DSC 680

Inman, Gracie

Project 2

04/28/24

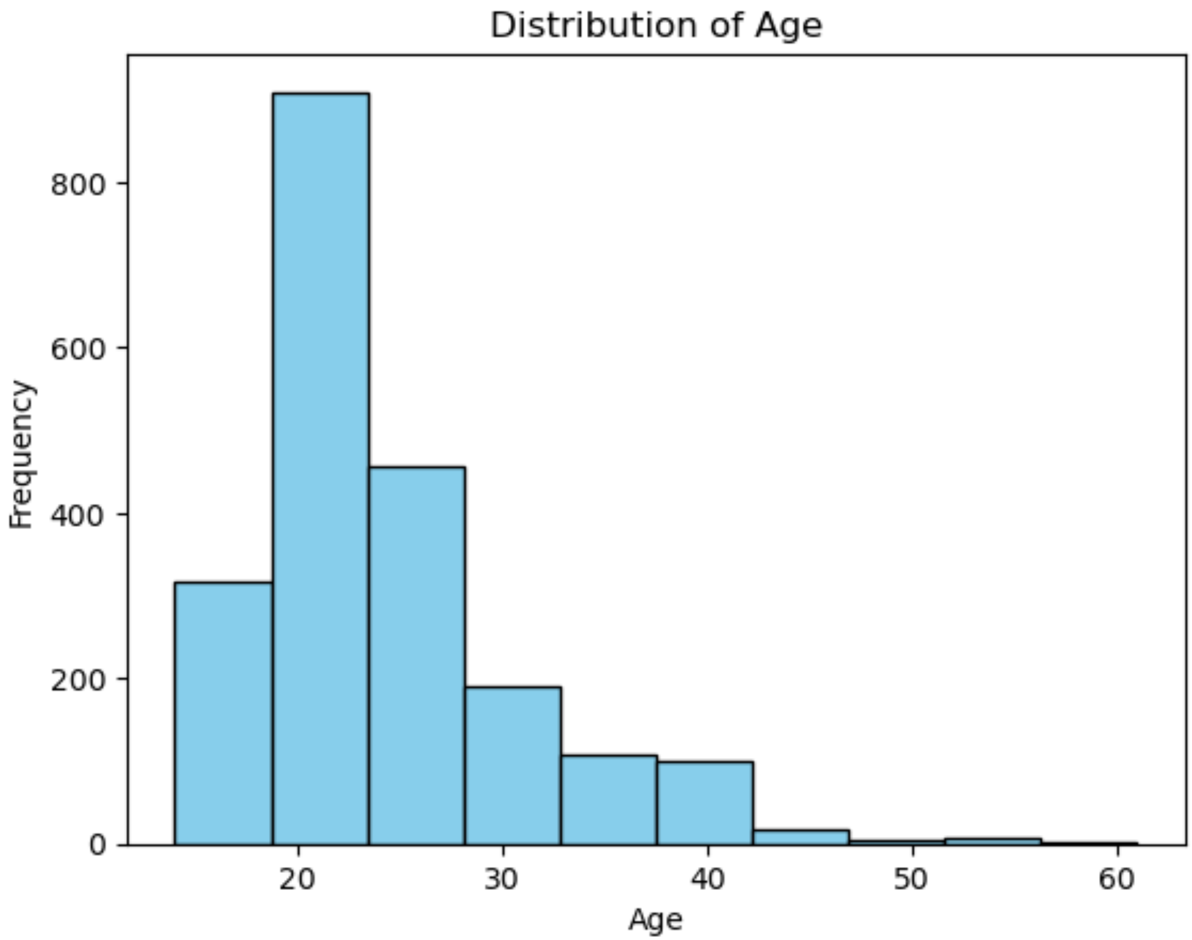
In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```
obesity = pd.read_csv("ObesityDataSet_raw_and_data_sinthetic.csv")
obesity.head()
```

Out[2]:

	Age	Gender	Height	Weight	CALC	FAVC	FCVC	NCP	SCC	SMOKE	CH2O	family_history_with_overweight	FAF	TU
0	21.0	Female	1.62	64.0	no	no	2.0	3.0	no	no	2.0	yes	0.0	1
1	21.0	Female	1.52	56.0	Sometimes	no	3.0	3.0	yes	yes	3.0	yes	3.0	0
2	23.0	Male	1.80	77.0	Frequently	no	2.0	3.0	no	no	2.0	yes	2.0	1
3	27.0	Male	1.80	87.0	Frequently	no	3.0	3.0	no	no	2.0	no	2.0	0
4	22.0	Male	1.78	89.8	Sometimes	no	2.0	1.0	no	no	2.0	no	0.0	0

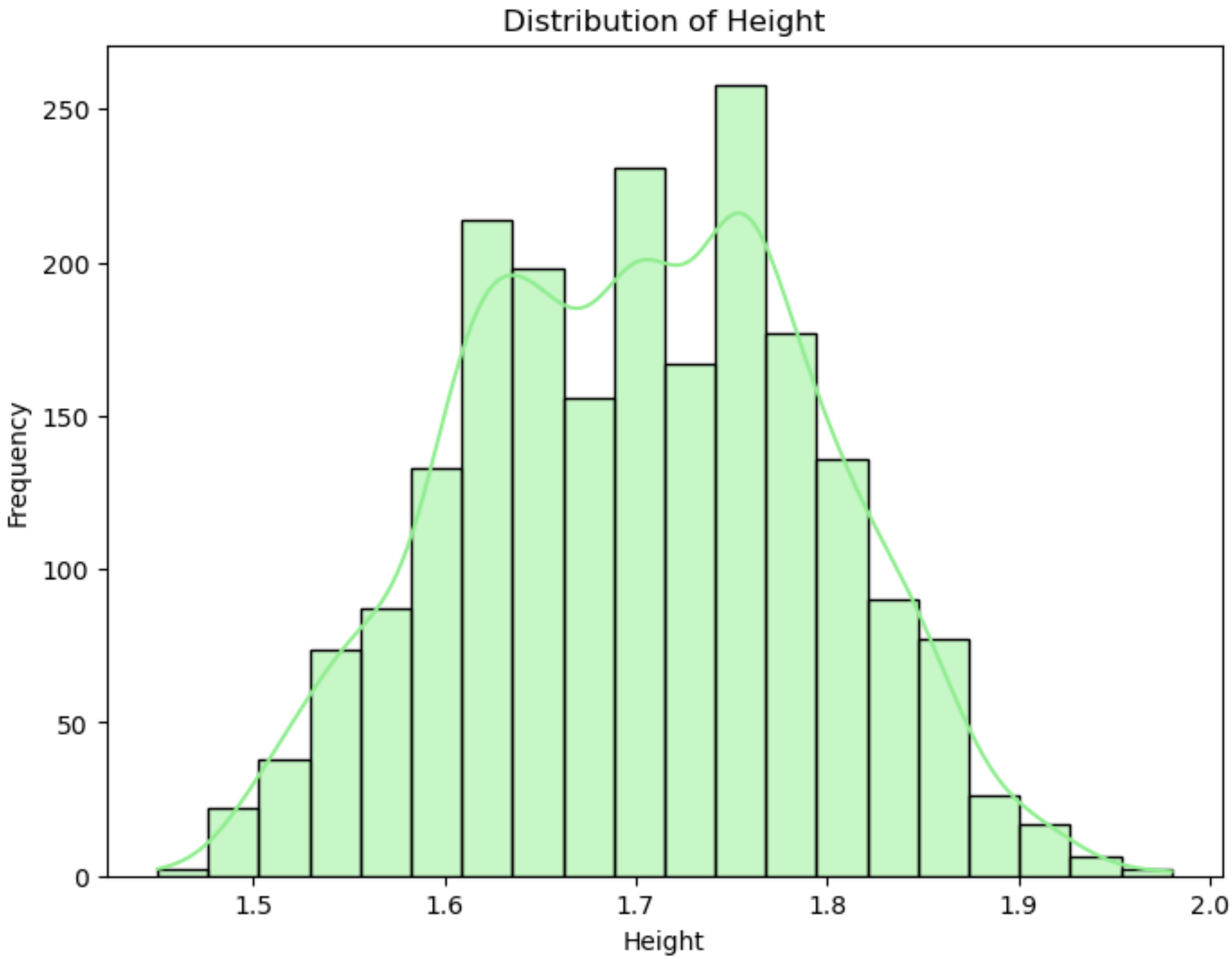
In [3]:
```
plt.hist(obesity['Age'], bins=10, color='skyblue', edgecolor='black')
plt.title('Distribution of Age')
plt.xlabel('Age')
plt.ylabel('Frequency')
plt.show()
```



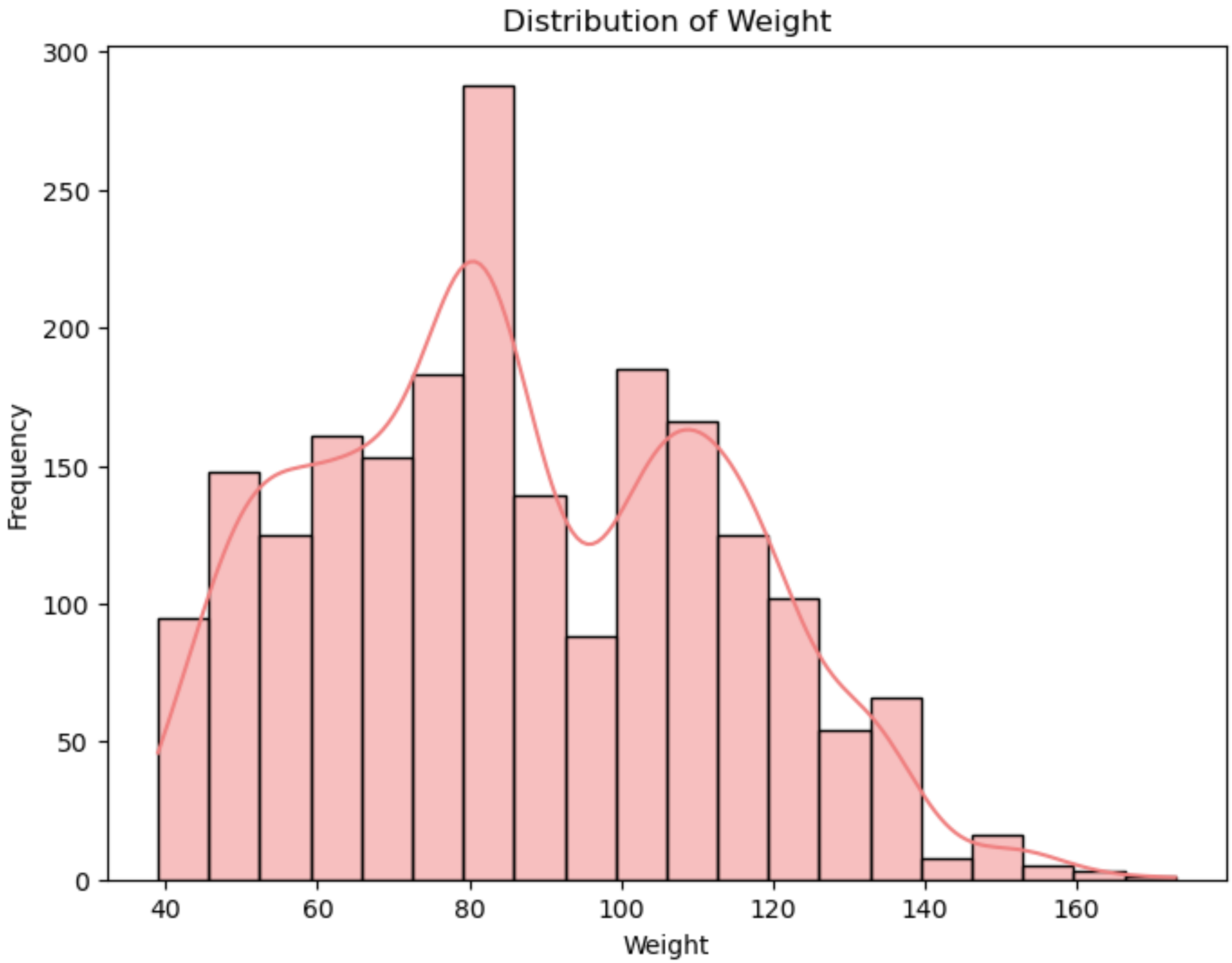
In [4]:
```
plt.figure(figsize=(6, 6))
gender_counts = obesity['Gender'].value_counts()
sns.barplot(x=gender_counts.index, y=gender_counts.values, palette='pastel')
plt.title('Gender Distribution')
plt.xlabel('Gender')
plt.ylabel('Count')
plt.show()
```


Gender Distribution
Count
1000
800
600
400
200
0
Male
Female
Gender

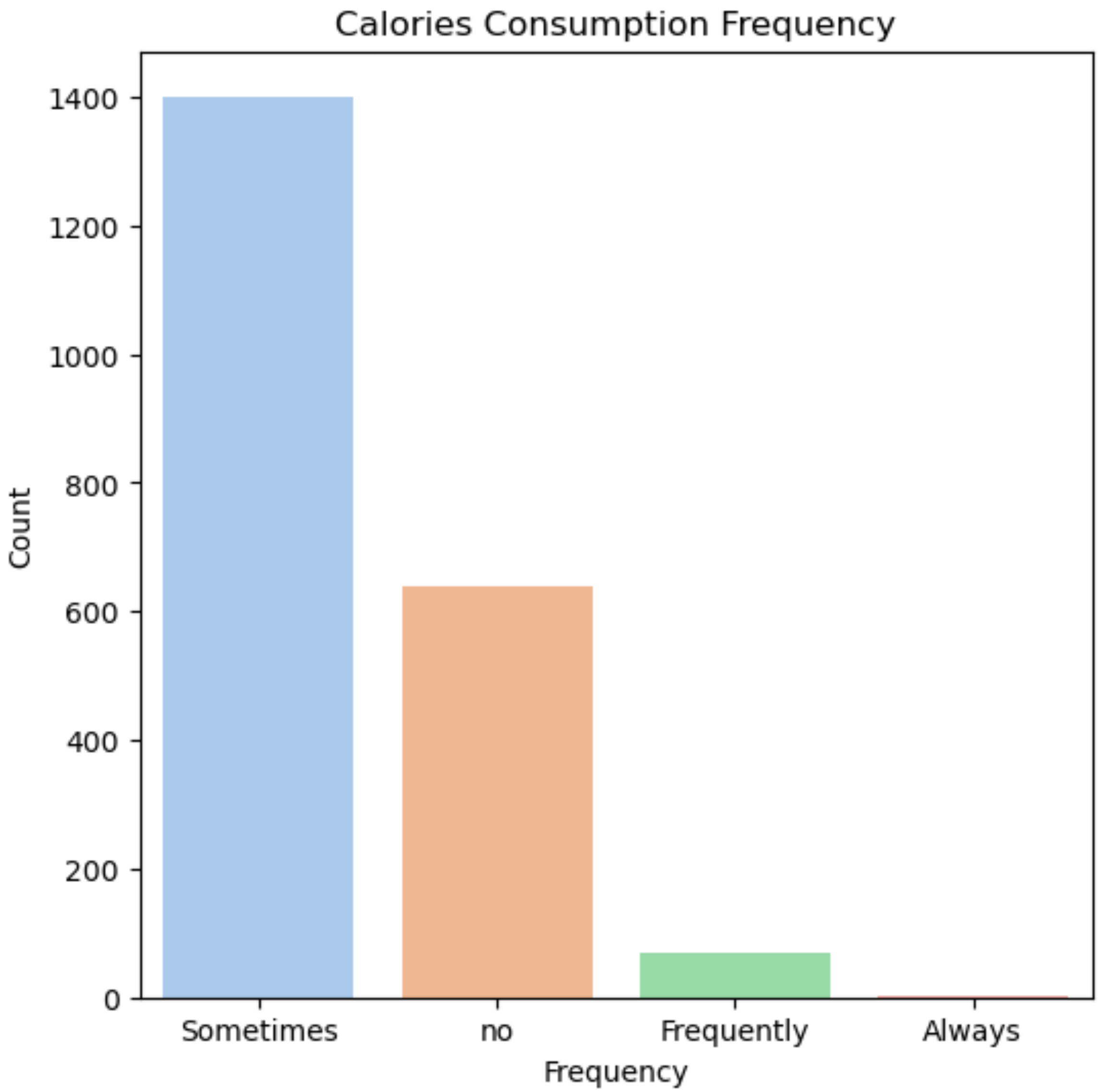
In [5]:
```
plt.figure(figsize=(8, 6))
sns.histplot(obesity['Height'], bins=20, kde=True, color='lightgreen')
plt.title('Distribution of Height')
plt.xlabel('Height')
plt.ylabel('Frequency')
plt.show()
```


Distribution of Height
Frequency
Height
0
50
100
150
200
250
1.5
1.6
1.7
1.8
1.9
2.0

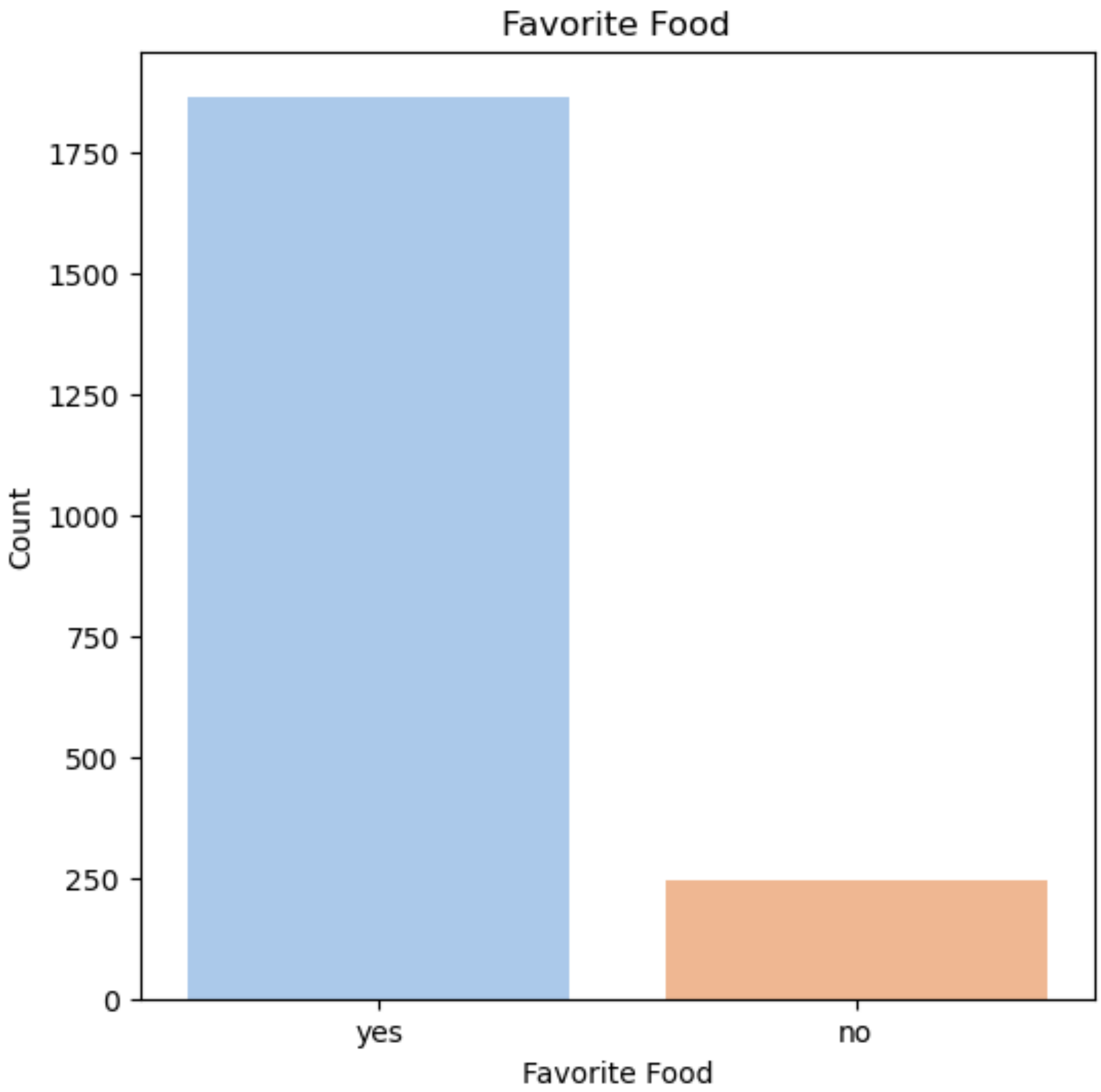
In [6]:
```
plt.figure(figsize=(8, 6))
sns.histplot(obesity['Weight'], bins=20, kde=True, color='lightcoral')
plt.title('Distribution of Weight')
plt.xlabel('Weight')
plt.ylabel('Frequency')
plt.show()
```


Distribution of Weight
Frequency
Weight
0
50
100
150
200
250
300
40
60
80
100
120
140
160

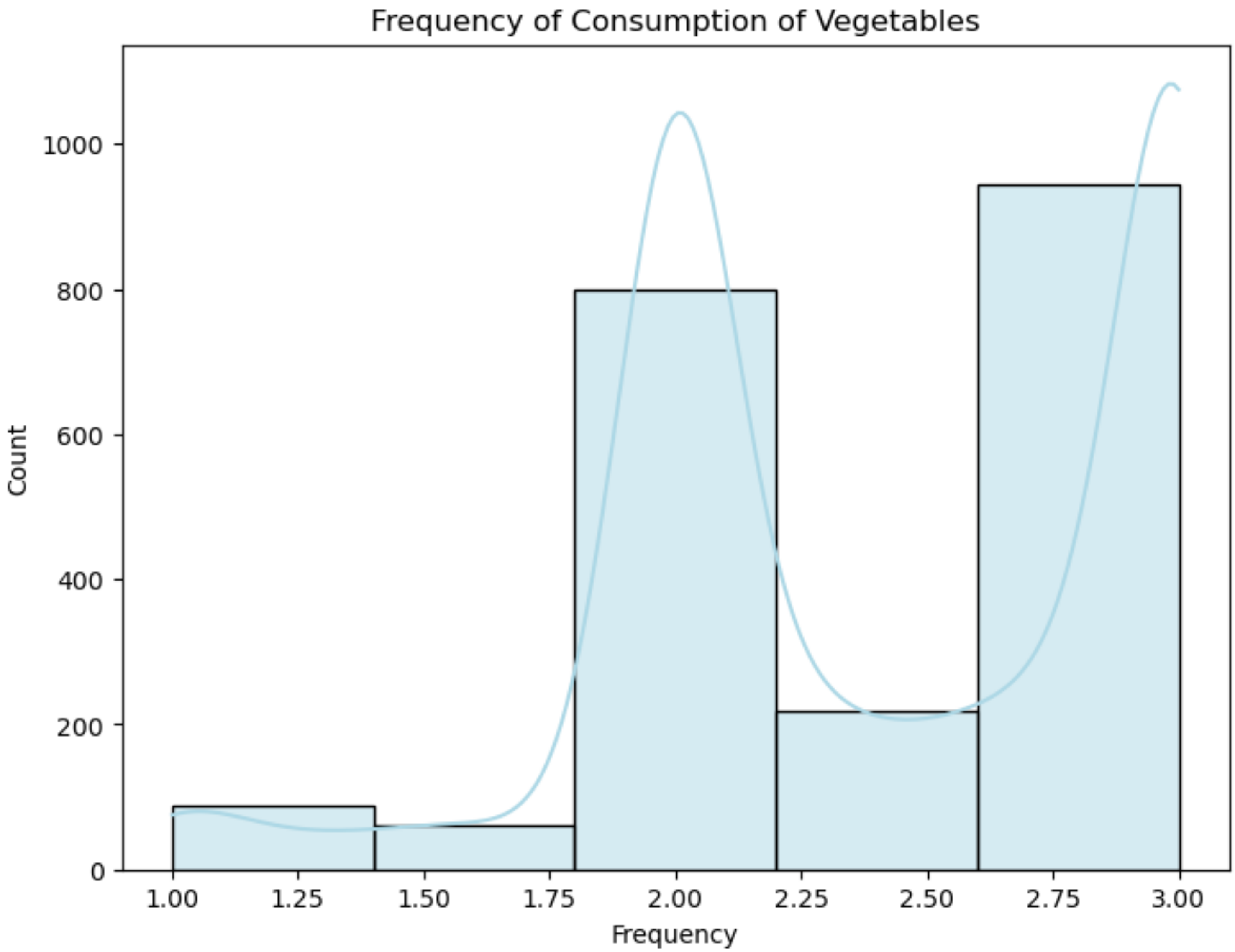
In [7]:
```
plt.figure(figsize=(6, 6))
calc_counts = obesity['CALC'].value_counts()
sns.barplot(x=calc_counts.index, y=calc_counts.values, palette='pastel')
plt.title('Calories Consumption Frequency')
plt.xlabel('Frequency')
plt.ylabel('Count')
plt.show()
```


Calories Consumption Frequency
Count
1400
1200
1000
800
600
400
200
0
Sometimes
no
Frequently
Always
Frequency

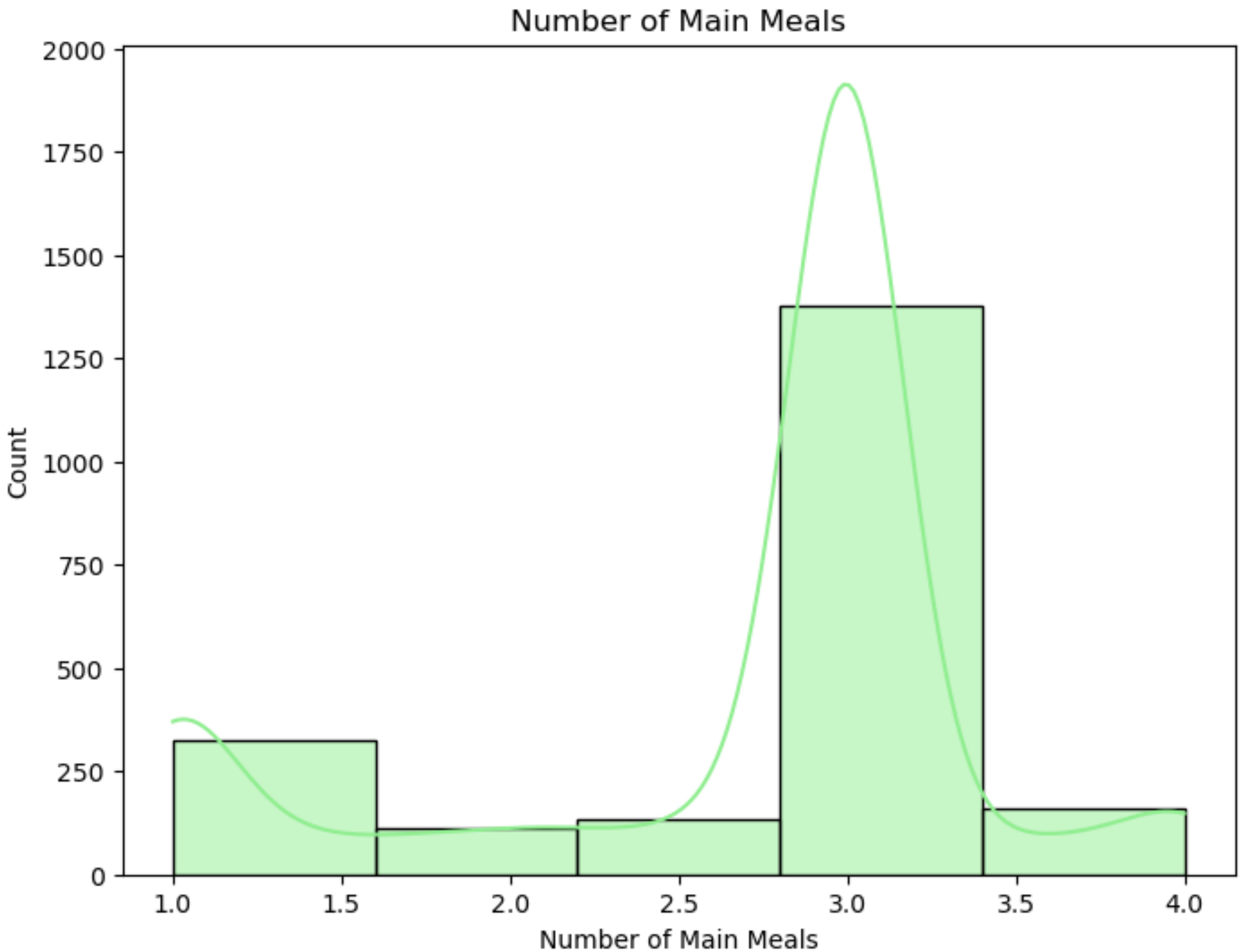
In [8]:
```
plt.figure(figsize=(6, 6))
favc_counts = obesity['FAVC'].value_counts()
sns.barplot(x=favc_counts.index, y=favc_counts.values, palette='pastel')
plt.title('Favorite Food')
plt.xlabel('Favorite Food')
plt.ylabel('Count')
plt.show()
```


Favorite Food
Count
1750
1500
1250
1000
750
500
250
0
yes
no
Favorite Food

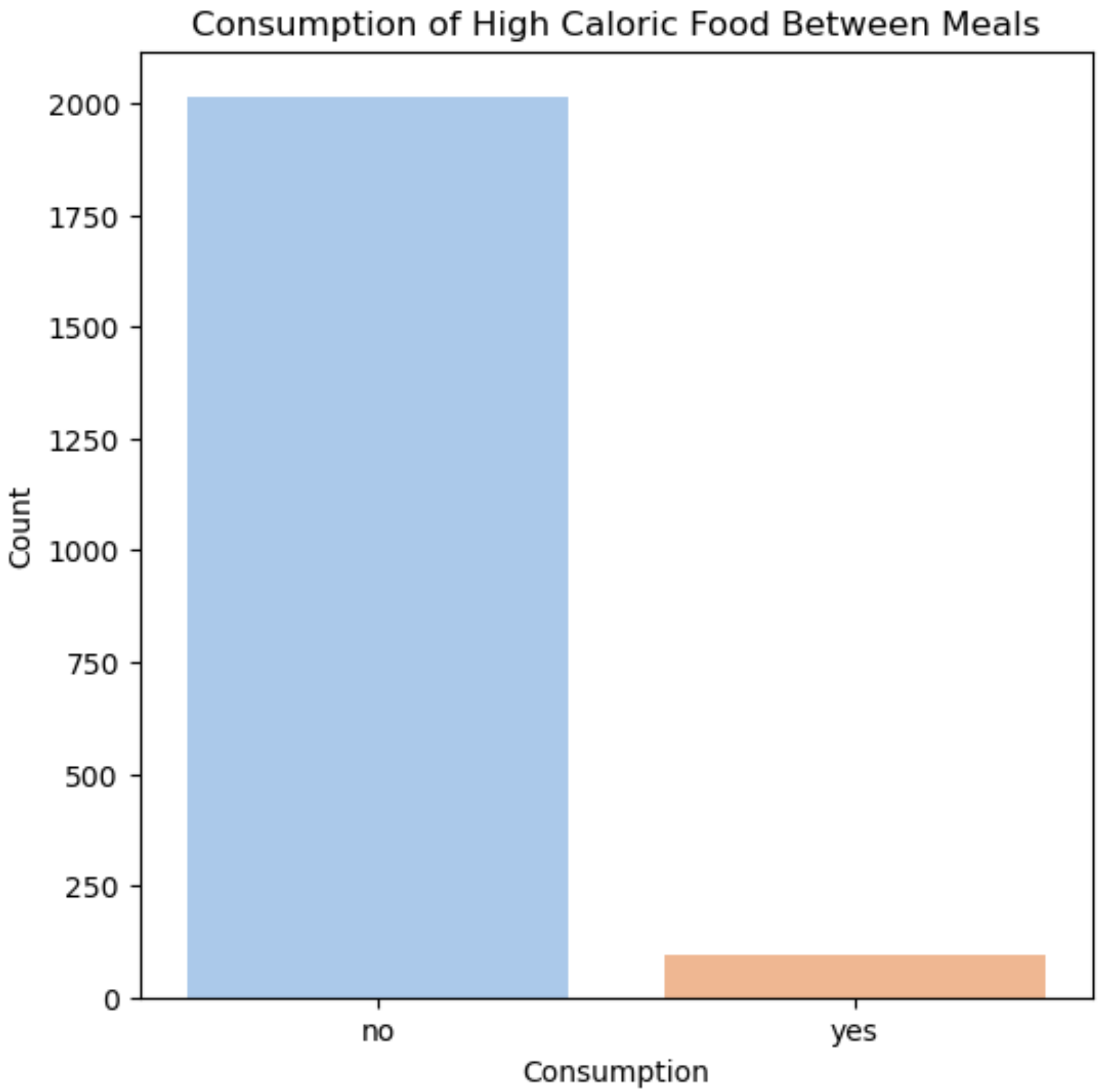
In [9]:
```
plt.figure(figsize=(8, 6))
sns.histplot(obesity['FCVC'], bins=5, kde=True, color='lightblue')
plt.title('Frequency of Consumption of Vegetables')
plt.xlabel('Frequency')
plt.ylabel('Count')
plt.show()
```


Frequency of Consumption of Vegetables
Count
0
200
400
600
800
1000
1.00
1.25
1.50
1.75
2.00
2.25
2.50
2.75
3.00
Frequency

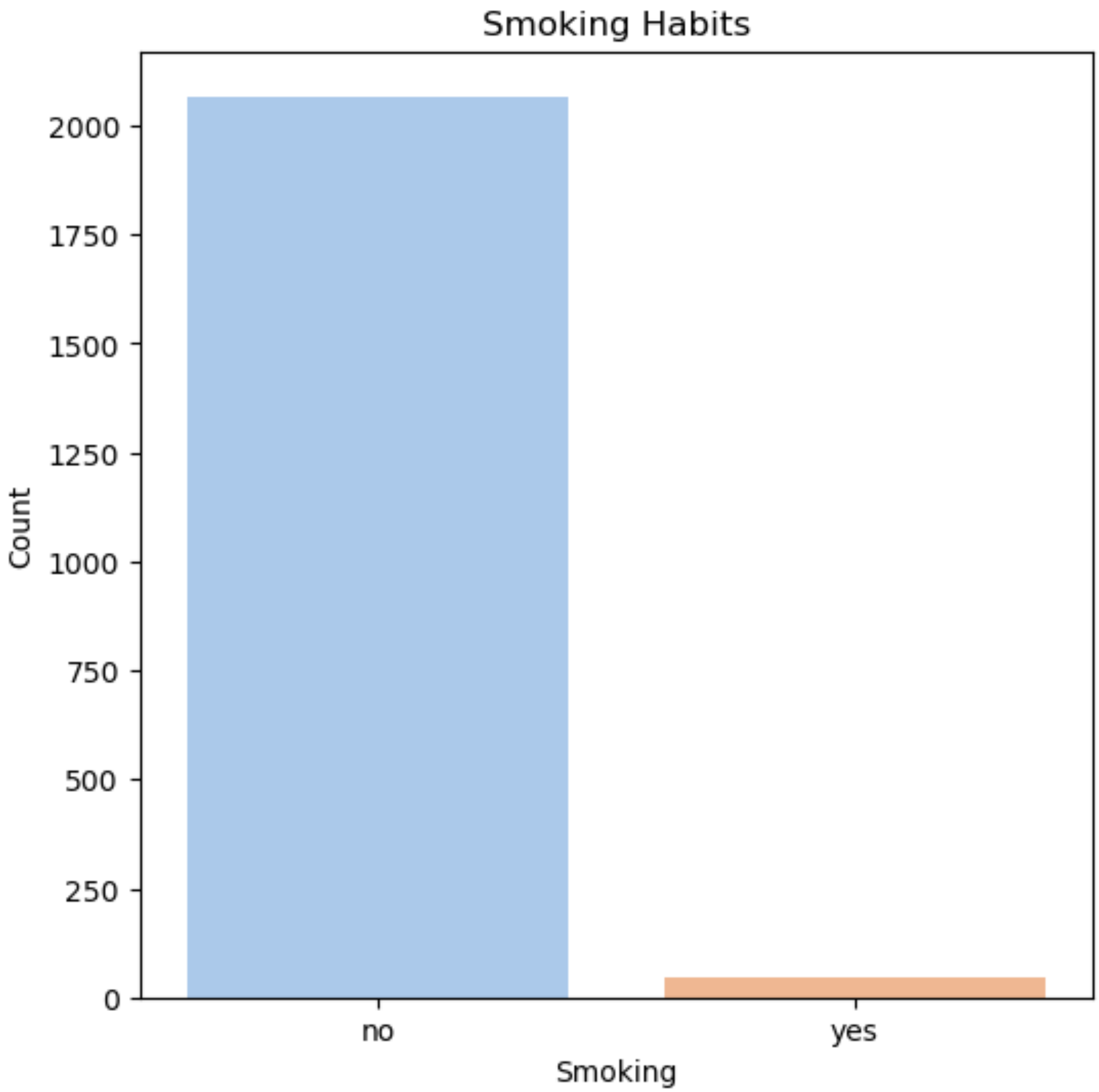
In [10]:
```
plt.figure(figsize=(8, 6))
sns.histplot(obesity['NCP'], bins=5, kde=True, color='lightgreen')
plt.title('Number of Main Meals')
plt.xlabel('Number of Main Meals')
plt.ylabel('Count')
plt.show()
```


Number of Main Meals
2000
1750
1500
1250
1000
750
500
250
0
Count
1.0
1.5
2.0
2.5
3.0
3.5
4.0
Number of Main Meals

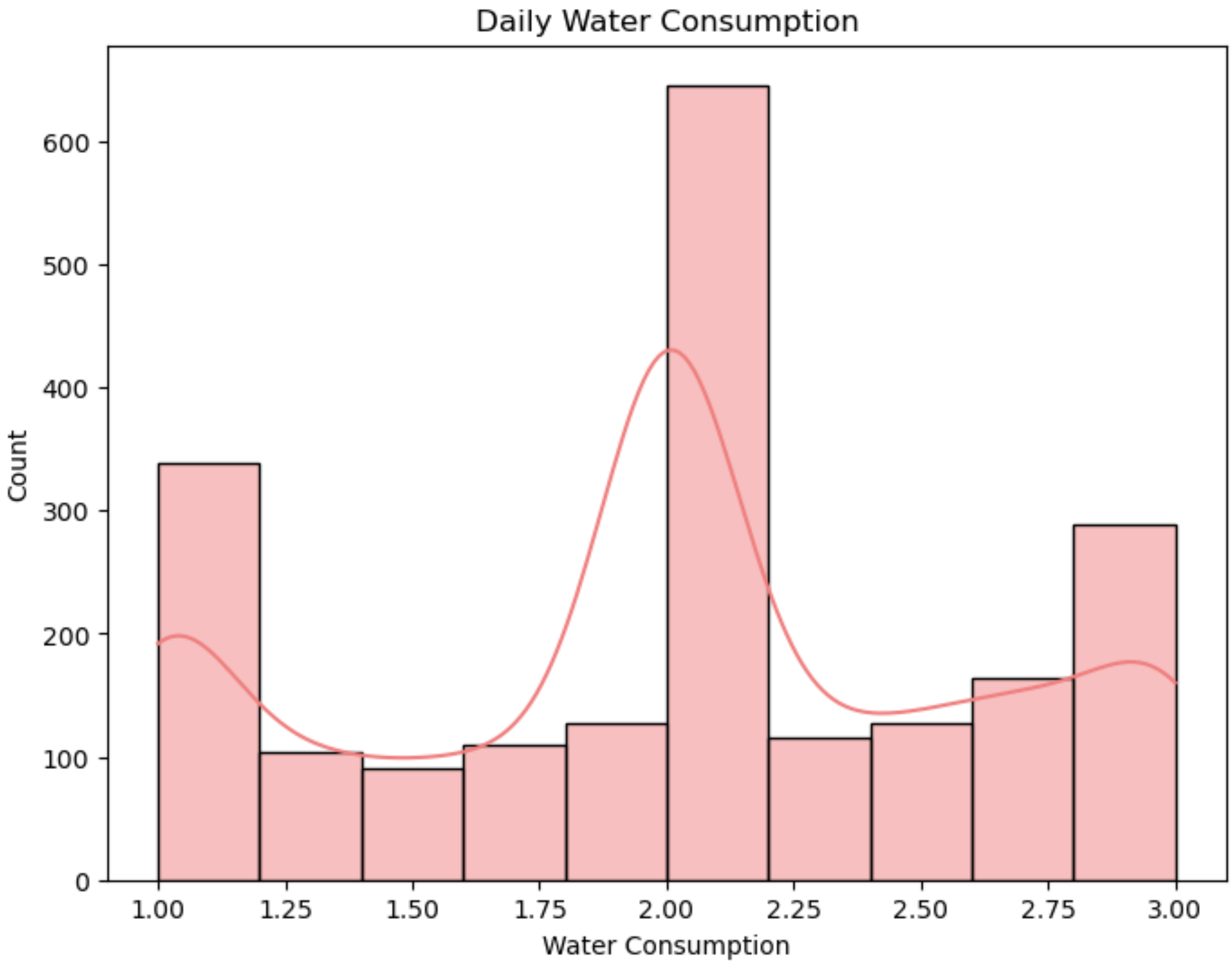
In [11]:
```
plt.figure(figsize=(6, 6))
scc_counts = obesity['SCC'].value_counts()
sns.barplot(x=scc_counts.index, y=scc_counts.values, palette='pastel')
plt.title('Consumption of High Caloric Food Between Meals')
plt.xlabel('Consumption')
plt.ylabel('Count')
plt.show()
```


Consumption of High Caloric Food Between Meals
2000
1750
1500
1250
1000
750
500
250
0
Count
no
yes
Consumption

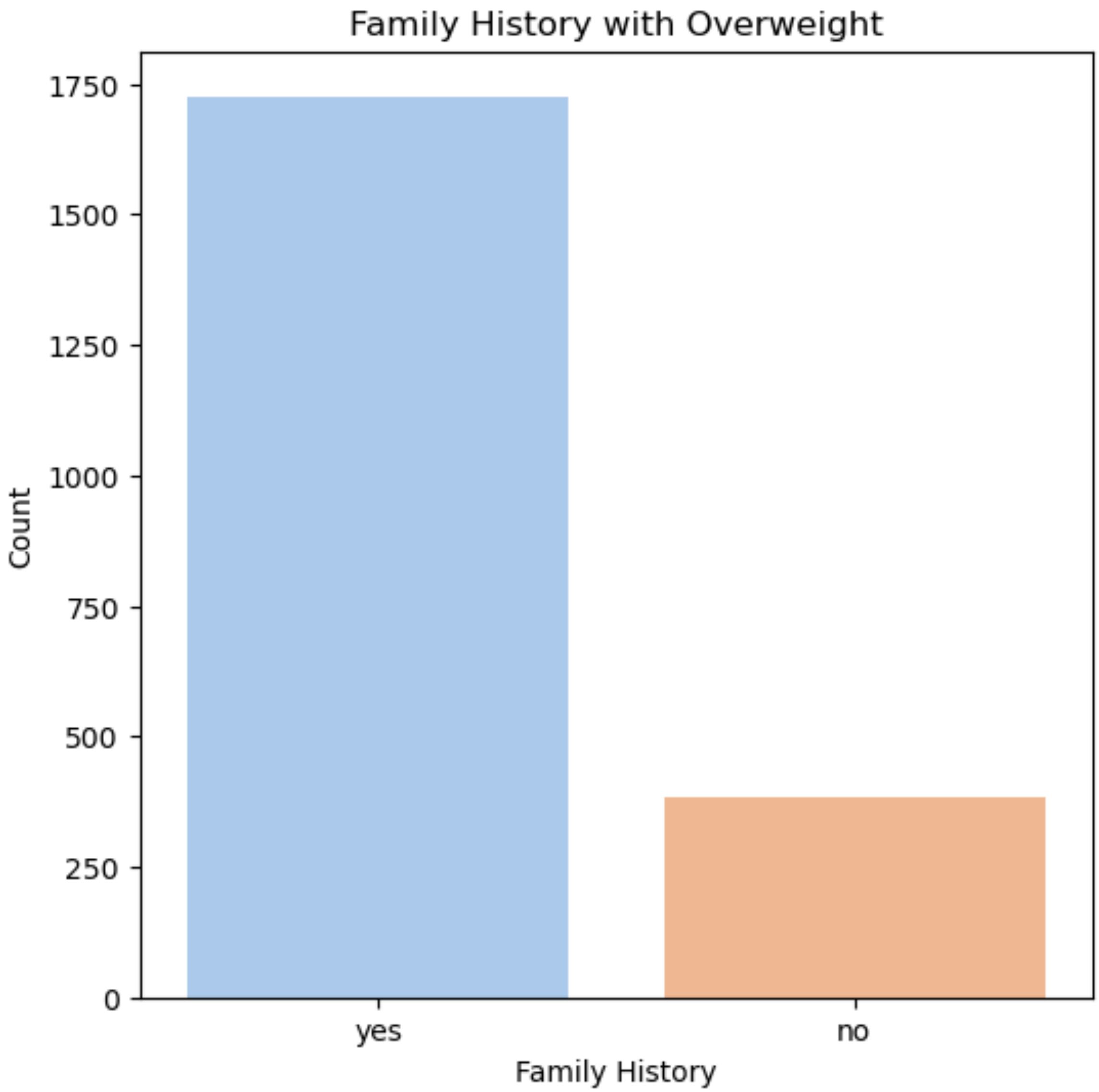
In [12]:
```
plt.figure(figsize=(6, 6))
smoke_counts = obesity['SMOKE'].value_counts()
sns.barplot(x=smoke_counts.index, y=smoke_counts.values, palette='pastel')
plt.title('Smoking Habits')
plt.xlabel('Smoking')
plt.ylabel('Count')
plt.show()
```


Smoking Habits
2000
1750
1500
1250
1000
750
500
250
0
Count
no
yes
Smoking

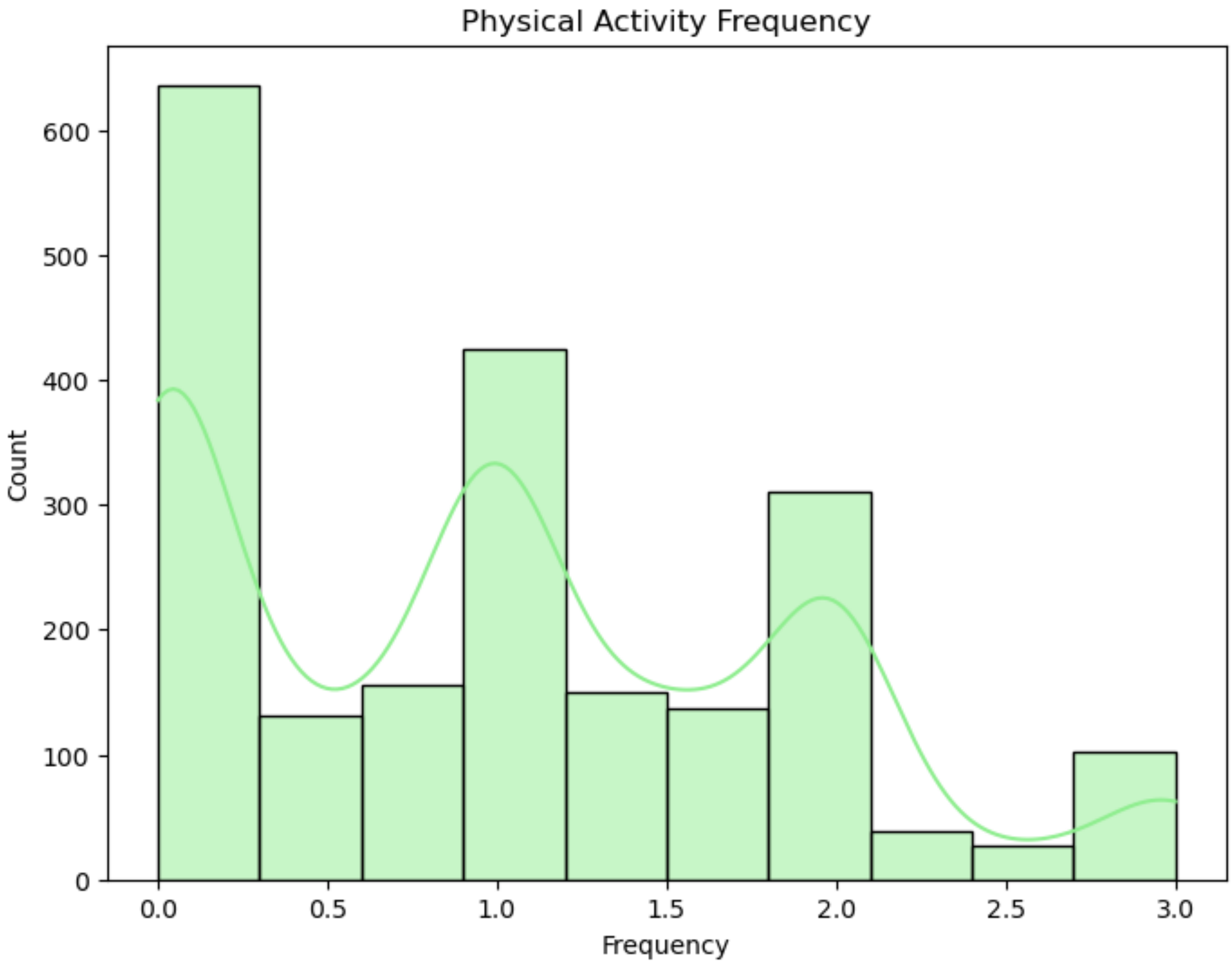
In [13]:
```
plt.figure(figsize=(8, 6))
sns.histplot(obesity['CH2O'], bins=10, kde=True, color='lightcoral')
plt.title('Daily Water Consumption')
plt.xlabel('Water Consumption')
plt.ylabel('Count')
plt.show()
```


Daily Water Consumption
Count
0
100
200
300
400
500
600
1.00
1.25
1.50
1.75
2.00
2.25
2.50
2.75
3.00
Water Consumption

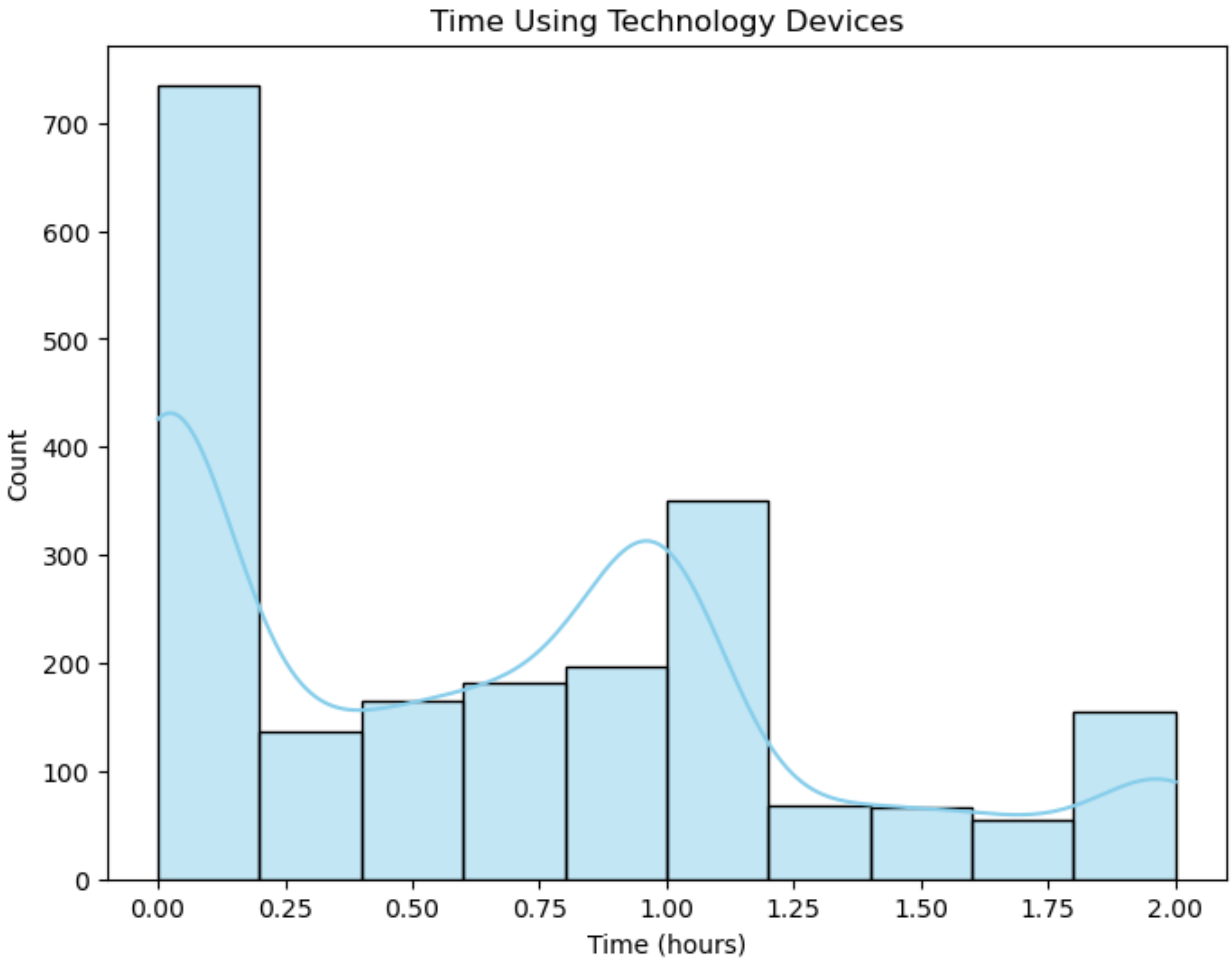
In [14]:
```
plt.figure(figsize=(6, 6))
fhwo_counts = obesity['family_history_with_overweight'].value_counts()
sns.barplot(x=fhwo_counts.index, y=fhwo_counts.values, palette='pastel')
plt.title('Family History with Overweight')
plt.xlabel('Family History')
plt.ylabel('Count')
plt.show()
```


Family History with Overweight
Count
1750
1500
1250
1000
750
500
250
0
yes
no
Family History

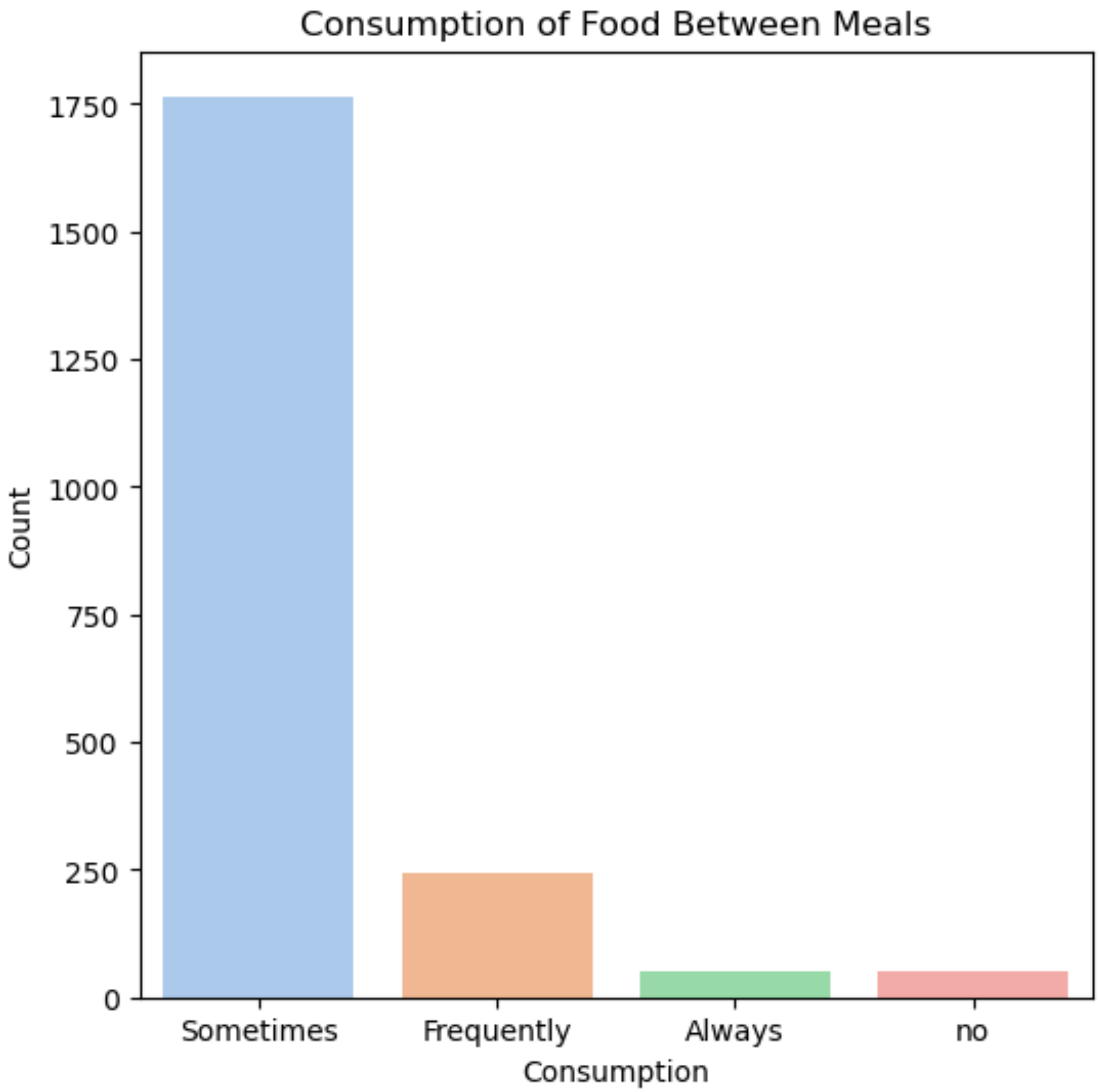
In [15]:
```
plt.figure(figsize=(8, 6))
sns.histplot(obesity['FAF'], bins=10, kde=True, color='lightgreen')
plt.title('Physical Activity Frequency')
plt.xlabel('Frequency')
plt.ylabel('Count')
plt.show()
```


Physical Activity Frequency
Count
600
500
400
300
200
100
0
0.0
0.5
1.0
1.5
2.0
2.5
3.0
Frequency

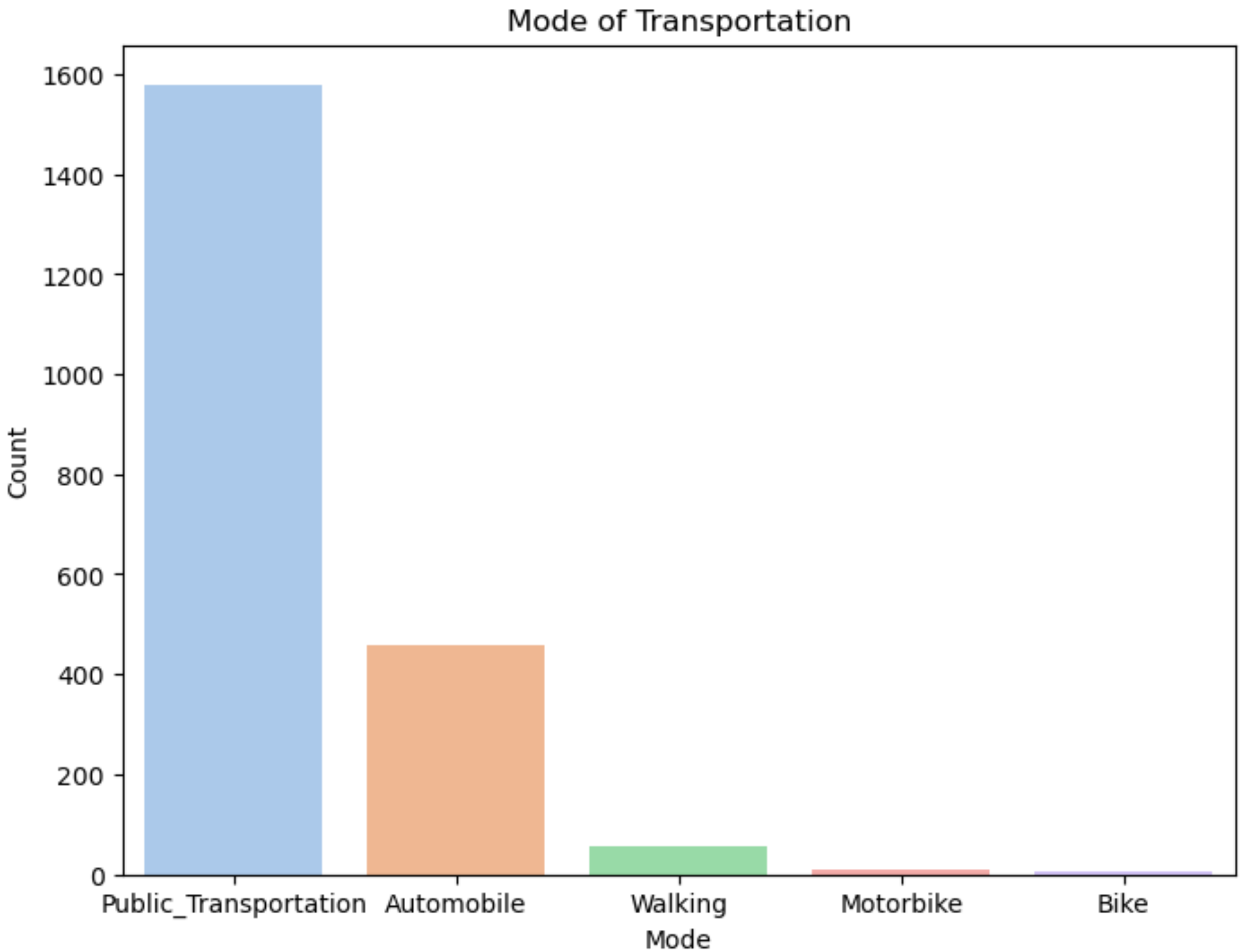
In [16]:
```
plt.figure(figsize=(8, 6))
sns.histplot(obesity['TUE'], bins=10, kde=True, color='skyblue')
plt.title('Time Using Technology Devices')
plt.xlabel('Time (hours)')
plt.ylabel('Count')
plt.show()
```


Time Using Technology Devices
Count
700
600
500
400
300
200
100
0
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
Time (hours)

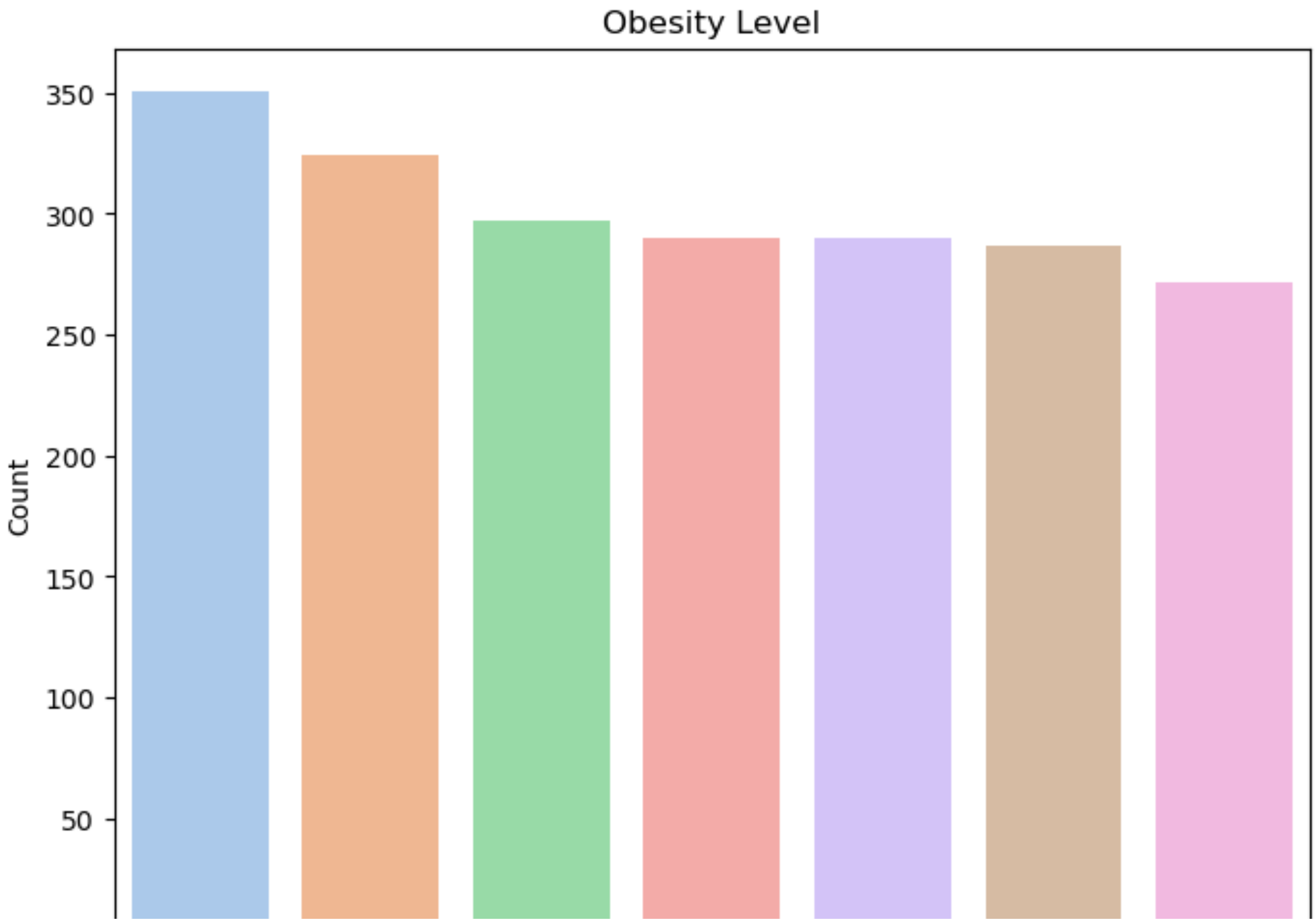
In [17]:
```
plt.figure(figsize=(6, 6))
caec_counts = obesity['CAEC'].value_counts()
sns.barplot(x=caec_counts.index, y=caec_counts.values, palette='pastel')
plt.title('Consumption of Food Between Meals')
plt.xlabel('Consumption')
plt.ylabel('Count')
plt.show()
```

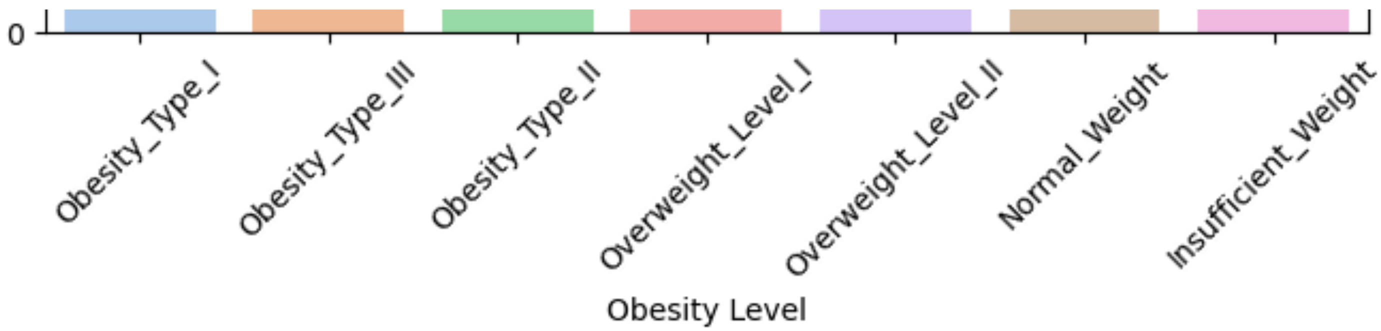

Consumption of Food Between Meals
Count
1750
1500
1250
1000
750
500
250
0
Sometimes
Frequently
Always
no
Consumption

In [18]:
```
plt.figure(figsize=(8, 6))
mtrans_counts = obesity['MTRANS'].value_counts()
sns.barplot(x=mtrans_counts.index, y=mtrans_counts.values, palette='pastel')
plt.title('Mode of Transportation')
plt.xlabel('Mode')
plt.ylabel('Count')
plt.show()
```


Mode of Transportation
1600
1400
1200
1000
800
600
400
200
0
Count
Public_Transportation
Automobile
Walking
Motorbike
Bike
Mode

In [19]:
```
plt.figure(figsize=(8, 6))
obesity_counts = obesity['NObeyesdad'].value_counts()
sns.barplot(x=obesity_counts.index, y=obesity_counts.values, palette='pastel')
plt.title('Obesity Level')
plt.xlabel('Obesity Level')
plt.ylabel('Count')
plt.xticks(rotation=45)
plt.show()
```





In [20]:
```
# Missing Values
missing_values = obesity.isnull().sum()

print("Columns with missing values:")
print(missing_values[missing_values > 0])
```

```
Columns with missing values:
Series([], dtype: int64)
```

In [21]:
```
# Unique values in categorical columns
non_numeric_columns = obesity.select_dtypes(exclude=['number']).columns
for column in non_numeric_columns:
    unique_values = obesity[column].unique()
    print(f"Unique values in '{column}'column:")
    print(unique_values)
    print()
```

```
Unique values in 'Gender'column:
['Female' 'Male']

Unique values in 'CALC'column:
['no' 'Sometimes' 'Frequently' 'Always']

Unique values in 'FAVC'column:
['no' 'yes']

Unique values in 'SCC'column:
['no' 'yes']

Unique values in 'SMOKE'column:
['no' 'yes']

Unique values in 'family_history_with_overweight'column:
['yes' 'no']

Unique values in 'CAEC'column:
['Sometimes' 'Frequently' 'Always' 'no']

Unique values in 'MTRANS'column:
['Public_Transportation' 'Walking' 'Automobile' 'Motorbike' 'Bike']

Unique values in 'NObeyesdad'column:
['Normal_Weight' 'Overweight_Level_I' 'Overweight_Level_II'
 'Obesity_Type_I' 'Insufficient_Weight' 'Obesity_Type_II'
 'Obesity_Type_III']
```

In [22]:
```
# Combine overweight and obesity variables into one category
overweight_categories = ['Overweight_Level_I', 'Overweight_Level_II']
obesity_categories = ['Obesity_Type_I', 'Obesity_Type_II', 'Obesity_Type_III']

# Replace overweight and obesity categories
obesity['NObeyesdad'] = obesity['NObeyesdad'].replace(overweight_categories, 'Overweight')
obesity['NObeyesdad'] = obesity['NObeyesdad'].replace(obesity_categories, 'Obesity')

# Verify
print(obesity['NObeyesdad'].value_counts())
```

```
Obesity                972
Overweight             580
Normal_Weight          287
Insufficient_Weight    272
Name: NObeyesdad, dtype: int64
```

In [23]:
```
# Encode categorical variables
from sklearn.preprocessing import LabelEncoder

label_encoders = {}

for column in non_numeric_columns:
    label_encoder = LabelEncoder()
    obesity[column] = label_encoder.fit_transform(obesity[column])  # Fit and transform the column
    label_encoders[column] = label_encoder  # Store the label encoder for each column

# Print the encoded values and their corresponding original values for each column
for column, encoder in label_encoders.items():
    encoded_values = obesity[column].unique()
    original_values = encoder.inverse_transform(encoded_values)  # Use the label encoder to inverse trans
    print(f"Encoded values in '{column}' column:")
    print(encoded_values)
    print("Original values:")
    print(original_values)
    print()
```

```
Encoded values in 'Gender' column:
[0 1]
```

```
Original values:
['Female' 'Male']

Encoded values in 'CALC' column:
[3 2 1 0]
Original values:
['no' 'Sometimes' 'Frequently' 'Always']

Encoded values in 'FAVC' column:
[0 1]
Original values:
['no' 'yes']

Encoded values in 'SCC' column:
[0 1]
Original values:
['no' 'yes']

Encoded values in 'SMOKE' column:
[0 1]
Original values:
['no' 'yes']

Encoded values in 'family_history_with_overweight' column:
[1 0]
Original values:
['yes' 'no']

Encoded values in 'CAEC' column:
[2 1 0 3]
Original values:
['Sometimes' 'Frequently' 'Always' 'no']

Encoded values in 'MTRANS' column:
[3 4 0 2 1]
Original values:
['Public_Transportation' 'Walking' 'Automobile' 'Motorbike' 'Bike']

Encoded values in 'NObeyesdad' column:
```

```
[1 3 2 0]
Original values:
['Normal_Weight' 'Overweight' 'Obesity' 'Insufficient_Weight']
```

In [24]:
```
# import libraries for models
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report
```

In [25]:
```
# Split data
X = obesity.drop(columns=['NObeyesdad'])
y = obesity['NObeyesdad']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

In [26]:
```
classifiers = {
    'KNN': KNeighborsClassifier(),
    'Naive Bayes': GaussianNB(),
    'Logistic Regression': LogisticRegression(max_iter=1000),
    'Random Forest': RandomForestClassifier(random_state=42)
}

for name, clf in classifiers.items():
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)
    print(f"Classifier: {name}")
    print(classification_report(y_test, y_pred))
    print("="*60)
```

```
Classifier: KNN
              precision    recall  f1-score   support

           0       0.81      0.96      0.88        56
           1       0.81      0.40      0.54        62
```

```
           2       0.97      1.00      0.98       199
           3       0.82      0.92      0.86       106

    accuracy                           0.89       423
   macro avg       0.85      0.82      0.82       423
weighted avg       0.88      0.89      0.87       423

============================================================
Classifier: Naive Bayes
              precision    recall  f1-score   support

           0       0.67      0.86      0.75        56
           1       0.54      0.35      0.43        62
           2       0.74      0.93      0.83       199
           3       0.56      0.31      0.40       106

    accuracy                           0.68       423
   macro avg       0.63      0.61      0.60       423
weighted avg       0.66      0.68      0.65       423

============================================================
Classifier: Logistic Regression
              precision    recall  f1-score   support

           0       0.85      0.98      0.91        56
           1       0.90      0.58      0.71        62
           2       0.97      0.97      0.97       199
           3       0.82      0.91      0.86       106

    accuracy                           0.90       423
   macro avg       0.88      0.86      0.86       423
weighted avg       0.90      0.90      0.90       423

============================================================
Classifier: Random Forest
              precision    recall  f1-score   support

           0       0.98      0.96      0.97        56
           1       0.89      0.87      0.88        62
```

```
           2       0.99      0.99      0.99       199
           3       0.93      0.94      0.93       106

    accuracy                           0.96       423
   macro avg       0.95      0.94      0.94       423
weighted avg       0.96      0.96      0.96       423

============================================================
```

```
/Users/gracieinman/anaconda3/lib/python3.10/site-packages/sklearn/linear_model/_logistic.py:458: Converg
enceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

In [27]:
```
# Random Forest model
rf = RandomForestClassifier(random_state=42)
rf.fit(X_train, y_train)

# feature importances
feature_importances_rf = rf.feature_importances_
importance_df_rf = pd.DataFrame({'Feature': X_train.columns, 'Importance': feature_importances_rf})
importance_df_rf = importance_df_rf.sort_values(by='Importance', ascending=False)

print(importance_df_rf)
```

```
                         Feature  Importance
3                         Weight    0.453374
2                         Height    0.093660
0                            Age    0.074502
7                            NCP    0.050600
14                          CAEC    0.048392
11  family_history_with_overweight    0.048151
6                           FCVC    0.041174
12                           FAF    0.038303
13                           TUE    0.037573
10                          CH2O    0.036945
1                         Gender    0.020621
15                        MTRANS    0.018938
4                           CALC    0.018063
5                           FAVC    0.011748
8                            SCC    0.005126
9                          SMOKE    0.002830
```

In [28]:

```
# Drop column
X = obesity.drop(columns=['Weight', 'NObeyesdad'])
y = obesity['NObeyesdad']

# Split the dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize and train
classifiers = {
    'KNN': KNeighborsClassifier(),
    'Naive Bayes': GaussianNB(),
    'Logistic Regression': LogisticRegression(max_iter=1000),
    'Random Forest': RandomForestClassifier(random_state=42)
}

for name, clf in classifiers.items():
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)
    print(f"Classifier: {name}")
    print(classification_report(y_test, y_pred))
    print("="*60)
```

```
Classifier: KNN
              precision    recall  f1-score   support

           0       0.68      0.91      0.78        56
           1       0.67      0.19      0.30        62
           2       0.79      0.96      0.87       199
           3       0.81      0.67      0.73       106

    accuracy                           0.77       423
   macro avg       0.74      0.68      0.67       423
weighted avg       0.76      0.77      0.74       423

============================================================
Classifier: Naive Bayes
              precision    recall  f1-score   support

           0       0.41      0.66      0.50        56
           1       0.41      0.18      0.25        62
           2       0.71      0.94      0.81       199
           3       0.59      0.23      0.33       106

    accuracy                           0.61       423
   macro avg       0.53      0.50      0.47       423
weighted avg       0.59      0.61      0.56       423

============================================================
Classifier: Logistic Regression
              precision    recall  f1-score   support

           0       0.55      0.55      0.55        56
           1       0.52      0.26      0.34        62
           2       0.72      0.90      0.80       199
           3       0.52      0.43      0.47       106

    accuracy                           0.64       423
   macro avg       0.58      0.54      0.54       423
weighted avg       0.62      0.64      0.62       423

============================================================
```

```
Classifier: Random Forest
              precision    recall  f1-score   support

           0       0.96      0.95      0.95        56
           1       0.72      0.77      0.74        62
           2       0.95      0.95      0.95       199
           3       0.87      0.85      0.86       106

    accuracy                           0.90       423
   macro avg       0.88      0.88      0.88       423
weighted avg       0.90      0.90      0.90       423

============================================================
```

```
/Users/gracieinman/anaconda3/lib/python3.10/site-packages/sklearn/linear_model/_logistic.py:458: Converg
enceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

In [29]:

```
# Random Forest model
rf = RandomForestClassifier(random_state=42)
rf.fit(X_train, y_train)

# feature importances
feature_importances_rf = rf.feature_importances_
importance_df_rf = pd.DataFrame({'Feature': X_train.columns, 'Importance': feature_importances_rf})
importance_df_rf = importance_df_rf.sort_values(by='Importance', ascending=False)

print(importance_df_rf)
```

```
                           Feature  Importance
0                              Age    0.149467
2                           Height    0.118054
6                              NCP    0.097367
5                             FCVC    0.092405
11                             FAF    0.084912
9                             CH2O    0.083776
12                             TUE    0.078851
13                            CAEC    0.070173
10  family_history_with_overweight    0.069628
3                             CALC    0.044621
14                          MTRANS    0.036946
1                           Gender    0.033221
4                             FAVC    0.024403
7                              SCC    0.011323
8                            SMOKE    0.004853
```

In [31]:

```python
# Drop columns
X = obesity.drop(columns=['Weight', 'Height', 'NObeyesdad'])
y = obesity['NObeyesdad']

# Split the dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize and train
classifiers = {
    'KNN': KNeighborsClassifier(),
    'Naive Bayes': GaussianNB(),
    'Logistic Regression': LogisticRegression(max_iter=1000),
    'Random Forest': RandomForestClassifier(random_state=42)
}

for name, clf in classifiers.items():
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)
    print(f"Classifier: {name}")
    print(classification_report(y_test, y_pred))
    print("="*60)
```

```
Classifier: KNN
              precision    recall  f1-score   support

           0       0.68      0.91      0.78        56
           1       0.63      0.19      0.30        62
           2       0.80      0.96      0.87       199
           3       0.81      0.67      0.73       106

    accuracy                           0.77       423
   macro avg       0.73      0.68      0.67       423
weighted avg       0.76      0.77      0.74       423

============================================================
Classifier: Naive Bayes
              precision    recall  f1-score   support

           0       0.39      0.66      0.49        56
           1       0.42      0.18      0.25        62
           2       0.71      0.94      0.81       199
           3       0.58      0.21      0.31       106

    accuracy                           0.61       423
   macro avg       0.52      0.50      0.46       423
weighted avg       0.59      0.61      0.56       423

============================================================
Classifier: Logistic Regression
              precision    recall  f1-score   support

           0       0.52      0.54      0.53        56
           1       0.50      0.24      0.33        62
           2       0.72      0.90      0.80       199
           3       0.52      0.42      0.47       106

    accuracy                           0.64       423
   macro avg       0.56      0.53      0.53       423
weighted avg       0.61      0.64      0.61       423

============================================================
```

```
Classifier: Random Forest
              precision    recall  f1-score   support

           0       0.95      0.95      0.95        56
           1       0.71      0.73      0.72        62
           2       0.91      0.94      0.93       199
           3       0.86      0.78      0.82       106

    accuracy                           0.87       423
   macro avg       0.86      0.85      0.85       423
weighted avg       0.87      0.87      0.87       423

============================================================
```

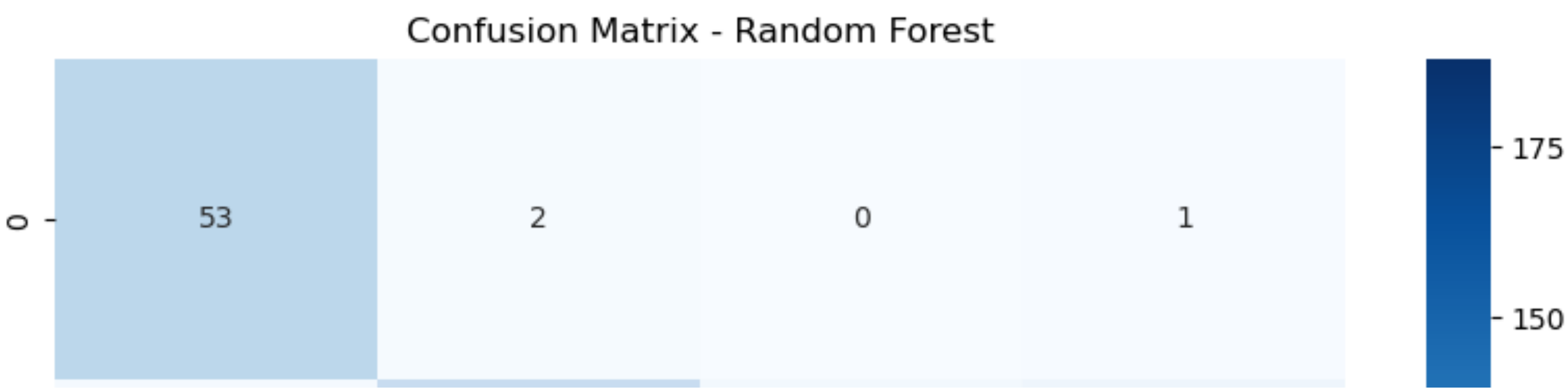
```
/Users/gracieinman/anaconda3/lib/python3.10/site-packages/sklearn/linear_model/_logistic.py:458: Converg
enceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

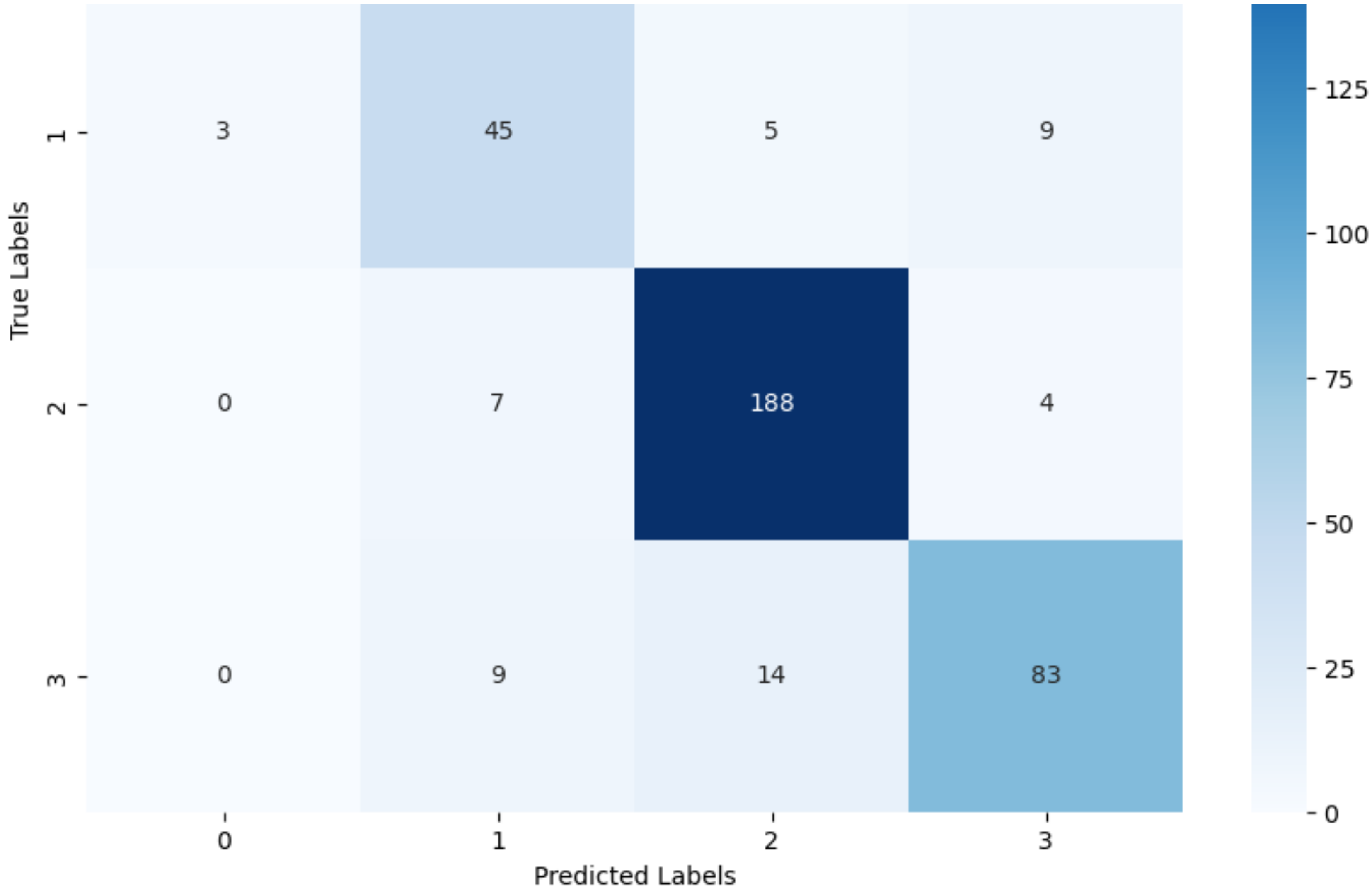
In [32]:

```
rf = RandomForestClassifier(random_state=42)
rf.fit(X_train, y_train)

# Feature importances
feature_importances_rf = rf.feature_importances_
importance_df_rf = pd.DataFrame({'Feature': X_train.columns, 'Importance': feature_importances_rf})
importance_df_rf = importance_df_rf.sort_values(by='Importance', ascending=False)

print(importance_df_rf)
```

	Feature	Importance
0	Age	0.169168
5	NCP	0.106146
4	FCVC	0.102093
10	FAF	0.099682
8	CH2O	0.096452
11	TUE	0.095572
9	family_history_with_overweight	0.076150
12	CAEC	0.074556
2	CALC	0.050542
1	Gender	0.042663
13	MTRANS	0.041167
3	FAVC	0.027501
6	SCC	0.013276
7	SMOKE	0.005030

In [34]:
```
from sklearn.metrics import confusion_matrix
y_pred_rf = rf.predict(X_test)
cm_rf = confusion_matrix(y_test, y_pred_rf)

# Display the confusion matrix
plt.figure(figsize=(10, 8))
sns.heatmap(cm_rf, annot=True, fmt='d', cmap='Blues', xticklabels=rf.classes_, yticklabels=rf.classes_)
plt.title('Confusion Matrix - Random Forest')
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.show()
```





3
45
5
9
0
7
188
4
0
9
14
83
1
2
3
True Labels
0
1
2
3
Predicted Labels
125
100
75
50
25
0

In [ ]: