>	Imports
[] Ļ 1 hücre gizli
>	Install Dataset
[] կ 2 hücre gizli
>	X - EDA - Graphs
[] l, 14 hücre gizli
>	Y - EDA - Graphs
[] L, 2 hücre gizli
	X - EDA
<u> </u>] Ļ 9 hücre gizli
>	Y - EDA
[] կ 4 hücre gizli
>	Decision Tree
[] L, 10 hücre gizli
>	Build Part - dt
[] L, 1 hücre gizli
>	Predict Part - dt

[] , 1 hücre gizli

Decision Tree Method Calling

```
options = {
    'max_depth': 15,
    'min_samples': 10,
}
attribute_types = {}

y_series = y["Rings"]
tree = build_dt(X, y_series, attribute_types, options)

accuracy = predict_dt(tree, X, options)

print(f"Accuracy: {accuracy}")

    Accuracy: 0.6061766818290639

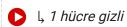
pprint(tree)

    ('Whole_weight <= 0.119': [{'Shell_weight <= 0.03025': [{'}</pre>
```

```
['Sex_I <= 0.5': [{'Shucked_weight <= 0.233': [{'Viscera_weight <= 0.05925': [{'Diamete
```

{ 'Shucked

> K-Fold



Random Decision Forest

```
def create_leaf(data):
    label_column = data[:, -1]
    leaf = np.mean(label_column)
    return leaf
```

```
def calculate_mse(data):
    actual_values = data[:, -1]
    if len(actual_values) == 0: # empty data
        mse = 0
    else:
        prediction = np.mean(actual_values)
        mse = np.mean((actual_values - prediction) **2)
    return mse
def determine_type_of_feature(df):
    feature_types = []
    n_unique_values_treshold = 15
    for feature in df.columns:
        if feature != "label":
            unique values = df[feature].unique()
            example_value = unique_values[0]
            if (isinstance(example_value, str)) or (len(unique_values) <= n_unique_values_tr</pre>
                feature_types.append("categorical")
            else:
                feature_types.append("continuous")
    return feature_types
```

```
def reg_decision_tree_algorithm(df, counter=0, min_samples=2, max_depth=5):
    if counter == 0:
        global COLUMN_HEADERS, FEATURE_TYPES
        COLUMN HEADERS = df.columns
        FEATURE_TYPES = determine_type_of_feature(df)
        data = df.values
        return data
    else:
        data = df
    if (len(data) < min_samples) or (counter == max_depth):</pre>
        leaf = create_leaf(data)
        return leaf
    else:
        counter += 1
        potential_splits = get_potential_splits(data)
        split_column, split_value = determine_best_split(data, potential_splits)
        data_below, data_above = split_data(data, split_column, split_value)
        feature_name = np.array(COLUMN_HEADERS)[split_column]
        if split column is not None:
            type_of_feature = FEATURE_TYPES[split_column]
            if type_of_feature == "continuous":
                question = "{} <= {}".format(feature_name, split value)</pre>
            else:
                question = "{} = {}".format(feature_name, split_value)
        else:
            question = "{} = {}".format(feature_name, split_value)
            type of feature = None
            question = "Unknown"
        sub_tree = {question: []}
        yes_answer = reg_decision_tree_algorithm(data_below, counter, min_samples, max_depth
        no_answer = reg_decision_tree_algorithm(data_above, counter, min_samples, max depth)
        if yes_answer == no_answer:
            sub tree = yes answer
        else:
            sub tree[question].append(yes answer)
            sub_tree[question].append(no_answer)
        return sub_tree
```

```
def random_forest_algorithm(train_df, n_trees, n_bootstrap, n_features, dt_max_depth):
    forest = []
    for i in range(n_trees):
        df_bootstrapped = train_df.sample(n=n_bootstrap, replace=True)
            tree = reg_decision_tree_algorithm(df_bootstrapped, max_depth=dt_max_depth)
            forest.append(tree)
    return forest

def calculate_accuracy_from_array(predictions, actual):
    correct_count = sum(p == a for p, a in zip(predictions, actual))
    total_count = len(actual)
    accuracy = correct_count / total_count
    return accuracy
```

Build Part - rdf

```
def build_rdf(X, y, attribute_types, N = 1, options = {"max_depth": 13, "min_samples": 2, "n
    tree = reg_decision_tree_algorithm(X, max_depth=options["max_depth"], min_samples=options[
    return tree
```

Predict Part - rdf

```
def predict_rdf(rdf, X, options):
    accuracy = calculate_accuracy_from_array(rdf, X)
    return accuracy
```

Random Forest Method Calling

```
options = {
    "max_depth": 90,
    "min_samples": 2,
    "max_features": None,
}

tree = build_rdf(X, y["Rings"], None, 1, options=options)

accuracy = predict_rdf(tree, X, options)
print(f"Accuracy: {accuracy}")
```

Accuracy: [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]

4/30/24, 8:46 PM