ICE: Outliers & Missing Data

Name: Cabot Steward

DATA 3300

> In this assignment, you will learn how to handle missing data and outliers using Python. The topics covered include determining where data is missing, visualizing missingness, and understanding structurally missing data. You will also explore the concept of data missing at random (MAR) and various imputation techniques for filling in missing values, such as simple imputation and multiple imputation methods.

> Additionally, the assignment will teach you how to visualize outliers, use methods to correct or handle outliers without removing them, and apply transformations to reduce their impact on your analysis. Finally, you will learn how to discretize continuous variables into categorical variables, which can help manage outliers and improve model performance. This comprehensive assignment will equip you with the skills to effectively manage and preprocess your data, ensuring robust and reliable analyses.

Client Scenario

MindWell Health, a pioneering mental health telehealth startup, is dedicated to providing accessible and effective mental health services to individuals working in various industries. To better understand the mental health needs and trends among employees at different companies, MindWell Health has collected an extensive dataset.

The data team at MindWell Health has tasked us with preprocessing this dataset to address challenges related to missing data and outliers.

By correcting these data preprocessing concerns, MindWell Health aims to derive accurate insights from the data, which will inform their service offerings and enable them to tailor mental health interventions. This will help them provide more personalized and effective care, ultimately supporting their mission to enhance mental well-being through innovative telehealth solutions.

Data Card

This dataset contains the following data:

- SurveyID
- Age: free response
- Gender: free response
- Country
- state: If you live in the United States, which state or territory do you live in?
- self_employed: Are you self-employed?
- family_history: Do you have a family history of mental illness?
- treatment: Have you sought treatment for a mental health condition?
- work_interfere: If you have a mental health condition, do you feel that it interferes with your work?
- no_employees: How many employees does your company or organization have?
- remote_work: Do you work remotely (outside of an office) at least 50% of the time?
- tech_company: Is your employer primarily a tech company/organization?
- benefits: Does your employer provide mental health benefits?
- care_options: Do you know the options for mental health care your employer provides?
- wellness_program: Has your employer ever discussed mental health as part of an employee wellness program?
- seek_help: Does your employer provide resources to learn more about mental health issues and how to seek help?
- comments: Any additional notes or comments

Let's begin by loading in our dependencies that will allow us to visualize our data for missingness and outliers, as well as some functions from `sklearn` that will allow us to impute (or estimate) missing values where appropriate!

In [46]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

from sklearn.impute import SimpleImputer
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer
```

```python
pd.set_option('display.max_columns', None)
```

In [47]:
```python
df = pd.read_csv('Tech_MentalHealth.csv')
# display data
df.head()
```

Out[47]:

	SurveyID	Age	Gender	Country	state	self_employed	family_history	treatment	wor
0	1	37.0	Female	United States	IL	No	No	Yes	
1	2	44.0	M	United States	IN	No	No	No	
2	3	32.0	Male	Canada	NaN	No	No	No	
3	4	NaN	Male	United Kingdom	NaN	No	Yes	Yes	
4	5	31.0	Male	United States	TX	No	No	No	

One way we can check for missingness is just simply using df.info(). How many total observations do we have, and do any columns contain missing values?

In [48]:
```python
# show info
df.info()

# we have 1248 entries
# age, state, work_interference, comments
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1248 entries, 0 to 1247
Data columns (total 17 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   SurveyID          1248 non-null   int64  
 1   Age               1221 non-null   float64
 2   Gender            1248 non-null   object 
 3   Country           1248 non-null   object 
 4   state             744 non-null    object 
 5   self_employed     1248 non-null   object 
 6   family_history    1248 non-null   object 
 7   treatment         1248 non-null   object 
 8   work_interfere    987 non-null    object 
 9   no_employees      1248 non-null   int64  
 10  remote_work       1248 non-null   object 
 11  tech_company      1248 non-null   object 
 12  benefits          1248 non-null   object 
 13  care_options      1248 non-null   object 
 14  wellness_program  1248 non-null   object 
 15  seek_help         1248 non-null   object 
 16  comments          162 non-null    object 
dtypes: float64(1), int64(2), object(14)
memory usage: 165.9+ KB
```

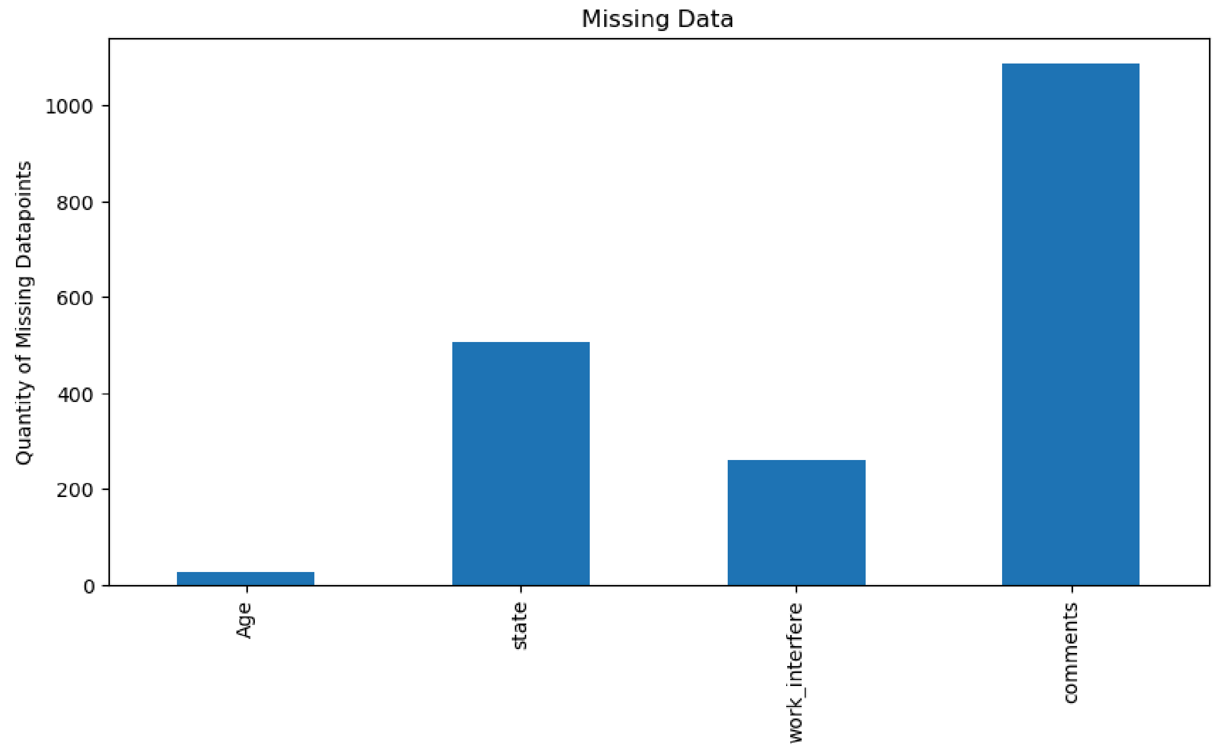
we have 1248 entries

age, state, work_interference, comments

Visualizing Missing Data

This isn't a great way to absorb this informaiton, or understand patterns of missingness however, so let's use some additional visualizations...

In [49]:
```
# Create a bar plot of missing data
missing_data = df.isnull().sum()
plt.figure(figsize=(10,5))
missing_data[missing_data > 0].plot(kind='bar')
plt.title('Missing Data')
plt.ylabel('Quantity of Missing Datapoints')
plt.show()
```

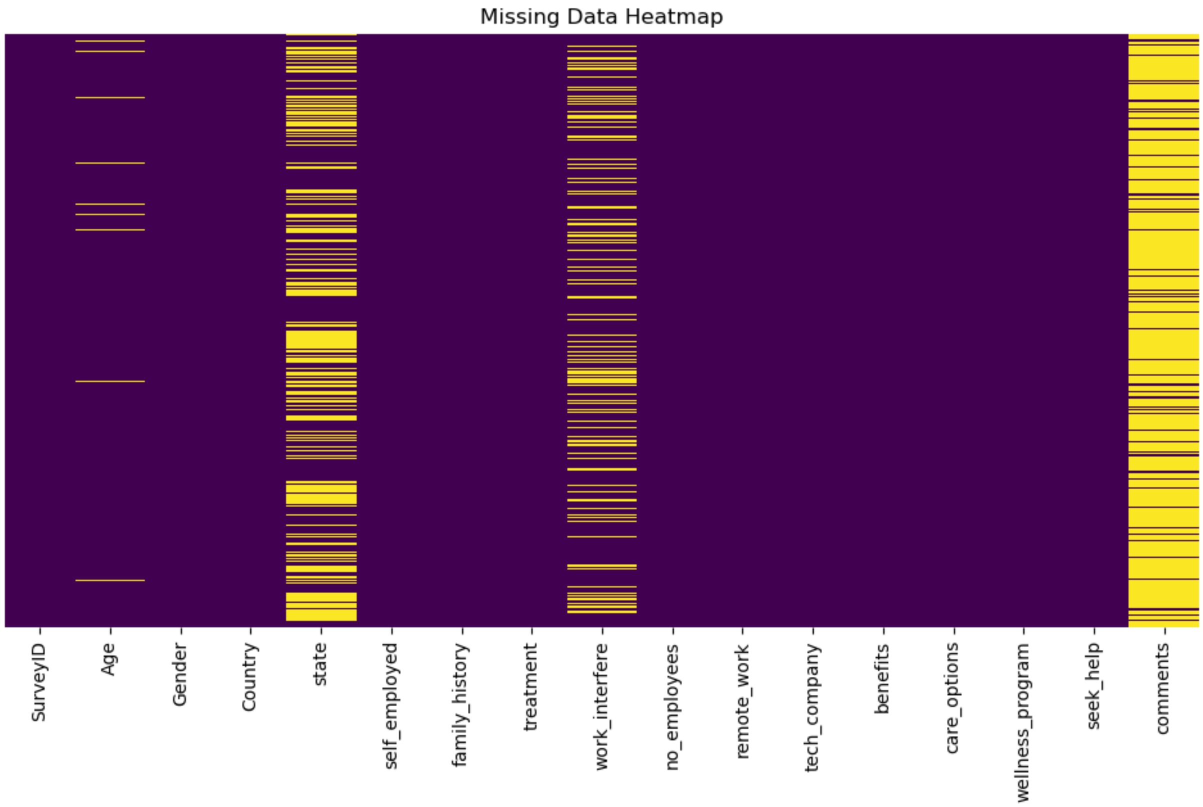


Which variable has the most missing data, and does it make sense that this column has the most missing values?

comments, yes

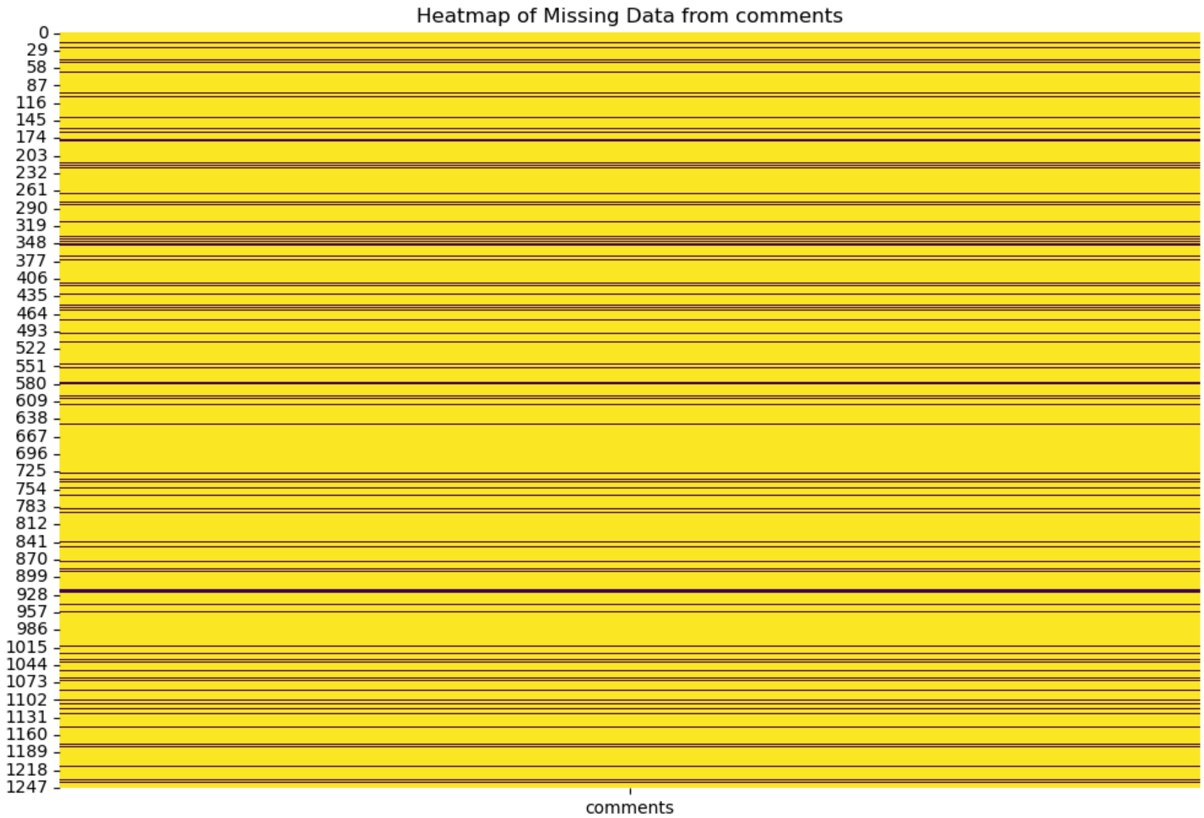
Next, let's use a heatmap visualization that allows us to see missingess across participants and variables...

In [50]:
```
plt.figure(figsize=(12,6))
sns.heatmap(df.isnull(), cmap='viridis', cbar=False, yticklabels=False)
plt.title('Missing Data Heatmap')
plt.show()
```



We'll zoom in a bit on one section...

In [51]:
```
df_sub = df[['comments']]
plt.figure(figsize=(12, 8))
sns.heatmap(df_sub.isnull(), cbar=False, cmap="viridis")
plt.title('Heatmap of Missing Data from comments')
plt.show()
```



What if any patterns appear amongst the missing data collectively across variables?

- The nulls are generally on columns that dont matter as much
- comments are generally left null

Let's examine just State and Country, and look at observations where state is missing. Is there a structural reason why state is missing? How can we deal with this form of missingness?

In [52]:
```
# pull up just columns state and Country and observations where state is missing
df[['state', 'Country']][df['state'].isnull()]
```

Out[52]:

	state	Country
2	NaN	Canada
3	NaN	United Kingdom
7	NaN	Canada
9	NaN	Canada
11	NaN	Bulgaria
...	...	...
1233	NaN	United Kingdom
1234	NaN	Australia
1236	NaN	Finland
1240	NaN	South Africa
1243	NaN	United Kingdom

504 rows × 2 columns

In [53]:
```
# where country is United States, check for NaNs in the state column
df[df['state'] == "United States"]['state'].isnull().sum()
```

Out[53]: 0

Negative

Fill in NaNs with New Value

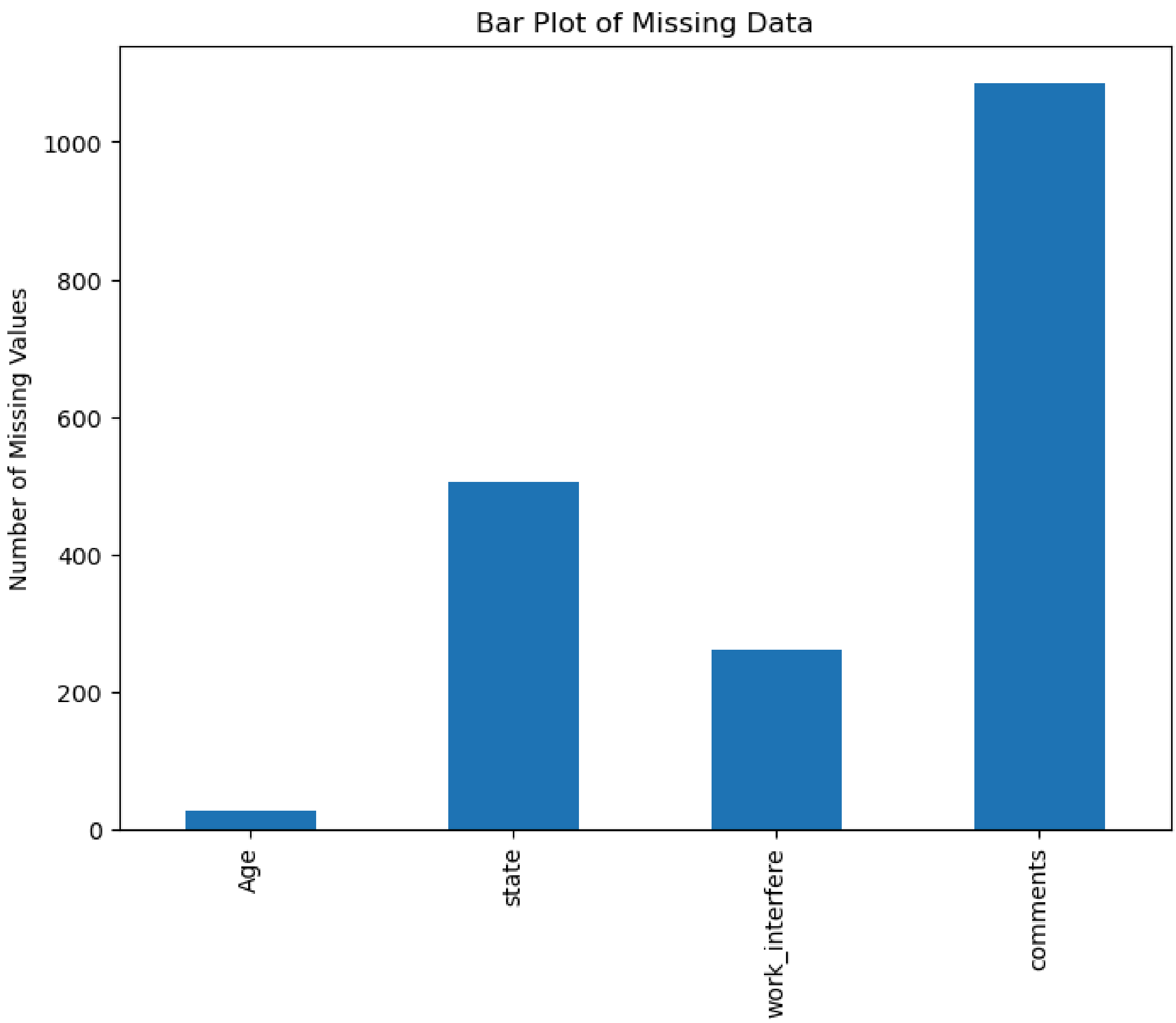
In [54]:
```
# fill in NaNs in the state column with the text 'NA'

df['state'].replace({np.nan:"NA"})
df.head()
```

Out[54]:

	SurveyID	Age	Gender	Country	state	self_employed	family_history	treatment	wor
0	1	37.0	Female	United States	IL	No	No	Yes	
1	2	44.0	M	United States	IN	No	No	No	
2	3	32.0	Male	Canada	NaN	No	No	No	
3	4	NaN	Male	United Kingdom	NaN	No	Yes	Yes	
4	5	31.0	Male	United States	TX	No	No	No	

In [55]:

```
# Create a bar plot of missing data
missing_data = df.isnull().sum()
plt.figure(figsize=(8, 6))
missing_data[missing_data > 0].plot(kind='bar')
plt.title('Bar Plot of Missing Data')
plt.ylabel('Number of Missing Values')
plt.show()
```



What does the work_interfere variable indicate on the survey, what type of missing data is suspected?

- work_interfere: If you have a mental health condition, do you feel that it interferes with your work?

if its missing its likely that they don't have a mental health condition

In [56]:
```
df['work_interfere'].fillna('No mental health condition', inplace=True)

missing_data = df.isnull().sum()
plt.figure(figsize=(8, 6))
missing_data[missing_data > 0].plot(kind='bar')
plt.title('Bar Plot of Missing Data')
plt.ylabel('Number of Missing Values')
plt.show()
```

```
C:\Users\tucke\AppData\Local\Temp\ipykernel_43740\3239272805.py:1: FutureWarning: A
value is trying to be set on a copy of a DataFrame or Series through chained assignm
ent using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because
the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method
({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform
the operation inplace on the original object.


  df['work_interfere'].fillna('No mental health condition', inplace=True)
```

Finally, let's investigate Age. What type of missing variable is it likely NOT to be?

Not a string or to large of number

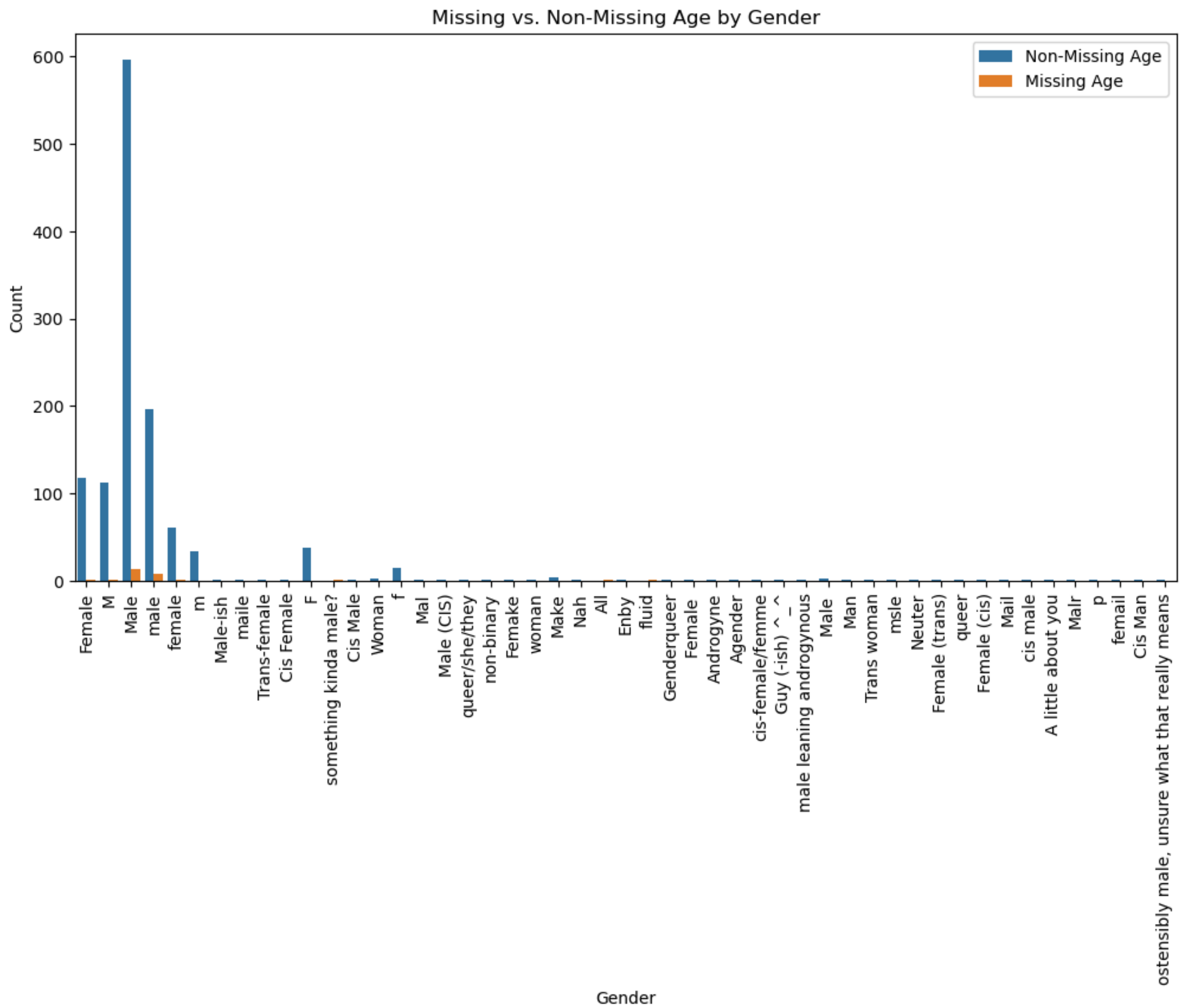
Visualizing Patterns of Missingess Against other Variables

In [57]:
```
df.columns
```

Out[57]:
```
Index(['SurveyID', 'Age', 'Gender', 'Country', 'state', 'self_employed',
       'family_history', 'treatment', 'work_interfere', 'no_employees',
       'remote_work', 'tech_company', 'benefits', 'care_options',
       'wellness_program', 'seek_help', 'comments'],
      dtype='object')
```

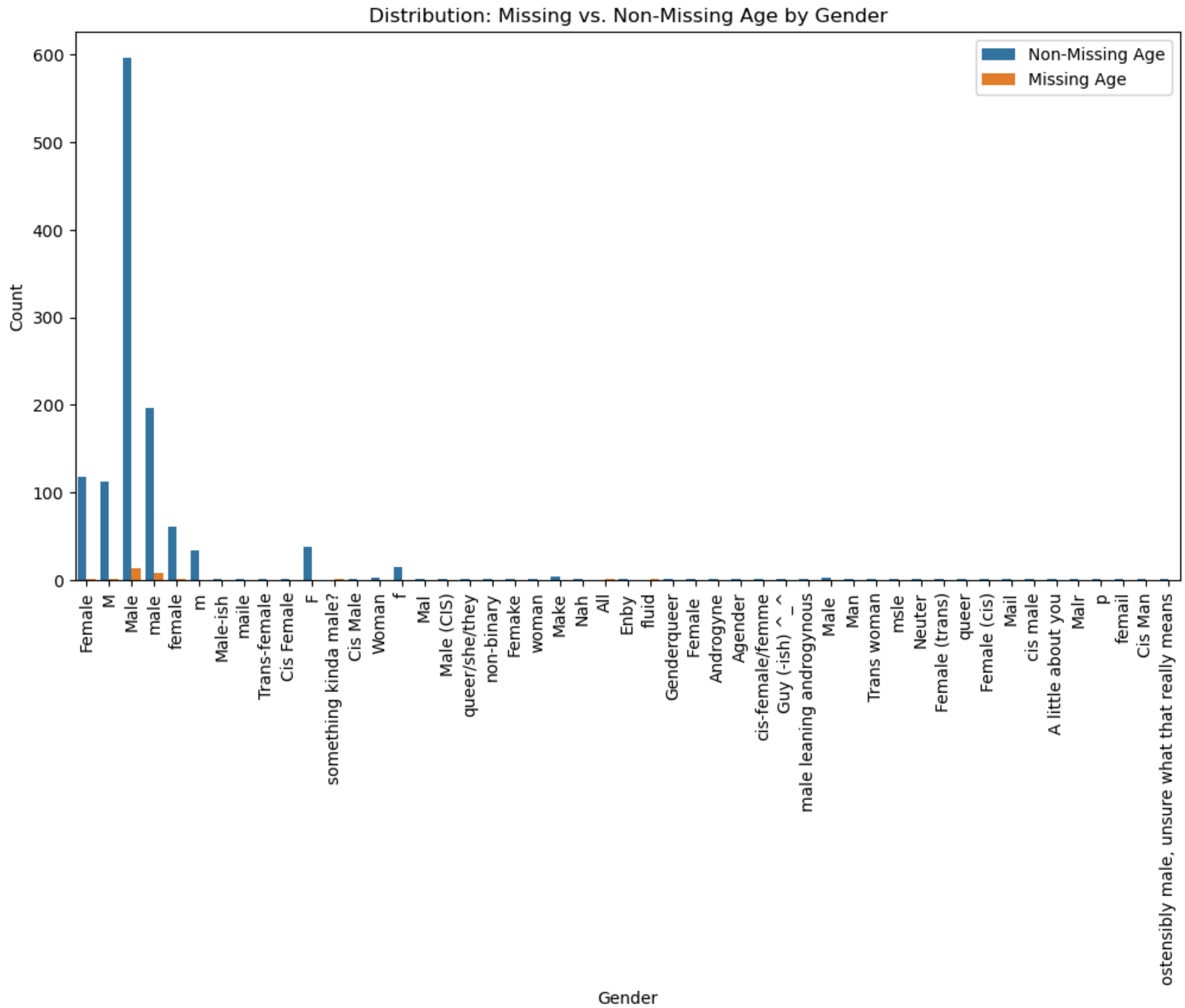
Let's examine Age missingess against other variables to see if any patterns emerge...

In [58]:
```python
# Create a boolean mask for missing Age values
df['missing_age'] = df['Age'].isnull().astype(int)

plt.figure(figsize=(12, 6))
sns.countplot(data=df, x='Gender', hue='missing_age')
plt.title('Missing vs. Non-Missing Age by Gender')
plt.xlabel('Gender')
plt.ylabel('Count')
plt.legend(['Non-Missing Age', 'Missing Age'])
plt.xticks(rotation=90)
plt.show()
```



In [59]:
```python
df['missing_age_mask'] = df['Age'].isnull().astype(int)
plt.figure(figsize=(12, 6))
sns.countplot(data=df, x='Gender', hue='missing_age_mask')
plt.title('Distribution: Missing vs. Non-Missing Age by Gender')
```

```
plt.xlabel('Gender')
plt.ylabel('Count')
plt.legend(['Non-Missing Age', 'Missing Age'])
plt.xticks(rotation=90)
plt.show()
```



What do these patterns indicate about the type of missingness?

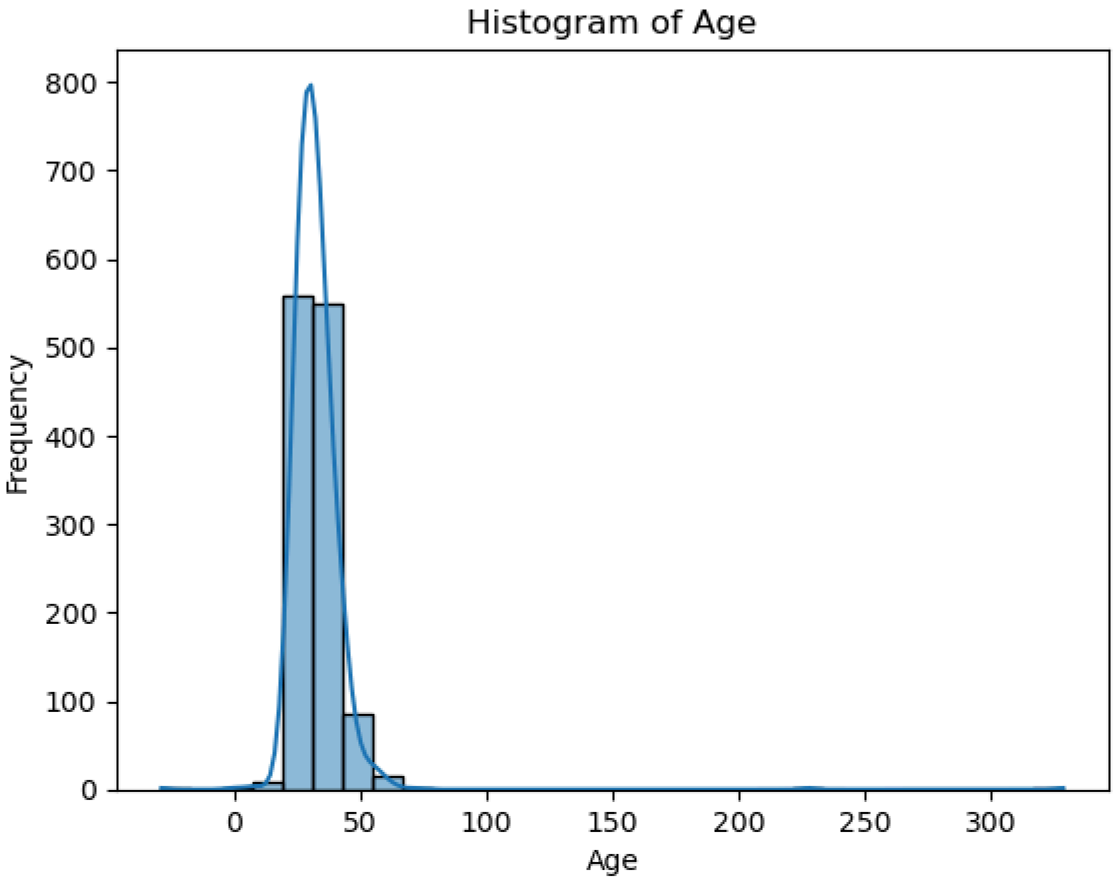
there are also missing values that are in the dataset, such as Nah

Visualizing Outliers

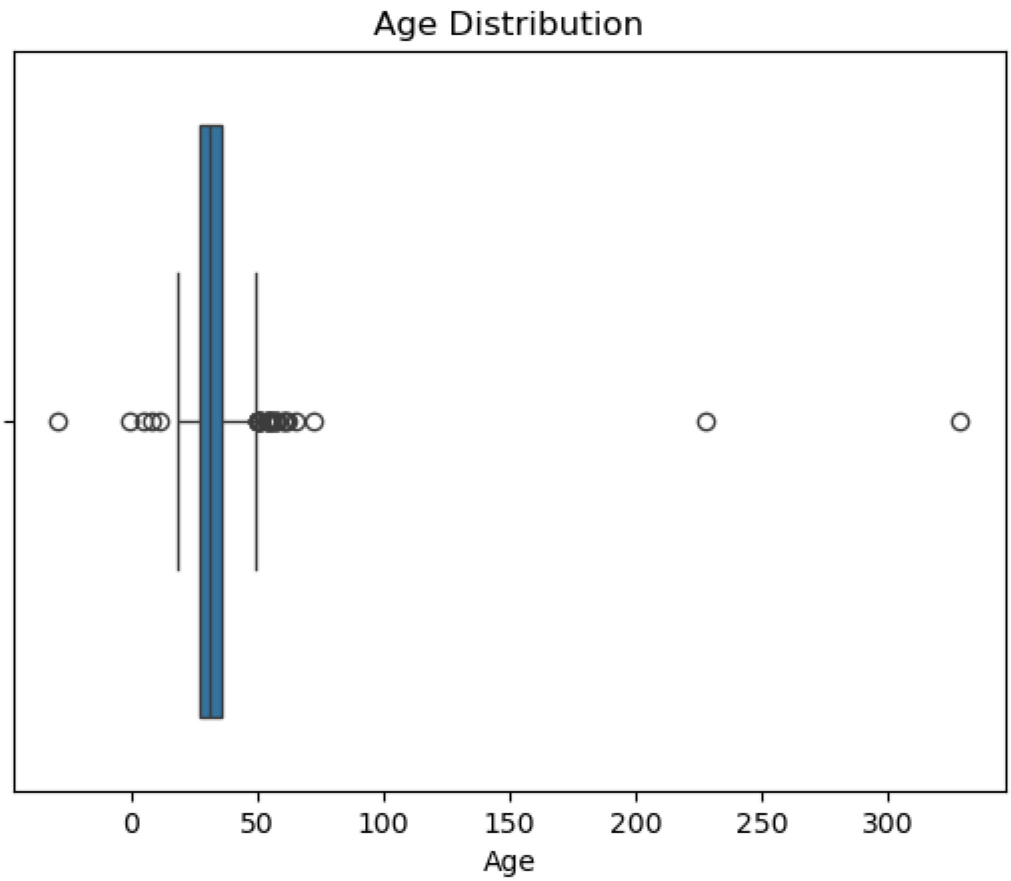
One way to impute missing data is to simply fill in missing values with the mean of age (simple imputation). What should we check before we use the mean?

In [60]:
```
# plot histogram
sns.histplot(df['Age'], bins=30, kde=True)
plt.title('Histogram of Age')
plt.xlabel('Age')
```

```
plt.ylabel('Frequency')
plt.show()
```



In [61]:
```
sns.boxplot(x=df['Age'])
plt.title('Age Distribution')
plt.xlabel('Age')
plt.show()
```



In [62]:
```
# describe Age
df['Age'].describe()
```

Out[62]:
```
count    1221.000000
mean       32.320229
std        12.702839
min       -29.000000
25%        27.000000
50%        31.000000
75%        36.000000
max       329.000000
Name: Age, dtype: float64
```

Are there any outliers? If so, do any seem to be issues of accuracy? How might we address those?

Yes there are outliers, we could get rid of any above 100, potentially divide them by 10

In [63]:
```
# show the index number of the outlier data points for Age
Q1 = df['Age'].quantile(0.25)
Q3 = df['Age'].quantile(0.75)
IQR = Q3 - Q1

lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

outliers = df[df['Age'] < lower_bound].index.tolist() + df[df['Age'] > upper_bound]
```

```
outlier_df = df.loc[outliers]

# display outliers
outlier_df.head()
```

Out[63]:

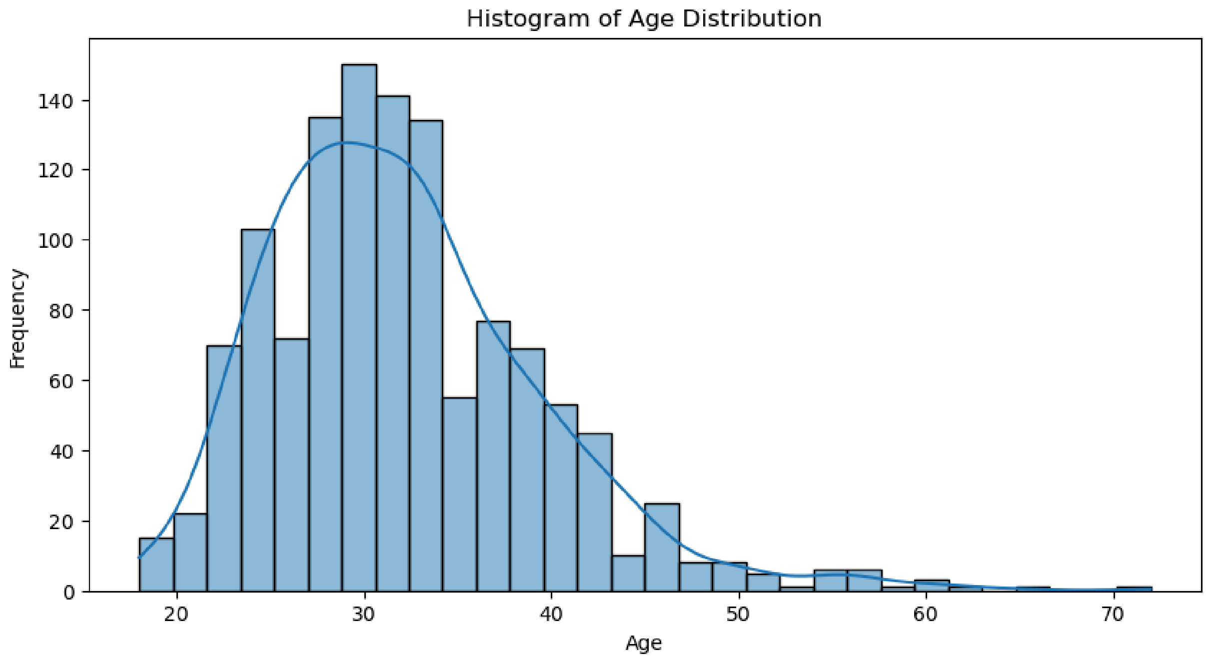
	SurveyID	Age	Gender	Country	state	self_employed	family_history	treatment
142	143	-29.0	Male	United States	MN	No	No	No
727	728	5.0	Male	United States	OH	No	No	No
979	980	8.0	A little about you	Bahamas, The	IL	Yes	Yes	Yes
1079	1080	11.0	male	United States	OH	Yes	No	No
1116	1117	-1.0	p	United States	AL	Yes	Yes	Yes

In [64]:
```
df['Age'].replace({-29: 29, 228: 28, 329: 29}, inplace=True) # in the Age column, r

df['Age'].replace([5, 8, 11, -1], np.nan, inplace=True) # in the Age column, replac
```

```
C:\Users\tucke\AppData\Local\Temp\ipykernel_43740\4079309046.py:1: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace on the original object.


  df['Age'].replace({-29: 29, 228: 28, 329: 29}, inplace=True) # in the Age column, replace -29, 228 and 329 with 29, 28, and 29 respectively
C:\Users\tucke\AppData\Local\Temp\ipykernel_43740\4079309046.py:3: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace on the original object.


  df['Age'].replace([5, 8, 11, -1], np.nan, inplace=True) # in the Age column, replace the observations where the value is just 5, 8, 11, and -1 with NaN
```

In [65]:

```
# view hist
plt.figure(figsize=(10, 5))
sns.histplot(df['Age'], bins=30, kde=True)

plt.title('Histogram of Age Distribution')
plt.xlabel('Age')
plt.ylabel('Frequency')

plt.show()
```



With the extreme outliers that were errors replaced or converted to missing values, should we impute using the mean, median, or most frequent (mode) value?

median

Simple Imputation with the Median Value of Age

In [66]:
```
# Display missing values before imputation
print("Missing values before imputation:")
print(df['Age'].isnull().sum())

imputer = SimpleImputer(strategy='median')

dfi = df.copy()

dfi[['Age']] = imputer.fit_transform(dfi[['Age']])

print("Missing values after imputation:")
print(dfi['Age'].isnull().sum())
```

```
Missing values before imputation:
31
Missing values after imputation:
0
```

In [67]:
```
# preview dfi
dfi
```

Out[67]:

	SurveyID	Age	Gender	Country	state	self_employed	family_history	treatment
0	1	37.0	Female	United States	IL	No	No	Yes
1	2	44.0	M	United States	IN	No	No	No
2	3	32.0	Male	Canada	NaN	No	No	No
3	4	31.0	Male	United Kingdom	NaN	No	Yes	Yes
4	5	31.0	Male	United States	TX	No	No	No
...	...	...	...	...	...	...	...	...
1243	1244	31.0	male	United Kingdom	NaN	No	No	Yes
1244	1245	32.0	Male	United States	IL	No	Yes	Yes
1245	1246	34.0	male	United States	CA	No	Yes	Yes
1246	1247	46.0	f	United States	NC	No	No	No
1247	1248	25.0	Male	United States	IL	No	Yes	Yes

1248 rows × 19 columns

Are there any potential issues with this simple method of imputation based on measures of central tendency?

Yes!

Multiple Imputation using `Iterative Imputer`

In [68]:

```
# view columns from df
df.columns
```

Out[68]:
```
Index(['SurveyID', 'Age', 'Gender', 'Country', 'state', 'self_employed',
       'family_history', 'treatment', 'work_interfere', 'no_employees',
       'remote_work', 'tech_company', 'benefits', 'care_options',
       'wellness_program', 'seek_help', 'comments', 'missing_age',
       'missing_age_mask'],
      dtype='object')
```

In [69]:
```
df['from_US'] = [1 if country == "United States" else 0 for country in df['Country'
df['is_male'] = [1 if Gender == "Male" else 0 for Gender in df['Gender']]
```

In [70]:
```
# Display missing values before imputation
dfm = df.copy()

features_for_imputation = ['work_interfere', 'no_employees', 'benefits']

for col in features_for_imputation:
    if dfm[col].dtype == 'object':
        dfm[col] = dfm[col].astype('category').cat.codes

iter_imputer = IterativeImputer(max_iter=3, random_state=0)
dfm[['Age']] = iter_imputer.fit_transform(dfm[['Age'] + features_for_imputation])[:

print("Missing values after imputation:", dfm['Age'].isnull().sum())
```

```
Missing values after imputation: 0
```

In [71]:
```
# check dfm
dfm
```

Out[71]:

	SurveyID	Age	Gender	Country	state	self_employed	family_history	treatm
0	1	37.000000	Female	United States	IL	No	No	
1	2	44.000000	M	United States	IN	No	No	
2	3	32.000000	Male	Canada	NaN	No	No	
3	4	32.159753	Male	United Kingdom	NaN	No	Yes	
4	5	31.000000	Male	United States	TX	No	No	
...	...	...	...	...	...	...	...	
1243	1244	31.952622	male	United Kingdom	NaN	No	No	
1244	1245	32.000000	Male	United States	IL	No	Yes	
1245	1246	34.000000	male	United States	CA	No	Yes	
1246	1247	46.000000	f	United States	NC	No	No	
1247	1248	25.000000	Male	United States	IL	No	Yes	

1248 rows × 21 columns

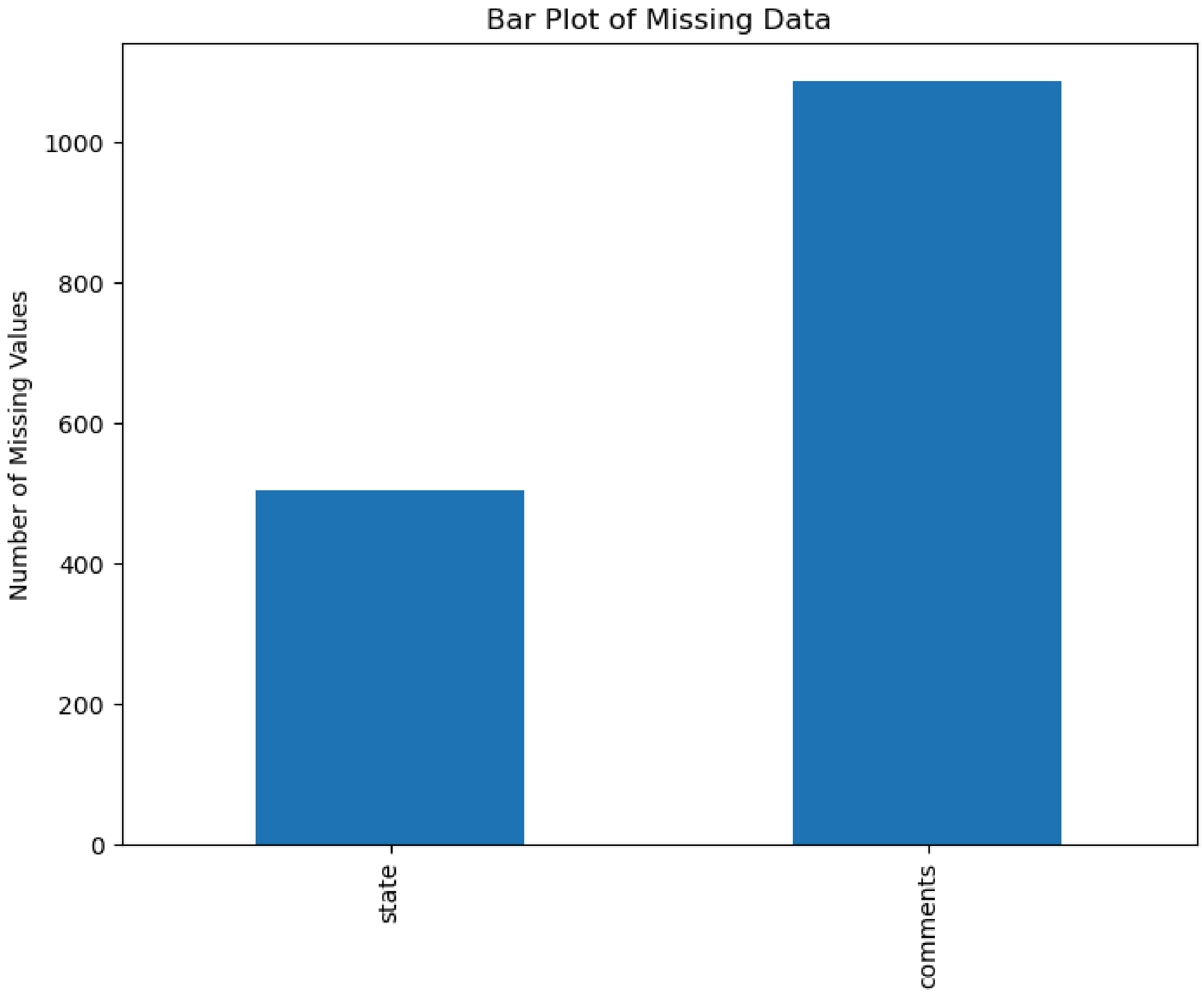
Compare the imputed values between the two methods...How do they differ?

In [72]:

```
comparison_df = pd.DataFrame({
    'Simple Imputation': dfi.loc[df['Age'].isnull(), 'Age'],
    'Multiple Imputation': dfm.loc[df['Age'].isnull(), 'Age']
})

print(comparison_df)
```

	Simple Imputation	Multiple Imputation
3	31.0	32.159753
16	31.0	30.735270
37	31.0	32.003896
60	31.0	30.808665
92	31.0	33.000768
106	31.0	31.793390
114	31.0	30.801077
134	31.0	31.872372
169	31.0	33.249892
185	31.0	30.734954
193	31.0	32.055192
215	31.0	33.043965
272	31.0	32.090349
327	31.0	31.990190
359	31.0	32.039805
380	31.0	32.168917
387	31.0	32.043300
413	31.0	30.815144
417	31.0	33.278984
473	31.0	32.068465
474	31.0	31.021462
482	31.0	31.912194
538	31.0	30.710380
727	31.0	30.946743
731	31.0	31.997891
773	31.0	30.756282
979	31.0	32.961405
1079	31.0	31.973103
1116	31.0	33.080228
1150	31.0	33.143311
1243	31.0	31.952622

In [73]:
```
df['Age'] = dfm['Age']# overwrite existing age column with imputed age column

missing_data = df.isnull().sum()
plt.figure(figsize=(8, 6))
missing_data[missing_data > 0].plot(kind='bar')
plt.title('Bar Plot of Missing Data')
plt.ylabel('Number of Missing Values')
plt.show()
```



Woo! All our missing data (apart form comments) has been handled! Now let's do one final check for outliers on our remaining quantitative column...

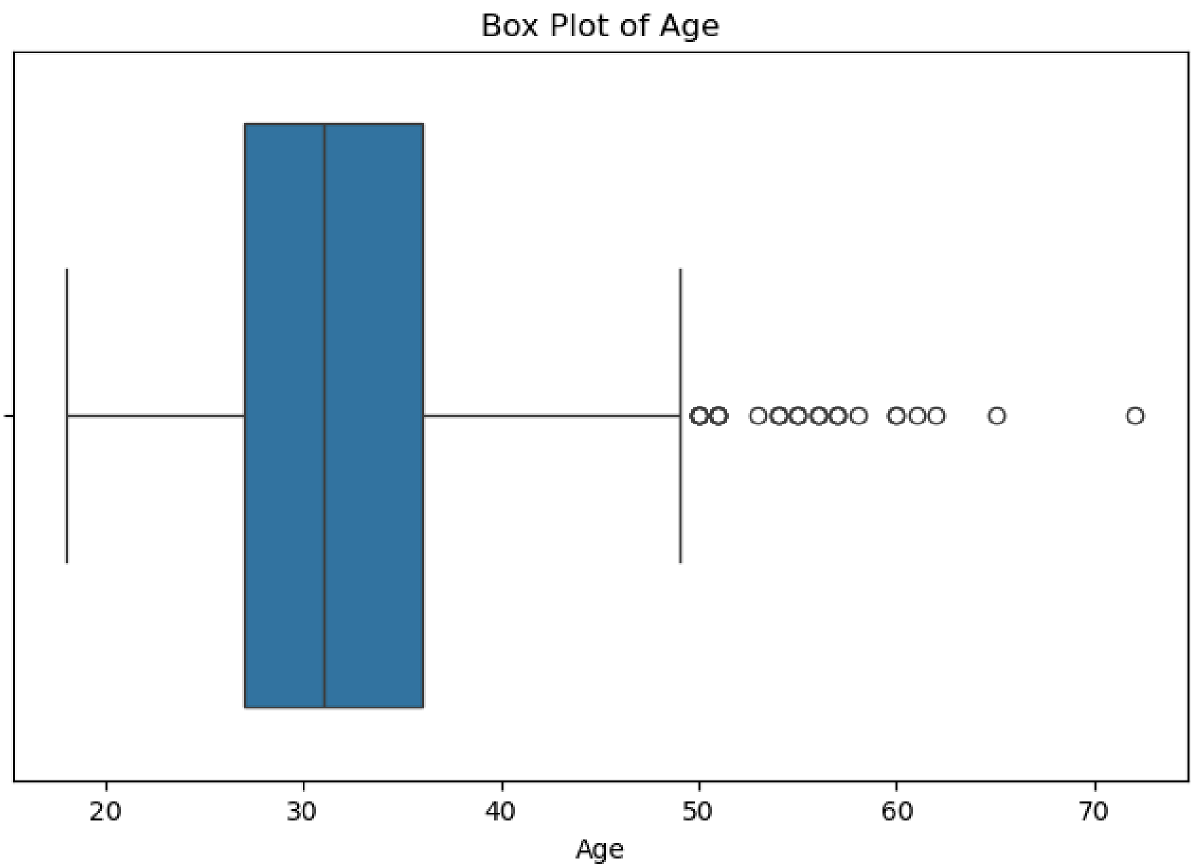
In [74]:
```
# describe variable
print(df['Age'].describe())
```

```
count    1248.000000
mean       32.041034
std         7.156150
min        18.000000
25%        27.000000
50%        31.000000
75%        36.000000
max        72.000000
Name: Age, dtype: float64
```

Are there any likely outliers, how do you know?

In [ ]:
```
plt.figure(figsize=(8, 5))
sns.boxplot(x=df['Age'])

plt.title('Box Plot of Age')
plt.xlabel('Age')
```

```
plt.show()
```



Technically yes but they are within reason

In [76]:
```
Q1 = df['no_employees'].quantile(0.25)
Q3 = df['no_employees'].quantile(0.75)
IQR = Q3 - Q1

upper_bound = Q3 + 1.5 * IQR

outliers = df[df['no_employees'] > upper_bound].index.tolist()

outlier_df = df.loc[outliers, :]

outlier_df
```

Out[76]:

	SurveyID	Age	Gender	Country	state	self_employed	family_history	treatment	wo
24	25	33.0	male	United States	CA	No	Yes	Yes	
155	156	27.0	Male	United States	CA	No	Yes	Yes	
351	352	26.0	male	United States	CA	No	Yes	Yes	
484	485	30.0	Male	United States	CA	Yes	Yes	Yes	

Looking at these outliers and their other information, do they appear to be inaccurate or true, albeit extreme, data points?

Looks good, I would replace the NaN in comments with a null

Transforming Outliers

In [77]:

```
# prompt: perform a log transformation on the no_employees column

df['no_employees_lg'] = np.log1p(df['no_employees'])  # log1p(x) = log(x + 1)

plt.figure(figsize=(8, 5))
df['no_employees_lg'].hist(bins=30, color='steelblue', edgecolor='black')

plt.title('Histogram of Log-Transformed no_employees')
plt.xlabel('log(no_employees)')
plt.ylabel('Frequency')

plt.show()
```

Binning Outliers

In [79]:
```
bins = [0, 10, 50, 100, 500, 1000, df['no_employees'].max()]
labels = ['1-10', '11-50', '51-100', '101-500', '501-1000', '1000+']
df['no_employee_cats'] = pd.cut(df['no_employees'], bins=bins, labels=labels) # dis
df.head()
```

Out[79]:

	SurveyID	Age	Gender	Country	state	self_employed	family_history	treatment
0	1	37.000000	Female	United States	IL	No	No	Yes
1	2	44.000000	M	United States	IN	No	No	No
2	3	32.000000	Male	Canada	NaN	No	No	No
3	4	32.159753	Male	United Kingdom	NaN	No	Yes	Yes
4	5	31.000000	Male	United States	TX	No	No	No

In [ ]:
```
# drop remaining columns not needed
# check for missing values
columns_to_drop = ['SurveyID']  # Add more columns if necessary
df.drop(columns=columns_to_drop, inplace=True)
```

```
missing_values = df.isnull().sum()

print("Missing values after cleaning:")
print(missing_values[missing_values > 0])
```

```
Missing values after cleaning:
state        504
comments    1086
dtype: int64
```

Why is it important we take steps to address missing data and outliers instead of just removing them?

Removing datapoints can be great and should often be done more often than taking the mean as the mean can skew the data more. BY removing data we can remove other fields that are valuable

Now go forth with your improved dataset...

In [83]:
```
df.to_csv('final.csv') # write to csv
```