

```
pip install ucimlrepo
```

```
Collecting ucimlrepo
  Downloading ucimlrepo-0.0.7-py3-none-any.whl.metadata (5.5 kB)
Requirement already satisfied: pandas>=1.0.0 in /usr/local/lib/python3.11/dist-packages (from ucimlrepo) (2.2.2)
Requirement already satisfied: certifi>=2020.12.5 in /usr/local/lib/python3.11/dist-packages (from pandas>=1.0.0->ucimlrepo) (2025.1.31)
Requirement already satisfied: numpy>=1.23.2 in /usr/local/lib/python3.11/dist-packages (from pandas>=1.0.0->ucimlrepo) (2.0.2)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.11/dist-packages (from pandas>=1.0.0->ucimlrepo) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-packages (from pandas>=1.0.0->ucimlrepo) (2025.1)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-packages (from pandas>=1.0.0->ucimlrepo) (2025.1)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.8.2->pandas>=1.0.0->ucimlrepo) (1.17.0)
Downloading ucimlrepo-0.0.7-py3-none-any.whl (8.0 kB)
Installing collected packages: ucimlrepo
Successfully installed ucimlrepo-0.0.7
```

```
#First two steps are to bring in the data from the referenced website
from ucimlrepo import fetch_ucirepo
```

```
national_health_and_nutrition_health_survey_2013_2014_nhanes_age_prediction_subset = fetch_ucirepo(id=887)
```

```
X = national_health_and_nutrition_health_survey_2013_2014_nhanes_age_prediction_subset.data.features
y = national_health_and_nutrition_health_survey_2013_2014_nhanes_age_prediction_subset.data.targets
```

```
print(national_health_and_nutrition_health_survey_2013_2014_nhanes_age_prediction_subset.metadata)
```

```
print(national_health_and_nutrition_health_survey_2013_2014_nhanes_age_prediction_subset.variables)
```

```
{'uci_id': 887, 'name': 'National Health and Nutrition Health Survey 2013-2014 (NHANES) Age Prediction Subset', 'repository_url': 'https://ucimlrepo.org/dataset/887'}

name      role      type demographic \
0      SEQN      ID      Continuous      None
1  age_group  Target  Categorical      Age
2  RIDAGEYR  Other  Continuous      Age
3  RIAGENDR  Feature  Continuous      Gender
4  PAQ605    Feature  Continuous      None
5  BMXBMI    Feature  Continuous      None
6  LBXGLU    Feature  Continuous      None
7  DIQ010    Feature  Continuous      None
8  LBXGLT    Feature  Continuous      None
9  LBXIN     Feature  Continuous      None

description units missing_values
0      Respondent Sequence Number      None      no
1      Respondent's Age Group (senior/non-senior)      None      no
2      Respondent's Age      None      no
3      Respondent's Gender      None      no
4  If the respondent engages in moderate or vigorous physical activity      None      no
5      Respondent's Body Mass Index      None      no
6      Respondent's Blood Glucose after fasting      None      no
7      If the Respondent is diabetic      None      no
8      Respondent's Oral Glucose Tolerance Test      None      no
9      Respondent's Blood Insulin Levels      None      no
```

```
import pandas as pd
import numpy as np
```

```
#Create a dataframe with Features and Targets
national = national_health_and_nutrition_health_survey_2013_2014_nhanes_age_prediction_subset
```

```
df = pd.DataFrame(national.data.features)
```

```
df = df.assign(age_group=national.data.targets)
```

```
df = df.assign(RIDAGEYR=national.data.targets)
```

```
#Index the insulin levels to create a new column with assigned labels based off the value in the insulin column
df['Category'] = pd.cut(df['LBXIN'],
                        bins=[0, 10, 15, 30, 110],
                        labels=['Low', 'Medium', 'High', 'Very High'],
                        right=False,
                        include_lowest=True)
```

```
df.head()
```

	RIAGENDR	PAQ605	BMXBMI	LBXGLU	DIQ010	LBXGLT	LBXIN	age_group	RIDAGEYR	Category
0	2.0	2.0	35.7	110.0	2.0	150.0	14.91	Adult	Adult	Medium
1	2.0	2.0	20.3	89.0	2.0	80.0	3.85	Adult	Adult	Low
2	1.0	2.0	23.2	89.0	2.0	68.0	6.14	Adult	Adult	Low
3	1.0	2.0	28.9	104.0	2.0	84.0	16.15	Adult	Adult	High
4	2.0	1.0	35.9	103.0	2.0	81.0	10.92	Adult	Adult	Medium

Next steps: [Generate code with df](#) [View recommended plots](#) [New interactive sheet](#)

```
#Using Label encoding to change Adult and Senior into 1 and 2 for the Cross Validation
d = {'Adult': 1, 'Senior': 2}
df['age_group'] = df['age_group'].map(d)
df['RIDAGEYR'] = df['RIDAGEYR'].map(d)
print(df)
```

	RIAGENDR	PAQ605	BMXBMI	LBXGLU	DIQ010	LBXGLT	LBXIN	age_group	\
0	2.0	2.0	35.7	110.0	2.0	150.0	14.91	1	
1	2.0	2.0	20.3	89.0	2.0	80.0	3.85	1	
2	1.0	2.0	23.2	89.0	2.0	68.0	6.14	1	
3	1.0	2.0	28.9	104.0	2.0	84.0	16.15	1	
4	2.0	1.0	35.9	103.0	2.0	81.0	10.92	1	
...	...	...	...	...	...	...	...	...	
2273	2.0	2.0	33.5	100.0	2.0	73.0	6.53	1	
2274	1.0	2.0	30.0	93.0	2.0	208.0	13.02	1	
2275	1.0	2.0	23.7	103.0	2.0	124.0	21.41	1	
2276	2.0	2.0	27.4	90.0	2.0	108.0	4.99	1	
2277	1.0	2.0	24.5	108.0	2.0	108.0	3.76	1	

	RIDAGEYR	Category
0	1	Medium
1	1	Low
2	1	Low
3	1	High
4	1	Medium
...	...	...
2273	1	Low
2274	1	Medium
2275	1	High
2276	1	Low
2277	1	Low

[2278 rows x 10 columns]

```
import pandas
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier
import matplotlib.pyplot as plt

#To give more familiar names to the column headers
df.rename(columns={'RIAGENDR': 'Gender', 'PAQ605': 'Exercise', 'BMXBMI': 'BMI', 'LBXGLU': 'Glucose', 'DIQ010': 'Diabetic', 'LBXGLT': 'Oral_Glucose', 'LBXIN': 'Insulin'})

print(df)
```

	Gender	Exercise	BMI	Glucose	Diabetic	Oral_Glucose	Insulin	\
0	2.0	2.0	35.7	110.0	2.0	150.0	14.91	
1	2.0	2.0	20.3	89.0	2.0	80.0	3.85	
2	1.0	2.0	23.2	89.0	2.0	68.0	6.14	
3	1.0	2.0	28.9	104.0	2.0	84.0	16.15	
4	2.0	1.0	35.9	103.0	2.0	81.0	10.92	
...	...	...	...	...	...	...	...	
2273	2.0	2.0	33.5	100.0	2.0	73.0	6.53	
2274	1.0	2.0	30.0	93.0	2.0	208.0	13.02	
2275	1.0	2.0	23.7	103.0	2.0	124.0	21.41	
2276	2.0	2.0	27.4	90.0	2.0	108.0	4.99	
2277	1.0	2.0	24.5	108.0	2.0	108.0	3.76	

  

	age_group	Age	Insulin_Category
0	1	1	Medium
1	1	1	Low
2	1	1	Low
3	1	1	High

```

4          1    1          Medium
...      ...  ...          ...
2273       1    1          Low
2274       1    1          Medium
2275       1    1          High
2276       1    1          Low
2277       1    1          Low

```

[2278 rows x 10 columns]

#Using Z-Score to detect outlier and then remove them from columns Glucose, BMI and Oral Glucose  
from scipy import stats

import numpy as np

z = np.abs(stats.zscore(df['Glucose']))

print(z)

```

0          0.584085
1          0.590024
2          0.590024
3          0.248625
4          0.192715
...
2273       0.024985
2274       0.366384
2275       0.192715
2276       0.534114
2277       0.472265
Name: Glucose, Length: 2278, dtype: float64

```

threshold\_z = 2

outlier\_indices = np.where(z > threshold\_z)[0]

no\_outliers = df.drop(outlier\_indices)

print("Original DataFrame Shape:", df.shape)

print("DataFrame Shape after Removing Outliers:", no\_outliers.shape)

```

Original DataFrame Shape: (2278, 10)
DataFrame Shape after Removing Outliers: (2237, 10)

```

z = np.abs(stats.zscore(no\_outliers['BMI']))

print(z)

```

0          1.092822
1          1.050235
2          0.646672
3          0.146537
4          1.120654
...
2273       0.786671
2274       0.299613
2275       0.577092
2276       0.062202
2277       0.465765
Name: BMI, Length: 2237, dtype: float64

```

threshold\_z = 2

outlier\_indices = no\_outliers[z > threshold\_z].index

no\_outliers2 = no\_outliers.drop(outlier\_indices)

print("Original DataFrame Shape:", no\_outliers.shape)

print("DataFrame Shape after Removing Outliers:", no\_outliers2.shape)

```

Original DataFrame Shape: (2237, 10)
DataFrame Shape after Removing Outliers: (2132, 10)

```

z = np.abs(stats.zscore(no\_outliers2['Oral\_Glucose']))

print(z)

```

0          1.016329
1          0.825439
2          1.141170
3          0.720195
4          0.799128
...
2273       1.009616
2274       2.542365
2275       0.332244
2276       0.088732

```

```
2277 0.088732
Name: Oral_Glucose, Length: 2132, dtype: float64
```

```
threshold_z = 2
```

```
outlier_indices = no_outliers2[z > threshold_z].index
df = no_outliers2.drop(outlier_indices)
print("Original DataFrame Shape:", no_outliers2.shape)
print("DataFrame Shape after Removing Outliers:", df.shape)
```

```
Original DataFrame Shape: (2132, 10)
DataFrame Shape after Removing Outliers: (2032, 10)
```

```
print(df)
```

```
Gender  Exercise  BMI  Glucose  Diabetic  Oral_Glucose  Insulin \
0      2.0      2.0  35.7   110.0      2.0      150.0    14.91
1      2.0      2.0  20.3    89.0      2.0      80.0     3.85
2      1.0      2.0  23.2    89.0      2.0      68.0     6.14
3      1.0      2.0  28.9   104.0      2.0      84.0    16.15
4      2.0      1.0  35.9   103.0      2.0      81.0    10.92
...      ...      ...      ...      ...      ...      ...
2272    1.0      2.0  22.5    98.0      2.0      79.0     3.51
2273    2.0      2.0  33.5   100.0      2.0      73.0     6.53
2275    1.0      2.0  23.7   103.0      2.0     124.0    21.41
2276    2.0      2.0  27.4    90.0      2.0     108.0     4.99
2277    1.0      2.0  24.5   108.0      2.0     108.0     3.76
```

```
age_group  Age  Insulin_Category
0          1    1             Medium
1          1    1              Low
2          1    1              Low
3          1    1              High
4          1    1             Medium
...      ...  ...             ...
2272      1    1              Low
2273      1    1              Low
2275      1    1              High
2276      1    1              Low
2277      1    1              Low
```

```
[2032 rows x 10 columns]
```

```
from sklearn.model_selection import cross_val_score, KFold
from sklearn.svm import SVC
```

```
#Run the cross validation with 10 folds
X = df.drop(columns=['Insulin_Category', 'Age'])
Y = df['Insulin_Category']
```

```
svm_classifier = SVC(kernel='linear')
num_folds = 10
kf = KFold(n_splits=num_folds, shuffle=True, random_state=42)
```

```
cross_val_results = cross_val_score(svm_classifier, X, Y, cv=kf)
```

```
print("Cross-Validation Results (Accuracy):")
for i, result in enumerate(cross_val_results, 1):
    print(f"  Fold {i}: {result * 100:.2f}%")
```

```
print(f'Mean Accuracy: {cross_val_results.mean() * 100:.2f}%')
```

```
Cross-Validation Results (Accuracy):
Fold 1: 98.53%
Fold 2: 99.02%
Fold 3: 99.51%
Fold 4: 99.01%
Fold 5: 100.00%
Fold 6: 100.00%
Fold 7: 100.00%
Fold 8: 99.51%
Fold 9: 100.00%
Fold 10: 100.00%
Mean Accuracy: 99.56%
```

