Description:

This code includes data preparation, analysis, and K-Nearest Neighbors (K-NN) classification. It performs analysis on a dataset and simulates a K-NN classification scenario for categorizing oranges based on their size into "Luxury Hotels," "Edible Fruit," or "Juice."

1. **Data Preparation and Analysis**

Code:

```
using DataFrames
using GLM
using Random
using Statistics

Random.seed!(1234)

df = DataFrame(randn(50, 9), :auto)
df.y = sum(eachcol(df)) .+ 1.0 .+ randn(50)

model = lm(formula, df)

# Calculate Mean Square Error of the model
deviance(model) / nrow(df)

mse(model, df) = sum(x->x^2, predict(model, df) - df.y) / nrow(df)
mse(model, df)

# Cross-validation
df.fold = shuffle!((1:nrow(df)) .% 10)

get_fold_data(df, fold) =
 (train=view(df, df.fold .!= fold, :),
  test=view(df, df.fold .== fold, :))

mean(0:9) do fold
    train, test = get_fold_data(df, fold)
    model_cv = lm(formula, train)
    return mse(model_cv, test)
end
```

Input:

- A dataset containing random values for 50 samples across 9 variables.
- A linear regression model ("model") is fit to the data.
- Cross-validation is performed using a shuffled fold index.

Output:

- Mean Square Error of the linear regression model on the dataset.
- Mean Square Error for cross-validated models.

2. Simulation for K-Nearest Neighbors Classification

Code:

```
using DataFrames
using GLM
using Random

mse(model, df) = sum(x->x^2, predict(model, df) - df.y) / nrow(df)
mset(model) = 1 + sum(x -> (1 - x) ^ 2, coef(model))

get_fold_data(df, fold) =
 (train=view(df, df.fold .!= fold, :),
  test=view(df, df.fold .== fold, :))

function runtest(id)
    df = DataFrame(randn(50, 9), :auto)
    df.y = sum(eachcol(df)[1:5]) .+ 1.0 .+ randn(50)
    formulas = [Term(:y) ~ sum([Term(Symbol(:x, i)) for i in 1:n]) for n in 1:9]
    models = [lm(f, df) for f in formulas]
    mse_wholes = [mse(m, df) for m in models]
    mse_ts = [mset(m) for m in models]
    df.fold = shuffle!((1:nrow(df)) .% 10)
    mse_cvs = map(formulas) do f
        return mean(0:9) do fold
            train, test = get_fold_data(df, fold)
            model_cv = lm(f, train)
            return mse(model_cv, test)
        end
    end
    return DataFrame(id=id, vars=1:9, mse_whole=mse_wholes,
mse_cv=mse_cvs, mse_t=mse_ts)
end

Random.seed!(12)
res = DataFrame([runtest() for _ in 1:10_000])

describe(res, :all)
cor(Matrix(res))
```

Input:

- Simulated dataset with random values.
- Linear regression models are trained with different formulas.
- Cross-validation is performed to assess model performance.

Output:

- Summary statistics and correlations for the simulation results.

3. K-Nearest Neighbors Classification

Description:

The following code simulates a K-Nearest Neighbors (K-NN) classification scenario for categorizing oranges based on their size into "Luxury Hotels," "Edible Fruit," or "Juice."

Code:

```
# Determine if the orange should be sent to luxury hotels, retail as edible fruit, or juice factories.
# Label one of these using a simulation function.

function category(orange_size)
    if orange_size > 6
        return "Luxury Hotels"
    end
    if orange_size > 4
        return "Edible Fruit"
    end
    return "Juice"
end
```

Input:

- Orange sizes

Output:

- Labels for oranges categorizing them into "Luxury Hotels," "Edible Fruit," or "Juice."

4. Basic Functions

Code:

```
# Count occurrences of elements in an array.

function counter(array_of_elements)
    counts = Dict()
    for element in array_of_elements
        counts[element] = get(counts, element, 0) + 1
    end
    counts
end
```

Input:

- An array of elements.

Output:

- A dictionary with counts of each unique element.

5. **Plotting Sample Data**

Code:

```
key_values = Dict(
    "Luxury Hotels" => [],
    "Juice" => [],
    "Edible Fruit" => []
)

# Fill the key_values dictionary with data
for (size, sell) in orange_sizes
    push!(key_values[sell], size)
end

# Plotting using Plots library
using Plots

label = "Juice"
y = key_values[label]
x = fill(5, length(y))
p = scatter!(x, y, xlims=(4, 6), ylims=(0, 10), label=label, title="Orange size and
sale", ylabel="Size in cms", color="blue")

label = "Edible Fruit"
y = key_values[label]
x = fill(5, length(y))
scatter!(p, x, y, label=label, color="red")
```

```
label = "Luxury Hotels"
y = key_values[label]
x = fill(5, length(y))
scatter!(p, x, y, label=label, color="orange")
```

Input:

- Data points for orange sizes and their corresponding categories.

Output:

- A plot showing the relationship between orange size and sale category.

6. K-Nearest Neighbors Classification

Code:

```
# Extract errors and sells from the data
errors_and_sells = []
for (size, sell) in orange_sizes
    push!(errors_and_sells, (size, sell))
end

# Take the K nearest errors and sells
k = 20
nearest_errors_and_sells = sort(errors_and_sells)[1:k]

# Extract the labels from the nearest errors and sells
nearest_sells = [sell for (_error, sell) in nearest_errors_and_sells]

# Count the labels
counted_nearest = counter(nearest_sells)

# Find the label with the highest count
highest_vote(counted_nearest)
```

Input:

- Orange sizes and their corresponding labels.

Output:

- The label that received the highest count among the K-nearest neighbors.