In [ ]:
```
import math
import json
import numpy as np
import pandas as pd
```

In [ ]:
```
# Read dataset
glove_df_train = pd.read_csv("../project_data/data/glove/glove.train.csv")
glove_df_test = pd.read_csv("../project_data/data/glove/glove.test.csv")
glove_df_eval = pd.read_csv("../project_data/data/glove/glove.eval.anon.csv"

misc_df_train = pd.read_csv("../project_data/data/misc/misc-attributes-train
misc_df_test = pd.read_csv("../project_data/data/misc/misc-attributes-test.c
misc_df_eval = pd.read_csv("../project_data/data/misc/misc-attributes-eval.c

# Add label to misc attribute
misc_df_train["label"] = glove_df_train["label"]
misc_df_test["label"] = glove_df_test["label"]
misc_df_eval["label"] = glove_df_eval["label"]
```

In [ ]:
```
# Print all unique values of the feature
def print_unique_values_of_feature(df):
    for column in df.columns:
        unique_values = df[column].unique()
        print(f"Column '{column}' #{len(unique_values)}: {unique_values}")

print_unique_values_of_feature(misc_df_train)
```

```
Column 'defendant_age' #103: ['not known' '19' '17' '29' '25' '27' '23' '30'
'58' '37' '11' '32' '21'
 '38' '33' '22' '62' '49' '34' '18' '44' '46' '51' '16' '68' '40' '26'
 '42' '35' '41' '31' '65' '20' '70' '24' 'Nineteen' '36' '61' '14' '28'
 'seventeen' '45' '10' '52' '47' '50' '78' '15' '48' '13'
 'twelve Years of Age' '12' '54' '53' '66' 'sixteen' '43' '60' '67' '39'
 '64' '55' '73' '57' '59' '71' '56' 'eighteen' '63' ' (46)' '9' '84' '96'
 '69' 'nineteen' 'thirteen' 'seven' 'fourteen'
 'not quite thirteen years old' '72' '75' '13 years' '8' '74' 'ten' '79'
 'I am going into the sixteenth Year of my Age' '85' '22a' '82' '83'
 'Nine' 'about sixteen years of age' 'Thirteen' '76' 'Fifteen' 'eleven'
 '24 years of age' 'fourteen years old' 'shop-foreman'
 'sixteen years of age' 'thirty' 'fifteen years of age']
Column 'defendant_gender' #3: ['male' 'female' 'indeterminate']
Column 'num_victims' #13: [ 1  0  2  3  4  5  6 10  7  9 11 12 13]
Column 'victim_genders' #64: ['male' nan 'female' 'indeterminate' 'male;mal
e' 'male;male;male'
 'male;female' 'male;male;female' 'indeterminate;male'
 'male;female;female' 'female;male' 'female;female;female;female'
 'male;male;indeterminate;male' 'female;female' 'male;male;male;male'
 'male;male;male;male;male' 'female;male;male' 'female;female;female'
 'male;female;male' 'male;indeterminate;male'
 'male;male;male;female;male;female' 'male;male;male;male;male;male'
 'male;indeterminate' 'male;male;male;indeterminate'
 'male;male;male;male;male;male;male;male;male;male'
```

```
 'female;indeterminate' 'male;male;male;female;male;female;male'
 'female;male;female' 'indeterminate;indeterminate'
 'female;female;female;male' 'indeterminate;male;male'
 'male;male;male;indeterminate;male' 'indeterminate;female'
 'female;female;female;female;female' 'male;male;male;male;male;male;male'
 'female;female;male' 'male;indeterminate;indeterminate'
 'indeterminate;female;male' 'indeterminate;male;female'
 'male;male;male;male;male;male;male;male;male'
 'male;male;indeterminate;male;indeterminate;male;indeterminate;male;male'
 'male;male;indeterminate' 'male;male;female;female'
 'female;male;male;female' 'male;female;male;male'
 'male;male;male;male;indeterminate' 'female;female;indeterminate'
 'female;female;male;male'
 'male;indeterminate;indeterminate;indeterminate;indeterminate;male;indeterm
inate;indeterminate;male;male;male'
 'male;male;female;male' 'male;male;male;female' 'male;female;female;male'
 'female;male;male;male' 'indeterminate;male;male;male'
 'male;male;male;male;female'
 'male;male;male;male;male;male;male;male;male;male;male;male;male'
 'female;female;male;female;female;female;male;female;male;female;female;fem
ale;male'
 'indeterminate;indeterminate;male'
 'indeterminate;male;male;male;male;male;male'
 'female;indeterminate;female;male;male' 'male;female;male;male;male'
 'female;female;male;female;male' 'female;male;female;female;male;male'
 'female;male;female;male']
Column 'offence_category' #9: ['theft' 'kill' 'breakingPeace' 'deception' 's
exual' 'violentTheft'
 'royalOffences' 'miscellaneous' 'damage']
Column 'offence_subcategory' #52: ['simpleLarceny' 'grandLarceny' 'manslaugh
ter' 'riot' 'perjury' 'burglary'
 'animalTheft' 'keepingABrothel' 'embezzlement' 'wounding' 'pocketpicking'
 'stealingFromMaster' 'libel' 'highwayRobbery' 'theftFromPlace' 'robbery'
 'mail' 'coiningOffences' 'forgery' 'fraud' 'receiving' 'shoplifting'
 'rape' 'returnFromTransportation' 'assault' 'bigamy' 'arson' 'other'
 'sodomy' 'murder' 'bankrupcy' 'concealingABirth' 'housebreaking'
 'infanticide' 'assaultWithIntent' 'kidnapping' 'indecentAssault'
 'pettyLarceny' 'illegalAbortion' 'assaultWithSodomiticalIntent'
 'threateningBehaviour' 'extortion' 'taxOffences' 'pervertingJustice'
 'conspiracy' 'seditiousWords' 'seducingAllegiance' 'seditiousLibel'
 'treason' 'religiousOffences' 'gameLawOffence' 'pettyTreason']
Column 'label' #2: [1 0]
```

In [ ]:

```python
# Preprocess dataset.
defendant_age_replace_dict = {
    "nineteen": "19",
    "Nineteen": "19",
    "sixteen": "16",
    "seven": "7",
    "eighteen": "18",
    "seventeen": "17",
    "thirteen": "13",
    "Thirteen": "13",
    "not quite thirteen years old": "13",
```

```
    "Nine": "9",
    "I am going into the sixteenth Year of my Age": "16",
    "eleven": "11",
    "thirty": "30",
    "sixteen years of age": "16",
    "fourteen years old": "14",
    "Fifteen": "15",
    "fifteen years of age": "15",
    "about sixteen years of age": "16",
    "13 years": "13",
    " (46)": "46",
    "fourteen": "14",
    "ten": "10",
    "twelve Years of Age": "12",
    "24 years of age": "24",
}


def data_pre_process(df):
    df["defendant_age"] = df["defendant_age"].replace(defendant_age_replace_

    # Convert int64 columns to str
    df["num_victims"] = df["num_victims"].astype(str)
    df["label"] = df["label"].astype(str)

    # df = df.drop(columns=["num_victims"])

    return df


misc_df_train = data_pre_process(misc_df_train)
misc_df_test = data_pre_process(misc_df_test)
misc_df_eval = data_pre_process(misc_df_eval)

print(f"Column 'defendant_age': {misc_df_train['defendant_age'].unique()}")
print("\nAll columns datatypes: ", misc_df_eval.dtypes)
```

```
Column 'defendant_age': ['not known' '19' '17' '29' '25' '27' '23' '30' '58'
'37' '11' '32' '21'
 '38' '33' '22' '62' '49' '34' '18' '44' '46' '51' '16' '68' '40' '26'
 '42' '35' '41' '31' '65' '20' '70' '24' '36' '61' '14' '28' '45' '10'
 '52' '47' '50' '78' '15' '48' '13' '12' '54' '53' '66' '43' '60' '67'
 '39' '64' '55' '73' '57' '59' '71' '56' '63' '9' '84' '96' '69' '7' '72'
 '75' '8' '74' '79' '85' '22a' '82' '83' '76' 'shop-foreman']

All columns datatypes:  defendant_age          object
defendant_gender       object
num_victims            object
victim_genders         object
offence_category       object
offence_subcategory    object
label                  object
dtype: object
```

In [ ]:

```
def tree_walk(row, tree):
    if "label" in tree:
        # print()
        return tree["label"]

    for key in tree.keys():
        new_key = row[key]
        # print(f"key: {key} -> new_key: {new_key}", end="  ")
        if new_key not in tree[key]:
            # print(f"for key: {key} new_key: {new_key} not in tree.")
            return "NoPath"

        return tree_walk(row, tree[key][new_key])


def test_accuracy(df, tree, store_eval=False):
    df_rows = df.shape[0]
    dict_rows = df.to_dict(orient="records")
    eval_list = []

    if df_rows != len(dict_rows):
        print(f"Error: Mismatch in data frame rows ({df_rows}) and dictionar
        raise ValueError

    correct_prediction = 0
    total_samples = len(dict_rows)
    # print("Total Samples: ", total_samples)

    for index, row in df.iterrows():
        predicted_label = tree_walk(row=dict_rows[index], tree=tree)

        # When there is no path in the Tree take the majority label.
        # Decided to go with this because model needs to predict when it see
        if predicted_label == "NoPath":
            predicted_label = get_majority_label(df)

        if store_eval:
            eval_list.append(predicted_label)

        if row["label"] == predicted_label:
            correct_prediction += 1

    # print("Accuracy: ", correct_prediction / total_samples)
    return correct_prediction / total_samples, eval_list


def get_majority_label(df, p_label="1", n_label="0", label_col_name="label")
    positive_count = df[label_col_name].value_counts()[p_label] if p_label i
    negative_count = df[label_col_name].value_counts()[n_label] if n_label i

    # print(f"positive_count: {positive_count}, negative_count: {negative_cc
    if positive_count > negative_count:
        return "1"
```

```
    else:
        return "0"


def get_max_key_by_value(map):
    max_key = ""
    max_val = float("-inf")

    for key, val in map.items():
        if val > max_val:
            max_val = val
            max_key = key

    # print("map: ", map, "max_key: ", max_key)
    return max_key


def get_data_frame_subset(df, attribute=None, attribute_value=None):
    if not attribute:
        print(f"Error: No attribute: {attribute} and it's attribute_value: {
        return None

    df = df[df[attribute] == attribute_value]  # Filter rows with value equa
    df = df.loc[:, df.columns != attribute]  # Remove the attribute column
    return df


def calculate_binary_entropy(pTrue=None, pFalse=None):
    try:
        if pTrue is None or pFalse is None:
            raise AttributeError

        if pTrue == 0.0 or pFalse == 0.0:
            return 0

        return -pTrue * math.log2(pTrue) - pFalse * math.log2(pFalse)

    except Exception:
        print(f"Cannot calculate_binary_entropy for pTrue: {pTrue}, pFalse:


def get_entropy(df, p_label="1", n_label="0", label_col_name="label"):
    label_data = df[label_col_name]
    label_size = label_data.size

    # When sub df has no entries return entropy 0 ie no uncertainty.
    if label_size == 0:
        return 0

    # print("label_size", label_size)
    positive_count = df[label_col_name].value_counts()[p_label] if p_label i
    negative_count = df[label_col_name].value_counts()[n_label] if n_label i
```

```
    # print(f"# of p sample: {positive_count}\n# of n sample: {negative_cour
    p_positive = positive_count / label_size
    p_negative = negative_count / label_size
    # print(p_positive, p_negative)

    return calculate_binary_entropy(pTrue=p_positive, pFalse=p_negative)


def get_best_info_gain_attribute(df):
    total_entropy = get_entropy(df, p_label="1", n_label="0")
    total_samples = df.shape[0]
    attributes = df.columns

    attr_possible_values_dict = {}
    for attr in attributes:
        if attr != "label" and attr not in attr_possible_values_dict:
            attr_possible_values_dict[attr] = list(df[attr].unique())

    information_gain = {}
    for attr, attr_values in attr_possible_values_dict.items():
        if attr not in information_gain:
            information_gain[attr] = 0
            # if attr == "defendant_gender":
            #     information_gain[attr] = 0.00005

        gain = 0
        for attr_value in attr_values:
            sub_df = get_data_frame_subset(df, attribute=attr, attribute_val
            samples = sub_df.shape[0]

            entropy = get_entropy(sub_df, p_label="1", n_label="0")
            gain += (samples / total_samples) * entropy

        information_gain[attr] += total_entropy - gain

    best_attribute = get_max_key_by_value(information_gain)
    return best_attribute, information_gain[best_attribute]


def id3(df, max_depth, tree=None, depth=1):
    best_attribute, _ = get_best_info_gain_attribute(df)
    best_attribute_possible_values = list(df[best_attribute].unique())
    current_depth = depth
    if not tree:
        tree = {}

    if best_attribute not in tree:
        tree[best_attribute] = {}

    for value in best_attribute_possible_values:
        tree[best_attribute][value] = {}

        # Get the dataset with rows set to the attribute value and the attri
```

```
        sub_df = get_data_frame_subset(df, attribute=best_attribute, attribu
        labels = sub_df["label"].unique()
        if len(list(labels)) == 1:
            tree[best_attribute][value]["label"] = list(labels)[0]

        elif max_depth and depth >= max_depth:
            tree[best_attribute][value]["label"] = get_majority_label(sub_df

        else:
            # When sub df has only label column then no need split further.
            if len(sub_df.columns) != 1:
                sub_tree, sub_tree_depth = id3(sub_df, max_depth, tree=None,
                tree[best_attribute][value] = sub_tree
                current_depth = max(sub_tree_depth, current_depth)
            else:
                # print("Best Attribute:", best_attribute, " Value:", value,
                tree[best_attribute][value]["label"] = get_majority_label(su

    return tree, current_depth
```

In [ ]:
```
# Tree with no depth limit ie full tree.
print("Full Tree")

tree, depth = id3(df=misc_df_train, max_depth=None)
# print("Depth:", depth, "Tree:", tree)

train_acc, _ = test_accuracy(misc_df_train, tree)
test_acc, _ = test_accuracy(misc_df_test, tree)
eval_acc, prediction_list = test_accuracy(misc_df_eval, tree, store_eval=Tru

print(f"Accuracy of tree on train dataset: ", train_acc)
print(f"Accuracy of tree on test dataset: ", test_acc)
print(f"Accuracy of tree on eval dataset: ", eval_acc)

df = pd.DataFrame(prediction_list)
df.to_csv("decision_tree_misc_eval_dataset_prediction.csv", index=True, heac
```

```
Full Tree
Accuracy of tree on train dataset:  0.7982857142857143
Accuracy of tree on test dataset:  0.728
Accuracy of tree on eval dataset:  0.6723809523809524
```

In [ ]:
```
# Tree with limiting depth limit
print("Limiting Depth Tree")

trees_dict = {}
accuracy_dict = {}
depths = [6]
# depths = [1, 2, 3, 4, 5, 6, 7, 8, 10]
for depth in depths:
    tree, _ = id3(df=misc_df_train, max_depth=depth)
    trees_dict[depth] = tree

    accuracy, _ = test_accuracy(misc_df_test, tree)
```

```
    accuracy_dict[depth] = accuracy
    # print(f"Depth: {depth}, Test accuracy: {accuracy}, Tree: {tree}")

best_hyper_param = get_max_key_by_value(accuracy_dict)
print("\nBest hyper parameter (depth):", best_hyper_param)

export_tree = "best-hyper-param-decision-tree.json"
print(f"Exporting tree to '{export_tree}'.")
with open(export_tree, "w") as f:
    json.dump(trees_dict[best_hyper_param], f, indent=4, default=str)

train_acc, _ = test_accuracy(misc_df_train, trees_dict[best_hyper_param])
eval_acc, prediction_list = test_accuracy(misc_df_eval, trees_dict[best_hype

print(f"Accuracy of tree on train dataset: ", train_acc)
print(f"Accuracy of tree on eval dataset: ", eval_acc)
```

```
Limiting Depth Tree

Best hyper parameter (depth): 6
Exporting tree to 'best-hyper-param-decision-tree.json'.
Accuracy of tree on train dataset:  0.7982857142857143
Accuracy of tree on eval dataset:  0.6723809523809524
```