Start coding or generate with AI.

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split
import os
```

Directotry

```
# Upload your dataset CSV (for example: nhanes_2017_2020.csv) via Colab file upload or specify path if dow
from google.colab import files
uploaded = files.upload()  # Upload the CSV file here manually
```

Choose Files demographic.csv (1).zip

- **demographic.csv (1).zip**(application/x-zip-compressed) - 376594 bytes, last modified: 7/3/2025 - 100% done

```
Saving demographic.csv (1).zip to demographic.csv (1).zip
```

download the file from kaggle

```
filename = list(uploaded.keys())[0]
print(f"Loaded file: {filename}")

df = pd.read_csv(filename)
```

```
Loaded file: demographic.csv (1).zip
```

```
name the file
```

```
# Step 4: Explore data columns and preview
print("Columns in dataset:")
print(df.columns)
print("\nFirst 5 rows:")
print(df.head())
```

```
Columns in dataset:
Index(['SEQN', 'SDDSRVYR', 'RIDSTATR', 'RIAGENDR', 'RIDAGEYR', 'RIDAGEMN',
       'RIDRETH1', 'RIDRETH3', 'RIDEXMON', 'RIDEXAGM', 'DMQMILIZ', 'DMQADFC',
       'DMDBORN4', 'DMDCITZN', 'DMDYRSUS', 'DMDEDUC3', 'DMDEDUC2', 'DMDMARTL',
       'RIDEXPRG', 'SIALANG', 'SIAPROXY', 'SIAINTRP', 'FIALANG', 'FIAPROXY',
       'FIAINTRP', 'MIALANG', 'MIAPROXY', 'MIAINTRP', 'AIALANGA', 'DMDHHSIZ',
       'DMDFMSIZ', 'DMDHHSZA', 'DMDHHSZB', 'DMDHHSZE', 'DMDHRGND', 'DMDHRAGE',
```

```
        'DMDHRBR4', 'DMDHREDU', 'DMDHRMAR', 'DMDHSEDU', 'WTINT2YR', 'WTMEC2YR',
        'SDMVPSU', 'SDMVSTRA', 'INDHHIN2', 'INDFMIN2', 'INDFMPIR'],
       dtype='object')

First 5 rows:
    SEQN  SDDSRVYR  RIDSTATR  RIAGENDR  RIDAGEYR  RIDAGEMN  RIDRETH1  \
0  73557         8         2         1        69       NaN         4
1  73558         8         2         1        54       NaN         3
2  73559         8         2         1        72       NaN         3
3  73560         8         2         1         9       NaN         3
4  73561         8         2         2        73       NaN         3

   RIDRETH3  RIDEXMON  RIDEXAGM  ...  DMDHREDU  DMDHRMAR  DMDHSEDU  \
0         4       1.0       NaN  ...       3.0       4.0       NaN
1         3       1.0       NaN  ...       3.0       1.0       1.0
2         3       2.0       NaN  ...       4.0       1.0       3.0
3         3       1.0     119.0  ...       3.0       1.0       4.0
4         3       1.0       NaN  ...       5.0       1.0       5.0

       WTINT2YR      WTMEC2YR  SDMVPSU  SDMVSTRA  INDHHIN2  INDFMIN2  INDFMPIR
0  13281.237386  13481.042095        1       112       4.0       4.0      0.84
1  23682.057386  24471.769625        1       108       7.0       7.0      1.78
2  57214.803319  57193.285376        1       109      10.0      10.0      4.51
3  55201.178592  55766.512438        2       109       9.0       9.0      2.52
4  63709.667069  65541.871229        2       116      15.0      15.0      5.00

[5 rows x 47 columns]
```

Select relevant columns

```
print("Columns in dataset:")
print(df.columns.tolist())
```

```
Columns in dataset:
['SEQN', 'SDDSRVYR', 'RIDSTATR', 'RIAGENDR', 'RIDAGEYR', 'RIDAGEMN', 'RIDRETH1', 'RIDRETH3', 'RIDEXMON
```

```
relevant_columns = ['RIDAGEYR', 'BMXHT', 'BMXWT', 'PAQ620']

# Check which of these columns exist in the dataframe
existing_columns = [col for col in relevant_columns if col in df.columns]
print(f"Using columns: {existing_columns}")

# Select only existing columns
data = df[existing_columns].copy()

# Rename columns to standard names for convenience
rename_dict = {
    'RIDAGEYR': 'Age',
    'BMXHT': 'Height_cm',
    'BMXWT': 'Weight_kg',
    'PAQ605': 'Exercise_Level',
    'PAQ620': 'Exercise_Level'  # adjust key according to your dataset
}

# Rename only the columns present
data.rename(columns={k: v for k, v in rename_dict.items() if k in data.columns}, inplace=True)
```

```
print("\nSelected data preview:")
print(data.head())
```

```
Using columns: ['RIDAGEYR']

Selected data preview:
   Age
0   69
1   54
2   72
3    9
4   73
```

Drop rows with missing values

```
data.dropna(inplace=True)
```

Filter Exercise_Level and encode it

```
if 'Exercise_Level' in data.columns:
    # Keep only rows with Exercise_Level 1 or 2 (yes or no)
    data = data[data['Exercise_Level'].isin([1, 2])]

    # Use .loc to avoid SettingWithCopyWarning and map values
    data.loc[:, 'Exercise_Level'] = data.loc[:, 'Exercise_Level'].map({1: 2, 2: 1})  # Active=2, Inactive=
else:
    print("Warning: 'Exercise_Level' column not found in dataset. Skipping filtering and mapping.")
```

```
Warning: 'Exercise_Level' column not found in dataset. Skipping filtering and mapping.
```

```
# Step 8: Preview cleaned data
print("\nCleaned data sample:")
print(data.head())
```

```
Cleaned data sample:
   Age
0   69
1   54
2   72
3    9
4   73
```

Define features , target and split data

```
# Step 1: Check all columns in your original dataframe
print("Columns in original dataframe:")
print(df.columns.tolist())

# Step 2: Look for weight-related columns (commonly BMXWT or similar)
weight_candidates = [col for col in df.columns if 'WT' in col.upper()]
print("Possible weight columns found:")
print(weight_candidates)
```

```
# Step 3: Define candidate columns for Age, Height, Weight, Exercise
candidate_cols = {
    'Age': 'RIDAGEYR',
    'Height_cm': 'BMXHT',
    'Weight_kg': None,         # To find below
    'Exercise_Level': None     # To find below
}

# Step 4: Select weight column from candidates found
for w_col in weight_candidates:
    if w_col in df.columns:
        candidate_cols['Weight_kg'] = w_col
        print(f"Using weight column: {w_col}")
        break

if candidate_cols['Weight_kg'] is None:
    raise KeyError("No weight column found in dataset. Cannot proceed.")

# Step 5: Find exercise column if exists
possible_exercise_cols = ['PAQ605', 'PAQ620', 'PAQ665', 'PAQ650']
for col in possible_exercise_cols:
    if col in df.columns:
        candidate_cols['Exercise_Level'] = col
        print(f"Using exercise column: {col}")
        break

# Step 6: Build list of columns that actually exist
existing_cols = [col for col in candidate_cols.values() if col is not None and col in df.columns]
print("Columns to select from dataframe:")
print(existing_cols)

# Step 7: Select columns
data = df[existing_cols].copy()

# Step 8: Rename to standard names
rename_map = {v: k for k, v in candidate_cols.items() if v is not None}
data.rename(columns=rename_map, inplace=True)

print("\nColumns after renaming:")
print(data.columns.tolist())

# Step 9: Drop rows with missing values
data.dropna(inplace=True)
print(f"Data shape after dropping NA: {data.shape}")

# Step 10: Filter Exercise_Level if it exists
if 'Exercise_Level' in data.columns:
    data = data[data['Exercise_Level'].isin([1, 2])]
    data.loc[:, 'Exercise_Level'] = data.loc[:, 'Exercise_Level'].map({1: 2, 2: 1})
else:
    print("Exercise_Level column missing; proceeding without it.")

print(f"Data shape after filtering Exercise_Level: {data.shape}")

# Step 11: Check if essential columns exist
essential_cols = ['Age', 'Weight_kg']
for col in essential_cols:
    if col not in data.columns:
```

```
        raise KeyError(f"Essential column '{col}' missing from data. Cannot proceed.")

# Step 12: Define features (Height_cm optional)
features = ['Age']
if 'Height_cm' in data.columns:
    features.append('Height_cm')
if 'Exercise_Level' in data.columns:
    features.append('Exercise_Level')

print(f"Features used: {features}")

X = data[features]
y = data['Weight_kg']

print(f"Feature matrix shape: {X.shape}")
print(f"Target vector shape: {y.shape}")

# Step 13: Split data
from sklearn.model_selection import train_test_split

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

print(f"Training samples: {X_train.shape[0]}, Testing samples: {X_test.shape[0]}")
```

```
Columns in original dataframe:
['SEQN', 'SDDSRVYR', 'RIDSTATR', 'RIAGENDR', 'RIDAGEYR', 'RIDAGEMN', 'RIDRETH1', 'RIDRETH3', 'RIDEXMON
Possible weight columns found:
['WTINT2YR', 'WTMEC2YR']
Using weight column: WTINT2YR
Columns to select from dataframe:
['RIDAGEYR', 'WTINT2YR']

Columns after renaming:
['Age', 'Weight_kg']
Data shape after dropping NA: (10175, 2)
Exercise_Level column missing; proceeding without it.
Data shape after filtering Exercise_Level: (10175, 2)
Features used: ['Age']
Feature matrix shape: (10175, 1)
Target vector shape: (10175,)
Training samples: 8140, Testing samples: 2035
```

Train Linear Regression model

```
model = LinearRegression()
model.fit(X_train, y_train)
```

```
▾ LinearRegression
LinearRegression()
```

Predict on test data

```
y_pred = model.predict(X_test)
```

Calculate Mean Squared Error

```
mse = mean_squared_error(y_test, y_pred)
print(f"\nMean Squared Error (MSE) on Test Set: {mse:.2f}")
```

```
Mean Squared Error (MSE) on Test Set: 717804367.81
```

Plot Actual vs Predicted

```
plt.figure(figsize=(7,7))
sns.scatterplot(x=y_test, y=y_pred)
plt.xlabel('Actual Weight (kg)')
plt.ylabel('Predicted Weight (kg)')
plt.title('Actual vs Predicted Weight')
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--')
plt.show()
```