Exercise 5

```
using LinearAlgebra, MLBase, GLMNet, DataFrames, CSV, Random, Plots, DecisionTree
```

```
categorize_value(pred_val) = argmin(abs.(pred_val .- 1))
calculate_accuracy(predicted_vals, actual_vals) = sum(predicted_vals .== actual_v
```

```
calculate_accuracy (generic function with 1 method)
```

```
function split_by_year(data_years, proportion)
    unique_years = unique(data_years)
    selected_ids = []
    for year in unique_years
        indices = findall(data_years .== year)
        sampled_ids = randsubseq(indices, proportion)
        append!(selected_ids, sampled_ids...)
    end
    return selected_ids
end
```

```
split_by_year (generic function with 1 method)
```

```
stock_data = CSV.read("/Users/michelletorres/Desktop/Homeworks AI/Smarket.csv", D
@show(stock_data)
@show(size(stock_data))
```

```
stock_data = 1250×9 DataFrame
```

Row	Year	Lag1	Lag2	Lag3	Lag4	Lag5	Volume	Today
	Int64	Float64	Float64	Float64	Float64	Float64	Float64	Float64
1	2001	0.381	-0.192	-2.624	-1.055	5.01	1.1913	0.959
2	2001	0.959	0.381	-0.192	-2.624	-1.055	1.2965	1.032
3	2001	1.032	0.959	0.381	-0.192	-2.624	1.4112	-0.623
4	2001	-0.623	1.032	0.959	0.381	-0.192	1.276	0.614
5	2001	0.614	-0.623	1.032	0.959	0.381	1.2057	0.213
6	2001	0.213	0.614	-0.623	1.032	0.959	1.3491	1.392
7	2001	1.392	0.213	0.614	-0.623	1.032	1.445	-0.403
8	2001	-0.403	1.392	0.213	0.614	-0.623	1.4078	0.027
9	2001	0.027	-0.403	1.392	0.213	0.614	1.164	1.303
10	2001	1.303	0.027	-0.403	1.392	0.213	1.2326	0.287
11	2001	0.287	1.303	0.027	-0.403	1.392	1.309	-0.498

12	2001	−0.498	0.287	1.303	0.027	−0.403	1.258	−0.189
13	2001	−0.189	−0.498	0.287	1.303	0.027	1.098	0.68
14	2001	0.68	−0.189	−0.498	0.287	1.303	1.0531	0.701
15	2001	0.701	0.68	−0.189	−0.498	0.287	1.1498	−0.562
16	2001	−0.562	0.701	0.68	−0.189	−0.498	1.2953	0.546
17	2001	0.546	−0.562	0.701	0.68	−0.189	1.1188	−1.747
18	2001	−1.747	0.546	−0.562	0.701	0.68	1.0484	0.359
19	2001	0.359	−1.747	0.546	−0.562	0.701	1.013	−0.151
20	2001	−0.151	0.359	−1.747	0.546	−0.562	1.0596	−0.841
21	2001	−0.841	−0.151	0.359	−1.747	0.546	1.1583	−0.623
22	2001	−0.623	−0.841	−0.151	0.359	−1.747	1.1072	−1.334
23	2001	−1.334	−0.623	−0.841	−0.151	0.359	1.0755	1.183
24	2001	1.183	−1.334	−0.623	−0.841	−0.151	1.0391	−0.865
25	2001	−0.865	1.183	−1.334	−0.623	−0.841	1.0752	−0.218
26	2001	−0.218	−0.865	1.183	−1.334	−0.623	1.1503	0.812
27	2001	0.812	−0.218	−0.865	1.183	−1.334	1.1537	−1.891
28	2001	−1.891	0.812	−0.218	−0.865	1.183	1.2572	−1.736
29	2001	−1.736	−1.891	0.812	−0.218	−0.865	1.1122	−1.851
30	2001	−1.851	−1.736	−1.891	0.812	−0.218	1.2085	−0.195
31	2001	−0.195	−1.851	−1.736	−1.891	0.812	1.3659	−0.556
32	2001	−0.556	−0.195	−1.851	−1.736	−1.891	1.2313	1.749
33	2001	1.749	−0.556	−0.195	−1.851	−1.736	1.1308	−0.766
34	2001	−0.766	1.749	−0.556	−0.195	−1.851	1.1141	−1.431
35	2001	−1.431	−0.766	1.749	−0.556	−0.195	1.2253	0.104
36	2001	0.104	−1.431	−0.766	1.749	−0.556	1.2949	−0.568
37	2001	−0.568	0.104	−1.431	−0.766	1.749	1.294	0.586
38	2001	0.586	−0.568	0.104	−1.431	−0.766	0.9292	0.998
39	2001	0.998	0.586	−0.568	0.104	−1.431	1.0918	0.645
40	2001	0.645	0.998	0.586	−0.568	0.104	1.1322	0.226
41	2001	0.226	0.645	0.998	0.586	−0.568	1.1141	−2.476
42	2001	−2.476	0.226	0.645	0.998	0.586	1.0859	−4.318
43	2001	−4.318	−2.476	0.226	0.645	0.998	1.229	1.483
44	2001	1.483	−4.318	−2.476	0.226	0.645	1.3609	−2.584
45	2001	−2.584	1.483	−4.318	−2.476	0.226	1.3974	0.587
46	2001	0.587	−2.584	1.483	−4.318	−2.476	1.2595	−1.962
47	2001	−1.962	0.587	−2.584	1.483	−4.318	1.54356	1.763
48	2001	1.763	−1.962	0.587	−2.584	1.483	1.1262	−2.408
49	2001	−2.408	1.763	−1.962	0.587	−2.584	1.2359	−1.792
50	2001	−1.792	−2.408	1.763	−1.962	0.587	1.3463	−0.406
51	2001	−0.406	−1.792	−2.408	1.763	−1.962	1.72395	1.991
52	2001	1.991	−0.406	−1.792	−2.408	1.763	1.3649	1.128
53	2001	1.128	1.991	−0.406	−1.792	−2.408	1.114	2.557
54	2001	2.557	1.128	1.991	−0.406	−1.792	1.3142	−2.443
55	2001	−2.443	2.557	1.128	1.991	−0.406	1.3334	−0.463

```
feature_matrix = Matrix(stock_data[:, 3:8])
println(names(stock_data))
labels = stock_data[:, 9]
```

```
["Year", "Lag1", "Lag2", "Lag3", "Lag4", "Lag5", "Volume", "Today", "Direction
1250-element PooledArrays.PooledVector{String7, UInt32, Vector{UInt32}}:
 "Up"
 "Up"
 "Down"
 "Up"
 "Up"
 "Up"
 "Down"
 "Up"
 "Up"
 "Up"
 "Down"
 "Down"
 "Up"
 ⋮
 "Up"
 "Down"
 "Down"
 "Down"
 "Down"
 "Up"
 "Up"
 "Up"
 "Down"
 "Up"
 "Down"
 "Down"
```

```
label_mapping = MLBase.labelmap(labels)
encoded_labels = labelencode(label_mapping, labels)
```

```
1250-element Vector{Int64}:
 1
 1
 2
 1
 1
 1
 2
 1
 1
 1
 2
 2
 1
 ⋮
 1
 2
 2
 2
 2
 1
 1
 1
 2
 1
 2
 2
```

```
training_indices = split_by_year(feature_matrix[:, 2], 0.7)
@show size(training_indices)
testing_indices = setdiff(1:length(feature_matrix[:, 2]), training_indices)
@show size(testing_indices)
```

```
size(training_indices) = (847,)
size(testing_indices) = (403,)
(403,)
```

Lasso Regression

```
lasso_model = glmnet(feature_matrix[training_indices, :], encoded_labels[training
lasso_cv = glmnetcv(feature_matrix[training_indices, :], encoded_labels[training_
optimal_lambda_lasso = lasso_model.lambda[argmin(lasso_cv.meanloss)]
lasso_model = glmnet(feature_matrix[training_indices, :], encoded_labels[training
lasso_predictions = GLMNet.predict(lasso_model, feature_matrix[testing_indices, :
lasso_predictions = categorize_value.(lasso_predictions)
calculate_accuracy(lasso_predictions, encoded_labels[testing_indices])
```

```
0.5086848635235732
```

Ridge Regression

```
ridge_model = glmnet(feature_matrix[training_indices, :], encoded_labels[training
ridge_cv = glmnetcv(feature_matrix[training_indices, :], encoded_labels[training_
optimal_lambda_ridge = ridge_model.lambda[argmin(ridge_cv.meanloss)]
ridge_model = glmnet(feature_matrix[training_indices, :], encoded_labels[training
ridge_predictions = GLMNet.predict(ridge_model, feature_matrix[testing_indices, :
ridge_predictions = categorize_value.(ridge_predictions)
calculate_accuracy(ridge_predictions, encoded_labels[testing_indices])
```

```
0.5086848635235732
```

Elastic Net

```
elastic_net_model = glmnet(feature_matrix[training_indices, :], encoded_labels[tr
elastic_net_cv = glmnetcv(feature_matrix[training_indices, :], encoded_labels[tra
optimal_lambda_en = elastic_net_model.lambda[argmin(elastic_net_cv.meanloss)]
elastic_net_model = glmnet(feature_matrix[training_indices, :], encoded_labels[tr
en_predictions = GLMNet.predict(elastic_net_model, feature_matrix[testing_indices
en_predictions = categorize_value.(en_predictions)
calculate_accuracy(en_predictions, encoded_labels[testing_indices])
```

```
0.5086848635235732
```

Decision Tree

```
tree_model = DecisionTreeClassifier(max_depth=2)
DecisionTree.fit!(tree_model, feature_matrix[training_indices, :], encoded_labels
tree_predictions = DecisionTree.predict(tree_model, feature_matrix[testing_indice
calculate_accuracy(tree_predictions, encoded_labels[testing_indices])
```

```
1.0
```

Random Forest

```
rf_model = DecisionTree.RandomForestClassifier(n_trees=20)
DecisionTree.fit!(rf_model, feature_matrix[training_indices, :], encoded_labels[t
rf_predictions = DecisionTree.predict(rf_model, feature_matrix[testing_indices, :
calculate_accuracy(rf_predictions, encoded_labels[testing_indices])
```

```
1.0
```

k-Nearest Neighbors

```
train_features = feature_matrix[training_indices, :]
train_labels = encoded_labels[training_indices]
kdtree_model = KDTree(train_features')
query_points = feature_matrix[testing_indices, :]
nearest_indices, distances = knn(kdtree_model, query_points', 5, true)
neighbors_labels = train_labels[hcat(nearest_indices...)]
label_counts = map(i -> counter(neighbors_labels[:, i]), 1:size(neighbors_labels,
knn_predictions = map(i -> parse(Int, string(argmax(label_counts[i]))), 1:size(ne
calculate_accuracy(knn_predictions, encoded_labels[testing_indices])
```

```
0.9181141439205955
```

```
svm_model = svmtrain(train_features', train_labels) # Support vector machine
svm_predictions, decision_values = svmpredict(svm_model, feature_matrix[testing_i
calculate_accuracy(svm_predictions, encoded_labels[testing_indices])
```

```
0.9751861042183623
```

```
accuracy_scores = zeros(7) # Compare overall accuracies
methods_list = ["Lasso", "Ridge", "ElasticNet", "DecisionTree", "RandomForest", "
test_labels = encoded_labels[testing_indices]
accuracy_scores[1] = calculate_accuracy(lasso_predictions, test_labels)
accuracy_scores[2] = calculate_accuracy(ridge_predictions, test_labels)
accuracy_scores[3] = calculate_accuracy(en_predictions, test_labels)
accuracy_scores[4] = calculate_accuracy(tree_predictions, test_labels)
accuracy_scores[5] = calculate_accuracy(rf_predictions, test_labels)
accuracy_scores[6] = calculate_accuracy(knn_predictions, test_labels)
accuracy_scores[7] = calculate_accuracy(svm_predictions, test_labels)
hcat(methods_list, accuracy_scores)
```

```
7×2 Matrix{Any}:
 "Lasso"         0.508685
 "Ridge"         0.508685
 "ElasticNet"    0.508685
 "DecisionTree"  1.0
 "RandomForest"  1.0
 "kNN"           0.918114
 "SVM"           0.975186
```

Confusion matrices

```
println("Confusion matrix Lasso")
pretty_table(confusmat(2, test_labels, lasso_predictions[:]))
println("Confusion matrix Ridge")
pretty_table(confusmat(2, test_labels, ridge_predictions[:]))
println("Confusion matrix Elastic Net")
pretty_table(confusmat(2, test_labels, en_predictions[:]))
println("Confusion matrix Decision Tree")
pretty_table(confusmat(2, test_labels, tree_predictions[:]))
println("Confusion matrix Random Forest")
pretty_table(confusmat(2, test_labels, rf_predictions[:]))
println("Confusion matrix kNN")
pretty_table(confusmat(2, test_labels, knn_predictions[:]))
println("Confusion matrix SVM")
pretty_table(confusmat(2, test_labels, svm_predictions[:]))
```

```
Confusion matrix Lasso
```

Col. 1	Col. 2
205	0

198	0

Confusion matrix Ridge

Col. 1	Col. 2
205	0
198	0

Confusion matrix Elastic Net

Col. 1	Col. 2
205	0
198	0

Confusion matrix Decision Tree

Col. 1	Col. 2
205	0
0	198

Confusion matrix Random Forest

Col. 1	Col. 2
205	0
0	198

Confusion matrix kNN

Col. 1	Col. 2
198	7
26	172

Confusion matrix SVM

Col. 1	Col. 2
204	1
9	189