Imports

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import random
from pprint import pprint
from sklearn import metrics
from sklearn.preprocessing import OneHotEncoder
```

Install Dataset

```
pip install ucimlrepo
```

```
Requirement already satisfied: ucimlrepo in /usr/local/lib/python3.10/dist-packages (0.0
```

```
from ucimlrepo import fetch_ucirepo

# fetch dataset
abalone = fetch_ucirepo(id=1)

# data (as pandas dataframes)
X = abalone.data.features
y = abalone.data.targets

# metadata
print(abalone.metadata)

# variable information
print(abalone.variables)
```

```
{'uci_id': 1, 'name': 'Abalone', 'repository_url': 'https://archive.ics.uci.edu/dataset/
             name     role         type demographic  \
0             Sex  Feature  Categorical        None
1          Length  Feature   Continuous        None
2        Diameter  Feature   Continuous        None
3          Height  Feature   Continuous        None
4    Whole_weight  Feature   Continuous        None
5  Shucked_weight  Feature   Continuous        None
6  Viscera_weight  Feature   Continuous        None
7    Shell_weight  Feature   Continuous        None
8           Rings   Target      Integer        None
```

```
                       description  units missing_values
0         M, F, and I (infant)   None             no
1     Longest shell measurement     mm             no
2       perpendicular to length     mm             no
3            with meat in shell     mm             no
4                whole abalone  grams             no
5               weight of meat  grams             no
6   gut weight (after bleeding)  grams             no
7            after being dried  grams             no
8  +1.5 gives the age in years   None             no
```

X - EDA - Graphs

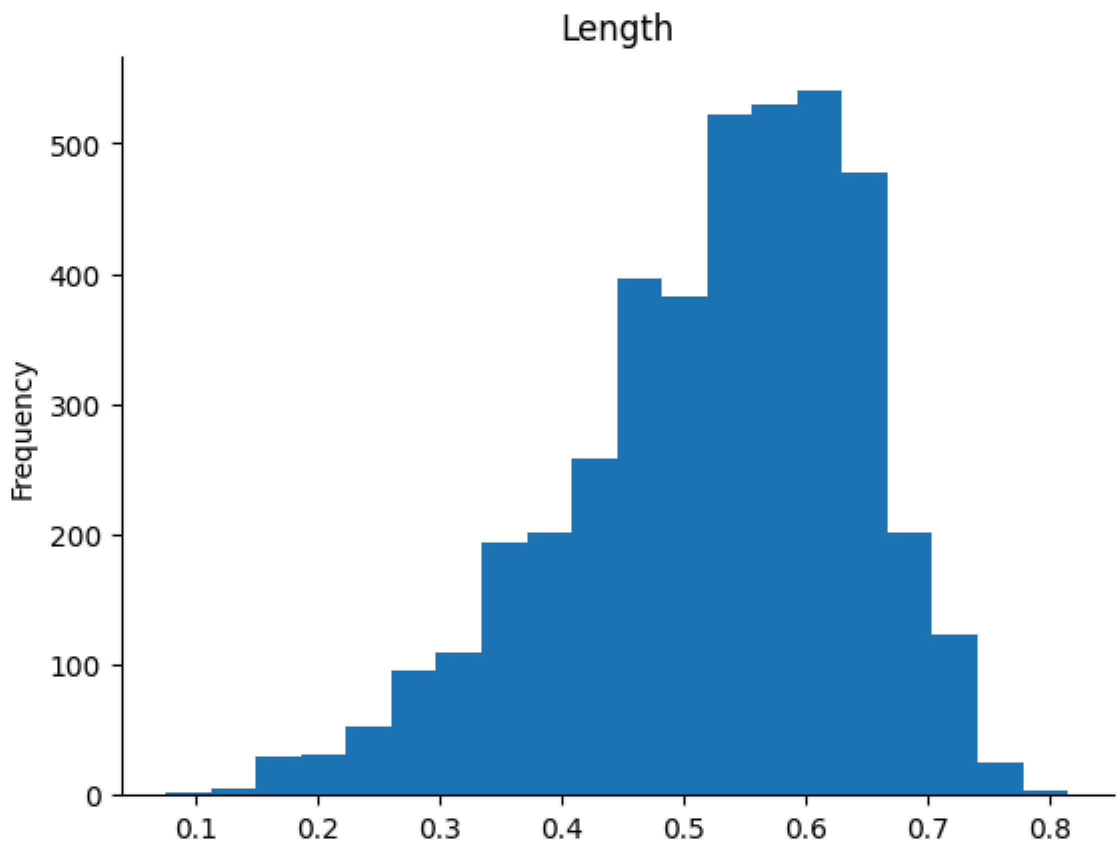
```
X.head()
```

	Sex	Length	Diameter	Height	Whole_weight	Shucked_weight	Viscera_weight	Shell_we
0	M	0.455	0.365	0.095	0.5140	0.2245	0.1010	(
1	M	0.350	0.265	0.090	0.2255	0.0995	0.0485	(
2	F	0.530	0.420	0.135	0.6770	0.2565	0.1415	(
3	M	0.440	0.365	0.125	0.5160	0.2155	0.1140	(
4	I	0.330	0.255	0.080	0.2050	0.0895	0.0395	(

Next steps: Generate code with X View recommended plots

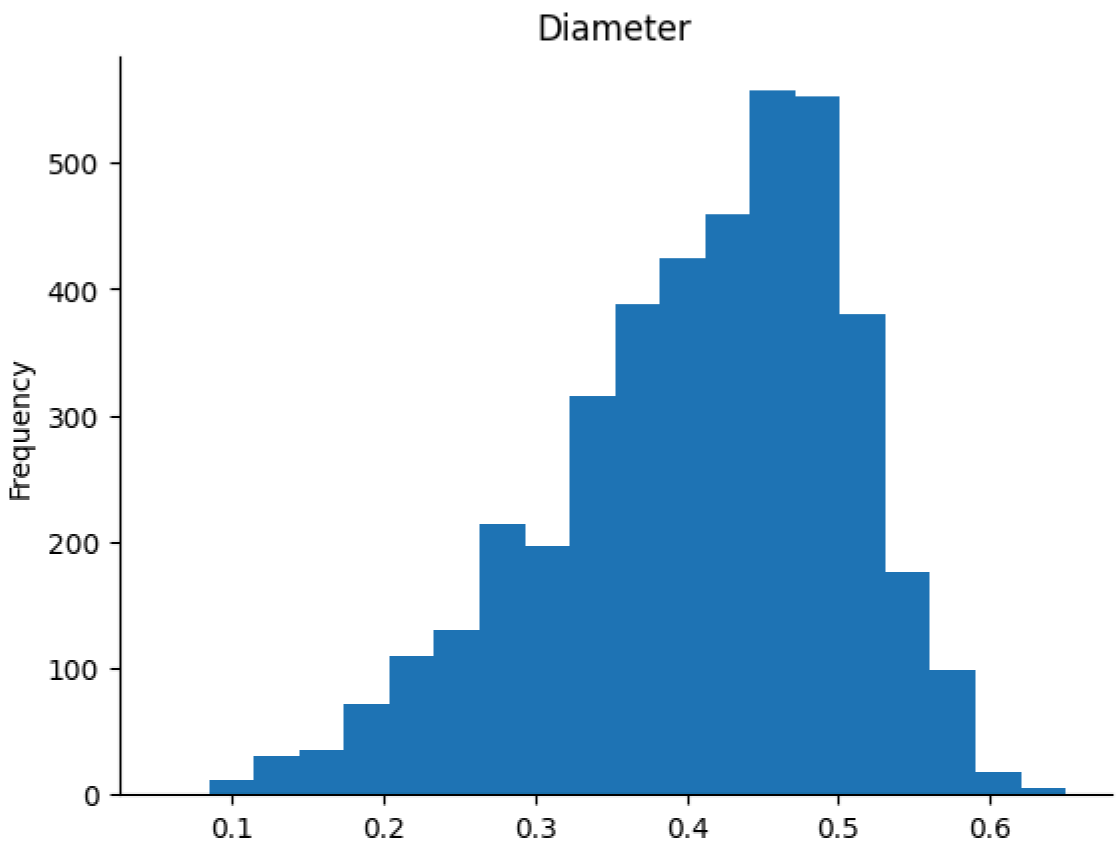
Length

```
# @title Length

from matplotlib import pyplot as plt
X['Length'].plot(kind='hist', bins=20, title='Length')
plt.gca().spines[['top', 'right',]].set_visible(False)
```



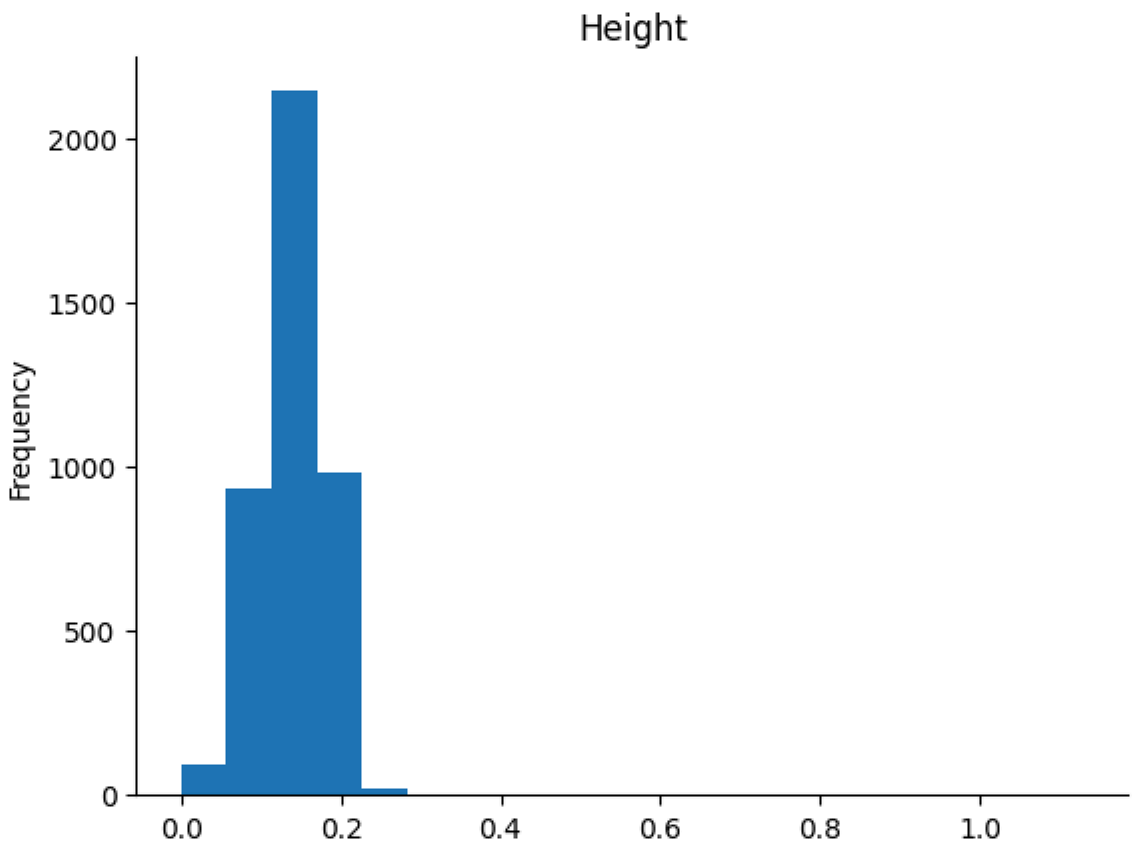
Diameter

```
# @title Diameter

from matplotlib import pyplot as plt
X['Diameter'].plot(kind='hist', bins=20, title='Diameter')
plt.gca().spines[['top', 'right',]].set_visible(False)
```



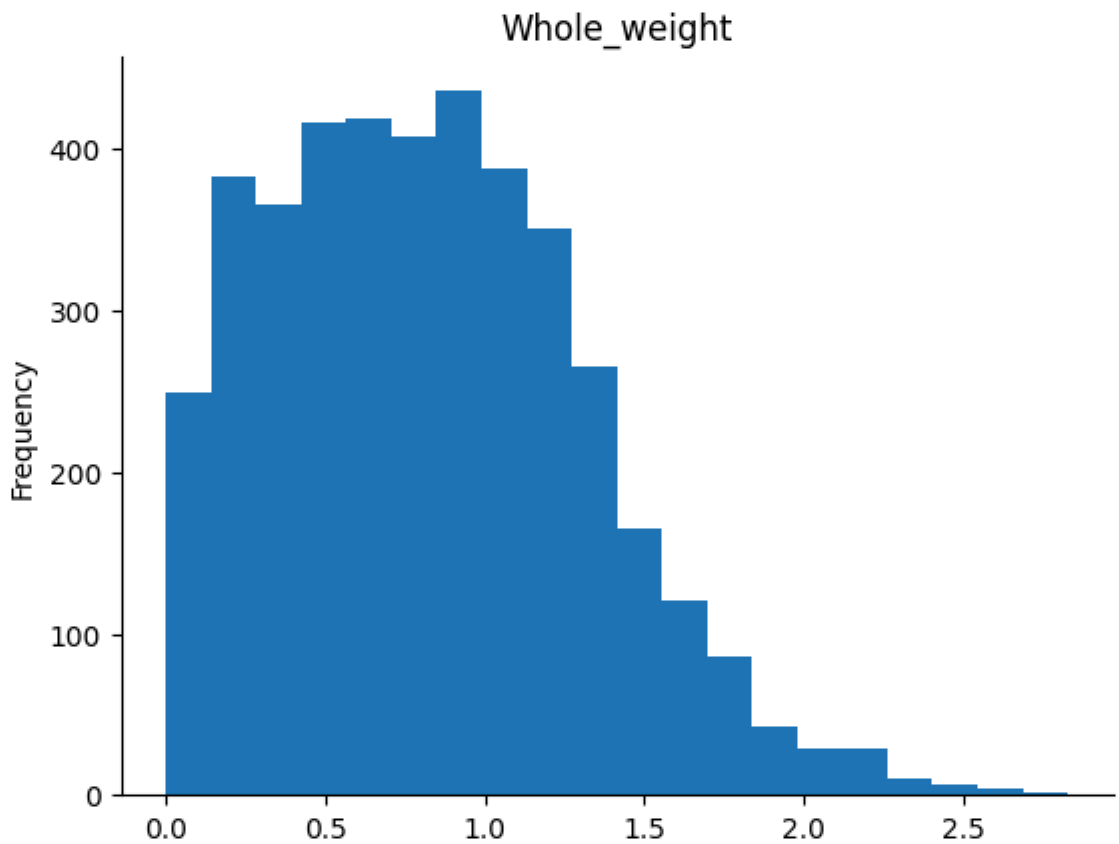
Height

```
# @title Height

from matplotlib import pyplot as plt
X['Height'].plot(kind='hist', bins=20, title='Height')
plt.gca().spines[['top', 'right',]].set_visible(False)
```



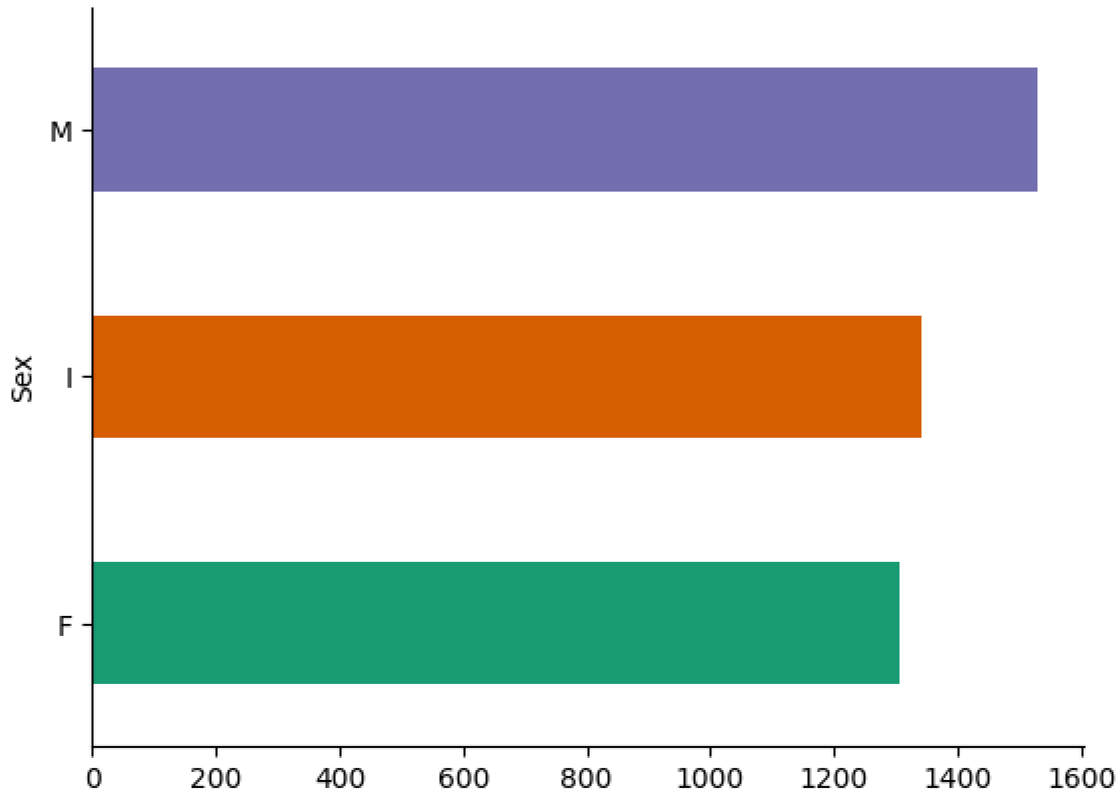
Whole_weight

```
# @title Whole_weight

from matplotlib import pyplot as plt
X['Whole_weight'].plot(kind='hist', bins=20, title='Whole_weight')
plt.gca().spines[['top', 'right',]].set_visible(False)
```



Sex

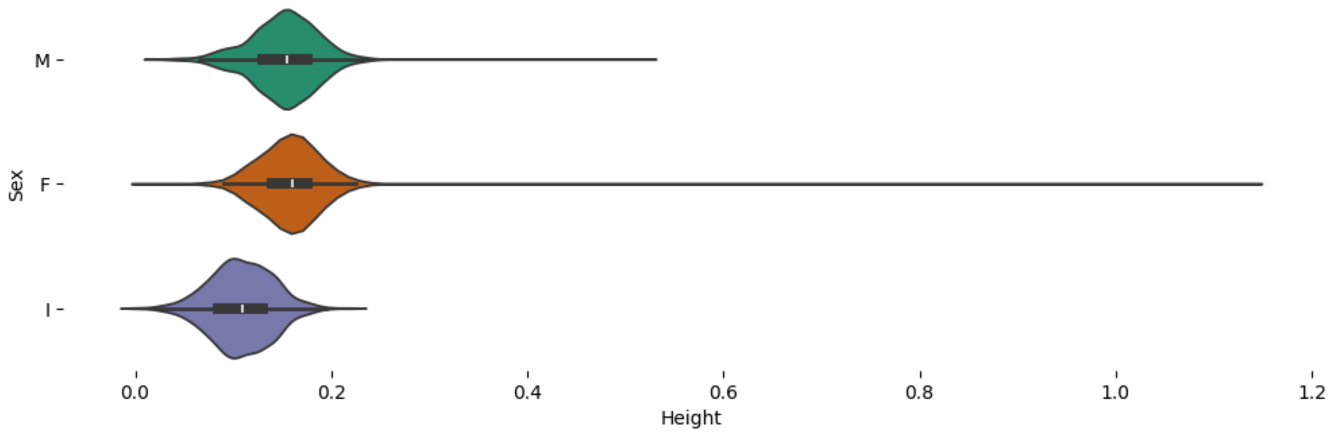
```
# @title Sex

from matplotlib import pyplot as plt
import seaborn as sns
X.groupby('Sex').size().plot(kind='barh', color=sns.palettes.mpl_palette('Dark2'))
plt.gca().spines[['top', 'right',]].set_visible(False)
```



Sex vs Height

```
# @title Sex vs Height

from matplotlib import pyplot as plt
import seaborn as sns
figsize = (12, 1.2 * len(X['Sex'].unique()))
plt.figure(figsize=figsize)
sns.violinplot(X, x='Height', y='Sex', inner='box', palette='Dark2')
sns.despine(top=True, right=True, bottom=True, left=True)
```

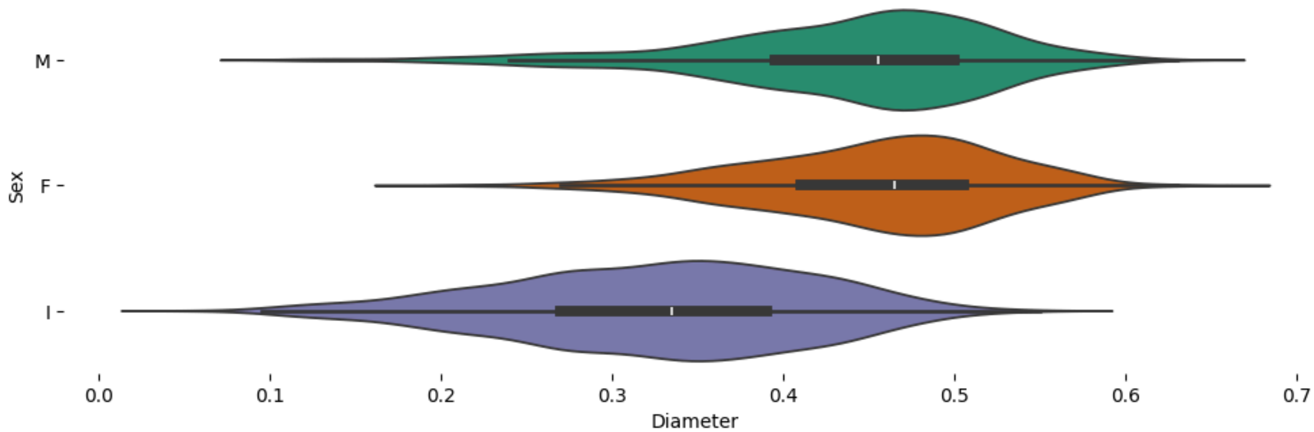
```
<ipython-input-295-7d81624abca8>:7: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0.

  sns.violinplot(X, x='Height', y='Sex', inner='box', palette='Dark2')
```



Sex vs Diameter

```
# @title Sex vs Diameter

from matplotlib import pyplot as plt
import seaborn as sns
figsize = (12, 1.2 * len(X['Sex'].unique()))
plt.figure(figsize=figsize)
sns.violinplot(X, x='Diameter', y='Sex', inner='box', palette='Dark2')
sns.despine(top=True, right=True, bottom=True, left=True)
```

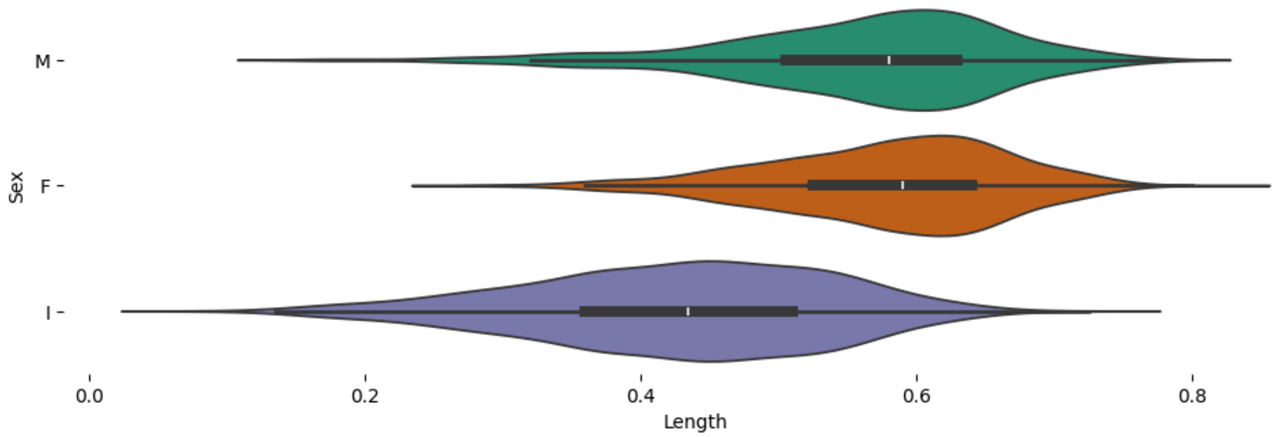
```
<ipython-input-296-9bc23b34b992>:7: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0.

  sns.violinplot(X, x='Diameter', y='Sex', inner='box', palette='Dark2')
```



Sex vs Length

```
# @title Sex vs Length

from matplotlib import pyplot as plt
import seaborn as sns
figsize = (12, 1.2 * len(X['Sex'].unique()))
plt.figure(figsize=figsize)
sns.violinplot(X, x='Length', y='Sex', inner='box', palette='Dark2')
sns.despine(top=True, right=True, bottom=True, left=True)
```

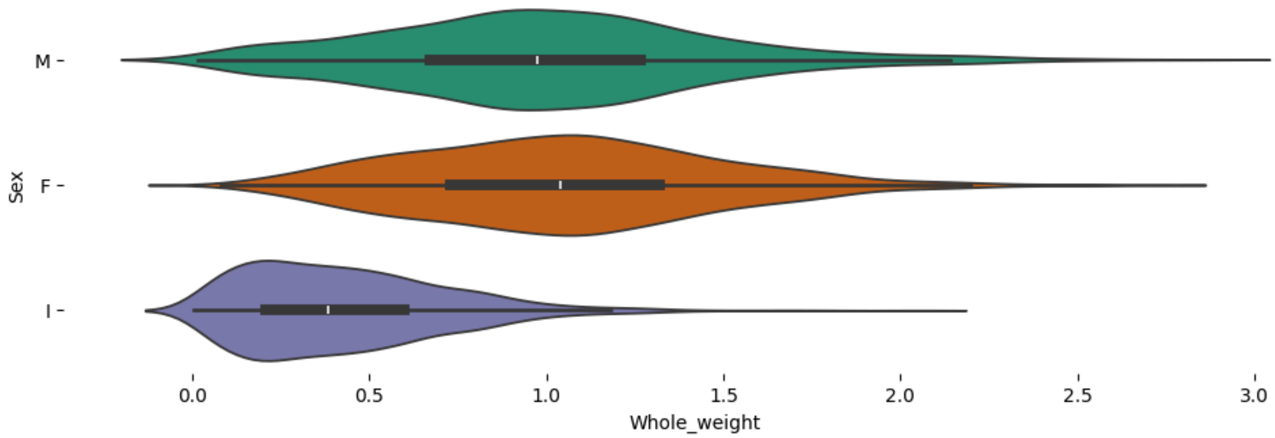
```
<ipython-input-297-954774488379>:7: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0.

  sns.violinplot(X, x='Length', y='Sex', inner='box', palette='Dark2')
```



Sex vs Whole_weight

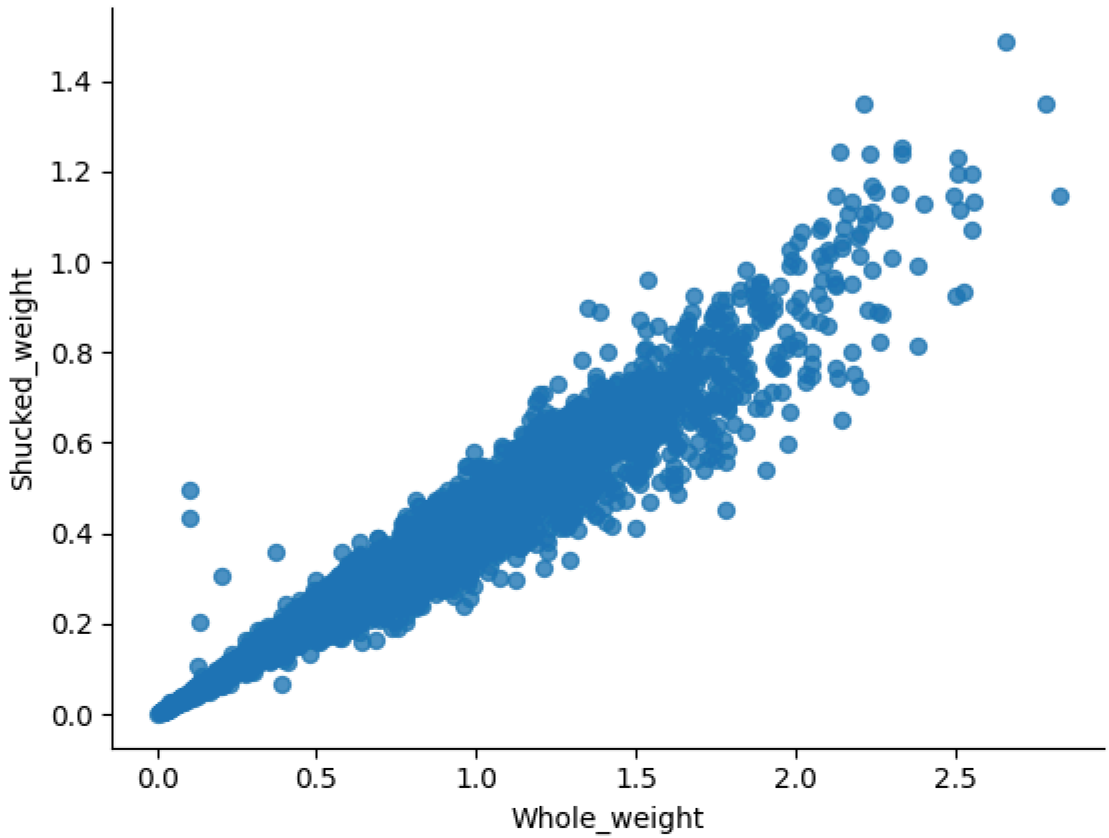
```
# @title Sex vs Whole_weight

from matplotlib import pyplot as plt
import seaborn as sns
figsize = (12, 1.2 * len(X['Sex'].unique()))
plt.figure(figsize=figsize)
sns.violinplot(X, x='Whole_weight', y='Sex', inner='box', palette='Dark2')
sns.despine(top=True, right=True, bottom=True, left=True)
```

```
<ipython-input-298-c718f362d9f5>:7: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0.

  sns.violinplot(X, x='Whole_weight', y='Sex', inner='box', palette='Dark2')
```



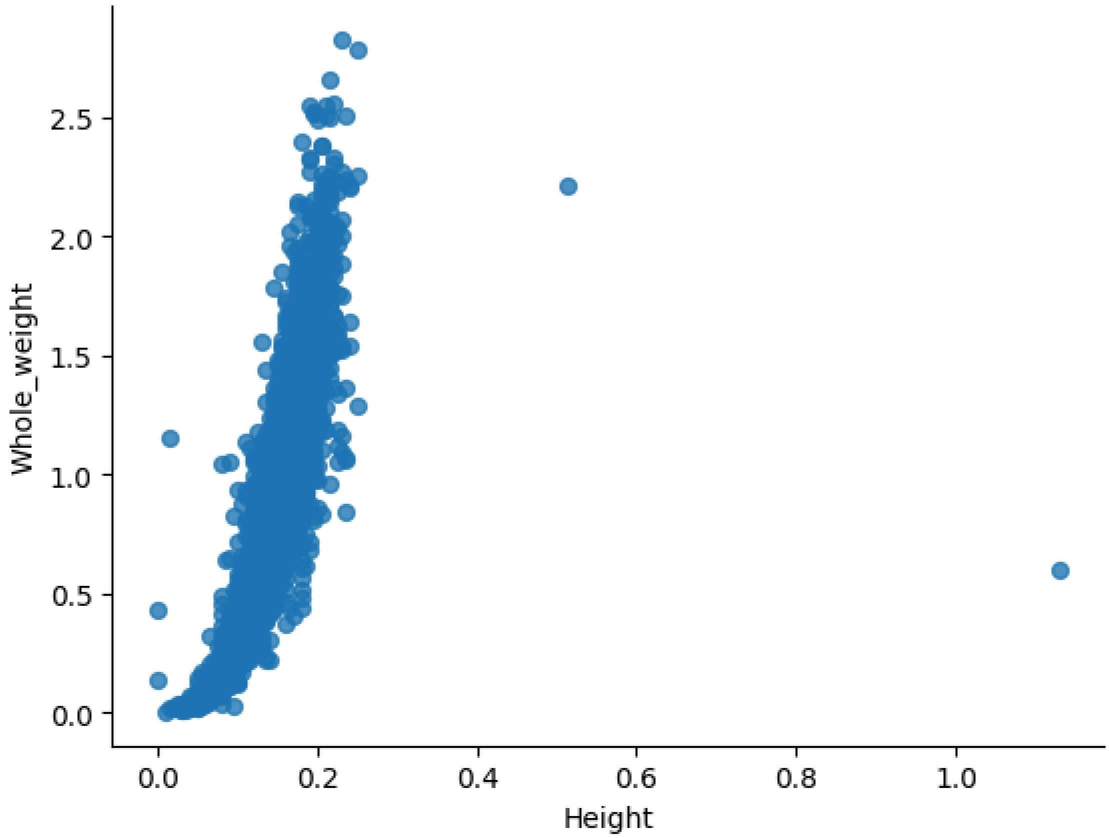
Whole_weight vs Shucked_weight

```
# @title Whole_weight vs Shucked_weight

from matplotlib import pyplot as plt
X.plot(kind='scatter', x='Whole_weight', y='Shucked_weight', s=32, alpha=.8)
plt.gca().spines[['top', 'right',]].set_visible(False)
```



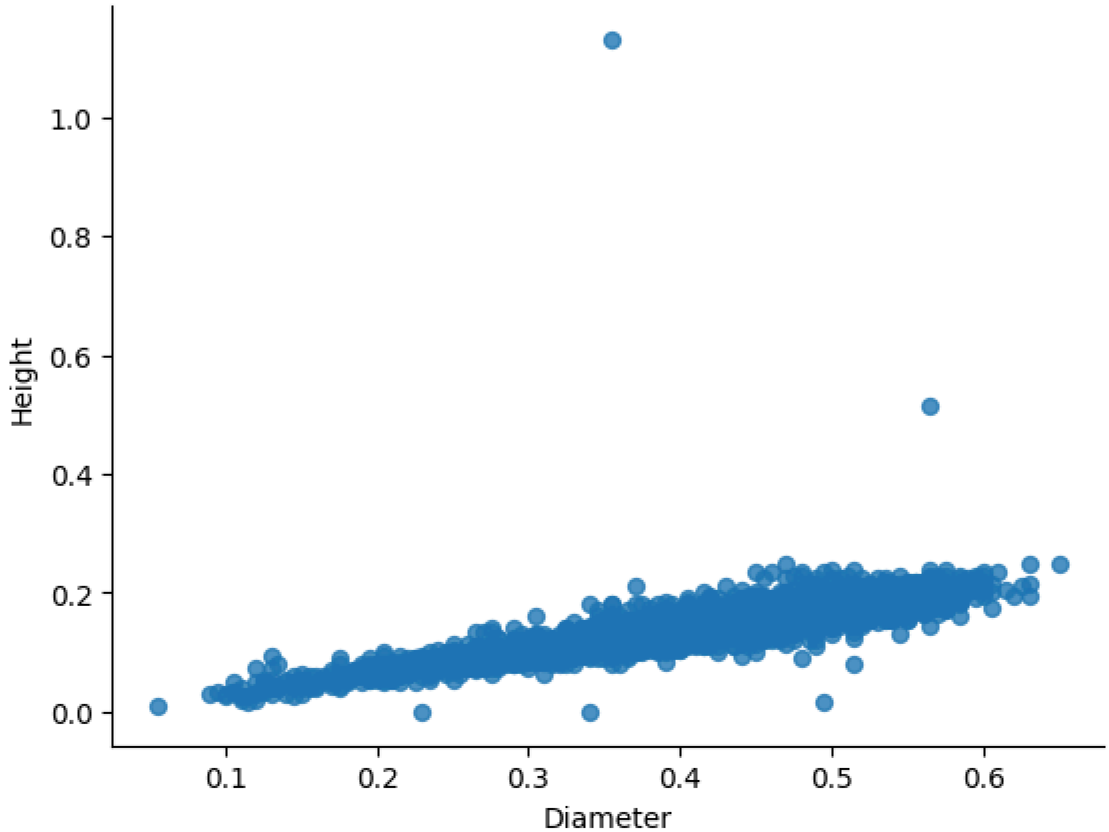
Height vs Whole_weight

```
# @title Height vs Whole_weight

from matplotlib import pyplot as plt
X.plot(kind='scatter', x='Height', y='Whole_weight', s=32, alpha=.8)
plt.gca().spines[['top', 'right',]].set_visible(False)
```



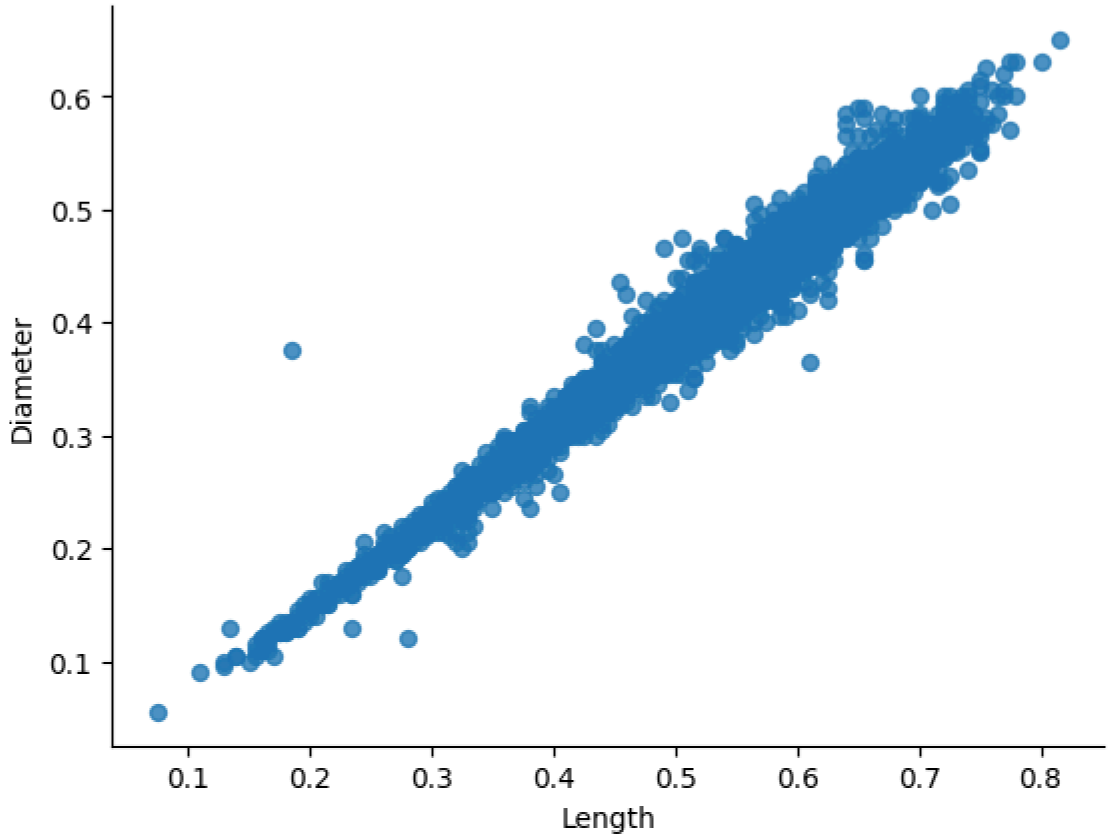
Diameter vs Height

```
# @title Diameter vs Height

from matplotlib import pyplot as plt
X.plot(kind='scatter', x='Diameter', y='Height', s=32, alpha=.8)
plt.gca().spines[['top', 'right',]].set_visible(False)
```



Length vs Diameter

```
# @title Length vs Diameter

from matplotlib import pyplot as plt
X.plot(kind='scatter', x='Length', y='Diameter', s=32, alpha=.8)
plt.gca().spines[['top', 'right',]].set_visible(False)
```



Y - EDA - Graphs

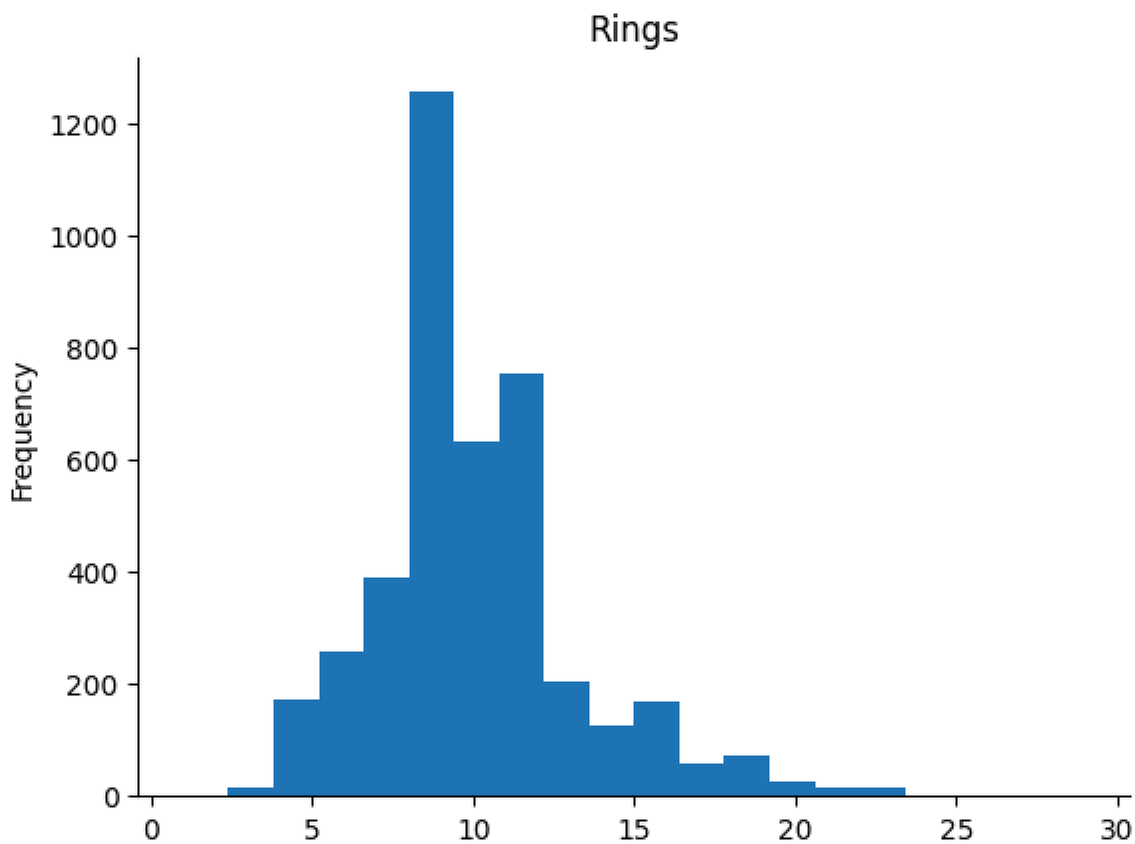
```
y.head()
```

	Rings
0	15
1	7
2	9
3	10
4	7

Next steps: Generate code with y View recommended plots

Rings

```
# @title Rings

from matplotlib import pyplot as plt
y['Rings'].plot(kind='hist', bins=20, title='Rings')
plt.gca().spines[['top', 'right',]].set_visible(False)
```



X - EDA

```
X.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4177 entries, 0 to 4176
Data columns (total 8 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   Sex             4177 non-null   object 
 1   Length          4177 non-null   float64
 2   Diameter        4177 non-null   float64
 3   Height          4177 non-null   float64
 4   Whole_weight    4177 non-null   float64
 5   Shucked_weight  4177 non-null   float64
 6   Viscera_weight  4177 non-null   float64
 7   Shell_weight    4177 non-null   float64
dtypes: float64(7), object(1)
memory usage: 261.2+ KB
```

```
X.describe()
```

	Length	Diameter	Height	Whole_weight	Shucked_weight	Viscera_weigh
count	4177.000000	4177.000000	4177.000000	4177.000000	4177.000000	4177.00000
mean	0.523992	0.407881	0.139516	0.828742	0.359367	0.18059
std	0.120093	0.099240	0.041827	0.490389	0.221963	0.10961
min	0.075000	0.055000	0.000000	0.002000	0.001000	0.00050
25%	0.450000	0.350000	0.115000	0.441500	0.186000	0.09350
50%	0.545000	0.425000	0.140000	0.799500	0.336000	0.17100
75%	0.615000	0.480000	0.165000	1.153000	0.502000	0.25300
max	0.815000	0.650000	1.130000	2.825500	1.488000	0.76000

```
X = X.fillna(value=np.nan)
missing_values = X.isna().sum()
print(missing_values)
```

```
Sex               0
Length            0
Diameter          0
Height            0
Whole_weight      0
Shucked_weight    0
Viscera_weight    0
Shell_weight      0
dtype: int64
```

```
X.nunique()
```

```
Sex                  3
Length             134
Diameter           111
Height              51
Whole_weight      2429
Shucked_weight    1515
Viscera_weight     880
Shell_weight       926
dtype: int64
```

```
encoder = OneHotEncoder(sparse_output=False)
encoded_sex = encoder.fit_transform(X[['Sex']])

encoded_sex_df = pd.DataFrame(encoded_sex, columns=encoder.get_feature_names_out(['Sex']))

X = X.drop('Sex', axis=1)

X = pd.concat([X, encoded_sex_df], axis=1)
```

```
X.head()
```

	Length	Diameter	Height	Whole_weight	Shucked_weight	Viscera_weight	Shell_weight
0	0.455	0.365	0.095	0.5140	0.2245	0.1010	0.150
1	0.350	0.265	0.090	0.2255	0.0995	0.0485	0.070
2	0.530	0.420	0.135	0.6770	0.2565	0.1415	0.210
3	0.440	0.365	0.125	0.5160	0.2155	0.1140	0.155
4	0.330	0.255	0.080	0.2050	0.0895	0.0395	0.055

Next steps: Generate code with X View recommended plots

```
index_of_shell_weight = X.columns.get_loc('Shell_weight')
print(index_of_shell_weight)
```

```
6
```

```
X[X["Shell_weight"].isnull()]
```

Length	Diameter	Height	Whole_weight	Shucked_weight	Viscera_weight	Shell_weight

```
X["Shell_weight"].fillna(X["Shell_weight"].mean(), inplace=True)
```

Y - EDA

```
y.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4177 entries, 0 to 4176
Data columns (total 1 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
```

```
 0   Rings   4177 non-null   int64
dtypes: int64(1)
memory usage: 32.8 KB
```

```
y.describe()
```

	Rings
count	4177.000000
mean	9.933684
std	3.224169
min	1.000000
25%	8.000000
50%	9.000000
75%	11.000000
max	29.000000

```
y = y.fillna(value=np.nan)
missing_values = y.isna().sum()
print(missing_values)
```

```
Rings    0
dtype: int64
```

```
y.nunique()
```

```
Rings    28
dtype: int64
```

Decision Tree

```
def classify_example(example, tree):
    question = list(tree.keys())[0]
    feature_name, comparison_operator, value = question.split()

    # condition check
    if example[feature_name] <= float(value):
        answer = tree[question][0]
    else:
        answer = tree[question][1]

    if not isinstance(answer, dict):
        return answer

    # iter branches
    else:
        residual_tree = answer
        return classify_example(example, residual_tree)


def calculate_accuracy(df, tree):

    df["classification"] = df.apply(classify_example, axis=1, args=(tree,))
    df["classification_correct"] = df["classification"] == y["Rings"]

    accuracy = df["classification_correct"].mean()

    return accuracy


def check_purity(data):

    label_column = data[:, -1]
    unique_classes = np.unique(label_column)

    if len(unique_classes) == 1:
        return True
    else:
        return False


def classify_data(data):

    label_column = data[:, -1]
    unique_classes, counts_unique_classes = np.unique(label_column, return_counts=True)

    index = counts_unique_classes.argmax()
    classification = unique_classes[index]

    return classification
```

```
def get_potential_splits(data):
    potential_splits = {}
    _, n_columns = data.shape
    for column_index in range(n_columns - 1):
        potential_splits[column_index] = []
        values = data[:, column_index]

        if np.issubdtype(values.dtype, np.number):
            unique_values = np.unique(values)
            for index in range(len(unique_values)):
                if index != 0:
                    current_value = unique_values[index]
                    previous_value = unique_values[index - 1]
                    potential_split = (current_value + previous_value) / 2
                    potential_splits[column_index].append(potential_split)

    return potential_splits


def split_data(data, split_column, split_value):
    if isinstance(data, pd.DataFrame):
        data_array = data.values
    else:
        data_array = data

    split_column_values = data_array[:, split_column]

    if split_value is None:
        return data_array, None

    data_below = data_array[np.where(split_column_values <= split_value)]
    data_above = data_array[np.where(split_column_values > split_value)]

    return data_below, data_above



def calculate_entropy(data):
    label_column = data[:, -1]
    _, counts = np.unique(label_column, return_counts=True)

    probabilities = counts / counts.sum()
    entropy = sum(probabilities * -np.log2(probabilities))

    return entropy
```

```
def calculate_overall_entropy(data_below, data_above, metric_function):
    n = len(data_below) + len(data_above)
    p_data_below = len(data_below) / n
    p_data_above = len(data_above) / n

    overall_entropy =  (p_data_below * metric_function(data_below)
                      + p_data_above * metric_function(data_above))

    return overall_entropy


def determine_best_split(data, potential_splits):
    best_overall_entropy = float('inf')
    best_split_column = None
    best_split_value = None

    for column_index in potential_splits:
        for value in potential_splits[column_index]:
            data_below, data_above = split_data(data, split_column=column_index, split_value

            current_overall_entropy = calculate_overall_entropy(data_below, data_above, metr

            if current_overall_entropy <= best_overall_entropy:
                best_overall_entropy = current_overall_entropy
                best_split_column = column_index
                best_split_value = value

    return best_split_column, best_split_value
```

```
def decision_tree_algorithm(df, counter=0, min_samples=10, max_depth=15):
    if counter == 0:
        global COLUMN_HEADERS
        COLUMN_HEADERS = df.columns
        data = df.values
    else:
        data = df

    if (check_purity(data)) or (len(data) < min_samples) or (counter == max_depth):
        classification = classify_data(data)

        return classification

    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 = COLUMN_HEADERS[split_column]
        question = "{} <= {}".format(feature_name, split_value)
        sub_tree = {question: []}

        yes_answer = decision_tree_algorithm(data_below, counter, min_samples, max_depth)
        no_answer = 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
```

Build Part - dt

```
def build_dt(X, y, attribute_types, options):
  df = pd.concat([X, y], axis=1)

  df = df.dropna()

  tree = decision_tree_algorithm(df, max_depth=options['max_depth'], min_samples=options['mi
  return tree
```

Predict Part - dt

```
def predict_dt(dt, X, options):
  accuracy = calculate_accuracy(X, dt)
  return accuracy
```

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)
```

```
{'Wh
```