Module 1 Activity

Assigned at the start of Module 1

Due at the end of Module 1

Weekly Discussion Forum Participation

Each week, you are required to participate in the module's discussion forum. The discussion forum consists of the week's Module Activity, which is released at the beginning of the module. You must complete/attempt the activity before you can post about the activity and anything that relates to the topic.

Grading of the Discussion

1. Initial Post:

Create your thread by **Day 5 (Saturday night at midnight, PST).**

2. Responses:

Respond to at least two other posts by **Day 7 (Monday night at midnight, PST).**

Grading Criteria:

Your participation will be graded as follows:

Full Credit (100 points):

- Submit your initial post by **Day 5.**
- Respond to at least two other posts by **Day 7.**

Half Credit (50 points):

- If your initial post is late but you respond to two other posts.
- If your initial post is on time but you fail to respond to at least two other posts.

No Credit (0 points):

- If both your initial post and responses are late.
- If you fail to submit an initial post and do not respond to any others.

Additional Notes:

- **Late Initial Posts:** Late posts will automatically receive half credit if two responses are completed on time.
- **Substance Matters:** Responses must be thoughtful and constructive. Comments like "Great post!" or "I agree!" without further explanation will not earn credit.
- **Balance Participation:** Aim to engage with threads that have fewer or no responses to ensure a balanced discussion.

Avoid:

- A number of posts within a very short time-frame, especially immediately prior to the posting deadline.
- Posts that complement another post, and then consist of a summary of that.

Data Modeling Activity

Objective

Familiarize yourself with key data modeling concepts (classification, regression, and clustering) by analyzing a dataset and discussing how different modeling approaches apply.

Instructions

1. Find a Dataset

Use the `kagglehub` library to download a dataset from Kaggle. Ensure that you have set up the library with your Kaggle API key. An example of how to do this is provided below:

```
# Install kagglehub if not already installed
# !pip install kagglehub

import kagglehub as kh

# Example: Downloading a dataset
# Replace 'dataset-owner/dataset-name' with the Kaggle dataset identifier
kh.download('dataset-owner/dataset-name', path='./datasets')

# Load the dataset
import pandas as pd
data = pd.read_csv("./datasets/your_dataset.csv")
```

```
# Display the first few rows of the dataset
data.head()

# Basic dataset summary
data.info()
data.describe()
```

---

2. Dataset Exploration

Answer the following questions about your dataset:

1. What types of features are present (numerical, categorical, etc.)?
2. Are there any missing values or data quality issues?
3. What potential problems can this dataset solve? Identify tasks for classification, regression, and clustering.

Add your answers in the markdown cell below.

---

3. Tasks

Classification

- Define a classification problem in the dataset (e.g., predicting if a customer will churn).
- Optionally, implement a simple logistic regression or decision tree model.

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report

# Replace 'target' with your target column name
X = data.drop(columns=["target"])
y = data["target"]

# Split the dataset into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Train a logistic regression model
model = LogisticRegression()
model.fit(X_train, y_train)

# Make predictions and evaluate the model
y_pred = model.predict(X_test)
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test,
y_pred))
```

Regression

- Define a regression problem in the dataset (e.g., predicting house prices).
- Optionally, implement a simple linear regression model.

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

# Replace 'target' with your target column name
X = data.drop(columns=["target"])
y = data["target"]

# Split the dataset into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Train a linear regression model
model = LinearRegression()
model.fit(X_train, y_train)

# Make predictions and evaluate the model
y_pred = model.predict(X_test)
print("Mean Squared Error:", mean_squared_error(y_test, y_pred))
print("R-squared Score:", r2_score(y_test, y_pred))
```

Clustering

- Define how clustering might reveal patterns in your dataset.
- Optionally, implement a simple k-means clustering algorithm.

```
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt

# Use only numerical features for clustering
X = data.select_dtypes(include=['float64', 'int64'])

# Train a k-means model
kmeans = KMeans(n_clusters=3, random_state=42)
kmeans.fit(X)

# Add cluster labels to the dataset
data["Cluster"] = kmeans.labels_

# Visualize the clusters (for two features, if applicable)
plt.scatter(X.iloc[:, 0], X.iloc[:, 1], c=data["Cluster"],
cmap="viridis")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.title("Clustering Visualization")
plt.show()
```

---

4. Reflection and Discussion

- How does the choice of supervised or unsupervised learning depend on the dataset and problem?
- What challenges did you anticipate when applying each modeling technique?

Add your reflections below.

Deliverables

1. A short description of your dataset and the problems you identified.
2. Code and results for any implemented models (classification, regression, clustering).
3. A brief reflection on the activity.

1. Find Dataset

Let's use Obesity or CVD risk dataset from Kaggle for this exercise.

This dataset include data for the estimation of obesity levels in individuals from the countries of Mexico, Peru and Colombia, based on their eating habits and physical condition.

The data contains `17` attributes and `2,111` records, the records are labeled with the class variable `NObesity` (Obesity Level), that allows **classification**/**clustering**/**Regression** of the data using the values as below

- Insufficient Weight
- Normal Weight
- Overweight Level I
- Overweight Level II
- Obesity Type I
- Obesity Type II
- Obesity Type III

```
# Install dependencies as needed:
!pip install kagglehub
```

```
Requirement already satisfied: kagglehub in
/usr/local/lib/python3.11/dist-packages (0.3.12)
Requirement already satisfied: packaging in
/usr/local/lib/python3.11/dist-packages (from kagglehub) (24.2)
Requirement already satisfied: pyyaml in
/usr/local/lib/python3.11/dist-packages (from kagglehub) (6.0.2)
Requirement already satisfied: requests in
/usr/local/lib/python3.11/dist-packages (from kagglehub) (2.32.3)
Requirement already satisfied: tqdm in /usr/local/lib/python3.11/dist-
packages (from kagglehub) (4.67.1)
Requirement already satisfied: charset-normalizer<4,>=2 in
```

```
/usr/local/lib/python3.11/dist-packages (from requests->kagglehub)
(3.4.2)
Requirement already satisfied: idna<4,>=2.5 in
/usr/local/lib/python3.11/dist-packages (from requests->kagglehub)
(3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in
/usr/local/lib/python3.11/dist-packages (from requests->kagglehub)
(2.4.0)
Requirement already satisfied: certifi>=2017.4.17 in
/usr/local/lib/python3.11/dist-packages (from requests->kagglehub)
(2025.4.26)
```

```python
import kagglehub as kh
import pandas as pd
import os # Import os for path manipulation

# Download Obesity or CVD risk dataset from
https://www.kaggle.com/datasets/aravindpcoder/obesity-or-cvd-risk-
classifyregressorcluster
download_path = kh.dataset_download("aravindpcoder/obesity-or-cvd-
risk-classifyregressorcluster")

# Construct the full path
full_file_path = os.path.join(download_path, "ObesityDataSet.csv")

# Load the CSV file into a pandas DataFrame
df = pd.read_csv(full_file_path)

# Now you can work with your DataFrame 'df'
df.head()
categorical_cols = df.select_dtypes(include=['object',
'category']).columns.tolist()
numerical_cols =
df.select_dtypes(include=['float64','int64']).columns.tolist()

df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2111 entries, 0 to 2110
Data columns (total 17 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   Gender                          2111 non-null   object
 1   Age                             2111 non-null   float64
 2   Height                          2111 non-null   float64
 3   Weight                          2111 non-null   float64
 4   family_history_with_overweight  2111 non-null   object
 5   FAVC                            2111 non-null   object
 6   FCVC                            2111 non-null   float64
 7   NCP                             2111 non-null   float64
```

```
 8   CAEC                            2111 non-null   object
 9   SMOKE                           2111 non-null   object
 10  CH2O                            2111 non-null   float64
 11  SCC                             2111 non-null   object
 12  FAF                             2111 non-null   float64
 13  TUE                             2111 non-null   float64
 14  CALC                            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
```

2. Dataset Exploration

Q1. What types of features are present (numerical, categorical, etc.)? This dataset contains the both numerical and categorical features.

Ans. Following are list of numerical features

- Gender
- family_history_with_overweight
- FAVC
- CAEC
- SMOKE
- SCC
- CALC
- MTRANS
- NObeyesdad

Following are list of categorical features

- Age
- Height
- Weight
- FCVC
- NCP
- CH2O
- FAF
- TUE

Q2. Are there any missing values or data quality issues?

Ans. This dataset doesn't have any missing values. All features has `2111` rows. There is no data quality issue here, but data volume is very low. It could have better if rows are in range of 100k.

Q3. What potential problems can this dataset solve? Identify tasks for classification, regression, and clustering.

Ans. We can use this dataset to solve following problems

- **Regression task**: For given test data, we can predict the weight of people based on their eating habbits and family obesity history and other features.
- **Classification model**: For given test data, we can predict the Obesity Level.
- **Clustering model**: We can cluster the given data into groups based on age group and their eating habbits along with obesity levels.

Following attributes related with eating habits are are useful to explore this dataset.

Abbreviation	Full Form
FAVC	Frequent consumption of high caloric food
FCVC	Frequency of consumption of vegetables
NCP	Number of main meals
CAEC	Consumption of food between meals
CH20	Consumption of water daily
CALC	Consumption of alcohol
SCC	Calories consumption monitoring
FAF	Physical activity frequency
TUE	Time using technology devices
MTRANS	Transportation used

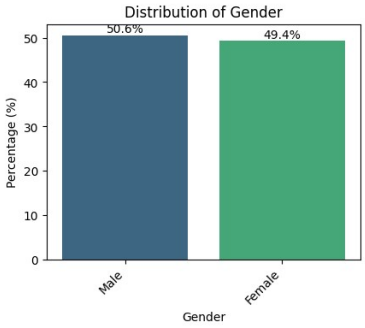
Let's plot graph to explore this dataset.

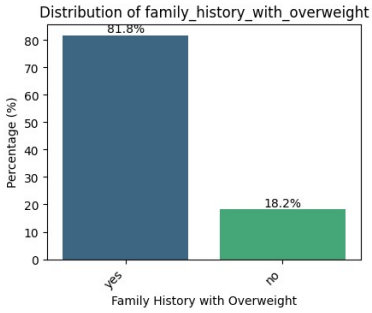
```
import math
import seaborn as sns
import matplotlib.pyplot as plt

column_label_mapping = {
    'FAVC': 'Frequent consumption of high caloric food',
    'FCVC': 'Frequency of consumption of vegetables',
    'NCP': 'Number of main meals',
    'CAEC': 'Consumption of food between meals',
    'CH2O': 'Consumption of water daily',
    'CALC': 'Consumption of alcohol',
    'SCC': 'Calories consumption monitoring',
    'FAF': 'Physical activity frequency',
    'TUE': 'Time using technology devices',
    'MTRANS': 'Transportation used',
    'NObeyesdad': 'Obesity Level',
    'Gender': 'Gender',
    'family_history_with_overweight': 'Family History with
Overweight',
```

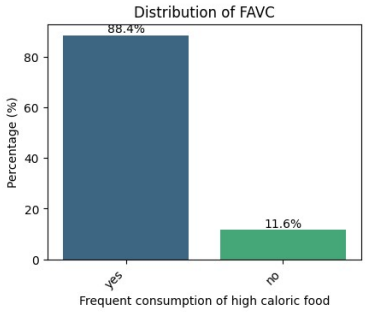
```
    'SMOKE': 'Smoking Habit',
    'Age': 'Age',
    'Height': 'Height',
    'Weight': 'Weight',
}
```

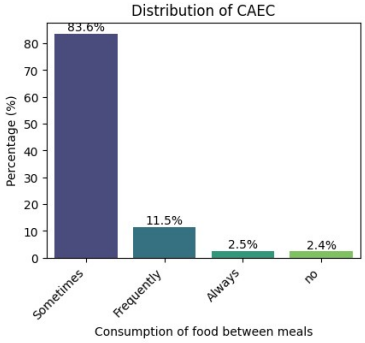
Distribution of categorical data

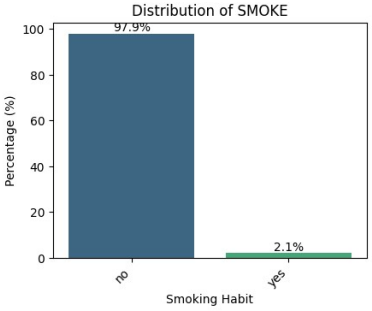
```
# Distribution of categorical data

categorical_cols = df.select_dtypes(include=['object',
'category']).columns.tolist()

n_cols = 3
n_rows = math.ceil(len(categorical_cols) / n_cols)

fig, axes = plt.subplots(n_rows, n_cols, figsize=(n_cols * 6, n_rows *
5))
fig.subplots_adjust(hspace=0.6, wspace=0.4)

axes = axes.flatten()

for i, col in enumerate(categorical_cols):
    # Calculate percentages for the current column
    percentage_data = df[col].value_counts(normalize=True) * 100
    percentage_df = percentage_data.reset_index()
    percentage_df.columns = [col, 'Percentage']

    # Sort by percentage for consistent bar order (optional)
    percentage_df = percentage_df.sort_values(by='Percentage',
ascending=False)

    ax = axes[i]

    # Create the bar plot on the current subplot
    sns.barplot(data=percentage_df, x=col, y='Percentage', hue=col,
ax=ax, palette='viridis', legend=False)


    # Add percentage labels on top of the bars
    for p in ax.patches:
        height = p.get_height()
        ax.annotate(f'{height:.1f}%', (p.get_x() + p.get_width()/2.,
height), ha='center', va='bottom')

    ax.set_title(f'Distribution of {col}', fontsize=12) # Dynamic
title formatting
    ax.set_xlabel(column_label_mapping[col], fontsize=10)
    ax.set_ylabel('Percentage (%)', fontsize=10)
    plt.setp(ax.get_xticklabels(), rotation=45, ha='right')
```

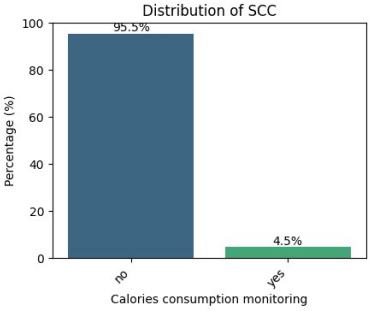


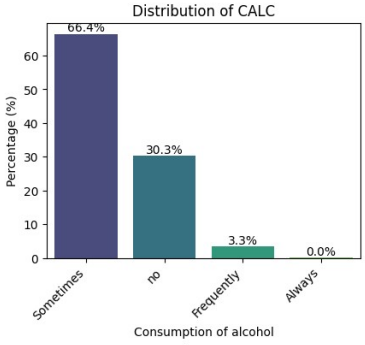


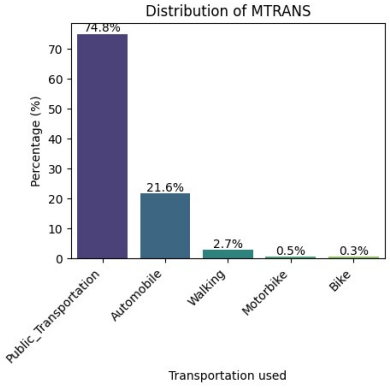


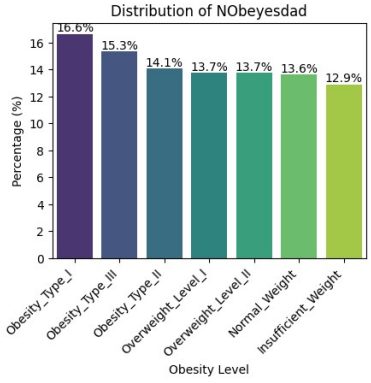












Categorical features observations

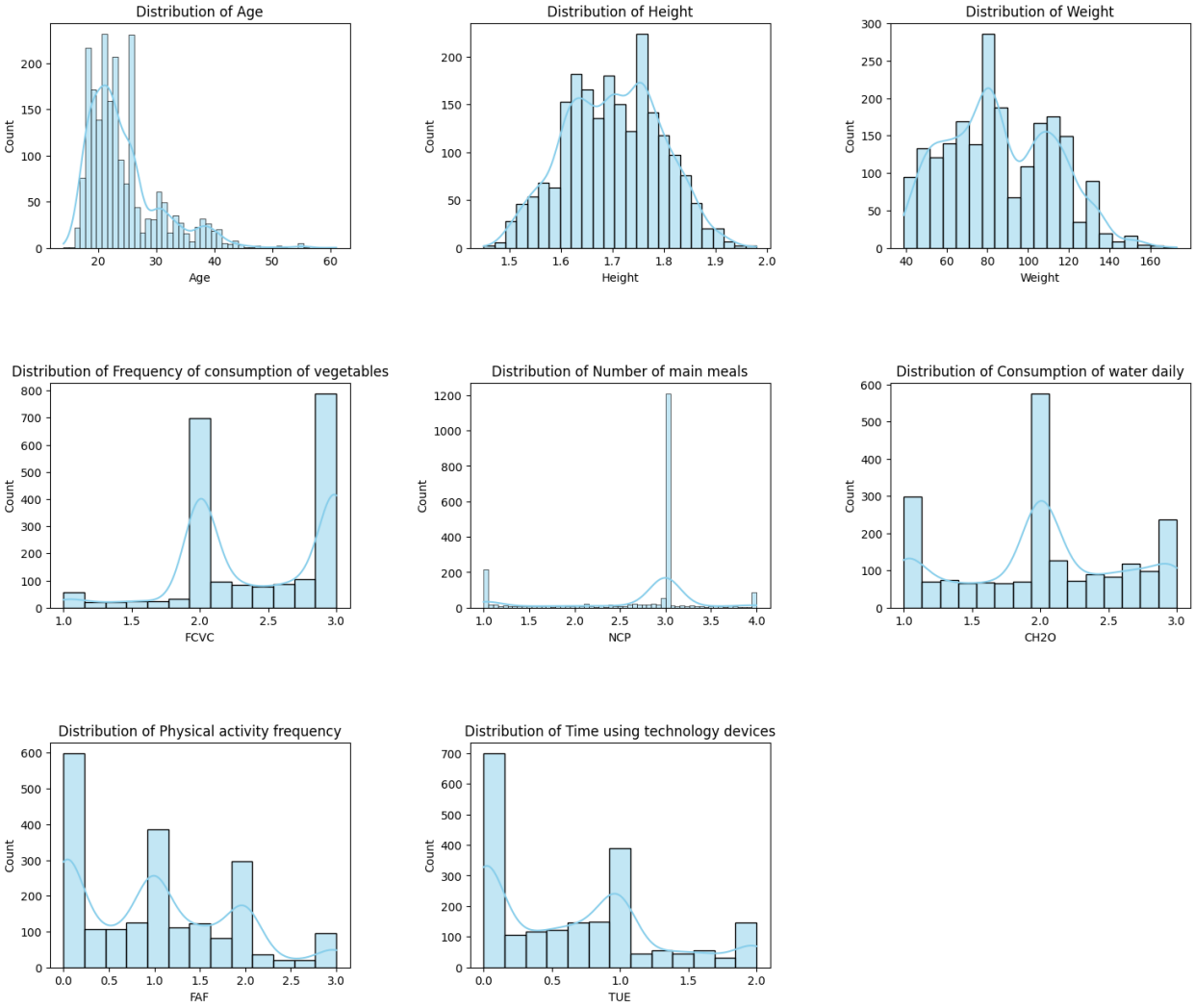
- Gender distribution is fairly equal in the dataset
- 82.0% people have a family history with Overweight
- 88% people frequently consume high calories food
- 84% people consume food between meals
- Only 2% people has smoking habbits
- Only 4.5% people monitores their calories consumptions
- 66% people consume alchol sometimes
- Almost 75% people uses public transport
- Though Obesity type I is higehst with 16.6% but there is fair distribution of obesity levels. Almost every obesity categoreis has people more than 10% of total populations.

Distribution of numerical data

```
df.describe()
```


```
{"summary":"{\n  \"name\": \"df\",\n  \"rows\": 8,\n  \"fields\": [\n
{\n      \"column\": \"Age\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 737.7175023586611,\n
\"min\": 6.3459682737322405,\n        \"max\": 2111.0,\n
\"num_unique_values\": 8,\n        \"samples\": [\n
24.312599908574136,\n          22.77789,\n          2111.0\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Height\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 745.8299972253056,\n        \"min\": 0.09330481986792,\n
\"max\": 2111.0,\n        \"num_unique_values\": 8,\n
\"samples\": [\n          1.7016773533870204,\n          1.700499,\n
2111.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Weight\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 718.4335873262646,\n        \"min\": 26.191171745204688,\n
\"max\": 2111.0,\n        \"num_unique_values\": 8,\n
\"samples\": [\n          86.58605812648035,\n          83.0,\n
2111.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"FCVC\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 745.6275281444425,\n        \"min\": 0.5339265785033023,\n
\"max\": 2111.0,\n        \"num_unique_values\": 7,\n
\"samples\": [\n          2111.0,\n          2.4190430615821885,\n
2.385502\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"NCP\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 745.4871624512587,\n        \"min\": 0.7780386488418594,\n
\"max\": 2111.0,\n        \"num_unique_values\": 7,\n
\"samples\": [\n          2111.0,\n          2.6856280497394596,\n
3.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"CH2O\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 745.7109998154965,\n        \"min\": 0.6129534517968702,\n
\"max\": 2111.0,\n        \"num_unique_values\": 8,\n
\"samples\": [\n          2.0080114040738986,\n          2.0,\n
2111.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"FAF\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 745.9653105606202,\n        \"min\": 0.0,\n        \"max\":
2111.0,\n        \"num_unique_values\": 8,\n        \"samples\": [\n
1.0102976958787304,\n          1.0,\n          2111.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"TUE\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 746.1043827244515,\n
\"min\": 0.0,\n        \"max\": 2111.0,\n
```

```
\"num_unique_values\": 7,\n        \"samples\": [\n          2111.0,\n
0.657865923732828,\n          1.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe"}


# Distribution of numerical data
numerical_cols =
df.select_dtypes(include=['float64','int64']).columns.tolist()

n_cols = 3
n_rows = math.ceil(len(numerical_cols) / n_cols)

fig, axes = plt.subplots(n_rows, n_cols, figsize=(n_cols * 6, n_rows *
5))
fig.subplots_adjust(hspace=0.6, wspace=0.4)

axes = axes.flatten()

for i, col in enumerate(numerical_cols):
    ax = axes[i]
    sns.histplot(data=df, x=col, kde=True, color='skyblue', ax=ax)

    ax.set_title(f'Distribution of {column_label_mapping[col]}',
fontsize=12)
    ax.set_xlabel(col, fontsize=10)
    ax.set_ylabel('Count', fontsize=10)

for j in range(i + 1, len(axes)):
    fig.delaxes(axes[j]) # Delete unused one
```

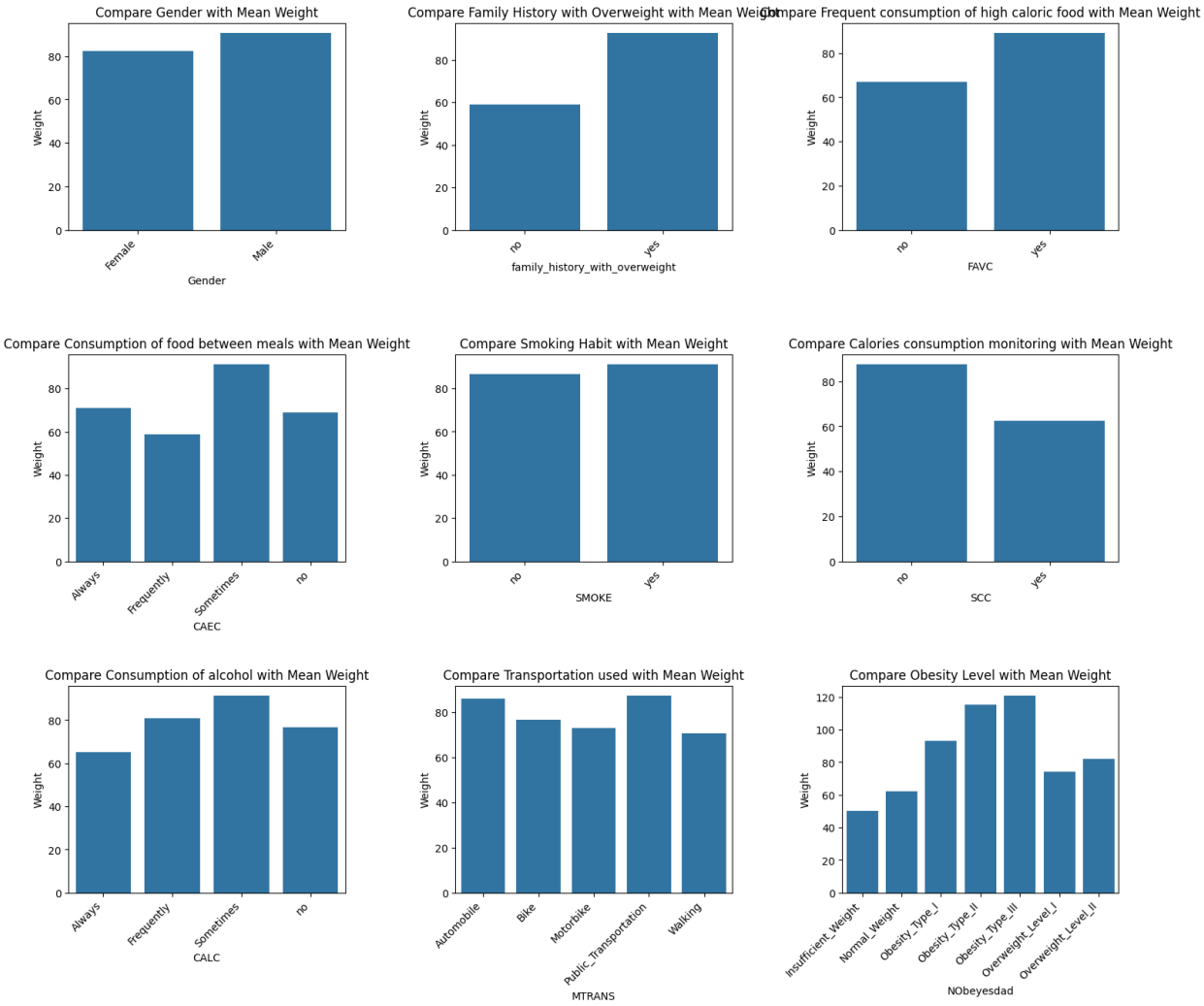


- 22.77 is the median age of people. Majority of people are with age in the range of 15 to 30.
- People height are concentrated between 1.5 to 1.9
- Ignoring the spike, Frequency of consumption of vegetables is low
- Ignoring the spike, Number of main meals is low
- Ignoring the spike, daily consuming water is is ok
- Ignoring the spike, Physical activity is low
- Ignoring the spike, usage of technology device is low

Compare categorical column with mean of weight

```
# Compare categorical column with mean of weight
n_cols = 3
n_rows = math.ceil(len(categorical_cols) / n_cols)

fig, axes = plt.subplots(n_rows, n_cols, figsize=(n_cols * 6, n_rows *
5))
```

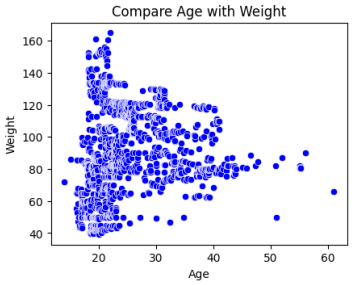
```
fig.subplots_adjust(hspace=0.6, wspace=0.4)

axes = axes.flatten()

for i, col in enumerate(categorical_cols):
    ax = axes[i]
    col_weight_df = df.groupby(df[col]).agg({'Weight':
'mean'}).reset_index()
    sns.barplot(data=col_weight_df, x=col, y='Weight', ax=ax)

    ax.set_title(f'Compare {column_label_mapping[col]} with Mean
Weight', fontsize=12)
    ax.set_xlabel(col, fontsize=10)
    ax.set_ylabel('Weight', fontsize=10)
    plt.setp(ax.get_xticklabels(), rotation=45, ha='right')

for j in range(i + 1, len(axes)):
    fig.delaxes(axes[j]) # Delete unused one
```

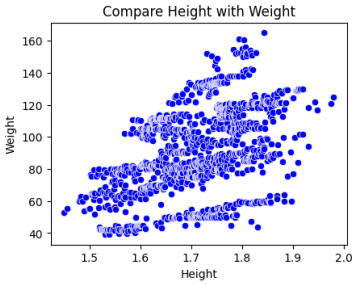


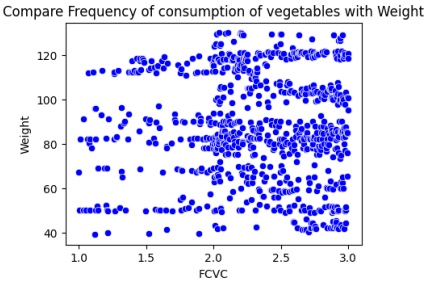
- Mean weight of male are slightly higher than female
- Family history with overweight has very high weight
- Those who frequently consume high calory food has very high weight
- Consume food between meal doesn't have any correlation with weight.
- Those who smokes has higher weight
- People who does not monitor calories has very high weight compared to others.
- Consuming alchohal doesn't have any correlation with high weight
- Mod of transportation also doesn't have any correlation with high weight
- Obesity type III has higher weight

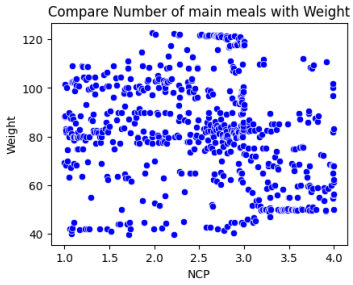
Compare numerical column with mean of weight

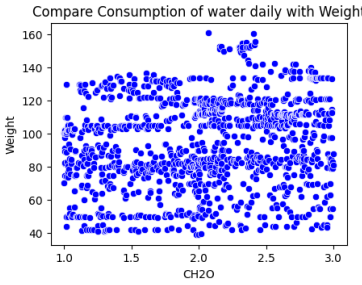
```
# Compare numerical column with mean of weight
target_cols = [
    col for col in numerical_cols
    if col != 'Weight'
    ]
n_cols = 3
n_rows = math.ceil(len(target_cols) / n_cols)

fig, axes = plt.subplots(n_rows, n_cols, figsize=(n_cols * 6, n_rows *
5))
fig.subplots_adjust(hspace=0.6, wspace=0.4)

axes = axes.flatten()

for i, col in enumerate(target_cols):
    if col == 'Weight': # Skip weight
      continue
    ax = axes[i]
    col_weight_df = df.groupby(df[col]).agg({'Weight':
'mean'}).reset_index()
    sns.scatterplot(data=col_weight_df, x=col, y='Weight',
color='blue', ax=ax)

    ax.set_title(f'Compare {column_label_mapping[col]} with Weight',
fontsize=12)
    ax.set_xlabel(col, fontsize=10)
    ax.set_ylabel('Weight', fontsize=10)

for j in range(i + 1, len(axes)):
    fig.delaxes(axes[j]) # Delete unused one
```

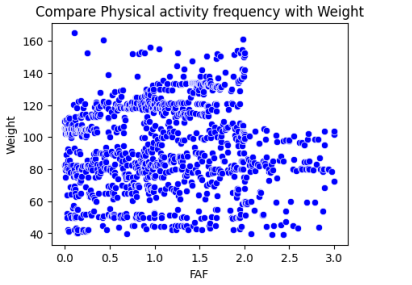


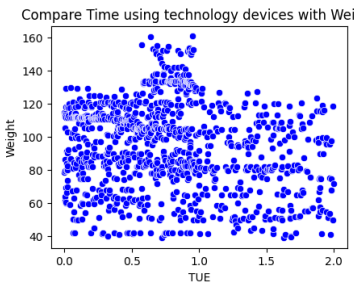












- There is no clear correlation of these features with weight

3. Tasks

Clustering

Goal is to cluster the given dataset into groups based on age group and their eating habbits along with obesity levels.

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.cluster import KMeans

# Numerical Pipeline
numerical_pipeline = Pipeline(steps=[
    ('scaler', StandardScaler())
```

```
])

# Categorical pipeline
categorical_pipeline = Pipeline(steps=[
    ('onehot', OneHotEncoder(handle_unknown='ignore')) # Convert
categories to 0/1 columns
])

preprocessor = ColumnTransformer(
    transformers=[
        ('num', numerical_pipeline, numerical_cols),
        ('cat', categorical_pipeline, categorical_cols)
    ],
    remainder='passthrough'
)

km = KMeans(n_clusters=5, random_state=42, n_init=10) # Added
random_state and n_init for reproducibility


full_pipeline_complex = Pipeline(steps=[
    ('preprocessor', preprocessor), # First, preprocess the data
    ('cluster', km) # Then, apply KMeans clustering
])

# For clustering, we typically use all relevant features
full_pipeline_complex.fit(df) # Fit the entire pipeline to our data

# The cluster labels are generated after the KMeans model is fitted.
cluster_labels = full_pipeline_complex.named_steps['cluster'].labels_

# Add cluster labels back to your original DataFrame
df['Cluster'] = cluster_labels

clusters = df.Cluster.unique()
print("KMean clusters ", clusters)

df.head(20)
```

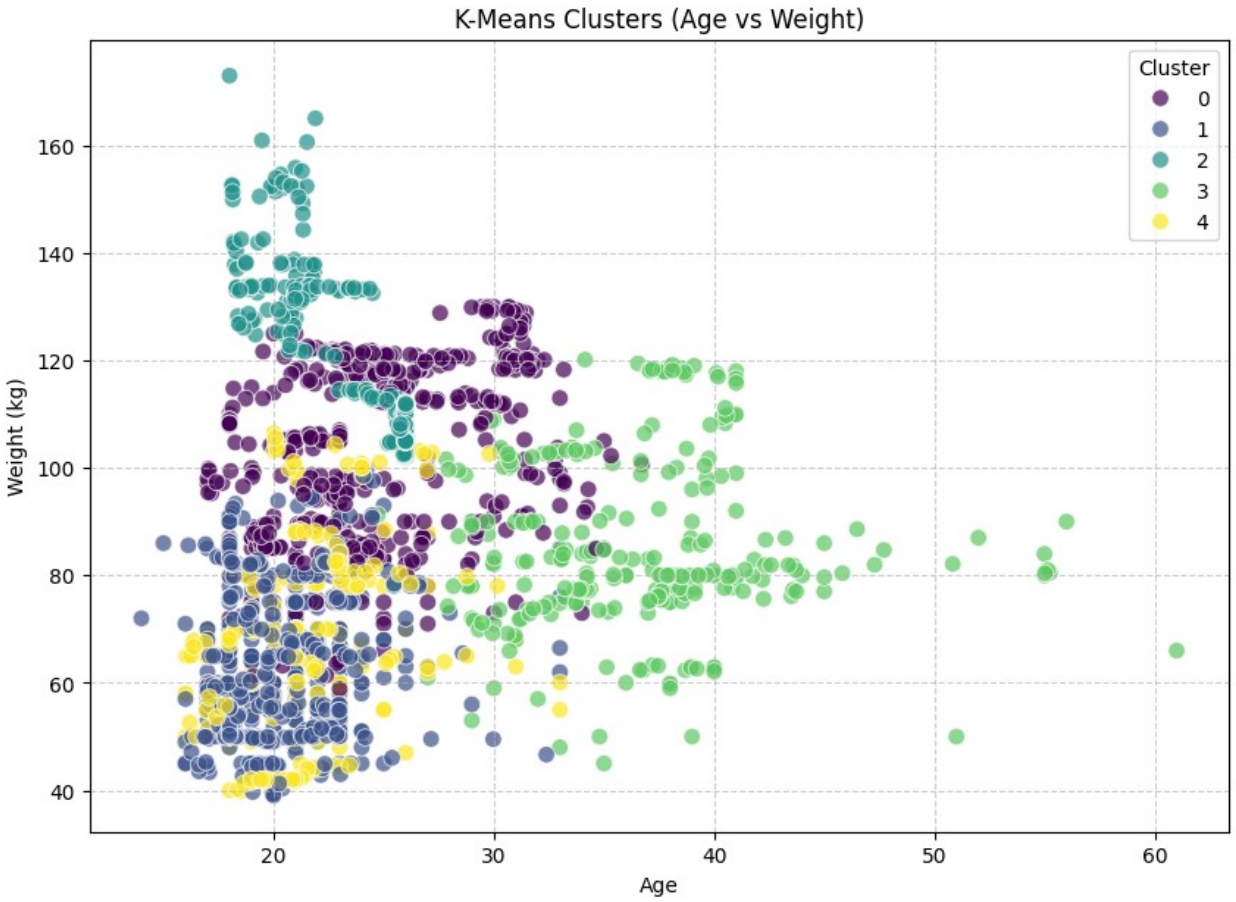
```
KMean clusters  [1 0 4 3 2]
```


```
{"summary":"{\n  \"name\": \"df\",\n  \"rows\": 2111,\n  \"fields\":
[\n    {\n      \"column\": \"Gender\",\n      \"properties\": {\n
\"dtype\": \"category\",\n        \"num_unique_values\": 2,\n
\"samples\": [\n          \"Male\",\n          \"Female\"\n        ],\
n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"Age\",\n      \"properties\": {\
n        \"dtype\": \"number\",\n        \"std\": 6.3459682737322405,\
n        \"min\": 14.0,\n        \"max\": 61.0,\n
\"num_unique_values\": 1402,\n        \"samples\": [\n
25.526746,\n          26.740655\n        ],\n
```

```
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Height\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\": 0.09330481986792,\
n        \"min\": 1.45,\n        \"max\": 1.98,\n
\"num_unique_values\": 1574,\n        \"samples\": [\n
1.760175,\n          1.688436\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"Weight\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 26.191171745204688,\n        \"min\":
39.0,\n        \"max\": 173.0,\n        \"num_unique_values\": 1525,\n
\"samples\": [\n          120.702935,\n          64.4\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"family_history_with_overweight\",\
n      \"properties\": {\n        \"dtype\": \"category\",\n
\"num_unique_values\": 2,\n        \"samples\": [\n          \"no\",\n
\"yes\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"FAVC\",\n      \"properties\": {\n        \"dtype\": \"category\",\n
\"num_unique_values\": 2,\n        \"samples\": [\n          \"yes\",\
n          \"no\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"FCVC\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 0.5339265785033023,\n        \"min\": 1.0,\n        \"max\":
3.0,\n        \"num_unique_values\": 810,\n        \"samples\": [\n
2.987148,\n          2.939727\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"NCP\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0.7780386488418594,\n        \"min\":
1.0,\n        \"max\": 4.0,\n        \"num_unique_values\": 635,\n
\"samples\": [\n          1.468948,\n          2.9948\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"CAEC\",\n      \"properties\": {\n
\"dtype\": \"category\",\n        \"num_unique_values\": 4,\n
\"samples\": [\n          \"Frequently\",\n          \"no\"\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"SMOKE\",\n      \"properties\": {\n        \"dtype\": \"category\",\
n        \"num_unique_values\": 2,\n        \"samples\": [\n
\"yes\",\n          \"no\"\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"CH2O\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0.6129534517968702,\n        \"min\":
1.0,\n        \"max\": 3.0,\n        \"num_unique_values\": 1268,\n
\"samples\": [\n          2.395387,\n          1.983973\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"SCC\",\n      \"properties\": {\n
\"dtype\": \"category\",\n        \"num_unique_values\": 2,\n
\"samples\": [\n          \"yes\",\n          \"no\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
```

```
n    },\n    {\n      \"column\": \"FAF\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 0.8505924308367011,\n
\"min\": 0.0,\n        \"max\": 3.0,\n        \"num_unique_values\":
1190,\n        \"samples\": [\n          1.655488,\n
2.433918\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"TUE\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 0.6089272596763761,\n        \"min\": 0.0,\n        \"max\":
2.0,\n        \"num_unique_values\": 1129,\n        \"samples\": [\n
1.416353,\n          0.878258\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"CALC\",\n      \"properties\": {\n        \"dtype\":
\"category\",\n        \"num_unique_values\": 4,\n        \"samples\":
[\n          \"Sometimes\",\n          \"Always\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"MTRANS\",\n      \"properties\":
{\n        \"dtype\": \"category\",\n        \"num_unique_values\":
5,\n        \"samples\": [\n          \"Walking\",\n
\"Bike\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"NObeyesdad\",\n      \"properties\": {\n        \"dtype\":
\"category\",\n        \"num_unique_values\": 7,\n        \"samples\":
[\n          \"Normal_Weight\",\n          \"Overweight_Level_I\"\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"Cluster\",\n
\"properties\": {\n        \"dtype\": \"int32\",\n
\"num_unique_values\": 5,\n        \"samples\": [\n          0,\n
2\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe","variable_name":"df"}
```


```
# Visualization of Clusters (e.g., Age vs Weight by Cluster)
plt.figure(figsize=(10, 7))
sns.scatterplot(
    data=df,
    x='Age',
    y='Weight',
    hue='Cluster', # Color points by their assigned cluster
    palette='viridis', # Choose a color palette
    s=70, # size of points
    alpha=0.7 # transparency
)
plt.title('K-Means Clusters (Age vs Weight)')
plt.xlabel('Age')
plt.ylabel('Weight (kg)')
plt.grid(True, linestyle='--', alpha=0.6)
plt.show()
```



We can observe these k-clusters as below

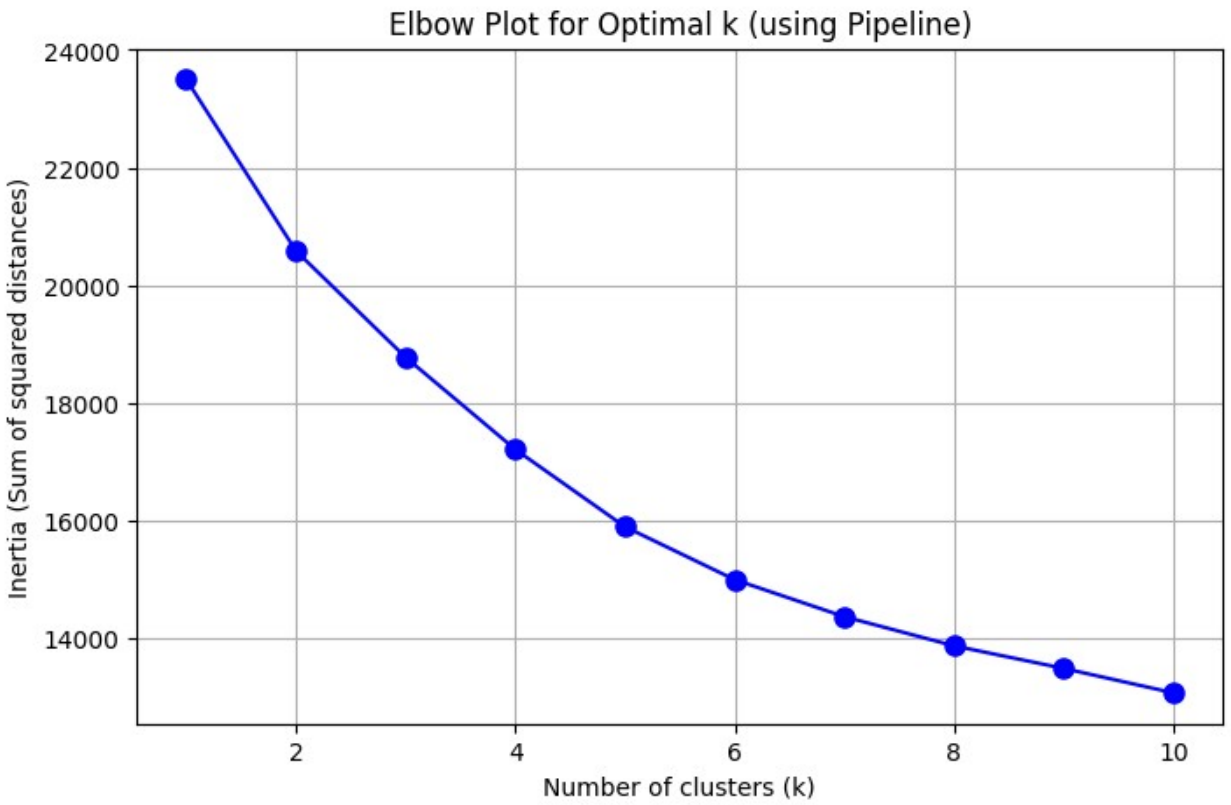
- **Cluster 0: Young, Healthy Habits:** This cluster likely represents younger individuals who exhibit healthy lifestyle choices and have lower cardiovascular risk factors. They tend to be within normal weight ranges, have healthy eating habits, and engage in regular physical activity.
- **Cluster 1: Middle-Aged, Moderate Risk:** This group includes middle-aged individuals with average health indicators. While they may not exhibit significant risk factors, there could be moderate concerns regarding their lifestyle choices, such as occasional unhealthy food consumption.
- **Cluster 2: Older, Higher Risk:** In this cluster, we find older individuals with a combination of unhealthy behaviors and risk factors. These individuals likely struggle with excess weight, have poor dietary habits, and do not engage in regular exercise.
- **Cluster 3: Younger, Unhealthy Lifestyle:** Here, individuals tend to be younger, but they've already developed unhealthy lifestyle habits that put them at risk. This might include things like poor eating habits and a sedentary lifestyle.
- **Cluster 4: Established Risk Factors:** This cluster consists of individuals with a long history of unhealthy habits, potentially leading to more pronounced risk factors.

Calculate the inertia for different values of k

In-order to try different value for `k`, we will beed to modify our pipeline as below.

```python
# Calculate the inertia for different values of k
import numpy as np
inertia = []
k_values = range(1, 11)
df.drop(columns=['Cluster'], inplace=True, axis=1, errors='ignore')
for k in k_values:
    # Create a new pipeline for each k, including preprocessing and
KMeans
    kmeans_pipeline = Pipeline(steps=[
        ('preprocessor', preprocessor), # First, preprocess the data
        ('cluster', KMeans(n_clusters=k, n_init=10, random_state=42))
# Then, apply KMeans
    ])

    # Fit the pipeline
    kmeans_pipeline.fit(df)

    # Get the inertia from the KMeans model within the pipeline
    inertia.append(kmeans_pipeline.named_steps['cluster'].inertia_)
    print(f"  k={k}: Inertia = {inertia[-1]:.2f}")

# Plot the Elbow Plot ---
plt.figure(figsize=(8, 5))
plt.plot(k_values, inertia, 'bo-', markersize=8)
plt.xlabel('Number of clusters (k)')
plt.ylabel('Inertia (Sum of squared distances)')
plt.title('Elbow Plot for Optimal k (using Pipeline)')
plt.grid(True)
plt.show()
```

```
  k=1: Inertia = 23501.24
  k=2: Inertia = 20584.15
  k=3: Inertia = 18773.26
  k=4: Inertia = 17206.47
  k=5: Inertia = 15891.42
  k=6: Inertia = 14991.51
  k=7: Inertia = 14362.00
  k=8: Inertia = 13866.59
  k=9: Inertia = 13482.58
  k=10: Inertia = 13066.53
```



Optimized value for k

The inertia is decreasing for every k, but the key is the *rate* of decrease. The "elbow" signifies the point where the decrease in inertia starts to slow down noticeably. While inertia continues to decrease even after k=5, the reduction is not as substantial as it is before that point. There **k=5** is optimal value here.

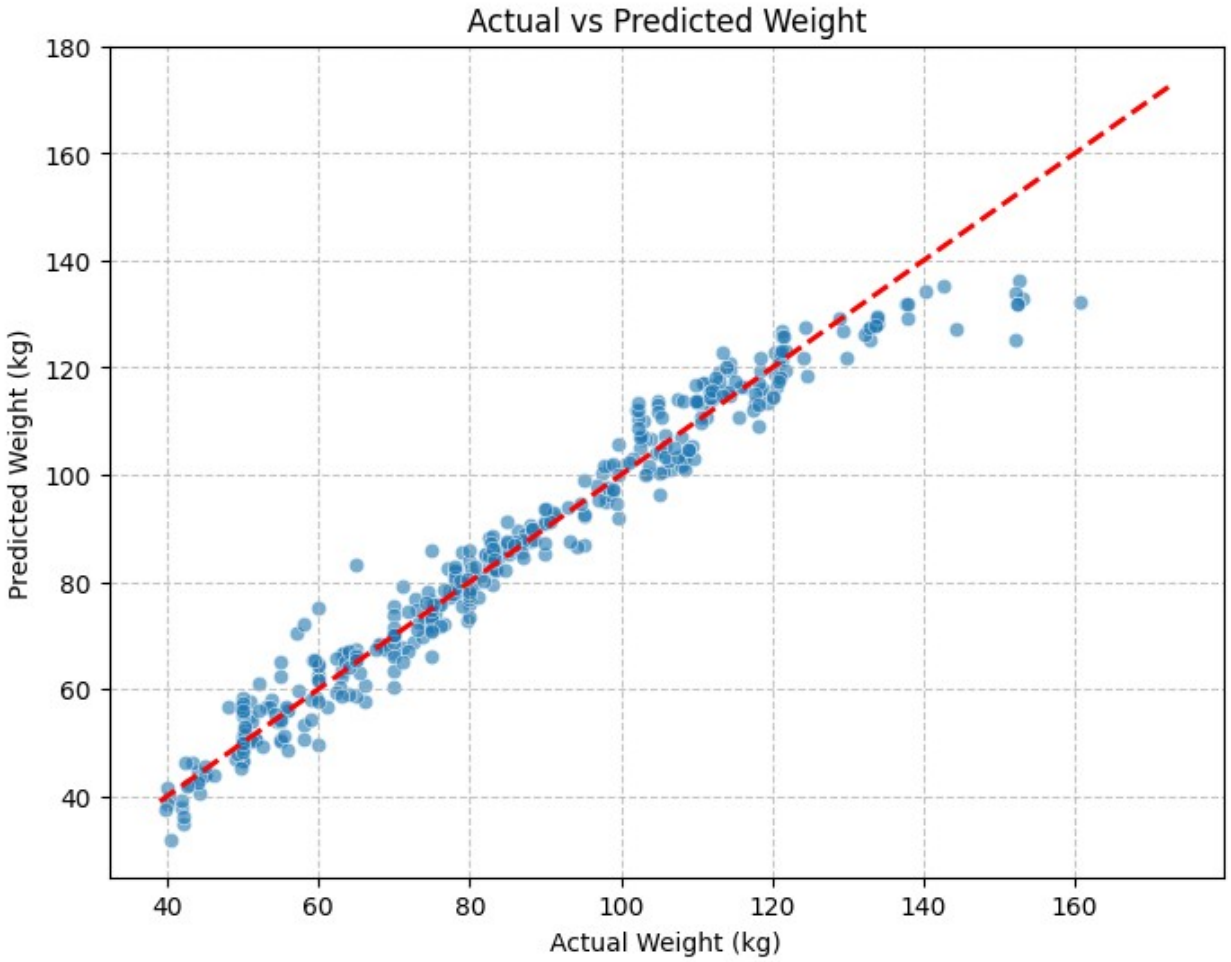
Regression

Goal is to predict the weight of people based on their eating habbits and family obesity history and other features.

```
# Regression model
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split

# Numerical Pipeline
numerical_pipeline = Pipeline(steps=[
    ('scaler', StandardScaler())
])

# Categorical pipeline
categorical_pipeline = Pipeline(steps=[
    ('onehot', OneHotEncoder(handle_unknown='ignore')) # Convert
categories to 0/1 columns
```

```
])

# Remvoe Weight as no preprocessing is required on Weight
numerical_cols_without_weight = [col for col in numerical_cols if col
!= 'Weight']

preprocessor = ColumnTransformer(
    transformers=[
        ('num', numerical_pipeline, numerical_cols_without_weight),
        ('cat', categorical_pipeline, categorical_cols)
    ],
    remainder='passthrough'
)

model = LinearRegression()

regression_pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor), # First, preprocess the data
    ('regression', model) # Then, apply regression
])

# Separate Features (X) and Target (y) for training
X = df.drop(columns=['Weight'])
y = df['Weight']

# Split data and fit the pipeline
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

regression_pipeline.fit(X_train, y_train)

y_pred = regression_pipeline.predict(X_test)
train_score = regression_pipeline.score(X_train, y_train)
test_score = regression_pipeline.score(X_test, y_test)

print(f"\nTraining R^2 Score: {train_score:.4f}")
print(f"Test R^2 Score: {test_score:.4f}")

# Visualize Predictions vs Actuals
plt.figure(figsize=(8, 6))
sns.scatterplot(x=y_test, y=y_pred, alpha=0.6)
plt.plot([y.min(), y.max()], [y.min(), y.max()], 'r--', lw=2) # Line
for perfect prediction
plt.xlabel('Actual Weight (kg)')
plt.ylabel('Predicted Weight (kg)')
plt.title('Actual vs Predicted Weight')
plt.grid(True, linestyle='--', alpha=0.7)
plt.show()
```

```
Training R^2 Score: 0.9630
Test R^2 Score: 0.9620
```



Observation

- Linear regression has performed well on both train and test data with accuracy of > 96%
- sklearn pipeline model helped to apply preprocessing on train and test data

Classification

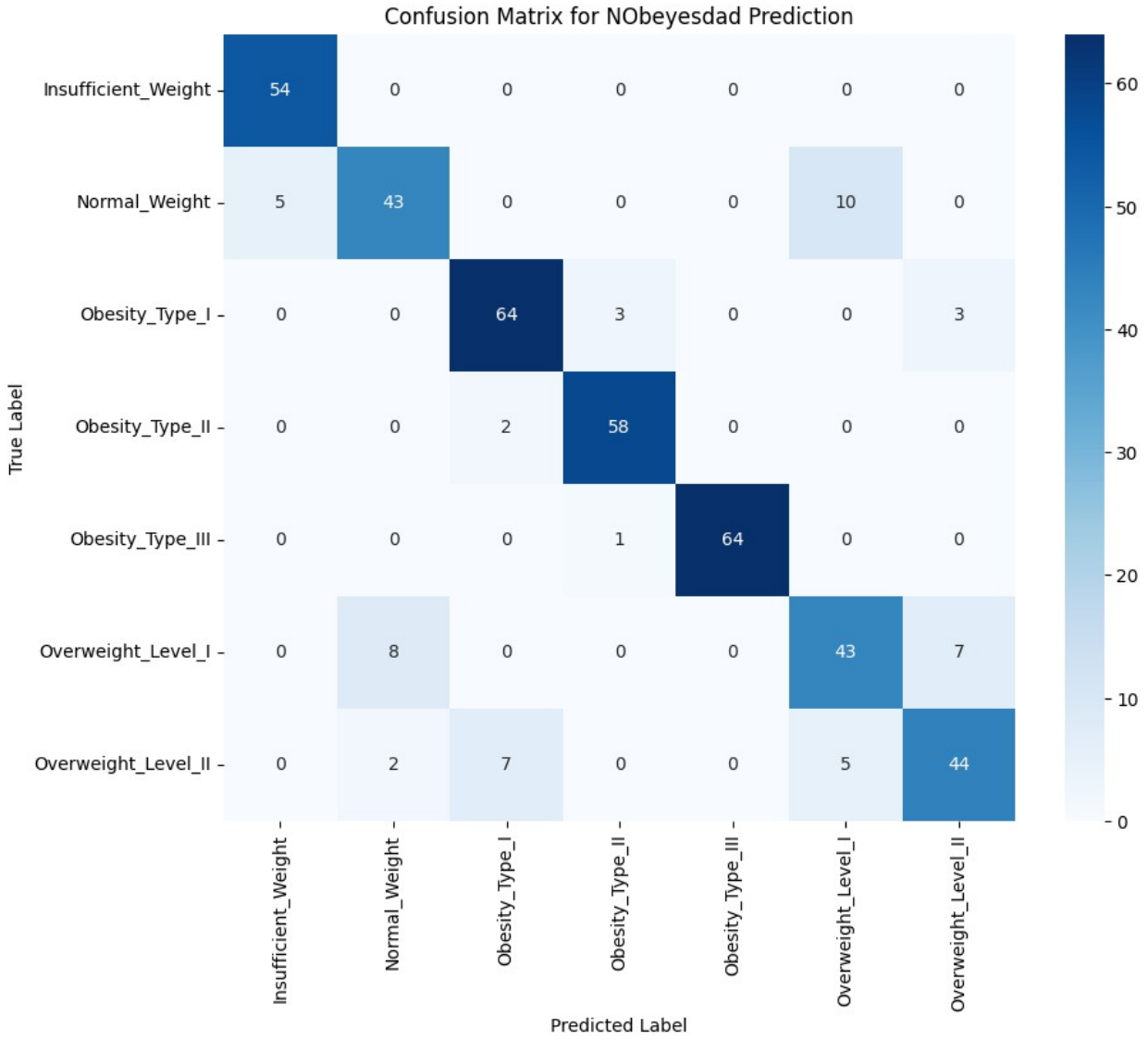
Goal is to predict the Obesity Level for given people's data.

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix


# Numerical Pipeline
numerical_pipeline = Pipeline(steps=[
```

```
    ('scaler', StandardScaler())
])

# Categorical pipeline
categorical_pipeline = Pipeline(steps=[
    ('onehot', OneHotEncoder(handle_unknown='ignore')) # Convert
categories to 0/1 columns
])

# Remove NObeyesdad as it is a target column and do not need similar
preprocessing
categorical_cols_cols_without_NObeyesdad = [col for col in
categorical_cols if col != 'NObeyesdad']

preprocessor = ColumnTransformer(
    transformers=[
        ('num', numerical_pipeline, numerical_cols),
        ('cat', categorical_pipeline,
categorical_cols_cols_without_NObeyesdad)
    ],
    remainder='passthrough'
)

model = LogisticRegression(multi_class='multinomial', solver='lbfgs',
max_iter=1000, random_state=42)

classification_pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('classifier', model)
])

# Separate Features (X) and Target (y) for training
X = df.drop(columns=['NObeyesdad'])
y = df['NObeyesdad']

# Split data and fit the pipeline
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42, stratify=y)
classification_pipeline.fit(X_train, y_train)

# Make predictions using the fitted pipeline
predicted_obesity_levels = classification_pipeline.predict(X_test)

y_pred = classification_pipeline.predict(X_test)

train_accuracy = classification_pipeline.score(X_train, y_train)
test_accuracy = classification_pipeline.score(X_test, y_test)

print(f"\nTraining Accuracy Score: {train_accuracy:.4f}")
print(f"Test Accuracy Score: {test_accuracy:.4f}")
```

```python
print("\nClassification Report (Test Set):")
print(classification_report(y_test, y_pred))

# Confusion Matrix visualization
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(10, 8))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=classification_pipeline.classes_,
            yticklabels=classification_pipeline.classes_)
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix for NObeyesdad Prediction')
plt.show()
```

```
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/
_logistic.py:1247: FutureWarning: 'multi_class' was deprecated in
version 1.5 and will be removed in 1.7. From then on, it will always
use 'multinomial'. Leave it to its default value to avoid this
warning.
  warnings.warn(


Training Accuracy Score: 0.9076
Test Accuracy Score: 0.8747

Classification Report (Test Set):
                     precision    recall  f1-score   support

Insufficient_Weight       0.92      1.00      0.96        54
      Normal_Weight       0.81      0.74      0.77        58
     Obesity_Type_I       0.88      0.91      0.90        70
    Obesity_Type_II       0.94      0.97      0.95        60
   Obesity_Type_III       1.00      0.98      0.99        65
 Overweight_Level_I       0.74      0.74      0.74        58
Overweight_Level_II       0.81      0.76      0.79        58

           accuracy                           0.87       423
          macro avg       0.87      0.87      0.87       423
       weighted avg       0.87      0.87      0.87       423
```



Classification Overall Performance

Accuracy: 0.87 (87%): This is the overall accuracy of our model, meaning 87% of all the predictions made on the test set were correct. This is a very good overall accuracy for a multi-class classification problem.

Precision: "Of all the times the model predicted this class, how many were actually correct?" (Minimizes False Positives)

- Excellent Precision (very few false positives):

Obesity_Type_III: 1.00 (Perfect! When it predicts this, it's always right.)

Obesity_Type_II: 0.94

Insufficient_Weight: 0.92

- Good Precision:

Obesity_Type_I: 0.88

Moderately Lower Precision:

Normal_Weight: 0.81

- Overweight_Level_II: 0.81

Overweight_Level_I: 0.74 (This is the lowest. The model sometimes misclassifies other types as Overweight_Level_I.)

Recall (Sensitivity): "Of all the actual instances of this class, how many did the model correctly identify?" (Minimizes False Negatives)

- Excellent Recall (few false negatives):

Insufficient_Weight: 1.00 (Perfect! It found all instances of this class.)

Obesity_Type_III: 0.98

Obesity_Type_II: 0.97

Obesity_Type_I: 0.91

- Moderately Lower Recall:

Overweight_Level_I: 0.74

Normal_Weight: 0.74

Overweight_Level_II: 0.76 (The model missed some actual instances of these classes.)

F1-Score: The harmonic mean of precision and recall. A high F1-score means the model has good balance of precision and recall.

- Strong Performance:

Insufficient_Weight: 0.96

Obesity_Type_II: 0.95

Obesity_Type_III: 0.99

Obesity_Type_I: 0.90

- Relatively Weaker Performance:

Normal_Weight: 0.77

Overweight_Level_I: 0.74

Overweight_Level_II: 0.79

4. Reflection and Discussion

Q1. How does the choice of supervised or unsupervised learning depend on the dataset and problem?

Ans. When I applied K-Mean clustering algorithm which is a unsupervised model, I didn't had to worry about defining the target column. It made applying algorithm easily. Compared to applying Regression and Classifications which are supervised model, I had to made sure that I have defined target columns. Therefore it highly depends on dataset to make sure we have right target columns.

Q2. What challenges did you anticipate when applying each modeling technique?

Ans. I had anticipated both data cleaning (missing values etc) and encoding. But dataset I choose didn't have any missing values therefore I didn't have to apply sklearn pipeline preprocessing.

Key take aways

- It took lot of effort to plot many graphs to understand the data
- Using sklearn Pipeline simplified the complexity and also avoid to write duplicate code for test vs train data.