NAMIBIA UNIVERSITY
OF SCIENCE AND TECHNOLOGY

FACULTY OF COMPUTING AND INFORMATICS
DEPARTMENT OF INFORMATICS

MASTERS OF DATA SCIENCE

Trends in Artificial Intelligence and Machine Learning (TAI911S)

GROUP ASSIGNMENT 1

STUDENT NAME: ABINER HAIKIYATA (224002090)
STUDENT NAME: LAHYA SOFIKA (225111004)

LECTURER: Prof Quenum

Import libraries for data processing, modeling, and visualization

```
using CSV, DataFrames, MLJ, Plots, MLJLinearModels, ScientificTypes, MLJBase, Tables,
CategoricalArrays, StatisticalMeasures
```

1. Explicitly import selected functions to avoid naming conflicts

```
begin
   import DataFrames: select, Not, rename!
   import MLJ: machine, fit!, predict, predict_mode
end
```

Explanation:

- We load necessary packages for reading CSVs, managing dataframes, modeling (MLJ), plotting, and evaluation.
- Explicit imports prevent naming conflicts, especially for select, which exists in multiple packages.

2. Load and clean the food wastage dataset

2. Loadning and Cleaning dataset

```
md"""
### 2. Loadning and Cleaning dataset
"""
```

	Country	Year	Food Category	Total Waste (Tons)	Economic_Loss	Avg Waste per Capita (Kg)	Popula (Milli
	String15	Int64	String31	Float64	Float64	Float64	Floa1
1	"Australia"	2019	"Fruits & Vegetables"	19268.6	18686.7	72.69	87.59
2	"Indonesia"	2019	"Prepared Food"	3916.97	4394.48	192.52	1153.9
3	"Germany"	2022	"Dairy Products"	9700.16	8909.16	166.94	1006.1
4	"France"	2023	"Fruits & Vegetables"	46299.7	40551.2	120.19	953.05
5	"France"	2023	"Beverages"	33096.6	36980.8	104.74	1105.4
6	"India"	2024	"Fruits & Vegetables"	11962.9	11196.0	136.21	1311.9
7	"Germany"	2024	"Prepared Food"	45038.7	39191.2	179.27	1349.4
8	"China"	2019	"Fruits & Vegetables"	12791.2	12233.3	90.8	1229.2
9	"UK"	2019	"Meat & Seafood"	14795.6	14347.0	128.91	450.33
10	"India"	2019	"Grains & Cereals"	12118.3	13631.2	141.75	359.26
⋮ more							
5000	"France"	2024	"Bakery Items"	8860.27	7360.38	51.5	879.67

```
begin
   food_waste_df = CSV.read("global_food_wastage_dataset.csv", DataFrame)
   # Rename specific columns to remove special characters and spaces
   rename!(food_waste_df,
      "Economic Loss (Million \$)" => "Economic_Loss",
      "Household Waste (%)" => "Household_Waste")
End
```

Explanation:

- Reads the dataset and stores it in food waste df.
- Renames columns for cleaner access (e.g., to avoid spaces and special characters).

3. Load the fetal health dataset

fetal_health_df =

	baseline value	accelerations	fetal_movement	uterine_contractions	light_decelerations	se
1	120	0.0	0.0	0.0	0.0	0.
2	132	0.006	0.0	0.006	0.003	0.
3	133	0.003	0.0	0.008	0.003	0.
4	134	0.003	0.0	0.008	0.003	0.
5	132	0.007	0.0	0.008	0.0	0.
6	134	0.001	0.0	0.01	0.009	0.
7	134	0.001	0.0	0.013	0.008	0.
8	122	0.0	0.0	0.0	0.0	0.
9	122	0.0	0.0	0.002	0.0	0.
10	122	0.0	0.0	0.003	0.0	0.
⋮ more						
2126	142	0.002	0.002	0.008	0.0	0.

fetal_health_df = CSV.read("fetal_health.csv", DataFrame)

Explanation:

- Reads a dataset that includes features and a categorical label (fetal_health) for classification tasks.

4. Prepare features for the food waste dataset (drop target columns)

```
# ----------------------------------------------------------------
# PROBLEM 1: REGRESSION MODEL
# ----------------------------------------------------------------
```

X_fw =

	Country	Year	Food Category	Total Waste (Tons)	Avg Waste per Capita (Kg)	Population (Million)
1	"Australia"	2019	"Fruits & Vegetables"	19268.6	72.69	87.59
2	"Indonesia"	2019	"Prepared Food"	3916.97	192.52	1153.99
3	"Germany"	2022	"Dairy Products"	9700.16	166.94	1006.11
4	"France"	2023	"Fruits & Vegetables"	46299.7	120.19	953.05
5	"France"	2023	"Beverages"	33096.6	104.74	1105.47
6	"India"	2024	"Fruits & Vegetables"	11962.9	136.21	1311.91
7	"Germany"	2024	"Prepared Food"	45038.7	179.27	1349.45
8	"China"	2019	"Fruits & Vegetables"	12791.2	90.8	1229.29
9	"UK"	2019	"Meat & Seafood"	14795.6	128.91	450.33
10	"India"	2019	"Grains & Cereals"	12118.3	141.75	359.26
⋮ more						
5000	"France"	2024	"Bakery Items"	8860.27	51.5	879.67

```
X_fw = select(food_waste_df, Not([:Economic_Loss, :Household_Waste]))
```

5. Convert string features to numeric codes

```
for col in names(X_fw)
    if eltype(X_fw[!, col]) <: AbstractString
        X_fw[!, col] = levelcode.(categorical(X_fw[!, col]))
    end
end
```

Explanation:

- X_fw will hold the independent variables for modeling.
- If any column contains strings, we convert them to categorical then to numerical codes (levelcode) for ML modeling.

6. Convert cleaned DataFrame to Matrix and then to MLJ-compatible table

```
X_fw_matrix = 5000×6 Matrix{Float64}:
              2.0  2019.0  5.0  19268.6    72.69    87.59
              9.0  2019.0  8.0   3916.97  192.52  1153.99
              7.0  2022.0  3.0   9700.16  166.94  1006.11
              6.0  2023.0  5.0  46299.7   120.19   953.05
              6.0  2023.0  2.0  33096.6   104.74  1105.47
              8.0  2024.0  5.0  11962.9   136.21  1311.91
              7.0  2024.0  8.0  45038.7   179.27  1349.45
              ⋮                                    ⋮
              9.0  2023.0  8.0  39980.4    32.34    27.08
              6.0  2021.0  2.0  47524.7    77.41  1087.46
              2.0  2021.0  2.0  32337.7   194.35  1336.32
              5.0  2018.0  7.0  20641.0    21.04    16.13
              2.0  2021.0  2.0  26566.6   197.14  1086.17
              6.0  2024.0  1.0   8860.27   51.5    879.67
```

Convert to matrix and handle missing values

X_fw_matrix = Matrix{Float64}(X_fw)

X_fw_matrix = Matrix{Float64}(X_fw)

X_fw_table = MLJBase.table(X_fw_matrix, names=propertynames(X_fw))

Explanation:

- Convert the DataFrame to a matrix of Float64 (numeric values).
- Convert that matrix into a MLJ table which is needed for training models.

7. Prepare target variables for regression

X_fw_table =

	Country	Year	Food Category	Total Waste (Tons)	Avg Waste per Capita (Kg)	Population (Million)
1	2.0	2019.0	5.0	19268.6	72.69	87.59
2	9.0	2019.0	8.0	3916.97	192.52	1153.99
3	7.0	2022.0	3.0	9700.16	166.94	1006.11
4	6.0	2023.0	5.0	46299.7	120.19	953.05
5	6.0	2023.0	2.0	33096.6	104.74	1105.47
6	8.0	2024.0	5.0	11962.9	136.21	1311.91
7	7.0	2024.0	8.0	45038.7	179.27	1349.45
8	5.0	2019.0	5.0	12791.2	90.8	1229.29
9	19.0	2019.0	7.0	14795.6	128.91	450.33
10	8.0	2019.0	6.0	12118.3	141.75	359.26

⋮ more

```
X_fw_table = MLJBase.table(X_fw_matrix, names=propertynames(X_fw))
```

```
y_econ =
Add cell (Ctrl + Enter) .48, 8909.16, 40551.2, 36980.8, 11196.0, 39191.2, 12233.3, 14347.0, 13631.2,
```

```
    # Prepare targets
y_econ = Float64.(skipmissing(food_waste_df.Economic_Loss)) |> collect
```

151 ms

```
y_household =
▸[53.64, 30.61, 48.08, 31.91, 36.06, 37.09, 68.93, 59.9, 59.01, 42.42, 31.16, 33.79, 45.43, 3
```

```
y_household = Float64.(skipmissing(food_waste_df.Household_Waste)) |> collect
```

y_econ = Float64.(skipmissing(food_waste_df.Economic_Loss)) |> collect

y_household = Float64.(skipmissing(food_waste_df.Household_Waste)) |> collect

Explanation:

- Extracts target values (y_econ, y_household) and removes missing values.

8. Create and train models for Economic Loss and Household Waste

```
model_fw = LinearRegressor(
            fit_intercept = true,
            solver = nothing)

# Create and fit models
model_fw = LinearRegressor()
```

```
37.7 μs
```

```
mach_econ =
trained Machine; caches model-specific representations of data
  model: LinearRegressor(fit_intercept = true, …)
  args:
    1:  Source @255 ⏎ ScientificTypesBase.Table{AbstractVector{ScientificTypesBase.Continu
    2:  Source @639 ⏎ AbstractVector{ScientificTypesBase.Continuous}

mach_econ = machine(model_fw, X_fw_table, y_econ) |> fit!
```

```
Training machine(LinearRegressor(fit_intercept = true, …), …).
```

```
Solver: MLJLinearModels.Analytical
  iterative: Bool false
  max_inner: Int64 200
```

```
mach_household =
trained Machine; caches model-specific representations of data
  model: LinearRegressor(fit_intercept = true, …)
  args:
    1:  Source @703 ⏎ ScientificTypesBase.Table{AbstractVector{ScientificTypesBase.Continu
    2:  Source @253 ⏎ AbstractVector{ScientificTypesBase.Continuous}

mach_household = machine(model_fw, X_fw_table, y_household) |> fit!
```

```
Training machine(LinearRegressor(fit_intercept = true, …), …).
```

```
Solver: MLJLinearModels.Analytical
  iterative: Bool false
  max_inner: Int64 200
```

mach_econ = machine(model_fw, X_fw_table, y_econ) |> fit!

mach_household = machine(model_fw, X_fw_table, y_household) |> fit!

Explanation:

- Create MLJ machines for each regression task and train them using fit!.

9. Prepare fetal health classification dataset

```
# ----------------------------------------------------------------
# PROBLEM 2: CLASSIFICATION MODEL
# ----------------------------------------------------------------
```

X_fetal =

	baseline value	accelerations	fetal_movement	uterine_contractions	light_decelerations	se
1	120	0.0	0.0	0.0	0.0	0.
2	132	0.006	0.0	0.006	0.003	0.
3	133	0.003	0.0	0.008	0.003	0.
4	134	0.003	0.0	0.008	0.003	0.
5	132	0.007	0.0	0.008	0.0	0.
6	134	0.001	0.0	0.01	0.009	0.
7	134	0.001	0.0	0.013	0.008	0.
8	122	0.0	0.0	0.0	0.0	0.
9	122	0.0	0.0	0.002	0.0	0.
10	122	0.0	0.0	0.003	0.0	0.
⋮ more						
2126	142	0.002	0.002	0.008	0.0	0.

```
X_fetal = select(fetal_health_df, Not(:fetal_health))
```

```
y_fetal =
▸CategoricalArrays.CategoricalVector{Int64, UInt32, Int64, CategoricalArrays.CategoricalValue{I
```

```
y_fetal = categorical(fetal_health_df.fetal_health)
```

X_fetal = select(fetal_health_df, Not(:fetal_health))

y_fetal = categorical(fetal_health_df.fetal_health)

```
X_fetal_matrix =
2126×21 Matrix{Float64}:
 120.0  0.0    0.0    0.0    0.0    0.0  …  0.0  120.0  137.0  121.0   73.0  1.0
 132.0  0.006  0.0    0.006  0.003  0.0     1.0  141.0  136.0  140.0   12.0  0.0
 133.0  0.003  0.0    0.008  0.003  0.0     1.0  141.0  135.0  138.0   13.0  0.0
 134.0  0.003  0.0    0.008  0.003  0.0     0.0  137.0  134.0  137.0   13.0  1.0
 132.0  0.007  0.0    0.008  0.0    0.0     0.0  137.0  136.0  138.0   11.0  1.0
 134.0  0.001  0.0    0.01   0.009  0.0  …  3.0   76.0  107.0  107.0  170.0  0.0
 134.0  0.001  0.0    0.013  0.008  0.0     3.0   71.0  107.0  106.0  215.0  0.0
   ⋮                                 ⋮    ⋱   ⋮                               ⋮
 140.0  0.0    0.0    0.005  0.001  0.0  …  0.0  145.0  143.0  145.0    2.0  0.0
 140.0  0.0    0.0    0.007  0.0    0.0     0.0  153.0  150.0  152.0    2.0  0.0
 140.0  0.001  0.0    0.007  0.0    0.0     0.0  152.0  148.0  151.0    3.0  1.0
 140.0  0.001  0.0    0.007  0.0    0.0     0.0  153.0  148.0  152.0    4.0  1.0
 140.0  0.001  0.0    0.006  0.0    0.0     0.0  152.0  147.0  151.0    4.0  1.0
 142.0  0.002  0.002  0.008  0.0    0.0  …  1.0  145.0  143.0  145.0    1.0  0.0
# Convert to numeric matrix
X_fetal_matrix = Matrix{Float64}(X_fetal)
```

Explanation:

- Separate predictors (X_fetal) and target (y_fetal) and convert target to categorical for classification.

10. Split data into training and testing for classification

X_fetal_table =

	baseline value	accelerations	fetal_movement	uterine_contractions	light_decelerations	seve
1	120.0	0.0	0.0	0.0	0.0	0.0
2	132.0	0.006	0.0	0.006	0.003	0.0
3	133.0	0.003	0.0	0.008	0.003	0.0
4	134.0	0.003	0.0	0.008	0.003	0.0
5	132.0	0.007	0.0	0.008	0.0	0.0
6	134.0	0.001	0.0	0.01	0.009	0.0
7	134.0	0.001	0.0	0.013	0.008	0.0
8	122.0	0.0	0.0	0.0	0.0	0.0
9	122.0	0.0	0.0	0.002	0.0	0.0
10	122.0	0.0	0.0	0.003	0.0	0.0

⋮ more

```
X_fetal_table = MLJBase.table(X_fetal_matrix, names=propertynames(X_fetal))
```

```
([279, 1880, 57, 905, 1114, 965, 165, 1350, 1028, … more ,915], [398, 1709, 399, 706, 1805, 1
```

Train-test split

```
train, test = partition(eachindex(y_fetal), 0.7, shuffle=true, rng=123)
```

Explanation:

- Then convert predictors into an MLJ table
- Randomly split the data into 70% training and 30% testing for classification tasks.

11. Load and train a logistic classifier

```
LogisticClassifier
@load LogisticClassifier pkg=MLJLinearModels
```

```
For silent loading, specify `verbosity=0`.
import MLJLinearModels ✓
```

```
model_fetal = LogisticClassifier(
                lambda = 2.220446049250313e-16,
                gamma = 0.0,
                penalty = :l2,
                fit_intercept = true,
                penalize_intercept = false,
                scale_penalty_with_samples = true,
                solver = nothing)
model_fetal = LogisticClassifier()
```

```
mach_fetal =
trained Machine; caches model-specific representations of data
  model: LogisticClassifier(lambda = 2.220446049250313e-16, …)
  args:
    1:  Source @604 ⏎ ScientificTypesBase.Table{AbstractVector{ScientificTypesBase.Continu
    2:  Source @979 ⏎ AbstractVector{ScientificTypesBase.Multiclass{3}}
mach_fetal = machine(model_fetal, X_fetal_table, y_fetal) |> fit!
```

```
Training machine(LogisticClassifier(lambda = 2.220446049250313e-16, …), …).
Solver: MLJLinearModels.LBFGS{Optim.Options{Float64, Nothing}, @NamedTuple{}}
  optim_options: Optim.Options{Float64, Nothing}
  lbfgs_options: @NamedTuple{} NamedTuple()
```

```
y_pred =
▸CategoricalArrays.CategoricalVector{Int64, UInt32, Int64, CategoricalArrays.CategoricalValue{I
# Evaluate
y_pred = predict_mode(mach_fetal, rows=test)
```

```
0.8589341692789969
begin
    import StatisticalMeasures ✓: accuracy
acc = accuracy(y_pred, y_fetal[test])  # This now returns a plain Float64
end
```

```
# Correct printing method
println("Classification Accuracy: ", round(acc * 100, digits=2), "%")
Classification Accuracy: 85.89%
```

@load LogisticClassifier pkg=MLJLinearModels

model_fetal = LogisticClassifier()

mach_fetal = machine(model_fetal, X_fetal_table, y_fetal) |> fit!

y_pred = predict_mode(mach_fetal, rows=test)

```
begin

    import StatisticalMeasures: accuracy

    acc = accuracy(y_pred, y_fetal[test])

end

# Print the classification accuracy

println("Classification Accuracy: ", round(acc * 100, digits=2), "%")
```

Explanation:

- Load a logistic regression model and fit it on the training data to classify fetal health status.
- Predicts classes using the model (predict_mode) and computes accuracy on the test set.
- Outputs classification performance.

12. Predict values for regression models

```
# ------------------------------------------------------------------
# VISUALIZATION
# ------------------------------------------------------------------

y_econ_pred =
[19359.8, 3824.22, 9638.11, 46270.5, 33079.6, 11879.1, 44925.4, 12800.7, 14730.4, 12111.4,

# Convert predictions to vectors
y_econ_pred = predict(mach_econ, X_fw_table) |> vec

y_household_pred =
[50.2663, 49.5133, 49.5072, 49.9248, 49.8922, 49.4456, 49.4884, 49.9962, 50.1959, 49.9493,

y_household_pred = predict(mach_household, X_fw_table) |> vec
```

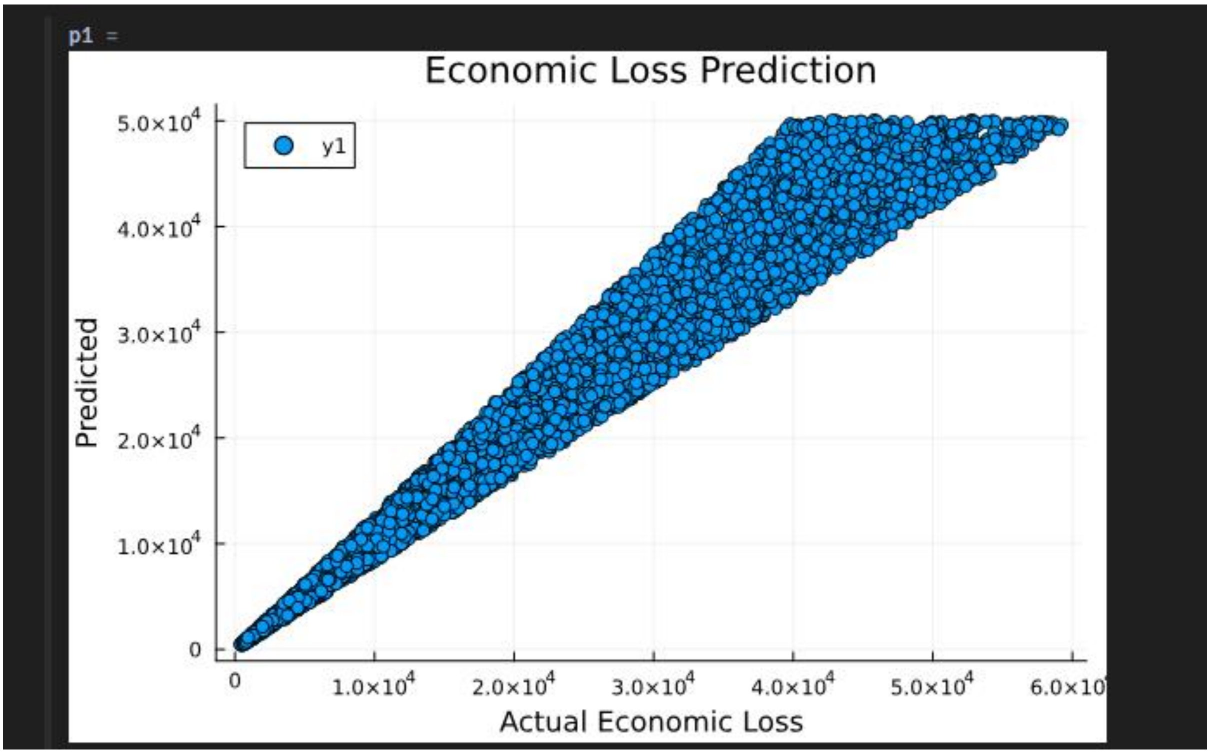
```
y_econ_pred = predict(mach_econ, X_fw_table) |> vec

y_household_pred = predict(mach_household, X_fw_table) |> vec
```

Explanation:

- Predict continuous values using the trained regression models and flatten the results to vectors.

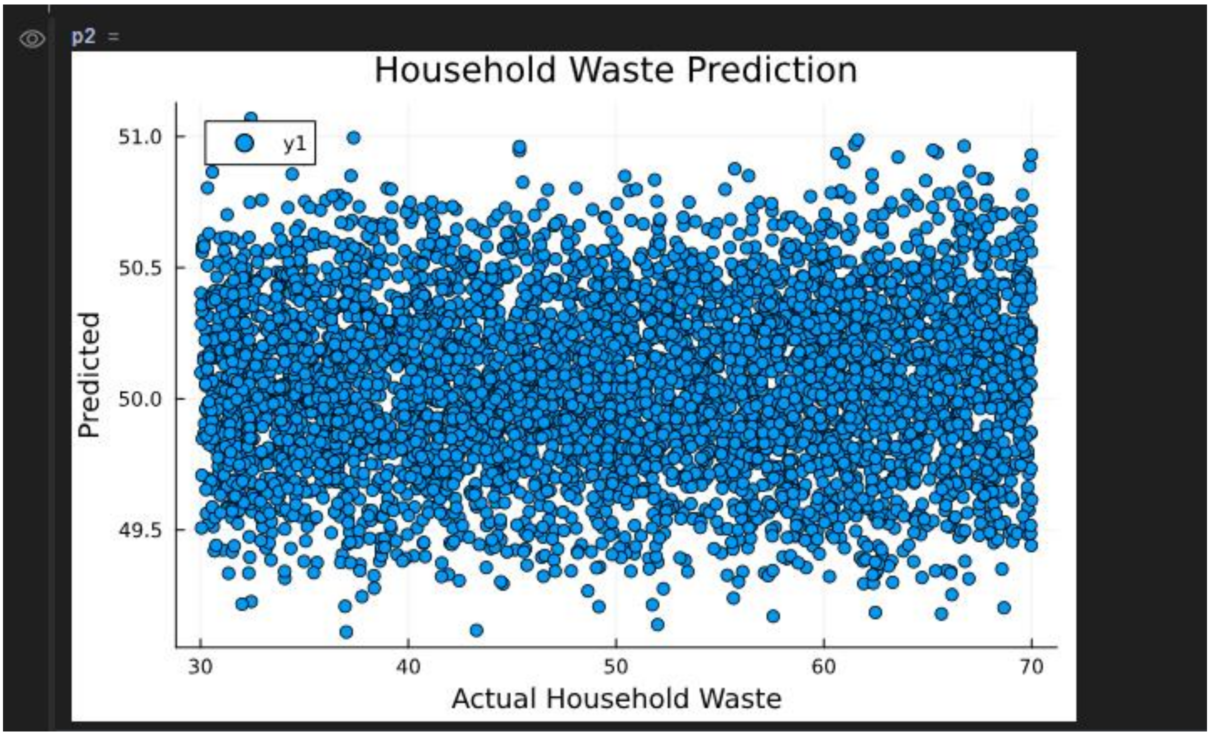
13. Visualize regression predictions



p1 = scatter(y_econ, y_econ_pred,

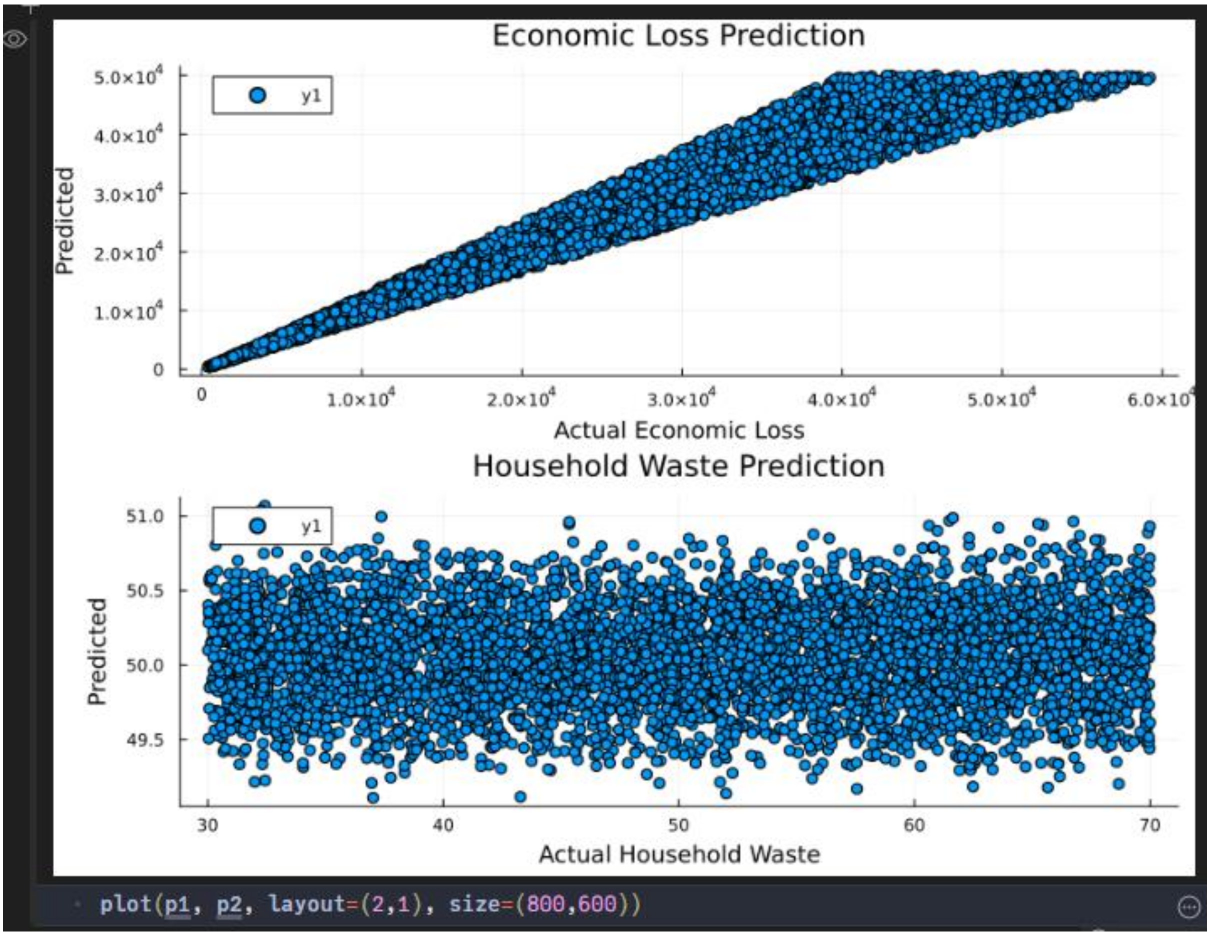
xlabel="Actual Economic Loss", ylabel="Predicted",

title="Economic Loss Prediction")



p2 = scatter(y_household, y_household_pred, xlabel="Actual Household Waste", ylabel="Predicted",

title="Household Waste Prediction")



plot(**p1, p2,** layout=(2,1), size=(800,600))

Explanation:

- Creates scatter plots comparing actual vs predicted values for both regression tasks.

14. Generate confusion matrix for classification evaluation

cm =

	Ground Truth		
Predicted	1	2	3
1	483	63	4
2	9	32	5
3	4	5	33

cm = confusion_matrix(y_pred, y_fetal[test])

Explanation:

- Displays the confusion matrix, showing how well the classifier distinguishes between classes.

Short Summary

The Above code performs three machine learning tasks:

1. **Regression** - On the food wastage dataset to predict economic loss and household waste.
2. **Classification** - On fetal health data using logistic regression.
3. **Evaluation and visualization** - Using accuracy and scatter plots.