Importing Libraries

```
import pandas as pd
```

Vertebrate Data

```
vertebrate_data = {
    "Vertebrate Name": ["human","python","salmon","whale","frog","komodo dragon","bat","pigeon","cat","leopard shark","turtle","peng
    "Body Temperature":["warm-blooded","cold-blooded","cold-blooded","warm-blooded","cold-blooded","cold-blooded","warm-blooded","wa
    "Skin Cover":["hair","scales","scales","hair","none","scales","hair","feathers","fur","scales","scales","feathers","quills","sca
    "Gives Birth":["yes","no","no","yes","no","no","yes","no","yes","yes","no","no","yes","no","no"],
    "Aquatic Creature":["no","no","yes","yes","semi","no","no","no","no","yes","semi","semi","no","yes","semi"],
    "Aerial Creature":["no","no","no","no","no","no","yes","yes","no","no","no","no","no","no","no"],
    "Has Legs":["yes","no","no","no","yes","yes","yes","yes","yes","no","yes","yes","yes","no","yes"],
    "Hibernates":["no","yes","no","no","yes","no","yes","no","no","no","no","no","yes","no","yes"],
    "Class Label":["mammal","reptile","fish","mammal","amphibian","reptile","mammal","bird","mammal","fish","reptile","bird","mammal
}
```

Loading datasets

```
weather_df = pd.read_csv("https://gist.githubusercontent.com/bigsnarfdude/515849391ad37fe593997fe0db98afaa/raw/f663366d17b7d05de61a1
vertebrate_df = pd.DataFrame(vertebrate_data)
```

Weather Data

```
weather_df
```

	outlook	temperature	humidity	windy	play
0	overcast	hot	high	False	yes
1	overcast	cool	normal	True	yes
2	overcast	mild	high	True	yes
3	overcast	hot	normal	False	yes
4	rainy	mild	high	False	yes
5	rainy	cool	normal	False	yes
6	rainy	cool	normal	True	no
7	rainy	mild	normal	False	yes
8	rainy	mild	high	True	no
9	sunny	hot	high	False	no
10	sunny	hot	high	True	no
11	sunny	mild	high	False	no
12	sunny	cool	normal	False	yes
13	sunny	mild	normal	True	yes

Next steps: Generate code with weather_df New interactive sheet

Weather Data Info

```
weather_df.info()
print("\n\n")
weather_df.describe()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14 entries, 0 to 13
Data columns (total 5 columns):
 #   Column       Non-Null Count  Dtype 
---  ------       --------------  ----- 
 0   outlook      14 non-null     object
 1   temperature  14 non-null     object
 2   humidity     14 non-null     object
 3   windy        14 non-null     bool  
 4   play         14 non-null     object
dtypes: bool(1), object(4)
memory usage: 594.0+ bytes
```

	outlook	temperature	humidity	windy	play
count	14	14	14	14	14
unique	3	3	2	2	2
top	rainy	mild	high	False	yes
freq	5	6	7	8	9

```
weather_df
```

	outlook	temperature	humidity	windy	play
0	overcast	hot	high	False	yes
1	overcast	cool	normal	True	yes
2	overcast	mild	high	True	yes
3	overcast	hot	normal	False	yes
4	rainy	mild	high	False	yes
5	rainy	cool	normal	False	yes
6	rainy	cool	normal	True	no
7	rainy	mild	normal	False	yes
8	rainy	mild	high	True	no
9	sunny	hot	high	False	no
10	sunny	hot	high	True	no
11	sunny	mild	high	False	no
12	sunny	cool	normal	False	yes
13	sunny	mild	normal	True	yes

Next steps: Generate code with `weather_df` New interactive sheet

Vertebrate Data

```
vertebrate_df
```

	Vertebrate Name	Body Temperature	Skin Cover	Gives Birth	Aquatic Creature	Aerial Creature	Has Legs	Hibernates	Class Label
0	human	warm-blooded	hair	yes	no	no	yes	no	mammal
1	python	cold-blooded	scales	no	no	no	no	yes	reptile
2	salmon	cold-blooded	scales	no	yes	no	no	no	fish
3	whale	warm-blooded	hair	yes	yes	no	no	no	mammal
4	frog	cold-blooded	none	no	semi	no	yes	yes	amphibian
5	komodo dragon	cold-blooded	scales	no	no	no	yes	no	reptile
6	bat	warm-blooded	hair	yes	no	yes	yes	yes	mammal
7	pigeon	warm-blooded	feathers	no	no	yes	yes	no	bird
8	cat	warm-blooded	fur	yes	no	no	yes	no	mammal
9	leopard shark	cold-blooded	scales	yes	yes	no	no	no	fish
10	turtle	cold-blooded	scales	no	semi	no	yes	no	reptile
11	penguin	warm-blooded	feathers	no	semi	no	yes	no	bird
12	porcupine	warm-blooded	quills	yes	no	no	yes	yes	mammal
13	eel	cold-blooded	scales	no	yes	no	no	no	fish

Next steps: Generate code with `vertebrate_df` New interactive sheet

Vertebate data info

```
vertebrate_df.info()
print("\n\n")
vertebrate_df.describe()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 15 entries, 0 to 14
Data columns (total 9 columns):
 #   Column            Non-Null Count  Dtype 
---  ------            --------------  ----- 
 0   Vertebrate Name   15 non-null     object
 1   Body Temperature  15 non-null     object
 2   Skin Cover        15 non-null     object
 3   Gives Birth       15 non-null     object
 4   Aquatic Creature  15 non-null     object
 5   Aerial Creature   15 non-null     object
 6   Has Legs          15 non-null     object
 7   Hibernates        15 non-null     object
 8   Class Label       15 non-null     object
dtypes: object(9)
memory usage: 1.2+ KB
```

	Vertebrate Name	Body Temperature	Skin Cover	Gives Birth	Aquatic Creature	Aerial Creature	Has Legs	Hibernates	Class Label
count	15	15	15	15	15	15	15	15	15
unique	15	2	6	2	3	2	2	2	5
top	human	cold-blooded	scales	no	no	no	yes	no	mammal
freq	1	8	6	9	7	13	10	10	5

Importing libraries for Classification

```
from sklearn.preprocessing import LabelEncoder,OrdinalEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report,confusion_matrix,accuracy_score
from sklearn.tree import DecisionTreeClassifier,export_text,plot_tree,_tree
import matplotlib.pyplot as plt
import seaborn as sns
import re
```

Preprocessing weather data

```
weather_X = weather_df.iloc[:,:-1]

weather_feature_bool = weather_X.select_dtypes(include="bool").columns.to_list()
weather_feature_string = weather_X.select_dtypes(include="object").columns.to_list()

weather_X[weather_feature_bool] = weather_X[weather_feature_bool].astype(int)

enc = OrdinalEncoder()
weather_X[weather_feature_string] = enc.fit_transform(weather_X[weather_feature_string])
weather_map = {}

for cols,category in zip(weather_feature_string,enc.categories_):
    weather_map[cols] = {i:cat for i,cat in enumerate(category)}

le = LabelEncoder()
weather_y = le.fit_transform(weather_df.iloc[:,-1])
weather_class_names = le.classes_
```

```
weather_X
```

	outlook	temperature	humidity	windy
0	0.0	1.0	0.0	0
1	0.0	0.0	1.0	1
2	0.0	2.0	0.0	1
3	0.0	1.0	1.0	0
4	1.0	2.0	0.0	0
5	1.0	0.0	1.0	0
6	1.0	0.0	1.0	1
7	1.0	2.0	1.0	0
8	1.0	2.0	0.0	1
9	2.0	1.0	0.0	0
10	2.0	1.0	0.0	1
11	2.0	2.0	0.0	0
12	2.0	0.0	1.0	0
13	2.0	2.0	1.0	1

Next steps: Generate code with weather_X New interactive sheet

Preprocessing vertebrate data

```
vertebrate_df.drop("Vertebrate Name",axis=1,inplace=True)
vertebrate_df.columns = [col.replace(" ","_") for col in vertebrate_df.columns]

vertebrate_X = vertebrate_df.iloc[:,:-1]
vertebrate_feature_bool = vertebrate_X.select_dtypes(include="bool").columns.to_list()
vertebrate_feature_string = vertebrate_X.select_dtypes(include="object").columns.to_list()

vertebrate_X[vertebrate_feature_bool] = vertebrate_X[vertebrate_feature_bool].astype(int)

enc = OrdinalEncoder()
vertebrate_X[vertebrate_feature_string] = enc.fit_transform(vertebrate_X[vertebrate_feature_string])
vertebrate_map = {}

for cols,category in zip(vertebrate_feature_string,enc.categories_):
    vertebrate_map[cols] = {i:cat for i,cat in enumerate(category)}

le = LabelEncoder()
vertebrate_y = le.fit_transform(vertebrate_df.iloc[:,-1])
vertebrate_class_names = le.classes_
```

```
/tmp/ipython-input-2677124015.py:11: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a
  vertebrate_X[vertebrate_feature_string] = enc.fit_transform(vertebrate_X[vertebrate_feature_string])
```

```
vertebrate_X
```

	Body_Temperature	Skin_Cover	Gives_Birth	Aquatic_Creature	Aerial_Creature	Has_Legs	Hibernates
0	1.0	2.0	1.0	0.0	0.0	1.0	0.0
1	0.0	5.0	0.0	0.0	0.0	0.0	1.0
2	0.0	5.0	0.0	2.0	0.0	0.0	0.0
3	1.0	2.0	1.0	2.0	0.0	0.0	0.0
4	0.0	3.0	0.0	1.0	0.0	1.0	1.0
5	0.0	5.0	0.0	0.0	0.0	1.0	0.0
6	1.0	2.0	1.0	0.0	1.0	1.0	1.0
7	1.0	0.0	0.0	0.0	1.0	1.0	0.0
8	1.0	1.0	1.0	0.0	0.0	1.0	0.0
9	0.0	5.0	1.0	2.0	0.0	0.0	0.0
10	0.0	5.0	0.0	1.0	0.0	1.0	0.0
11	1.0	0.0	0.0	1.0	0.0	1.0	0.0
12	1.0	4.0	1.0	0.0	0.0	1.0	1.0
13	0.0	5.0	0.0	2.0	0.0	0.0	0.0
14	0.0	3.0	0.0	1.0	0.0	1.0	1.0

Next steps: Generate code with vertebrate_X New interactive sheet

Train test splitting datasets

```
weather_X_train,weather_X_test,weather_y_train,weather_y_test = train_test_split(weather_X,weather_y,random_state=18,test_size=0.2,s
vertebrate_X_train,vertebrate_X_test,vertebrate_y_train,vertebrate_y_test = train_test_split(vertebrate_X,vertebrate_y,random_state=
```

Showing target classes of both datasets

```
print("weather classes",weather_class_names)
print("vertebrate classes",vertebrate_class_names)
```

```
weather classes ['no' 'yes']
vertebrate classes ['amphibian' 'bird' 'fish' 'mammal' 'reptile']
```

Fitting Decision Tree Classifier on Weather data for rules

```
weather_model = DecisionTreeClassifier(
    random_state=18,
    max_depth=4,
    criterion='gini',
)
```

```
weather_model.fit(weather_X_train,weather_y_train)
weather_y_pred = weather_model.predict(weather_X_test)
print(f"Accuracy : {accuracy_score(weather_y_test, weather_y_pred):.4f}\n")
weather_report = pd.DataFrame(classification_report(weather_y_test,weather_y_pred,target_names=weather_class_names,output_dict=True)
```

```
Accuracy : 1.0000
```

Fitting Decision Tree Classifier on Vertebrate data for rules

```
vertebrate_model = DecisionTreeClassifier(
    random_state=18,
    max_depth=4,
    criterion='gini',
)
vertebrate_model.fit(vertebrate_X_train,vertebrate_y_train)
vertebrate_y_pred = vertebrate_model.predict(vertebrate_X_test)
print(f"Accuracy : {accuracy_score(vertebrate_y_test, vertebrate_y_pred):.4f}\n")
vertebrate_report = pd.DataFrame(classification_report(vertebrate_y_test,vertebrate_y_pred,target_names=vertebrate_class_names,outpu
```

```
Accuracy : 0.6667

/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

Weather data model report

```
weather_report
```

	no	yes	accuracy	macro avg	weighted avg
precision	1.0	1.0	1.0	1.0	1.0
recall	1.0	1.0	1.0	1.0	1.0
f1-score	1.0	1.0	1.0	1.0	1.0
support	1.0	2.0	1.0	3.0	3.0

Next steps: Generate code with weather_report New interactive sheet

Vertebrate data model report

```
vertebrate_report
```

	amphibian	bird	fish	mammal	reptile	accuracy	macro avg	weighted avg
precision	1.0	1.0	0.0	0.666667	0.0	0.666667	0.533333	0.555556
recall	1.0	1.0	0.0	1.000000	0.0	0.666667	0.600000	0.666667
f1-score	1.0	1.0	0.0	0.800000	0.0	0.666667	0.560000	0.600000
support	1.0	1.0	1.0	2.000000	1.0	0.666667	6.000000	6.000000

Next steps: Generate code with vertebrate_report New interactive sheet

Generate Rules

```
def text_rules_to_flat_decoded(text_rules, category_map=None, bool_cols=None,prefix="class_"):
    category_map = category_map or {}
    bool_cols = bool_cols or []

    def decode_condition(condition):
        match = re.match(r'(\w+)\s*([<>=]+)\s*([\d.]+)', condition)
        if not match:
            return condition

        name, operator, value = match.groups()
        threshold = float(value)

        if name in bool_cols:
            if '<=' in operator:
                return f"it's {name}"
            else:
                return f"it's not {name}"

        if name in category_map:
            threshold_int = int(threshold)
            mapping = category_map[name]

            if '<=' in operator:
                categories = [mapping[i] for i in sorted(mapping.keys()) if i <= threshold_int]
                if len(categories) == 1:
                    return f'{name} is {categories[0]}'
                else:
                    return f'{name} in {categories}'
            else:
                categories = [mapping[i] for i in sorted(mapping.keys()) if i > threshold_int]
                if len(categories) == 1:
                    return f'{name} is {categories[0]}'
                else:
                    return f'{name} in {categories}'

        return condition

    lines = text_rules.strip().split('\n')
    rules = []
    current_conditions = []

    for line in lines:
        depth = (len(line) - len(line.lstrip('| '))) // 4
        clean_line = line.lstrip('|- ').strip()
        current_conditions = current_conditions[:depth]

        if 'class:' in clean_line:
            class_match = re.search(r'class:\s*(\S+)', clean_line)
            if class_match:
                class_label = class_match.group(1)
                decoded_conditions = [decode_condition(c) for c in current_conditions]
                condition_str = " && ".join(decoded_conditions) if decoded_conditions else "True"
                rules.append(f"({condition_str}) -> {prefix}{class_label}")
        else:
            current_conditions.append(clean_line)

    return rules
```

Weather data rules

```
raw_rules = export_text(weather_model, feature_names=weather_X.columns,class_names=weather_class_names)
print(raw_rules)
for rule in text_rules_to_flat_decoded(raw_rules,weather_map,weather_feature_bool,"Play "):
    print(rule)
```

```
|--- outlook <= 0.50
|   |--- class: yes
|--- outlook >  0.50
|   |--- humidity <= 0.50
|   |   |--- windy <= 0.50
|   |   |   |--- outlook <= 1.50
|   |   |   |   |--- class: yes
|   |   |   |--- outlook >  1.50
|   |   |   |   |--- class: no
|   |   |--- windy >  0.50
|   |   |   |--- class: no
|   |--- humidity >  0.50
|   |   |--- windy <= 0.50
|   |   |   |--- class: yes
|   |   |--- windy >  0.50
|   |   |   |--- temperature <= 1.00
|   |   |   |   |--- class: no
|   |   |   |--- temperature >  1.00
|   |   |   |   |--- class: yes

(outlook is overcast) -> Play yes
(outlook in ['rainy', 'sunny'] && humidity is high && it's windy && outlook in ['overcast', 'rainy']) -> Play yes
(outlook in ['rainy', 'sunny'] && humidity is high && it's windy && outlook is sunny) -> Play no
(outlook in ['rainy', 'sunny'] && humidity is high && it's not windy) -> Play no
(outlook in ['rainy', 'sunny'] && humidity is normal && it's windy) -> Play yes
(outlook in ['rainy', 'sunny'] && humidity is normal && it's not windy && temperature in ['cool', 'hot']) -> Play no
(outlook in ['rainy', 'sunny'] && humidity is normal && it's not windy && temperature is mild) -> Play yes
```

Vertebrate data rules

```
raw_rules = export_text(vertebrate_model, feature_names=vertebrate_X.columns,class_names=vertebrate_class_names)
print(raw_rules)
for rule in text_rules_to_flat_decoded(raw_rules,vertebrate_map,vertebrate_feature_bool,"It's a "):
    print(rule)
```

```
|--- Gives_Birth <= 0.50
|   |--- Has_Legs <= 0.50
|   |   |--- class: fish
|   |--- Has_Legs >  0.50
|   |   |--- Skin_Cover <= 4.00
|   |   |   |--- Hibernates <= 0.50
|   |   |   |   |--- class: bird
|   |   |   |--- Hibernates >  0.50
|   |   |   |   |--- class: amphibian
|   |   |--- Skin_Cover >  4.00
|   |   |   |--- class: reptile
|--- Gives_Birth >  0.50
|   |--- class: mammal

(Gives_Birth is no && Has_Legs is no) -> It's a fish
(Gives_Birth is no && Has_Legs is yes && Skin_Cover in ['feathers', 'fur', 'hair', 'none', 'quills'] && Hibernates is no) -> It's a b
```

```
(Gives_Birth is no && Has_Legs is yes && Skin_Cover in ['feathers', 'fur', 'hair', 'none', 'quills'] && Hibernates is yes) -> It's a
(Gives_Birth is no && Has_Legs is yes && Skin_Cover is scales) -> It's a reptile
(Gives_Birth is yes) -> It's a mammal
```

Feature Importance

```
importance_df = pd.DataFrame({
    'feature':    weather_X.columns,
    'importance': weather_model.feature_importances_
}).sort_values('importance', ascending=False)

print("feature importance of weather data\n")
print(importance_df.to_string(index=False))
```

```
feature importance of weather data

    feature  importance
    outlook    0.410714
temperature    0.196429
   humidity    0.196429
      windy    0.196429
```

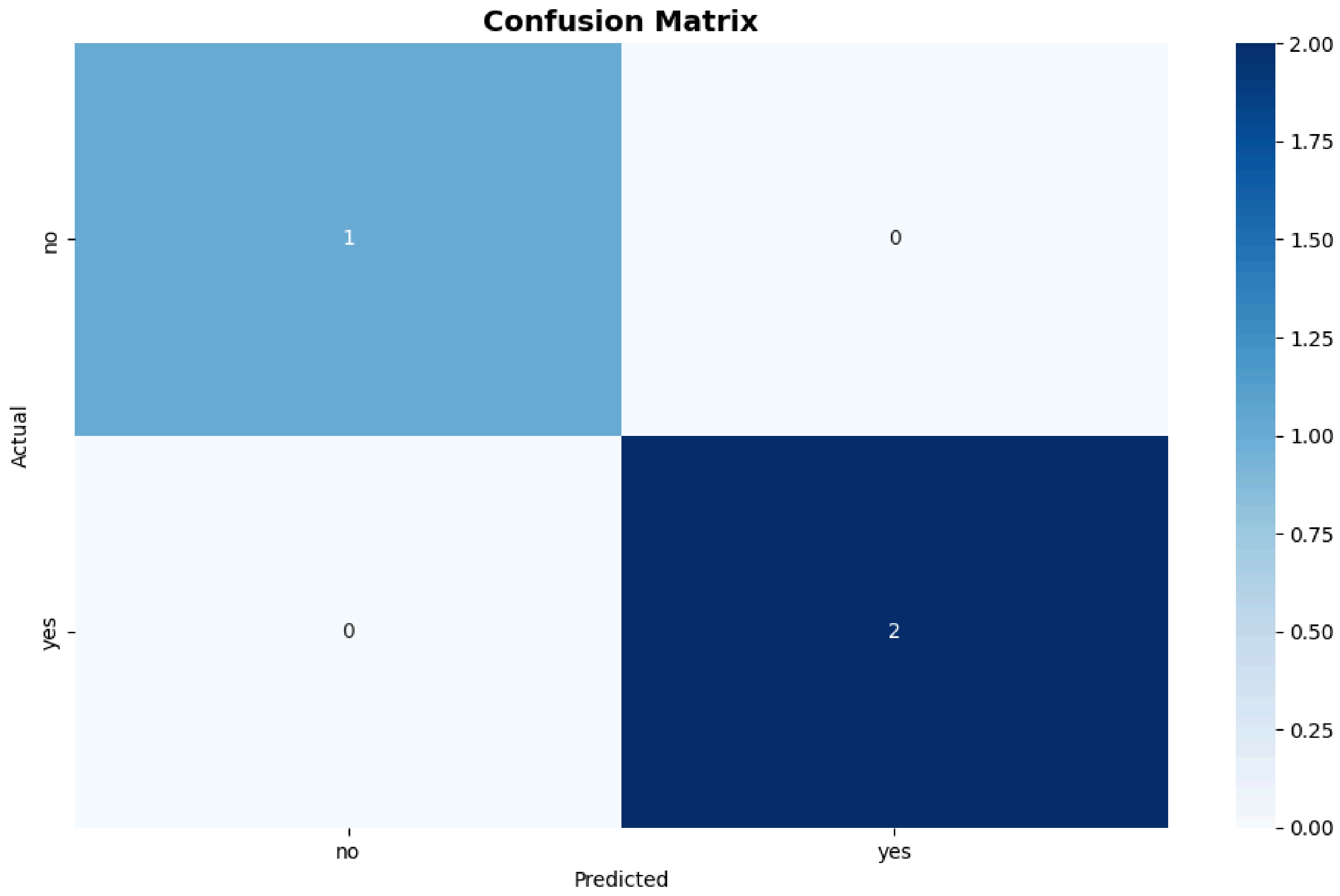
```
importance_df = pd.DataFrame({
    'feature':    vertebrate_X.columns,
    'importance': vertebrate_model.feature_importances_
}).sort_values('importance', ascending=False)

print("feature importance of vertebrate data\n")
print(importance_df.to_string(index=False))
```

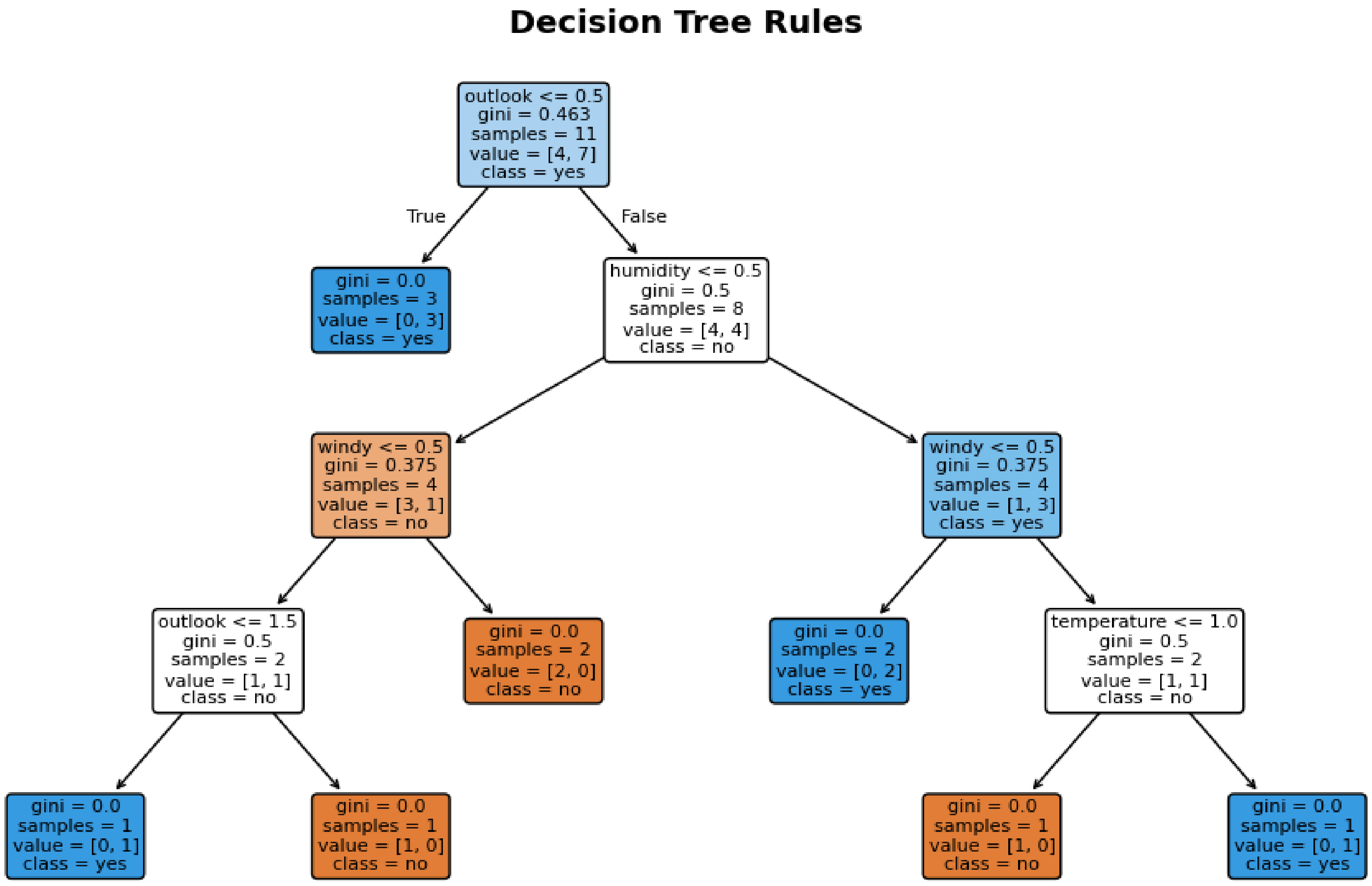
```
feature importance of vertebrate data

         feature  importance
     Gives_Birth    0.370968
        Has_Legs    0.266129
      Skin_Cover    0.217742
      Hibernates    0.145161
Body_Temperature    0.000000
 Aerial_Creature    0.000000
Aquatic_Creature    0.000000
```

Weather data ruleset evaluation

```
weather_df['predicted_label'] = le.inverse_transform(weather_model.predict(weather_X))

proba = weather_model.predict_proba(weather_X)
for i, cls in enumerate(weather_class_names):
    weather_df[f'prob_{cls}'] = proba[:, i].round(3)

print("\n--- Sample predictions ---")
print(weather_df[["play", 'predicted_label']].head(10))

fig, axes = plt.subplots(2, 1, figsize=(10, 12))

cm = confusion_matrix(weather_y_test, weather_y_pred)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=weather_class_names, yticklabels=weather_class_names, ax=axes[0])
axes[0].set_title('Confusion Matrix', fontsize=14, fontweight='bold')
axes[0].set_xlabel('Predicted')
axes[0].set_ylabel('Actual')

plot_tree(weather_model, feature_names=weather_X.columns, class_names=weather_class_names,
          filled=True, rounded=True, fontsize=8, ax=axes[1])
axes[1].set_title('Decision Tree Rules', fontsize=14, fontweight='bold')

plt.tight_layout()
plt.savefig('weather classifier_output.png', dpi=150, bbox_inches='tight')
plt.show()
print("Plot saved → classifier_output.png")
```

```
--- Sample predictions ---
  play predicted_label
0  yes            bird
1  yes            bird
2  yes            bird
3  yes            bird
4  yes            bird
5  yes            bird
6   no       amphibian
7  yes            bird
8   no       amphibian
9   no       amphibian
```



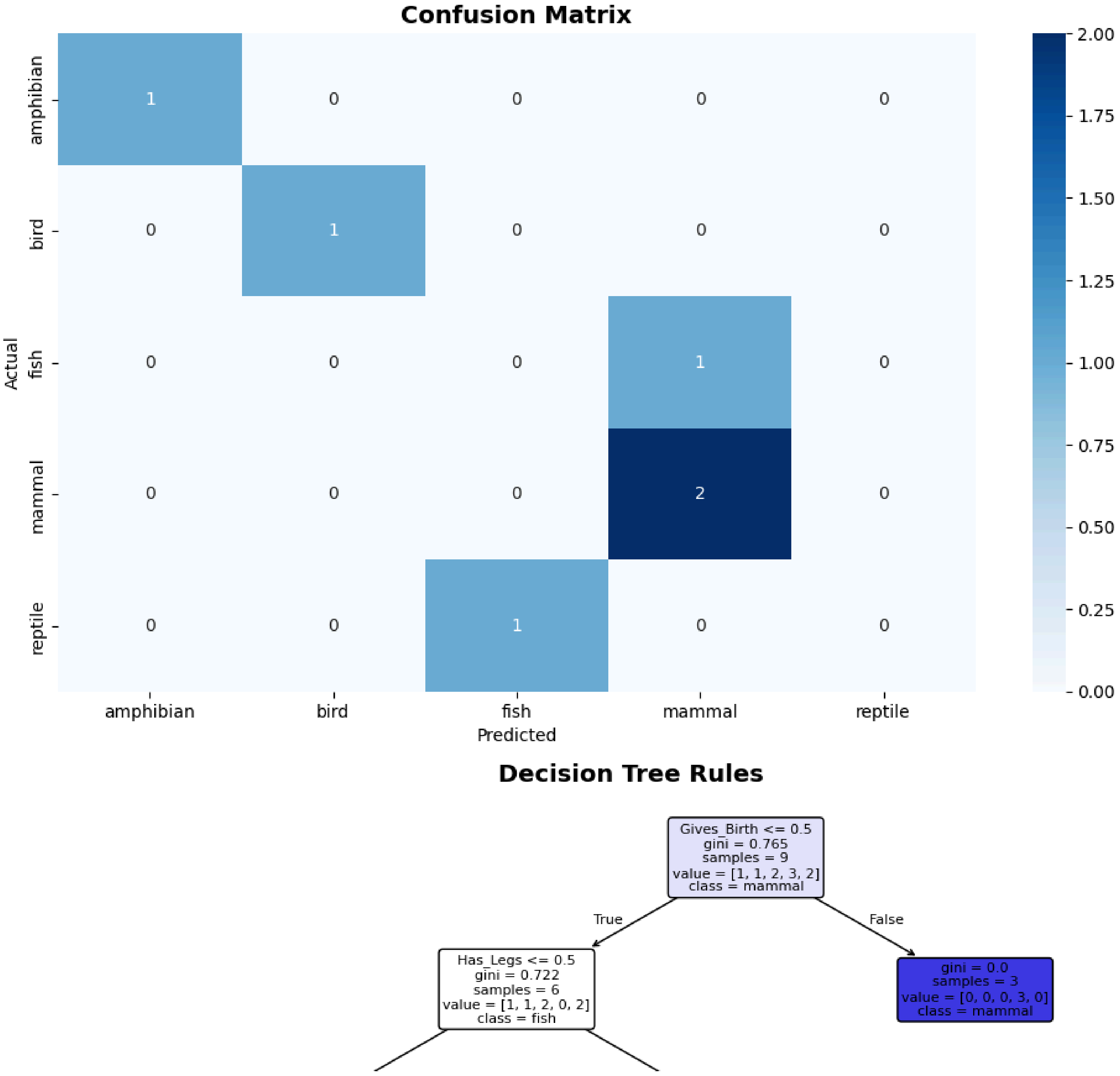


```
Plot saved → classifier output.png
```

Vertebrate data ruleset evaluation

```
vertebrate_df['predicted_label'] = le.inverse_transform(vertebrate_model.predict(vertebrate_X))

proba = vertebrate_model.predict_proba(vertebrate_X)
for i, cls in enumerate(vertebrate_class_names):
    vertebrate_df[f'prob_{cls}'] = proba[:, i].round(3)

print("\n--- Sample predictions ---")
print(vertebrate_df[["Class_Label", 'predicted_label']].head(10))

fig, axes = plt.subplots(2, 1, figsize=(10, 12))

cm = confusion_matrix(vertebrate_y_test, vertebrate_y_pred)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=vertebrate_class_names, yticklabels=vertebrate_class_names, ax=axes[0])
axes[0].set_title('Confusion Matrix', fontsize=14, fontweight='bold')
axes[0].set_xlabel('Predicted')
axes[0].set_ylabel('Actual')

plot_tree(vertebrate_model, feature_names=vertebrate_X.columns, class_names=vertebrate_class_names,
          filled=True, rounded=True, fontsize=8, ax=axes[1])
axes[1].set_title('Decision Tree Rules', fontsize=14, fontweight='bold')

plt.tight_layout()
plt.savefig('vertebrate classifier_output.png', dpi=150, bbox_inches='tight')
plt.show()
print("Plot saved → classifier_output.png")
```

```
--- Sample predictions ---
  Class_Label predicted_label
0      mammal          mammal
1     reptile            fish
2        fish            fish
3      mammal          mammal
4   amphibian       amphibian
5     reptile         reptile
6      mammal          mammal
7        bird            bird
8      mammal          mammal
9        fish          mammal
```


Confusion Matrix

Decision Tree Rules