Bumble is a popular dating app where users connect based on mutual interests. This milestone focuses on analyzing Bumble's user data to identify trends and improve matchmaking. The analysis involves cleaning the data by handling missing values and outliers, processing it to create new features, and exploring demographics and lifestyle habits. Visualizations will be used to highlight key insights, aiming to optimize user experience and inform business decisions.

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```
df = pd.read_csv('bumble.csv')
df.head(5)
```

Out[2]:

	age	status	gender	body_type	diet	drinks	education	ethnicity	he
0	22	single	m	a little extra	strictly anything	socially	working on college/university	asian, white	
1	35	single	m	average	mostly other	often	working on space camp	white	
2	38	available	m	thin	anything	socially	graduated from masters program	NaN	
3	23	single	m	thin	vegetarian	socially	working on college/university	white	
4	29	single	m	athletic	NaN	socially	graduated from college/university	asian, black, other	

In [3]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 59946 entries, 0 to 59945
Data columns (total 17 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   age          59946 non-null  int64  
 1   status       59946 non-null  object 
 2   gender       59946 non-null  object 
 3   body_type    54650 non-null  object 
 4   diet         35551 non-null  object 
 5   drinks       56961 non-null  object 
 6   education    53318 non-null  object 
 7   ethnicity    54266 non-null  object 
 8   height       59943 non-null  float64
 9   income       59946 non-null  int64  
 10  job          51748 non-null  object 
 11  last_online  59946 non-null  object 
 12  location     59946 non-null  object 
 13  pets         40025 non-null  object 
 14  religion     39720 non-null  object 
 15  sign         48890 non-null  object 
 16  speaks       59896 non-null  object 
dtypes: float64(1), int64(2), object(14)
memory usage: 7.8+ MB
```

In [4]:
```
df.columns
```

Out[4]:
```
Index(['age', 'status', 'gender', 'body_type', 'diet', 'drinks', 'education',
       'ethnicity', 'height', 'income', 'job', 'last_online', 'location',
       'pets', 'religion', 'sign', 'speaks'],
      dtype='object')
```

In [5]:
```
df.index
```

Out[5]:
```
RangeIndex(start=0, stop=59946, step=1)
```

Part 1: Data Cleaning

1. Inspecting Missing Data

1.Which columns in the dataset have missing values, and what percentage of data is missing in each column?

2.Are there columns where more than 50% of the data is missing? Drop those columns where missing values are >50%.

3.Missing numerical data (e.g., height, income) should be handled by imputing the median value of height and income for the corresponding category, such as gender, age group, or location. This ensures that the imputed values are contextually relevant and reduce potential biases in the analysis.

1. Which columns in the dataset have missing values, and what percentage of data is missing in each column?

In [6]:
```
missing_values = df.isnull().sum()
missing_values
```

Out[6]:
```
age               0
status            0
gender            0
body_type      5296
diet          24395
drinks         2985
education      6628
ethnicity      5680
height            3
income            0
job            8198
last_online       0
location          0
pets          19921
religion      20226
sign          11056
speaks           50
dtype: int64
```

We used **isnull().sum()** to determine the total number of missing values in each column. From the results, we can see that all columns except age, status, gender, income, last_online, and location have missing values in the dataset.

In [7]:
```
missing_value_percentage = missing_values * 100 / len(df)
missing_value_percentage

missing_data = pd.DataFrame({ 'Missing Values': missing_values,'Percentage Missi
missing_data
```

Out[7]:

	Missing Values	Percentage Missing
age	0	0.000000
status	0	0.000000
gender	0	0.000000
body_type	5296	8.834618
diet	24395	40.694959
drinks	2985	4.979482
education	6628	11.056618
ethnicity	5680	9.475194
height	3	0.005005
income	0	0.000000
job	8198	13.675641
last_online	0	0.000000
location	0	0.000000
pets	19921	33.231575
religion	20226	33.740366
sign	11056	18.443266
speaks	50	0.083408

To calculate the percentage , I used the percentage formula - **missing_value_percentage = missing_values * 100 / len(df)**

2.Are there columns where more than 50% of the data is missing? Drop those columns where missing values are >50%.

Conclusion : From the above output, we can observe that no column has more than 50% missing data. Therefore, there is no need to drop any columns.

3.Missing numerical data (e.g., height, income) should be handled by imputing the median value of height and income for the corresponding category, such as gender, age group, or location. This ensures that the imputed values are contextually relevant and reduce potential biases in the analysis.

In [8]:
```
calculate_median = df.groupby(["gender"])["height"].transform("median")
df["height"] = df["height"].fillna(calculate_median)
print(calculate_median)
```

```
0        70.0
1        70.0
2        70.0
3        70.0
4        70.0
         ...
59941    65.0
59942    70.0
59943    70.0
59944    70.0
59945    70.0
Name: height, Length: 59946, dtype: float64
```

We used groupby, transform, and fillna to calculate and fill the missing values in the 'height' column with its median, as it is the only numerical column with missing data. The missing values in the 'height' column were replaced using the previously calculated median with fillna().

2.Data Types

Accurate data types are critical for meaningful analysis and visualization. For example, numeric fields like income or height must be stored as numbers for statistical computations, while dates like last_online must be converted to datetime format for time-based calculations.

1.Are there any inconsistencies in the data types across columns (e.g., numerical data stored as strings)?

2.Which columns require conversion to numerical data types for proper analysis (e.g., income)?

3.Does the last_online column need to be converted into a datetime format? What additional insights can be gained by analyzing this as a date field?

1.Are there any inconsistencies in the data types across columns (e.g., numerical data stored as strings)?

In [9]:
```
df.dtypes
```

Out[9]:
```
age              int64
status          object
gender          object
body_type       object
diet            object
drinks          object
education       object
ethnicity       object
height         float64
income           int64
job             object
last_online     object
location        object
pets            object
religion        object
sign            object
speaks          object
dtype: object
```

- Used **dtypes** to check data types of each column.
- There are no inconsistencies in the data types across columns , except for last_online which needed to be stored as datetime

2.Which columns require conversion to numerical data types for proper analysis (e.g., income)?

There are no columns that require conversion to numerical data type.

3.Does the last_online column need to be converted into a datetime format? What additional insights can be gained by analyzing this as a date field?

In [10]:
```
df["last_online"]= pd.to_datetime(df["last_online"] , format= "%Y-%m-%d-%H-%M")
df["last_online"]
```

Out[10]:
```
0       2012-06-28 20:30:00
1       2012-06-29 21:41:00
2       2012-06-27 09:10:00
3       2012-06-28 14:22:00
4       2012-06-27 21:26:00
               ...
59941   2012-06-12 21:47:00
59942   2012-06-29 11:01:00
59943   2012-06-27 23:37:00
59944   2012-06-23 13:01:00
59945   2012-06-29 00:42:00
Name: last_online, Length: 59946, dtype: datetime64[ns]
```

- The last_online column is in object type We need to convert it into datetime format.
- Here we use **to_datetime()** for data type conversion.

3.Outliers

Outliers are extreme values in the dataset that can distort averages, correlations, and overall trends. In the context of Bumble, an outlier in age (e.g., 110 years old) or income (e.g., $1,000,000 or -1) could represent errors or rare, valid cases.

1.Are there any apparent outliers in numerical columns such as age, height, or income? What are the ranges of values in these columns?

2.Any -1 values in numerical columns like income should be replaced with 0, as they may represent missing or invalid data.

3.For other outliers, rather than deleting them, calculate the mean and median values using only the middle 80% of the data (removing extreme high and low values). This approach ensures that outliers do not disproportionately impact the analysis while retaining as much meaningful data as possible.

1.Are there any apparent outliers in numerical columns such as age, height, or income? What are the ranges of values in these columns?

In [11]:
```
df.describe()
```

Out[11]:

	age	height	income	last_online
count	59946.000000	59946.000000	59946.000000	59946
mean	32.340290	68.295282	20033.222534	2012-05-22 06:43:35.300770560
min	18.000000	1.000000	-1.000000	2011-06-27 01:52:00
25%	26.000000	66.000000	-1.000000	2012-05-29 20:37:15
50%	30.000000	68.000000	-1.000000	2012-06-27 14:30:00
75%	37.000000	71.000000	-1.000000	2012-06-30 01:09:00
max	110.000000	95.000000	1000000.000000	2012-07-01 08:57:00
std	9.452779	3.994738	97346.192104	NaN

- Yes there are outliers in the numerical columns which are age, height and income.

In [12]:
```
age_range = (df['age'].min() , df['age'].max())
height_range = (df['height'].min() , df['height'].max())
income_range = (df['income'].min() , df['income'].max())

print(f"The age range is : {age_range} ")
print(f"The height range is : {height_range} ")
print(f"The income range is : {income_range} ")
```

```
The age range is : (18, 110)
The height range is : (1.0, 95.0)
The income range is : (-1, 1000000)
```

- To identify outliers,used **describe()** to check if they fall below the 25th percentile or above the 75th percentile.
- Here we used **max()** and **min()** to calculate the range, which provides the maximum and minimum values of the data, respectively.

2.Any -1 values in numerical columns like income should be replaced with 0, as they may represent missing or invalid data.

In [13]:
```
columns = df[["income", "age", "height"]]
columns.replace(-1, 0, inplace= True)
print(columns)
```

```
       income  age  height
0           0   22    75.0
1       80000   35    70.0
2           0   38    68.0
3       20000   23    71.0
4           0   29    66.0
...       ...  ...     ...
59941       0   59    62.0
59942       0   24    72.0
59943  100000   42    71.0
59944       0   27    73.0
59945       0   39    68.0

[59946 rows x 3 columns]
```

```
C:\Users\user\AppData\Local\Temp\ipykernel_11068\1453007559.py:2: SettingWithCopy
Warning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stabl
e/user_guide/indexing.html#returning-a-view-versus-a-copy
  columns.replace(-1, 0, inplace= True)
```

- Here we Used **replace()** to replace the -1 values with 0 in these columns.

3.For other outliers, rather than deleting them, calculate the mean and median values using only the middle 80% of the data (removing extreme high and low values). This approach ensures that outliers do not disproportionately impact the analysis while retaining as much meaningful data as possible.

In [14]:
```
column_names = ["income", "age", "height"]
for columns in column_names:
    Q1 = df[columns].quantile(0.10)
    Q3 = df[columns].quantile(0.90)
    filtered_values = df[(df[columns] > Q1) & (df[columns] < Q3)]
    mean_values = filtered_values[columns].mean()
    median_values = filtered_values[columns].median()
    print(f"\n {columns}: filtered_median: {median_values}, filtered_mean: {mean
```

```
income: filtered_median: 20000.0, filtered_mean: 26109.89010989011

age: filtered_median: 30.0, filtered_mean: 31.357685563997663

height: filtered_median: 68.0, filtered_mean: 68.25442646465552
```

- Here we used **quantile(0.10)** and **quantile(0.90)** to find middele 80% values to find mean() and median()

4.Missing Data Visualization

Visualizing missing data helps identify patterns of incompleteness in the dataset, which can guide data cleaning strategies. Understanding which columns have high levels of missing data ensures decisions about imputation or removal are well-informed.

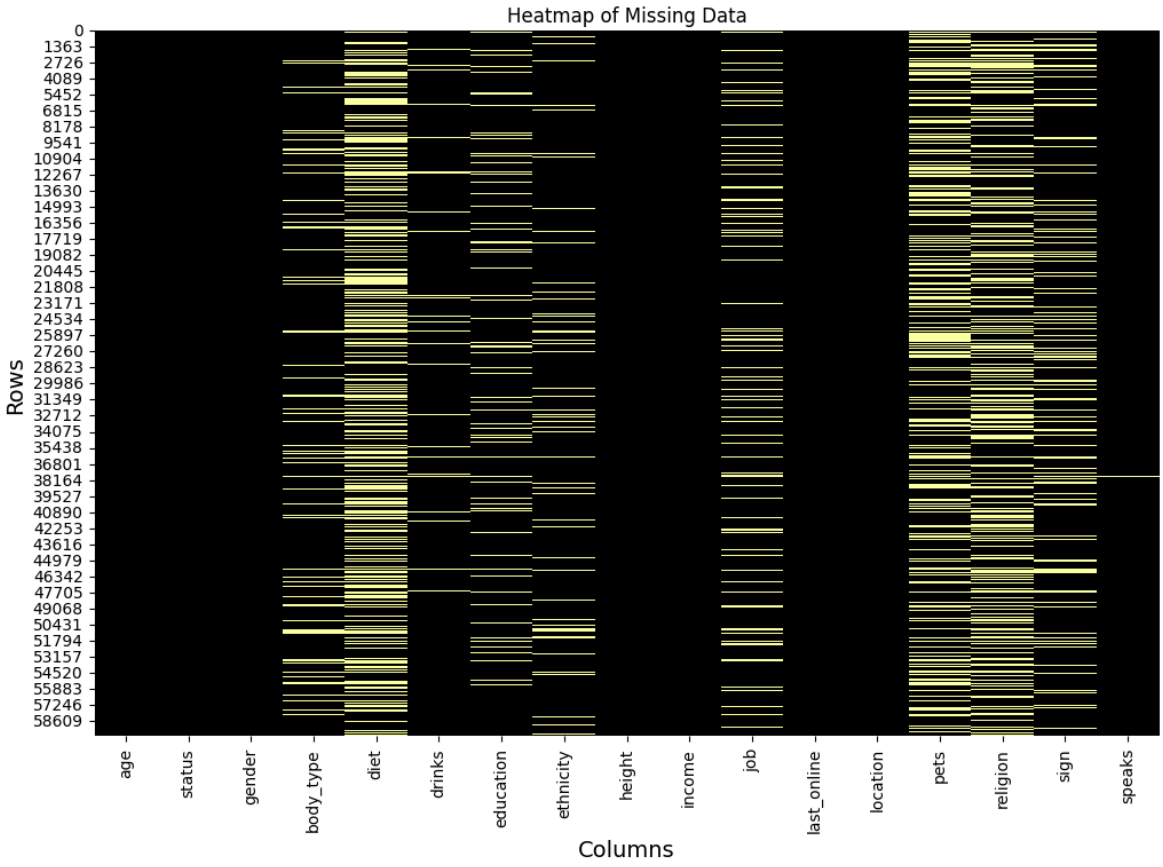
1.Create a heatmap to visualize missing values across the dataset. Which columns show consistent missing data patterns?

1.Create a heatmap to visualize missing values across the dataset. Which columns show consistent missing data patterns?

In [15]:
```
plt.figure(figsize=(12, 8))

sns.heatmap(df.isnull(), cbar = False, cmap = "inferno")
plt.title('Heatmap of Missing Data')
plt.xlabel('Columns', fontsize=14)
plt.ylabel('Rows', fontsize=14)
```

Out[15]:
```
Text(120.72222222222221, 0.5, 'Rows')
```



- Here we created the heatmap to visualize missing values across the dataset, We used seaborn library for heatmap and used isnull function
- The columns diet, religion and pets shows consistent missing data patterns.

Part 2: Data Processing

1. Binning and Grouping

Grouping continuous variables, such as age or income, into bins helps simplify analysis and identify trends among specific groups. For instance, grouping users into age ranges can reveal distinct patterns in behavior or preferences across demographics.

1.Bin the age column into categories such as "18-25", "26-35", "36-45", and "46+" to create a new column, age_group. How does the distribution of users vary across these age ranges?

2.Group income into categories like "Low Income," "Medium Income," and "High Income" based on meaningful thresholds (e.g., quartiles). What insights can be derived from these groups?

1.Bin the age column into categories such as "18-25", "26-35", "36-45", and "46+" to create a new column, age_group. How does the distribution of users vary across these age ranges?

In [16]:
```
bins = [18,25, 35, 45, 110]
label = ["18-25", "26-35", "36-45","46+"]
df["age_group"] = pd.cut(df["age"], bins, labels= label)
df["age_group"]
```

Out[16]:
```
0        18-25
1        26-35
2        36-45
3        18-25
4        26-35
         ...
59941      46+
59942    18-25
59943    36-45
59944    26-35
59945    36-45
Name: age_group, Length: 59946, dtype: category
Categories (4, object): ['18-25' < '26-35' < '36-45' < '46+']
```

In [17]:
```
df[['age','age_group']].head(10)
```

Out[17]:

	age	age_group
0	22	18-25
1	35	26-35
2	38	36-45
3	23	18-25
4	29	26-35
5	29	26-35
6	32	26-35
7	31	26-35
8	24	18-25
9	37	36-45

- Here we specify the bins and label for age
- Then use **cut()** function, this pd.cut() function is used to categorize the age column in the DataFrame into the defined bins.

2.Group income into categories like "Low Income," "Medium Income," and "High Income" based on meaningful thresholds (e.g., quartiles). What insights can be derived from these groups?

In [18]:
```
Q1 = df["income"].quantile(0.25)
Q2 = df["income"].quantile(0.50)
Q3 = df["income"].quantile(0.90)

df['income_category'] = np.where(df['income'] <= Q1,"Low Income",
                           np.where(df['income'] <=Q2,"Medium income","High incom

distribution_count = df['income_category'].value_counts()
distribution_count
```

Out[18]:
```
income_category
Low Income     48442
High income    11504
Name: count, dtype: int64
```

In [19]:
```
df[['income','income_category']]
```

Out[19]:

	income	income_category
0	-1	Low Income
1	80000	High income
2	-1	Low Income
3	20000	High income
4	-1	Low Income
...	...	...
59941	-1	Low Income
59942	-1	Low Income
59943	100000	High income
59944	-1	Low Income
59945	-1	Low Income

59946 rows × 2 columns

- Here we grouped income into categories as Low Income, Medium Income and High Income using 25th, 50th and 90th percentile.
- Create new column use np.where()

2. Derived Features

Derived features are new columns created based on the existing data to add depth to the analysis. These features often reveal hidden patterns or provide new dimensions to

explore.

1.Create a new feature, profile_completeness, by calculating the percentage of non-missing values for each user profile. How complete are most user profiles, and how does completeness vary across demographics?

1.Create a new feature, profile_completeness, by calculating the percentage of non-missing values for each user profile. How complete are most user profiles, and how does completeness vary across demographics?

In [20]:
```
df["profile_completeness"] = df.notnull().sum(axis=1) / df.shape[1] * 100
print(df["profile_completeness"])
print(df.groupby("status")["profile_completeness"].mean())
```

```
0        100.000000
1        100.000000
2         84.210526
3         94.736842
4         89.473684
             ...
59941     84.210526
59942    100.000000
59943     94.736842
59944    100.000000
59945     94.736842
Name: profile_completeness, Length: 59946, dtype: float64
status
available         91.364470
married           90.747029
seeing someone    91.115871
single            90.774803
unknown           84.210526
Name: profile_completeness, dtype: float64
```

- values in each user profile by using **notnull().sum()** divided by the total number of columns with **shape[]**, and multiplying by 100.
- The average profile_completeness was analyzed across different demographics by grouping the data by status.
- The analysis revealed that the 'unknown' category had lower profile completeness compared to other status categories.

3. Unit Conversion

Standardizing units across datasets is essential for consistency, especially when working with numerical data. In the context of the Bumble dataset, users' heights are given in inches, which may not be intuitive for all audiences.

1.Convert the height column from inches to centimeters using the conversion factor (1 inch = 2.54 cm). Store the converted values in a new column, height_cm.

In [21]:
```
df['height_cm'] = df['height'] *2.54
df['height_cm']
```

Out[21]:
```
0        190.50
1        177.80
2        172.72
3        180.34
4        167.64
          ...
59941    157.48
59942    182.88
59943    180.34
59944    185.42
59945    172.72
Name: height_cm, Length: 59946, dtype: float64
```

In [22]:
```
print(df[["height" , "height_cm"]])
```

```
       height  height_cm
0        75.0     190.50
1        70.0     177.80
2        68.0     172.72
3        71.0     180.34
4        66.0     167.64
...       ...        ...
59941    62.0     157.48
59942    72.0     182.88
59943    71.0     180.34
59944    73.0     185.42
59945    68.0     172.72

[59946 rows x 2 columns]
```

- Here we convert hight column from inches to centimeters
- We are multiplying the df['height'] by 2.54to convert to centimeter

Part 3: Data Analysis

1. Demographic Analysis

Understanding the demographics of users is essential for tailoring marketing strategies, improving user experience, and designing features that resonate with the platform's

audience. Insights into gender distribution, orientation, and relationship status can help Bumble refine its matchmaking algorithms and engagement campaigns.

1.What is the gender distribution (gender) across the platform? Are there any significant imbalances?

2.What are the proportions of users in different status categories (e.g., single, married, seeing someone)? What does this suggest about the platform's target audience?

3.How does status vary by gender? For example, what proportion of men and women identify as single?

1.What is the gender distribution (gender) across the platform? Are there any significant imbalances?

In [23]:
```
gender_count = df['gender'].value_counts()
print(f"Gender distribution (COUNT): \n{gender_count}")
```

```
Gender distribution (COUNT):
gender
m    35829
f    24117
Name: count, dtype: int64
```

- We can see that the number of male users are higher than the number of female users.

2.What are the proportions of users in different status categories (e.g., single, married, seeing someone)? What does this suggest about the platform's target audience?

In [24]:
```
status_count = df['status'].value_counts()
print(f"Status distribution (COUNT): \n{status_count}")
```

```
Status distribution (COUNT):
status
single            55697
seeing someone     2064
available          1865
married             310
unknown              10
Name: count, dtype: int64
```

In [25]:
```
status_distribution_percent = df['status'].value_counts()*100/len(df['status'])
print(f"Status distribution (PERCENTAGE): \n{status_distribution_percent}")
```

```
Status distribution (PERCENTAGE):
status
single            92.911954
seeing someone     3.443099
available          3.111133
married            0.517132
unknown            0.016682
Name: count, dtype: float64
```

- Here, we use the **value_counts()** function to count the number of users in each status category.

Analysis / Recommendations:

- The output shows that 92% of users are single, which is much higher than expected. *Since most users are single, marketing campaigns should focus on this group and highlight features for singles seeking meaningful connections, casual dating, or long-term relationships.

3.How does status vary by gender? For example, what proportion of men and women identify as single?

In [26]:
```
status_gender_distribution = df.groupby(["status", "gender"]).size() / len(df) *
status_gender_df = status_gender_distribution.reset_index(name="Percentage")
status_gender_df_sorted = status_gender_df.sort_values(by="Percentage", ascendin
print(status_gender_df_sorted)
```

```
          status gender  Percentage
7         single      m   55.680112
6         single      f   37.231842
1      available      m    2.016815
5  seeing someone      m    1.769926
4  seeing someone      f    1.673173
0      available      f    1.094318
3        married      m    0.291929
2        married      f    0.225203
9        unknown      m    0.010009
8        unknown      f    0.006673
```

- Calculate Proportions: Groups data by status and gender, then calculates the percentage using size() and len().
- Create DataFrame: Converts the result into a DataFrame with reset_index() and names the new column "Percentage."
- Sort and Display: Sorts the DataFrame by "Percentage" in descending order using sort_values() and prints the result.

Analysis-

- Single Users: The majority of users (55.68% male, 37.23% female) are "single," indicating Bumble's focus on relationships and connections.
- Available Users: A small percentage (2.02% male, 1.09% female) are "available," likely open to casual interactions or new connections.

Recommendations-

- Focus on Singles: Marketing should target singles, emphasizing features for meaningful connections, casual dates, and long-term relationships.
- Engage "Available" and "Seeing Someone" Users: Offer features for users open to casual dating or making new connections while in non-committed relationships.

2. Correlation Analysis

Correlation analysis helps uncover relationships between variables, guiding feature engineering and hypothesis generation. For example, understanding how age correlates with income or word count in profiles can reveal behavioral trends that inform platform design.

1.What are the correlations between numerical columns such as age, income, gender Are there any strong positive or negative relationships?

2.How does age correlate with income? Are older users more likely to report higher income levels?

1.What are the correlations between numerical columns such as age, income, gender Are there any strong positive or negative relationships?

In [27]:
```
arr = df[['age','income','height']]
arr.corr()
```

Out[27]:

	age	income	height
age	1.000000	-0.001004	-0.022253
income	-0.001004	1.000000	0.065048
height	-0.022253	0.065048	1.000000

- Select Columns: creates a subset arr from the DataFrame df containing the columns age, income, and height.
- Calculate Correlation: The corr() function calculates the correlation matrix between the selected columns to understand how they are related to each other.

Analysis -

- Weak Correlation: Age, income, and height show little to no correlation.
- Slight Positive Link: Income and height have a very weak positive correlation.

2.How does age correlate with income? Are older users more likely to report higher income levels?

In [28]:
```
correlation = df["age"].corr(df["income"])
print(correlation)
```

```
-0.0010038398754361308
```

- The correlation between age and income was calculated using .corr(), resulting in a very weak negative correlation of -0.00100.

3. Diet and Lifestyle Analysis

Lifestyle attributes such as diet, drinks provide insights into user habits and preferences. Analyzing these factors helps identify compatibility trends and inform product features like filters or match recommendations.

1.How do dietary preferences (diet) distribute across the platform? For example, what percentage of users identify as vegetarian, vegan, or follow "anything" diets?

2.How do drinking habits (drinks) vary across different diet categories? Are users with stricter diets (e.g., vegan) less likely to drink?

1.How do dietary preferences (diet) distribute across the platform? For example, what percentage of users identify as vegetarian, vegan, or follow "anything" diets?

In [29]:
```
diet_percentage = df['diet'].value_counts() * 100 / len(df['diet'])
diet_percentage
```

Out[29]:
```
diet
mostly anything        27.666567
anything               10.314283
strictly anything       8.529343
mostly vegetarian       5.745171
mostly other            1.679845
strictly vegetarian     1.459647
vegetarian              1.112668
strictly other          0.754012
mostly vegan            0.563841
other                   0.552164
strictly vegan          0.380342
vegan                   0.226871
mostly kosher           0.143462
mostly halal            0.080072
strictly halal          0.030027
strictly kosher         0.030027
halal                   0.018350
kosher                  0.018350
Name: count, dtype: float64
```

- The code calculates the percentage of each unique value in the 'diet' column using value_counts() and divides by the total number of entries in the column.

Analysis-

- 27% of users follow a "mostly anything" diet, while 10% prefer "anything."
- Only 5% of users are mostly vegetarian.

2.How do drinking habits (drinks) vary across different diet categories? Are users with stricter diets (e.g., vegan) less likely to drink?

In [30]:
```
drink_percentage_by_diet = df.groupby("diet")["drinks"].size() / len(df) * 100
drinks_by_diet_df = drink_percentage_by_diet.reset_index(name="Drink Percentage"
sorted_drinks_df = drinks_by_diet_df.sort_values(by="Drink Percentage", ascendin
print(sorted_drinks_df)
```

```
                  diet  Drink Percentage
3       mostly anything         27.666567
0              anything         10.314283
10    strictly anything          8.529343
8     mostly vegetarian          5.745171
6          mostly other          1.679845
15  strictly vegetarian          1.459647
17           vegetarian          1.112668
13       strictly other          0.754012
7          mostly vegan          0.563841
9                 other          0.552164
14       strictly vegan          0.380342
16                vegan          0.226871
5         mostly kosher          0.143462
4          mostly halal          0.080072
11       strictly halal          0.030027
12      strictly kosher          0.030027
1                halal          0.018350
2               kosher          0.018350
```

- The code uses groupby("diet") to group the data by diet type and then applies .size() to count the occurrences of "drinks" for each group.
- It calculates the percentage of drinkers for each diet by dividing the count by the total number of users (len(df)) and multiplying by 100.
- The result is stored in a DataFrame using reset_index(), and the DataFrame is sorted in descending order using sort_values(), then printed.

Analysis-

- Popular Diets: "Mostly anything" and "anything" have the highest drink percentages (27.67% and 10.31%).
- Low Drinkers: Strict diets like "strictly vegan" and "strictly halal" show very low drink percentages.
- Small Groups: Niche diets like "mostly kosher" and "strictly halal" have minimal drinkers

4. Geographical Insights

Analyzing geographical data helps Bumble understand its user base distribution, enabling targeted regional campaigns and feature localization. For instance, identifying the top cities with active users can guide marketing efforts in those areas.

1.Extract city and state information from the location column. What are the top 5 cities and states with the highest number of users?

2.How does age vary across the top cities? Are certain cities dominated by younger or older users?

3.What are the average income levels in the top states or cities? Are there regional patterns in reported income?

1.Extract city and state information from the location column. What are the top 5 cities and states with the highest number of users?

In [31]:
```
df[['city','state']] = df['location'].str.split(', ',expand = True, n=1)
df[['city','state']].head(5)
```

Out[31]:

	city	state
0	south san francisco	california
1	oakland	california
2	san francisco	california
3	berkeley	california
4	san francisco	california

In [32]:
```
top_cities = df["city"].value_counts(ascending = False).head(5)
top_states = df["state"].value_counts(ascending = False).head(5)
print(f"Top 5 cities with highest number of users\n {top_cities}" )
print(f"Top 5 states with highest number of users\n {top_states}")
```

```
Top 5 cities with highest number of users
 city
san francisco    31064
oakland           7214
berkeley          4212
san mateo         1331
palo alto         1064
Name: count, dtype: int64
Top 5 states with highest number of users
 state
california       59855
new york            17
illinois             8
massachusetts        5
texas                4
Name: count, dtype: int64
```

- The code splits the location column into city and state using str.split(', ', n=1).
- It then counts the top 5 most frequent cities and states using value_counts().
- Finally, it prints the top 5 cities and states with the highest number of users.

Analysis-

- Top City: San Francisco has the highest number of users with 31,064, followed by Oakland and Berkeley, indicating a strong user presence in California's Bay Area.
- State Distribution: California leads with 59,855 users, while other states like New York, Illinois, and Texas have significantly fewer users, showing a concentration of users in California.

- Sparse Representation: States such as New York, Illinois, and Texas have very few users, suggesting a potential opportunity for expanding Bumble's user base in these areas.

2.How does age vary across the top cities? Are certain cities dominated by younger or older users?

In [33]:
```
city_with_average_age = df.groupby('city')['age'].mean()
print(f" Cities with high average age is :\n {city_with_average_age.sort_values(
print(f" Cities with low average age is :\n {city_with_average_age.sort_values(a
```

```
 Cities with high average age is :
 city
forest knolls     62.5
bellingham        59.0
port costa        53.0
seaside           50.0
redwood shores    47.0
Name: age, dtype: float64
 Cities with low average age is :
 city
fayetteville      20.0
isla vista        19.0
canyon            19.0
canyon country    19.0
long beach        19.0
Name: age, dtype: float64
```

- The code groups the data by city and calculates the average age using groupby('city')['age'].mean().
- It sorts the cities by average age in descending order with sort_values(ascending=False) and displays the top and bottom 5 cities using head(5) and tail(5).
- This shows cities with the highest and lowest average ages.

Analysis-

- Older Cities: Forest Knolls has the highest average age (62.5), followed by Bellingham and Port Costa, indicating these cities have an older population.
- Younger Cities: Fayetteville, Isla Vista, and Canyon have the lowest average ages (around 19), suggesting a younger user base in these areas.
- Age Gap: The data highlights a significant age gap between cities, with some areas having a notably older demographic, while others are dominated by younger users.

3.What are the average income levels in the top states or cities? Are there regional patterns in reported income?

In [34]:
```
top_five_states = df["state"].value_counts().nlargest(5).index
filtered_states_df = df[df["state"].isin(top_five_states)]
average_income_by_state = filtered_states_df.groupby("state")["income"].mean().s
print(average_income_by_state)
```

```
state
new york         31763.823529
california       20043.465609
massachusetts     5999.200000
texas             4999.250000
illinois            -1.000000
Name: income, dtype: float64
```

- Top 5 States: value_counts() and nlargest(5) find the top 5 states with the most users.
- Filter Data: isin() filters the DataFrame to include only these top states.
- Calculate and Sort: groupby() and mean() calculate the average income, which is sorted in descending order and displayed.

Analysis-

- Highest Income: New York has the highest average income ($31,763$), $followed by California with$20,043, indicating a higher earning population in these states.
- Lower Income: Massachusetts, Texas, and Illinois show lower average incomes, with Illinois having a negative value, which might suggest missing or erroneous data.

5. Height Analysis

Physical attributes like height are often considered important in dating preferences. Analyzing height patterns helps Bumble understand user demographics and preferences better.

1.What is the average height of users across different gender categories?

2.How does height vary by age_group? Are there noticeable trends among younger vs. older users?

3.What is the distribution of height within body_type categories (e.g., athletic, curvy, thin)? Do the distributions align with expectations?

1.What is the average height of users across different gender categories?

In [35]:
```
height_gender = df.groupby("gender")["height"].mean()
print(height_gender)
```

```
gender
f    65.103869
m    70.443468
Name: height, dtype: float64
```

- To calculate the average height of users by gender, we used .mean() after grouping by the gender column.

Analysis-

- The result shows that male users have an average height of 70.44, while female users have an average height of 65.10.

2.How does height vary by age_group? Are there noticeable trends among younger vs. older users?

In [36]:
```
height_by_age = df.groupby("age_group")["height"].mean()
print(height_by_age)
```

```
age_group
18-25    68.224532
26-35    68.406764
36-45    68.325095
46+      67.941167
Name: height, dtype: float64
```

```
C:\Users\user\AppData\Local\Temp\ipykernel_11068\4010552321.py:1: FutureWarning:
The default of observed=False is deprecated and will be changed to True in a futu
re version of pandas. Pass observed=False to retain current behavior or observed=
True to adopt the future default and silence this warning.
  height_by_age = df.groupby("age_group")["height"].mean()
```

- To know how the height vary by age_group, calculated the mean for height using .mean() by grouping the age_group column.

Analysis-

- Consistent Heights: Users across all age groups have similar average heights, ranging from 67.94 to 68.41 inches.
- Slight Variation: The 26-35 age group has the highest average height (68.41), while the 46+ group has the lowest (67.94).

3.What is the distribution of height within body_type categories (e.g., athletic, curvy, thin)? Do the distributions align with expectations?

In [37]:
```
height_body_type = df.groupby("body_type")["height"].mean().sort_values(ascendin
print(height_body_type)
```

```
body_type
athletic          69.707336
jacked            69.292162
used up           69.180282
overweight        68.948198
a little extra    68.820084
fit               68.546062
skinny            68.544176
average           68.100805
thin              67.866058
rather not say    67.272727
full figured      66.464817
curvy             65.210245
Name: height, dtype: float64
```

- The average height within each body type category was calculated by grouping the data by the body_type column and using the mean of the height values.

Analysis-

- Taller Body Types: "Athletic" users have the highest average height (69.71), followed by "Jacked" and "Used Up" body types.
- Shorter Body Types: "Curvy" and "Full Figured" have the lowest average heights (65.21 and 66.46), with "Rather Not Say" also being shorter (67.27).

6. Income Analysis

Income is often an important factor for users on dating platforms. Understanding its distribution and relationship with other variables helps refine features like user search filters or personalized recommendations.

1.What is the distribution of income across the platform? Are there specific income brackets that dominate? (don't count 0)

2.How does income vary by age_group and gender? Are older users more likely to report higher incomes?

1.What is the distribution of income across the platform? Are there specific income brackets that dominate? (don't count 0)

In [38]:
```
income_without_zero = df[df['income'] != 0]
income_without_zero['income'].value_counts().sort_values(ascending = False)
```

Out[38]:
```
income
-1          48442
 20000       2952
 100000      1621
 80000       1111
 30000       1048
 40000       1005
 50000        975
 60000        736
 70000        707
 150000       631
 1000000      521
 250000       149
 500000        48
Name: count, dtype: int64
```

Analysis -

- The dominating income category is 20000 with the number of users as 2952.

2.How does income vary by age_group and gender? Are older users more likely to report higher incomes?

In [39]:
```
average_income_by_age = income_without_zero.groupby('age_group')['income'].mean(
print(f"Income Distribution Across Age Groups:\n\n{average_income_by_age}")
```

```
Income Distribution Across Age Groups:

age_group
18-25    20631.935171
26-35    19687.870969
36-45    21072.974729
46+      18873.628708
Name: income, dtype: float64
```

```
C:\Users\user\AppData\Local\Temp\ipykernel_11068\3541258303.py:1: FutureWarning:
The default of observed=False is deprecated and will be changed to True in a futu
re version of pandas. Pass observed=False to retain current behavior or observed=
True to adopt the future default and silence this warning.
  average_income_by_age = income_without_zero.groupby('age_group')['income'].mean
()
```

We are grouping the data by the 'age group' column and calculating the mean income for each group using .groupby().

Analysis -

- Older users do not necessarily report higher incomes. In fact, the 26-35 age group has the highest average income, suggesting that income tends to peak in the late 20s and early 30s, when individuals are more established in their careers.

Part 4: Data Visualization

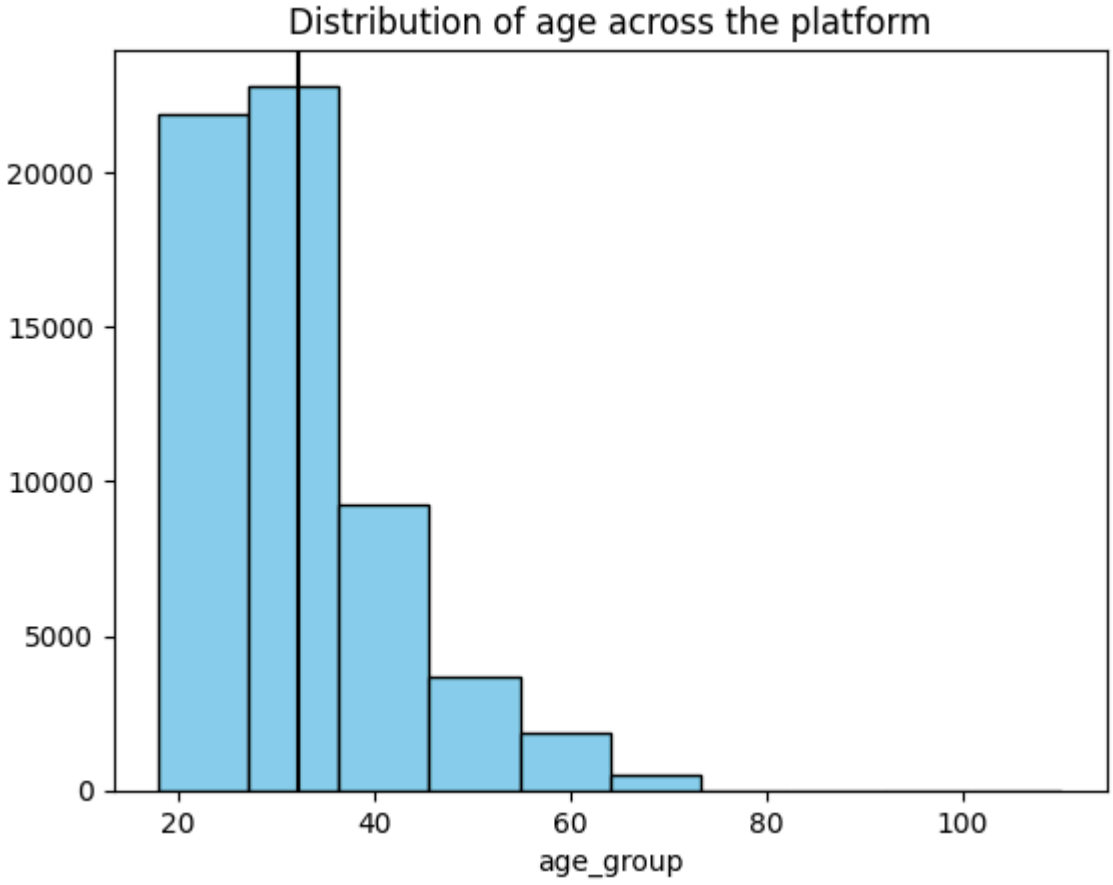
1. Age Distribution

Understanding the distribution of user ages can reveal whether the platform caters to specific demographics or age groups. This insight is essential for targeted marketing and user experience design.

1.Plot a histogram of age with a vertical line indicating the mean age. What does the distribution reveal about the most common age group on the platform? 2.How does the age distribution differ by gender? Are there age groups where one gender is more prevalent?

1.Plot a histogram of age with a vertical line indicating the mean age. What does the distribution reveal about the most common age group on the platform?

In [40]:
```
plt.hist(df["age"], color = "skyblue", edgecolor= "black")
mean_age = df["age"].mean()
plt.axvline(mean_age, color= "black", label= "mean age")
plt.xlabel("age_group")
plt.title("Distribution of age across the platform")
plt.show()
```

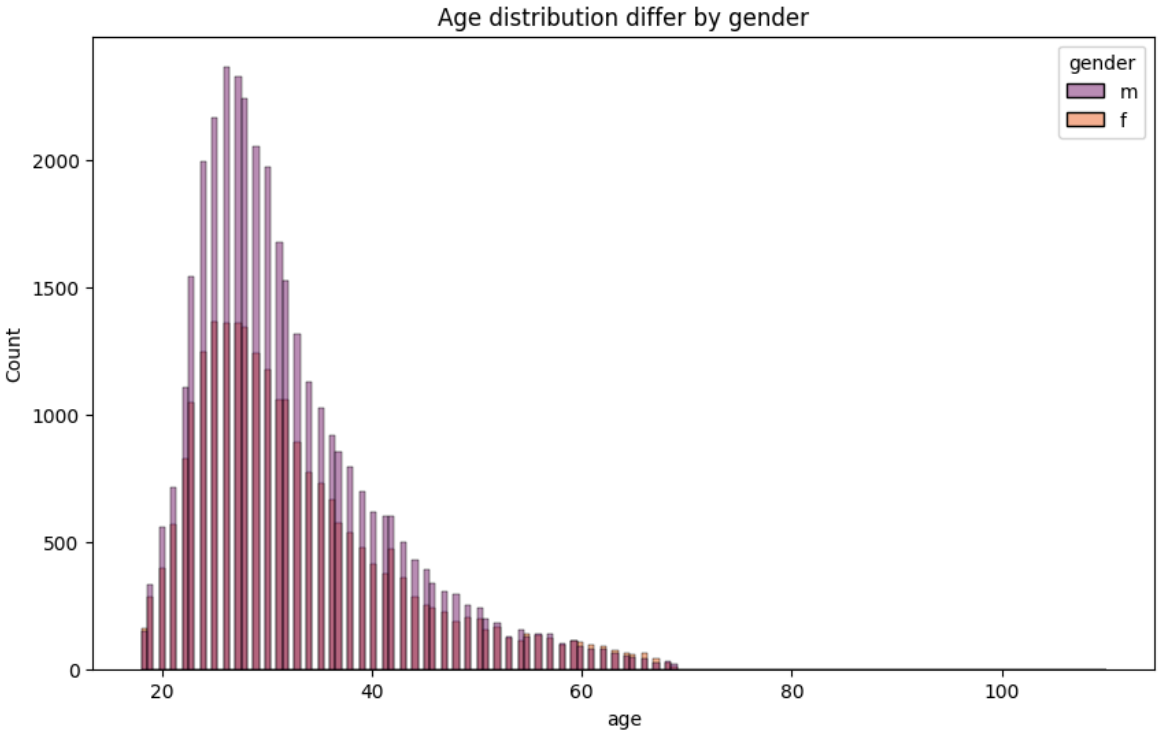


- We used a histogram to visualize the distribution of the most common age group. By calculating the mean age and representing it with a vertical line using axvline, the result shows that users in their 30s are the most prevalent age group on the platform.

2.How does the age distribution differ by gender? Are there age groups where one gender is more prevalent?

In [41]:
```
plt.figure(figsize=(10, 6))
sns.histplot(data=df, x="age", hue="gender", palette = "inferno")
plt.title("Age distribution differ by gender")
```

Out[41]:
```
Text(0.5, 1.0, 'Age distribution differ by gender')
```



2. Income and Age

Visualizing the relationship between income and age helps uncover patterns in reported income levels across age groups, which could inform user segmentation strategies.

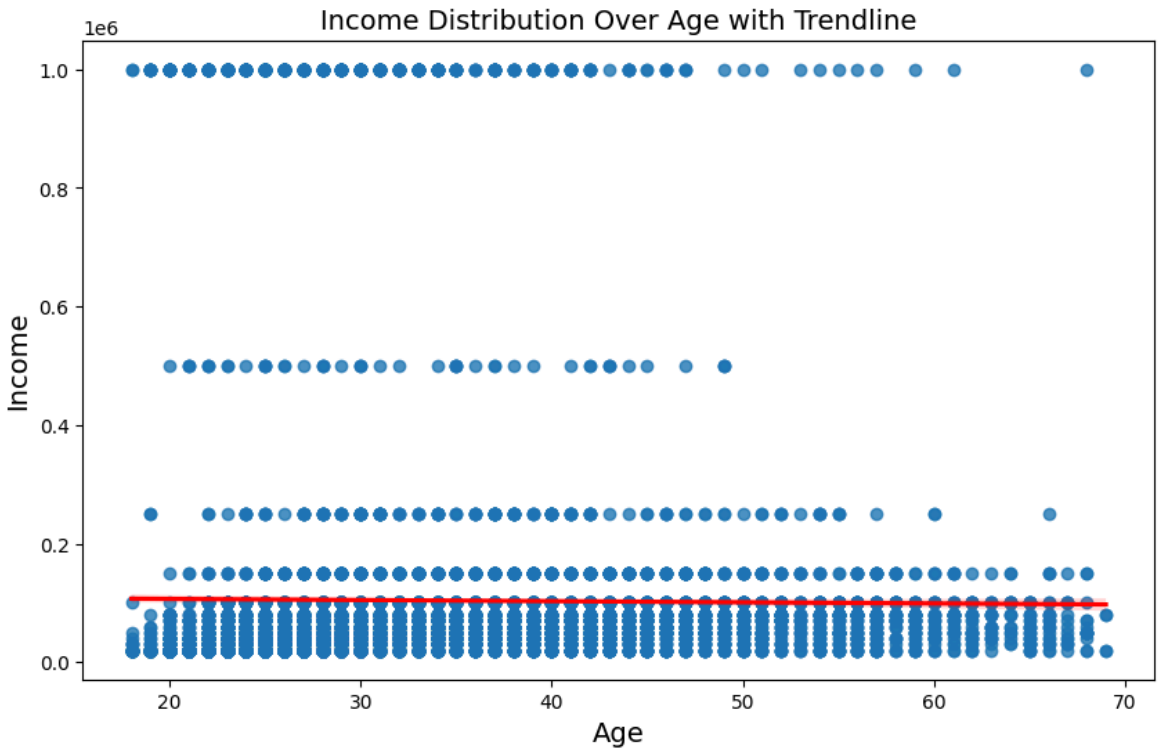
1.Use a scatterplot to visualize the relationship between income and age, with a trend line indicating overall patterns. Are older users more likely to report higher incomes?

2.Create boxplots of income grouped by age_group. Which age group reports the highest median income?

3.Analyze income levels within gender and status categories. For example, are single men more likely to report higher incomes than single women?

1.Use a scatterplot to visualize the relationship between income and age, with a trend line indicating overall patterns. Are older users more likely to report higher incomes?

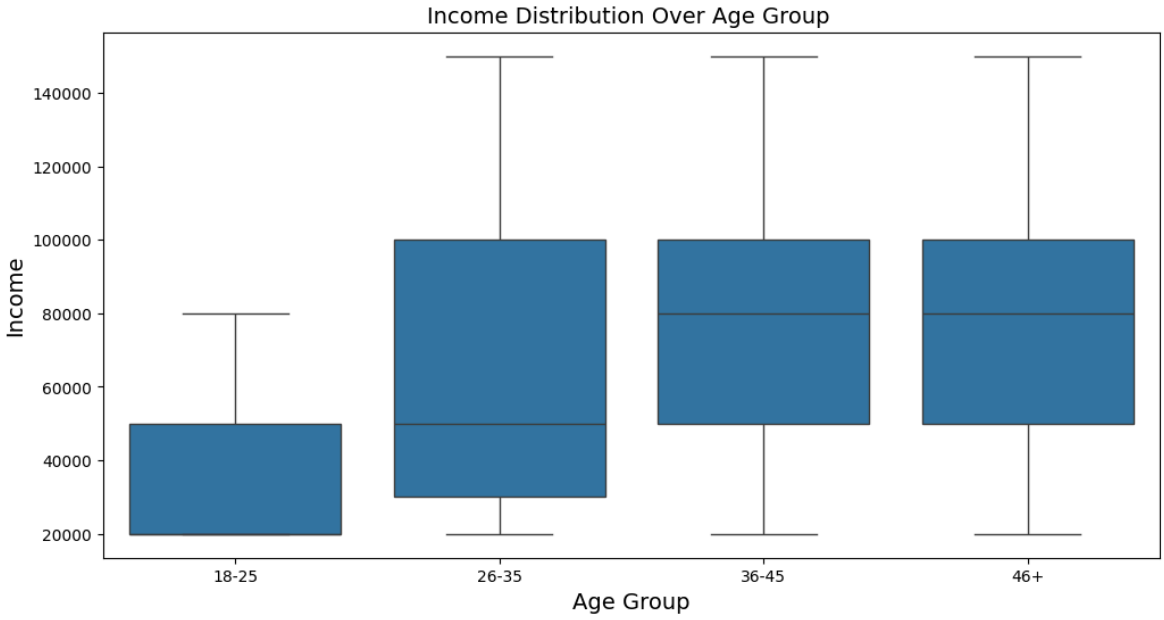
In [42]:
```
plt.figure(figsize=(10, 6))
sns.regplot(data=df[df['income']>0], x='age', y='income', line_kws={'color': 're
plt.xlabel('Age',fontsize =14)
plt.ylabel('Income',fontsize =14)
plt.title('Income Distribution Over Age with Trendline',fontsize =14)
plt.show()
```



- A scatter plot was created using Seaborn's regplot to explore the relationship between age and income. The result indicates that most users, regardless of age, have similar income levels, with income not showing a clear increase as age progresses.

2.Create boxplots of income grouped by age_group. Which age group reports the highest median income?

In [43]:
```
plt.figure(figsize=(12,6))
sns.boxplot(data= df[df['income'] > 0], x="age_group", y='income',showfliers = F
plt.xlabel('Age Group',fontsize =14)
plt.ylabel('Income',fontsize =14)
plt.title('Income Distribution Over Age Group ',fontsize =14)
plt.show()
```



- Boxplots of income grouped by age group were created using sns.boxplot(). The results show that the median income for the 36-45 and 46+ age groups is almost identical.

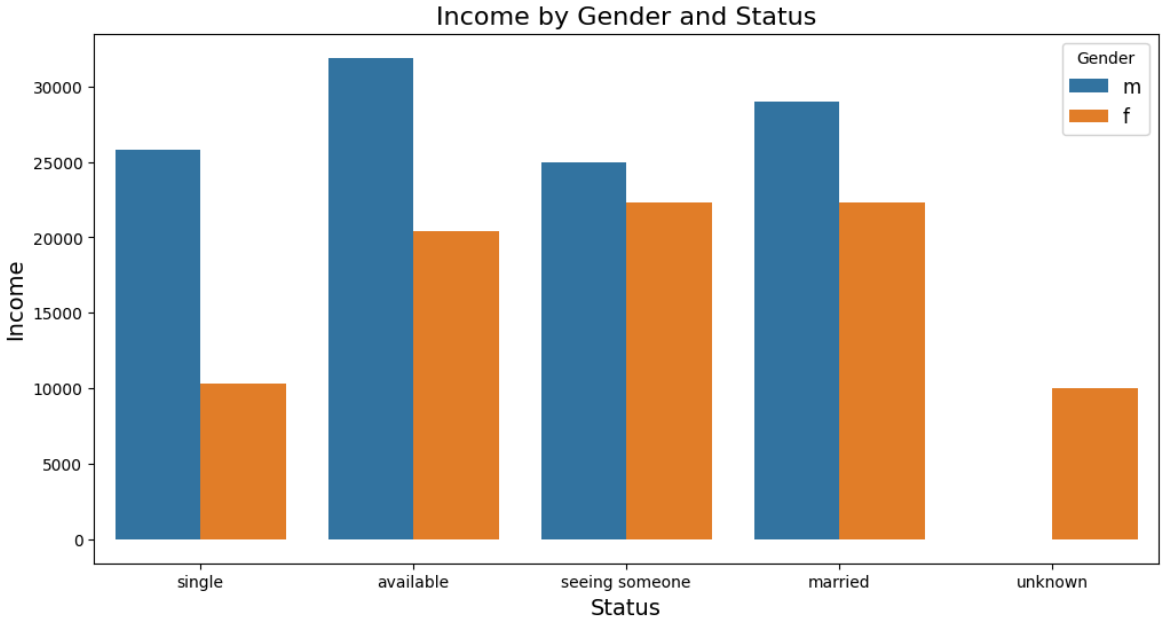
3.Analyze income levels within gender and status categories. For example, are single men more likely to report higher incomes than single women?

In [44]:
```
plt.figure(figsize=(12, 6))
sns.barplot(data=df, x="status", y="income", hue="gender", ci=None)
plt.xlabel("Status", fontsize=14)
plt.ylabel("Income", fontsize=14)
plt.title("Income by Gender and Status", fontsize=16)
plt.legend(title="Gender", fontsize=12)
plt.show()
```

```
C:\Users\user\AppData\Local\Temp\ipykernel_11068\2820912845.py:2: FutureWarning:

The `ci` parameter is deprecated. Use `errorbar=None` for the same effect.

  sns.barplot(data=df, x="status", y="income", hue="gender", ci=None)
```



- To analyze income levels across gender and status categories, the mean income was calculated using .mean() after grouping by gender and status. A barplot was created with Seaborn, using gender as the hue. The results show that single men tend to have higher incomes compared to single women.

3. Pets and Preferences

Pets are often a key lifestyle preference and compatibility factor. Analyzing how pets preferences distribute across demographics can provide insights for filters or recommendations.

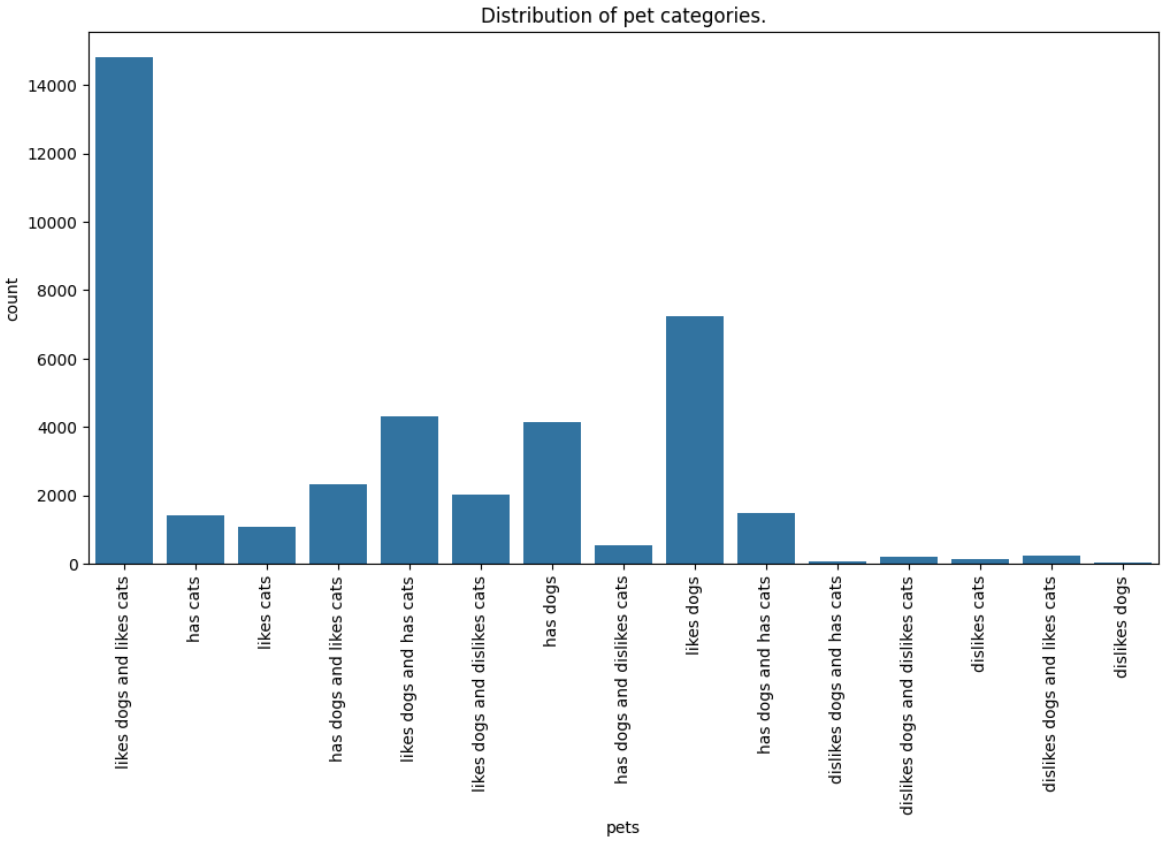
1.Create a bar chart showing the distribution of pets categories (e.g., likes dogs, likes cats). Which preferences are most common?

2.How do pets preferences vary across gender and age_group? Are younger users more likely to report liking pets compared to older users?

1.Create a bar chart showing the distribution of pets categories (e.g., likes dogs, likes cats). Which preferences are most common?

In [45]:
```
plt.figure(figsize=(12,6))
sns.countplot(data= df , x = "pets")
plt.xticks(rotation = 90)
plt.title("Distribution of pet categories.")
```

Out[45]:
```
Text(0.5, 1.0, 'Distribution of pet categories.')
```



- The distribution of pet preferences was visualized using sns.countplot(). The results indicate that the "likes dogs and cats" category has the highest number of users, while the "dislikes dogs" category has very few users.

2.How do pets preferences vary across gender and age_group? Are younger users more likely to report liking pets compared to older users?

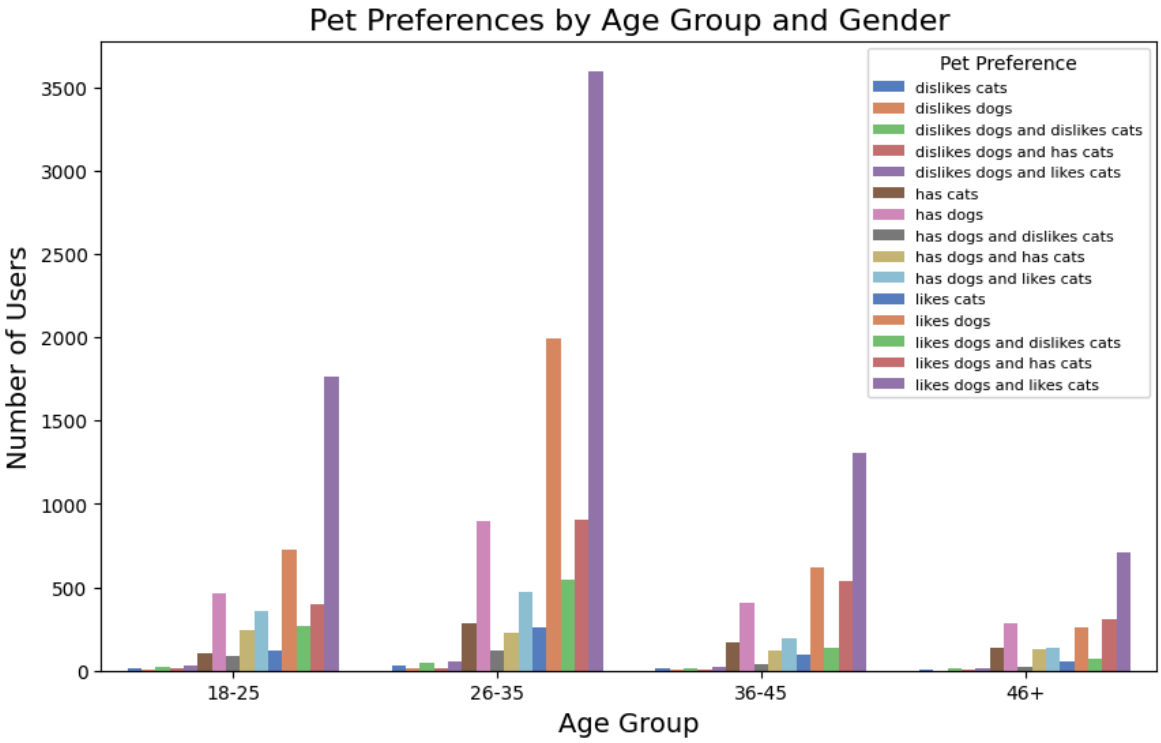
In [46]:
```
pet_distribution_by_group = df.groupby(['age_group', 'gender', 'pets']).size().r
plt.figure(figsize=(10, 6))
sns.barplot(data=pet_distribution_by_group, x="age_group", y="count", hue="pets"

plt.xlabel("Age Group", fontsize=14)
plt.ylabel("Number of Users", fontsize=14)
plt.title("Pet Preferences by Age Group and Gender", fontsize=16)
plt.legend(title="Pet Preference", fontsize=8)
plt.show()
```

```
C:\Users\user\AppData\Local\Temp\ipykernel_11068\3787535282.py:1: FutureWarning:
The default of observed=False is deprecated and will be changed to True in a futu
re version of pandas. Pass observed=False to retain current behavior or observed=
True to adopt the future default and silence this warning.
  pet_distribution_by_group = df.groupby(['age_group', 'gender', 'pets']).size().
reset_index(name='count')
C:\Users\user\AppData\Local\Temp\ipykernel_11068\3787535282.py:3: FutureWarning:

The `ci` parameter is deprecated. Use `errorbar=None` for the same effect.

  sns.barplot(data=pet_distribution_by_group, x="age_group", y="count", hue="pet
s", palette="muted", ci = None)
```



4. Signs and Personality

Users' self-reported zodiac signs (sign) can offer insights into personality preferences or trends. While not scientifically grounded, analyzing this data helps explore fun and engaging patterns.

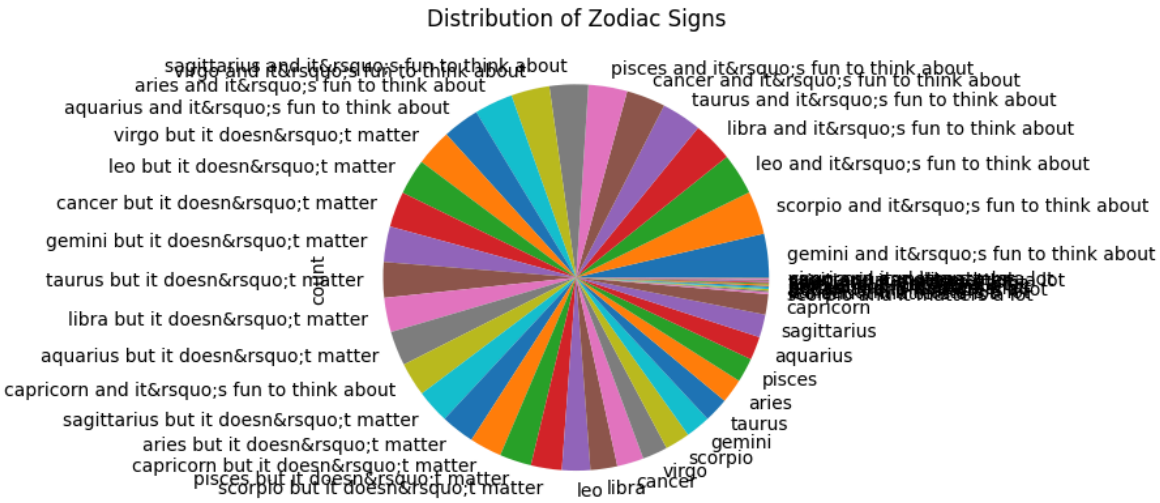
1.Create a pie chart showing the distribution of zodiac signs (sign) across the platform. Which signs are most and least represented? Is this the right chart? If not, replace with right chart.

2.How does sign vary across gender and status? Are there noticeable patterns or imbalances?

1.Create a pie chart showing the distribution of zodiac signs (sign) across the platform. Which signs are most and least represented? Is this the right chart? If not, replace with right chart.

In [47]:
```
zodiac_counts = df["sign"].value_counts()
zodiac_counts.plot(kind='pie')
plt.title("Distribution of Zodiac Signs")
```

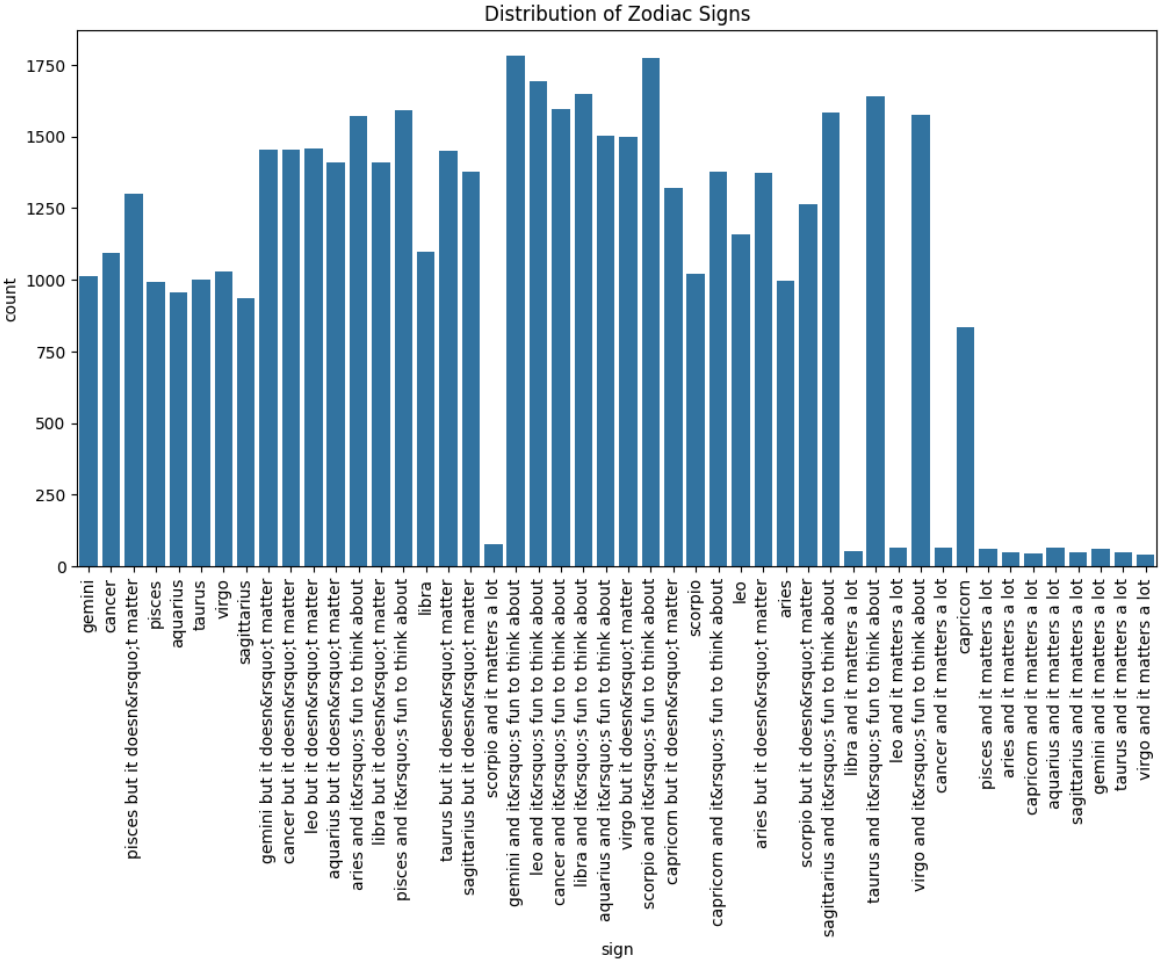
Out[47]:
```
Text(0.5, 1.0, 'Distribution of Zodiac Signs')
```



- A pie chart was used to visualize the distribution of zodiac signs across the platform. However, the chart does not provide clear insights, making it an unsuitable choice for this representation.

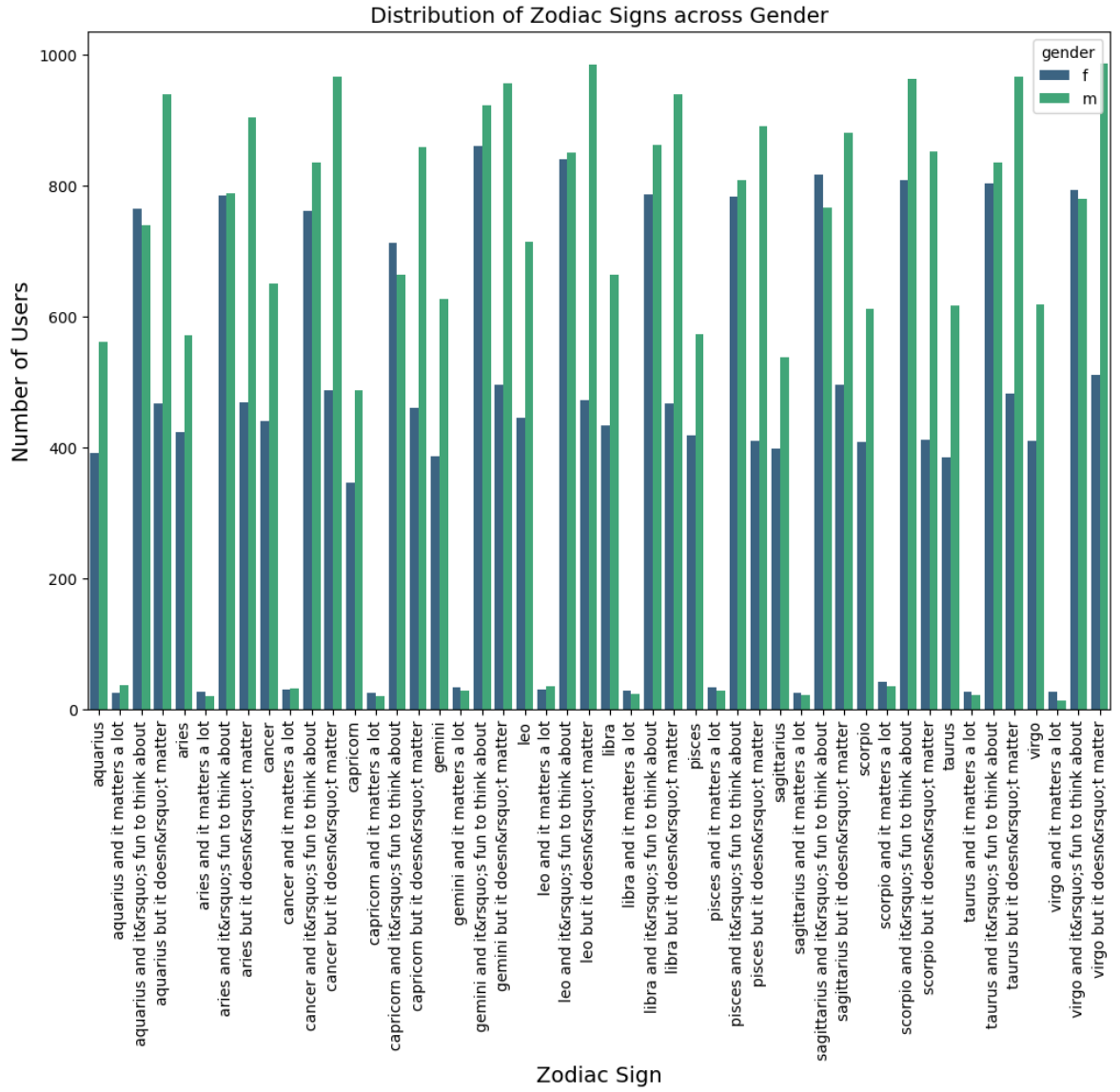
In [48]:
```
plt.figure(figsize=(12,6))
sns.countplot(data= df , x = "sign")
plt.xticks(rotation = 90)
plt.title("Distribution of Zodiac Signs")
```

Out[48]:
```
Text(0.5, 1.0, 'Distribution of Zodiac Signs')
```

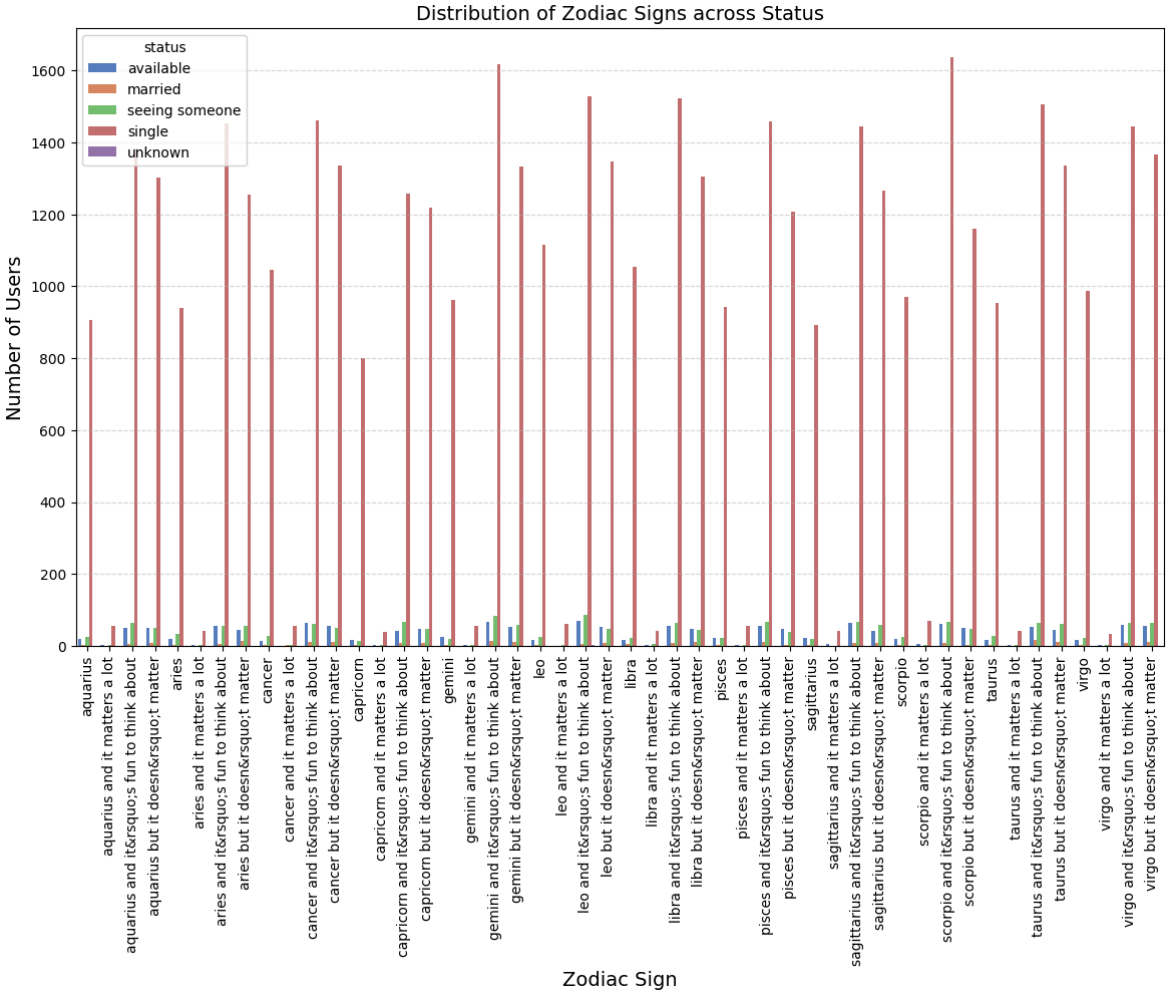


2.How does sign vary across gender and status? Are there noticeable patterns or imbalances?

In [49]:
```
sign_across_gender = df.groupby(['sign', 'gender']).size().reset_index(name='cou
plt.figure(figsize=(12, 8))
sns.barplot(data=sign_across_gender, x='sign', y='count', hue='gender', palette=
plt.title('Distribution of Zodiac Signs across Gender', fontsize=14)
plt.xlabel('Zodiac Sign', fontsize=14)
plt.ylabel('Number of Users', fontsize=14)
plt.xticks(rotation=90)
plt.show()
```



In [50]:
```
sign_across_status = df.groupby(['sign', 'status']).size().reset_index(name='cou
plt.figure(figsize=(14,8))
sns.barplot(data=sign_across_status, x='sign', y='count', hue='status', palette=
plt.title('Distribution of Zodiac Signs across Status', fontsize=14)
plt.xlabel('Zodiac Sign', fontsize=14)
plt.ylabel('Number of Users', fontsize=14)
plt.xticks(rotation=90)
plt.grid(True, axis= 'y', linestyle = 'dashed', alpha = 0.5)
plt.show()
```


Distribution of Zodiac Signs across Status
status
available
married
seeing someone
single
unknown
Number of Users
0
200
400
600
800
1000
1200
1400
1600
aquarius
aquarius and it matters a lot
aquarius and it’s fun to think about
aquarius but it doesn’t matter
aries
aries and it matters a lot
aries and it’s fun to think about
aries but it doesn’t matter
cancer
cancer and it matters a lot
cancer and it’s fun to think about
cancer but it doesn’t matter
capricorn
capricorn and it matters a lot
capricorn and it’s fun to think about
capricorn but it doesn’t matter
gemini
gemini and it matters a lot
gemini and it’s fun to think about
gemini but it doesn’t matter
leo
leo and it matters a lot
leo and it’s fun to think about
leo but it doesn’t matter
libra
libra and it matters a lot
libra and it’s fun to think about
libra but it doesn’t matter
pisces
pisces and it matters a lot
pisces and it’s fun to think about
pisces but it doesn’t matter
sagittarius
sagittarius and it matters a lot
sagittarius and it’s fun to think about
sagittarius but it doesn’t matter
scorpio
scorpio and it matters a lot
scorpio and it’s fun to think about
scorpio but it doesn’t matter
taurus
taurus and it matters a lot
taurus and it’s fun to think about
taurus but it doesn’t matter
virgo
virgo and it matters a lot
virgo and it’s fun to think about
virgo but it doesn’t matter
Zodiac Sign