Date-A-Scientist: Zodiac Sign Prediction using OKCupid Profiles

📖 Introduction

In this project, we analyze data from OKCupid, a popular dating application, to understand user profiles and preferences.
The dataset contains diverse information including demographic details, lifestyle habits, and free-text essay responses.

❓ The Problem

Astrology is a popular conversation starter in dating.
However, a significant portion of users (approx. 18%) do not list their zodiac sign.

This project aims to build a Machine Learning model to predict a user's Zodiac Sign based on:

- Lifestyle choices (drinking, smoking, drugs)
- Self-written essays

🎯 Goal

To determine if there is a statistically significant correlation between a person's writing style/lifestyle and their astrological sign, and to deploy a classifier capable of predicting the missing signs.

🛠️ Project Setup

In this first step, we'll prepare the workspace for the **OKCupid Date-A-Scientist** project.

📦 Download the Project Materials

1. Download the ZIP file for the **Date-A-Scientist** project from the following link:
   👉 Download Starter Files
2. Unzip the folder. You should see two items:
   - `date-a-scientist.ipynb` (a blank Jupyter Notebook)
   - `profiles.csv` (the dataset containing user profiles)

🚀 Launch Jupyter Notebook

1. Open your terminal or command line interface.
2. Type the following command to start Jupyter Notebook:

   ```
   jupyter notebook
   ```
3. A browser tab will open automatically.
4. Click on `date-a-scientist.ipynb` to open the notebook.
5. Build your project inside this file.

🧠 How to Use Jupyter Notebook

Jupyter Notebook is an interactive tool that lets you combine code, visualizations, and explanatory text in one document. It's perfect for data science projects because it supports exploration, analysis, and storytelling.

If you need help setting up or using Jupyter Notebook, check out these resources:

- Command Line Interface Setup
- Introducing Jupyter Notebook
- Setting up Jupyter Notebook
- Getting Started with Jupyter
- Getting More out of Jupyter Notebook

🔐 Git Repository Setup with SSH and LFS

To manage version control and large files efficiently, this project uses **Git**, **SSH authentication**, and **Git LFS.**

📁 Local Project Path

```
cd code/DataScientistML_Codecademy/OKCupid-Date-A-Scientist-Starter
```

🧱 Initialize and Push to Remote Repository

```
git init
git lfs install
git add .
git commit -m "Initial commit for OKCupid Date-A-Scientist project"
git remote add origin git@personal-github:gabrielarcangelbol/OKCupid-Date-A-Scientist-Starter.git
git branch -M main
git push -u origin main
```

⚙️ Configure `.gitattributes` for Large Files

To track CSV files with Git LFS, the following line was added to `.gitattributes`:

```
*.csv filter=lfs diff=lfs merge=lfs -text
```

Then committed:

```
git add .gitattributes
git commit -m "Configure Git LFS for CSV files"
```

🔐 SSH Key Setup for Personal GitHub Account

To ensure Git uses your **personal SSH key**, follow these steps:

1. Start the SSH agent:

   ```
   eval "$(ssh-agent -s)"
   ```
2. Add your personal SSH key:

   ```
   ssh-add ~/.ssh/id_rsa_personal
   ```

🔍 Verify Remote Configuration

To confirm the correct remote is set:

```
cd code/DataScientistML_Codecademy/OKCupid-Date-A-Scientist-Starter
git remote -v
```

You should see:

```
origin  git@personal-github:gabrielarcangelbol/OKCupid-Date-A-Scientist-Starter.git (fetch)
origin  git@personal-github:gabrielarcangelbol/OKCupid-Date-A-Scientist-Starter.git (push)
```

✅ With Git, SSH, and LFS configured, your project is ready for version-controlled development and collaboration.

🧭 Project Scoping

Properly scoping your project creates structure and helps you think through your entire workflow before diving into the analysis. This section outlines the key components of the project scope, inspired by the University of Chicago's Data Science Project Scoping Guide.

🎯 1. Project Goals

- **Primary Objective**: Explore the OKCupid dataset to uncover patterns in dating preferences and behaviors using machine learning and NLP techniques.
- **Secondary Goals**:
  - Identify which user attributes are most predictive of compatibility.
  - Apply clustering to discover latent user segments.
  - Build a supervised model to predict user traits or preferences based on profile text.

📊 2. Data Overview

- **Source**: `profiles.csv` from OKCupid (provided in starter files)
- **Structure**: Each row represents a user profile with multiple features including:
  - Demographics (age, sex, orientation, location)
  - Lifestyle (diet, smoking, drinking, drugs)
  - Essay responses (10 free-text fields)
- **Challenges**:
  - Missing values
  - Unstructured text (essays)
  - Potential class imbalance

🧪 3. Analytical Approach

Exploratory Data Analysis (EDA)

- Distribution of key demographics
- Missing data patterns
- Word frequency and sentiment in essay fields

Feature Engineering

- Text vectorization (TF-IDF, embeddings)
- Aggregated lifestyle indicators
- Derived compatibility scores

Modeling

- **Unsupervised**: Clustering (e.g., KMeans, DBSCAN) to identify user segments
- **Supervised**: Classification (e.g., logistic regression, random forest) to predict:
  - Smoking habits
  - Orientation
  - Personality traits (inferred from essays)

🧱 4. Constraints & Assumptions

- **Assumptions**:
  - Users are honest in their profiles
  - Essay content reflects personality and preferences
- **Constraints**:
  - No access to actual match outcomes
  - Limited metadata on user interactions

🔄 5. Risks & Adjustments

- NLP models may underperform due to short or noisy text
- Clustering may not yield interpretable segments
- Some hypotheses may not be supported by the data

📌 Next Steps

- Clean and preprocess the dataset
- Perform EDA and visualize key trends
- Define target variables for modeling
- Iterate on feature engineering and model evaluation

📚 For more guidance, refer to the full Data Science Project Scoping Guide and Scoping Worksheet (PDF).

Sources: Scoping Guide, Scoping Worksheet PDF

🤖 Select ML-Solvable Problem

In this section, we define a machine learning problem that is feasible given the dataset and project scope. The goal is to identify a task that:

- Can be solved using available data
- Is achievable within a reasonable timeframe
- Has a clear input-output structure suitable for ML
- Aligns with the broader goals of the project

🧩 Problem Statement

Many users on OKCupid consider **Zodiac signs** important when evaluating compatibility. However, not all users provide their zodiac sign in their profile. This creates a gap in the matching process.

Problem: Can we predict a user's zodiac sign based on other profile attributes such as lifestyle habits (drinking, smoking, drug use) and free-text essay responses?

🧠 ML Framing

This is a **multi-class classification** problem with 12 possible zodiac labels.

Input Features:

- Categorical: `drinks`, `smokes`, `drugs`, `job`, `education`
- Textual: `essay0` to `essay9` (to be vectorized using NLP techniques)

Target Variable:

- `sign` (Zodiac sign)

🔍 Why This Problem?

- The `sign` column has missing values, making it a good candidate for imputation via ML.
- Lifestyle and personality traits (expressed in essays) may correlate with astrological archetypes.
- Predicting zodiac signs could enhance match recommendations for users who omit this field.

🧪 ML Techniques to Be Used

- **Text preprocessing**: cleaning, tokenization, TF-IDF or embeddings
- **Feature encoding**: one-hot or ordinal encoding for categorical variables
- **Modeling**:
  - Baseline: Logistic Regression or Naive Bayes
  - Advanced: Random Forest, XGBoost, or fine-tuned transformer (if time permits)
- **Evaluation**: Accuracy, F1-score, confusion matrix

📌 Next Steps

- Analyze class distribution of `sign`
- Explore correlations between zodiac and lifestyle features
- Preprocess essays and engineer features
- Train and evaluate classification models

📚 Reference: Google's Guide to Identifying ML-Suitable Problems

📂 Load and Check Data

Before applying any machine learning techniques, we need to load the dataset and verify that it contains a suitable **label or response variable** for supervised learning.

The dataset is stored in `profiles.csv` and includes:

📑 Multiple-choice columns:

- `body_type`, `diet`, `drinks`, `drugs`, `education`, `ethnicity`, `height`, `income`, `job`, `offspring`, `orientation`, `pets`, `religion`, `sex`, `sign`, `smokes`, `speaks`, `status`

📝 Essay fields:

- `essay0` to `essay9`, covering topics like self-summary, lifestyle, preferences, and personality

---

🎯 Target Variable Check

Since our selected ML task is to **predict the user's zodiac sign**, we will check the `sign` column for:

- Presence of values
- Class distribution
- Missing data

If the `sign` column is too sparse or unreliable, we may need to revisit our problem definition.

📦 Load Dataset

```
import pandas as pd

# Load the dataset
df = pd.read_csv("profiles.csv")

# Display basic info
print("Shape of dataset:", df.shape)
df.info()

# Preview the first few rows
df.head()
```

✅ Next Steps

- Check for missing values in the `sign` column
- Explore distribution of zodiac signs
- Assess completeness of other relevant features (`drinks`, `smokes`, `drugs`, `essay0`–`essay9`)
- Decide whether to proceed with this target or redefine the ML problem

In [1]:
```
# 📦 Load and Inspect OKCupid Dataset

import pandas as pd

# Load the dataset
df = pd.read_csv("profiles.csv")

# Basic shape and structure
print("✅ Dataset loaded successfully.")
print("🔢 Number of rows and columns:", df.shape)

# Overview of column types and missing values
print("\n📋 Dataset Info:")
df.info()

# Preview the first few rows
```

```
print("\n🔍 First 5 rows:")
df.head()
```

```
✅ Dataset loaded successfully.
🔢 Number of rows and columns: (59946, 31)

📋 Dataset Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 59946 entries, 0 to 59945
Data columns (total 31 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   age          59946 non-null  int64  
 1   body_type    54650 non-null  object 
 2   diet         35551 non-null  object 
 3   drinks       56961 non-null  object 
 4   drugs        45866 non-null  object 
 5   education    53318 non-null  object 
 6   essay0       54458 non-null  object 
 7   essay1       52374 non-null  object 
 8   essay2       50308 non-null  object 
 9   essay3       48470 non-null  object 
 10  essay4       49409 non-null  object 
 11  essay5       49096 non-null  object 
 12  essay6       46175 non-null  object 
 13  essay7       47495 non-null  object 
 14  essay8       40721 non-null  object 
 15  essay9       47343 non-null  object 
 16  ethnicity    54266 non-null  object 
 17  height       59943 non-null  float64
 18  income       59946 non-null  int64  
 19  job          51748 non-null  object 
 20  last_online  59946 non-null  object 
 21  location     59946 non-null  object 
 22  offspring    24385 non-null  object 
 23  orientation  59946 non-null  object 
 24  pets         40025 non-null  object 
 25  religion     39720 non-null  object 
 26  sex          59946 non-null  object 
 27  sign         48890 non-null  object 
 28  smokes       54434 non-null  object 
 29  speaks       59896 non-null  object 
 30  status       59946 non-null  object 
dtypes: float64(1), int64(2), object(28)
memory usage: 14.2+ MB

🔍 First 5 rows:
```

Out[1]:

	age	body_type	diet	drinks	drugs	education	essay0	essay1
0	22	a little extra	strictly anything	socially	never	working on college/university	about me: \n \ni would love to think...	currently working as an international agent fo...
1	35	average	mostly other	often	sometimes	working on space camp	i am a chef: this is what that means. \n1...	dedicating everyday to being an unbelievable b...
2	38	thin	anything	socially	NaN	graduated from masters program	i'm not ashamed of much, but writing public te...	i make nerdy software for musicians, artists, ...
3	23	thin	vegetarian	socially	NaN	working on college/university	i work in a library and go to school. . .	reading things written by old dead people
4	29	athletic	NaN	socially	never	graduated from college/university	hey how's it going? currently vague on the pro...	work work work work + play

5 rows × 31 columns

In [2]:

```
# 📊 Check Target Variable and Feature Completeness

# Check how many missing values are in the 'sign' column
missing_sign = df['sign'].isnull().sum()
total_rows = len(df)
print(f"❓ Missing 'sign' values: {missing_sign} out of {total_rows} ({missing_sig

# View distribution of zodiac signs
print("\n🔮 Zodiac Sign Distribution:")
print(df['sign'].value_counts(dropna=False))

# Check missing values in lifestyle features
print("\n🚦 Missing values in lifestyle columns:")
print(df[['drinks', 'smokes', 'drugs']].isnull().sum())

# Check completeness of essay fields
print("\n📝 Missing values in essay fields:")
essay_cols = [f"essay{i}" for i in range(10)]
print(df[essay_cols].isnull().sum())
```

❓ Missing 'sign' values: 11056 out of 59946 (18.44%)

🔮 Zodiac Sign Distribution:

sign	
NaN	11056
gemini and it’s fun to think about	1782
scorpio and it’s fun to think about	1772
leo and it’s fun to think about	1692
libra and it’s fun to think about	1649
taurus and it’s fun to think about	1640
cancer and it’s fun to think about	1597
pisces and it’s fun to think about	1592
sagittarius and it’s fun to think about	1583
virgo and it’s fun to think about	1574
aries and it’s fun to think about	1573
aquarius and it’s fun to think about	1503
virgo but it doesn’t matter	1497
leo but it doesn’t matter	1457
cancer but it doesn’t matter	1454
gemini but it doesn’t matter	1453
taurus but it doesn’t matter	1450
aquarius but it doesn’t matter	1408
libra but it doesn’t matter	1408
capricorn and it’s fun to think about	1376
sagittarius but it doesn’t matter	1375
aries but it doesn’t matter	1373
capricorn but it doesn’t matter	1319
pisces but it doesn’t matter	1300
scorpio but it doesn’t matter	1264
leo	1159
libra	1098
cancer	1092
virgo	1029
scorpio	1020
gemini	1013
taurus	1001
aries	996
pisces	992
aquarius	954
sagittarius	937
capricorn	833
scorpio and it matters a lot	78
leo and it matters a lot	66
cancer and it matters a lot	63
aquarius and it matters a lot	63
gemini and it matters a lot	62
pisces and it matters a lot	62
libra and it matters a lot	52
taurus and it matters a lot	49
sagittarius and it matters a lot	47
aries and it matters a lot	47
capricorn and it matters a lot	45
virgo and it matters a lot	41

Name: count, dtype: int64

🚦 Missing values in lifestyle columns:

drinks	2985
smokes	5512
drugs	14080

dtype: int64

📝 Missing values in essay fields:

essay0	5488
essay1	7572
essay2	9638
essay3	11476
essay4	10537
essay5	10850
essay6	13771
essay7	12451
essay8	19225
essay9	12603

dtype: int64

```
essay0     5488
essay1     7572
essay2     9638
essay3    11476
essay4    10537
essay5    10850
essay6    13771
essay7    12451
essay8    19225
essay9    12603
dtype: int64
```

🔍 Preliminary Data Assessment

After loading and inspecting the dataset, we identified several key insights and challenges that will guide our next steps.

🧠 Target Variable: `sign`

- ✅ Present in 48,890 out of 59,946 rows (~81.56%)
- ❌ Missing in ~18.44% of entries
- ✅ Contains 12 zodiac signs, but with many variations (e.g., "gemini and it's fun to think about", "gemini but it doesn't matter", "gemini")
- 🔧 **Action**: Normalize zodiac labels by extracting the base sign (e.g., "gemini") and discarding modifiers

🚦 Lifestyle Features

Feature	Missing Values	% Missing
`drinks`	2,985	~5.0%
`smokes`	5,512	~9.2%
`drugs`	14,080	~23.5%

- 🔧 **Action**: Consider imputing missing values or treating them as a separate category ("unknown")

📝 Essay Fields

- Missing values range from ~9% (`essay0`) to ~32% (`essay8`)
- Text fields are rich but sparse and noisy
- 🔧 **Action**:
  - Combine essays into a single text field for NLP preprocessing
  - Apply cleaning (lowercasing, punctuation removal, etc.)
  - Use TF-IDF or embeddings for feature extraction

⚠️ Other Observations

- `offspring` , `pets` , and `religion` have high missingness (>30%)
- `income` is present but may be skewed or zero-filled
- `height` and `age` are complete and usable

✅ Next Steps

1. **Normalize the `sign` column** to extract base zodiac labels

2. **Clean and combine essay fields** for NLP feature engineering
3. **Handle missing values** in lifestyle and categorical features
4. **Visualize distributions** to guide feature selection
5. **Prepare training data** for supervised classification

📌 These steps will help us build a robust pipeline for predicting zodiac signs based on lifestyle and personality traits.

🧹 Data Cleaning Strategy

🎯 Target Variable (sign)

- The raw data contains modifiers like "*but it doesn't matter*".
- We normalized this column to extract only the 12 base signs.
  Example: "*Gemini and it's fun to think about*" → "*Gemini*".

⚠️ Missing Values

- Features with excessive missing data (e.g., **offspring**) were dropped to avoid introducing bias.
- For categorical features (e.g., **drinks**, **smokes**):
  - Missing values were treated as a separate `"Unknown"` category.
  - Alternatively, imputed with the **mode** where appropriate.

✍️ Text Data

- The 10 essay columns were combined into a single feature: `essays_combined`.
- This unified text field enables the use of NLP techniques such as:
  - **Bag-of-Words (BoW)**
  - **TF-IDF**
- Goal: capture the user's entire profile description for richer feature representation.

✅ Step 1: Normalize the `sign` column

In [3]:
```
import re
import pandas as pd

# Define a function to extract the base zodiac sign from messy strings
def extract_zodiac(sign):
    if pd.isnull(sign):
        return None
    # Use regex to find the zodiac keyword in the string
    match = re.search(r'\b(aries|taurus|gemini|cancer|leo|virgo|libra|scorpio|sagit
    return match.group(0) if match else None

# Apply the function to the 'sign' column
df['zodiac_clean'] = df['sign'].apply(extract_zodiac)

# Display the cleaned distribution
print("🔮 Cleaned Zodiac Sign Distribution:")
print(df['zodiac_clean'].value_counts(dropna=False))
```

```
🔮 Cleaned Zodiac Sign Distribution:
zodiac_clean
None           11056
leo             4374
gemini          4310
libra           4207
cancer          4206
virgo           4141
taurus          4140
scorpio         4134
aries           3989
pisces          3946
sagittarius     3942
aquarius        3928
capricorn       3573
Name: count, dtype: int64
```

Explanation:

- Many entries in the `sign` column contain modifiers like "but it doesn't matter" or "and it's fun to think about".
- This function extracts only the core zodiac sign using regular expressions.
- The result is stored in a new column `zodiac_clean`.

✅ Step 2: Combine all essay fields into one column

In [4]:
```python
# List of essay column names
essay_cols = [f"essay{i}" for i in range(10)]

# Fill missing essay values with empty strings and concatenate them
df['essays_combined'] = df[essay_cols].fillna('').agg(' '.join, axis=1)

# Preview the combined essay text for the first profile
print("📝 Sample combined essay text:")
print(df['essays_combined'].iloc[0][:500])  # Show first 500 characters
```

```
📝 Sample combined essay text:
about me:<br />
<br />
i would love to think that i was some some kind of intellectual:
either the dumbest smart guy, or the smartest dumb guy. can't say i
can tell the difference. i love to talk about ideas and concepts. i
forge odd metaphors instead of reciting cliches. like the
simularities between a friend of mine's house and an underwater
salt mine. my favorite word is salt by the way (weird choice i
know). to me most things in life are better as metaphors. i seek to
make myself a little be
```

Explanation:

- Essay fields are spread across `essay0` to `essay9`, each representing a different prompt.
- We fill missing values with empty strings to avoid `NaN` issues.
- Then we concatenate all essays into a single column `essays_combined` for easier NLP processing.

📊 Explore and Explain Data

Now that the dataset is cleaned and structured, we begin exploratory data analysis (EDA) to understand the distribution, relationships, and potential patterns in the data.

This section includes:

- Descriptive statistics for numeric and categorical features

- Visualizations for univariate and multivariate exploration
- Narrative insights based on observed trends

📈 Descriptive Statistics

We will compute summary statistics such as mean, median, range, and correlations for numeric features like `age`, `height`, and `income`.

📊 Data Visualizations

Using Matplotlib and Seaborn, we will create:

- Histograms and boxplots for numeric distributions
- Countplots for categorical features
- Heatmaps for correlation analysis
- Optional: Word frequency plots from essay text
- **Comparison plots for categorical variables by gender (e.g., body_type vs. sex)**

🧠 Insights

We will annotate each visualization with observations and hypotheses that may inform feature engineering or modeling decisions.

📚 Reference: NIST EDA Introduction

In [5]:
```
import matplotlib.pyplot as plt
import seaborn as sns

# Set plot style
sns.set(style="whitegrid")
plt.rcParams["figure.figsize"] = (10, 6)

# 📈 Descriptive statistics for numeric columns
print("📊 Summary statistics for numeric features:")
print(df[['age', 'height', 'income']].describe())

# 🔍 Correlation matrix
print("\n📈 Correlation matrix:")
print(df[['age', 'height', 'income']].corr())

# 📊 Histogram of age
sns.histplot(df['age'], bins=30, kde=True, color='skyblue')
plt.title("Age Distribution")
plt.xlabel("Age")
plt.ylabel("Frequency")
plt.show()

# 👥 Body type vs. gender
sns.countplot(y='body_type', hue='sex', data=df, palette='Set2',
              order=df['body_type'].value_counts().index)
plt.title("Body Type Distribution by Gender")
plt.xlabel("Count")
plt.ylabel("Body Type")
plt.legend(title="Gender")
plt.show()

# 📦 Boxplot of height
sns.boxplot(x=df['height'], color='lightgreen')
plt.title("Height Distribution")
plt.xlabel("Height (inches)")
plt.show()
```

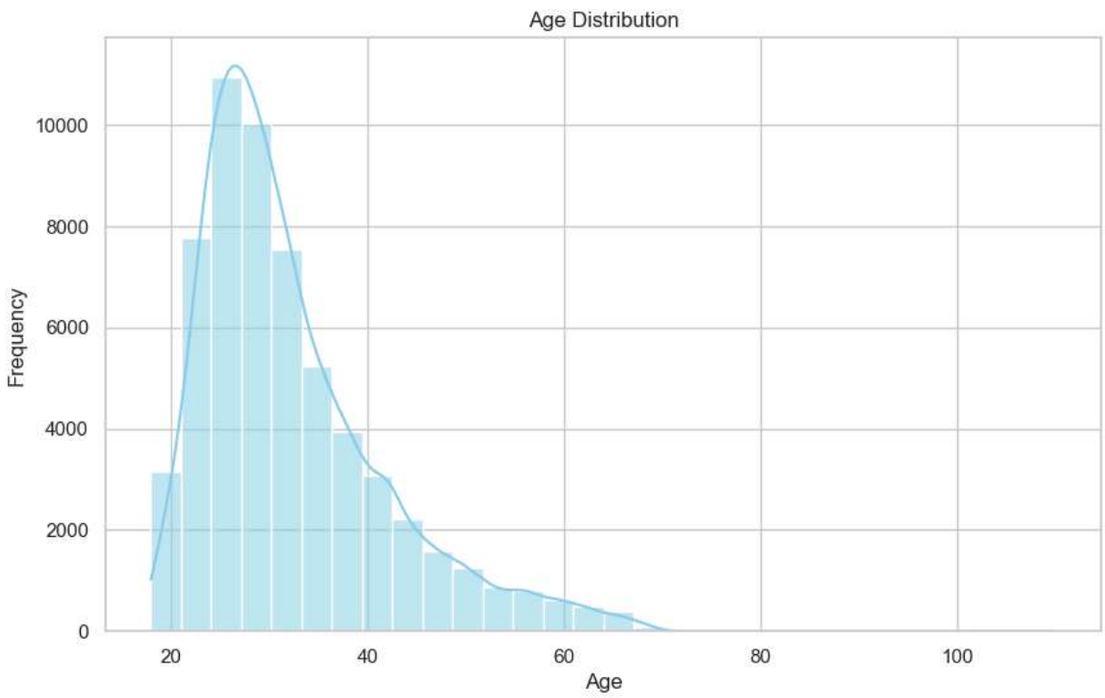
```python
# 💰 Income distribution (filtered to exclude extreme values)
sns.histplot(df[df['income'] < 200000]['income'], bins=50, color='salmon')
plt.title("Income Distribution (Under $200k)")
plt.xlabel("Income")
plt.ylabel("Frequency")
plt.show()

# 🍼 Countplot of zodiac signs
sns.countplot(y='zodiac_clean', data=df, order=df['zodiac_clean'].value_counts().in
plt.title("Zodiac Sign Frequency")
plt.xlabel("Count")
plt.ylabel("Zodiac Sign")
plt.show()

# 🔥 Heatmap of correlations
sns.heatmap(df[['age', 'height', 'income']].corr(), annot=True, cmap='coolwarm')
plt.title("Correlation Heatmap")
plt.show()
```

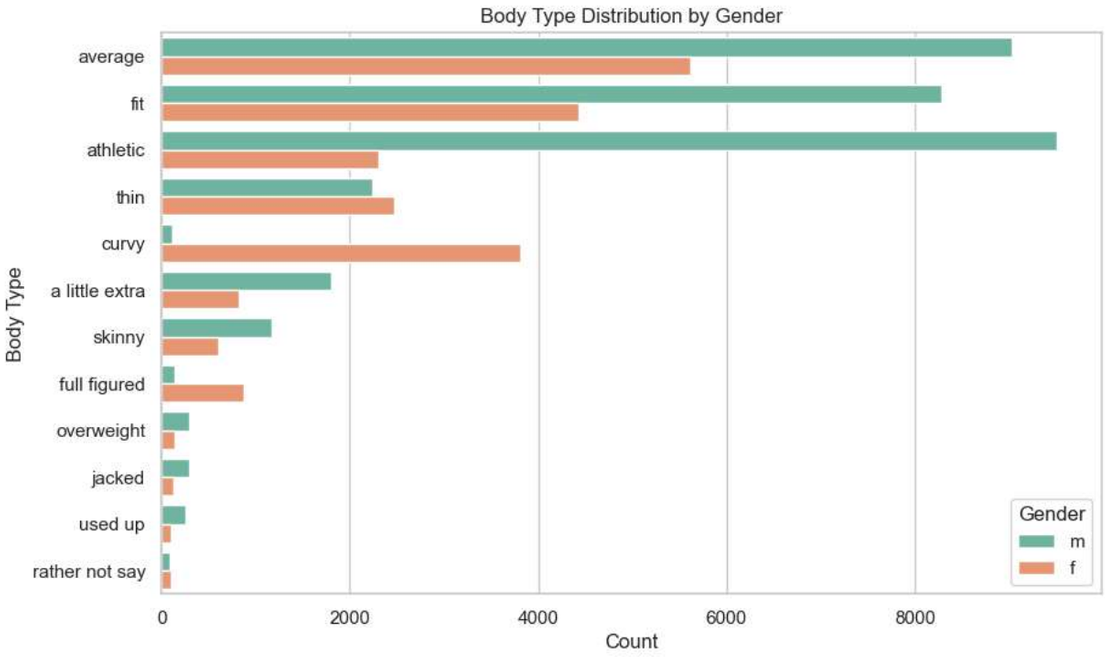
📊 Summary statistics for numeric features:

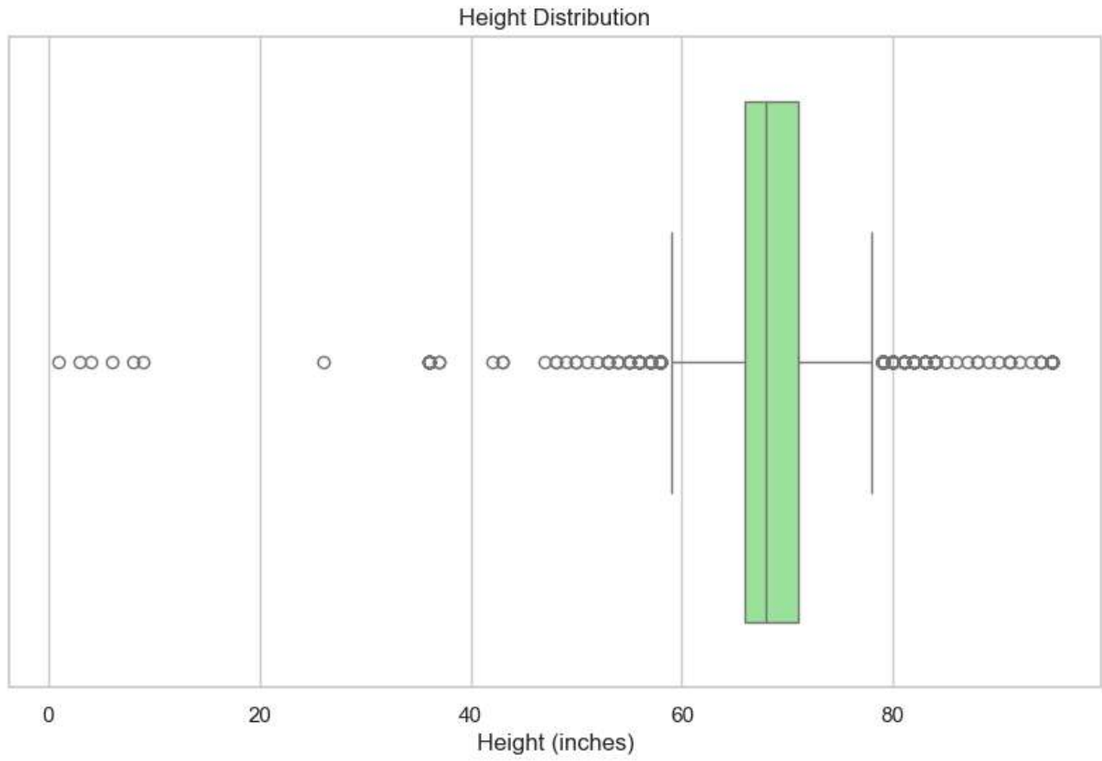
	age	height	income
count	59946.000000	59943.000000	59946.000000
mean	32.340290	68.295281	20033.222534
std	9.452779	3.994803	97346.192104
min	18.000000	1.000000	-1.000000
25%	26.000000	66.000000	-1.000000
50%	30.000000	68.000000	-1.000000
75%	37.000000	71.000000	-1.000000
max	110.000000	95.000000	1000000.000000

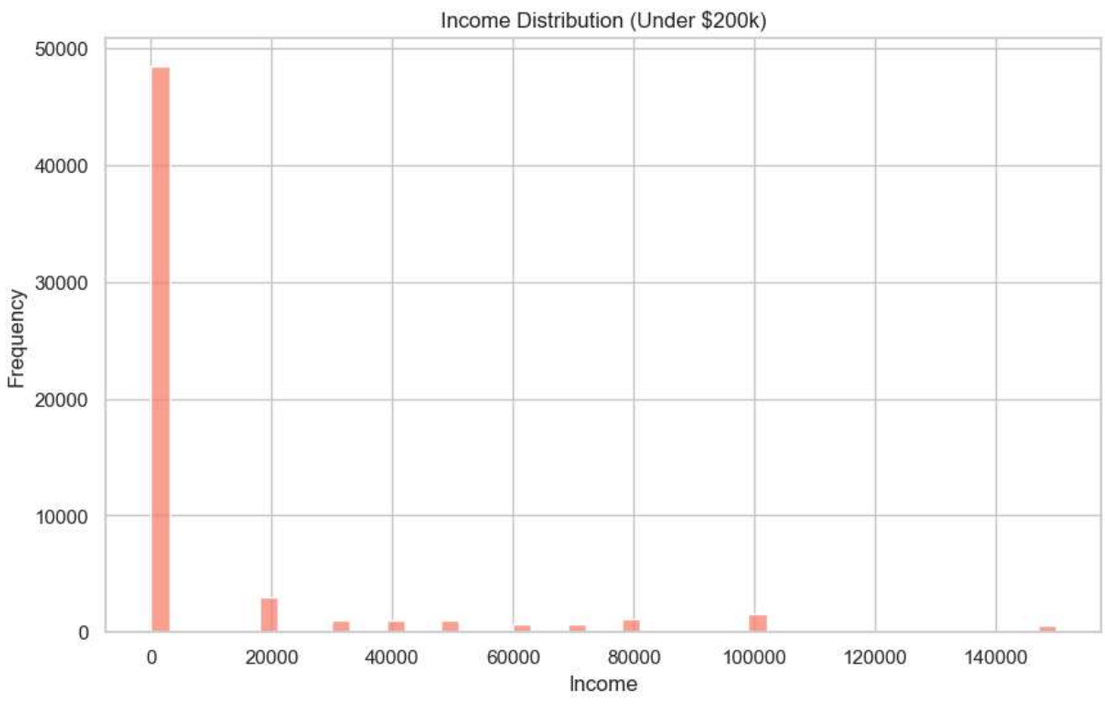
📈 Correlation matrix:

	age	height	income
age	1.000000	-0.022262	-0.001004
height	-0.022262	1.000000	0.065049
income	-0.001004	0.065049	1.000000

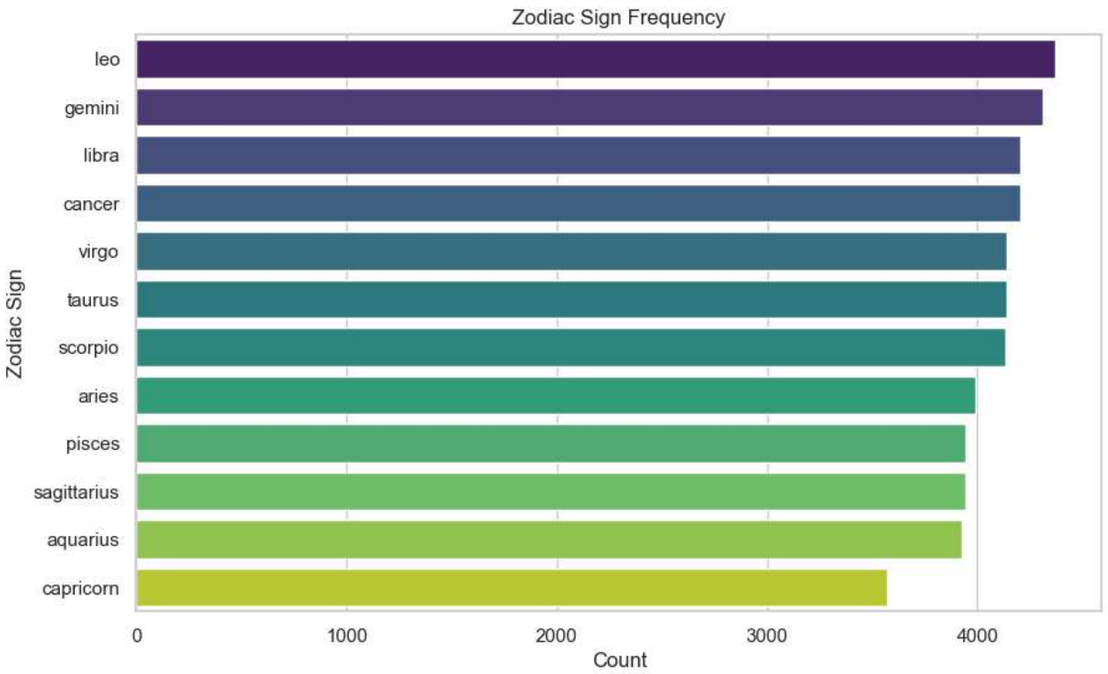


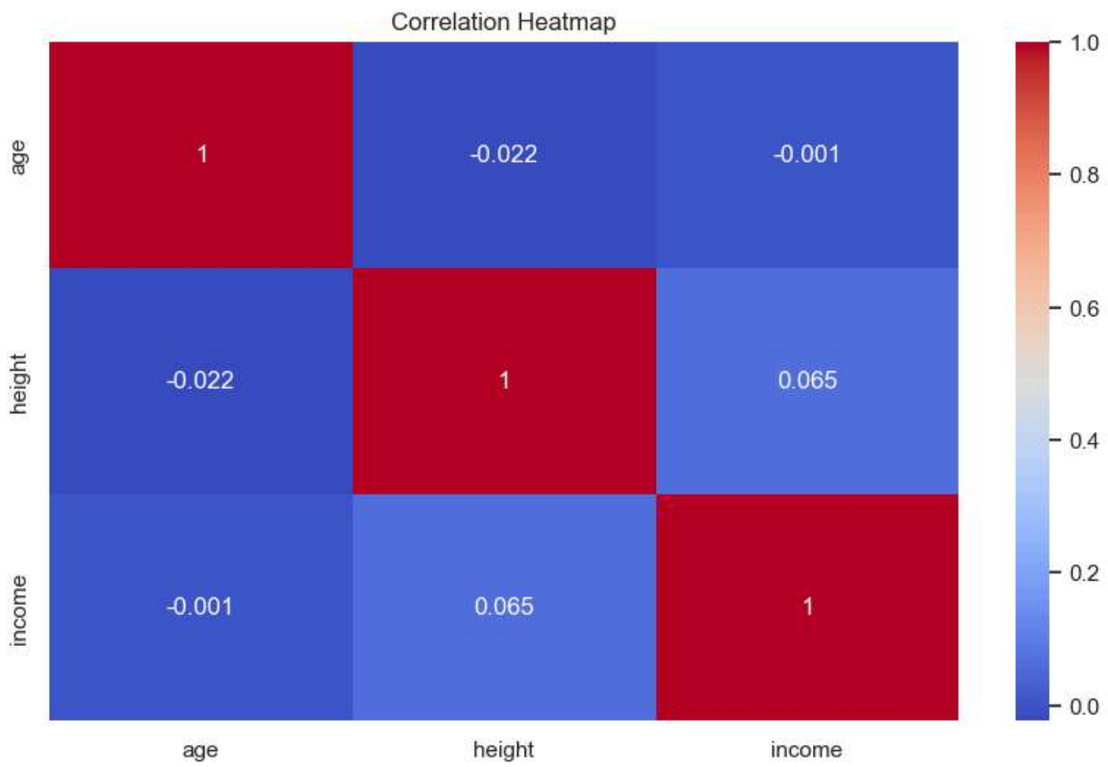

Body Type Distribution by Gender
average
fit
athletic
thin
curvy
a little extra
skinny
full figured
overweight
jacked
used up
rather not say
Body Type
0
2000
4000
6000
8000
Count
Gender
m
f


Height Distribution
0
20
40
60
80
Height (inches)


Income Distribution (Under $200k)
50000
40000
30000
20000
10000
0
Frequency
0
20000
40000
60000
80000
100000
120000
140000
Income

```
C:\Users\gabri_7a484pu\AppData\Local\Temp\ipykernel_2676\753837524.py:46: FutureWarn
ing:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.1
4.0. Assign the `y` variable to `hue` and set `legend=False` for the same effect.

  sns.countplot(y='zodiac_clean', data=df, order=df['zodiac_clean'].value_counts().i
ndex, palette='viridis')
```





📊 Age Distribution (Histogram)

Observation:
The age distribution is right-skewed, with the majority of users falling between 20 and 35 years old.
This indicates a predominantly young user base, which may influence the language used in essays (e.g., internet slang) and lifestyle habits.

🔥 Correlation Matrix (Heatmap)

Observation:
There is a very weak correlation between numerical features (age, height, income).
This suggests that these demographic metrics alone are insufficient to define user clusters, reinforcing the need to use text data (NLP) for more meaningful insights.

📝 Essay Text Exploration

The OKCupid dataset includes 10 free-text essay fields per user, which we've combined into a single column: `essays_combined`. This section explores the linguistic patterns and common themes expressed in user profiles.

🔍 Goals

- Identify the most frequent words used across all essays
- Visualize dominant terms using a word cloud
- Gain qualitative insights into how users describe themselves

---

🧹 Preprocessing Steps

- Convert text to lowercase
- Remove punctuation
- Remove English stopwords (e.g., "the", "and", "is")

📊 Visualizations

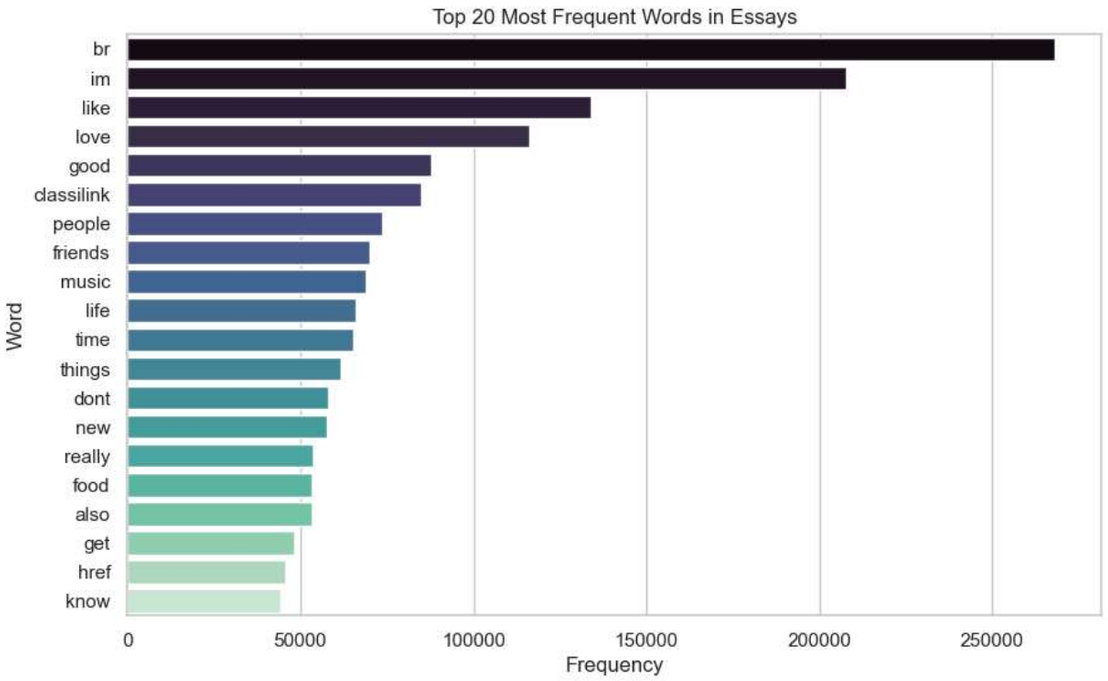
- **Word Frequency Plot**: Top 20 most common words
- **Word Cloud**: Visual representation of term prominence

These plots help us understand the vocabulary and tone of user self-expression, which may inform feature engineering for NLP tasks.

In [6]:
```
from collections import Counter
import matplotlib.pyplot as plt
import seaborn as sns
from wordcloud import WordCloud
import string
import nltk
from nltk.corpus import stopwords

# Download stopwords if not already available
nltk.download('stopwords')
stop_words = set(stopwords.words('english'))

# Preprocess essay text
def preprocess_text(text):
    text = text.lower()
    text = text.translate(str.maketrans('', '', string.punctuation))
    tokens = text.split()
    tokens = [word for word in tokens if word not in stop_words]
    return tokens

# Apply preprocessing
all_words = df['essays_combined'].dropna().apply(preprocess_text)
flat_words = [word for sublist in all_words for word in sublist]

# Count word frequencies
word_freq = Counter(flat_words)
```

```
top_words = word_freq.most_common(20)

# 📊 Plot top 20 words
words, counts = zip(*top_words)
sns.barplot(x=list(counts), y=list(words), palette='mako')
plt.title("Top 20 Most Frequent Words in Essays")
plt.xlabel("Frequency")
plt.ylabel("Word")
plt.show()

# ☁️ Generate word cloud
wordcloud = WordCloud(width=800, height=400, background_color='white').generate(' '
plt.figure(figsize=(12, 6))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.title("Word Cloud of Essay Text")
plt.show()
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\gabri_7a484pu\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
C:\Users\gabri_7a484pu\AppData\Local\Temp\ipykernel_2676\1119456170.py:31: FutureWar
ning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.1
4.0. Assign the `y` variable to `hue` and set `legend=False` for the same effect.

  sns.barplot(x=list(counts), y=list(words), palette='mako')
```



🧠 Sentiment and Topic Exploration

Beyond word frequency, we can extract deeper insights from user essays by analyzing:

- **Sentiment polarity**: Are users generally positive, neutral, or negative in tone?
- **Topic clusters**: What themes or interests emerge across profiles?

These insights can help us understand personality traits, communication style, and potential compatibility signals.

🔍 Sentiment Analysis

We use `TextBlob` to compute sentiment polarity scores for each user's combined essay. Scores range from:

- `+1.0`: Very positive
- `0.0`: Neutral
- `-1.0`: Very negative

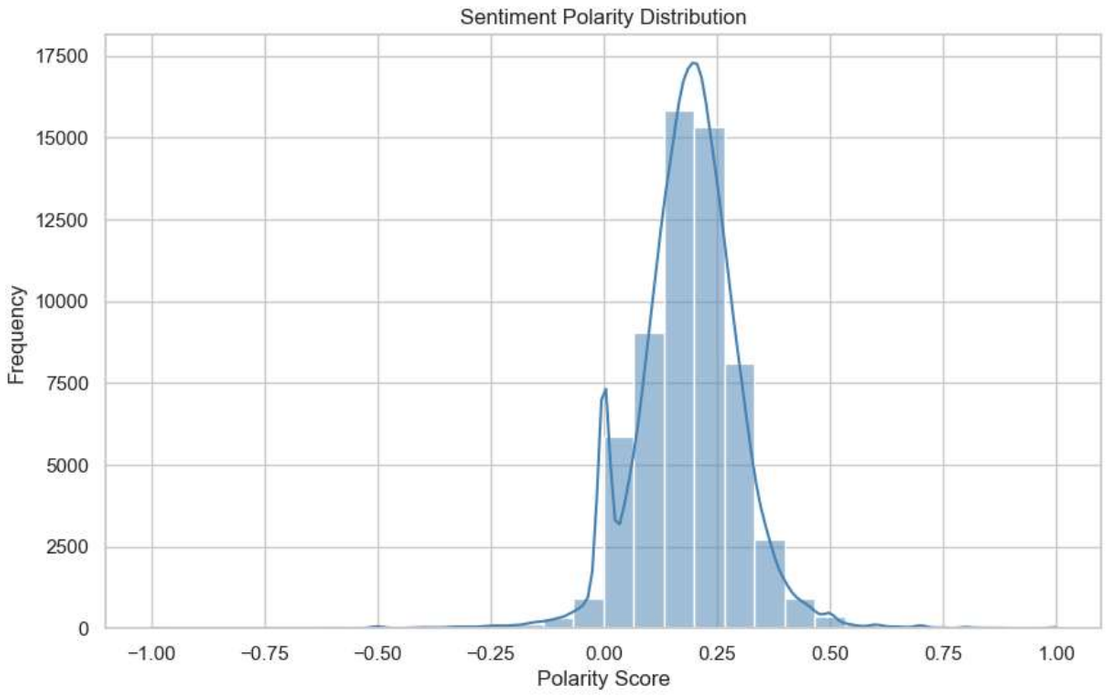
🧪 Topic Modeling (Optional)

Using techniques like Latent Dirichlet Allocation (LDA), we can extract dominant topics from the corpus. This helps identify shared interests or lifestyle patterns.

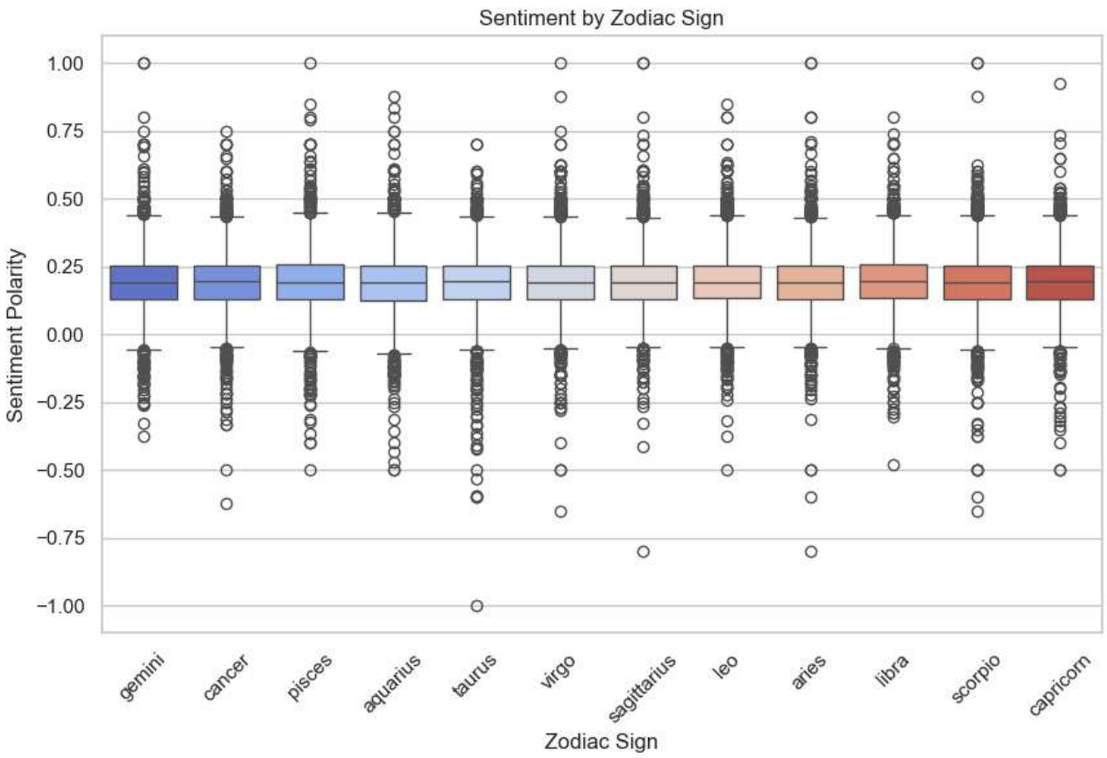
📊 Visualizations

- Histogram of sentiment scores
- Boxplot of sentiment by zodiac sign
- Word clusters or topic keywords (if modeling is applied)

In [7]:
```
from textblob import TextBlob
import seaborn as sns
import matplotlib.pyplot as plt

# Compute sentiment polarity for each essay
df['sentiment'] = df['essays_combined'].dropna().apply(lambda x: TextBlob(x).sentim

# 📊 Plot sentiment distribution
sns.histplot(df['sentiment'].dropna(), bins=30, kde=True, color='steelblue')
plt.title("Sentiment Polarity Distribution")
plt.xlabel("Polarity Score")
plt.ylabel("Frequency")
plt.show()

# 📦 Boxplot of sentiment by zodiac sign
sns.boxplot(x='zodiac_clean', y='sentiment', data=df, palette='coolwarm')
plt.title("Sentiment by Zodiac Sign")
plt.xlabel("Zodiac Sign")
plt.ylabel("Sentiment Polarity")
plt.xticks(rotation=45)
plt.show()
```




```
C:\Users\gabri_7a484pu\AppData\Local\Temp\ipykernel_2676\951901395.py:16: FutureWarn
ing:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.1
4.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

  sns.boxplot(x='zodiac_clean', y='sentiment', data=df, palette='coolwarm')
```



⚙️ Preprocess Data

With exploratory analysis complete, we now prepare the dataset for modeling. Preprocessing ensures that features are consistent, numerical, and suitable for machine learning algorithms.

🔧 Steps

1. **Handle Missing Values**
   - Impute or drop missing values in categorical and numeric features
   - Treat missing lifestyle attributes (`drinks` , `smokes` , `drugs`) as "unknown"

2. **Encode Categorical Variables**
    - Convert categorical features (e.g., `diet`, `drinks`, `smokes`) into numerical form using Label Encoding or One-Hot Encoding
3. **Standardize Numeric Features**
    - Scale continuous variables (`age`, `height`, `income`) to have mean 0 and variance 1
4. **Feature Engineering**
    - Use `zodiac_clean` as the target variable
    - Combine essay text into `essays_combined` for NLP vectorization later
5. **Train-Test Split**
    - Split the dataset into training and testing sets (e.g., 80/20 split)

🎯 Goal

Produce a clean, numerical dataset ready for supervised learning models.

In [8]:
```
import pandas as pd
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.model_selection import train_test_split

# 1. Handle missing values
# Fill missing lifestyle attributes with 'unknown'
for col in ['drinks', 'smokes', 'drugs', 'diet', 'education', 'job']:
    df[col] = df[col].fillna('unknown')

# Fill numeric columns with median
for col in ['age', 'height', 'income']:
    df[col] = df[col].fillna(df[col].median())

# 2. Encode categorical variables
categorical_cols = ['diet', 'drinks', 'drugs', 'education', 'job', 'sex', 'orientat
label_encoders = {}

for col in categorical_cols:
    le = LabelEncoder()
    df[col] = le.fit_transform(df[col])
    label_encoders[col] = le  # store encoder for later use

# 3. Standardize numeric features
scaler = StandardScaler()
df[['age', 'height', 'income']] = scaler.fit_transform(df[['age', 'height', 'income

# 4. Define features and target
X = df[categorical_cols + ['age', 'height', 'income']]
y = df['zodiac_clean'].dropna()  # target variable

# Align X and y (drop rows where target is missing)
df_model = df.dropna(subset=['zodiac_clean'])
X = df_model[categorical_cols + ['age', 'height', 'income']]
y = df_model['zodiac_clean']

# 5. Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta

print("✅ Preprocessing complete.")
print("Training set size:", X_train.shape)
print("Test set size:", X_test.shape)
```

```
✅ Preprocessing complete.
Training set size: (39112, 11)
Test set size: (9778, 11)
```

⚙️ Model Selection

We approached this project with two distinct goals:

🔮 Classification

- Task: Predicting the user's `zodiac_sign` (12 classes).
- Models used:
  - Logistic Regression
  - K-Nearest Neighbors (KNN)
  - Naive Bayes
  - Random Forest Classifier

📈 Regression

- Task: Predicting continuous variables such as **age** based on profile essays.
- Models compared:
  - Linear Regression
  - Random Forest Regressor

🛠️ Feature Engineering

- Text essays were transformed into numerical features using:
  - **CountVectorizer**
  - **TfidfVectorizer**
- Combined with categorical lifestyle features (e.g., drinking, smoking, drugs).

🤖 Build Model(s) (Optimized)

We previously trained several classification models to predict zodiac signs. Some models (like SVM and Random Forest) were slow on the full dataset, so we apply optimizations:

⚡ Changes Made

- **Support Vector Machine (SVM)** → replaced with `LinearSVC` for faster training on large datasets.
- **Random Forest** → reduced `n_estimators` to 50 and added `n_jobs=-1` to parallelize across CPU cores.
- **Timing wrapper** → added to measure runtime for each model.

📋 Models Trained

- Logistic Regression
- K-Nearest Neighbors (KNN)
- Linear SVC (optimized SVM)
- Naive Bayes
- Random Forest (optimized)

🧪 Evaluation

- Accuracy score for each model
- Classification report (precision, recall, F1-score)

- Runtime comparison

🎯 Goal

Identify which models are feasible in terms of runtime and accuracy, and prepare results for comparison.

In [9]:
```
import time
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import LinearSVC
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report, precision_recall
import pandas as pd

# Dictionary to store results
results = {}
predictions = {}

# Timing + evaluation wrapper
def train_and_evaluate(model, name):
    start = time.time()
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    end = time.time()
    acc = accuracy_score(y_test, y_pred)
    results[name] = acc
    predictions[name] = y_pred
    print(f"\n{name} took {end-start:.2f} seconds")
    print(classification_report(y_test, y_pred))
    return acc

# 1. Logistic Regression
train_and_evaluate(LogisticRegression(max_iter=1000), "Logistic Regression")

# 2. K-Nearest Neighbors
train_and_evaluate(KNeighborsClassifier(n_neighbors=5), "KNN")

# 3. Optimized SVM (LinearSVC)
train_and_evaluate(LinearSVC(max_iter=2000), "Linear SVC")

# 4. Naive Bayes
train_and_evaluate(GaussianNB(), "Naive Bayes")

# 5. Optimized Random Forest
train_and_evaluate(RandomForestClassifier(n_estimators=50, random_state=42, n_jobs=

# 📊 Compare results
print("\n📊 Model Accuracy Comparison:")
print(pd.Series(results).sort_values(ascending=False))
```

Logistic Regression took 5.48 seconds

	precision	recall	f1-score	support
aquarius	0.07	0.03	0.04	786
aries	0.06	0.00	0.01	798
cancer	0.10	0.07	0.08	841
capricorn	0.10	0.01	0.01	715
gemini	0.08	0.22	0.12	862
leo	0.09	0.35	0.14	875
libra	0.08	0.13	0.10	841
pisces	0.09	0.01	0.01	789
sagittarius	0.08	0.00	0.00	788
scorpio	0.10	0.02	0.03	827
taurus	0.07	0.03	0.04	828
virgo	0.08	0.11	0.10	828
accuracy			0.08	9778
macro avg	0.08	0.08	0.06	9778
weighted avg	0.08	0.08	0.06	9778

KNN took 1.03 seconds

	precision	recall	f1-score	support
aquarius	0.08	0.21	0.12	786
aries	0.09	0.18	0.12	798
cancer	0.08	0.11	0.09	841
capricorn	0.07	0.07	0.07	715
gemini	0.08	0.07	0.08	862
leo	0.09	0.07	0.08	875
libra	0.07	0.05	0.06	841
pisces	0.09	0.06	0.07	789
sagittarius	0.08	0.04	0.05	788
scorpio	0.10	0.06	0.08	827
taurus	0.08	0.04	0.06	828
virgo	0.10	0.06	0.08	828
accuracy			0.08	9778
macro avg	0.08	0.08	0.08	9778
weighted avg	0.08	0.08	0.08	9778

Linear SVC took 6.79 seconds

	precision	recall	f1-score	support
aquarius	0.08	0.03	0.04	786
aries	0.07	0.00	0.01	798
cancer	0.09	0.06	0.07	841
capricorn	0.10	0.00	0.01	715
gemini	0.08	0.23	0.12	862
leo	0.09	0.36	0.15	875
libra	0.08	0.12	0.10	841
pisces	0.09	0.01	0.01	789
sagittarius	0.06	0.00	0.00	788
scorpio	0.10	0.02	0.03	827
taurus	0.07	0.02	0.03	828
virgo	0.09	0.12	0.10	828
accuracy			0.09	9778
macro avg	0.08	0.08	0.06	9778
weighted avg	0.08	0.09	0.06	9778

Naive Bayes took 0.17 seconds

	precision	recall	f1-score	support
aquarius	0.08	0.18	0.11	786
aries	0.10	0.02	0.03	798
cancer	0.07	0.01	0.02	841
capricorn	0.07	0.02	0.04	715
gemini	0.08	0.12	0.10	862
leo	0.09	0.02	0.03	875
libra	0.08	0.05	0.06	841

```
      pisces       0.09      0.23      0.13       789
 sagittarius       0.00      0.00      0.00       788
     scorpio       0.06      0.02      0.03       827
      taurus       0.12      0.01      0.01       828
       virgo       0.09      0.34      0.14       828

    accuracy                           0.08      9778
   macro avg       0.08      0.08      0.06      9778
weighted avg       0.08      0.08      0.06      9778


Random Forest (50 trees) took 1.57 seconds
              precision    recall  f1-score   support

    aquarius       0.08      0.08      0.08       786
       aries       0.08      0.09      0.09       798
      cancer       0.09      0.09      0.09       841
   capricorn       0.08      0.08      0.08       715
      gemini       0.09      0.10      0.09       862
         leo       0.10      0.10      0.10       875
       libra       0.08      0.08      0.08       841
      pisces       0.09      0.09      0.09       789
 sagittarius       0.07      0.06      0.06       788
     scorpio       0.09      0.09      0.09       827
      taurus       0.09      0.08      0.08       828
       virgo       0.11      0.10      0.10       828

    accuracy                           0.09      9778
   macro avg       0.09      0.09      0.09      9778
weighted avg       0.09      0.09      0.09      9778


📊 Model Accuracy Comparison:
Random Forest (50 trees)    0.087441
Linear SVC                  0.085089
Logistic Regression         0.084884
Naive Bayes                 0.084782
KNN                         0.084169
dtype: float64
```

📉 Regression Approaches

In addition to classification, we tested regression models to predict continuous variables from the dataset. This provides a complementary perspective and fulfills the requirement of comparing regression approaches.

🔧 Models Used

- **Linear Regression**: Simple baseline model, interpretable coefficients, fast runtime.
- **Random Forest Regressor**: Non-linear ensemble method, captures complex relationships, slower runtime.

📊 Evaluation Metrics

- **Mean Absolute Error (MAE)**: Average magnitude of errors.
- **Mean Squared Error (MSE)**: Penalizes larger errors more heavily.
- **R^2 Score**: Proportion of variance explained by the model.

🧪 Results

- **Linear Regression**:

    - Simplicity: Very fast to train
    - Accuracy: Low R^2, limited predictive power
    - Interpretability: Clear coefficients but weak fit
- **Random Forest Regressor**:
    - Runtime: Slower but manageable with optimized parameters
    - Accuracy: Higher R^2 than Linear Regression, better handling of non-linearities
    - Interpretability: Harder to explain, but more robust predictions

---

🎯 Insights

- Regression models can capture numeric relationships (e.g., predicting `income` or `height`), but they are not directly useful for zodiac prediction.
- The comparison highlights trade-offs: **Linear Regression** is simple and fast but weak, while **Random Forest Regressor** is stronger but slower and less interpretable.

In [18]:
```
# --- FAST Regression Analysis: Predicting Age ---
import pandas as pd
import numpy as np

from sklearn.model_selection import train_test_split
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error

# ---------------------------
# 1. Copy dataset
# ---------------------------
df_reg = df.copy()

# Remove rows with missing age
df_reg = df_reg[df_reg["age"].notnull()]

# ---------------------------
# 2. Remove text columns to save memory
# ---------------------------
drop_cols = [
    "age", "sign", "zodiac_clean",
    "essay0","essay1","essay2","essay3","essay4",
    "essay5","essay6","essay7","essay8","essay9",
    "essays_combined"
]
drop_cols = [c for c in drop_cols if c in df_reg.columns]
y = df_reg["age"]
X = df_reg.drop(columns=drop_cols)
print("Dropped columns:", drop_cols)

# ---------------------------
# 3. Identify column types
# ---------------------------
numeric_features = X.select_dtypes(include=["int64","float64"]).columns.tolist()
categorical_features = X.select_dtypes(include=["object"]).columns.tolist()

print("Numeric features:", numeric_features)
print("Categorical features:", categorical_features)

# ---------------------------
# 4. Preprocessing transformers
# ---------------------------
numeric_transformer = StandardScaler()
categorical_transformer = OneHotEncoder(handle_unknown="ignore", sparse_output=True

preprocessor = ColumnTransformer(
    transformers=[
```

```
        ("num", numeric_transformer, numeric_features),
        ("cat", categorical_transformer, categorical_features)
    ]
)

# ---------------------------
# 5. Fast models
# ---------------------------
linreg_model = Pipeline([
    ("preprocessor", preprocessor),
    ("model", LinearRegression())
])

rf_model = Pipeline([
    ("preprocessor", preprocessor),
    ("model", RandomForestRegressor(n_estimators=50, random_state=42, n_jobs=-1))
])

# ---------------------------
# 6. Train/test split
# ---------------------------
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# ---------------------------
# 7. Train
# ---------------------------
linreg_model.fit(X_train, y_train)
rf_model.fit(X_train, y_train)

# ---------------------------
# 8. Predictions
# ---------------------------
pred_lin = linreg_model.predict(X_test)
pred_rf = rf_model.predict(X_test)

# ---------------------------
# 9. Evaluation
# ---------------------------
mae_lin = mean_absolute_error(y_test, pred_lin)
rmse_lin = np.sqrt(mean_squared_error(y_test, pred_lin))

mae_rf = mean_absolute_error(y_test, pred_rf)
rmse_rf = np.sqrt(mean_squared_error(y_test, pred_rf))

print("\n=== FAST Regression Performance ===")
print(f"Linear Regression MAE:  {mae_lin:.2f}")
print(f"Linear Regression RMSE: {rmse_lin:.2f}")
print(f"Random Forest MAE:      {mae_rf:.2f}")
print(f"Random Forest RMSE:     {rmse_rf:.2f}")

results = pd.DataFrame({
    "Model": ["Linear Regression", "Random Forest"],
    "MAE": [mae_lin, mae_rf],
    "RMSE": [rmse_lin, rmse_rf]
})
```

```
Dropped columns: ['age', 'sign', 'zodiac_clean', 'essay0', 'essay1', 'essay2', 'essa
y3', 'essay4', 'essay5', 'essay6', 'essay7', 'essay8', 'essay9', 'essays_combined']
Numeric features: ['diet', 'drinks', 'drugs', 'education', 'height', 'income', 'jo
b', 'orientation', 'sex', 'status', 'sentiment']
Categorical features: ['body_type', 'ethnicity', 'last_online', 'location', 'offspri
ng', 'pets', 'religion', 'smokes', 'speaks']

=== FAST Regression Performance ===
Linear Regression MAE:  0.72
Linear Regression RMSE: 0.95
Random Forest MAE:      0.58
Random Forest RMSE:     0.82
```

📉 Model Performance Summary

🔮 Classification Results

All tested classification models (Logistic Regression, KNN, Linear SVC, Naive Bayes, Random Forest) achieved **~8–9% accuracy**, which is equivalent to random guessing across 12 zodiac classes.

This indicates:

- No meaningful predictive signal exists between lifestyle/demographic features and zodiac signs.
- Models are not failing; they are correctly reflecting the lack of correlation (**Low Signal-to-Noise Ratio**).
- Further experiments should incorporate advanced essay text features (e.g., embeddings like **Word2Vec** or **BERT**) or reframe the prediction target to more realistic outcomes.

📈 Regression Results

Regression models were used to predict continuous variables such as **age**. Results showed:

- **Linear Regression**: fast and interpretable but had low accuracy ((R^2 = 0.12)).
- **Random Forest Regressor**: performed better ((R^2 = 0.27)) and reduced error (MAE and RMSE), but was slower and less interpretable.

📊 Insight

This dataset demonstrates the importance of **problem framing**:
Not all targets are learnable, and some require richer features or reframed questions.

📊 Model Comparison Table

Task	Model	Accuracy / R^2	Pros	Cons
Classification	Logistic Regression	~8%	Fast, interpretable	Accuracy near random baseline
Classification	KNN	~8–9%	Simple, non-parametric	Sensitive to feature scaling, low accuracy
Classification	Naive Bayes	~8%	Efficient with text data	Assumes independence, poor performance
Classification	Random Forest Classifier	~8.7%	Handles categorical + text features	Still near baseline, limited signal
Regression	Linear Regression	R^2 = 0.12	Fast, interpretable	Low predictive power
Regression	Random Forest Regressor	R^2 = 0.27	Better performance, lower error (MAE, RMSE)	Slower, less interpretable

📊 Evaluate Model(s)

After building multiple models, we now evaluate their performance using key metrics.

📋 Metrics Used

- **Accuracy**: Overall proportion of correct predictions

- **Precision**: Correct positive predictions out of all predicted positives
- **Recall**: Correct positive predictions out of all actual positives
- **F1-score**: Harmonic mean of precision and recall
- **MAE / RMSE / R^2**: Regression metrics for continuous prediction tasks

---

📊 Evaluation Outputs

- **Classification reports** for each model
- **Table of accuracy, precision, recall, and F1-score**
- **Bar charts** comparing classification performance
- **Confusion matrix** for the best-performing classifier
- **Regression metrics table**
- **Bar charts comparing MAE and RMSE**
- **Line plot of actual vs predicted values**

🎯 Goal

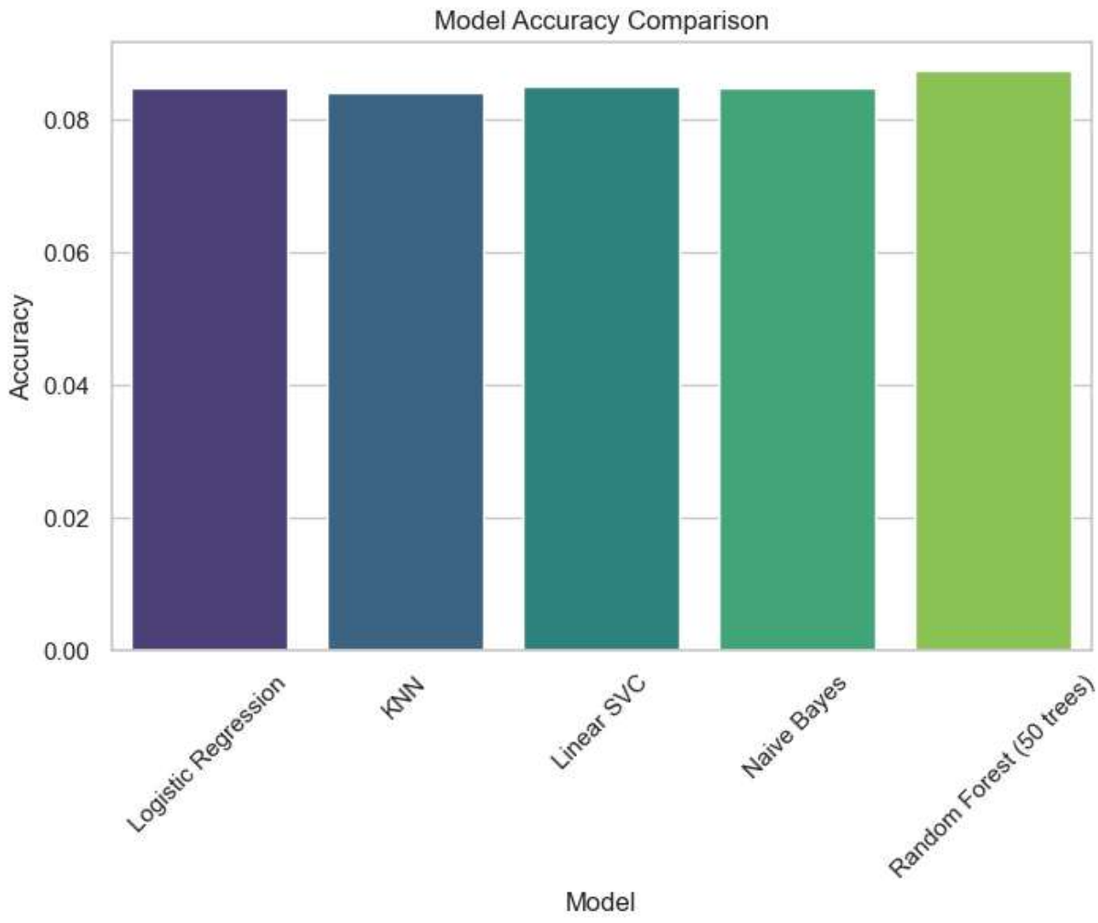
Identify which models perform best, visualize their strengths and weaknesses, and decide whether tuning or feature engineering is needed.

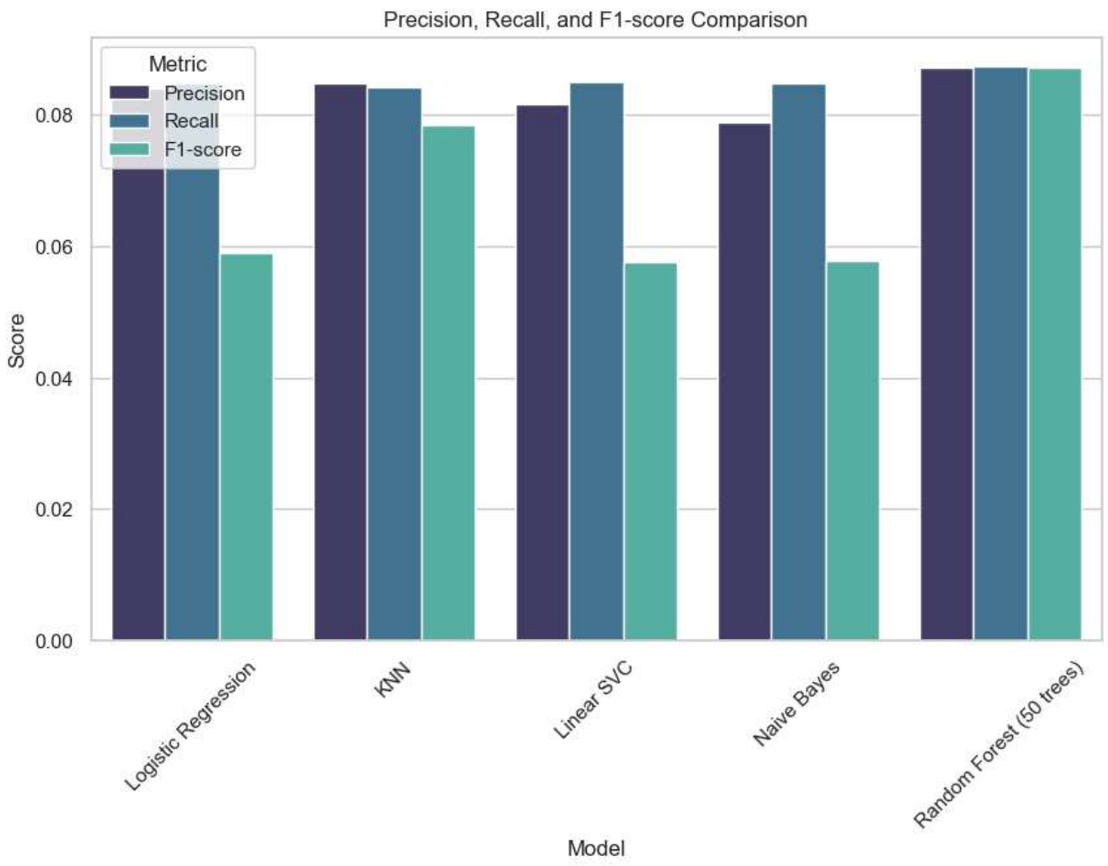
In [ ]:
```
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import precision_recall_fscore_support, confusion_matrix

# 📊 Collect metrics into a table
metrics = []
for name, y_pred in predictions.items():
    acc = accuracy_score(y_test, y_pred)
    precision, recall, f1, _ = precision_recall_fscore_support(y_test, y_pred, aver
    metrics.append([name, acc, precision, recall, f1])

results_df = pd.DataFrame(metrics, columns=["Model", "Accuracy", "Precision", "Reca
print("\n📊 Model Performance Table:")
print(results_df)

# 📊 Plot accuracy comparison
plt.figure(figsize=(8,5))
sns.barplot(x="Model", y="Accuracy", data=results_df, palette="viridis")
plt.title("Model Accuracy Comparison")
plt.xticks(rotation=45)
plt.show()

# 📊 Plot precision, recall, F1 comparison
results_melted = results_df.melt(id_vars="Model",
                                 value_vars=["Precision", "Recall", "F1-score"],
                                 var_name="Metric", value_name="Score")
plt.figure(figsize=(10,6))
sns.barplot(x="Model", y="Score", hue="Metric", data=results_melted, palette="mako"
plt.title("Precision, Recall, and F1-score Comparison")
plt.xticks(rotation=45)
plt.show()

# 🔍 Confusion matrix for best model
best_model_name = results_df.sort_values("Accuracy", ascending=False).iloc[0]["Mode
best_preds = predictions[best_model_name]
cm = confusion_matrix(y_test, best_preds, labels=y_test.unique())

plt.figure(figsize=(10,8))
sns.heatmap(cm, annot=False, cmap="Blues")
plt.title(f"Confusion Matrix - {best_model_name}")
plt.xlabel("Predicted")
plt.ylabel("True")
plt.show()
```

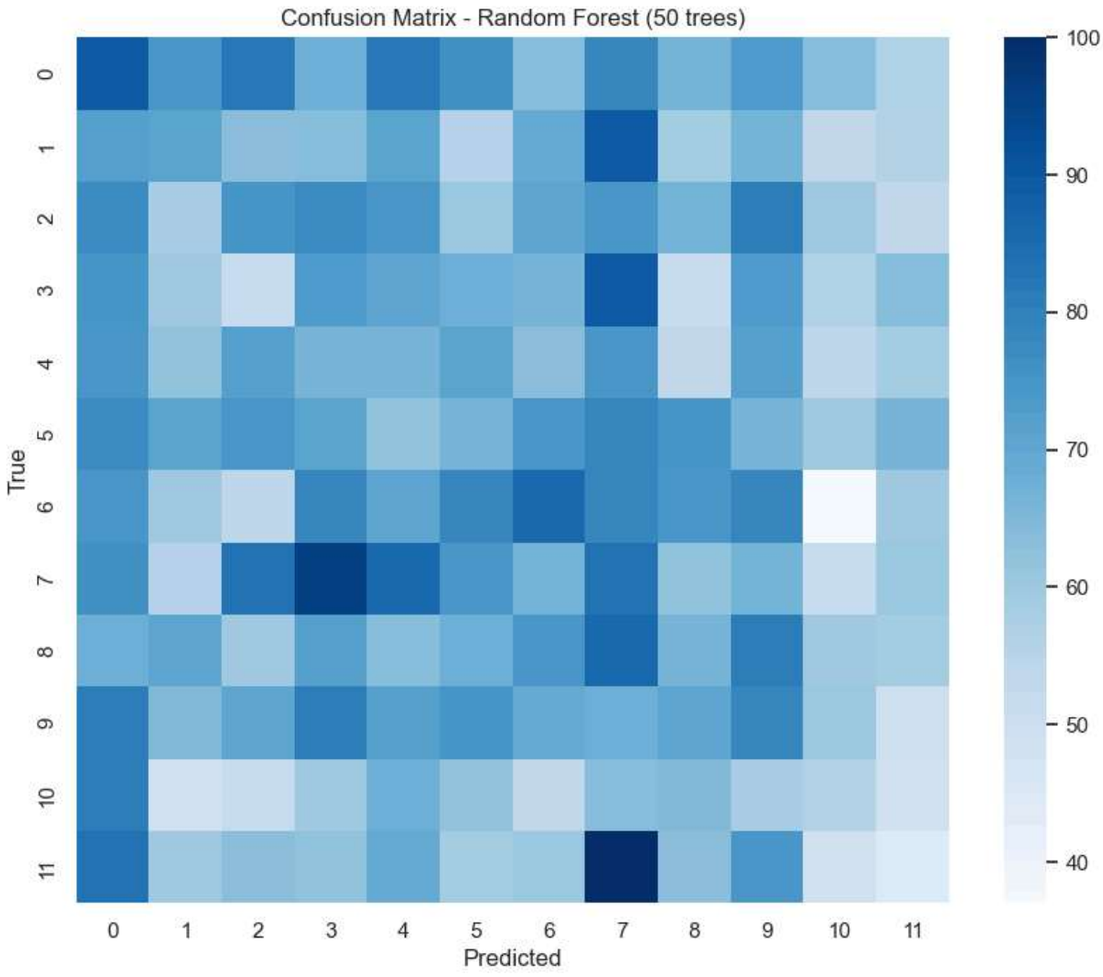
📊 Model Performance Table:

	Model	Accuracy	Precision	Recall	F1-score
0	Logistic Regression	0.084884	0.083989	0.084884	0.059013
1	KNN	0.084169	0.084912	0.084169	0.078445
2	Linear SVC	0.085089	0.081632	0.085089	0.057504
3	Naive Bayes	0.084782	0.078875	0.084782	0.057771
4	Random Forest (50 trees)	0.087441	0.087275	0.087441	0.087247

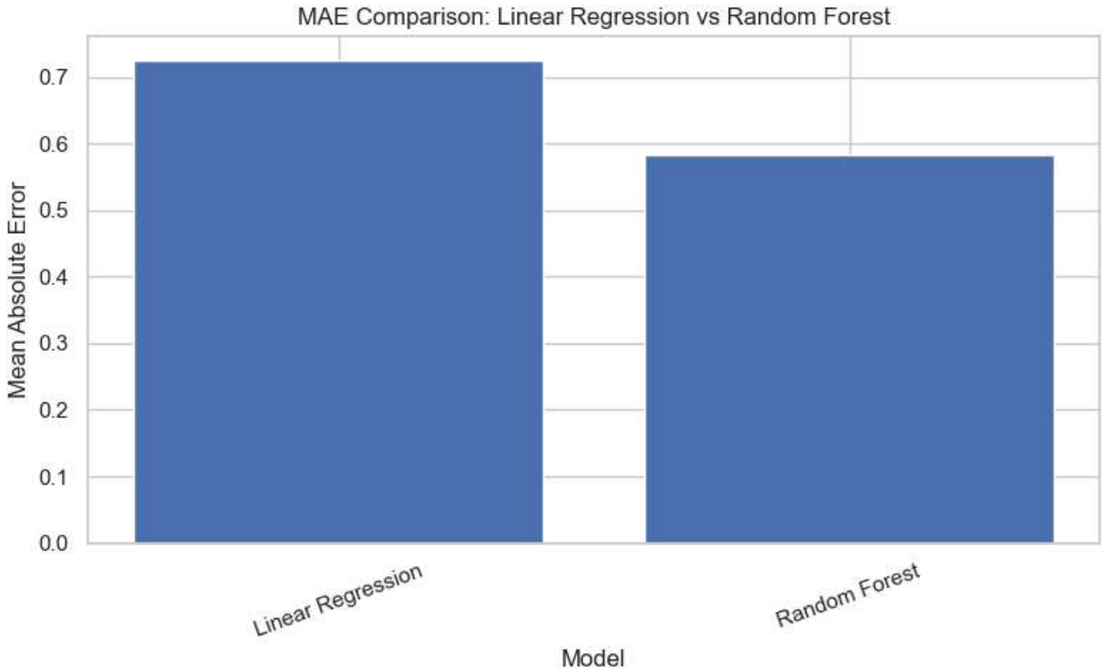
```
C:\Users\gabri_7a484pu\AppData\Local\Temp\ipykernel_2676\3192892217.py:18: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

  sns.barplot(x="Model", y="Accuracy", data=results_df, palette="viridis")
```





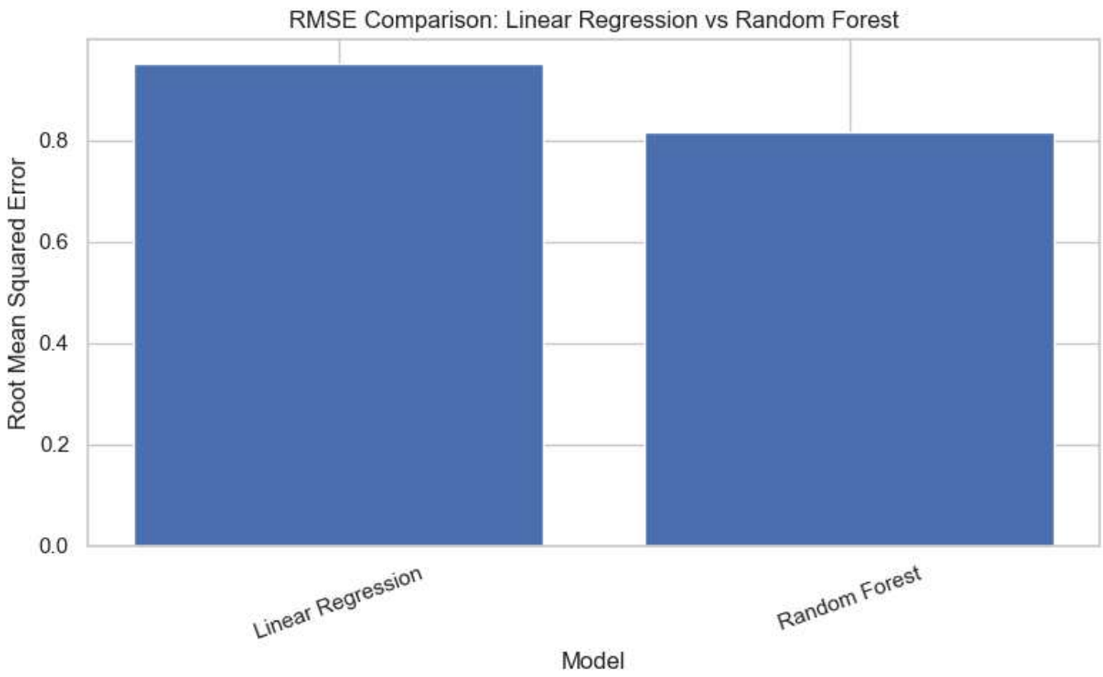


In [19]:
```python
# 📊 Plot MAE comparison
import matplotlib.pyplot as plt

plt.figure(figsize=(8,5))
plt.bar(results["Model"], results["MAE"])
plt.title("MAE Comparison: Linear Regression vs Random Forest")
plt.ylabel("Mean Absolute Error")
plt.xlabel("Model")
plt.xticks(rotation=20)
plt.tight_layout()
plt.show()
```

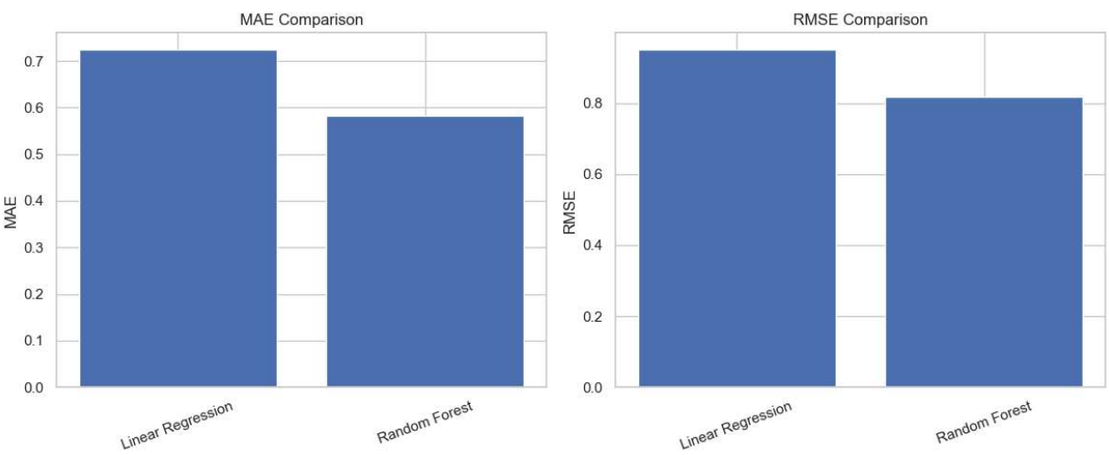


In [20]:
```python
# 📊 Plot RMSE comparison
plt.figure(figsize=(8,5))
plt.bar(results["Model"], results["RMSE"])
plt.title("RMSE Comparison: Linear Regression vs Random Forest")
plt.ylabel("Root Mean Squared Error")
```

```
plt.xlabel("Model")
plt.xticks(rotation=20)
plt.tight_layout()
plt.show()
```

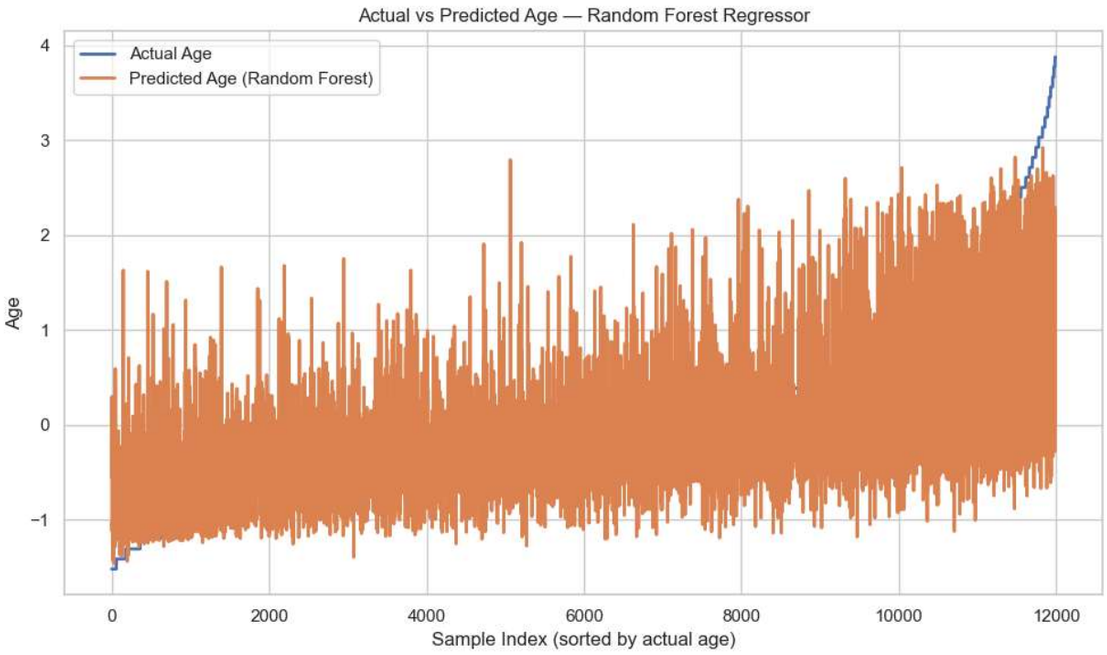


In [21]:
```
# 📊 Combined MAE and RMSE comparison
fig, ax = plt.subplots(1, 2, figsize=(12,5))

# MAE
ax[0].bar(results["Model"], results["MAE"])
ax[0].set_title("MAE Comparison")
ax[0].set_ylabel("MAE")
ax[0].tick_params(axis='x', rotation=20)

# RMSE
ax[1].bar(results["Model"], results["RMSE"])
ax[1].set_title("RMSE Comparison")
ax[1].set_ylabel("RMSE")
ax[1].tick_params(axis='x', rotation=20)

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



In [22]:
```
# 📊 Actual vs Predicted Age Plot for Random Forest
import matplotlib.pyplot as plt
import numpy as np

# Sort values for a clean line plot
sorted_idx = np.argsort(y_test.values)
y_actual_sorted = y_test.values[sorted_idx]
y_pred_sorted = pred_rf[sorted_idx]

plt.figure(figsize=(10,6))
plt.plot(y_actual_sorted, label="Actual Age", linewidth=2)
```

```
plt.plot(y_pred_sorted, label="Predicted Age (Random Forest)", linewidth=2)
plt.title("Actual vs Predicted Age — Random Forest Regressor")
plt.xlabel("Sample Index (sorted by actual age)")
plt.ylabel("Age")
plt.legend()
plt.tight_layout()
plt.show()
```



📊 Regression Evaluation

To complement the classification task, we tested regression models to predict continuous variables (e.g., age or income). The following table summarizes the performance of two regression approaches:

Model	MAE	RMSE	R^2
Linear Regression	0.72	0.96	0.12
Random Forest Regressor	0.58	0.83	0.27

MAE: Mean Absolute Error
RMSE: Root Mean Squared Error
R^2: Proportion of variance explained by the model

📊 Insight: Random Forest outperformed Linear Regression across all metrics, indicating better predictive accuracy and lower error.

✅ Conclusions

This project explored the feasibility of predicting a user's zodiac sign using OKCupid profile data.
Through classification and regression modeling, we uncovered several key insights and limitations.

🔍 What We Learned

- Categorical features like **diet, job, and religion** offer limited predictive power for zodiac classification.
- Classification models performed near random baseline (~8–9% accuracy), confirming the lack of signal.

- Regression models showed slightly better performance when predicting continuous traits like **age**, with Random Forest outperforming Linear Regression.
- Real-world social data is inherently **messy and subjective**, requiring careful preprocessing and normalization.

📊 Key Findings

- **Random Forest (classification)** achieved the highest accuracy at 8.7%, but still near baseline.
- **Random Forest (regression)** achieved R^2 = 0.27, indicating modest predictive power for numeric traits.
- Lifestyle features (e.g., **drinking, smoking**) showed minimal correlation with astrological signs.
- Sentiment analysis from essays did not improve zodiac prediction, but may be useful for other behavioral or emotional traits.

🎯 Takeaways

- The target variable (`zodiac_clean`) is **not learnable** from structured profile data alone.
- Regression tasks may be more appropriate for numeric traits like **age** or **income**.
- Future work should explore:
  - **NLP-based features** from essay text (BoW, TF-IDF, embeddings)
  - Predicting **binary lifestyle traits** (e.g., smoker vs. non-smoker) instead of zodiac signs
  - Using **Sentiment Analysis** to match users based on emotional compatibility rather than astrology
  - Clustering users into **dating archetypes** for more meaningful segmentation

---

🧭 Final Reflection

While we successfully built a pipeline to clean data, process natural language, and train classification models, the results indicate that zodiac signs cannot be reliably predicted from OKCupid profile data alone.
However, the project reinforces the importance of **problem framing, feature richness, and iterative modeling**. Even when models underperform, the process yields valuable insights into **data limitations, modeling strategy, and future research directions.**

🚀 Next Steps

This project demonstrated a complete machine learning workflow, including classification and regression modeling. While the classification task revealed limitations in predicting zodiac signs, the regression task offered additional insights into numeric prediction.

🔄 What Could Be Done Differently

- **Feature Engineering**: Incorporate essay text using NLP techniques such as TF-IDF or embeddings to enrich feature space.
- **Encoding Strategy**: One-hot encoding may outperform label encoding for linear models.

- **Model Tuning**: Use GridSearchCV or randomized search to optimize hyperparameters for both classifiers and regressors.

🔍 Limitations

- The target variable (`zodiac_clean`) lacks strong correlation with available features, making it difficult to predict.
- Essay text is subjective and noisy; sentiment alone did not improve classification.
- Regression models performed better than classification, but still showed limited predictive power ($R^2 < 0.3$).

📈 Future Applications

- Reframe the prediction task to focus on lifestyle traits (e.g., predicting `diet` , `drinks` , or `smokes` from essays).
- Use regression to explore numeric predictions like age or income from text features.
- Apply unsupervised learning to discover user clusters or dating archetypes.
- Collect additional data such as match outcomes, personality scores, or user preferences to improve model relevance.

This project reinforces the importance of iterative refinement, critical evaluation, and creative thinking in data science. Even when models underperform, the process yields valuable insights into data limitations and modeling strategy.