# DAKD 2024 EXERCISE 2: SUPERVISED LEARNING

## Fill in your name, student id number and email address


### name: Arttu Kuitunen

### student id: 1500550

### email: arsaku@utu.fi


The previous exercise was about *data understanding* and *data preparation*, which formed the basis for the modeling phase of the data mining process. Many modeling techniques make assumptions about the data, so the exploration and preparation phases can't be ignored. Now, as we have checked the validity of data and familiarized ourselves with it, we can move on to the next stage of the Cross-Industry Standard Process for Data Mining (CRISP-DM), which is modeling.

The questions to be answered at this stage could include:

- What kind of model architecture best fits our data?
- How well does the model perform technically?
- Could we improve its performance?
- How do we evaluate the model's performance?

*Machine learning* is a subfield of artificial intelligence that provides automatic, objective and data-driven techniques for modeling the data. Its two main branches are *supervised learning* and *unsupervised learning*, and in this exercise, we are going to use the former, supervised learning, for classification and regression tasks.

For classification, data remains the same as in the previous exercise, but I've already cleaned it up for you. Some data pre-processing steps are still required to ensure that it's in an appropriate format so that models can learn from it. Even though we are not conducting any major data exploration nor data preparation this time, *you should never forget it in your future data analyses*.

---

## General Guidance for Exercises

- **Complete all tasks:** Make sure to answer all questions, even if you cannot get your script to fully work.

- **Code clarity:** Write clear and readable code. Include comments to explain what your code does.
- **Effective visualizations:** Ensure all plots have labeled axes, legends, and captions. Your visualizations should clearly represent the underlying data.
- **Notebook organization:** You can add more code or markdown cells to improve the structure of your notebook as long as it maintains a logical flow.
- **Submission:** Submit both the .ipynb and .html or .pdf versions of your notebook. Before finalizing your notebook, use the "Restart & Run All" feature to ensure it runs correctly.
- **Grading criteria:**
  - The grading scale is *Fail*/*Pass*/*Pass with honors* (+1).
  - To pass, you must complete the required parts 1-4.
  - To achieve Pass with honors, complete the bonus exercises.
- **Technical issues:**
  - If you encounter problems, start with an online search to find solutions but do not simply copy and paste code. Understand any code you use and integrate it appropriately.
  - Cite all external sources used, whether for code or explanations.
  - If problems persist, ask for help in the course discussion forum, at exercise sessions, or via email at tuhlei@utu.fi, aibekt@utu.fi.
- **Use of AI and large language models:**
  - We do not encourage the use of AI tools like ChatGPT. If you use them, critically evaluate their outputs.
  - Describe how you used the AI tools in your work, including your input and how the output was beneficial.
- **Time management:** Do not leave your work until the last moment. No feedback will be available during weekends.
- **Additional notes:**
  - You can find the specific deadlines and session times for each assignment on the Moodle course page.
  - Ensure all your answers are concise—typically a few sentences per question.
  - Your .ipynb notebook is expected to be run to completion, which means that it should execute without errors when all cells are run in sequence. are run in sequence.

The guided exercise session is held on the 27th of November at 14:15-16:00, at lecture hall X, Natura building.

The deadline is the 2nd of December at 23:59. Late submissions will not be accepted unless there is a valid excuse for an extension which should be asked **before** the original deadline.

---

## Packages needed for this exercise:

You can use other packages as well, but this excercise can be completed with those below.

In [12]:
```
import numpy as np
import pandas as pd

# Visualization packages - matplotlib and seaborn
# Remember that pandas is also handy and capable when it comes to plotting!
import seaborn as sns
import matplotlib.pyplot as plt

# Machine learning package - scikit-learn
from sklearn import metrics
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split, cross_val_score, LeaveOneOut
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import Ridge

# Show the plots inline in the notebook
%matplotlib inline
```

---

# 1. Classification using k-nearest neighbors

We start exploring the world of data modeling by using the K-Nearest Neightbors (k-NN) algorithm. The k-NN algorithm is a classic supervised machine learning technique based on the assumption that data points with similar features tend to belong to the same class, and thus are likely to be near each other in feature space.

In our case, we'll use the k-NN algorithm to *predict the presence of cardiovascular disease* (CVD) using all the other variables as features in the given data set. I.e. the target variable that we are interested in is `cardio`. Let's have a brief look at the features again:

| Feature | Type | Explanation | | :- | :- | :- | age | numeric | The age of the patient in years | sex | binary | Female == 0, Male == 1 | height | numeric | Measured weight of the patient (kg) | weight | numeric | Measured weight of the patient (cm) | ap_hi | numeric | Measured Systolic blood pressure | ap_lo | numeric | Measured Diastolic blood pressure | smoke | binary | A subjective feature based on asking the patient whether or not he/she smokes | alco | binary | A subjective feature based on asking the patient whether or not he/she consumes alcohol | active | binary | A subjective feature based on asking the patient whether or not he/she exercises regularly | cholesterol | categorical | Cholesterol associated risk information evaluated by a doctor | gluc | categorical | Glucose associated risk information evaluated by a doctor

But first, we need data for the task. The code for loading the data into the environment is provided for you. The code should work but make sure that you have the CSV file of the data in the same directory where you have this notebook file.

**Exercise 1 A)**

Take a random sample of 1000 rows from the dataframe using a fixed random seed. Print the first 15 rows to check that everything is ok with the dataframe.

*Note: As mentioned, the data remains the same, but cholesterol has been one-hot-encoded for you already. There's a new variable,* `gluc` *(about glucose aka blood sugar levels), which is also one-hot-encoded for you. It has similar values as* `cholesterol` .

In [13]:
```
### Loading code provided
# ------------------------------------------------------
# The data file should be at the same location than the
# exercise file to make sure the following lines work!
# Otherwise, fix the path.
# ------------------------------------------------------

# Path for the data
data_path = 'ex2_cardio_data.csv'

# Read the CSV file
cardio_data = pd.read_csv(data_path)
```

In [14]:
```
### Code - Resample and print 15 rows
random_state = 40
cardio_sample = cardio_data.sample(n=1000, random_state=random_state)

cardio_sample.head(15)
```

Out[14]:

| | age | sex | height | weight | ap_hi | ap_lo | smoke | alco | active | cardio | cholesterol_n |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **1885** | 40 | 0 | 165 | 65.0 | 120 | 80 | 0 | 0 | 1 | 0 | |
| **1538** | 54 | 0 | 162 | 64.0 | 140 | 80 | 0 | 0 | 1 | 0 | |
| **3072** | 50 | 1 | 162 | 54.0 | 90 | 60 | 0 | 0 | 1 | 0 | |
| **4961** | 57 | 0 | 164 | 67.0 | 155 | 90 | 0 | 0 | 1 | 1 | |
| **3174** | 64 | 0 | 155 | 84.0 | 160 | 100 | 0 | 0 | 0 | 0 | |
| **495** | 62 | 0 | 154 | 59.0 | 115 | 70 | 0 | 0 | 1 | 0 | |
| **2671** | 51 | 0 | 175 | 85.0 | 120 | 70 | 0 | 0 | 1 | 0 | |
| **5** | 53 | 0 | 152 | 56.0 | 103 | 65 | 0 | 0 | 1 | 0 | |
| **1124** | 59 | 1 | 175 | 81.0 | 120 | 70 | 1 | 1 | 1 | 0 | |
| **5531** | 52 | 1 | 165 | 82.9 | 180 | 90 | 1 | 0 | 1 | 1 | |
| **1984** | 42 | 1 | 168 | 70.0 | 110 | 70 | 1 | 0 | 1 | 0 | |
| **5148** | 59 | 1 | 172 | 76.0 | 140 | 90 | 0 | 0 | 1 | 1 | |
| **3172** | 57 | 1 | 178 | 66.0 | 120 | 80 | 1 | 1 | 1 | 0 | |
| **5488** | 58 | 0 | 170 | 70.0 | 120 | 80 | 0 | 0 | 1 | 1 | |
| **1612** | 40 | 1 | 165 | 64.0 | 120 | 80 | 0 | 0 | 1 | 0 | |

---

We have the data so now, let's put it to use. All the analyses will be done based on this sample of 1000.

To teach the k-NN algorithm (or any other machine learning algorithm) to recognize patterns, we need training data. However, to assess how well a model has learned these patterns, we require test data which is new and unseen by the trained model. It's important to note that the test set is not revealed to the model until after the training is complete.

So, to *estimate the performance of a model*, we may use a basic train-test split. The term "split" is there because we literally split the data into two sets.

Before the exercise itself, we might as well discuss about the reproducibility of experiments we conduct in research. It can be quite a nightmare for some if code spewed out only random results. To address this, we can set a **random seed** to ensure that any random processes, such as splitting our dataset into training and test sets, yield consistent results across multiple runs. By using a fixed random seed, we enhance the reproducibility of our experiments, making it easier to validate findings. In fact, we already used one when sampling our subset from the loaded dataset.

**Exercise 1 B)**

Gather the features into one array and the target variable into another array. Create training and test data by splitting the data into training (80%) and test (20%) sets. Use a fixed random seed to ensure that even if you execute this cell hundreds of times, you will get the same split each time.

- Do you need stratification for our dataset? Explain your decision.

In [15]:
```
cardio_sample['cardio']
cardio_sample.describe()
```

Out[15]:

| | age | sex | height | weight | ap_hi | ap_lo | |
|---|---|---|---|---|---|---|---|
| **count** | 1000.000000 | 1000.000000 | 1000.00000 | 1000.00000 | 1000.000000 | 1000.000000 | 1000. |
| **mean** | 52.055000 | 0.355000 | 164.53800 | 72.34330 | 123.023000 | 87.226000 | 0. |
| **std** | 6.777604 | 0.478753 | 8.50711 | 13.52471 | 16.659772 | 83.746537 | 0. |
| **min** | 29.000000 | 0.000000 | 68.00000 | 33.00000 | 12.000000 | 20.000000 | 0. |
| **25%** | 47.000000 | 0.000000 | 159.00000 | 63.00000 | 115.000000 | 80.000000 | 0. |
| **50%** | 53.000000 | 0.000000 | 165.00000 | 70.00000 | 120.000000 | 80.000000 | 0. |
| **75%** | 57.000000 | 1.000000 | 170.00000 | 80.00000 | 130.000000 | 80.000000 | 0. |
| **max** | 64.000000 | 1.000000 | 190.00000 | 165.00000 | 200.000000 | 1100.000000 | 1. |

In [16]:
```
### Code - Train-test split
def prepare_cardio_data(cardio_sample):

    # Features
    X = cardio_sample.drop('cardio', axis=1)
    # Target variable
    y = cardio_sample['cardio']

    X_train, X_test, y_train, y_test = train_test_split(
        X, y,
        test_size=0.2,
        random_state=40,
        stratify=y
    )

    return X_train, X_test, y_train, y_test

X_train, X_test, y_train, y_test = prepare_cardio_data(cardio_sample)

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

```
(      age  sex  height  weight  ap_hi  ap_lo  smoke  alco  active  \
638     55    1     176    74.0    100     60      0     0       1
3325    40    1     164    78.0    120     80      1     0       1
3439    55    0     161    61.0    130     90      0     0       1
3130    60    0     155    99.0    120     80      0     0       1
4366    58    0     168    74.0    120     80      0     0       0
...    ...  ...     ...     ...    ...    ...    ...   ...     ...
1757    52    1     167    79.0    120     80      1     1       1
1031    47    0     162    92.0    110     90      0     0       0
282     46    1     181    74.0    100     60      0     0       1
3948    54    1     160    52.0    120     80      0     0       0
2818    60    0     157    70.0    150     80      0     0       1

      cholesterol_normal  cholesterol_at_risk  cholesterol_elevated  \
638                    1                    0                     0
3325                   1                    0                     0
3439                   1                    0                     0
3130                   1                    0                     0
4366                   1                    0                     0
...                  ...                  ...                   ...
1757                   1                    0                     0
1031                   1                    0                     0
282                    1                    0                     0
3948                   1                    0                     0
2818                   1                    0                     0

      gluc_normal  gluc_at_risk  gluc_elevated
638             1             0              0
3325            1             0              0
3439            1             0              0
3130            1             0              0
4366            1             0              0
...           ...           ...            ...
1757            1             0              0
1031            1             0              0
282             1             0              0
3948            1             0              0
2818            0             0              1

[800 rows x 15 columns],       age  sex  height  weight  ap_hi  ap_lo  smoke  alco
active  \
1065    44    1     170    56.0    120     80      0     0       1
2929    55    1     160    65.0    120     80      0     0       1
3076    49    0     158    62.0    130     95      0     0       1
4446    53    0     152    59.0    110     70      0     0       1
531     52    0     160    80.0    120     80      0     0       1
...    ...  ...     ...     ...    ...    ...    ...   ...     ...
5355    54    0     161    71.0    140     90      0     0       1
4425    56    0     158    69.0    130     90      0     0       1
2840    49    0     162    68.0    130     80      0     0       1
4920    63    0     151    64.0    140     70      0     0       1
4165    50    0     154    58.0    110     80      0     0       0

      cholesterol_normal  cholesterol_at_risk  cholesterol_elevated  \
1065                   1                    0                     0
2929                   1                    0                     0
```

```
3076                  1                     0                     0
4446                  1                     0                     0
531                   0                     0                     1
...                 ...                   ...                   ...
5355                  1                     0                     0
4425                  1                     0                     0
2840                  1                     0                     0
4920                  1                     0                     0
4165                  0                     1                     0

      gluc_normal  gluc_at_risk  gluc_elevated
1065            1             0              0
2929            1             0              0
3076            1             0              0
4446            1             0              0
531             0             0              1
...           ...           ...            ...
5355            1             0              0
4425            1             0              0
2840            1             0              0
4920            1             0              0
4165            1             0              0

[200 rows x 15 columns], 638     0
3325    0
3439    0
3130    0
4366    1
       ..
1757    0
1031    0
282     0
3948    0
2818    0
Name: cardio, Length: 800, dtype: int64, 1065    0
2929    0
3076    0
4446    1
531     0
       ..
5355    1
4425    1
2840    0
4920    1
4165    0
Name: cardio, Length: 200, dtype: int64)
```

<Yes, straification is a good idea here. Task is to do a binary classification of no CVD or yes CVD, and samples with imbalanced proportions can skew the results. Especially now when data is not huge.>

---

**Exercise 1 C)**

Standardize the numerical features in both the train and test sets.

- Explain how the k-NN model makes predictions about whether or not a patient has cardiovascular disease (CVD) when the features are not standardized. Specifically, discuss how the varying scales of different features can influence the model's predictions, and how standardization would change this influence.

*Note: Some good information about preprocessing and how to use it for train and test data can be found https://scikit-learn.org/stable/modules/preprocessing.html.*

In [17]:
```
### Code - Standardization

# Initialize the scaler
scaler = StandardScaler()

# Fit on training data and transform both training and test
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Convert back to DataFrames to maintain column names
X_train_scaled = pd.DataFrame(X_train_scaled, columns=X_train.columns, index=X_tra
X_test_scaled = pd.DataFrame(X_test_scaled, columns=X_test.columns, index=X_test.i
```

<Unstandardized data is biased towards those variables with large values or larger scale, since the neighbouring is measured with distances. There fore same proportional difference would be larger in numerical values for those variables with larger scale. Normalization shifts all mean values to zero and standard deviation to 1. Then the difference in distance is proportional and gives similiar weight to different variables regardless of their raw values and scale differences. >

---

It's time for us to train the model!

**Exercise 1 D)**

Train a k-NN model with $k = 3$. Print out the confusion matrix and use it to compute the accuracy, the precision and the recall.

- What does each cell in the confusion matrix represents in the context of our dataset?
- How does the model perform with the different classes? Where do you think the differences come from? Interpret the performance metrics you just computed.
- With our dataset, why should you be a little more cautious when interpreting the accuracy?

*Note: We are very aware that there are functions available for these metrics, but this time, please calculate them using the confusion matrix.*

In [18]:
```
### Code - the kNN classifier
# I got some help from tutorial: https://www.youtube.com/watch?v=xEZUIfpCkRs
# Train k-NN model
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train_scaled, y_train)

# Make predictions
y_pred = knn.predict(X_test_scaled)

# Generate confusion matrix
conf_matrix = metrics.confusion_matrix(y_test, y_pred)

# Get values from confusion matrix
tn, fp, fn, tp = conf_matrix.ravel()

conf_df = pd.DataFrame(conf_matrix,
                       columns=['Predicted No CVD', 'Predicted CVD'],
                       index=['Actual No CVD', 'Actual CVD'])

# Calculating mertics
# tp = true positive,
# tn = true negative,
# fp = false positive,
# fn = false negative
accuracy = (tp + tn) / (tp + tn + fp + fn)
tp_precision = tp / (tp + fp)
tp_recall = tp / (tp + fn)

tn_precision = tn / (tn + fn)
tn_recall = tn / (tn + fp)

f_score_tp = 2 * tp_precision * tp_recall / (tp_precision + tp_recall)
f_score_tn = 2 * tn_precision * tn_recall / (tn_precision + tn_recall)

no_cvd_ratio = (tn + fp) / (tn + fn + fp + tp)

# Print results for both classes
print("Confusion Matrix:")
print(conf_matrix)
print("\nMetrics:")
print(f"Accuracy: {accuracy.round(4)}")
print(f"TP Precision: {tp_precision.round(4)}")
print(f"TP Recall: {tp_recall.round(4)}")
print(f"TN Precision: {tn_precision.round(4)}")
print(f"TN Recall: {tn_recall.round(4)}")
print(f"F-Score: {f_score_tp.round(4)}")
print(f"F-Score: {f_score_tn.round(4)}")
print(f"No CVD Ratio: {no_cvd_ratio.round(4)}")
conf_df
```

```
Confusion Matrix:
[[109  34]
 [ 29  28]]

Metrics:
Accuracy: 0.6850
TP Precision: 0.4516
TP Recall: 0.4912
TN Precision: 0.7899
TN Recall: 0.7622
F-Score: 0.4706
F-Score: 0.7758
No CVD Ratio: 0.7150
```

Out[18]:

| | Predicted No CVD | Predicted CVD |
|---|---|---|
| **Actual No CVD** | 109 | 34 |
| **Actual CVD** | 29 | 28 |

< Precision measures should be calculated for both classes. Since there is much more TN values than TP values (naturally more people are not having the disease) it lowers the overall score of the true positives. In this case we have more actual no CVD patients and I calculated the ratio to ~72%, which means if the model predicted everyone to not have CVD, the model would have a precision of ~72% when viewed with TN perspective.

Matrix values are:

```
TP = is predicted to have and has actual CVD,

TN = not predicted to have and does not have CVD,

FP = is predicted to have but does not have CVD,

FN = not predicted to have but actually has CVD,
```

>

---

## 2. Classification accuracy using leave-one-out cross-validation

While the train-test split may provide us with an unbiased estimate of the performance, we only evaluate the model once. Especially when dealing with small datasets, a test set itself will be very small. How can we be sure that the evaluation is accurate with this small test set and not just a good (or bad) luck? And what if we'd like to compare two models and the other seems to be better -- how can we be sure that it's not just a coincidence?

Well, there's a great help available and it's called cross-validation. With its help, we can split the dataset into multiple different training and test sets, which allows us to evaluate models across various data partitions. This time, we'll take a closer look at the leave-one-out cross-validation.

**Exercise 2**

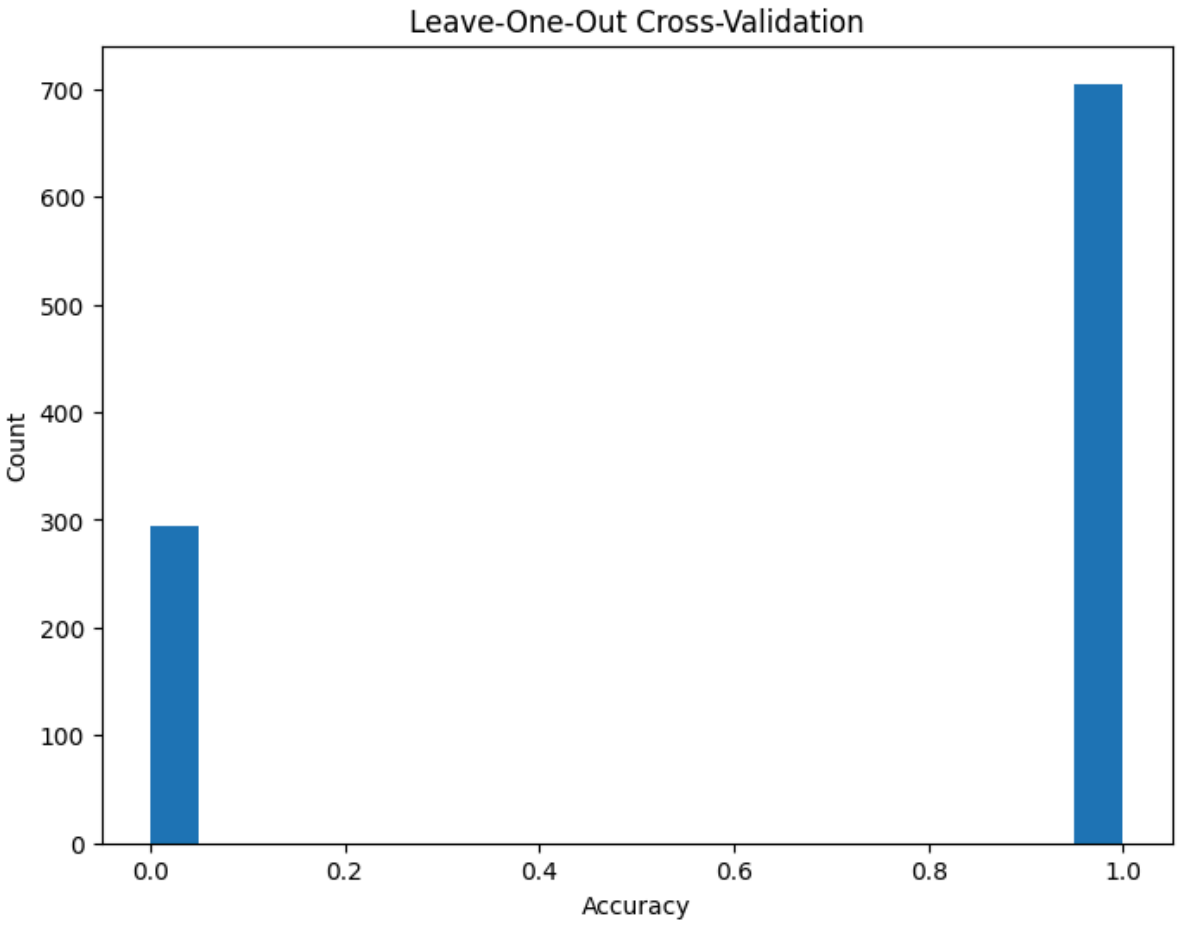
Let's keep the focus on detecting the CVD, so once again we utilize the k-NN model (with $k = 3$) to predict the precense of the disease. Now, apply leave-one-out cross-validation to assess whether the k-NN model is suitable for addressing the problem. You may use the entire sample of 1000 on this task.

- What can you say about the accuracy compared to the previous task?
- What do you think: Does the k-NN model work for the problem in hand? Explain your answer.

*Tip: This can certainly be done manually, but* `cross_val_score` *is also a very handy function.*

In [19]:
```
### Code - Leave-one-out cross-validation

# Using the full data set.
scaler = StandardScaler()
X = cardio_sample.drop('cardio', axis=1)
y = cardio_sample['cardio']
X_scaled = scaler.fit_transform(X)

# Create k-NN model like in previous task
knn = KNeighborsClassifier(n_neighbors=3)

# Perform leave-one-out
loo = LeaveOneOut()
scores = cross_val_score(knn, X_scaled, y, cv=loo)

# Print results
print("Leave-One-Out Cross-Validation Results:")
print(f"Accuracy: {scores.mean():.3f}")


# Figure to see counts.
plt.figure(figsize=(8, 6))
plt.hist(scores, bins=20)
plt.title('Leave-One-Out Cross-Validation')
plt.xlabel('Accuracy')
plt.ylabel('Count')
plt.show()
```

```
Leave-One-Out Cross-Validation Results:
Accuracy: 0.705
```




<Accuracy is notably better than in the previous model. Reason obviously is the leave one out methdos where predictions are made iteratiely for only one value (the one left out). Yet we are only grasping the accuracy of predicting all as not CVD. With these results I would say we might find something better. Looking at the next task maybe we will find better model with changing the numbe of neighbors. >

---

## 3. Model selection with leave-one-out cross-validation

So far, we've trained one model at a time and I've given the value of k for you. Accuracy is what it is (no spoilers here), but could we still do a little better? Let's explore that possibility through a process known as hyperparameter tuning. The cross-validation is especially important tool for this task. Note here, that model selection and model evaluation (or assessment) are two different things: We use model selection to estimate the performance of various models to identify the model which is most likely to provide the "best" predictive performance for the task. And when we have found this most suitable model, we *assess* its perfomance and generalisation power on unseen data.

This time, we're going to train multiple models, let's say 30, and our goal is to select the best K-Nearest Neighbors model from this set. Most models come with various hyperparameters that require careful selection, and the k-NN model is no exception. Although we're talking about the number of neighbors here, it's important to note that k-NN also has several other hyperparameters, such as the used distance measure. However, for the sake of simplicity, this time we'll focus solely on fine-tuning the number of nearest neighbors, that is, the value of k, and use default values for all the other hyperparameters.

Let's focus on the model selection part here for the sake of comprehending the cross-validation itself. We'll get later on the whole pipeline, which also includes model assessment.

**Exercise 3**

Find the optimal k value from a set of $k = 1...30$ using leave-one-out cross-validation. Plot the accuracies vs. the k values. Again, you may use the entire sample of 1000 on this task.

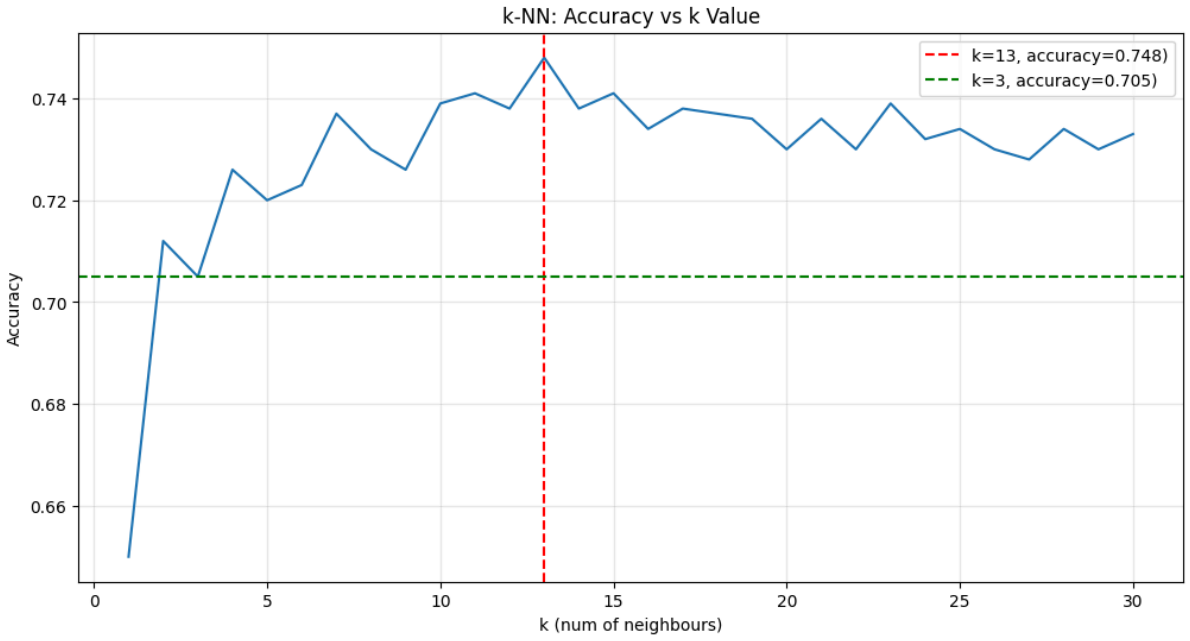
- Which value of k produces the best accuracy when using leave-one-out cross-validation? Compare the result to the previous model with $k = 3$.
- If the number of k is still increased, what is the limit that the accuracy approaches? Why?
- Discuss the impact of choosing a very small or very large number of neighbors on the k-NN model's ability to distinguish between the healthy individuals and the ones with CVD.

In [20]:
```
# I used datacamp as a reference for iterating the k -values:
# https://www.datacamp.com/tutorial/k-nearest-neighbor-classification-scikit-learr

### Code - Select best model

k_values = range(1, 31)
k_scores = []
loo = LeaveOneOut()

# Leave-one-out for each k
for k in k_values:
    knn = KNeighborsClassifier(n_neighbors=k)
    scores = cross_val_score(knn, X_scaled, y, cv=loo)
    k_scores.append(scores.mean())

# Find the best k
df_k_scores = pd.Series(k_scores)
best_k = df_k_scores.idxmax() + 1

best_k
```

Out[20]: 13

In [21]:
```
### Code - Plot the accuracies vs. the values for k
```

```
# Plot results
plt.figure(figsize=(12, 6))
plt.plot(k_values, k_scores)
plt.grid(True, alpha=0.3)
plt.title('k-NN: Accuracy vs k Value')
plt.xlabel('k (num of neighbours)')
plt.ylabel('Accuracy')

# Add best k value
plt.axvline(x=best_k, color='r', linestyle='--',
            label=f'k={best_k}, accuracy={max(k_scores).round(3)})')

# Add k=3 reference
k3_score = k_scores[2]
plt.axhline(y=k3_score, color='g', linestyle='--',
            label=f'k=3, accuracy={k3_score.round(3)})')

plt.legend()
plt.show()

# Print best result
print(f"Best k value: {best_k}")
print(f"Best accuracy: {max(k_scores).round(4)}")
print(f"k=3 accuracy: {k3_score.round(4)}")
```




```
Best k value: 13
Best accuracy: 0.748
k=3 accuracy: 0.705
```

<

K = 13 is the optimal value. Accuracy is 0.748 vs 0.705 for the k=3. I assume this is because the proportion of neighbours vs sample size rises relatively high. This results to values closer to the populations mean. In other words, the model loses local accuracy.

Small k-values are more prown to changes very close to it. So if there are two individuals with similiar characteristics and the other one is healthy and other is sick, the model would probably end up predicting the at least the other wrong, since they would be predicted same way.

Large k-values would generalize gourps better, and be less effected by odd values, that would have a big local impact otherwise. If the k-value rises too much, the model would be more and more a reflection of the proportions of the classes but lose accuracy.

>

---

# 4. Ridge regression

The previous exercises were about classification. Now, we are ready to see another kind of supervised learning - regression - as we are changing our main goal from predicting discrete classes (healthy/sick) to estimating continuous values. The following exercises are going to involve utilizing one regression model, Ridge Regression, and our goal is to evaluate the performance of this model.

Let's change the dataset to make the following exercises more intuitive. The new dataset is about brushtail possums and it includes variables such as

| Feature | Type | Explanation | | :- | :- | :- |sex|binary| Sex, either male (0) or female (1) |age|numeric| Age in years |len_head|numeric| Head length in mm |width_skull|numeric| Skull width in mm |len_earconch|numeric| Ear conch length in mm |width_eye|numeric| Distance from medial canthus to lateral canthusof right eye, i.e., eye width in mm |len_foot|numeric| Foot length in mm |len_tail|numeric| Tail length in mm |chest|numeric| Chest grit in mm |belly |numeric| Belly grit in mm |len_total|numeric| Total length in mm

In this case, our target variable will be *the age of the possum*. The data for this exercise has been modified from the original source.

There's the code chunk for loading data provided again. Again, the data file should be located in the same directory as this notebook file!

In [22]:
```
### Loading code provided
# ------------------------------------------------------
# The data file should be at the same location than the
# exercise file to make sure the following lines work!
# Otherwise, fix the path.
# ------------------------------------------------------

# Data path
data_path = 'ex2_possum_data.csv'
```

```
# Load the data
possum_data = pd.read_csv(data_path)
```

---

Regression allows us to examine relationships between two or more variables. This relationship is represented by an *equation*, which itself represents how a change in one variable affects another on average. For example, we could examine how a change in possum's total length affects, on average, its estimated age.

We start by examining those relationships between the variables in the given dataset.

**Exercise 4 A)**

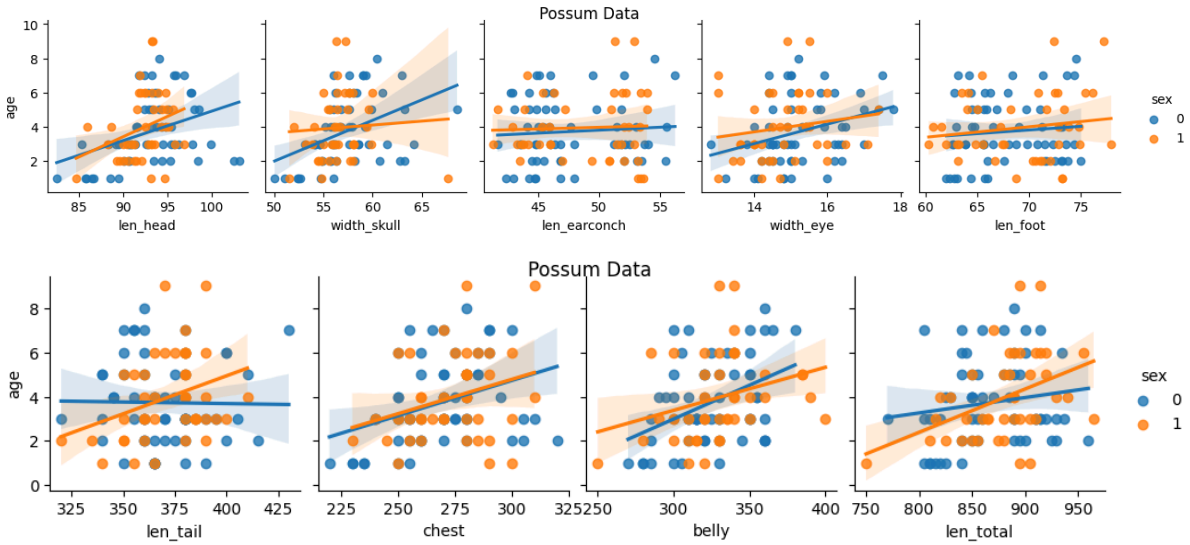
Plot pairwise relationships between the age variable and the others where you color the samples based on the sex variable.

- Which body dimensions seem to be most correlated with age? And are there any variables that seem to have no correlation with it?
- Are there any differences in the correlations between males and females?

*Tip:* `seaborn.pairplot()` *is handy with the parameters* `(x,y)_vars` *and* `hue`*. You actually can fit a linear model to draw a regression line with the parameter* `kind` *set to* `"reg"`.

In [23]:
```
### Code - Pairplot

# Two plots for better readability
figure1_columns = ['len_head', 'width_skull', 'len_earconch', 'width_eye', 'len_fo
figure2_columns = [ 'len_tail', 'chest', 'belly', 'len_total']

# First row of plots
plt_1 = sns.pairplot(possum_data,
           x_vars=figure1_columns,
           y_vars=['age'],
           hue='sex',
           kind='reg')

# Second row of plots
plt_2 = sns.pairplot(possum_data,
           x_vars=figure2_columns,
           y_vars=['age'],
           hue='sex',
           kind='reg')

plt_1.fig.suptitle('Possum Data', y=1.02)
plt_2.fig.suptitle('Possum Data', y=1.02)
```

Out[23]:
```
Text(0.5, 1.02, 'Possum Data')
```




< Head length seems to have a correlation with age but no significant difference between sex.

Skull width has difference with sex, blue (sex = 0) has wider skull in general.

Tail length seems to be correlated with age in sex group 1.

Chest size is larger with older possums for both sexes.

Belly and total lenght also grows with age. Total length seems to be higher with sex=1, probablay due to tail length.

Seems like overall possums are similiar size regardless of sex, but with some differing characteristics i.e tail, skull, headlength.

Foot length, Earconch length, and tail lenght (for sex=0) has no correlation to age.

>

---

Before the regression analysis itself, let's check that our dataset is in a proper format. We'll also perform the train-test split as we're going to test the overall performance of the model using the test set.

**Exercise 4 B)**

Do you need to prepare the data a little? Explain your decision. Perform the train-test (80/20) split.

*Note: Set the features in the dataframe named as* `possum_X` *so you can play around with the upcoming code snippet.*

In [24]:
```
### Code - Data preparation

# Features and target
possum_X = possum_data.drop('age', axis=1)
y = possum_data['age']
columns = possum_X.columns

# Scale numerical features
scaler = StandardScaler()
possum_X = scaler.fit_transform(possum_X)

# Perform train-test split
X_train, X_test, y_train, y_test = train_test_split(
    possum_X, y, test_size=0.2, random_state=40
)

print("Training set shape:", X_train.shape)
print("Test set shape:", X_test.shape)
```

```
Training set shape: (80, 10)
Test set shape: (21, 10)
```

<Taking examples from earlier preparations. We need to scale the body measurements, since their scale and sizes are different. >

---

Regarding Ridge Regression, we'll focus on the hyperparameter called $\lambda$ (read as 'lambda'), the regularization term (or penalty term or L2 penalty, how ever we'd like to call it).

**Exercise 4 C)**

Fit a ridge regression model with the whole training set. For the hyperparameter 'lambda', use 64. Evaluate the model using the test set and describe the results. For evaluating on the test set, use a metric called mean absolute error (MAE).

- How well did the model perform in estimating the possums' ages?
- How do you interpret the MAE in our case when the target variable is age?

In [25]:
```
## Code - Ridge regression
# Note: sklearn uses alpha instead of lambda as the parameter name
ridge_model = Ridge(alpha=64)
ridge_model.fit(X_train, y_train)

# Make predictions on test set
y_pred = ridge_model.predict(X_test)

# Calculate MAE
mae = metrics.mean_absolute_error(y_test, y_pred)
print(f"Mean Absolute Error: {mae.round(3)} years")
```

```
Mean Absolute Error: 1.008 years
```

<Mean absolute error gives the a value that the prediction is off on average. In this case it seems that, with the given parameters the model can predict the age quite accurately.>

Now that we have fitted the regression model, let's break it down for better understanding what is actually happening here. Remember that the model here is essentially just a linear regression model with an added regularization term to deal with e.g overfitting and multicollinearity. We can write the equation used by the model to predict an opossum's age as:

$$\text{Predicted age} = w_1 \times \text{Sex} + w_2 \times \text{Head length} + w_3 \times \text{Skull width} + \ldots + w_{10} \times \text{Total l}$$

As mentioned earlier, regression focuses on the relationships between the features and the target variable. In the equation above, each feature contibutes a certain amount to the predicted age, based on the weight $w_i$ learned for that feature. For example, if the total length of an opossum has a large positive weight, it suggests that opossums with greater length are predicted to be older. On the other hand, if the skull width of an opossum has a negative weight, it indicates that opossums with wider skulls are predicted to be younger. In this case, as skull width increases, the predicted age decreases.

Different classes have different class attributes that you can access after e.g. fitting a model, and the `Ridge` class is no exception: For example, the `coef_` variable contains the learned weights $w_1, \ldots, w_{10}$ that represent the relationship between the features and the target (a.k.a age) variable. The `intercept_` variable holds the bias term (or the intercept, however we wanna call it).

We can now write down the equation used by our fitted model. You can experiment with it by adjusting the regularization term or using a different sample, if you'd like, to see how the weights and bias change. This is just extra!

In [26]:
```
# NOTE: To make this code chunk to work with the already fitted model,
#       the model variable needs to be named as `ridge_model`. Also, the
#       initial feature dataframe is named here as `possum_X`.


coefficients = ridge_model.coef_ # CHANGE THE VARIABLE NAME IF NOT WROTE AS THIS
bias = ridge_model.intercept_ # # CHANGE THE VARIABLE NAME IF NOT WROTE AS THIS
feature_names = columns # CHANGE THE VARIABLE NAME HERE IF NOT AS WROTE AS THIS

# Let's write the equation
equation = 'Predicted age = '
for i in range(len(coefficients)):
    equation += f'{coefficients[i]:.3f}*{feature_names[i]} + '

equation += f'{bias:.3f}'
print(equation)
```

```
Predicted age = 0.064*sex + 0.102*len_head + 0.071*width_skull + 0.066*len_earconc
h + 0.188*width_eye + -0.056*len_foot + 0.010*len_tail + 0.156*chest + 0.208*belly
+ 0.023*len_total + 3.952
```

---

## BONUS: Feature selection - most useful features in predicting cardiovascular diseases

You can stop here and get the "pass" grade! To get the pass with honors, you need to do the following exercise. This means you'll get one bonus point for the exam.

The exercise may require you to do some research of your own. You are also required to **explain** the steps you choose with your own words, and show that you tried to understand the idea behind the task. There's no single correct solution for this so just explain what you did and especially ***why*** you did it. Please note that submitting only code will not be awarded a pass with honors.

---

Due to the lack of resources and time, doctors can't measure all the values represented in the given cardio dataset. Fortunately, eager students are willing to help: Your task is to identify five [5] most useful features for predicting the presence of the CVD from the dataset. The steps needed for this job are presented above except the feature selection part. You must remember not to leak any information from the test set when selecting the features, i.e., you try to find those five features using only the training set.

Regarding the feature selection itself, you're asked to use Random Forest. To do this, use the Random Forest classifier's built-in feature importance estimation in scikit-learn. Explain briefly the working of the model on the given cardio dataset: How does the model select features that are relevant in predicting CVD?

Evaluate the model of your choice using accuracy and the area under the ROC curve (AUC). Draw the corresponding curve in a plot. **Discuss** your findings and results.

What goes wrong in your AUC analysis, if you use the predictions from the `predict()` function instead of the `predict_proba()` function to calculate the AUC?

In [27]:
```
## Code - Bonus task
```

<Write your answer here>