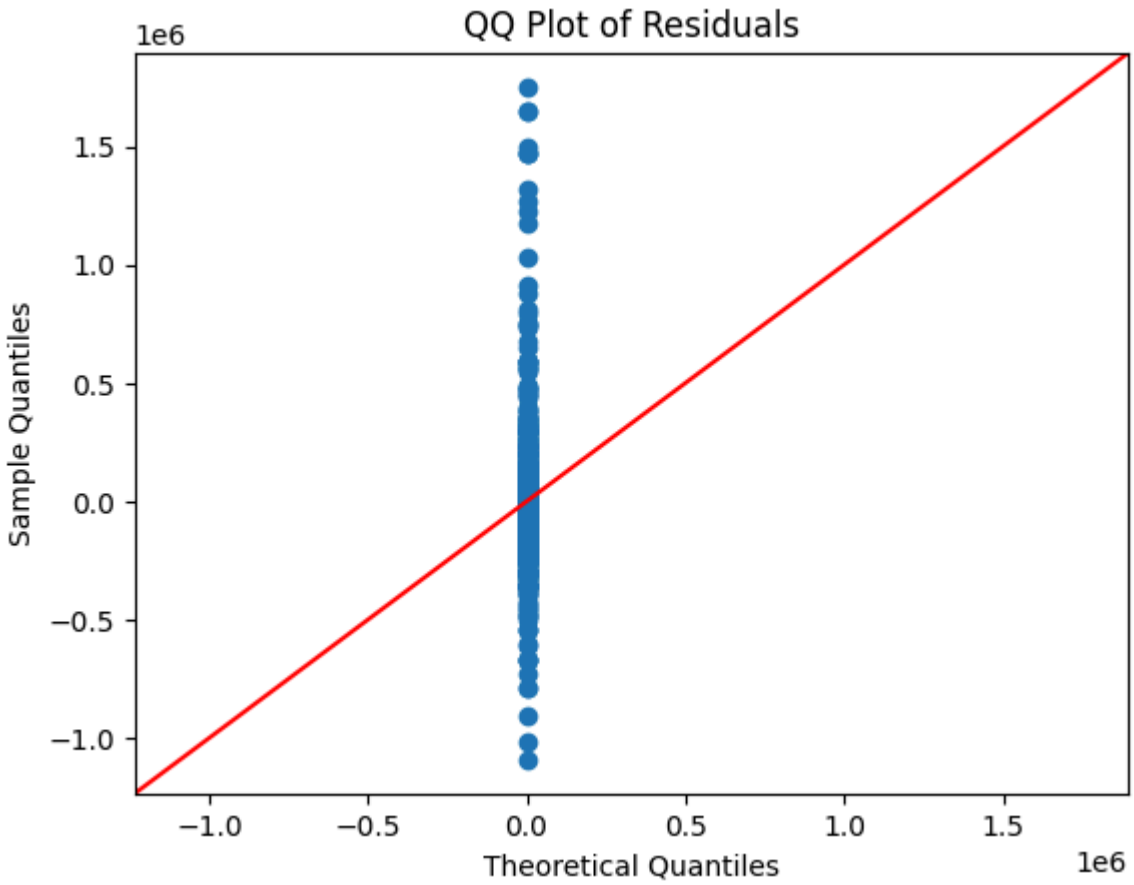
In [7]:

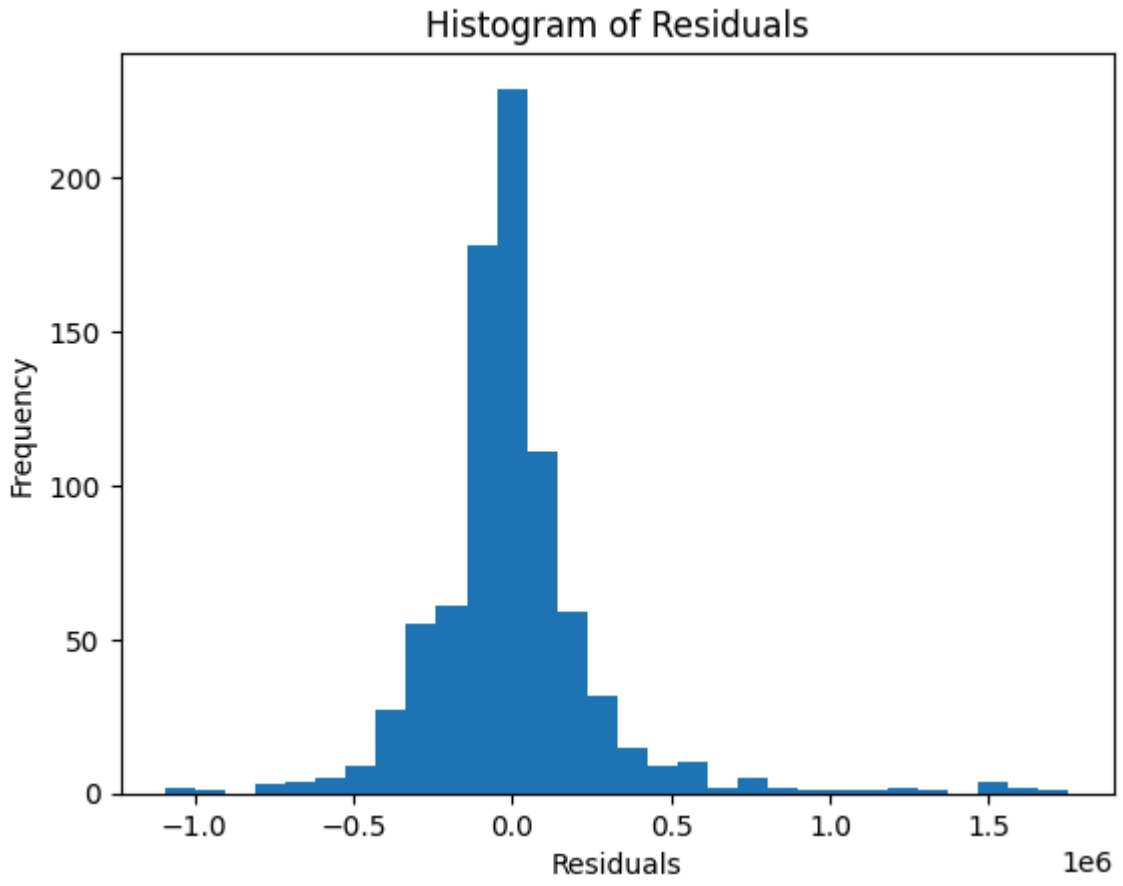
```
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression

# Define the independent variables (X) and the dependent variable (y)
X = df_no_outliers_agg[['duration', 'attendees']]  # Independent variables
y = df_no_outliers_agg['total_cost_per_meeting']  # Dependent variable

# Fit the Linear Regression model
model = LinearRegression()
model.fit(X, y)

# Predicted values and residuals
y_pred = model.predict(X)
residuals = y - y_pred

# Check if residuals are close to zero (residuals vs fitted values)
plt.scatter(y_pred, residuals)
plt.axhline(y=0, color='r', linestyle='--')
plt.title('Residuals vs Fitted Values')
plt.xlabel('Fitted Values')
plt.ylabel('Residuals')
plt.show()
```



Check for Normality of Residuals

In [8]:
```
# QQ plot for residuals
sm.qqplot(residuals, line='45')
plt.title('QQ Plot of Residuals')
plt.show()

# Histogram of residuals
plt.hist(residuals, bins=30)
plt.title('Histogram of Residuals')
plt.xlabel('Residuals')
plt.ylabel('Frequency')
plt.show()
```





Check for Multicollinearity

In [9]:
```
from statsmodels.stats.outliers_influence import variance_inflation_factor

# VIF for each independent variable
```

```
vif = pd.DataFrame()
vif['Variable'] = X.columns
vif['VIF'] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]

print(vif)
```

```
    Variable       VIF
0   duration  2.487538
1  attendees  2.487538
```

Check for Independence of Residuals

In [10]:
```
# Durbin-Watson test for autocorrelation
from statsmodels.stats.stattools import durbin_watson

dw_stat = durbin_watson(residuals)
print(f"Durbin-Watson statistic: {dw_stat}")
```

```
Durbin-Watson statistic: 2.058953314100529
```

Regression

In [11]:
```
# Prepare data for OLS regression
X = df_no_outliers_agg[['duration', 'attendees']]  # Independent variables
X = sm.add_constant(X)  # Add constant for intercept
y = df_no_outliers_agg['total_cost_per_meeting']  # Dependent variable

# Fit the OLS regression model
model = sm.OLS(y, X).fit()

# Output regression summary
print(model.summary())
```

```
                              OLS Regression Results
==============================================================================
Dep. Variable:     total_cost_per_meeting   R-squared:                       0.429
Model:                                OLS   Adj. R-squared:                  0.427
Method:                     Least Squares   F-statistic:                     310.8
Date:                    Sun, 20 Oct 2024   Prob (F-statistic):          1.89e-101
Time:                            16:29:24   Log-Likelihood:                -11614.
No. Observations:                     832   AIC:                         2.323e+04
Df Residuals:                         829   BIC:                         2.325e+04
Df Model:                               2
Covariance Type:                 nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const      -2.478e+05   2.91e+04     -8.515      0.000   -3.05e+05   -1.91e+05
duration    4.861e+05    3.4e+04     14.304      0.000    4.19e+05    5.53e+05
attendees   6.105e+04   2993.143     20.395      0.000    5.52e+04    6.69e+04
==============================================================================
Omnibus:                      401.747   Durbin-Watson:                   2.059
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             4141.647
Skew:                           1.933   Prob(JB):                         0.00
Kurtosis:                      13.224   Cond. No.                         26.4
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

Feature Engineering

Log Transformation and Interaction Terms

Now we will apply log transformations to stabilize the variance, create interaction terms between salary and attendees, and re-run the regression model.

In [12]:
```
# Ensure there are no zero or negative values before log transformation (apply a small shif
df_no_outliers_agg['log_total_cost_per_meeting'] = np.log(df_no_outliers_agg['total_cost_pe
df_no_outliers_agg['log_duration'] = np.log(df_no_outliers_agg['duration'] + 1)
df_no_outliers_agg['log_attendees'] = np.log(df_no_outliers_agg['attendees'] + 1)

# Create interaction term between salary and attendees (use original df for this)
df_no_outliers_agg['interaction_salary_attendees'] = df_no_outliers['salary'] * df_no_outli

# Check for infinite values or NaNs
print("Columns with infinite values:")
print(df_no_outliers_agg.columns[np.isinf(df_no_outliers_agg).any()])
print("\nColumns with NaN values:")
print(df_no_outliers_agg.columns[df_no_outliers_agg.isna().any()])

# Remove rows with infinite values or NaNs
df_no_outliers_agg = df_no_outliers_agg.replace([np.inf, -np.inf], np.nan).dropna()

# Prepare the data for regression including the interaction term
X_interaction = df_no_outliers_agg[['log_duration', 'log_attendees', 'interaction_salary_at
X_interaction = sm.add_constant(X_interaction)  # Add constant for intercept
y_interaction = df_no_outliers_agg['log_total_cost_per_meeting']

# Fit the new OLS regression model
model_interaction = sm.OLS(y_interaction, X_interaction).fit()

# Output the updated regression summary
print(model_interaction.summary())
```

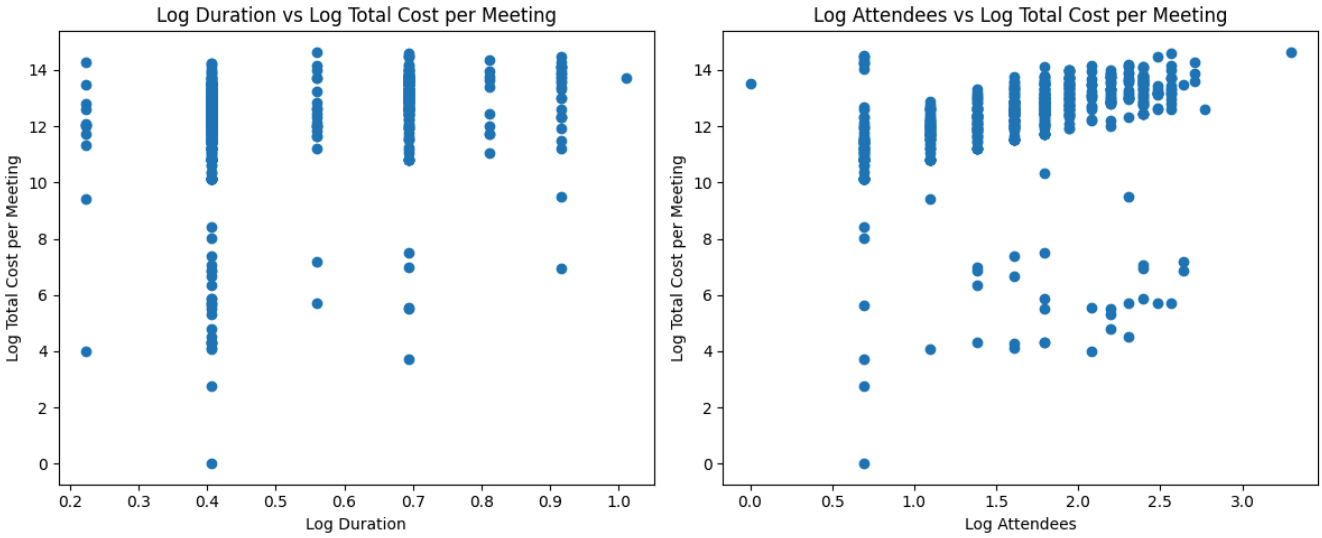
```
Columns with infinite values:
Index([], dtype='object')

Columns with NaN values:
Index(['interaction_salary_attendees'], dtype='object')
                            OLS Regression Results
==============================================================================
Dep. Variable:     log_total_cost_per_meeting   R-squared:                       0.156
Model:                                    OLS   Adj. R-squared:                  0.152
Method:                         Least Squares   F-statistic:                     38.43
Date:                        Sun, 20 Oct 2024   Prob (F-statistic):           8.54e-23
Time:                                16:29:24   Log-Likelihood:                -1235.9
No. Observations:                         627   AIC:                             2480.
Df Residuals:                             623   BIC:                             2498.
Df Model:                                   3
Covariance Type:                    nonrobust
================================================================================================
                                   coef    std err          t      P>|t|      [0.025      0.
975]
------------------------------------------------------------------------------------------------
const                            8.9668      0.318     28.203      0.000       8.342
9.591
log_duration                     2.7402      0.435      6.296      0.000       1.885
3.595
log_attendees                    1.0822      0.124      8.747      0.000       0.839
1.325
interaction_salary_attendees -3.489e-08   1.89e-07     -0.185      0.853   -4.05e-07    3.36
e-07
==============================================================================
Omnibus:                      462.967   Durbin-Watson:                   2.030
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             5099.974
Skew:                          -3.359   Prob(JB):                         0.00
Kurtosis:                      15.251   Cond. No.                     3.55e+06
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 3.55e+06. This might indicate that there are
strong multicollinearity or other numerical problems.
```

Check Linearity Post Log-Transformation

We will again check the linearity between the independent variables and the transformed total cost per meeting.

In [13]:
```
# Scatterplot for log-transformed duration vs log total cost per meeting
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.scatter(df_no_outliers_agg['log_duration'], df_no_outliers_agg['log_total_cost_per_meet
plt.title('Log Duration vs Log Total Cost per Meeting')
plt.xlabel('Log Duration')
plt.ylabel('Log Total Cost per Meeting')

# Scatterplot for log-transformed attendees vs log total cost per meeting
plt.subplot(1, 2, 2)
plt.scatter(df_no_outliers_agg['log_attendees'], df_no_outliers_agg['log_total_cost_per_mee
plt.title('Log Attendees vs Log Total Cost per Meeting')
```

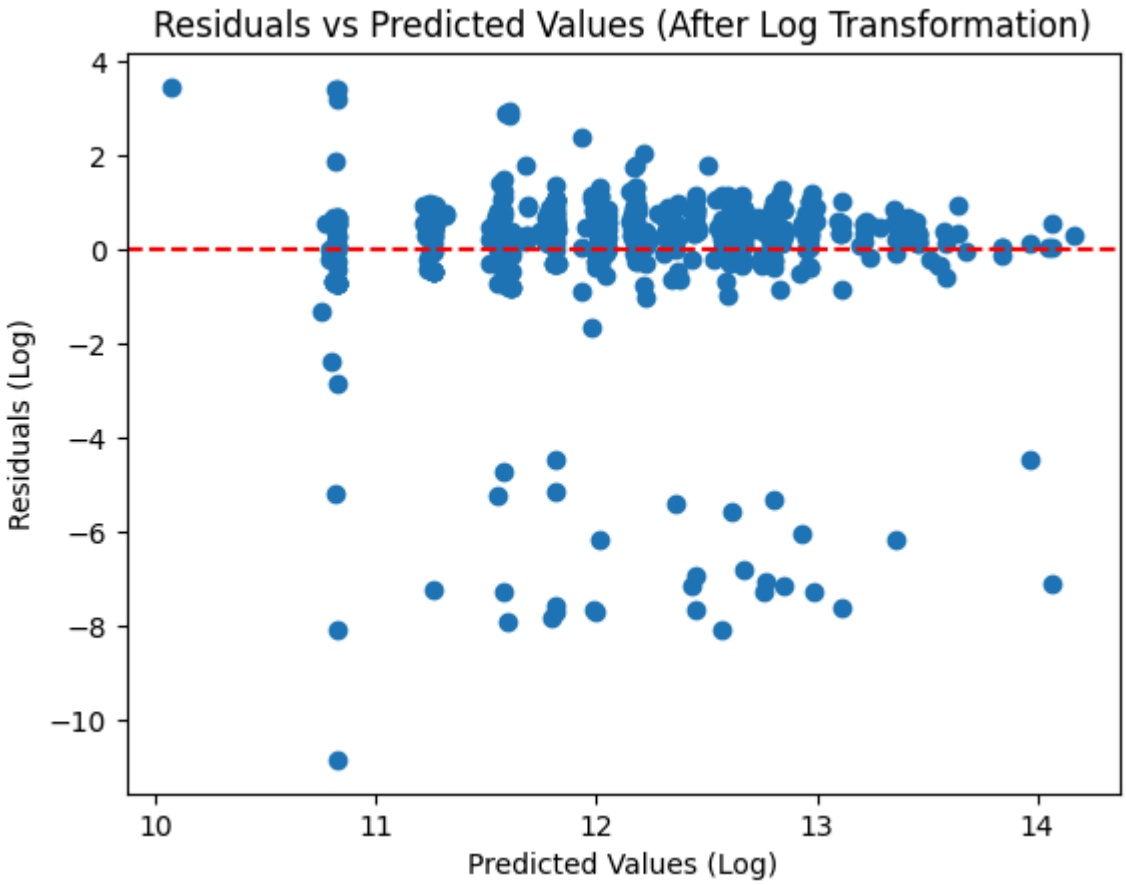
```
plt.xlabel('Log Attendees')
plt.ylabel('Log Total Cost per Meeting')

plt.tight_layout()
plt.show()
```



Residual Analysis After Transformation

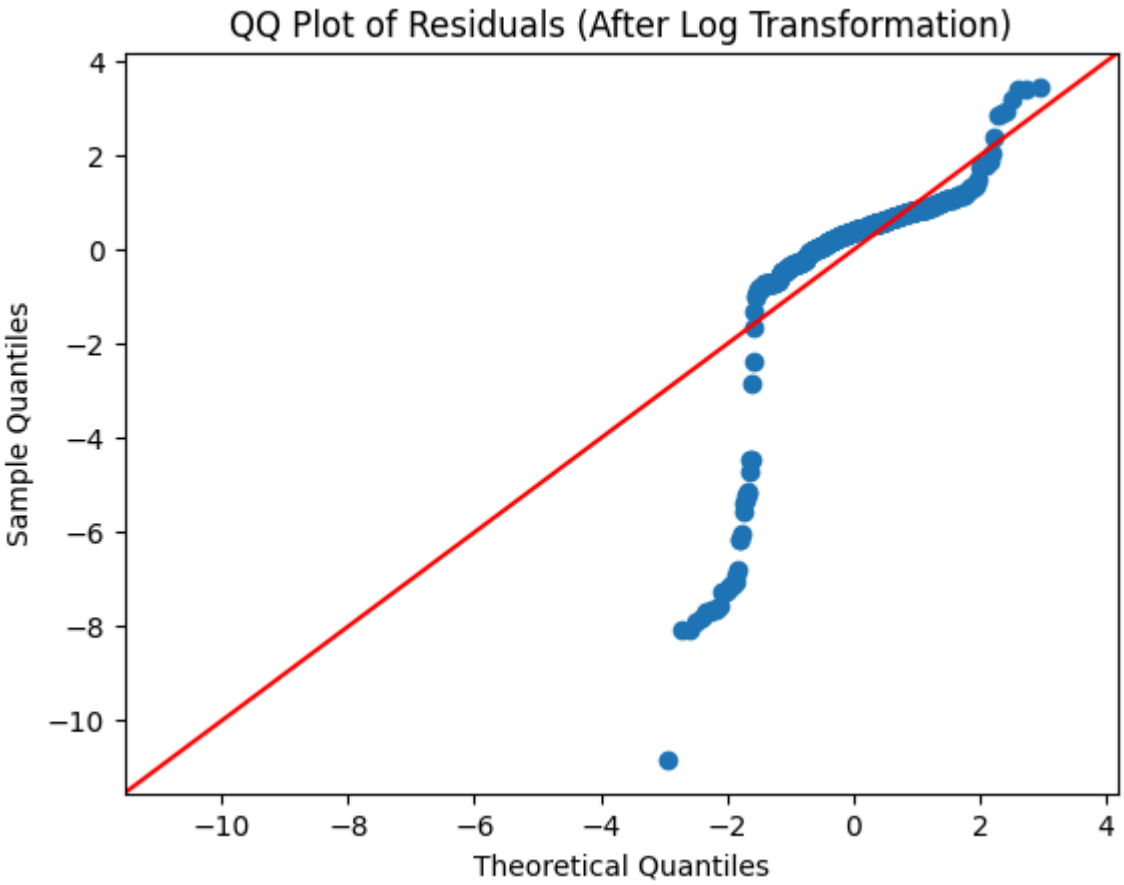
Let's check the residuals after the transformation and interaction term inclusion.

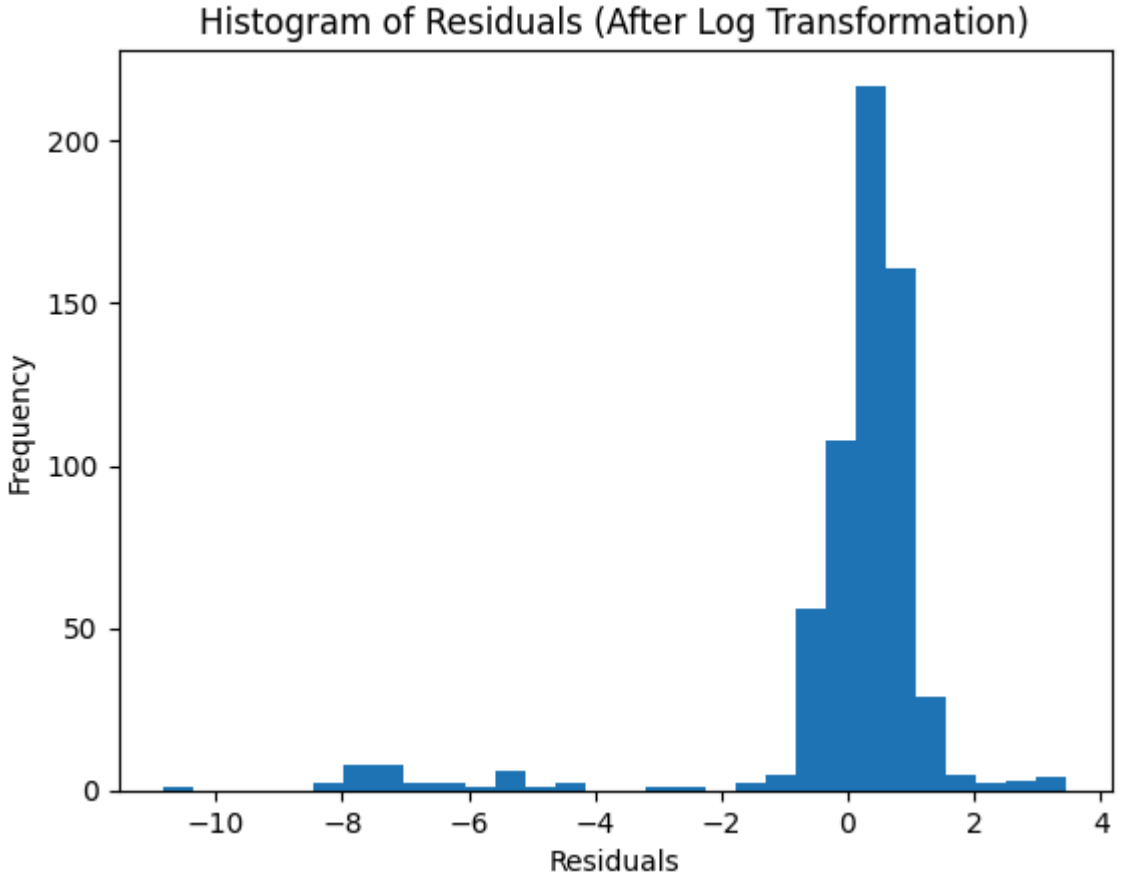
In [14]:
```
# Predicted values and residuals for the new model
y_pred_interaction = model_interaction.predict(X_interaction)
residuals_interaction = y_interaction - y_pred_interaction

# Residuals vs Predicted values plot (log-transformed model)
plt.scatter(y_pred_interaction, residuals_interaction)
plt.axhline(y=0, color='r', linestyle='--')
plt.title('Residuals vs Predicted Values (After Log Transformation)')
plt.xlabel('Predicted Values (Log)')
plt.ylabel('Residuals (Log)')
plt.show()
```



Check for Normality of Residuals

In [15]:
```
# QQ plot for residuals of the log-transformed model
sm.qqplot(residuals_interaction, line='45')
plt.title('QQ Plot of Residuals (After Log Transformation)')
plt.show()

# Histogram of residuals
plt.hist(residuals_interaction, bins=30)
plt.title('Histogram of Residuals (After Log Transformation)')
plt.xlabel('Residuals')
plt.ylabel('Frequency')
plt.show()
```





Re-check for Multicollinearity with VIF for Interaction Model

In [16]:
```
# Recalculate VIF for the interaction model
vif_interaction = pd.DataFrame()
vif_interaction['Variable'] = X_interaction.columns
vif_interaction['VIF'] = [variance_inflation_factor(X_interaction.values, i) for i in range

print(vif_interaction)
```

```
                      Variable        VIF
0                        const  20.869431
1                 log_duration   1.004161
2                log_attendees   1.000232
3  interaction_salary_attendees   1.004174
```

Feature Engineering: Salary Tiers

We'll create a new categorical column for salary tiers: Junior, Mid-level, Senior, Executive.

In [17]:
```
# Define salary bins and corresponding labels
salary_bins = [0, 75000, 150000, 300000, float('inf')]
salary_labels = ['Junior', 'Mid-level', 'Senior', 'Executive']

# Create a new column 'salary_tier' based on the bins
df_no_outliers['salary_tier'] = pd.cut(df_no_outliers['salary'], bins=salary_bins, labels=s

# Check the distribution of the new salary tier feature
print(df_no_outliers['salary_tier'].value_counts())

# Display the first few rows to verify the new column
print(df_no_outliers.head())
```

```
salary_tier
Mid-level    577
Junior       403
Senior       359
Executive      0
Name: count, dtype: int64
   meeting_id  duration  attendees    salary  total_cost  total_cost_per_meeting salary_tier
0      755788      0.75          1  250000.0    187500.0                310356.0      Senior
1      755788      0.75          1  163808.0    122856.0                310356.0      Senior
2      653454      0.25          1  250000.0     62500.0                168452.0      Senior
3      653454      0.25          1  135000.0     33750.0                168452.0   Mid-level
4      653454      0.25          1  163808.0     40952.0                168452.0      Senior
```

```
C:\Users\kipjo\AppData\Local\Temp\ipykernel_108560\2800376173.py:6: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guid
e/indexing.html#returning-a-view-versus-a-copy
  df_no_outliers['salary_tier'] = pd.cut(df_no_outliers['salary'], bins=salary_bins, labels=
salary_labels, right=False)
```

Model Development

Random Forest Regression

In [18]:
```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score

# Prepare the dataset: we'll use numeric columns and the new salary_tier categorical featur
# First, we need to convert the salary_tier to dummy variables (one-hot encoding)
df_no_outliers_encoded = pd.get_dummies(df_no_outliers, columns=['salary_tier'], drop_first

# Define the features (X) and the target (y)
X_rf = df_no_outliers_encoded[['duration', 'attendees', 'salary_tier_Mid-level', 'salary_ti
y_rf = df_no_outliers_encoded['total_cost_per_meeting']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_rf, y_rf, test_size=0.3, random_state

# Initialize the Random Forest model
rf_model = RandomForestRegressor(n_estimators=100, random_state=42)

# Train the model
rf_model.fit(X_train, y_train)

# Make predictions
y_pred_train = rf_model.predict(X_train)
y_pred_test = rf_model.predict(X_test)

# Evaluate the model on training and testing sets
train_rmse = mean_squared_error(y_train, y_pred_train, squared=False)  # RMSE
test_rmse = mean_squared_error(y_test, y_pred_test, squared=False)     # RMSE
train_r2 = r2_score(y_train, y_pred_train)
test_r2 = r2_score(y_test, y_pred_test)

# Display the evaluation metrics
print(f"Training RMSE: {train_rmse:.2f}")
print(f"Test RMSE: {test_rmse:.2f}")
print(f"Training R-squared: {train_r2:.4f}")
print(f"Test R-squared: {test_r2:.4f}")
```

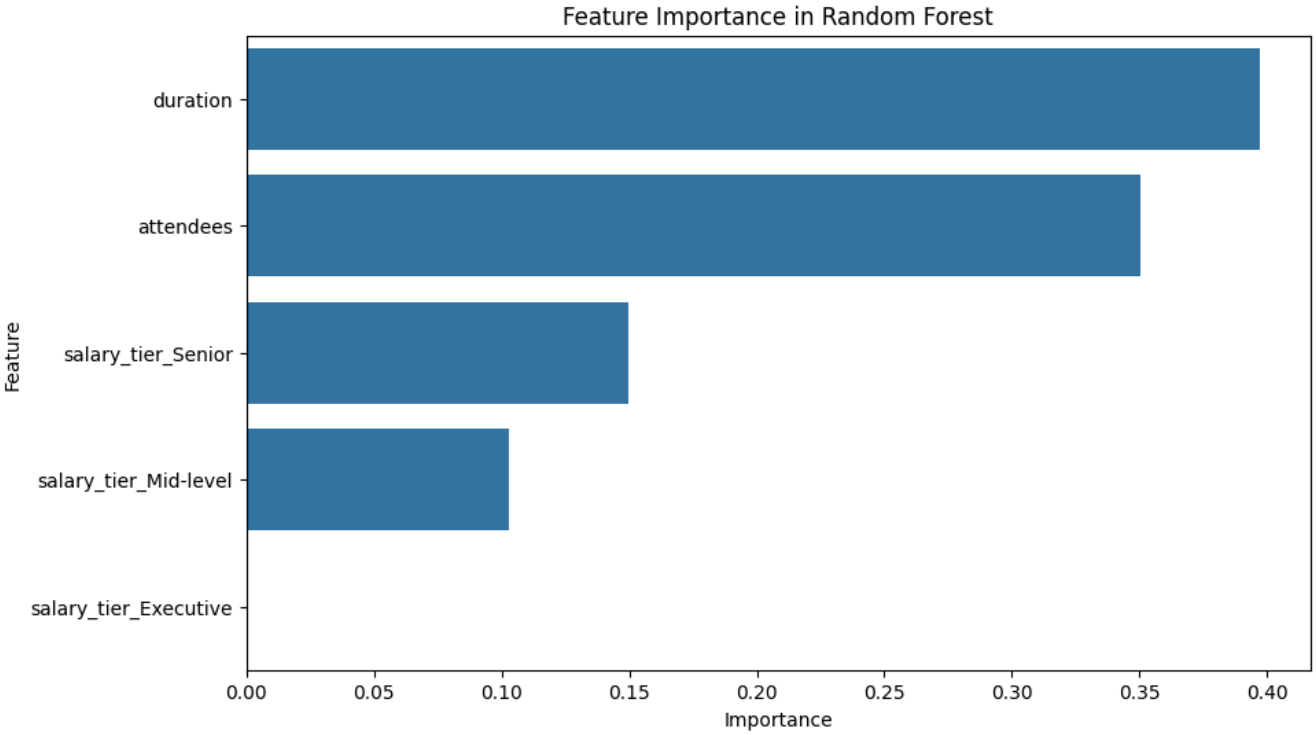
```
Training RMSE: 297021.41
Test RMSE: 310295.42
Training R-squared: 0.4410
Test R-squared: 0.3382
c:\Users\kipjo\AppData\Local\Programs\Python\Python312\Lib\site-packages\sklearn\metrics\_re
gression.py:492: FutureWarning: 'squared' is deprecated in version 1.4 and will be removed i
n 1.6. To calculate the root mean squared error, use the function'root_mean_squared_error'.
  warnings.warn(
c:\Users\kipjo\AppData\Local\Programs\Python\Python312\Lib\site-packages\sklearn\metrics\_re
gression.py:492: FutureWarning: 'squared' is deprecated in version 1.4 and will be removed i
n 1.6. To calculate the root mean squared error, use the function'root_mean_squared_error'.
  warnings.warn(
```

Feature Importance

Let's look at which features are most important in the Random Forest model.

In [19]:
```
importances = rf_model.feature_importances_
feature_names = X_rf.columns
importance_df = pd.DataFrame({'Feature': feature_names, 'Importance': importances}).sort_va

# Plot feature importance
```

```
plt.figure(figsize=(10, 6))
sns.barplot(x='Importance', y='Feature', data=importance_df)
plt.title('Feature Importance in Random Forest')
plt.show()
```



Model Evaluation and Comparison

Hyperparameter tuning for Random Forest

In [20]:

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import GridSearchCV

# Define the parameter grid for tuning
param_grid = {
    'n_estimators': [100, 200, 300],   # Number of trees in the forest
    'max_depth': [10, 20, 30, None],   # Maximum depth of the trees
    'min_samples_split': [2, 5, 10],   # Minimum number of samples required to split a node
}

# Initialize the Random Forest Regressor
rf = RandomForestRegressor(random_state=42)

# Use GridSearchCV to find the best parameters
grid_search = GridSearchCV(estimator=rf, param_grid=param_grid,
                           scoring='neg_mean_squared_error', cv=5, verbose=2, n_jobs=-1)

# Fit the model
grid_search.fit(X_train, y_train)

# Output the best parameters
print(f"Best parameters from grid search: {grid_search.best_params_}")

# Evaluate the tuned model on the test set
best_rf = grid_search.best_estimator_
y_pred_rf = best_rf.predict(X_test)
```

```
# Calculate RMSE and R-squared
rmse_rf_tuned = np.sqrt(mean_squared_error(y_test, y_pred_rf))
r2_rf_tuned = r2_score(y_test, y_pred_rf)
print(f"Tuned Random Forest RMSE: {rmse_rf_tuned}")
print(f"Tuned Random Forest R-squared: {r2_rf_tuned}")
```

```
Fitting 5 folds for each of 36 candidates, totalling 180 fits
Best parameters from grid search: {'max_depth': 10, 'min_samples_split': 10, 'n_estimators':
100}
Tuned Random Forest RMSE: 306520.6583940997
Tuned Random Forest R-squared: 0.3541652822999366
```

Testing Gradient Boosting with XGBoost

In [21]:
```
from xgboost import XGBRegressor

# Initialize the XGBoost regressor
xgb_model = XGBRegressor(random_state=42, n_estimators=200, learning_rate=0.1, max_depth=5)

# Fit the model
xgb_model.fit(X_train, y_train)

# Predict using the test data
y_pred_xgb = xgb_model.predict(X_test)

# Calculate RMSE and R-squared for XGBoost
rmse_xgb = np.sqrt(mean_squared_error(y_test, y_pred_xgb))
r2_xgb = r2_score(y_test, y_pred_xgb)

print(f"XGBoost RMSE: {rmse_xgb}")
print(f"XGBoost R-squared: {r2_xgb}")
```

```
XGBoost RMSE: 322589.94143180357
XGBoost R-squared: 0.28467478960968706
```

Cross-validation for Random Forest and XGBoost

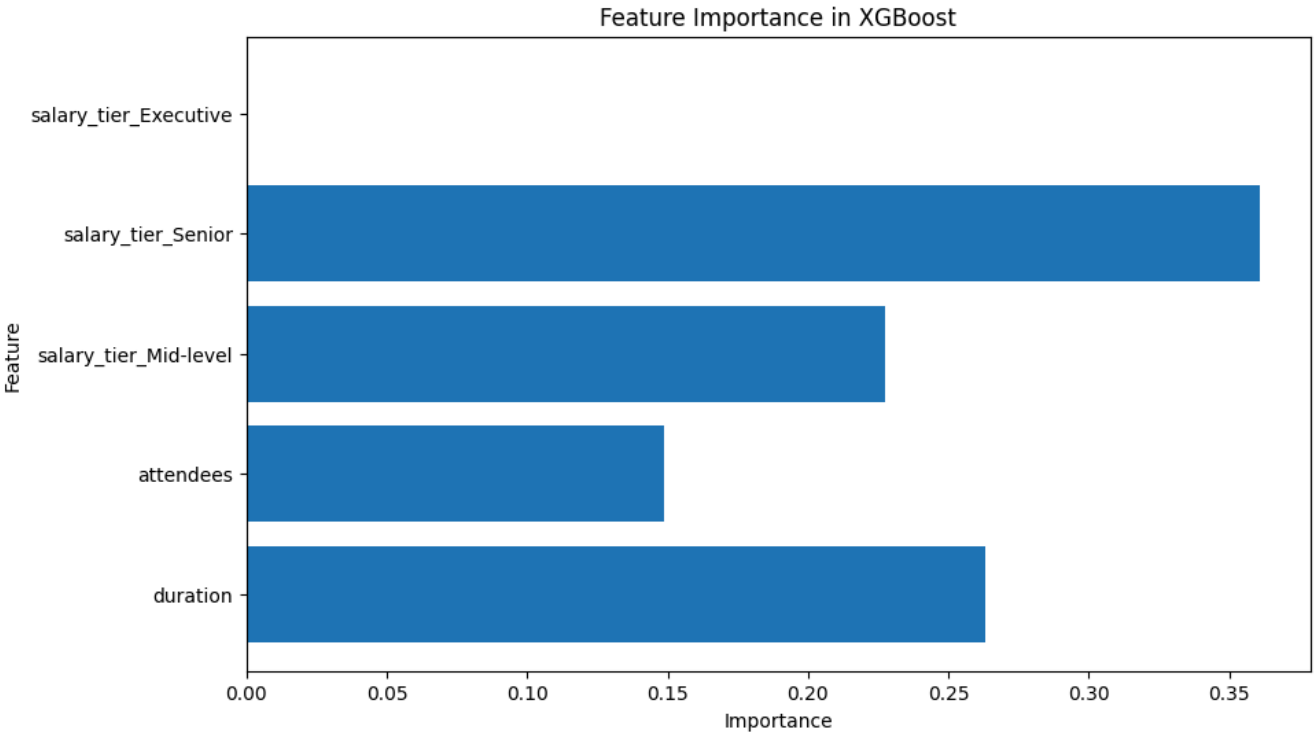
In [22]:
```
from sklearn.model_selection import cross_val_score

# Cross-validate Random Forest
cv_rf_scores = cross_val_score(best_rf, X, y, cv=5, scoring='neg_mean_squared_error')
cv_rf_rmse = np.sqrt(-cv_rf_scores)
print(f"Cross-validated RMSE for Random Forest: {cv_rf_rmse.mean()}")

# Cross-validate XGBoost
cv_xgb_scores = cross_val_score(xgb_model, X, y, cv=5, scoring='neg_mean_squared_error')
cv_xgb_rmse = np.sqrt(-cv_xgb_scores)
print(f"Cross-validated RMSE for XGBoost: {cv_xgb_rmse.mean()}")
```

```
Cross-validated RMSE for Random Forest: 288807.5933110194
Cross-validated RMSE for XGBoost: 304033.7442193608
```

Feature Importance for XGBoost

In [23]:
```
xgb_importance = xgb_model.feature_importances_

# Plot the feature importances
```

```
plt.figure(figsize=(10, 6))
plt.barh(X_train.columns, xgb_importance)
plt.title("Feature Importance in XGBoost")
plt.xlabel("Importance")
plt.ylabel("Feature")
plt.show()
```



Comparison of Models

In [24]:
```
# %% Comparison of models
print(f"Tuned Random Forest RMSE: {rmse_rf_tuned}")
print(f"Tuned Random Forest R-squared: {r2_rf_tuned}")
print(f"Cross-validated Random Forest RMSE: {cv_rf_rmse.mean()}")

print(f"\nXGBoost RMSE: {rmse_xgb}")
print(f"XGBoost R-squared: {r2_xgb}")
print(f"Cross-validated XGBoost RMSE: {cv_xgb_rmse.mean()}")
```

```
Tuned Random Forest RMSE: 306520.6583940997
Tuned Random Forest R-squared: 0.3541652822999366
Cross-validated Random Forest RMSE: 288807.5933110194

XGBoost RMSE: 322589.94143180357
XGBoost R-squared: 0.28467478960968706
Cross-validated XGBoost RMSE: 304033.7442193608
```

Standardise features

In [25]:
```
from sklearn.preprocessing import StandardScaler

# Standardize duration and attendees
scaler = StandardScaler()
df_no_outliers_agg[['duration', 'attendees']] = scaler.fit_transform(df_no_outliers_agg[['d
```

Test Log Transformation for Skewed Features

In [26]:
```
# Apply log transformation to reduce skewness
df_no_outliers_agg['log_total_cost_per_meeting'] = np.log(df_no_outliers_agg['total_cost_pe
```

Handling Missing Data

In [27]:
```
df_no_outliers_agg.fillna(df_no_outliers_agg.mean())
```

Out[27]:

	meeting_id	duration	attendees	total_cost_per_meeting	log_total_cost_per_meeting	log_durat
0	101675	2.804490	1.544454	1050.0	6.957497	0.916
1	104083	-0.629915	1.544454	250000.0	12.429220	0.405
2	104571	1.087288	-1.242676	1950000.0	14.483340	0.693
3	106402	1.087288	1.544454	1000000.0	13.815512	0.693
4	109135	-0.629915	-1.242676	40000.0	10.596660	0.405
...	...	...	...	...	...	
827	996580	-0.629915	-0.003951	500000.0	13.122365	0.405
828	996847	1.087288	-0.932995	300000.0	12.611541	0.693
829	997563	-0.629915	-0.003951	375000.0	12.834684	0.405
830	997813	-0.629915	-0.313632	208500.0	12.247699	0.405
831	999152	-0.629915	-0.003951	250000.0	12.429220	0.405

627 rows × 8 columns

Remove Unnecessary Features

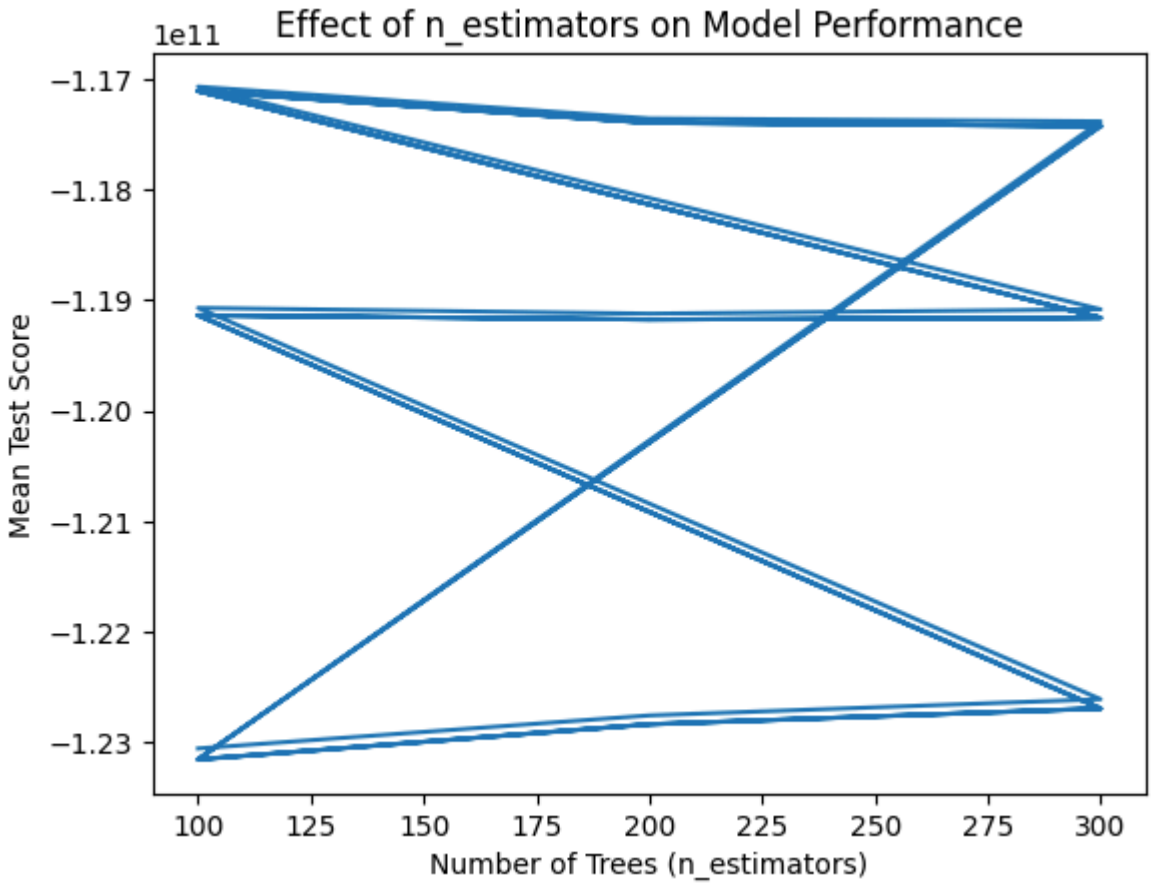
In [28]:
```
# Drop unnecessary columns
df_final = df_no_outliers_agg.drop(columns=['meeting_id', 'total_cost_per_meeting'])
```

Final Model and Insights

Hyperparameter tuning

n estimators

In [29]:
```
results = grid_search.cv_results_
plt.plot(results['param_n_estimators'], results['mean_test_score'])
plt.xlabel('Number of Trees (n_estimators)')
plt.ylabel('Mean Test Score')
plt.title('Effect of n_estimators on Model Performance')
plt.show()
```



Diagnosing Best Parameters with GridSearchCV

In [30]:
```
# Perform grid search (already done)
grid_search.fit(X_train, y_train)

# Access the results
cv_results = pd.DataFrame(grid_search.cv_results_)

# Sort the results to see the top-performing combinations
cv_results = cv_results.sort_values('rank_test_score')

# Display the 10 hyperparameter combinations with mean_test_score closest to zero
cv_results['abs_mean_test_score'] = cv_results['mean_test_score'].abs()
cv_results_sorted = cv_results.sort_values(by='abs_mean_test_score')
print(cv_results_sorted[['params', 'mean_test_score', 'std_test_score']].head(10))
```

```
Fitting 5 folds for each of 36 candidates, totalling 180 fits
                                               params  mean_test_score  std_test_score
6   {'max_depth': 10, 'min_samples_split': 10, 'n_...    -1.170702e+11    2.313774e+10
15  {'max_depth': 20, 'min_samples_split': 10, 'n_...    -1.171100e+11    2.314837e+10
24  {'max_depth': 30, 'min_samples_split': 10, 'n_...    -1.171100e+11    2.314837e+10
33  {'max_depth': None, 'min_samples_split': 10, '...    -1.171100e+11    2.314837e+10
7   {'max_depth': 10, 'min_samples_split': 10, 'n_...    -1.173579e+11    2.335401e+10
8   {'max_depth': 10, 'min_samples_split': 10, 'n_...    -1.173836e+11    2.320294e+10
25  {'max_depth': 30, 'min_samples_split': 10, 'n_...    -1.173971e+11    2.334660e+10
16  {'max_depth': 20, 'min_samples_split': 10, 'n_...    -1.173971e+11    2.334660e+10
34  {'max_depth': None, 'min_samples_split': 10, '...    -1.173971e+11    2.334660e+10
35  {'max_depth': None, 'min_samples_split': 10, '...    -1.174271e+11    2.320544e+10
```

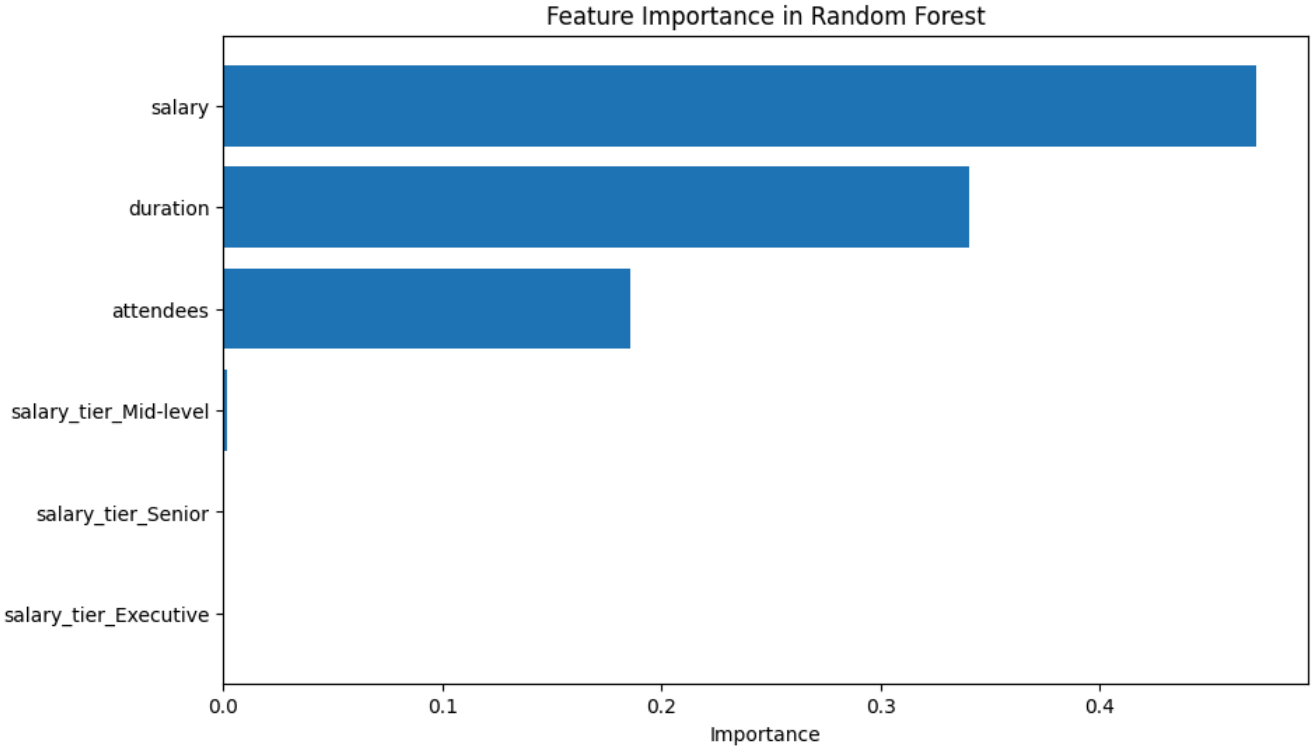
Finalised Model

In [31]:

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split, cross_val_score, GridSearchCV
from sklearn.metrics import mean_squared_error, r2_score
import numpy as np

# One-hot encode the 'salary_tier' column and ensure relevant features are included
# We'll drop columns like 'total_cost', 'meeting_id', or any others irrelevant to the predi
df_no_outliers_encoded = pd.get_dummies(df_no_outliers, columns=['salary_tier'], drop_first

# Select only the relevant independent variables for the model
X = df_no_outliers_encoded[['duration', 'attendees', 'salary', 'salary_tier_Mid-level', 'sa
y = df_no_outliers_encoded['total_cost_per_meeting']  # Dependent variable

# Split into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Implement cross-validation and hyperparameter tuning with GridSearchCV
param_grid = {
    'n_estimators': [50, 75, 100, 125, 150, 175, 200, 225, 250, 275, 300, 325, 350, 375, 40
    'max_depth': [5, 7, 10, 12, 15, 17, 20, 22, 25, 27, 30, 32, 35, 37, 40],
    'min_samples_split': [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16]
}
# Set up the GridSearchCV with 5-fold cross-validation
rf = RandomForestRegressor(random_state=42)
grid_search = GridSearchCV(estimator=rf, param_grid=param_grid, cv=5, n_jobs=-1, scoring='r

# Fit the model to the training data
grid_search.fit(X_train, y_train)

# Get the best parameters
best_params = grid_search.best_params_
print(f"Best parameters from grid search: {best_params}")

# Train Random Forest Regressor with the best hyperparameters found
rf_best = RandomForestRegressor(
    n_estimators=best_params['n_estimators'],
    max_depth=best_params['max_depth'],
    min_samples_split=best_params['min_samples_split'],
    random_state=42
)
rf_best.fit(X_train, y_train)

# Make predictions on training and test sets
y_train_pred = rf_best.predict(X_train)
y_test_pred = rf_best.predict(X_test)

# Evaluate performance
train_rmse = np.sqrt(mean_squared_error(y_train, y_train_pred))
test_rmse = np.sqrt(mean_squared_error(y_test, y_test_pred))
train_r2 = r2_score(y_train, y_train_pred)
test_r2 = r2_score(y_test, y_test_pred)

print(f"Training RMSE: {train_rmse:.4f}")
print(f"Test RMSE: {test_rmse:.4f}")
print(f"Training R-squared: {train_r2:.4f}")
print(f"Test R-squared: {test_r2:.4f}")

# Cross-Validation Scores for better understanding of performance
cv_scores = cross_val_score(rf_best, X_train, y_train, cv=5, scoring='neg_mean_squared_erro
cv_rmse = np.sqrt(-cv_scores)
print(f"Cross-validated RMSE (5-fold): {cv_rmse.mean():.4f} ± {cv_rmse.std():.4f}")
```

```
# Feature Importance
feature_importances = rf_best.feature_importances_
features = X.columns

# Sort the features by importance
sorted_idx = np.argsort(feature_importances)[::-1]

# Plot the feature importances
import matplotlib.pyplot as plt
plt.figure(figsize=(10, 6))
plt.barh(range(len(sorted_idx)), feature_importances[sorted_idx], align='center')
plt.yticks(range(len(sorted_idx)), [features[i] for i in sorted_idx])
plt.xlabel('Importance')
plt.title('Feature Importance in Random Forest')
plt.gca().invert_yaxis()
plt.show()
```

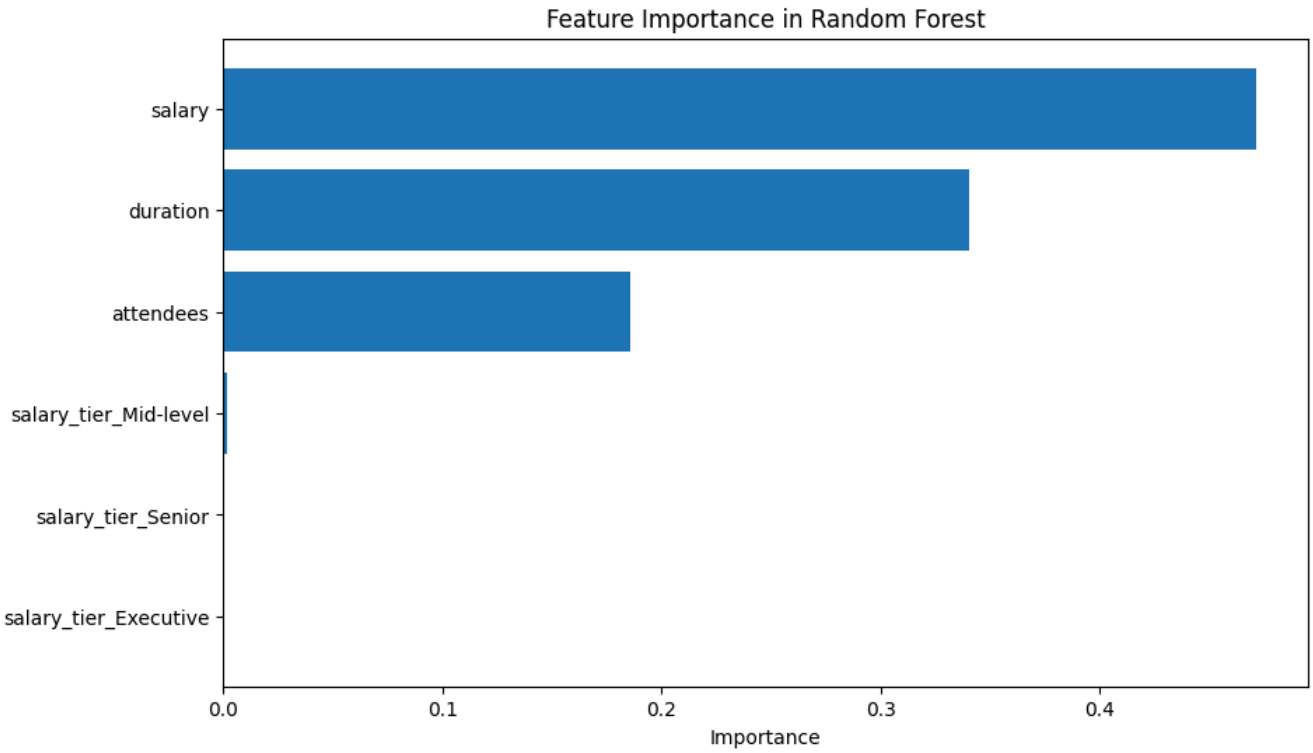
```
c:\Users\kipjo\AppData\Local\Programs\Python\Python312\Lib\site-packages\numpy\ma\core.py:2846: RuntimeWarning: invalid value encountered in cast
  _data = np.array(data, dtype=dtype, copy=copy,
Best parameters from grid search: {'max_depth': 5, 'min_samples_split': 16, 'n_estimators': 50}
Training RMSE: 302786.0180
Test RMSE: 294204.9445
Training R-squared: 0.4047
Test R-squared: 0.4398
Cross-validated RMSE (5-fold): 329497.1301 ± 45122.7276
```



In [32]:
```
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.metrics import mean_squared_error, r2_score
import numpy as np
import matplotlib.pyplot as plt

# One-hot encode the 'salary_tier' column and ensure relevant features are included
df_no_outliers_encoded = pd.get_dummies(df_no_outliers, columns=['salary_tier'], drop_first

# Remove irrelevant columns like 'meeting_id' and 'total_cost'
```

```
X = df_no_outliers_encoded[['duration', 'attendees', 'salary', 'salary_tier_Mid-level', 'sa
y = df_no_outliers_encoded['total_cost_per_meeting']  # Dependent variable

# Split into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Apply the best hyperparameters from GridSearchCV
rf_best = RandomForestRegressor(
    n_estimators=50,  # From GridSearchCV results
    max_depth=5,      # From GridSearchCV results
    min_samples_split=16,  # From GridSearchCV results
    random_state=42
)

# Train the Random Forest model
rf_best.fit(X_train, y_train)

# Make predictions on training and test sets
y_train_pred = rf_best.predict(X_train)
y_test_pred = rf_best.predict(X_test)

# Evaluate performance
train_rmse = np.sqrt(mean_squared_error(y_train, y_train_pred))
test_rmse = np.sqrt(mean_squared_error(y_test, y_test_pred))
train_r2 = r2_score(y_train, y_train_pred)
test_r2 = r2_score(y_test, y_test_pred)

print(f"Training RMSE: {train_rmse:.4f}")
print(f"Test RMSE: {test_rmse:.4f}")
print(f"Training R-squared: {train_r2:.4f}")
print(f"Test R-squared: {test_r2:.4f}")

# Cross-Validation Scores for better understanding of performance
cv_scores = cross_val_score(rf_best, X_train, y_train, cv=5, scoring='neg_mean_squared_erro
cv_rmse = np.sqrt(-cv_scores)
print(f"Cross-validated RMSE (5-fold): {cv_rmse.mean():.4f} ± {cv_rmse.std():.4f}")

# Feature Importance
feature_importances = rf_best.feature_importances_
features = X.columns

# Sort the features by importance
sorted_idx = np.argsort(feature_importances)[::-1]

# Plot the feature importances
plt.figure(figsize=(10, 6))
plt.barh(range(len(sorted_idx)), feature_importances[sorted_idx], align='center')
plt.yticks(range(len(sorted_idx)), [features[i] for i in sorted_idx])
plt.xlabel('Importance')
plt.title('Feature Importance in Random Forest')
plt.gca().invert_yaxis()
plt.show()
```

```
Training RMSE: 302786.0180
Test RMSE: 294204.9445
Training R-squared: 0.4047
Test R-squared: 0.4398
Cross-validated RMSE (5-fold): 329497.1301 ± 45122.7276
```



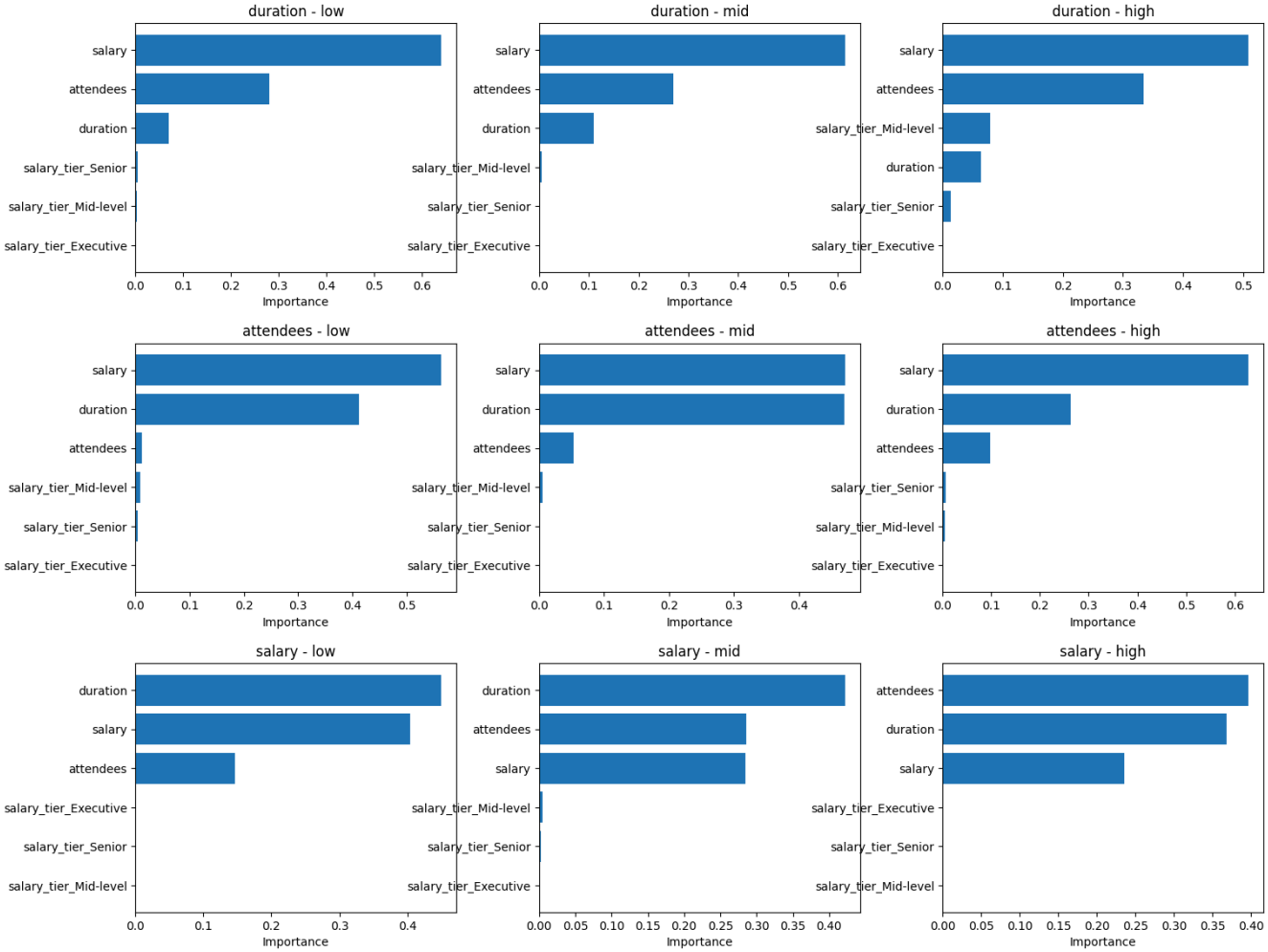
In [33]:

```python
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split, cross_val_score, GridSearchCV
from sklearn.metrics import mean_squared_error, r2_score
import numpy as np
import matplotlib.pyplot as plt

# Apply IQR for segmentation across multiple columns
def apply_iqr_segment(df, columns):
    segments = {}
    for column in columns:
        Q1 = df[column].quantile(0.25)
        Q3 = df[column].quantile(0.75)
        segments[column] = {
            'low': df[df[column] <= Q1],
            'mid': df[(df[column] > Q1) & (df[column] <= Q3)],
            'high': df[df[column] > Q3]
        }
    return segments

# Function to train and evaluate model
def train_and_evaluate(X, y, param_grid):
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=4

    # Set up the GridSearchCV with 5-fold cross-validation
    rf = RandomForestRegressor(random_state=42)
    grid_search = GridSearchCV(estimator=rf, param_grid=param_grid, cv=5, n_jobs=-1, scorir
    grid_search.fit(X_train, y_train)

    # Get the best parameters
    best_params = grid_search.best_params_
    print(f"Best parameters: {best_params}")

    # Train Random Forest with best parameters
    rf_best = RandomForestRegressor(
        n_estimators=best_params['n_estimators'],
        max_depth=best_params['max_depth'],
        min_samples_split=best_params['min_samples_split'],
```

```
        random_state=42
    )
    rf_best.fit(X_train, y_train)

    # Make predictions
    y_train_pred = rf_best.predict(X_train)
    y_test_pred = rf_best.predict(X_test)

    # Evaluate performance
    train_rmse = np.sqrt(mean_squared_error(y_train, y_train_pred))
    test_rmse = np.sqrt(mean_squared_error(y_test, y_test_pred))
    train_r2 = r2_score(y_train, y_train_pred)
    test_r2 = r2_score(y_test, y_test_pred)

    print(f"Training RMSE: {train_rmse:.4f}, Test RMSE: {test_rmse:.4f}")
    print(f"Training R-squared: {train_r2:.4f}, Test R-squared: {test_r2:.4f}")

    # Cross-validation
    cv_scores = cross_val_score(rf_best, X_train, y_train, cv=5, scoring='neg_mean_squared_
    cv_rmse = np.sqrt(-cv_scores)
    print(f"Cross-validated RMSE (5-fold): {cv_rmse.mean():.4f} ± {cv_rmse.std():.4f}")

    # Return feature importances
    return rf_best.feature_importances_, X.columns

# Apply IQR segmentation on 'duration', 'salary', and 'attendees'
segments = apply_iqr_segment(df_no_outliers, ['duration', 'attendees', 'salary'])

# One-hot encode the 'salary_tier' column after IQR segmentation
for key, segment in segments.items():
    for sub_key in segment:
        segments[key][sub_key] = pd.get_dummies(segments[key][sub_key], columns=['salary_ti

# Define features and target for each segment
param_grid = {
    'n_estimators': [50, 100, 150],  # Reduce the number of trees to evaluate
    'max_depth': [10, 20, 30],       # Reduce the range of tree depth
    'min_samples_split': [5, 10]     # Simplify min_samples_split options
}

# Plot setup
fig, axes = plt.subplots(nrows=len(segments), ncols=3, figsize=(15, 12))
plt.tight_layout(pad=4.0)

# Train and evaluate model for each segment
for row_idx, (feature, iqr_segment) in enumerate(segments.items()):
    for col_idx, (sub_segment_name, sub_segment_data) in enumerate(iqr_segment.items()):
        # Drop irrelevant features like meeting_id and total_cost
        X = sub_segment_data.drop(columns=['total_cost_per_meeting', 'meeting_id', 'total_c
        y = sub_segment_data['total_cost_per_meeting']  # Dependent variable
        print(f"\n### Results for {feature} - {sub_segment_name} ###")
        feature_importances, features = train_and_evaluate(X, y, param_grid)

        # Visualize feature importances in subplots
        sorted_idx = np.argsort(feature_importances)[::-1]
        axes[row_idx, col_idx].barh(range(len(sorted_idx)), feature_importances[sorted_idx]
        axes[row_idx, col_idx].set_yticks(range(len(sorted_idx)))
        axes[row_idx, col_idx].set_yticklabels([features[i] for i in sorted_idx])
        axes[row_idx, col_idx].set_xlabel('Importance')
        axes[row_idx, col_idx].set_title(f'{feature} - {sub_segment_name}')
        axes[row_idx, col_idx].invert_yaxis()
```

```
plt.show()
```

```
### Results for duration - low ###
c:\Users\kipjo\AppData\Local\Programs\Python\Python312\Lib\site-packages\numpy\ma\core.py:2846: RuntimeWarning: invalid value encountered in cast
  _data = np.array(data, dtype=dtype, copy=copy,
Best parameters: {'max_depth': 10, 'min_samples_split': 10, 'n_estimators': 150}
Training RMSE: 237440.3082, Test RMSE: 267434.3609
Training R-squared: 0.4260, Test R-squared: 0.1259
Cross-validated RMSE (5-fold): 288944.6013 ± 34883.1297

### Results for duration - mid ###
Best parameters: {'max_depth': 10, 'min_samples_split': 10, 'n_estimators': 50}
Training RMSE: 319240.0117, Test RMSE: 331700.3187
Training R-squared: 0.4992, Test R-squared: 0.3948
Cross-validated RMSE (5-fold): 406467.2815 ± 38063.8572

### Results for duration - high ###
Best parameters: {'max_depth': 10, 'min_samples_split': 10, 'n_estimators': 150}
Training RMSE: 289300.6571, Test RMSE: 614974.2158
Training R-squared: 0.5714, Test R-squared: -0.0555
Cross-validated RMSE (5-fold): 405779.3235 ± 95495.5605

### Results for attendees - low ###
Best parameters: {'max_depth': 10, 'min_samples_split': 10, 'n_estimators': 150}
Training RMSE: 282525.4911, Test RMSE: 370940.1081
Training R-squared: 0.3463, Test R-squared: 0.0997
Cross-validated RMSE (5-fold): 330328.3629 ± 36085.3320

### Results for attendees - mid ###
Best parameters: {'max_depth': 20, 'min_samples_split': 10, 'n_estimators': 150}
Training RMSE: 275320.7074, Test RMSE: 275128.1691
Training R-squared: 0.5841, Test R-squared: 0.1339
Cross-validated RMSE (5-fold): 349234.6257 ± 62609.9729

### Results for attendees - high ###
Best parameters: {'max_depth': 20, 'min_samples_split': 5, 'n_estimators': 100}
Training RMSE: 145977.9467, Test RMSE: 77535.2095
Training R-squared: 0.8746, Test R-squared: 0.9442
Cross-validated RMSE (5-fold): 256729.9585 ± 77902.7914

### Results for salary - low ###
Best parameters: {'max_depth': 10, 'min_samples_split': 5, 'n_estimators': 100}
Training RMSE: 250136.9110, Test RMSE: 286780.3076
Training R-squared: 0.2565, Test R-squared: 0.1383
Cross-validated RMSE (5-fold): 269357.9209 ± 53078.5001

### Results for salary - mid ###
Best parameters: {'max_depth': 10, 'min_samples_split': 10, 'n_estimators': 100}
Training RMSE: 271621.0273, Test RMSE: 308760.8703
Training R-squared: 0.5136, Test R-squared: 0.1878
Cross-validated RMSE (5-fold): 363148.2718 ± 30056.5116

### Results for salary - high ###
Best parameters: {'max_depth': 10, 'min_samples_split': 10, 'n_estimators': 150}
Training RMSE: 268240.5810, Test RMSE: 334722.7722
Training R-squared: 0.5901, Test R-squared: 0.5143
Cross-validated RMSE (5-fold): 354896.5258 ± 39915.9234
```


duration - low
salary
attendees
duration
salary_tier_Senior
salary_tier_Mid-level
salary_tier_Executive
0.0 0.1 0.2 0.3 0.4 0.5 0.6
Importance
duration - mid
salary
attendees
duration
salary_tier_Mid-level
salary_tier_Senior
salary_tier_Executive
0.0 0.1 0.2 0.3 0.4 0.5 0.6
Importance
duration - high
salary
attendees
salary_tier_Mid-level
duration
salary_tier_Senior
salary_tier_Executive
0.0 0.1 0.2 0.3 0.4 0.5
Importance
attendees - low
salary
duration
attendees
salary_tier_Mid-level
salary_tier_Senior
salary_tier_Executive
0.0 0.1 0.2 0.3 0.4 0.5
Importance
attendees - mid
salary
duration
attendees
salary_tier_Mid-level
salary_tier_Senior
salary_tier_Executive
0.0 0.1 0.2 0.3 0.4
Importance
attendees - high
salary
duration
attendees
salary_tier_Senior
salary_tier_Mid-level
salary_tier_Executive
0.0 0.1 0.2 0.3 0.4 0.5 0.6
Importance
salary - low
duration
salary
attendees
salary_tier_Executive
salary_tier_Senior
salary_tier_Mid-level
0.0 0.1 0.2 0.3 0.4
Importance
salary - mid
duration
attendees
salary
salary_tier_Mid-level
salary_tier_Senior
salary_tier_Executive
0.00 0.05 0.10 0.15 0.20 0.25 0.30 0.35 0.40
Importance
salary - high
attendees
duration
salary
salary_tier_Executive
salary_tier_Senior
salary_tier_Mid-level
0.00 0.05 0.10 0.15 0.20 0.25 0.30 0.35 0.40
Importance