# eco482project (5) (2) (1) (2)

April 6, 2025

[1]:

```
import pandas as pd
import numpy as np
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.pipeline import Pipeline
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import RandomForestClassifier
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
import seaborn as sns
import statsmodels.api as sm
#Bias + how i rates j + own i rating + demographic of i + demographic of j
#Define y clearly in presentation
file = 'speed_dating_with_participant_ids.csv'
df = pd.read_csv(file)
df
```

```
/tmp/ipykernel_100/3825651112.py:20: DtypeWarning: Columns
(4,11,12,40,41,42,74,75,76,77,78,79,80,81,82,83,84,85,86,87,88,89,90,110) have
mixed types. Specify dtype option on import or set low_memory=False.
  df = pd.read_csv(file)
```

[1]:

```
        id  has_null  wave  gender age  age_o  d_age d_d_age  \
0     8202         1    21    male  23     22      1   [0-1]
1     8193         1    21    male  23     26      3   [2-3]
2     8194         1    21    male  23     22      1   [0-1]
3     8195         1    21    male  23     27      4   [4-6]
4     8196         1    21    male  23     25      2   [2-3]
…     …          …    …      … ..   …      …      …
8373  7494         1    21  female   ?     30     30  [7-37]
8374  7496         1    21  female   ?     30     30  [7-37]
8375  7497         1    21  female   ?     27     27  [7-37]
```

```
8376  7483         1    21  female   ?    22      22  [7-37]
8377  7495         1    21  female   ?    28      28  [7-37]

                                       race  \
0     Asian/Pacific Islander/Asian-American
1     Asian/Pacific Islander/Asian-American
2     Asian/Pacific Islander/Asian-American
3     Asian/Pacific Islander/Asian-American
4     Asian/Pacific Islander/Asian-American
…                                     …
8373            European/Caucasian-American
8374            European/Caucasian-American
8375            European/Caucasian-American
8376            European/Caucasian-American
8377            European/Caucasian-American

                                     race_o  …  d_like d_guess_prob_liked  \
0     Asian/Pacific Islander/Asian-American  …   [6-8]              [5-6]
1               European/Caucasian-American  …   [6-8]              [5-6]
2               European/Caucasian-American  …   [6-8]             [7-10]
3     Asian/Pacific Islander/Asian-American  …   [6-8]             [7-10]
4     Asian/Pacific Islander/Asian-American  …   [6-8]              [5-6]
…                                     …  …     …                …
8373            European/Caucasian-American  …   [0-5]              [0-4]
8374            European/Caucasian-American  …   [6-8]             [7-10]
8375                 Black/African American  …   [0-5]              [5-6]
8376  Asian/Pacific Islander/Asian-American  …   [6-8]             [7-10]
8377            European/Caucasian-American  …   [0-5]              [5-6]

     met decision decision_o match Unnamed: 124 Unnamed: 125 Unnamed: 126  \
0      0        1          1     1          NaN          NaN          NaN
1      0        1          0     0          NaN          NaN          NaN
2      0        1          1     1          NaN          NaN          NaN
3      1        1          1     1          NaN          NaN          NaN
4      0        0          1     0          NaN          NaN          NaN
…    ..       …          …   …            …            …            …
8373   ?        0          0     0          NaN          NaN          NaN
8374   0        0          1     0          NaN          NaN          NaN
8375   0        0          0     0          NaN          NaN          NaN
8376   0        0          0     0          NaN          NaN          NaN
8377   0        0          1     0          NaN          NaN          NaN

     participant_id
0                 0
1                 0
2                 0
3                 0
```

```
4                  0
…               …
8373             526
8374             526
8375             526
8376             526
8377             526

[8378 rows x 128 columns]
```

[2]:
```python
columns_to_check = [
    "age", "race", "field", "gender", "attractive_o", "sinsere_o",␣
 ↪"intelligence_o",
    "funny_o", "ambitous_o", "attractive", "sincere", "intelligence", "funny",
    "ambition", "expected_num_matches", "decision_o", "guess_prob_liked",␣
 ↪"d_age",
]

# Replace "?" with NaN
df = df.replace("?", pd.NA)

# Drop rows with NaN in any of the specified columns
df = df.dropna(subset=columns_to_check)

df = df[df["guess_prob_liked"] != 5]
df
```

[2]:
```
        id  has_null  wave  gender age age_o  d_age d_d_age  \
0     8202         1    21    male  23    22      1   [0-1]
3     8195         1    21    male  23    27      4   [4-6]
4     8196         1    21    male  23    25      2   [2-3]
5     8197         1    21    male  23    24      1   [0-1]
6     8198         1    21    male  23    26      3   [2-3]
…     …          …   …      … ..  …   …     …
8278  6827         1    18  female  55    33     22  [7-37]
8279  6826         1    18  female  55    23     32  [7-37]
8280  6825         1    18  female  55    33     22  [7-37]
8281  6823         1    18  female  55    33     22  [7-37]
8282  6824         1    18  female  55    27     28  [7-37]

                                         race  \
0     Asian/Pacific Islander/Asian-American
3     Asian/Pacific Islander/Asian-American
4     Asian/Pacific Islander/Asian-American
5     Asian/Pacific Islander/Asian-American
6     Asian/Pacific Islander/Asian-American
…                                         …
```

```
8278  Asian/Pacific Islander/Asian-American
8279  Asian/Pacific Islander/Asian-American
8280  Asian/Pacific Islander/Asian-American
8281  Asian/Pacific Islander/Asian-American
8282  Asian/Pacific Islander/Asian-American

                                     race_o  …  d_like d_guess_prob_liked  \
0     Asian/Pacific Islander/Asian-American  …   [6-8]              [5-6]
3     Asian/Pacific Islander/Asian-American  …   [6-8]             [7-10]
4     Asian/Pacific Islander/Asian-American  …   [6-8]              [5-6]
5             European/Caucasian-American  …   [6-8]              [0-4]
6                 Latino/Hispanic American  …   [0-5]              [0-4]
…                                         …  …     …                  …
8278  Asian/Pacific Islander/Asian-American  …   [6-8]              [0-4]
8279  Asian/Pacific Islander/Asian-American  …   [6-8]              [0-4]
8280  Asian/Pacific Islander/Asian-American  …   [0-5]              [0-4]
8281          European/Caucasian-American  …   [6-8]              [0-4]
8282                                  Other  …   [0-5]              [0-4]

     met decision decision_o match Unnamed: 124 Unnamed: 125 Unnamed: 126  \
0      0        1          1     1          NaN          NaN          NaN
3      1        1          1     1          NaN          NaN          NaN
4      0        0          1     0          NaN          NaN          NaN
5      0        0          0     0          NaN          NaN          NaN
6      0        0          0     0          NaN          NaN          NaN
…    ..       …          …     …            …            …            …
8278   0        1          0     0          NaN          NaN          NaN
8279   0        1          0     0          NaN          NaN          NaN
8280   0        0          1     0          NaN          NaN          NaN
8281   0        1          0     0          NaN          NaN          NaN
8282   0        0          0     0          NaN          NaN          NaN

     participant_id
0                 0
3                 0
4                 0
5                 0
6                 0
…               …
8278            523
8279            523
8280            523
8281            523
8282            523

[6316 rows x 128 columns]
```

[3]:
```
df["guess_prob_liked"] = pd.to_numeric(df["guess_prob_liked"], errors="coerce")
df["decision_o"] = pd.to_numeric(df["decision_o"], errors="coerce")

# Apply the logic using np.where
import numpy as np

df["prediction_like"] = np.where(
    ((df["guess_prob_liked"] > 5) & (df["decision_o"] == 0)) |
    ((df["guess_prob_liked"] < 5) & (df["decision_o"] == 1)),
    0,  # Miscalibrated
    1   # Calibrated
)
```

[4]:
```
df = df.loc[:, ~df.columns.str.contains('^Unnamed')]
```

[5]:
```
characteristics = {
    "attractive": "attractive_o",
    "sincere": "sinsere_o",
    "intelligence": "intelligence_o",
    "funny": "funny_o",
    "ambition": "ambitous_o"
}

# Make sure all relevant columns are numeric
for col in list(characteristics.keys()) + list(characteristics.values()):
    df[col] = pd.to_numeric(df[col], errors='coerce')

# Dictionary to store the average bias per person per trait
participant_biases = {}

# Loop through each characteristic
for self_col, other_col in characteristics.items():
    # Calculate self - other rating (row-wise)
    df[f'diff_{self_col}'] = df[self_col] - df[other_col]

    # Compute average difference per participant
    avg_bias = df.groupby("participant_id")[f'diff_{self_col}'].mean()

    # Save the average bias
    df[f'bias_{self_col}'] = df["participant_id"].map(avg_bias)

# Drop intermediate columns if desired
df.drop(columns=[f'diff_{col}' for col in characteristics.keys()], inplace=True)

# Show the updated DataFrame
df.head()
```

```
/tmp/ipykernel_100/1577411713.py:11: 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_guide/indexing.html#returning-a-view-versus-a-copy
  df[col] = pd.to_numeric(df[col], errors='coerce')
/tmp/ipykernel_100/1577411713.py:19: 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_guide/indexing.html#returning-a-view-versus-a-copy
  df[f'diff_{self_col}'] = df[self_col] - df[other_col]
/tmp/ipykernel_100/1577411713.py:25: 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_guide/indexing.html#returning-a-view-versus-a-copy
  df[f'bias_{self_col}'] = df["participant_id"].map(avg_bias)
```

[5]:

```
     id  has_null  wave gender age age_o  d_age d_d_age  \
0  8202         1    21   male  23    22      1   [0-1]
3  8195         1    21   male  23    27      4   [4-6]
4  8196         1    21   male  23    25      2   [2-3]
5  8197         1    21   male  23    24      1   [0-1]
6  8198         1    21   male  23    26      3   [2-3]

                                      race  \
0  Asian/Pacific Islander/Asian-American
3  Asian/Pacific Islander/Asian-American
4  Asian/Pacific Islander/Asian-American
5  Asian/Pacific Islander/Asian-American
6  Asian/Pacific Islander/Asian-American

                                    race_o  …  decision decision_o match  \
0  Asian/Pacific Islander/Asian-American  …         1          1     1
3  Asian/Pacific Islander/Asian-American  …         1          1     1
4  Asian/Pacific Islander/Asian-American  …         0          1     0
5           European/Caucasian-American  …         0          0     0
6              Latino/Hispanic American  …         0          0     0

  participant_id prediction_like bias_attractive bias_sincere  \
0              0               1             2.0          4.5
3              0               1             2.0          4.5
4              0               1             2.0          4.5
```

```
5              0               1            2.0          4.5
6              0               1            2.0          4.5

  bias_intelligence bias_funny bias_ambition
0               4.0       3.75          2.25
3               4.0       3.75          2.25
4               4.0       3.75          2.25
5               4.0       3.75          2.25
6               4.0       3.75          2.25

[5 rows x 131 columns]
```

[6]:
```python
df["expected_num_matches"] = pd.to_numeric(df["expected_num_matches"],␣
↪errors="coerce")
df["match"] = pd.to_numeric(df["match"], errors="coerce")

# Compute total actual matches per participant
total_matches = df.groupby("participant_id")["match"].sum()

# Compute expected matches per participant (mean across their rows)
expected_matches = df.groupby("participant_id")["expected_num_matches"].mean()

# Compute matching bias: actual - expected
matching_bias = total_matches - expected_matches

# Add matching bias to each row of the DataFrame
df["matching_bias"] = df["participant_id"].map(matching_bias)
df.head()
```

[6]:
```
     id  has_null  wave gender age age_o  d_age d_d_age  \
0  8202         1    21   male  23    22      1   [0-1]
3  8195         1    21   male  23    27      4   [4-6]
4  8196         1    21   male  23    25      2   [2-3]
5  8197         1    21   male  23    24      1   [0-1]
6  8198         1    21   male  23    26      3   [2-3]

                                  race  \
0  Asian/Pacific Islander/Asian-American
3  Asian/Pacific Islander/Asian-American
4  Asian/Pacific Islander/Asian-American
5  Asian/Pacific Islander/Asian-American
6  Asian/Pacific Islander/Asian-American

                                race_o  …  decision_o match  \
0  Asian/Pacific Islander/Asian-American  …           1     1
3  Asian/Pacific Islander/Asian-American  …           1     1
4  Asian/Pacific Islander/Asian-American  …           1     0
```

```
5            European/Caucasian-American  …           0     0
6               Latino/Hispanic American  …           0     0

  participant_id prediction_like bias_attractive bias_sincere  \
0              0               1             2.0          4.5
3              0               1             2.0          4.5
4              0               1             2.0          4.5
5              0               1             2.0          4.5
6              0               1             2.0          4.5

  bias_intelligence bias_funny bias_ambition matching_bias
0               4.0       3.75          2.25           2.0
3               4.0       3.75          2.25           2.0
4               4.0       3.75          2.25           2.0
5               4.0       3.75          2.25           2.0
6               4.0       3.75          2.25           2.0

[5 rows x 132 columns]
```

[7]:

```
df["field"] = df["field"].astype(str).str.strip().str.lower()
df["field_o"] = np.nan  # initialize the new column

# Ensure age columns are numeric
df["age"] = pd.to_numeric(df["age"], errors="coerce")
df["age_o"] = pd.to_numeric(df["age_o"], errors="coerce")

# Define preference  importance column mappings
preference_match_columns = {
    "pref_o_attractive": "attractive_important",
    "pref_o_sincere": "sincere_important",
    "pref_o_intelligence": "intellicence_important",
    "pref_o_funny": "funny_important",
    "pref_o_ambitious": "ambtition_important",
    "pref_o_shared_interests": "shared_interests_important"
}

# Ensure all involved columns are numeric
for pref_col, imp_col in preference_match_columns.items():
    df[pref_col] = pd.to_numeric(df[pref_col], errors="coerce")
    df[imp_col] = pd.to_numeric(df[imp_col], errors="coerce")

df["same_field"] = 0

for wave in df["wave"].unique():
    wave_df = df[df["wave"] == wave]
    males = wave_df[wave_df["gender"] == "male"]
    females = wave_df[wave_df["gender"] == "female"]
```

```python
    for idx, male in males.iterrows():
        matches = females[
            (females["age"] == male["age_o"])
        ]
        for pref_col, imp_col in preference_match_columns.items():
            matches = matches[matches[imp_col] == male[pref_col]]

        if not matches.empty:
            # Choose the first matching partner
            matched_female = matches.iloc[0]
            if matched_female["field"] == male["field"]:
                df.loc[idx, "same_field"] = 1
            df.loc[idx, "field_o"] = matched_female["field"]

    for idx, female in females.iterrows():
        matches = males[
            (males["age"] == female["age_o"])
        ]
        for pref_col, imp_col in preference_match_columns.items():
            matches = matches[matches[imp_col] == female[pref_col]]

        if not matches.empty:
            matched_male = matches.iloc[0]
            if matched_male["field"] == female["field"]:
                df.loc[idx, "same_field"] = 1
            df.loc[idx, "field_o"] = matched_male["field"]

print(df["same_field"].sum())
df.head()
```

```
/tmp/ipykernel_100/3114844432.py:42: FutureWarning: Setting an item of
incompatible dtype is deprecated and will raise in a future error of pandas.
Value 'mechanical engineering' has dtype incompatible with float64, please
explicitly cast to a compatible dtype first.
  df.loc[idx, "field_o"] = matched_female["field"]
```

```
152
```

[7]:

```
     id  has_null  wave gender  age  age_o  d_age d_d_age  \
0  8202         1    21   male   23   22.0      1   [0-1]
3  8195         1    21   male   23   27.0      4   [4-6]
4  8196         1    21   male   23   25.0      2   [2-3]
5  8197         1    21   male   23   24.0      1   [0-1]
6  8198         1    21   male   23   26.0      3   [2-3]

                                  race  \
0  Asian/Pacific Islander/Asian-American
```

```
3  Asian/Pacific Islander/Asian-American
4  Asian/Pacific Islander/Asian-American
5  Asian/Pacific Islander/Asian-American
6  Asian/Pacific Islander/Asian-American

                                 race_o  …  participant_id prediction_like  \
0  Asian/Pacific Islander/Asian-American  …               0               1
3  Asian/Pacific Islander/Asian-American  …               0               1
4  Asian/Pacific Islander/Asian-American  …               0               1
5           European/Caucasian-American  …               0               1
6              Latino/Hispanic American  …               0               1

  bias_attractive bias_sincere bias_intelligence bias_funny  bias_ambition  \
0             2.0          4.5               4.0       3.75           2.25
3             2.0          4.5               4.0       3.75           2.25
4             2.0          4.5               4.0       3.75           2.25
5             2.0          4.5               4.0       3.75           2.25
6             2.0          4.5               4.0       3.75           2.25

   matching_bias                field_o  same_field
0            2.0  mechanical engineering           0
3            2.0                     NaN           0
4            2.0                medicine           0
5            2.0           public health           0
6            2.0             social work           0

[5 rows x 134 columns]
```

[8]:
```
df["gender_o"] = df["gender"].apply(lambda g: "female" if g == "male" else␣
 ↪"male")
```

[9]:
```
# List of predictors: d_age, samerace, interests_correlate, same_field,␣
 ↪bias_attractive, bias_sincere, bias_intelligence, bias_funny, bias_ambition,␣
 ↪matching_bias, gender, race, age, field
```

[10]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import ttest_ind
import numpy as np

# ---------------------------------------------
# Step 0: Prepare Encodings for Gender, Race, Field
# ---------------------------------------------
df['gender_num'] = df['gender'].str.lower().map({'male': 1, 'female': 0})
df['gender_num_o'] = df['gender_o'].str.lower().map({'male': 1, 'female': 0})
df['race_num'] = df['race'].astype('category').cat.codes
```

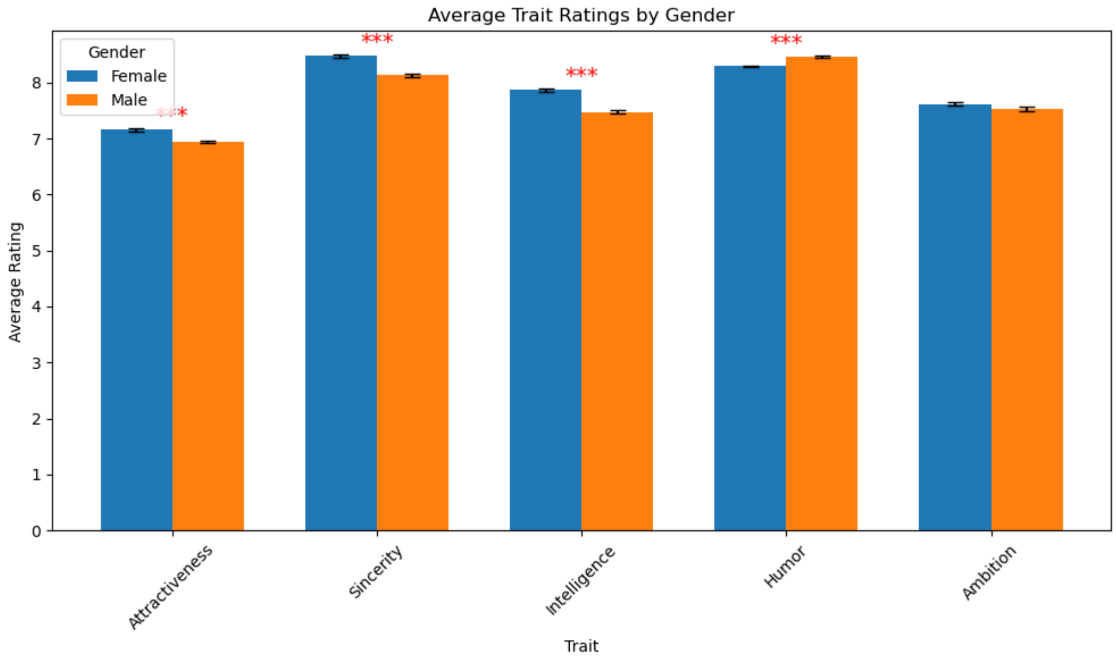
```
df['race_o_num'] = df['race_o'].astype('category').cat.codes

all_fields = pd.concat([df["field"], df["field_o"]], ignore_index=True).
 ↪astype('category')
consistent_categories = all_fields.cat.categories
df["field"] = pd.Categorical(df["field"], categories=consistent_categories)
df["field_o"] = pd.Categorical(df["field_o"], categories=consistent_categories)
df["field_num"] = df["field"].cat.codes
df["field_num_o"] = df["field_o"].cat.codes

# ---------------------------------------------
# Step 1: Convert other key columns to numeric
# ---------------------------------------------
numeric_cols = [
    "decision", "decision_o", "guess_prob_liked", "like",
    "expected_num_matches", "matching_bias",
    "bias_attractive", "bias_sincere", "bias_intelligence",
    "bias_funny", "bias_ambition",
    "attractive", "sincere", "intelligence", "funny", "ambition"
]
df[numeric_cols] = df[numeric_cols].apply(pd.to_numeric, errors="coerce")

# Asterisk function
def p_to_stars(p):
    if p < 0.001:
        return '***'
    elif p < 0.01:
        return '**'
    elif p < 0.05:
        return '*'
    else:
        return ''

# Function to compute mean and SE
def mean_and_se(df, group_col, target_cols):
    means = df.groupby(group_col)[target_cols].mean()
    stds = df.groupby(group_col)[target_cols].std()
    counts = df.groupby(group_col)[target_cols].count()
    ses = stds / np.sqrt(counts)
    return means.T, ses.T

# ---------------------------------------------
# Step 2: Correlation Analysis
# ---------------------------------------------
print("Correlation between guess_prob_liked and decision_o:",␣
 ↪df["guess_prob_liked"].corr(df["decision_o"]))
print("Correlation between like and decision:", df["like"].corr(df["decision"]))
```

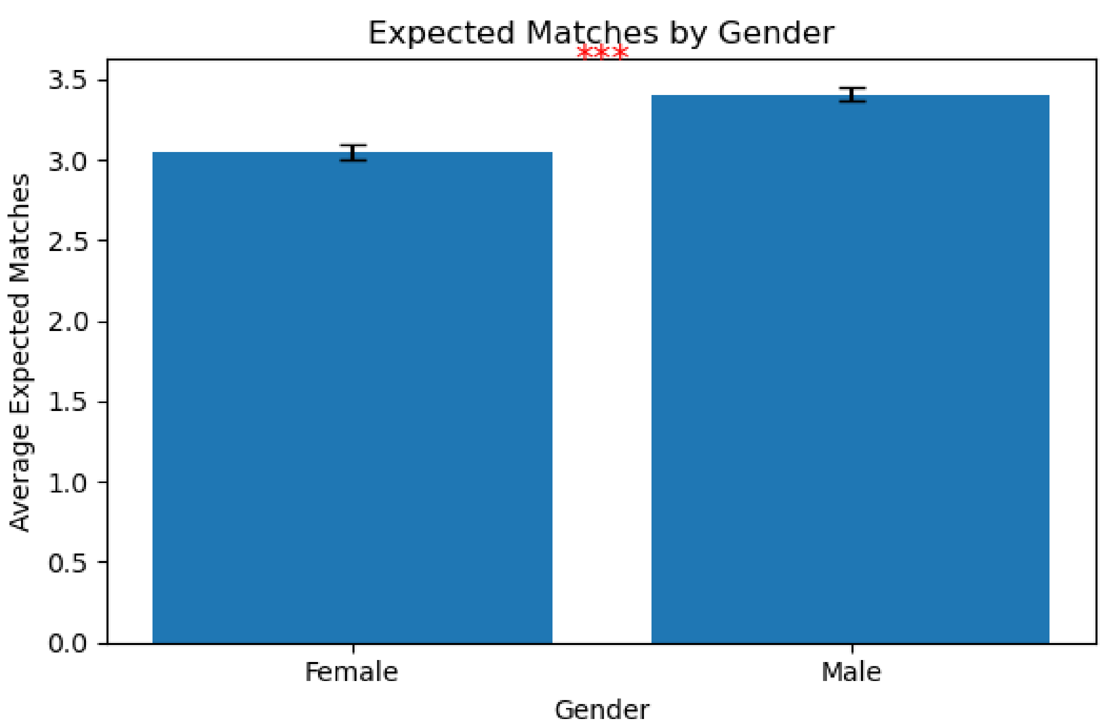
```python
# ---------------------------------------------
# Step 3: Trait Ratings by Gender with Asterisks + SE
# ---------------------------------------------
traits = ["attractive", "sincere", "intelligence", "funny", "ambition"]
trait_names = ["Attractiveness", "Sincerity", "Intelligence", "Humor",␣
↪"Ambition"]

trait_means, trait_ses = mean_and_se(df, "gender_num", traits)
trait_means.index = trait_names
trait_ses.index = trait_names

# T-tests
pvals = []
for trait in traits:
    f = df[df["gender_num"] == 0][trait].dropna()
    m = df[df["gender_num"] == 1][trait].dropna()
    _, p = ttest_ind(f, m, equal_var=False)
    pvals.append(p_to_stars(p))

fig, ax = plt.subplots(figsize=(10, 6))
x = np.arange(len(trait_names))
bar_width = 0.35

# Bar plots with error bars
ax.bar(x - bar_width/2, trait_means[0].values, yerr=trait_ses[0].values,␣
↪width=bar_width, label="Female", capsize=5)
ax.bar(x + bar_width/2, trait_means[1].values, yerr=trait_ses[1].values,␣
↪width=bar_width, label="Male", capsize=5)

for i, stars in enumerate(pvals):
    if stars:
        ymax = max(trait_means.iloc[i][0] + trait_ses.iloc[i][0], trait_means.
↪iloc[i][1] + trait_ses.iloc[i][1])
        ax.text(i, ymax + 0.1, stars, ha='center', color='red', fontsize=14)

ax.set_xticks(x)
ax.set_xticklabels(trait_names, rotation=45)
ax.set_title("Average Trait Ratings by Gender")
ax.set_ylabel("Average Rating")
ax.set_xlabel("Trait")
ax.legend(title="Gender")
plt.tight_layout()
plt.grid(False)
plt.show()

# ---------------------------------------------
```

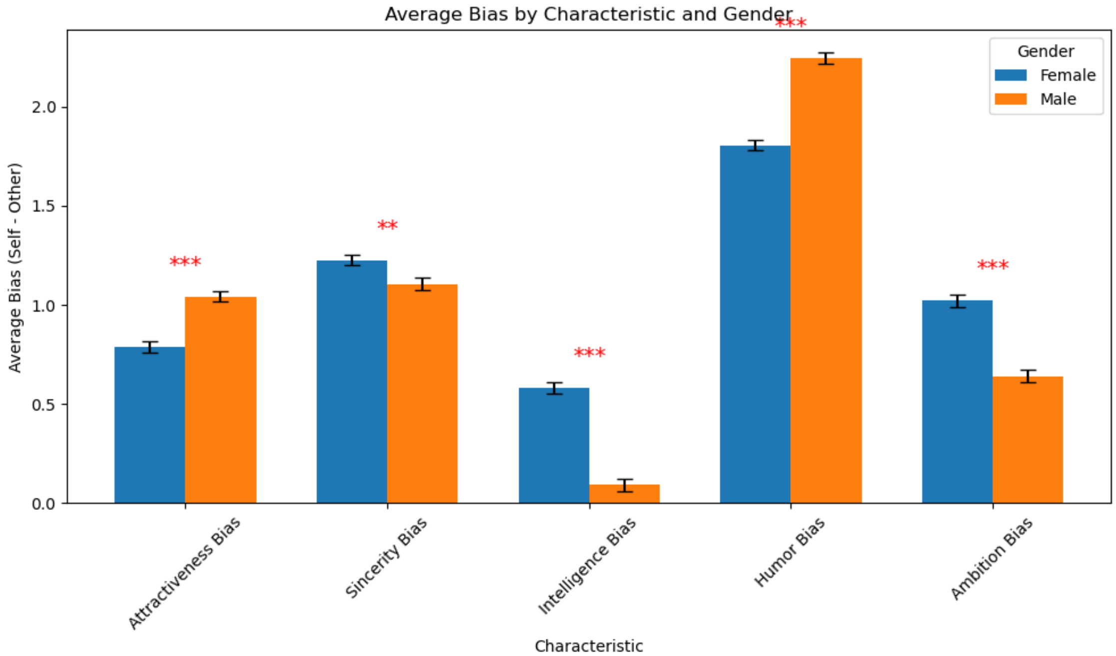
```
# Step 4: Expected Matches by Gender
# ---------------------------------------------
group0 = df[df["gender_num"] == 0]["expected_num_matches"].dropna()
group1 = df[df["gender_num"] == 1]["expected_num_matches"].dropna()
_, p = ttest_ind(group0, group1, equal_var=False)
asterisks = p_to_stars(p)

means = [group0.mean(), group1.mean()]
ses = [group0.std() / np.sqrt(len(group0)), group1.std() / np.sqrt(len(group1))]

plt.figure(figsize=(6, 4))
ax = plt.gca()
ax.bar(["Female", "Male"], means, yerr=ses, capsize=5)
plt.title("Expected Matches by Gender")
plt.ylabel("Average Expected Matches")
plt.xlabel("Gender")
plt.tight_layout()
plt.grid(False)

if asterisks:
    ymax = max(means[i] + ses[i] for i in range(2))
    ax.text(0.5, ymax + 0.1, asterisks, ha='center', fontsize=14, color='red')

plt.show()

# ---------------------------------------------
# Step 5: Self-Other Bias by Gender with SE
# ---------------------------------------------
bias_cols = ["bias_attractive", "bias_sincere", "bias_intelligence",␣
↪"bias_funny", "bias_ambition"]
bias_names = ["Attractiveness Bias", "Sincerity Bias", "Intelligence Bias",␣
↪"Humor Bias", "Ambition Bias"]

bias_means, bias_ses = mean_and_se(df, "gender_num", bias_cols)
bias_means.index = bias_names
bias_ses.index = bias_names

pvals_bias = []
for col in bias_cols:
    f = df[df["gender_num"] == 0][col].dropna()
    m = df[df["gender_num"] == 1][col].dropna()
    _, p = ttest_ind(f, m, equal_var=False)
    pvals_bias.append(p_to_stars(p))

fig, ax = plt.subplots(figsize=(10, 6))
x = np.arange(len(bias_names))
```

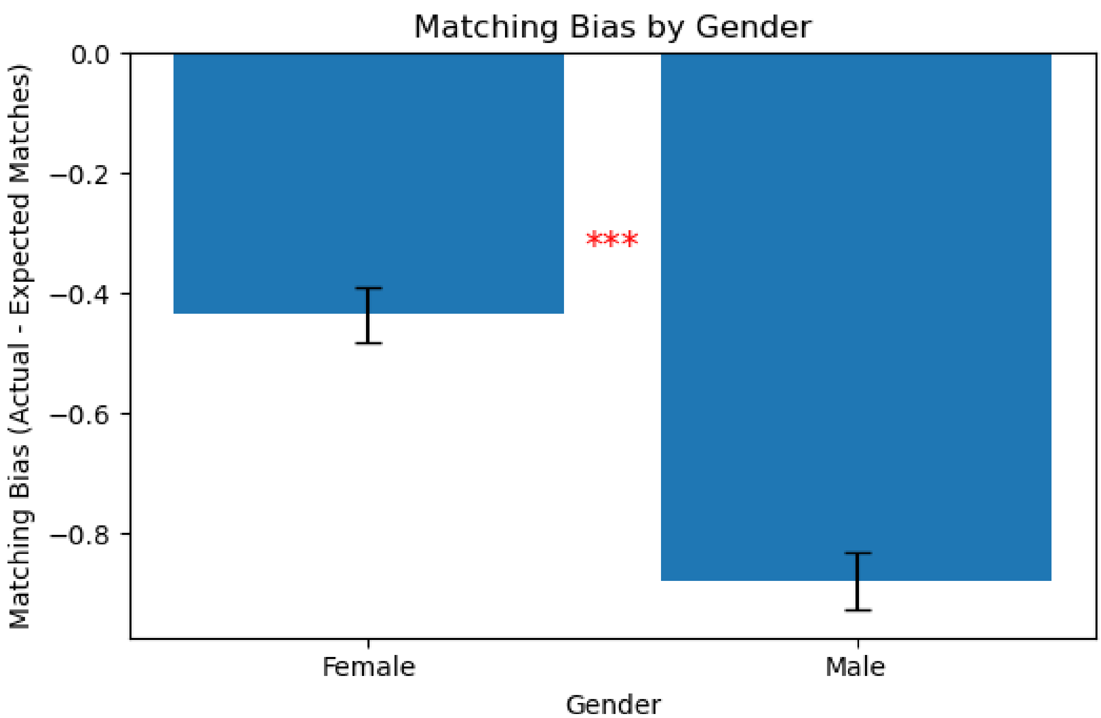
```python
ax.bar(x - bar_width/2, bias_means[0].values, yerr=bias_ses[0].values, width=bar_width, label="Female", capsize=5)
ax.bar(x + bar_width/2, bias_means[1].values, yerr=bias_ses[1].values, width=bar_width, label="Male", capsize=5)

for i, stars in enumerate(pvals_bias):
    if stars:
        ymax = max(bias_means.iloc[i][0] + bias_ses.iloc[i][0], bias_means.iloc[i][1] + bias_ses.iloc[i][1])
        ax.text(i, ymax + 0.1, stars, ha='center', color='red', fontsize=14)

ax.set_xticks(x)
ax.set_xticklabels(bias_names, rotation=45)
ax.set_title("Average Bias by Characteristic and Gender")
ax.set_ylabel("Average Bias (Self - Other)")
ax.set_xlabel("Characteristic")
ax.legend(title="Gender")
plt.tight_layout()
plt.grid(False)
plt.show()

# ---------------------------------------------
# Step 6: Matching Bias by Gender with SE
# ---------------------------------------------
g0 = df[df["gender_num"] == 0]["matching_bias"].dropna()
g1 = df[df["gender_num"] == 1]["matching_bias"].dropna()
_, p = ttest_ind(g0, g1, equal_var=False)
asterisks = p_to_stars(p)

means = [g0.mean(), g1.mean()]
ses = [g0.std() / np.sqrt(len(g0)), g1.std() / np.sqrt(len(g1))]

plt.figure(figsize=(6, 4))
ax = plt.gca()
ax.bar(["Female", "Male"], means, yerr=ses, capsize=5)
plt.title("Matching Bias by Gender")
plt.xlabel("Gender")
plt.ylabel("Matching Bias (Actual - Expected Matches)")
plt.tight_layout()
plt.grid(False)

if asterisks:
    ymax = max(means[i] + ses[i] for i in range(2))
    ax.text(0.5, ymax + 0.05, asterisks, ha='center', color='red', fontsize=14)

plt.show()
```

```
Correlation between guess_prob_liked and decision_o: 0.1409880632116935
Correlation between like and decision: 0.5150175200729944
```



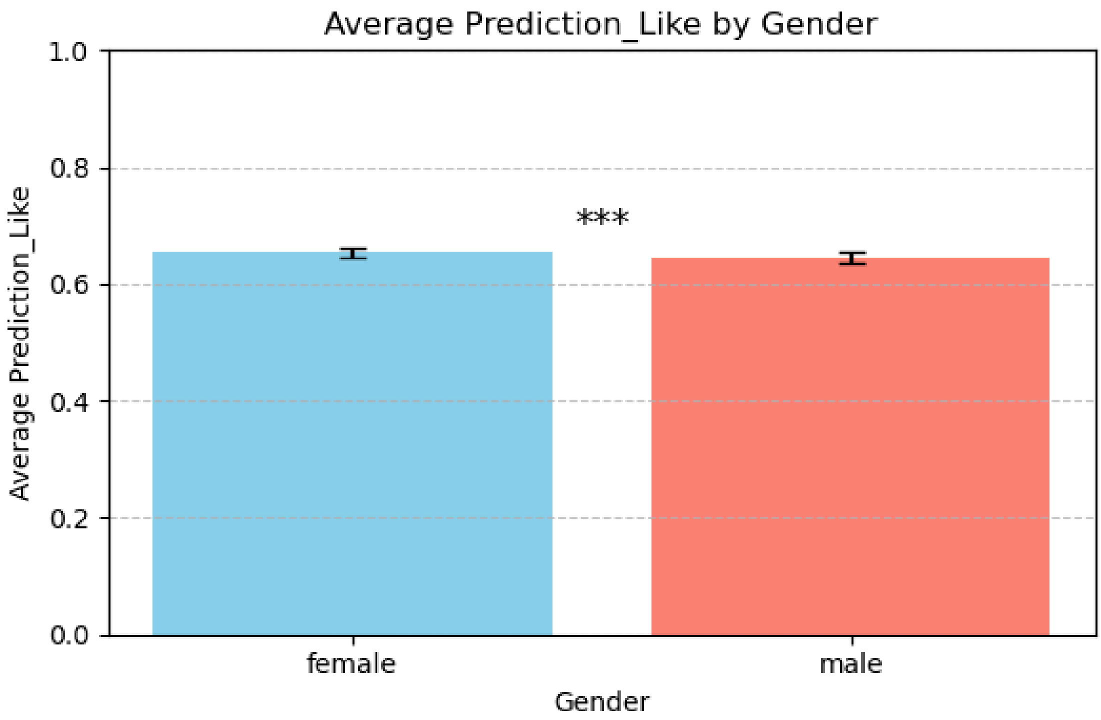




Average Bias by Characteristic and Gender
Gender
Female
Male
Average Bias (Self - Other)
0.0
0.5
1.0
1.5
2.0
***
**
***
***
***
Attractiveness Bias
Sincerity Bias
Intelligence Bias
Humor Bias
Ambition Bias
Characteristic


Matching Bias by Gender
Matching Bias (Actual - Expected Matches)
0.0
−0.2
−0.4
−0.6
−0.8
***
Female
Male
Gender

[11]:
```python
import matplotlib.pyplot as plt
import scipy.stats as stats
import numpy as np
import pandas as pd

# Clean gender column
df['gender'] = df['gender'].str.lower().str.strip()

# Compute group means and standard errors
group_stats = df.groupby('gender')['prediction_like'].agg(['mean', 'count',␣
↪'std'])
group_stats['sem'] = group_stats['std'] / np.sqrt(group_stats['count'])

# Perform t-test between gender groups (assuming two groups)
groups = df['gender'].unique()
if len(groups) == 2:
    group1 = df[df['gender'] == groups[0]]['decision']
    group2 = df[df['gender'] == groups[1]]['decision']
    t_stat, p_value = stats.ttest_ind(group1, group2, equal_var=False)

    # Determine significance level
    if p_value < 0.001:
        sig_label = '***'
    elif p_value < 0.01:
        sig_label = '**'
    elif p_value < 0.05:
        sig_label = '*'
    else:
        sig_label = 'n.s.'  # Not significant
else:
    sig_label = ''
    print("Warning: More than two gender groups, skipping t-test.")

# Plot with error bars
plt.figure(figsize=(6, 4))
bars = plt.bar(group_stats.index, group_stats['mean'],
               yerr=group_stats['sem'], capsize=5,
               color=['skyblue', 'salmon'])

plt.title("Average Prediction_Like by Gender")
plt.ylabel("Average Prediction_Like")
plt.xlabel("Gender")
plt.ylim(0, 1)
plt.xticks(rotation=0)
plt.grid(axis='y', linestyle='--', alpha=0.7)

# Annotate with significance stars
```

```
if sig_label and sig_label != 'n.s.':
    max_height = group_stats['mean'].max() + group_stats['sem'].max() + 0.02
    plt.text(0.5, max_height, sig_label, ha='center', fontsize=14)

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



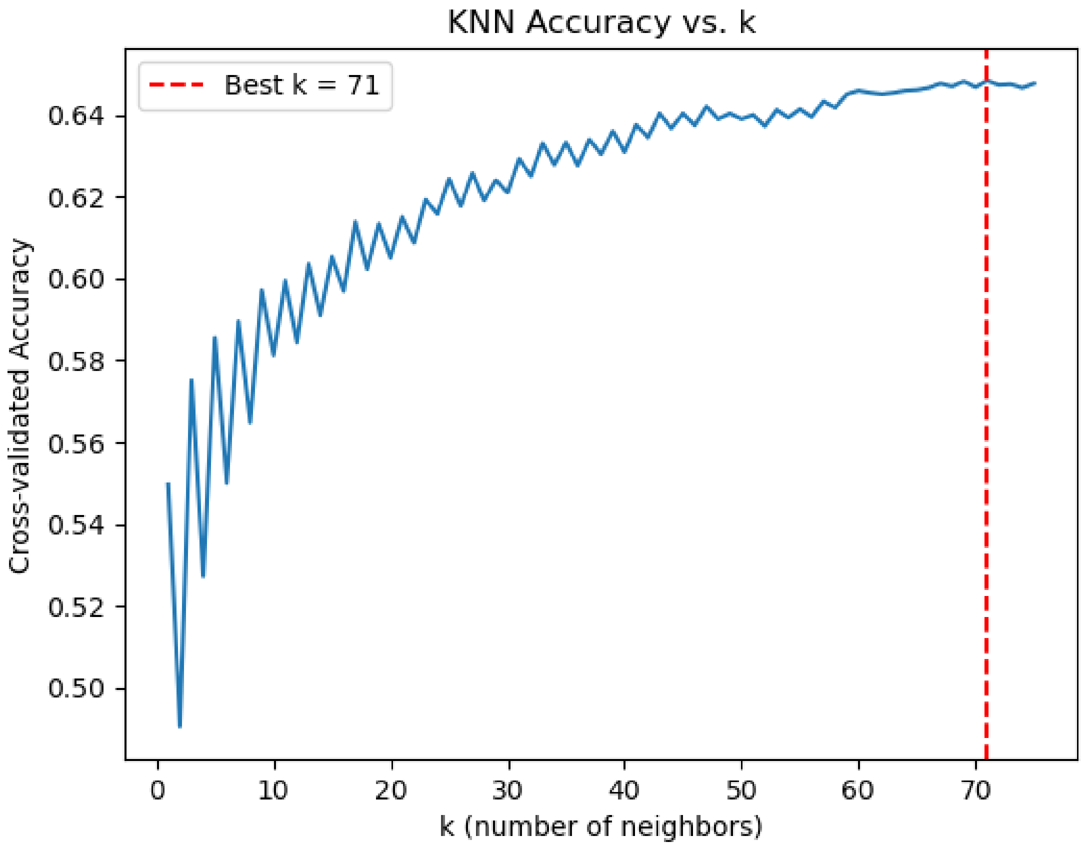

[12]:
```
import pandas as pd
import numpy as np
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.metrics import classification_report
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler

# Encode categorical variables
df['gender_num'] = df['gender'].str.lower().map({'male': 1, 'female': 0})
df['gender_num_o'] = df['gender_o'].str.lower().map({'male': 1, 'female': 0})
df['race_num'] = df['race'].astype('category').cat.codes
df['race_o_num'] = df['race_o'].astype('category').cat.codes
all_fields = pd.concat([df["field"], df["field_o"]], ignore_index=True).
 ↪astype('category')
consistent_categories = all_fields.cat.categories
```

```python
# Apply the same categories to both columns
df["field"] = pd.Categorical(df["field"], categories=consistent_categories)
df["field_o"] = pd.Categorical(df["field_o"], categories=consistent_categories)

# Now generate codes
df["field_num"] = df["field"].cat.codes
df["field_num_o"] = df["field_o"].cat.codes

# Define feature columns (include newly added partner-related variables)
features = [
    'd_age', 'samerace', 'interests_correlate', 'same_field',
    'bias_attractive', 'bias_sincere', 'bias_intelligence',
    'bias_funny', 'bias_ambition',
    'gender_num', 'race_num', 'age', 'field_num',
    'race_o_num', 'age_o',
    'attractive_partner', 'sincere_partner', 'funny_partner',
    'intelligence_partner', 'ambition_partner', 'gender_num_o',
    'field_num_o', 'attractive', 'sincere', 'intelligence', 'funny',
    'ambition'
]

# Prepare feature matrix and target
X = df[features]
y = pd.to_numeric(df['prediction_like'], errors='coerce')

# Drop rows with missing values
model_df = pd.concat([X, y], axis=1).dropna()
X_clean = model_df[features]
y_clean = model_df['prediction_like']

# Use cross-validation to find best k
k_range = range(1, int(np.sqrt(len(X_clean))) + 1)
cv_scores = []

for k in k_range:
    knn_pipeline = Pipeline([
        ('scaler', StandardScaler()),
        ('knn', KNeighborsClassifier(n_neighbors=k))
    ])
    scores = cross_val_score(knn_pipeline, X_clean, y_clean, cv=5,␣
 ↪scoring='accuracy')
    cv_scores.append(scores.mean())

best_k = k_range[np.argmax(cv_scores)]
print(f"Best k: {best_k} with cross-validated accuracy: {max(cv_scores):.4f}")
```

```python
# Final train/test split and model fit
X_train, X_test, y_train, y_test = train_test_split(X_clean, y_clean,␣
↪test_size=0.2, random_state=42)

final_pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('knn', KNeighborsClassifier(n_neighbors=best_k))
])
final_pipeline.fit(X_train, y_train)
y_pred = final_pipeline.predict(X_test)

print("\n=== Final KNN Classification Report ===")
print(classification_report(y_test, y_pred))
```

```
Best k: 71 with cross-validated accuracy: 0.6483

=== Final KNN Classification Report ===
              precision    recall  f1-score   support

           0       0.79      0.04      0.07       399
           1       0.66      0.99      0.79       751

    accuracy                           0.66      1150
   macro avg       0.72      0.52      0.43      1150
weighted avg       0.71      0.66      0.54      1150
```

[13]:
```python
import matplotlib.pyplot as plt

plt.plot(k_range, cv_scores)
plt.xlabel("k (number of neighbors)")
plt.ylabel("Cross-validated Accuracy")
plt.title("KNN Accuracy vs. k")
plt.axvline(x=best_k, color='red', linestyle='--', label=f"Best k = {best_k}")
plt.legend()
plt.show()
#The best value of k is 75, but the value of a higher k begins to diminish␣
↪around k=40. To avoid underfitting and oversmoothing, we pick k=40
```

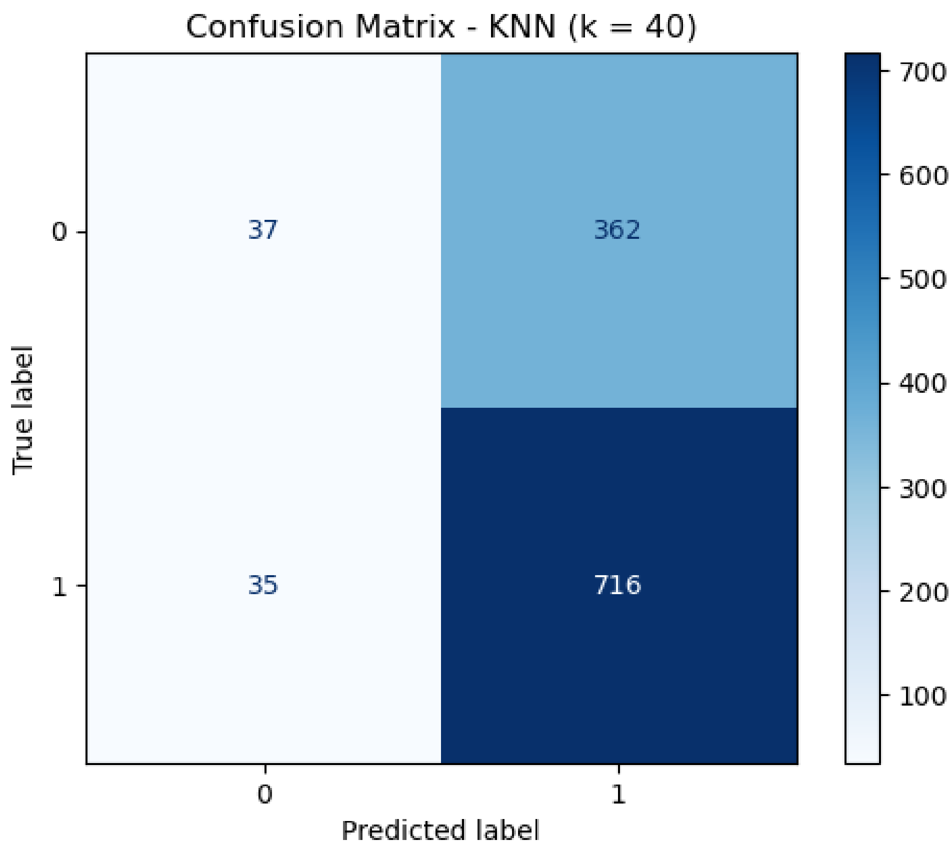


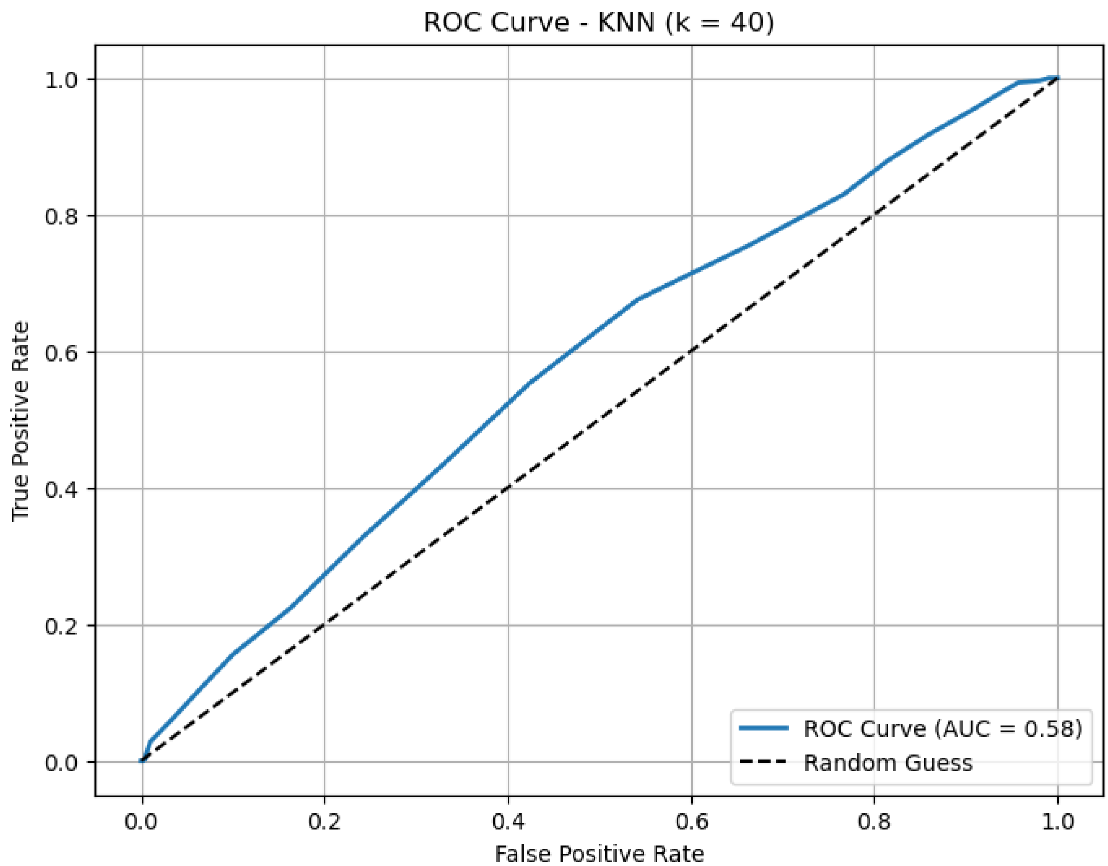

[14]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.metrics import classification_report, confusion_matrix,␣
↪ConfusionMatrixDisplay, roc_auc_score, roc_curve
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler

# === Encode Categorical Variables ===
df['gender_num'] = df['gender'].str.lower().map({'male': 1, 'female': 0})
df['gender_num_o'] = df['gender_o'].str.lower().map({'male': 1, 'female': 0})
df['race_num'] = df['race'].astype('category').cat.codes
df['race_o_num'] = df['race_o'].astype('category').cat.codes

all_fields = pd.concat([df["field"], df["field_o"]], ignore_index=True).
↪astype('category')
consistent_categories = all_fields.cat.categories
df["field"] = pd.Categorical(df["field"], categories=consistent_categories)
```

```
df["field_o"] = pd.Categorical(df["field_o"], categories=consistent_categories)
df["field_num"] = df["field"].cat.codes
df["field_num_o"] = df["field_o"].cat.codes

# === Define Features and Target ===
features = [
    'd_age', 'samerace', 'interests_correlate', 'same_field',
    'bias_attractive', 'bias_sincere', 'bias_intelligence',
    'bias_funny', 'bias_ambition',
    'gender_num', 'race_num', 'age', 'field_num',
    'race_o_num', 'age_o',
    'attractive_partner', 'sincere_partner', 'funny_partner',
    'intelligence_partner', 'ambition_partner', 'gender_num_o',
    'field_num_o', 'attractive', 'sincere', 'intelligence', 'funny',
    'ambition'
]

X = df[features]
y = pd.to_numeric(df['prediction_like'], errors='coerce')

# === Drop missing values ===
model_df = pd.concat([X, y], axis=1).dropna()
X_clean = model_df[features]
y_clean = model_df['prediction_like']

# === Train-Test Split ===
X_train, X_test, y_train, y_test = train_test_split(X_clean, y_clean,␣
↪test_size=0.2, random_state=42)

# === Define and Fit KNN Pipeline (k=40) ===
knn_pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('knn', KNeighborsClassifier(n_neighbors=40))
])
knn_pipeline.fit(X_train, y_train)
y_pred = knn_pipeline.predict(X_test)
y_prob = knn_pipeline.predict_proba(X_test)[:, 1]

# === Evaluation ===
print("\n=== Final KNN Classification Report (k=40) ===")
print(classification_report(y_test, y_pred))

# === Confusion Matrix ===
cm = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot(cmap="Blues")
plt.title("Confusion Matrix - KNN (k = 40)")
```

```python
plt.show()

# === ROC Curve and AUC ===
fpr, tpr, _ = roc_curve(y_test, y_prob)
auc = roc_auc_score(y_test, y_prob)

plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, label=f'ROC Curve (AUC = {auc:.2f})', linewidth=2)
plt.plot([0, 1], [0, 1], 'k--', label='Random Guess')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve - KNN (k = 40)')
plt.legend(loc='lower right')
plt.grid(True)
plt.show()

print(f"AUC Score: {auc:.4f}")
```

```
=== Final KNN Classification Report (k=40) ===
              precision    recall  f1-score   support

           0       0.51      0.09      0.16       399
           1       0.66      0.95      0.78       751

    accuracy                           0.65      1150
   macro avg       0.59      0.52      0.47      1150
weighted avg       0.61      0.65      0.57      1150
```


Confusion Matrix - KNN (k = 40)
37
362
35
716
0
1
True label
0
1
Predicted label
700
600
500
400
300
200
100




AUC Score: 0.5803

[15]:
```
#cross validation for KNN
cv_scores = cross_val_score(knn_pipeline, X_clean, y_clean, cv=5,␣
↪scoring='accuracy')

print("Cross-validated scores (accuracy):", cv_scores)
print("Mean accuracy:", cv_scores.mean())
#KNN model's performance is pretty consistent, with ~65%-68% accuracy whether␣
↪you do one train-test split or 5-fold CV
```

```
Cross-validated scores (accuracy): [0.64608696 0.63565217 0.63913043 0.63391304
0.59965187]
Mean accuracy: 0.6308868959775986
```

[16]:
```
from sklearn.metrics import confusion_matrix, recall_score, mean_squared_error
from sklearn.model_selection import cross_val_score

# === False Positive Rate (FPR) ===
cm = confusion_matrix(y_test, y_pred)
```

```
tn, fp, fn, tp = cm.ravel()
fpr = fp / (fp + tn)
print(f"False Positive Rate (FPR): {fpr:.4f}")

# === Cross-Validated MSE ===
mse_scores = cross_val_score(final_pipeline, X_clean, y_clean, cv=5,␣
↪scoring='neg_mean_squared_error')
mean_mse = -mse_scores.mean()
print(f"Cross-Validated Mean Squared Error (MSE): {mean_mse:.4f}")

# === Recall Score ===
recall = recall_score(y_test, y_pred)
print(f"Recall Score: {recall:.4f}")
```

```
False Positive Rate (FPR): 0.9073
Cross-Validated Mean Squared Error (MSE): 0.3517
Recall Score: 0.9534
```

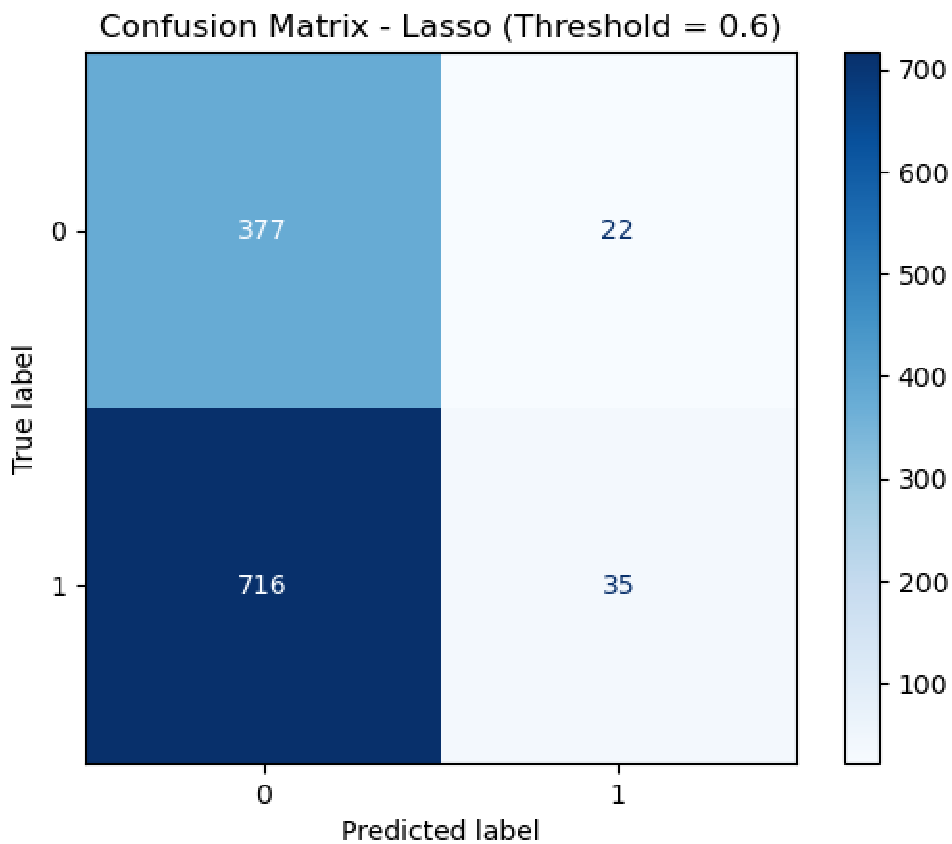
[17]:

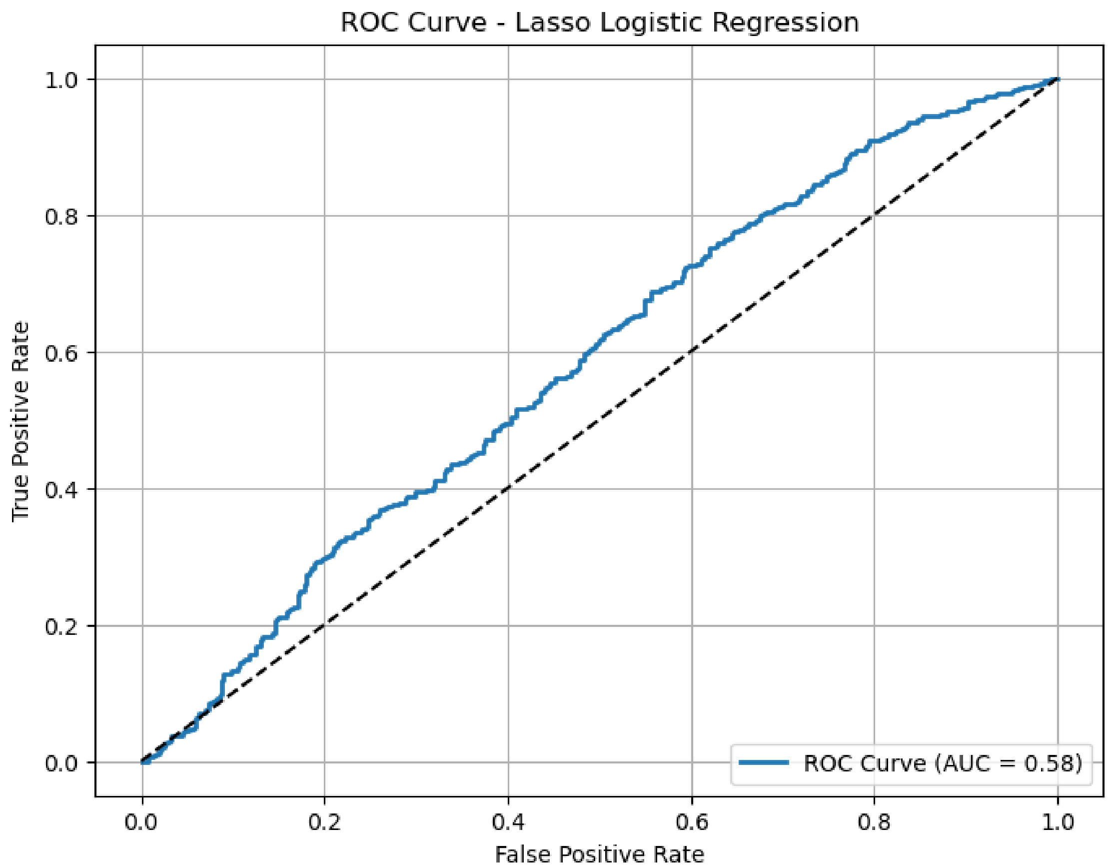
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import (
    classification_report, confusion_matrix, ConfusionMatrixDisplay,
    roc_auc_score, roc_curve
)
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler

# === STEP 1: Encode categorical variables ===
df['gender_num'] = df['gender'].str.lower().map({'male': 1, 'female': 0})
df['gender_num_o'] = df['gender_o'].str.lower().map({'male': 1, 'female': 0})
df['race_num'] = df['race'].astype('category').cat.codes
df['race_o_num'] = df['race_o'].astype('category').cat.codes
all_fields = pd.concat([df["field"], df["field_o"]], ignore_index=True).
↪astype('category')
df["field"] = pd.Categorical(df["field"], categories=all_fields.cat.categories)
df["field_o"] = pd.Categorical(df["field_o"], categories=all_fields.cat.
↪categories)
df["field_num"] = df["field"].cat.codes
df["field_num_o"] = df["field_o"].cat.codes

# === STEP 2: Define predictors and target ===
features = [
    'd_age', 'samerace', 'interests_correlate', 'same_field',
    'bias_attractive', 'bias_sincere', 'bias_intelligence',
```

```
    'bias_funny', 'bias_ambition',
    'gender_num', 'race_num', 'age', 'field_num',
    'race_o_num', 'age_o',
    'attractive_partner', 'sincere_partner', 'funny_partner',
    'intelligence_partner', 'ambition_partner', 'gender_num_o',
    'field_num_o', 'attractive', 'sincere', 'intelligence', 'funny',
    'ambition'
]
X = df[features]
y = pd.to_numeric(df['prediction_like'], errors='coerce')

# === STEP 3: Drop missing values ===
model_df = pd.concat([X, y], axis=1).dropna()
X_clean = model_df[features]
y_clean = model_df['prediction_like']

# === STEP 4: Train-test split ===
X_train, X_test, y_train, y_test = train_test_split(X_clean, y_clean,
 ↪test_size=0.2, random_state=42)

# === STEP 5: Lasso Logistic Regression Pipeline ===
lasso_pipe = Pipeline([
    ('scaler', StandardScaler()),
    ('logreg', LogisticRegression(penalty='l1', solver='liblinear',
 ↪max_iter=1000, class_weight='balanced'))
])

# === STEP 6: Hyperparameter tuning ===
param_grid = {'logreg__C': np.logspace(-4, 4, 20)}
grid = GridSearchCV(lasso_pipe, param_grid, cv=5, scoring='accuracy')
grid.fit(X_train, y_train)

# === STEP 7: Predict and Threshold Tuning ===
y_prob = grid.predict_proba(X_test)[:, 1]

# Adjust threshold (try different cutoffs like 0.5, 0.6, 0.7)
threshold = 0.6
y_pred = (y_prob > threshold).astype(int)

# === STEP 8: Evaluation ===
print(f"Best C: {grid.best_params_['logreg__C']}")
print(f"\n=== Lasso Logistic Regression Classification Report (Threshold =
 ↪{threshold}) ===")
print(classification_report(y_test, y_pred))

# === Confusion Matrix ===
cm = confusion_matrix(y_test, y_pred)
```

```python
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=grid.classes_)
disp.plot(cmap='Blues')
plt.title(f"Confusion Matrix - Lasso (Threshold = {threshold})")
plt.show()

# === ROC Curve and AUC ===
fpr, tpr, _ = roc_curve(y_test, y_prob)
auc = roc_auc_score(y_test, y_prob)

plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, label=f'ROC Curve (AUC = {auc:.2f})', linewidth=2)
plt.plot([0, 1], [0, 1], 'k--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve - Lasso Logistic Regression')
plt.legend(loc='lower right')
plt.grid(True)
plt.show()

print(f"AUC Score: {auc:.4f}")
```

```
Best C: 11.288378916846883

=== Lasso Logistic Regression Classification Report (Threshold = 0.6) ===
              precision    recall  f1-score   support

           0       0.34      0.94      0.51       399
           1       0.61      0.05      0.09       751

    accuracy                           0.36      1150
   macro avg       0.48      0.50      0.30      1150
weighted avg       0.52      0.36      0.23      1150
```


Confusion Matrix - Lasso (Threshold = 0.6)
True label
Predicted label
0
1
0
1
377
22
716
35
700
600
500
400
300
200
100




```
AUC Score: 0.5817
```

[18]:
```
from sklearn.metrics import recall_score, confusion_matrix, make_scorer
from sklearn.model_selection import cross_val_score
from sklearn.metrics import mean_squared_error

# === False Positive Rate ===
tn, fp, fn, tp = confusion_matrix(y_test, y_pred).ravel()
false_positive_rate = fp / (fp + tn)
print(f"False Positive Rate: {false_positive_rate:.4f}")

# === Cross-validated Negative MSE ===
neg_mse_scores = cross_val_score(
    grid.best_estimator_, X_clean, y_clean,
    scoring='neg_mean_squared_error', cv=5
)
mean_neg_mse = neg_mse_scores.mean()
print(f"Cross-Validated Score (Negative MSE): {mean_neg_mse:.4f}")

# === Recall Rate ===
```

```python
recall = recall_score(y_test, y_pred)
print(f"Recall Rate: {recall:.4f}")
```

```
False Positive Rate: 0.0551
Cross-Validated Score (Negative MSE): -0.4905
Recall Rate: 0.0466
```

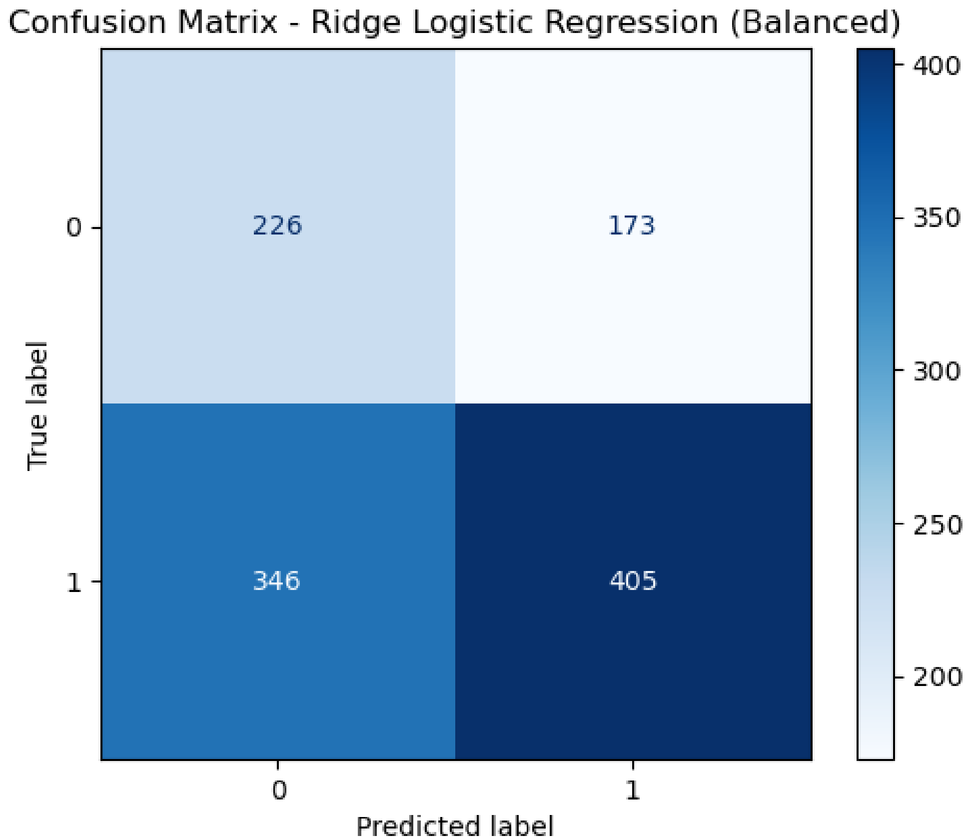
[19]:

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import classification_report, confusion_matrix,␣
 ↪ConfusionMatrixDisplay, roc_auc_score, roc_curve
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler

# === STEP 1: Encode categorical variables ===
df['gender_num'] = df['gender'].str.lower().map({'male': 1, 'female': 0})
df['gender_num_o'] = df['gender_o'].str.lower().map({'male': 1, 'female': 0})
df['race_num'] = df['race'].astype('category').cat.codes
df['race_o_num'] = df['race_o'].astype('category').cat.codes
all_fields = pd.concat([df["field"], df["field_o"]], ignore_index=True).
 ↪astype('category')
consistent_categories = all_fields.cat.categories
df["field"] = pd.Categorical(df["field"], categories=consistent_categories)
df["field_o"] = pd.Categorical(df["field_o"], categories=consistent_categories)
df["field_num"] = df["field"].cat.codes
df["field_num_o"] = df["field_o"].cat.codes

# === STEP 2: Define predictors and target ===
features = [
    'd_age', 'samerace', 'interests_correlate', 'same_field',
    'bias_attractive', 'bias_sincere', 'bias_intelligence',
    'bias_funny', 'bias_ambition',
    'gender_num', 'race_num', 'age', 'field_num',
    'race_o_num', 'age_o',
    'attractive_partner', 'sincere_partner', 'funny_partner',
    'intelligence_partner', 'ambition_partner', 'gender_num_o',
    'field_num_o', 'attractive', 'sincere', 'intelligence', 'funny',
    'ambition'
]
X = df[features]
y = pd.to_numeric(df['prediction_like'], errors='coerce')

# === STEP 3: Drop missing values ===
model_df = pd.concat([X, y], axis=1).dropna()
```

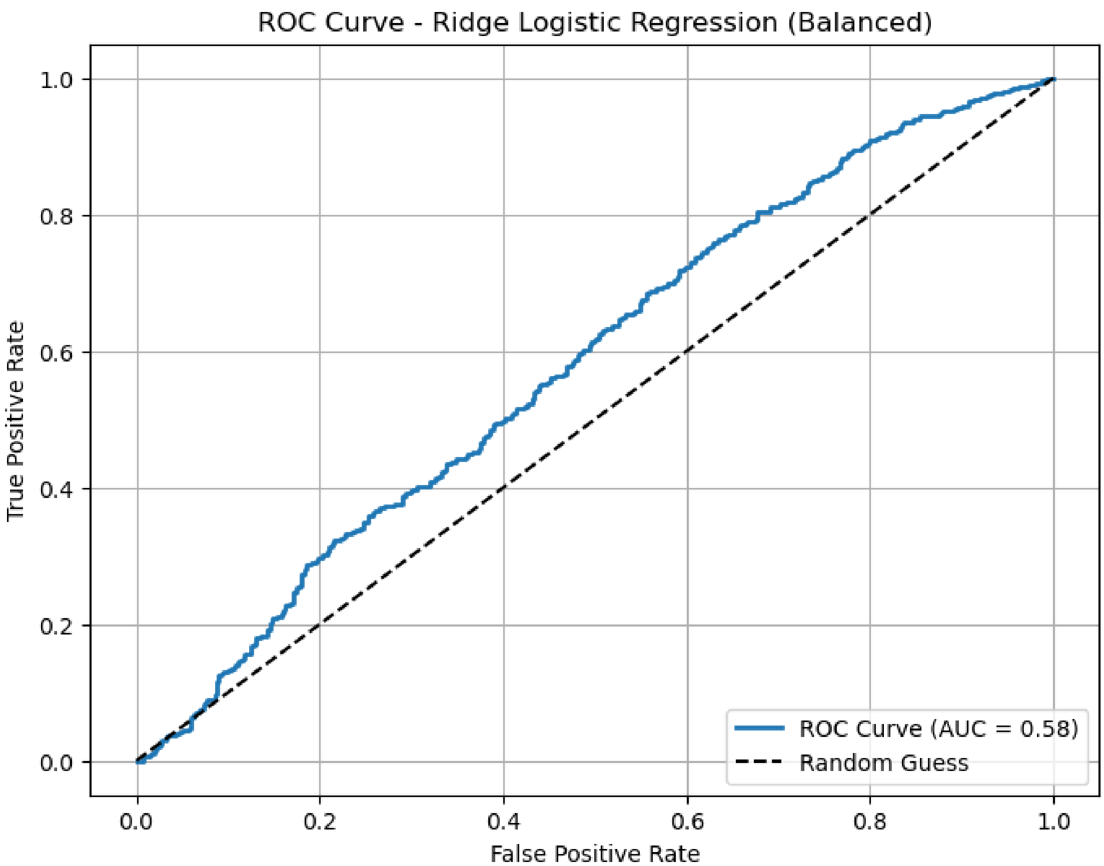
```
X_clean = model_df[features]
y_clean = model_df['prediction_like']

# === STEP 4: Train-test split ===
X_train, X_test, y_train, y_test = train_test_split(X_clean, y_clean,␣
↪test_size=0.2, random_state=42)

# === STEP 5: Ridge Logistic Regression Pipeline with Class Weights ===
ridge_pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('logreg', LogisticRegression(
        penalty='l2',
        solver='liblinear',
        max_iter=1000,
        class_weight='balanced'
    ))
])

# === STEP 6: Hyperparameter tuning ===
param_grid = {'logreg__C': np.logspace(-4, 4, 20)}
grid = GridSearchCV(ridge_pipeline, param_grid, cv=5, scoring='accuracy')
grid.fit(X_train, y_train)

# === STEP 7: Predictions ===
y_pred = grid.predict(X_test)
y_prob = grid.predict_proba(X_test)[:, 1]

# === STEP 8: Evaluation Metrics ===
print(f"Best C (Regularization Strength): {grid.best_params_['logreg__C']}")
print("\n=== Ridge Logistic Regression (Balanced) Classification Report ===")
print(classification_report(y_test, y_pred))

# === STEP 9: Confusion Matrix ===
cm = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=grid.classes_)
disp.plot(cmap='Blues')
plt.title("Confusion Matrix - Ridge Logistic Regression (Balanced)")
plt.show()

# === STEP 10: AUC and ROC Curve ===
fpr, tpr, _ = roc_curve(y_test, y_prob)
auc = roc_auc_score(y_test, y_prob)

plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, label=f'ROC Curve (AUC = {auc:.2f})', linewidth=2)
plt.plot([0, 1], [0, 1], 'k--', label='Random Guess')
plt.xlabel('False Positive Rate')
```

```
plt.ylabel('True Positive Rate')
plt.title('ROC Curve - Ridge Logistic Regression (Balanced)')
plt.legend(loc='lower right')
plt.grid(True)
plt.show()

print(f"\nAUC Score: {auc:.4f}")

# === STEP 11: Coefficients Summary ===
coefs = grid.best_estimator_.named_steps['logreg'].coef_[0]
coef_df = pd.DataFrame({
    'Feature': features,
    'Coefficient': coefs,
    'Odds Ratio': np.exp(coefs)
})
coef_df = coef_df.reindex(coef_df['Coefficient'].abs().
 ↪sort_values(ascending=False).index)

print("\nTop predictors (by absolute coefficient value):")
print(coef_df)
```

```
Best C (Regularization Strength): 0.615848211066026

=== Ridge Logistic Regression (Balanced) Classification Report ===
              precision    recall  f1-score   support

           0       0.40      0.57      0.47       399
           1       0.70      0.54      0.61       751

    accuracy                           0.55      1150
   macro avg       0.55      0.55      0.54      1150
weighted avg       0.59      0.55      0.56      1150
```



Confusion Matrix - Ridge Logistic Regression (Balanced)
226
173
346
405
0
1
True label
0
1
Predicted label
400
350
300
250
200




```
AUC Score: 0.5818

Top predictors (by absolute coefficient value):
                 Feature  Coefficient  Odds Ratio
6      bias_intelligence    -0.240615    0.786144
5           bias_sincere     0.189271    1.208368
23               sincere    -0.164546    0.848278
7             bias_funny    -0.161962    0.850474
24          intelligence     0.151954    1.164107
25                 funny     0.147839    1.159326
8          bias_ambition     0.107552    1.113548
26              ambition    -0.102188    0.902860
4        bias_attractive    -0.076819    0.926058
2    interests_correlate     0.065205    1.067377
18  intelligence_partner    -0.054384    0.947068
12             field_num    -0.053281    0.948113
16       sincere_partner    -0.049120    0.952067
10              race_num    -0.036928    0.963745
14                 age_o    -0.033391    0.967160
```

```
15    attractive_partner   -0.032646    0.967881
20           gender_num_o    0.029064    1.029491
9              gender_num   -0.029064    0.971354
0                   d_age   -0.029041    0.971377
1                samerace   -0.028815    0.971597
21            field_num_o   -0.026908    0.973451
19       ambition_partner   -0.024227    0.976064
13             race_o_num   -0.013148    0.986938
3              same_field   -0.010577    0.989479
22             attractive   -0.003882    0.996126
11                    age    0.002006    1.002008
17          funny_partner    0.000114    1.000114
```

[20]:
```python
from sklearn.metrics import recall_score, confusion_matrix, make_scorer
from sklearn.model_selection import cross_val_score

# === False Positive Rate ===
tn, fp, fn, tp = confusion_matrix(y_test, y_pred).ravel()
false_positive_rate = fp / (fp + tn)
print(f"\nFalse Positive Rate: {false_positive_rate:.4f}")

# === Cross-Validated Negative MSE ===
neg_mse_scores = cross_val_score(
    grid.best_estimator_, X_clean, y_clean,
    scoring='neg_mean_squared_error', cv=5
)
mean_neg_mse = neg_mse_scores.mean()
print(f"Cross-Validated Score (Negative MSE): {mean_neg_mse:.4f}")

# === Recall Rate ===
recall = recall_score(y_test, y_pred)
print(f"Recall Rate: {recall:.4f}")
```

```
False Positive Rate: 0.4336
Cross-Validated Score (Negative MSE): -0.4895
Recall Rate: 0.5393
```

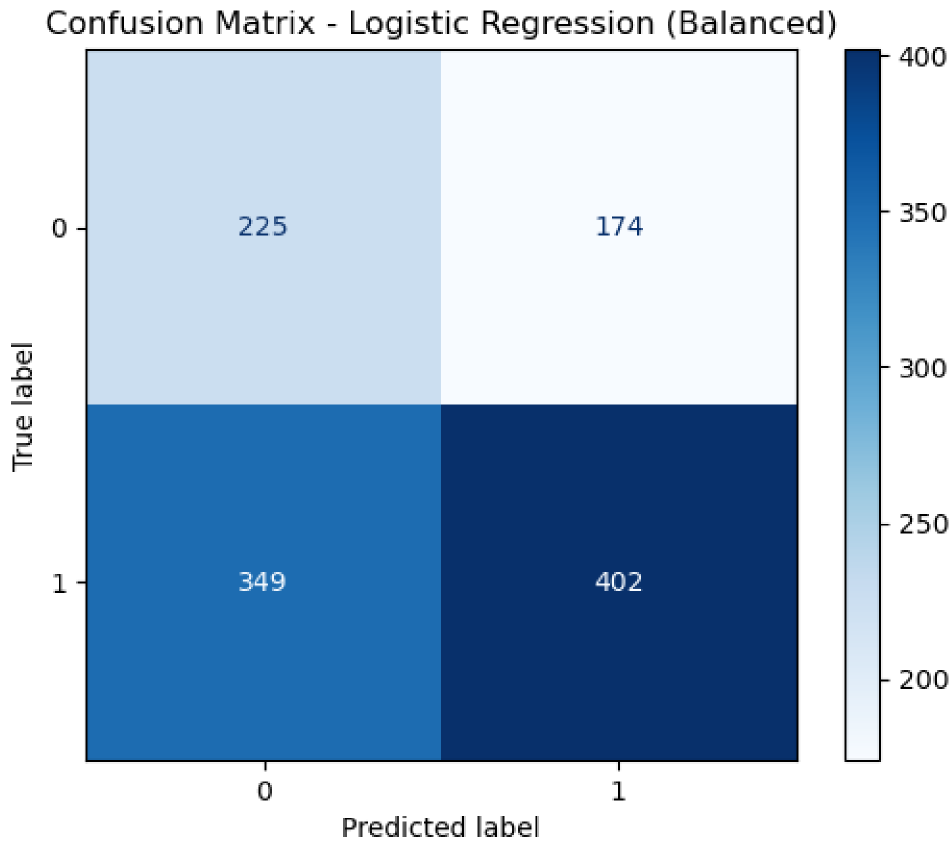
[21]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix,␣
 ↪ConfusionMatrixDisplay, roc_auc_score, roc_curve
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
```

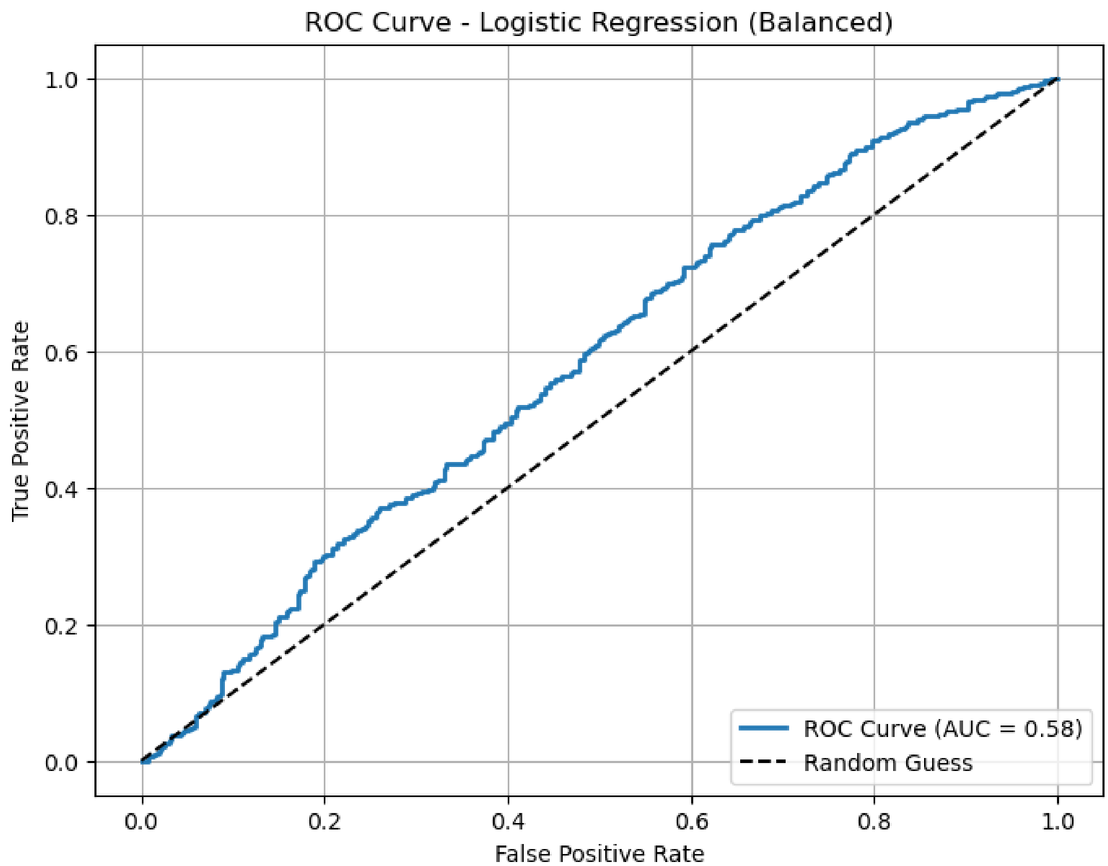
```python
# === STEP 1: Encode categorical variables ===
df['gender_num'] = df['gender'].str.lower().map({'male': 1, 'female': 0})
df['gender_num_o'] = df['gender_o'].str.lower().map({'male': 1, 'female': 0})
df['race_num'] = df['race'].astype('category').cat.codes
df['race_o_num'] = df['race_o'].astype('category').cat.codes

all_fields = pd.concat([df["field"], df["field_o"]], ignore_index=True).
↪astype('category')
consistent_categories = all_fields.cat.categories
df["field"] = pd.Categorical(df["field"], categories=consistent_categories)
df["field_o"] = pd.Categorical(df["field_o"], categories=consistent_categories)
df["field_num"] = df["field"].cat.codes
df["field_num_o"] = df["field_o"].cat.codes

# === STEP 2: Define predictors and outcome ===
features = [
    'd_age', 'samerace', 'interests_correlate', 'same_field',
    'bias_attractive', 'bias_sincere', 'bias_intelligence',
    'bias_funny', 'bias_ambition',
    'gender_num', 'race_num', 'age', 'field_num',
    'race_o_num', 'age_o',
    'attractive_partner', 'sincere_partner', 'funny_partner',
    'intelligence_partner', 'ambition_partner', 'gender_num_o',
    'field_num_o', 'attractive', 'sincere', 'intelligence', 'funny',
    'ambition'
]
X = df[features]
y = pd.to_numeric(df['prediction_like'], errors='coerce')

# === STEP 3: Drop missing values ===
model_df = pd.concat([X, y], axis=1).dropna()
X_clean = model_df[features]
y_clean = model_df['prediction_like']

# === STEP 4: Train/test split ===
X_train, X_test, y_train, y_test = train_test_split(X_clean, y_clean,␣
↪test_size=0.2, random_state=42)

# === STEP 5: Logistic Regression Pipeline with Class Balancing ===
logit_pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('logreg', LogisticRegression(penalty=None, solver='lbfgs', max_iter=1000,␣
↪class_weight='balanced'))
])
logit_pipeline.fit(X_train, y_train)
```

```python
# === STEP 6: Predictions and Probabilities ===
y_pred = logit_pipeline.predict(X_test)
y_prob = logit_pipeline.predict_proba(X_test)[:, 1]

# === STEP 7: Classification Report ===
print("\n=== Logistic Regression Classification Report (Balanced) ===")
print(classification_report(y_test, y_pred))

# === STEP 8: Confusion Matrix ===
cm = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot(cmap='Blues')
plt.title("Confusion Matrix - Logistic Regression (Balanced)")
plt.show()

# === STEP 9: ROC Curve & AUC ===
fpr, tpr, _ = roc_curve(y_test, y_prob)
auc = roc_auc_score(y_test, y_prob)

plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, label=f'ROC Curve (AUC = {auc:.2f})', linewidth=2)
plt.plot([0, 1], [0, 1], 'k--', label='Random Guess')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve - Logistic Regression (Balanced)')
plt.legend(loc='lower right')
plt.grid(True)
plt.show()

print(f"\nAUC Score: {auc:.4f}")

# === STEP 10: Coefficients Summary ===
logreg_model = logit_pipeline.named_steps['logreg']
coefs = logreg_model.coef_[0]
coef_df = pd.DataFrame({'Feature': features, 'Coefficient': coefs})
print("\n=== Logistic Regression Coefficients (Balanced) ===")
print(coef_df.sort_values(by='Coefficient', key=abs, ascending=False))
```

```
=== Logistic Regression Classification Report (Balanced) ===
              precision    recall  f1-score   support

           0       0.39      0.56      0.46       399
           1       0.70      0.54      0.61       751

    accuracy                           0.55      1150
   macro avg       0.54      0.55      0.53      1150
```

```
weighted avg       0.59      0.55      0.56      1150
```






```
AUC Score: 0.5816

=== Logistic Regression Coefficients (Balanced) ===
                 Feature  Coefficient
6      bias_intelligence    -0.256378
5           bias_sincere     0.198173
23               sincere    -0.172667
24          intelligence     0.166952
7             bias_funny    -0.165158
25                 funny     0.149857
8          bias_ambition     0.117470
26              ambition    -0.111364
4        bias_attractive    -0.075939
2    interests_correlate     0.065024
18  intelligence_partner    -0.054669
12             field_num    -0.053585
16       sincere_partner    -0.049315
10              race_num    -0.037101
14                 age_o    -0.033461
```

```
15    attractive_partner    -0.032829
9             gender_num    -0.029148
20          gender_num_o     0.029148
0                  d_age    -0.029006
1               samerace    -0.028844
21           field_num_o    -0.026981
19      ambition_partner    -0.024045
13            race_o_num    -0.013002
3             same_field    -0.010689
22            attractive    -0.004807
11                   age     0.001831
17         funny_partner     0.000234
```

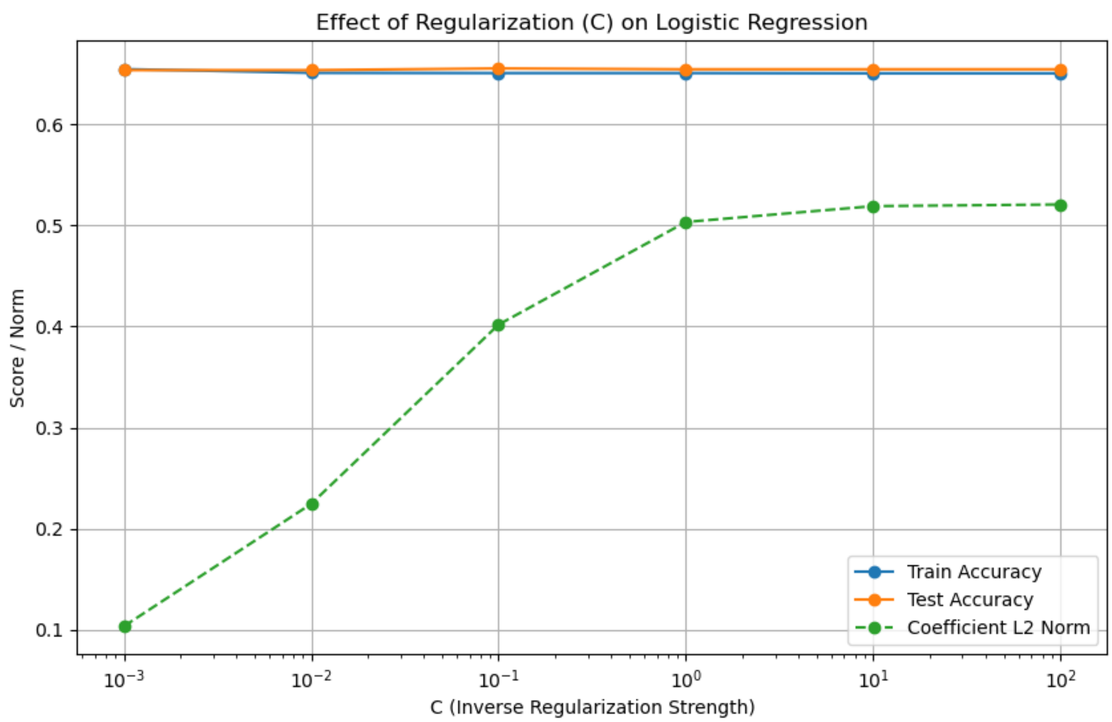
[22]:
```python
from sklearn.metrics import recall_score
from sklearn.model_selection import cross_val_score

# === False Positive Rate ===
tn, fp, fn, tp = confusion_matrix(y_test, y_pred).ravel()
false_positive_rate = fp / (fp + tn)
print(f"\nFalse Positive Rate: {false_positive_rate:.4f}")

# === Cross-Validated Score (Negative MSE) ===
neg_mse_scores = cross_val_score(
    logit_pipeline, X_clean, y_clean,
    cv=5, scoring='neg_mean_squared_error'
)
mean_neg_mse = neg_mse_scores.mean()
print(f"Cross-Validated Score (Negative MSE): {mean_neg_mse:.4f}")

# === Recall Rate ===
recall = recall_score(y_test, y_pred)
print(f"Recall Rate: {recall:.4f}")
```

```
False Positive Rate: 0.4361
Cross-Validated Score (Negative MSE): -0.4897
Recall Rate: 0.5353
```

[23]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
```

```python
# Encode categorical variables
df['gender_num'] = df['gender'].str.lower().map({'male': 1, 'female': 0})
df['gender_num_o'] = df['gender_o'].str.lower().map({'male': 1, 'female': 0})
df['race_num'] = df['race'].astype('category').cat.codes
all_fields = pd.concat([df["field"], df["field_o"]], ignore_index=True).
 ↪astype('category')
consistent_categories = all_fields.cat.categories

# Apply the same categories to both columns
df["field"] = pd.Categorical(df["field"], categories=consistent_categories)
df["field_o"] = pd.Categorical(df["field_o"], categories=consistent_categories)

# Now generate codes
df["field_num"] = df["field"].cat.codes
df["field_num_o"] = df["field_o"].cat.codes

# Define predictors and outcome
features = [
    'd_age', 'samerace', 'interests_correlate', 'same_field',
    'bias_attractive', 'bias_sincere', 'bias_intelligence',
    'bias_funny', 'bias_ambition',
    'gender_num', 'race_num', 'age', 'field_num',
    'race_o_num', 'age_o',
    'attractive_partner', 'sincere_partner', 'funny_partner',
    'intelligence_partner', 'ambition_partner', 'gender_num_o',
    'field_num_o', 'attractive', 'sincere', 'intelligence', 'funny',
    'ambition'
]
X = df[features]
y = pd.to_numeric(df['prediction_like'], errors='coerce')

# Drop rows with any missing data
model_df = pd.concat([X, y], axis=1).dropna()
X_clean = model_df[features]
y_clean = model_df['prediction_like']

# Train/test split
X_train, X_test, y_train, y_test = train_test_split(X_clean, y_clean,␣
 ↪test_size=0.2, random_state=42)

# Loop through different values of C
C_values = [0.001, 0.01, 0.1, 1, 10, 100]
train_acc = []
test_acc = []
coef_norm = []

for C in C_values:
```

```
    pipeline = Pipeline([
        ('scaler', StandardScaler()),
        ('logreg', LogisticRegression(C=C, penalty='l2', solver='liblinear'))
    ])

    pipeline.fit(X_train, y_train)
    y_train_pred = pipeline.predict(X_train)
    y_test_pred = pipeline.predict(X_test)

    train_acc.append(accuracy_score(y_train, y_train_pred))
    test_acc.append(accuracy_score(y_test, y_test_pred))

    coefs = pipeline.named_steps['logreg'].coef_
    coef_norm.append(np.linalg.norm(coefs))  # L2 norm of coefficients

# Plot performance vs. C
plt.figure(figsize=(10, 6))
plt.plot(C_values, train_acc, label='Train Accuracy', marker='o')
plt.plot(C_values, test_acc, label='Test Accuracy', marker='o')
plt.plot(C_values, coef_norm, label='Coefficient L2 Norm', marker='o',␣
 ↪linestyle='--')
plt.xscale('log')
plt.xlabel('C (Inverse Regularization Strength)')
plt.ylabel('Score / Norm')
plt.title('Effect of Regularization (C) on Logistic Regression')
plt.legend()
plt.grid(True)
plt.show()
```



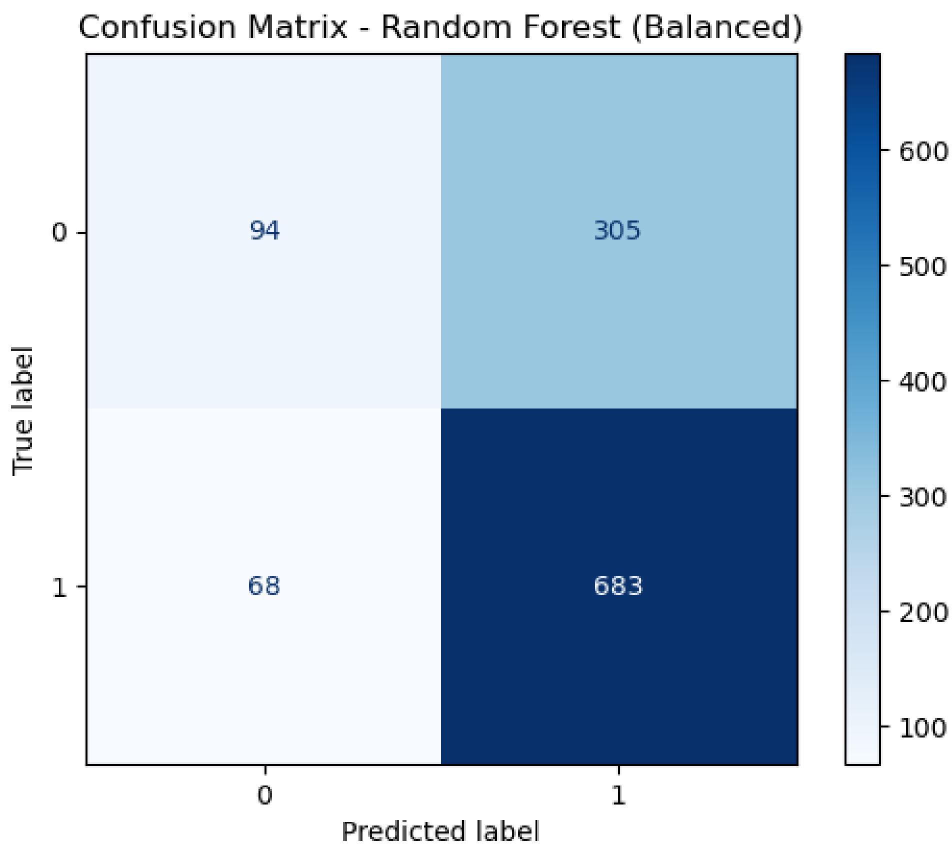

[24]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import classification_report, confusion_matrix,␣
↪ConfusionMatrixDisplay, roc_auc_score, roc_curve
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler

# === STEP 1: Encode categorical variables ===
df['gender_num'] = df['gender'].str.lower().map({'male': 1, 'female': 0})
df['gender_num_o'] = df['gender_o'].str.lower().map({'male': 1, 'female': 0})
df['race_num'] = df['race'].astype('category').cat.codes
df['race_o_num'] = df['race_o'].astype('category').cat.codes

all_fields = pd.concat([df["field"], df["field_o"]], ignore_index=True).
↪astype('category')
consistent_categories = all_fields.cat.categories
df["field"] = pd.Categorical(df["field"], categories=consistent_categories)
df["field_o"] = pd.Categorical(df["field_o"], categories=consistent_categories)
df["field_num"] = df["field"].cat.codes
df["field_num_o"] = df["field_o"].cat.codes
```

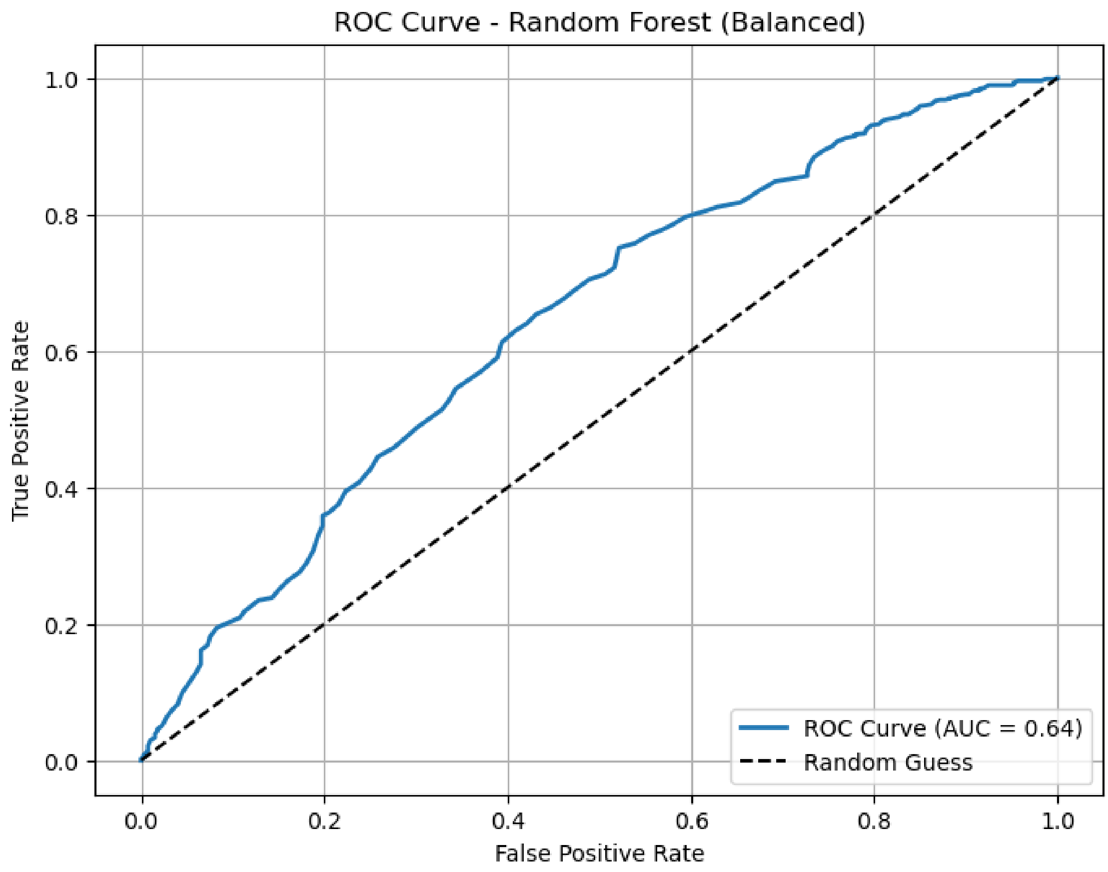
```python
# === STEP 2: Define predictors ===
features = [
    'd_age', 'samerace', 'interests_correlate', 'same_field',
    'bias_attractive', 'bias_sincere', 'bias_intelligence',
    'bias_funny', 'bias_ambition',
    'gender_num', 'race_num', 'age', 'field_num',
    'race_o_num', 'age_o',
    'attractive_partner', 'sincere_partner', 'funny_partner',
    'intelligence_partner', 'ambition_partner', 'gender_num_o',
    'field_num_o', 'attractive', 'sincere', 'intelligence', 'funny',
    'ambition'
]
X = df[features]
y = pd.to_numeric(df['prediction_like'], errors='coerce')

# === STEP 3: Drop missing values ===
model_df = pd.concat([X, y], axis=1).dropna()
X_clean = model_df[features]
y_clean = model_df['prediction_like']

# === STEP 4: Train-test split ===
X_train, X_test, y_train, y_test = train_test_split(X_clean, y_clean,␣
 ↪test_size=0.2, random_state=42)

# === STEP 5: Pipeline with class balancing ===
rf_pipe = Pipeline([
    ('scaler', StandardScaler()),
    ('rf', RandomForestClassifier(random_state=42, class_weight='balanced'))
])

# === STEP 6: Grid Search Tuning ===
param_grid = {
    'rf__n_estimators': [100, 200],
    'rf__max_depth': [None, 5, 10],
    'rf__min_samples_split': [2, 5],
    'rf__min_samples_leaf': [1, 2]
}
grid = GridSearchCV(rf_pipe, param_grid, cv=5, scoring='accuracy')
grid.fit(X_train, y_train)

# === STEP 7: Evaluation ===
y_pred = grid.predict(X_test)
y_prob = grid.predict_proba(X_test)[:, 1]

print(f"Best Parameters: {grid.best_params_}")
print("\n=== Classification Report (Balanced Random Forest) ===")
print(classification_report(y_test, y_pred))
```

```python
# === STEP 8: Confusion Matrix ===
cm = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot(cmap='Blues')
plt.title("Confusion Matrix - Random Forest (Balanced)")
plt.show()

# === STEP 9: ROC Curve and AUC ===
fpr, tpr, _ = roc_curve(y_test, y_prob)
auc_score = roc_auc_score(y_test, y_prob)

plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, label=f'ROC Curve (AUC = {auc_score:.2f})', linewidth=2)
plt.plot([0, 1], [0, 1], 'k--', label='Random Guess')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve - Random Forest (Balanced)')
plt.legend(loc='lower right')
plt.grid(True)
plt.show()

print(f"\nAUC Score: {auc_score:.4f}")

# === STEP 10: Feature Importance ===
best_rf = grid.best_estimator_.named_steps['rf']
importances = best_rf.feature_importances_
importance_df = pd.DataFrame({'Feature': features, 'Importance': importances})
importance_df = importance_df.sort_values(by='Importance', ascending=False)

# Print top features
print("\n=== Feature Importances (Balanced RF) ===")
print(importance_df)

# Plot
plt.figure(figsize=(10, 6))
plt.barh(importance_df['Feature'], importance_df['Importance'])
plt.xlabel("Importance Score")
plt.title("Random Forest Feature Importances (Balanced)")
plt.gca().invert_yaxis()
plt.tight_layout()
plt.show()
```

```
Best Parameters: {'rf__max_depth': None, 'rf__min_samples_leaf': 1,
'rf__min_samples_split': 2, 'rf__n_estimators': 200}

=== Classification Report (Balanced Random Forest) ===
```

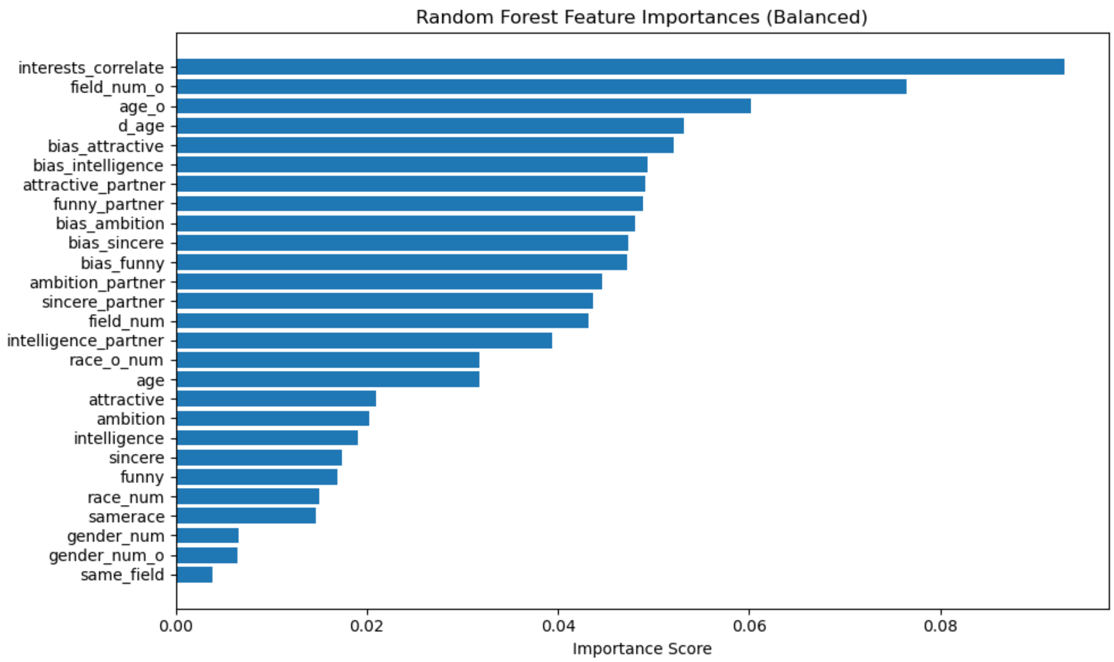
| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 0.58 | 0.24 | 0.34 | 399 |
| 1 | 0.69 | 0.91 | 0.79 | 751 |
| accuracy | | | 0.68 | 1150 |
| macro avg | 0.64 | 0.57 | 0.56 | 1150 |
| weighted avg | 0.65 | 0.68 | 0.63 | 1150 |



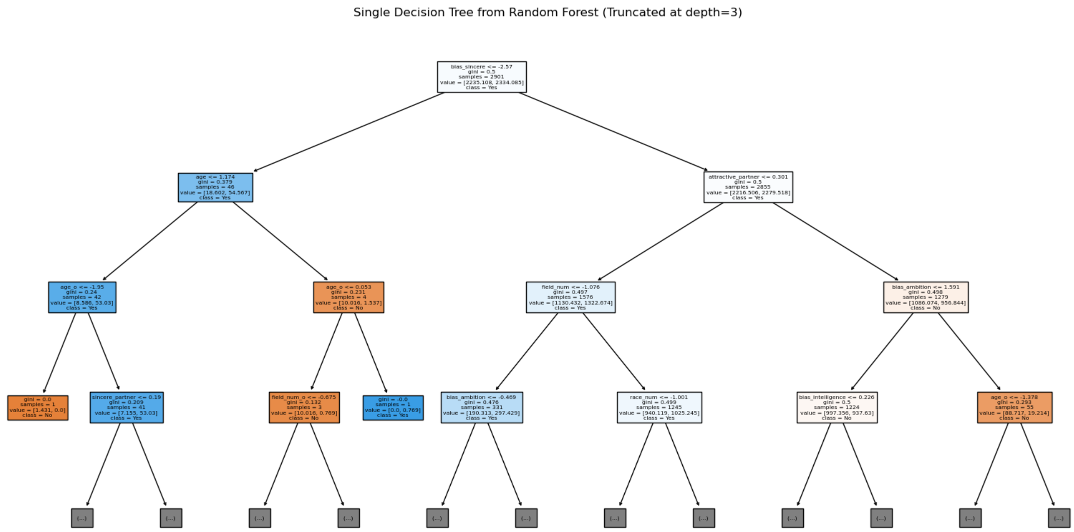



```
AUC Score: 0.6431

=== Feature Importances (Balanced RF) ===
                 Feature  Importance
2    interests_correlate    0.093009
21           field_num_o    0.076490
14                 age_o    0.060217
0                  d_age    0.053130
4        bias_attractive    0.052048
6      bias_intelligence    0.049330
15    attractive_partner    0.049122
17         funny_partner    0.048944
8          bias_ambition    0.048091
5           bias_sincere    0.047338
7             bias_funny    0.047194
19      ambition_partner    0.044592
16       sincere_partner    0.043707
12             field_num    0.043232
18  intelligence_partner    0.039408
```

```
13          race_o_num    0.031732
11                 age    0.031721
22          attractive    0.020900
26            ambition    0.020169
24        intelligence    0.018985
23             sincere    0.017391
25               funny    0.016894
10            race_num    0.015024
1             samerace    0.014583
9           gender_num    0.006520
20        gender_num_o    0.006472
3           same_field    0.003758
```




[25]:
```
from sklearn.tree import plot_tree, export_text
import matplotlib.pyplot as plt

# === STEP 11: Visualize One Decision Tree ===
# Extract one tree from the best Random Forest model
one_tree = best_rf.estimators_[0]  # You can change index to view other trees

# Plot the decision tree
plt.figure(figsize=(20, 10))
plot_tree(one_tree, feature_names=features, class_names=["No", "Yes"],␣
 ↪filled=True, max_depth=3)
plt.title("Single Decision Tree from Random Forest (Truncated at depth=3)")
plt.show()
```




[26]:
```
from sklearn.metrics import recall_score
from sklearn.model_selection import cross_val_score

# === False Positive Rate ===
tn, fp, fn, tp = confusion_matrix(y_test, y_pred).ravel()
false_positive_rate = fp / (fp + tn)
print(f"\nFalse Positive Rate: {false_positive_rate:.4f}")

# === Cross-Validated Score (Negative MSE) ===
neg_mse_scores = cross_val_score(
    grid.best_estimator_, X_clean, y_clean,
    cv=5, scoring='neg_mean_squared_error'
)
mean_neg_mse = neg_mse_scores.mean()
print(f"Cross-Validated Score (Negative MSE): {mean_neg_mse:.4f}")

# === Recall Rate ===
recall = recall_score(y_test, y_pred)
print(f"Recall Rate: {recall:.4f}")
```

```
False Positive Rate: 0.7644
Cross-Validated Score (Negative MSE): -0.3749
Recall Rate: 0.9095
```

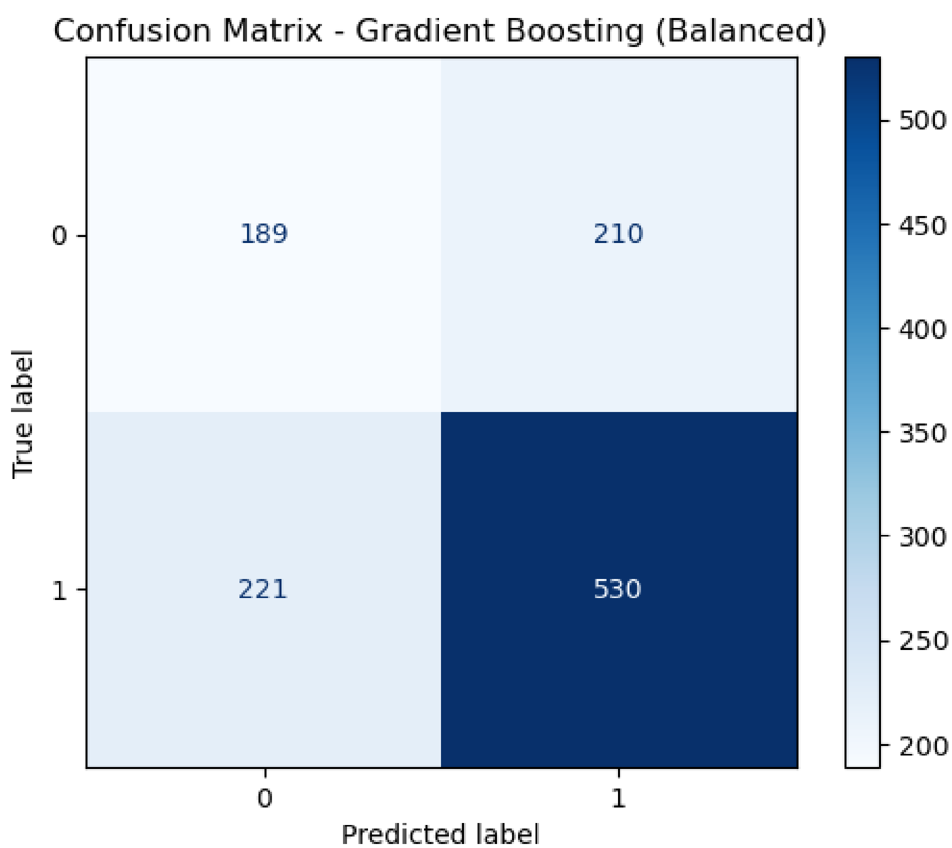
[27]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

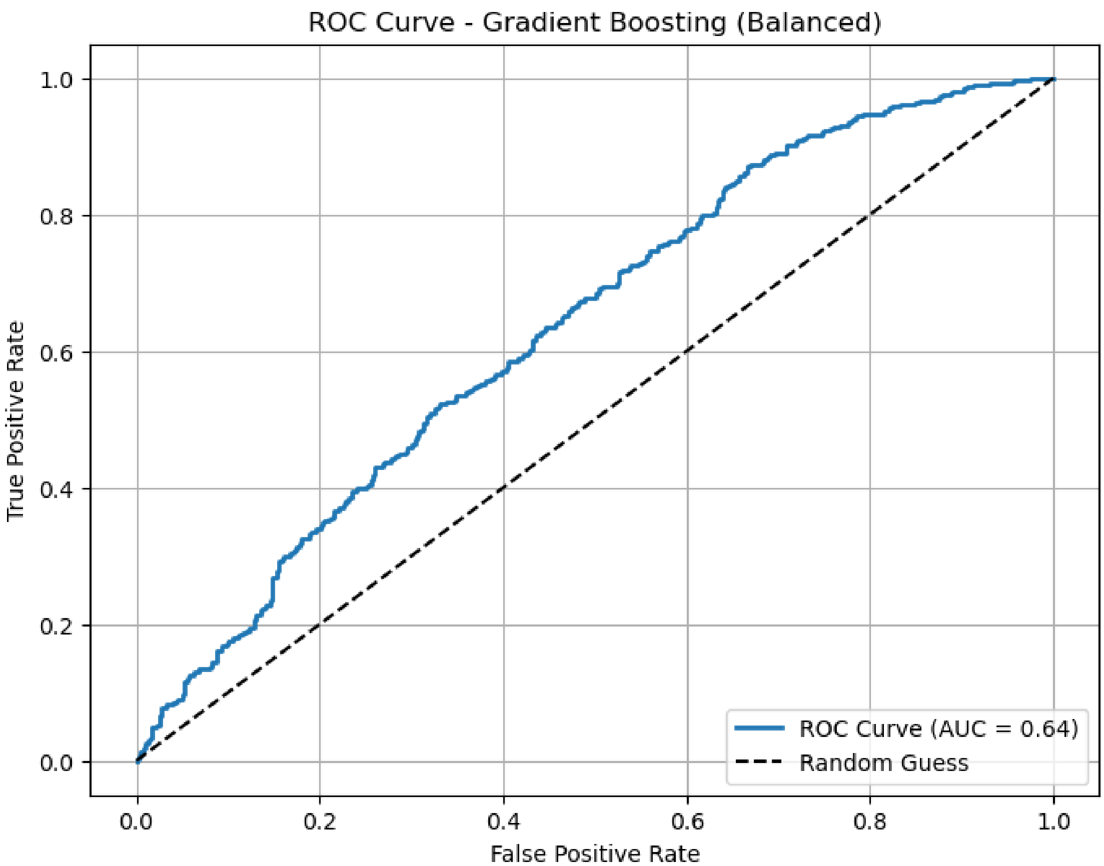
```python
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import classification_report, roc_auc_score, roc_curve,␣
↪confusion_matrix, ConfusionMatrixDisplay
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.utils.class_weight import compute_sample_weight

# === STEP 1: Encode categorical variables ===
df['gender_num'] = df['gender'].str.lower().map({'male': 1, 'female': 0})
df['gender_num_o'] = df['gender_o'].str.lower().map({'male': 1, 'female': 0})
df['race_num'] = df['race'].astype('category').cat.codes
df['race_o_num'] = df['race_o'].astype('category').cat.codes

all_fields = pd.concat([df["field"], df["field_o"]], ignore_index=True).
↪astype('category')
consistent_categories = all_fields.cat.categories
df["field"] = pd.Categorical(df["field"], categories=consistent_categories)
df["field_o"] = pd.Categorical(df["field_o"], categories=consistent_categories)
df["field_num"] = df["field"].cat.codes
df["field_num_o"] = df["field_o"].cat.codes

# === STEP 2: Define predictors and target ===
features = [
    'd_age', 'samerace', 'interests_correlate', 'same_field',
    'bias_attractive', 'bias_sincere', 'bias_intelligence',
    'bias_funny', 'bias_ambition',
    'gender_num', 'race_num', 'age', 'field_num',
    'race_o_num', 'age_o',
    'attractive_partner', 'sincere_partner', 'funny_partner',
    'intelligence_partner', 'ambition_partner', 'gender_num_o',
    'field_num_o', 'attractive', 'sincere', 'intelligence', 'funny',
    'ambition'
]
X = df[features]
y = pd.to_numeric(df['prediction_like'], errors='coerce')

# === STEP 3: Drop missing values ===
model_df = pd.concat([X, y], axis=1).dropna()
X_clean = model_df[features]
y_clean = model_df['prediction_like']

# === STEP 4: Train/test split ===
X_train, X_test, y_train, y_test = train_test_split(X_clean, y_clean,␣
↪test_size=0.2, random_state=42)

# === STEP 5: Sample weights for class balancing ===
```

```python
sample_weights = compute_sample_weight(class_weight='balanced', y=y_train)

# === STEP 6: Boosting pipeline ===
boost_pipe = Pipeline([
    ('scaler', StandardScaler()),
    ('boost', GradientBoostingClassifier(random_state=42))
])

# === STEP 7: Hyperparameter tuning ===
param_grid = {
    'boost__n_estimators': [50, 100, 150],
    'boost__learning_rate': [0.01, 0.05, 0.1],
    'boost__max_depth': [2, 3, 5]
}

grid = GridSearchCV(boost_pipe, param_grid, cv=5, scoring='accuracy')
grid.fit(X_train, y_train, boost__sample_weight=sample_weights)

# === STEP 8: Predict and evaluate ===
y_pred = grid.predict(X_test)
y_prob = grid.predict_proba(X_test)[:, 1]

print(f"Best parameters: {grid.best_params_}")
print("\n=== Gradient Boosting (Balanced) Classification Report ===")
print(classification_report(y_test, y_pred))

# === Confusion matrix ===
cm = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot(cmap='Blues')
plt.title("Confusion Matrix - Gradient Boosting (Balanced)")
plt.show()

# === ROC Curve and AUC ===
fpr, tpr, _ = roc_curve(y_test, y_prob)
auc = roc_auc_score(y_test, y_prob)

plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, label=f'ROC Curve (AUC = {auc:.2f})', linewidth=2)
plt.plot([0, 1], [0, 1], 'k--', label='Random Guess')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve - Gradient Boosting (Balanced)')
plt.legend(loc='lower right')
plt.grid(True)
plt.show()
```

```python
print(f"\nAUC Score: {auc:.4f}")
```

```
Best parameters: {'boost__learning_rate': 0.1, 'boost__max_depth': 5,
'boost__n_estimators': 150}

=== Gradient Boosting (Balanced) Classification Report ===
              precision    recall  f1-score   support

           0       0.46      0.47      0.47       399
           1       0.72      0.71      0.71       751

    accuracy                           0.63      1150
   macro avg       0.59      0.59      0.59      1150
weighted avg       0.63      0.63      0.63      1150
```






```
AUC Score: 0.6364
```

[28]:

```python
from sklearn.metrics import recall_score
from sklearn.model_selection import cross_val_score

# === False Positive Rate ===
tn, fp, fn, tp = confusion_matrix(y_test, y_pred).ravel()
false_positive_rate = fp / (fp + tn)
print(f"\nFalse Positive Rate: {false_positive_rate:.4f}")

# === Cross-Validated Score (Negative MSE) ===
neg_mse_scores = cross_val_score(
    grid.best_estimator_, X_clean, y_clean,
    cv=5, scoring='neg_mean_squared_error'
)
mean_neg_mse = neg_mse_scores.mean()
print(f"Cross-Validated Score (Negative MSE): {mean_neg_mse:.4f}")

# === Recall Rate ===
```

```python
recall = recall_score(y_test, y_pred)
print(f"Recall Rate: {recall:.4f}")
```

```
False Positive Rate: 0.5263
Cross-Validated Score (Negative MSE): -0.4104
Recall Rate: 0.7057
```

[ ]:

```python

```