Exercise 1: The Shapley value

First, we implement the original Shapley value for a cooperative game. In the AI quiz we have five players: Timnit, Margret, Samy, Jeff and Larry. The overall payoff function for a set of players S is given as

$$\text{payoff}(S) = 10t + 10m + 10s + 2j + 20(t \wedge m) + 20(t \wedge m \wedge s) - 30((t \vee m \vee s) \wedge j)$$

where t, m, s, j, l indicate whether Timnit, Margret, Samy, Jeff and Larry are in the set S. The function `payoff(set)` implements this functionality, where a set is a list/set of the respective characters (`'t', 's', 'm', 'j', 'l'`).

(a) Calculate the payoff of S = {`t,m`} (result should be 40) and S = {`t,j,s`} (result should be -8).

(b) Implement the original, exact Shapley value algorithm.

- Start by implementing the function `payoff(set)` assigning a Boolean value for each player. It should return the overall performance as given in the task.
- Define a function `all_unique_subsets()` returning all subsets of a set.
- Finally, implement `shapley()`.

(c) Implement the permutation based approximation of the Shapley value with a fixed number of iterations (`shapley_perm()`).

(d) In this subsection, we want to identify whether the Shapley values or specific sets fulfill the axioms. Please write the functions `symmetry_check`, `dummy_check`, `additivity_check`, `efficiency_check` to check the respective properties, all of which return a Boolean value. We briefly recall the properties below:

(i) *Symmetry:* Assess whether two features with the same contributions have the same Shapley value. I.e., if for $S \subseteq P \setminus \{j, k\}$

$$v(S \cup \{j\}) - v(S) = v(S \cup \{k\}) - v(S).$$

then the Shapley values should be identical.

(ii) *Dummy:* Given player j for whom for all subsets $S \subseteq P$ no contribution is made ($v(S) = v(S \cup \{j\})$) then also the Shapley value must be zero.

(iii) *Additivity:* If v is the sum of two games v_1 and v_2 (i.e. $v(S) = v_1(S) + v_2(S)$) then the payout is the sum of payouts: $\phi_{j,v} = \phi_{j,v_1} + \phi_{j,v_2}$.

(iv) *Efficiency:* Player contributions add up to the total payout of the game:

$$\sum_{j=1}^{p} \phi_j = v(P)$$

Exercise 2: SHAP

Now we apply Shapley as a local feature relevance quantification tool. Therefore we implement the marginal SHAP payoff function.

(a) Load the FIFA dataset into a data table (e.g. using the pandas library in python) and predict the Man of the Match probability through a random forest.
Hint: Note that many of the float variables in the data set contain missing values and hence, consider integer variables only. Transform the target variable 'Man of the Match' into a binary format suitable for a prediction task and select only explanatory variables of type integer (e.g. dtype int64). Don't forget to split the data into a training and test set before fitting the random forest! Use an instance from your test set to generate an exemplary prediction for the probability of a team having the 'Man of the Match' amongst them.

(b) Implement the marginal sampling based SHAP value function `m_vfunc()`. Compute the marginal sampling based value functions $v(j)$ for the instance.

(c) We could use the value function implemented above in combination with our Shapley implementations from Exercise 1 to compute SHAP. A more efficient equivalent definition has been proposed, which is based on locally fitting a weighted linear model: KernelSHAP. Implement KernelSHAP and calculate the SHAP values for the instance. Do this by writing the following functions:

- `shap_weights(mask)`: A function to return the weights for a given `mask` containing the sampled coalitions ((binary) coalition feature space).
- `replace_dataset(obs, X, nr_samples)`: A function to create the dataset of `nr_samples` samples generated during the SHAP calculations and mapping the binary feature space back to the original feature space. The positions in the mask indicate where we replace the original values with the ones of the given observation `obs`.
- `shap_data(obs, X, nr_samples, predict)`: A function summarizing `mask, pred, weight`.
- `kernel_shap(obs, X, nr_samples, predict)`: Implement the sampling based KernelSHAP function.

(d) Calculate `kernel_shap(X_test[1, ], X_train, 1000, classifier_RF)` using the random forest from (a).