ML - Obesity Prediction

May 23, 2024

1 Importing Packages

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
```

2 Loading Data & Basic Analysis

[2]:
```
data= pd.read_csv('ObesityDataSet_raw_and_data_sinthetic.csv')
df = data.copy()
df.head()
```

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

  family_history_with_overweight  FAF  TUE       CAEC                 MTRANS  \
0                            yes  0.0  1.0  Sometimes  Public_Transportation
1                            yes  3.0  0.0  Sometimes  Public_Transportation
2                            yes  2.0  1.0  Sometimes  Public_Transportation
3                             no  2.0  0.0  Sometimes                Walking
4                             no  0.0  0.0  Sometimes  Public_Transportation

            NObeyesdad
0        Normal_Weight
1        Normal_Weight
2        Normal_Weight
3   Overweight_Level_I
4  Overweight_Level_II
```

[3]:
```
df.tail()
```

[3]:
```
            Age  Gender    Height      Weight       CALC FAVC  FCVC  NCP  SCC  \
2106  20.976842  Female  1.710730  131.408528  Sometimes  yes   3.0  3.0   no
2107  21.982942  Female  1.748584  133.742943  Sometimes  yes   3.0  3.0   no
2108  22.524036  Female  1.752206  133.689352  Sometimes  yes   3.0  3.0   no
2109  24.361936  Female  1.739450  133.346641  Sometimes  yes   3.0  3.0   no
2110  23.664709  Female  1.738836  133.472641  Sometimes  yes   3.0  3.0   no

     SMOKE      CH2O family_history_with_overweight       FAF       TUE  \
2106    no  1.728139                            yes  1.676269  0.906247
2107    no  2.005130                            yes  1.341390  0.599270
2108    no  2.054193                            yes  1.414209  0.646288
2109    no  2.852339                            yes  1.139107  0.586035
2110    no  2.863513                            yes  1.026452  0.714137

           CAEC                 MTRANS        NObeyesdad
2106  Sometimes  Public_Transportation  Obesity_Type_III
2107  Sometimes  Public_Transportation  Obesity_Type_III
2108  Sometimes  Public_Transportation  Obesity_Type_III
2109  Sometimes  Public_Transportation  Obesity_Type_III
2110  Sometimes  Public_Transportation  Obesity_Type_III
```

[4]:
```python
df.shape
```

[4]: (2111, 17)

[5]:
```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2111 entries, 0 to 2110
Data columns (total 17 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   Age                             2111 non-null   float64
 1   Gender                          2111 non-null   object
 2   Height                          2111 non-null   float64
 3   Weight                          2111 non-null   float64
 4   CALC                            2111 non-null   object
 5   FAVC                            2111 non-null   object
 6   FCVC                            2111 non-null   float64
 7   NCP                             2111 non-null   float64
 8   SCC                             2111 non-null   object
 9   SMOKE                           2111 non-null   object
 10  CH2O                            2111 non-null   float64
 11  family_history_with_overweight  2111 non-null   object
 12  FAF                             2111 non-null   float64
 13  TUE                             2111 non-null   float64
 14  CAEC                            2111 non-null   object
 15  MTRANS                          2111 non-null   object
```

```
 16  NObeyesdad                      2111 non-null   object
dtypes: float64(8), object(9)
memory usage: 280.5+ KB
```

[6]:
```python
df.describe().T
```

[6]:
```
         count       mean        std    min        25%        50%         75%  \
Age     2111.0  24.312600   6.345968  14.00  19.947192  22.777890   26.000000
Height  2111.0   1.701677   0.093305   1.45   1.630000   1.700499    1.768464
Weight  2111.0  86.586058  26.191172  39.00  65.473343  83.000000  107.430682
FCVC    2111.0   2.419043   0.533927   1.00   2.000000   2.385502    3.000000
NCP     2111.0   2.685628   0.778039   1.00   2.658738   3.000000    3.000000
CH2O    2111.0   2.008011   0.612953   1.00   1.584812   2.000000    2.477420
FAF     2111.0   1.010298   0.850592   0.00   0.124505   1.000000    1.666678
TUE     2111.0   0.657866   0.608927   0.00   0.000000   0.625350    1.000000

           max
Age      61.00
Height    1.98
Weight  173.00
FCVC      3.00
NCP       4.00
CH2O      3.00
FAF       3.00
TUE       2.00
```

[7]:
```python
df.describe(include=["O"]).T
```

[7]:
```
                               count unique                    top  freq
Gender                          2111      2                   Male  1068
CALC                            2111      4              Sometimes  1401
FAVC                            2111      2                    yes  1866
SCC                             2111      2                     no  2015
SMOKE                           2111      2                     no  2067
family_history_with_overweight  2111      2                    yes  1726
CAEC                            2111      4              Sometimes  1765
MTRANS                          2111      5  Public_Transportation  1580
NObeyesdad                      2111      7         Obesity_Type_I   351
```

[8]:
```python
df.isnull().sum()
```

[8]:
```
Age                               0
Gender                            0
Height                            0
Weight                            0
CALC                              0
FAVC                              0
FCVC                              0
```

```
NCP                               0
SCC                               0
SMOKE                             0
CH2O                              0
family_history_with_overweight    0
FAF                               0
TUE                               0
CAEC                              0
MTRANS                            0
NObeyesdad                        0
dtype: int64
```

[9]:
```python
df.duplicated().sum()
```

[9]:
```
24
```

- Keeping duplicates values as we don't have a primary key and the rows could be of different patients

3 Exploratory Data Analysis

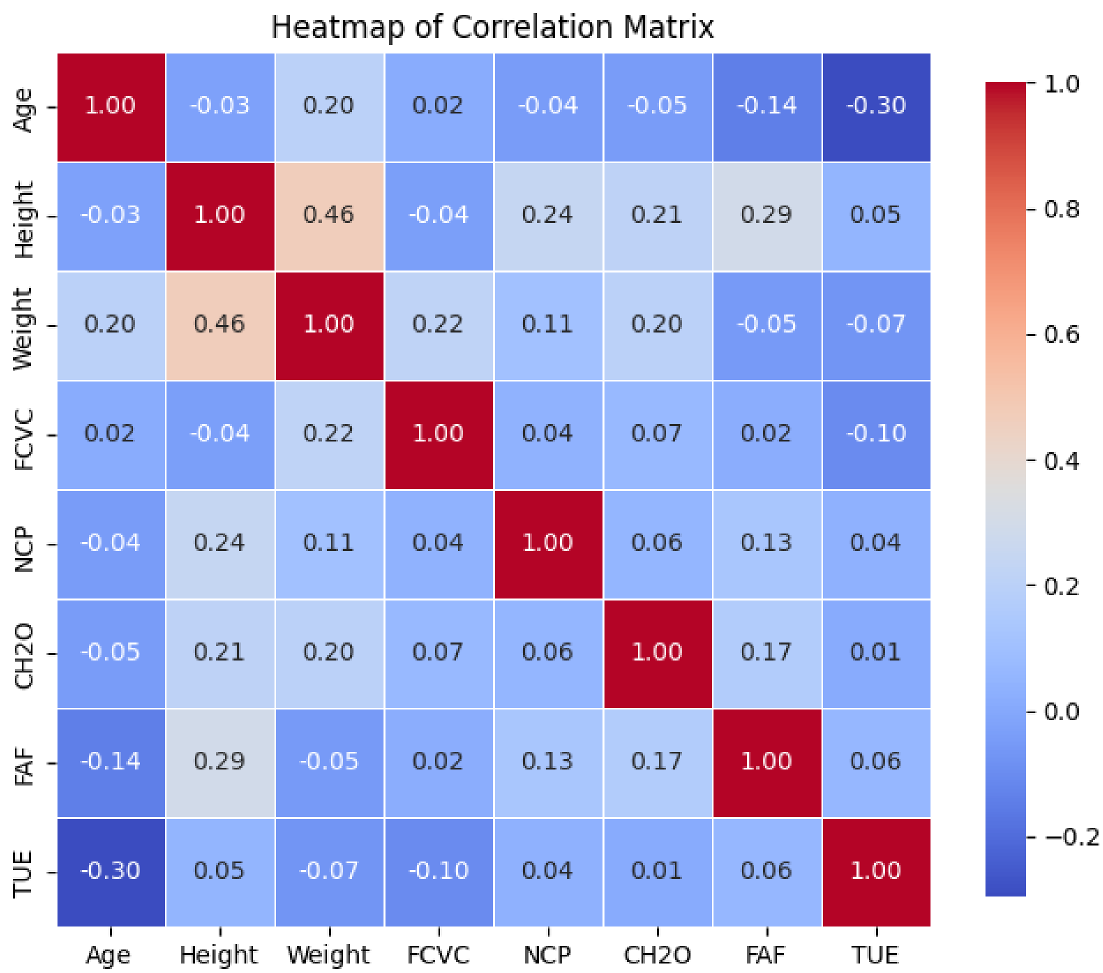
[10]:
```python
df.columns
```

[10]:
```
Index(['Age', 'Gender', 'Height', 'Weight', 'CALC', 'FAVC', 'FCVC', 'NCP',
       'SCC', 'SMOKE', 'CH2O', 'family_history_with_overweight', 'FAF', 'TUE',
       'CAEC', 'MTRANS', 'NObeyesdad'],
      dtype='object')
```

3.1 Correlation matrix

[11]:
```python
# Select only numeric columns for correlation analysis
numerical_data = df.select_dtypes(include=['float64'])

# Calculate the correlation matrix
correlation_matrix = numerical_data.corr()
```

[12]:
```python
plt.figure(figsize=(8, 12))
sns.heatmap(correlation_matrix, annot=True, fmt=".2f", cmap='coolwarm',␣
 ↪square=True, linewidths=.5, cbar_kws={"shrink": .5})

plt.title('Heatmap of Correlation Matrix')
plt.show()
```



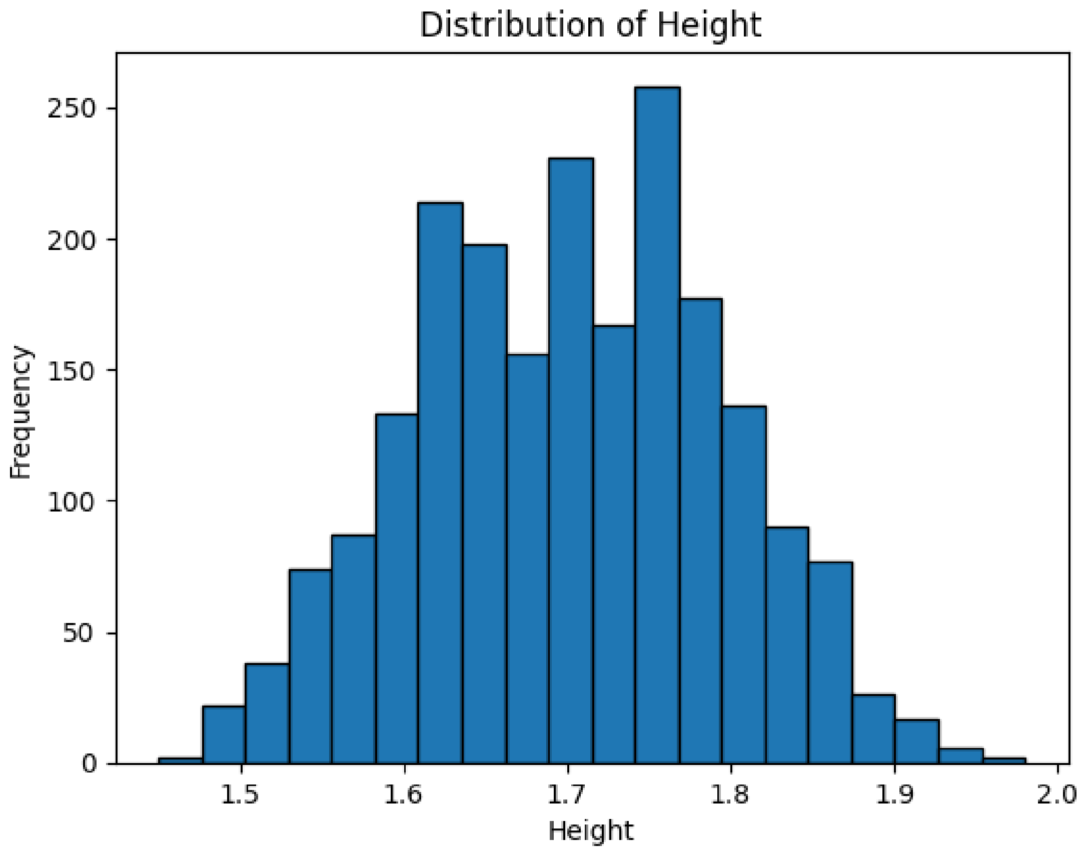
[ ]:

```

```

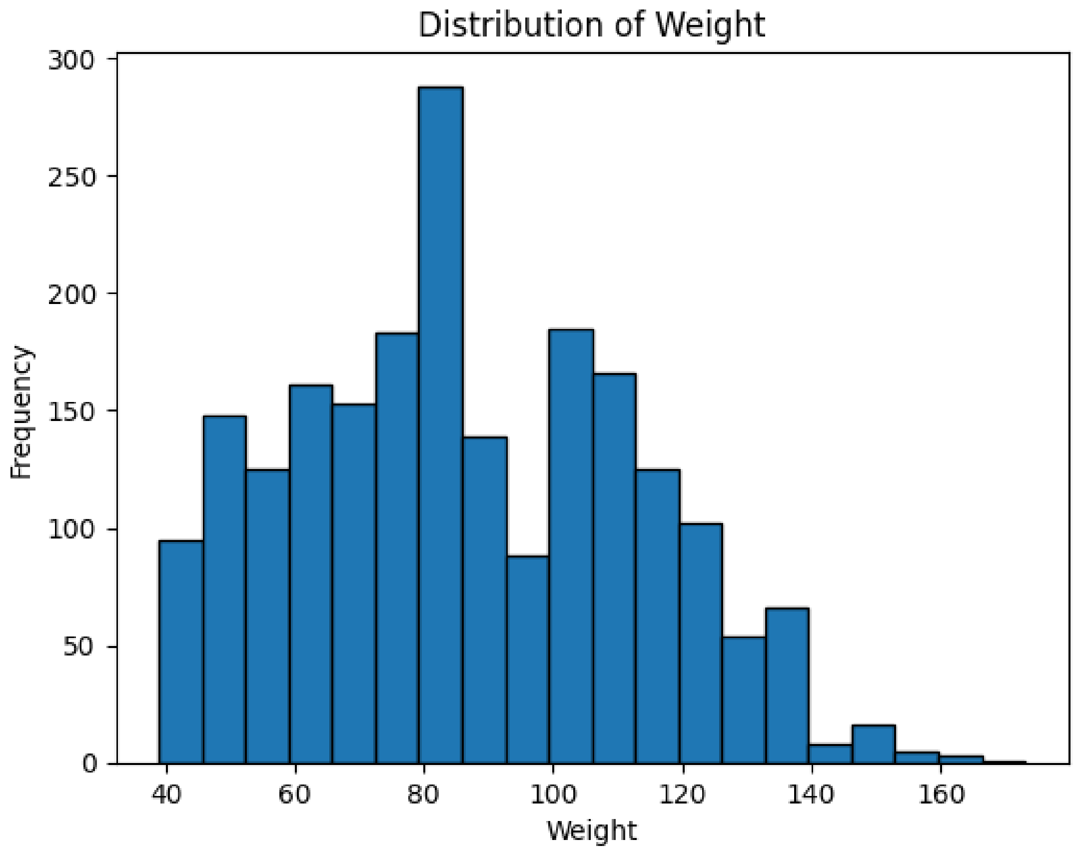
3.2 Height and Weight Distribution

[13]:

```
plt.hist(df['Height'], bins=20, edgecolor='black')
plt.title('Distribution of Height')
plt.xlabel('Height')
plt.ylabel('Frequency')

plt.show()
```

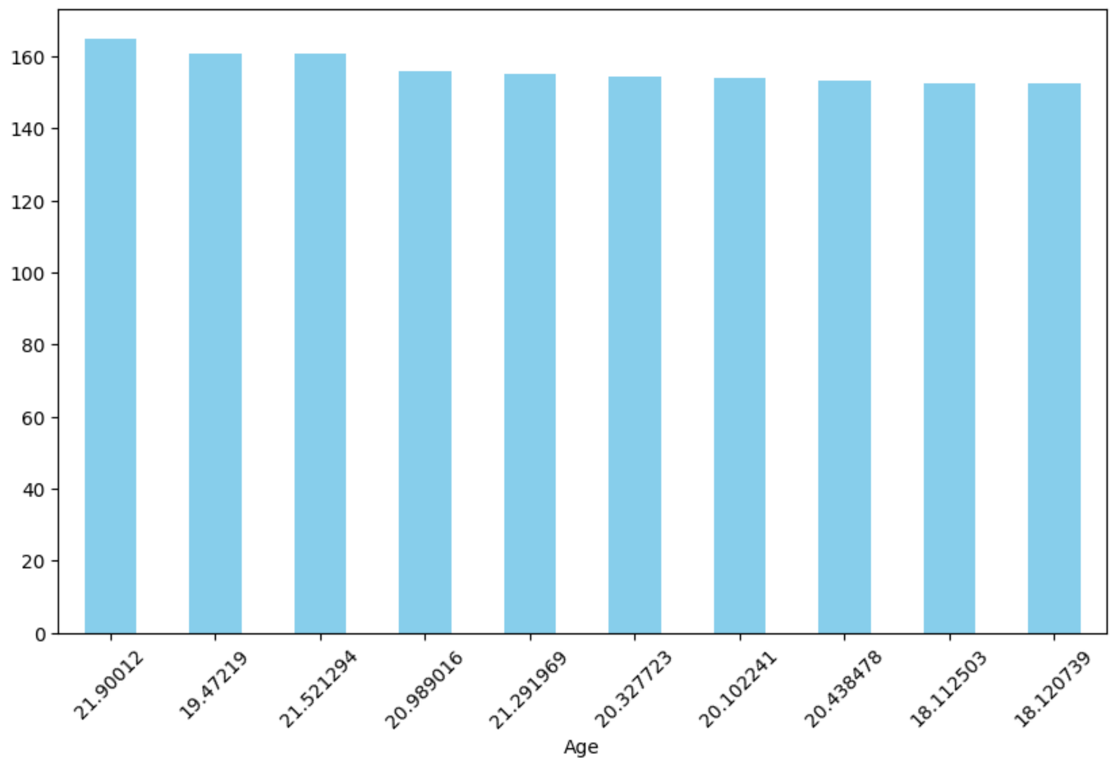


[14]:
```
plt.hist(df['Weight'], bins=20, edgecolor='black')
plt.title('Distribution of Weight')
plt.xlabel('Weight')
plt.ylabel('Frequency')

plt.show()
```



3.3 Top 10 Ages with Highest Weight

[15]:
```
Top10_age = df.groupby('Age')['Weight'].mean().sort_values(ascending=False).
  ↪head(10)

plt.figure(figsize=(10, 6))
Top10_age.plot(kind='bar', color='skyblue')

plt.xticks(rotation=45)
plt.show()
Top10_age
```



[15]:

```
Age
21.900120    165.057269
19.472190    160.935351
21.521294    160.639405
20.989016    155.872093
21.291969    155.242672
20.327723    154.618446
20.102241    153.959945
20.438478    153.149491
18.112503    152.720545
18.120739    152.567671
Name: Weight, dtype: float64
```

[ ]:

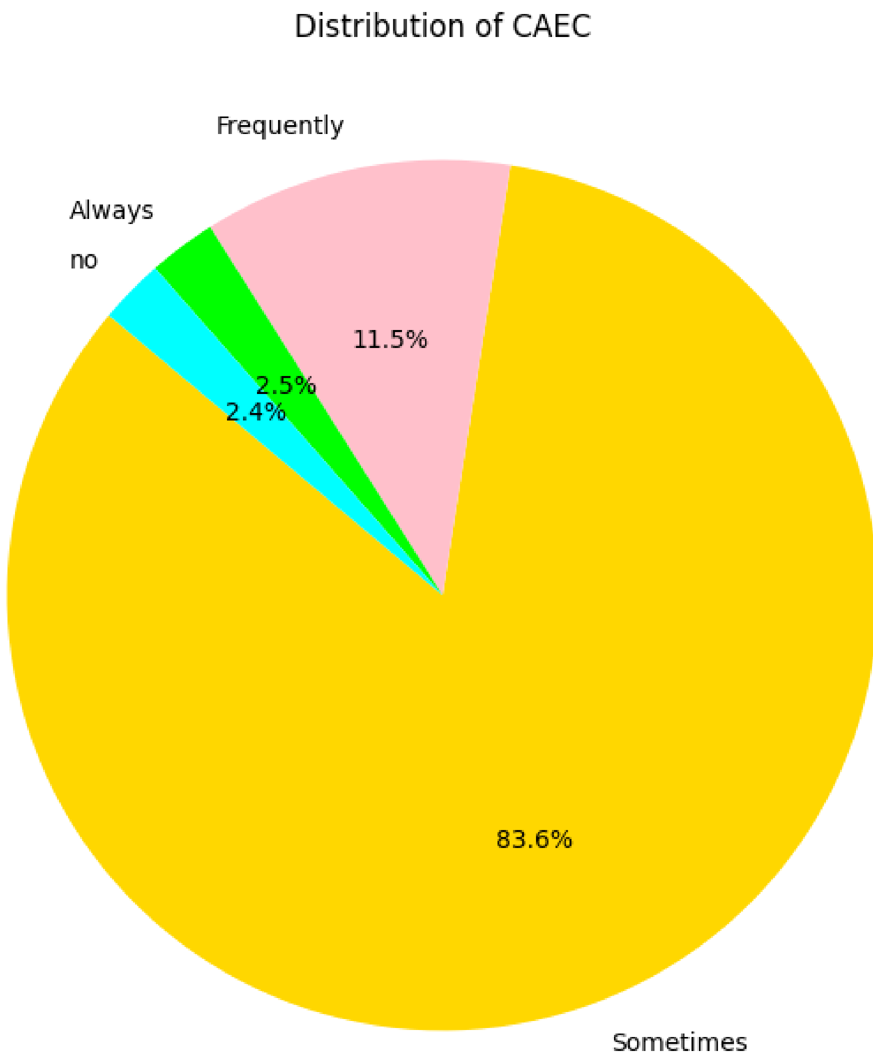
```

```

3.4 Distribution of CAEC values

[16]:

```
caec_count = df['CAEC'].value_counts()

plt.figure(figsize=(8, 8))
caec_count.plot(kind='pie', autopct='%1.1f%%', startangle=140, colors=['gold',␣
 ↪'pink', 'lime', 'cyan'])
```

```
plt.title('Distribution of CAEC')
plt.ylabel('')
plt.show()

caec_count
```



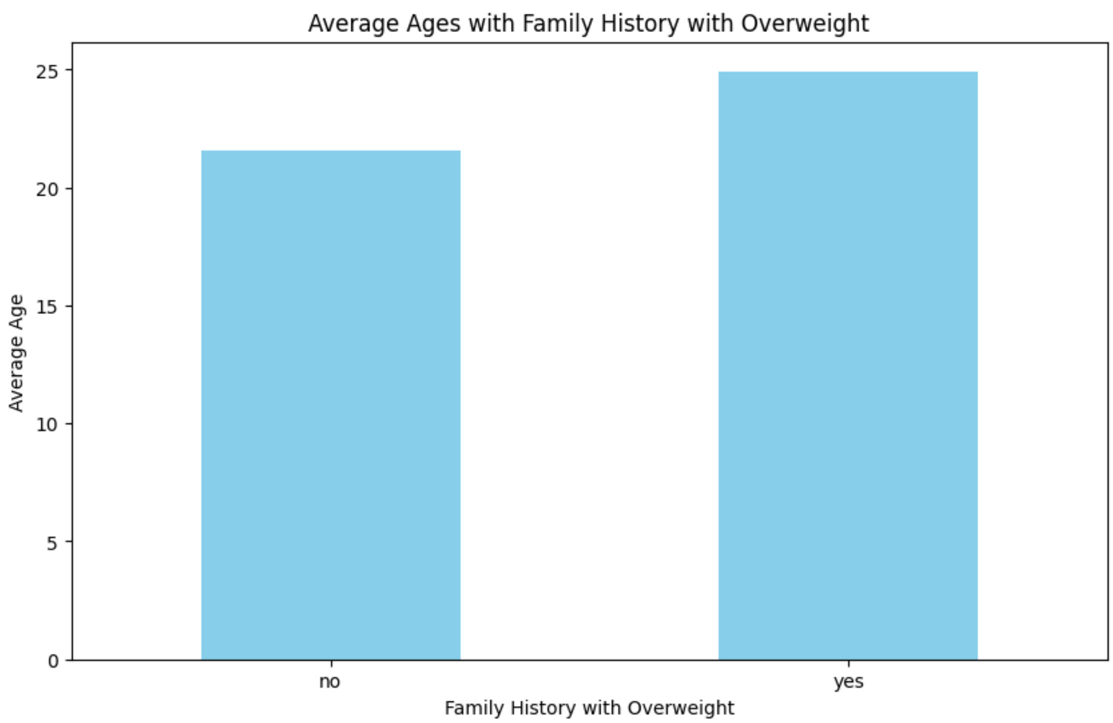

```
[16]: CAEC
      Sometimes     1765
      Frequently     242
      Always          53
      no              51
```

```
Name: count, dtype: int64
```

[ ]:

3.5 Average Ages with Family History with Overweight

[17]:
```
avg_age_with_familyHistory = df.
 ↪groupby('family_history_with_overweight')['Age'].mean()

plt.figure(figsize=(10, 6))
avg_age_with_familyHistory.plot(kind='bar', color='skyblue')
plt.title('Average Ages with Family History with Overweight')
plt.xlabel('Family History with Overweight')
plt.ylabel('Average Age')
plt.xticks(rotation=0)

plt.show()
avg_age_with_familyHistory
```



[17]:
```
family_history_with_overweight
no     21.549015
yes    24.929043
Name: Age, dtype: float64
```

[ ]:

```

```

4 Data Preprocessing

[18]:

```python
from sklearn.preprocessing import LabelEncoder, StandardScaler, OneHotEncoder
from sklearn.model_selection import train_test_split
```

[19]:

```python
categorical_cols = df.select_dtypes(include=['object']).columns.tolist()
categorical_cols
```

[19]:

```
['Gender',
 'CALC',
 'FAVC',
 'SCC',
 'SMOKE',
 'family_history_with_overweight',
 'CAEC',
 'MTRANS',
 'NObeyesdad']
```

[20]:

```python
continuous_cols = df.select_dtypes(include=['float64']).columns.tolist()
continuous_cols
```

[20]:

```
['Age', 'Height', 'Weight', 'FCVC', 'NCP', 'CH2O', 'FAF', 'TUE']
```

[21]:

```python
# Applying Label Encoding to categorical columns
label_encoder = LabelEncoder()

for col in categorical_cols:
    df[col] = label_encoder.fit_transform(df[col])
```

[22]:

```python
# Applying Standard Scaling to continuous columns
scaler = StandardScaler()
df[continuous_cols] = scaler.fit_transform(df[continuous_cols])
```

[23]:

```python
# Applying One-Hot Encoding to nominal categorical variable with more than two␣
↪categories.
onehot_encoder = OneHotEncoder(categories='auto',sparse_output=False)

nominal_cols = ['CALC', 'CAEC', 'MTRANS']
df1 = pd.get_dummies(df, columns=nominal_cols)
```

[24]:

```python
df1
```

[24]:

```
          Age  Gender    Height    Weight  FAVC      FCVC       NCP  SCC  \
0   -0.522124       0 -0.875589 -0.862558     0 -0.785019  0.404153    0
1   -0.522124       0 -1.947599 -1.168077     0  1.088342  0.404153    1
```

```
2    -0.206889       1  1.054029 -0.366090     0 -0.785019  0.404153    0
3     0.423582       1  1.054029  0.015808     0  1.088342  0.404153    0
4    -0.364507       1  0.839627  0.122740     0 -0.785019 -2.167023    0
…          …    …          …        …       …        …          …   …
2106 -0.525774       0  0.097045  1.711763     1  1.088342  0.404153    0
2107 -0.367195       0  0.502844  1.800914     1  1.088342  0.404153    0
2108 -0.281909       0  0.541672  1.798868     1  1.088342  0.404153    0
2109  0.007776       0  0.404927  1.785780     1  1.088342  0.404153    0
2110 -0.102119       0  0.398344  1.790592     1  1.088342  0.404153    0

      SMOKE      CH2O  …  CALC_3  CAEC_0  CAEC_1  CAEC_2  CAEC_3  MTRANS_0  \
0         0 -0.013073  …    True   False   False    True   False     False
1         1  1.618759  …   False   False   False    True   False     False
2         0 -0.013073  …   False   False   False    True   False     False
3         0 -0.013073  …   False   False   False    True   False     False
4         0 -0.013073  …   False   False   False    True   False     False
…       …          …  …  …      …       …     …      …       …
2106      0 -0.456705  …   False   False   False    True   False     False
2107      0 -0.004702  …   False   False   False    True   False     False
2108      0  0.075361  …   False   False   False    True   False     False
2109      0  1.377801  …   False   False   False    True   False     False
2110      0  1.396035  …   False   False   False    True   False     False

      MTRANS_1  MTRANS_2  MTRANS_3  MTRANS_4
0        False     False      True     False
1        False     False      True     False
2        False     False      True     False
3        False     False     False      True
4        False     False      True     False
…          …        …        …        …
2106     False     False      True     False
2107     False     False      True     False
2108     False     False      True     False
2109     False     False      True     False
2110     False     False      True     False

[2111 rows x 27 columns]
```

[25]:
```
# Split the dataset into features and target variable
X = df1.drop('NObeyesdad', axis=1)
y = df1['NObeyesdad']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)

print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
```

```
(1688, 26) (423, 26) (1688,) (423,)
```

```
[ ]:
```

5 Algorithm Search

```
[26]: from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, accuracy_score,␣
 ↪confusion_matrix
from sklearn.preprocessing import LabelEncoder
```

```
[27]: classifiers = {
    "Logistic Regression": LogisticRegression(max_iter=200),
    "Decision Tree": DecisionTreeClassifier(),
    "Random Forest": RandomForestClassifier(),
    "Support Vector Machine": SVC(),
    "K-Nearest Neighbors": KNeighborsClassifier()
}

results = {}
for name, clf in classifiers.items():
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    target_names = label_encoder.classes_.astype(str)
    report = classification_report(y_test, y_pred, target_names=target_names)
    results[name] = {
        "accuracy": accuracy,
        "classification_report": report
    }
    print(f"Classifier: {name}\nAccuracy: {accuracy}\n")
    print(f"Classification Report:\n{report}\n")
```

```
Classifier: Logistic Regression
Accuracy: 0.8699763593380615

Classification Report:
                     precision    recall  f1-score   support

Insufficient_Weight       0.84      1.00      0.91        56
      Normal_Weight       0.91      0.63      0.74        62
     Obesity_Type_I       0.93      0.90      0.92        78
    Obesity_Type_II       0.90      0.97      0.93        58
```

```
   Obesity_Type_III       1.00      1.00      1.00        63
 Overweight_Level_I       0.75      0.75      0.75        56
Overweight_Level_II       0.74      0.84      0.79        50

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


Classifier: Decision Tree
Accuracy: 0.9456264775413712

Classification Report:
                     precision    recall  f1-score   support

Insufficient_Weight       0.93      0.98      0.96        56
      Normal_Weight       0.90      0.90      0.90        62
     Obesity_Type_I       0.95      0.94      0.94        78
    Obesity_Type_II       0.95      0.95      0.95        58
   Obesity_Type_III       1.00      1.00      1.00        63
 Overweight_Level_I       0.93      0.91      0.92        56
Overweight_Level_II       0.96      0.94      0.95        50

           accuracy                           0.95       423
          macro avg       0.95      0.95      0.95       423
       weighted avg       0.95      0.95      0.95       423


Classifier: Random Forest
Accuracy: 0.950354609929078

Classification Report:
                     precision    recall  f1-score   support

Insufficient_Weight       0.98      0.96      0.97        56
      Normal_Weight       0.87      0.94      0.90        62
     Obesity_Type_I       0.99      0.95      0.97        78
    Obesity_Type_II       0.98      0.98      0.98        58
   Obesity_Type_III       1.00      1.00      1.00        63
 Overweight_Level_I       0.88      0.88      0.88        56
Overweight_Level_II       0.96      0.94      0.95        50

           accuracy                           0.95       423
          macro avg       0.95      0.95      0.95       423
       weighted avg       0.95      0.95      0.95       423


Classifier: Support Vector Machine
```

```
Accuracy: 0.933806146572104

Classification Report:
                      precision    recall  f1-score   support

Insufficient_Weight       0.96      0.96      0.96        56
      Normal_Weight       0.84      0.87      0.86        62
     Obesity_Type_I       0.96      0.97      0.97        78
    Obesity_Type_II       0.97      0.98      0.97        58
   Obesity_Type_III       1.00      1.00      1.00        63
 Overweight_Level_I       0.84      0.84      0.84        56
Overweight_Level_II       0.96      0.88      0.92        50

           accuracy                           0.93       423
          macro avg       0.93      0.93      0.93       423
       weighted avg       0.93      0.93      0.93       423


Classifier: K-Nearest Neighbors
Accuracy: 0.8321513002364066

Classification Report:
                      precision    recall  f1-score   support

Insufficient_Weight       0.78      0.95      0.85        56
      Normal_Weight       0.85      0.37      0.52        62
     Obesity_Type_I       0.84      0.94      0.88        78
    Obesity_Type_II       0.90      0.97      0.93        58
   Obesity_Type_III       0.98      1.00      0.99        63
 Overweight_Level_I       0.74      0.82      0.78        56
Overweight_Level_II       0.72      0.76      0.74        50

           accuracy                           0.83       423
          macro avg       0.83      0.83      0.81       423
       weighted avg       0.84      0.83      0.82       423
```

[ ]:

```

```

6 Best Algorithm

- Random Forest has the Best Accuracy: 95%

[29]:

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

```
y_pred_= model.predict(X_test)

myAccuracy= accuracy_score(y_test, y_pred_)

confusionReport= classification_report(y_test, y_pred_)

confusionMatrix= confusion_matrix(y_test, y_pred_)

print('Random Forest')
print('Classification Report')
print(confusionReport)
print(f'Accuracy: {myAccuracy}')
print('Confusion Matrix')
print(confusionMatrix)
```

```
Random Forest
Classification Report
              precision    recall  f1-score   support

           0       1.00      0.96      0.98        56
           1       0.88      0.95      0.91        62
           2       0.99      0.94      0.96        78
           3       0.97      0.98      0.97        58
           4       1.00      1.00      1.00        63
           5       0.94      0.89      0.92        56
           6       0.91      0.96      0.93        50

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

Accuracy: 0.9550827423167849
Confusion Matrix
[[54  2  0  0  0  0  0]
 [ 0 59  0  0  0  1  2]
 [ 0  1 73  2  0  0  2]
 [ 0  0  1 57  0  0  0]
 [ 0  0  0  0 63  0  0]
 [ 0  5  0  0  0 50  1]
 [ 0  0  0  0  0  2 48]]
```

```
[ ]:
```