Decision tree predictor

In [1]:
```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from sklearn.preprocessing import StandardScaler
import pickle
```

Load Data

load the training and testing data

In [2]:
```
# Load feature vectors and labels
X = np.load("../X.npy")
y = np.load("../y.npy")

with open("../label_map.pkl", "rb") as f:
    label_map = pickle.load(f)
class_names = [label_map[i] for i in range(len(label_map))]
```

Split the data

In [3]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, str
```

Scale Photos

In [4]:
```
# Normalize to improve model performance
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

In [5]:
```
import joblib
joblib.dump(scaler, "DT_scaler.pkl")
```

Out[5]:
```
['DT_scaler.pkl']
```

Train the module

In [6]:
```
from sklearn.ensemble import RandomForestClassifier

rf = RandomForestClassifier(
    n_estimators=600,          # More trees = better ensemble (200–500 is goo
    max_depth=None,              # Slightly deeper trees (try 20–30)
    min_samples_split=4,       # Prevents overfitting, needs at least 4 sampl
    min_samples_leaf=2,        # Prevents very small leaves (try 1–4)
```

```
    max_features='sqrt',      # Good for classification tasks (try also 'log
    class_weight='balanced',  # Handles class imbalance by weighting classes
    bootstrap=True,           # Enables bootstrap sampling (default)
    random_state=42,          # For reproducibility
    n_jobs=-1                 # Use all CPU cores to speed up training
)

rf.fit(X_train, y_train)
```

Out[6]:

RandomForestClassifier

```
RandomForestClassifier(class_weight='balanced', min_samples_leaf=2,
                       min_samples_split=4, n_estimators=600, n_job
s=-1,
                       random_state=42)
```

Predict the testing data

In [7]:
```
y_pred_rf = rf.predict(X_test)
print("Random Forest Report:\n", classification_report(y_test, y_pred_rf, ta
```

```
Random Forest Report:
               precision    recall  f1-score   support

        cats       0.78      0.72      0.75       200
       panda       0.82      0.84      0.83       199
     spiders       0.82      0.86      0.84       200

    accuracy                           0.81       599
   macro avg       0.81      0.81      0.81       599
weighted avg       0.81      0.81      0.81       599
```

Confusion Matrix Results

In [8]:
```
import pandas as pd
from sklearn.metrics import confusion_matrix

cm = confusion_matrix(y_test, y_pred_rf)
class_names = ['cat', 'panda', 'spider']
cm_df = pd.DataFrame(cm, index=class_names, columns=class_names)

print("Confusion Matrix:")
print(cm_df)
```

```
Confusion Matrix:
        cat  panda  spider
cat     145     28      27
panda    20    167      12
spider   20      8     172
```

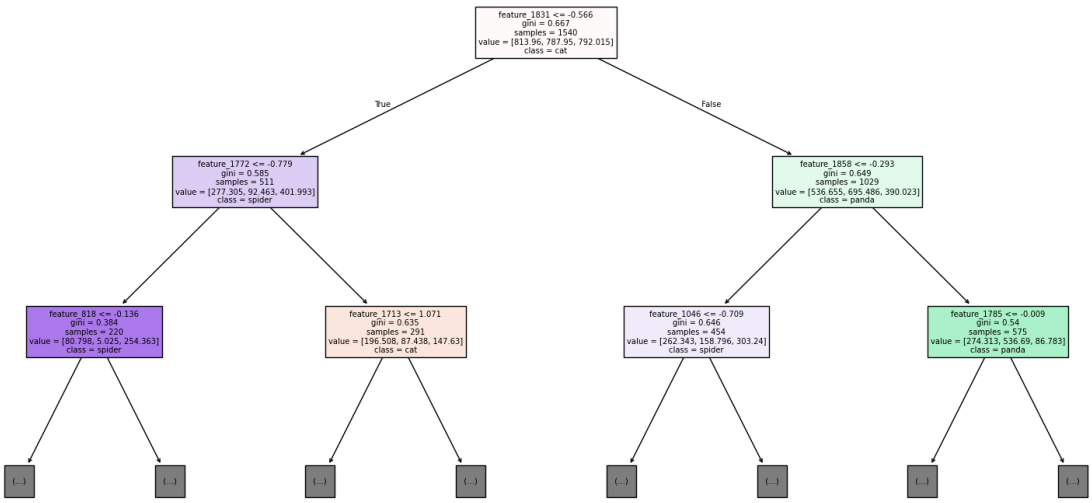
Print The Final Tree Module

In [9]:
```
from sklearn.tree import export_text

estimator = rf.estimators_[0]
feature_names = [f"feature_{i}" for i in range(X.shape[1])]
tree_rules = export_text(estimator, feature_names=feature_names)
```

In [10]:
```
from sklearn.tree import plot_tree
import matplotlib.pyplot as plt

plt.figure(figsize=(20, 10))
plot_tree(estimator, filled=True, feature_names=feature_names, class_names=c
plt.show()
```



Load the module

In [11]:
```
import joblib
joblib.dump(rf, "DT_model.pkl")
```

Out[11]:
```
['DT_model.pkl']
```

In [ ]:
```

```