Solution 1:

a)

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

$$\text{payoff}(\{\texttt{t},\texttt{m}\}) = 10 + 10 + 20 = 40$$

$$\text{payoff}(\{\texttt{t},\texttt{j},\texttt{s}\}) = 10 + 10 + 2 - 30 = -8$$

b) Pseudocode of `payoff_func()`

Algorithm 1 `payoff_func()`

```
Require: coalition: Coalition vector
t ← boolean if 't' is in coalition
s ← boolean if 's' is in coalition
m ← boolean if 'm' is in coalition
j ← boolean if 'j' is in coalition
l ← boolean if 'l' is in coalition
return 10 * t + 10 * m + 2 * j + 20 * (t and m) + 20 * (t and m and s) - 30 * ((t or m or s) and j)
```

Pseudocode of `all_unique_subsets()`

Algorithm 2 `all_unique_subsets()`

```
Require: population: vector containing all available players
if population = ∅ then subsets ← ∅
else if population ≠ ∅ then subsets ← all subsets of population
end if
return subsets
```

Pseudocode of `shapley()`

Algorithm 3 `shapley()`

```
Require: population: vector containing all available players
Require: member: individual player, i.e. feature of interest
Require: vfunc: value function
remainder ← everyone from the population but member
all_sets ← all_unique_subsets(remainder)
F ← length of population
for s in all_sets do
    S ← length s
    diff ← vfunc(s + member) - vfunc(s)
    factor ← S! * (F - S - 1)! / F!
    val ← val + factor * diff
end for
return val
```

c) Pseudocode of `shapley_perm()`

Algorithm 4 `shapley_perm()`

```
Require: member: individual player, i.e. feature of interest
Require: population: vector containing all available players
Require: vfunc: value function
Require: its: number of iterations
for i in its do
    perm ← permutation of population
    member_ix ← index of member in population
    s ← coalition of perm until member_ix
    diff ← difference of vfunc of s with member minus vfunc of s
    vals[i] ← diff
end for
val ← sum of vals divided by length of vals
return val
```

d) (i) Pseudocode of `symmetry_check()`

Algorithm 5 `symmetry_check()`

```
Require: j: first feature index
Require: k: second feature index
Require: population: vector containing all available players
Require: vfunc: value function
Require: shapley_func: Shapley function
remainder ← everyone from the population but j, k
all_S ← all_unique_subsets(remainder)
for S in all_S do
    surplus_j ← difference of vfunc of S with j minus vfunc of S
    surplus_k ← difference of vfunc of S with k minus vfunc of S
    save surplus_j and surplus_k in vectors surpluss_j and surpluss_k, respectively, for every iteration
end for
if surpluss_j equal surpluss_k then
    print equal surplus
    val_j ← shapley_func(j, population, vfunc)
    val_k ← shapley_func(k, population, vfunc)
    return val_j == val_k
end if
return TRUE
```

(ii) Pseudocode of `dummy_check()`

Algorithm 6 `dummy_check()`

```
Require: j: feature index
Require: population: vector containing all available players
Require: vfunc: value function
Require: shapley_func: Shapley function
remainder ← everyone from the population but j
all_S ← all_unique_subsets(remainder)
for S in all_S do
    surplus_j ← difference of vfunc of S with j minus vfunc of S
    save surplus_j in vector surpluss_j for every iteration
end for
if sum of |surpluss_j| > 0 then
    print has contribution
    val_j ← shapley_func(j, population, vfunc)
    return val_j > 0
end if
return TRUE
```

(iii) Pseudocode of `additivity_check()`

Algorithm 7 `additivity_check()`

Require: `population`: vector containing all available players
Require: `vfunc1`: value function 1
Require: `vfunc2`: value function 2
Require: `shapley_func`: Shapley function

`combined` ← addition of `vfunc1` and `vfunc2`
`vals1` ← Shapley values for all features using `vfunc1`
`vals2` ← Shapley values for all features using `vfunc2`
`vals_comb` ← Shapley values for all features using `combined`
`vals_additive` ← `vals1` + `vals2`
return `vals_comb` == `vals_additive`

(iv) Pseudocode of `efficiency_check()`

Algorithm 8 `efficiency_check()`

Require: `population`: vector containing all available players
Require: `vfunc`: value function
Require: `shapley_func`: Shapley function

`payoff_total` ← `vfunc` of `population`
`shapley_vals` ← Shapley values for all features using `vfunc`
`total_shapley_vals` ← sum of `shapley_vals`
return `payoff_total` == `total_shapley_vals`

Solution 2:

a) Pseudocode for predicting the Man of the Match probability through a random forest

Algorithm 9 Man of the Match

`df` ← read in *fifa.csv*
`df`['Man of the Match'] ← replace 'Yes' by TRUE (else FALSE)
`df` ← adapt `df` if needed for random forest model (e.g. removing NAs)
`train, test` ← split `df` in train and test data
`rf` ← random forest fit using train

b) Pseudocode of `m_vfunc()`

Algorithm 10 `m_vfunc()`

Require: `j`: feature index
Require: `obs`: observation
Require: `X`: feature matrix
Require: `predict`: ML model
Require: `nr_samples`: number of samples

`remainder` ← all features in `X` but `j`
`X_tmp` ← sample `nr_samples` samples from `X` (with replacing)
`X_tmp` ← replace features `J` with respective values from `obs`
`pred` ← use `model` for prediction with `X_tmp`
return mean of `pred`

c) Pseudocode of `shap_weights()`

Algorithm 11 `shap_weights()`

```
Require: mask: (binary) coalition feature space
p ← number of features in mask
zs ← coalition size
nominator ← p - 1
denominator ← (binomial coefficient of p over zs) * zs * (p - zs)
return nominator / denominator
```

Pseudocode of `replace_dataset()`

Algorithm 12 `replace_dataset()`

```
Require: obs: observation
Require: X: feature matrix
Require: nr_samples: number of samples
X_new ← sample nr_samples samples from X (with replacing)
obs_rep ← matrix with nr_samples columns containing obs in each column
mask ← matrix with randomly drawn entries from a binomial distribution (B(0, 0.5))
X_new ← replace entries where mask equals 1 with entry from obs
return X_new, mask
```

Pseudocode of `shap_data()`

Algorithm 13 `shap_data()`

```
Require: obs: observation
Require: X: feature matrix
Require: nr_samples: number of samples
Require: predict: prediction model
X_new, mask ← replace_dataset(obs, X, nr_samples)
weight ← shap_weights(mask)
pred ← predict(X_new)
return mask, pred, weight
```

Pseudocode of `kernel_shap()`

Algorithm 14 `kernel_shap()`

```
Require: obs: observation
Require: X: feature matrix
Require: nr_samples: number of samples
Require: predict: prediction model
mask, pred, weight ← shap_data(obs, X, nr_samples, predict)
lm ← weighted linear regression model using mask, pred and weight
return coefficients of lm
```

d) `kernel_shap(X_test[1, ], X_train, 1000, rf)`

(Intercept)	Goal.Scored	Ball.Possession..	Attempts
1.65457557	-3.28284372	-1.35771246	-0.64907257
On.Target	Off.Target	Blocked	Corners
-0.53569517	0.06829845	0.55281192	-2.18349575
Offsides	Free.Kicks	Saves	Pass.Accuracy..
-0.74949804	-0.51869972	2.86721030	-0.75230186
Passes	Distance.Covered..Kms.	Fouls.Committed	Yellow.Card
-0.42714693	-0.84923274	-0.96198507	0.70772339
Yellow...Red	Red	Goals.in.PSO	
0.18069980	-0.12571821	0.08003492	