Last Doc Here:

SLIDE 1: Matt

WIth the upcoming eclipse, we're a little concerned about the world ending.

One thing is for certain: fungus will survive no matter how things go down.

And if you think we're kidding…here's a quick clip to give you flavor of where are heads are yet:

Colab: https://colab.research.google.com/drive/1HnM6xy1pLLL1i6vgZGxzj9ZUYPRlOE-Q

Dataset

This dataset is from the UCI machine learning library and is over 30 years old. "This dataset includes descriptions of hypothetical samples corresponding to 23 species of gilled mushrooms in the Agaricus and Lepiota Family Mushroom drawn from The Audubon Society Field Guide to North American Mushrooms (1981)"

Installation:
You will need the following libraries

```
import plotly.express as px
import pandas as pd
from sklearn.model_selection import train_test_split
import numpy as np
from sklearn.preprocessing import LabelEncoder
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix,
classification_report
from sklearn.tree import export_graphviz
import matplotlib.pyplot as plt
import seaborn as sns
import os
from IPython.display import Image
import pydotplus
!pip install pydotplus
!pip install plotly
```

Data Processing {nate}

- Made very large dictionary associating the labels to their meaning

```
# Preprocess the data
def preprocess_data(data):
    # Remove columns that are not informative
    data = data.drop(columns=['veil-type'])

    # Encode categorical variables
    label_encoders = {}
    for column in data.columns:
        le = LabelEncoder()
        data[column] = le.fit_transform(data[column])
        label_encoders[column] = le

    # Separate features and target variable
    X = data.drop(columns=['class'])
    y = data['class']

    return X, y, label_encoders

# Split the dataset into training and testing sets
def split_dataset(X, y, test_size=0.2, random_state=42):
    X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=test_size, random_state=random_state)
    return X_train, X_test, y_train, y_test
```

Data was mostly clean as it was labeled according to the dictionary. Using the preprocess_data function, we used a label encoder to turn unique values in each feature to a separate integer.

Questions during presentation:

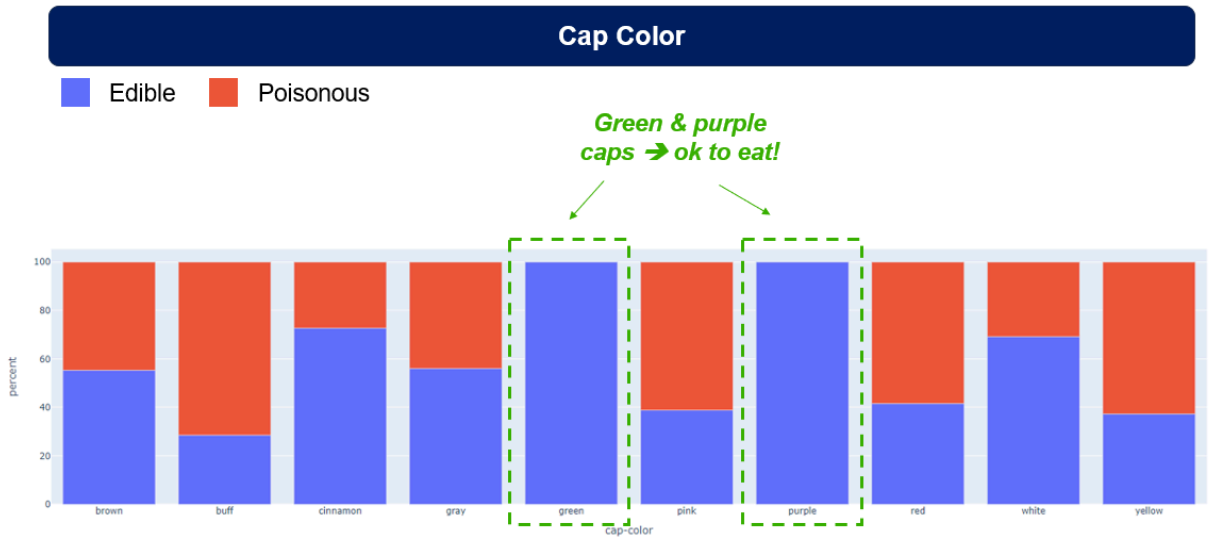
1) Why does the model predict poisonous / safe with 100% accuracy? (Ken)

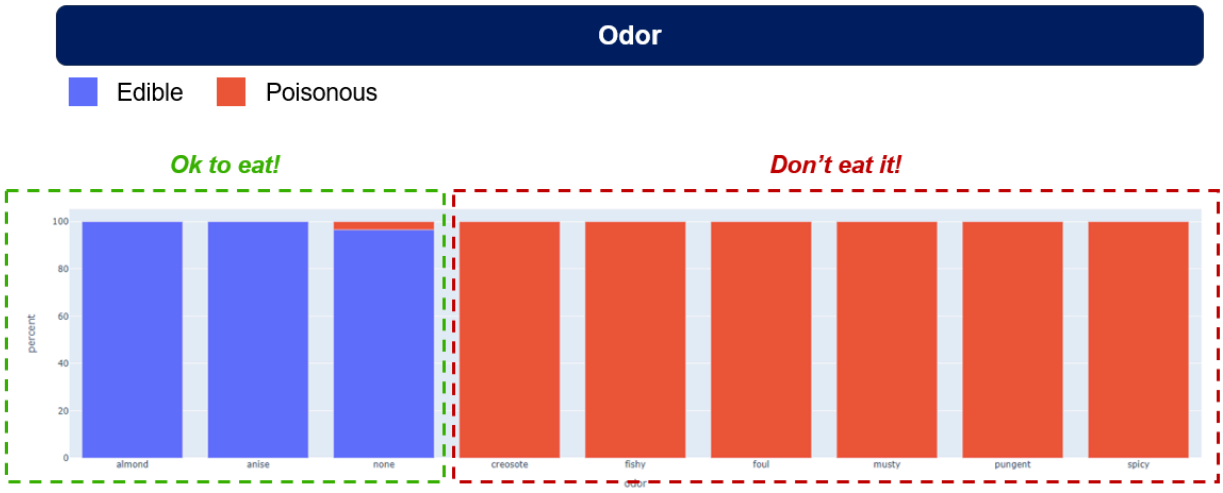
The parallel between mushroom classification and decision trees are profound. In the wild, the traits distinguishing safe from harmful mushrooms form a natural decision-making pathway, similar to a decision tree's logic in machine learning.

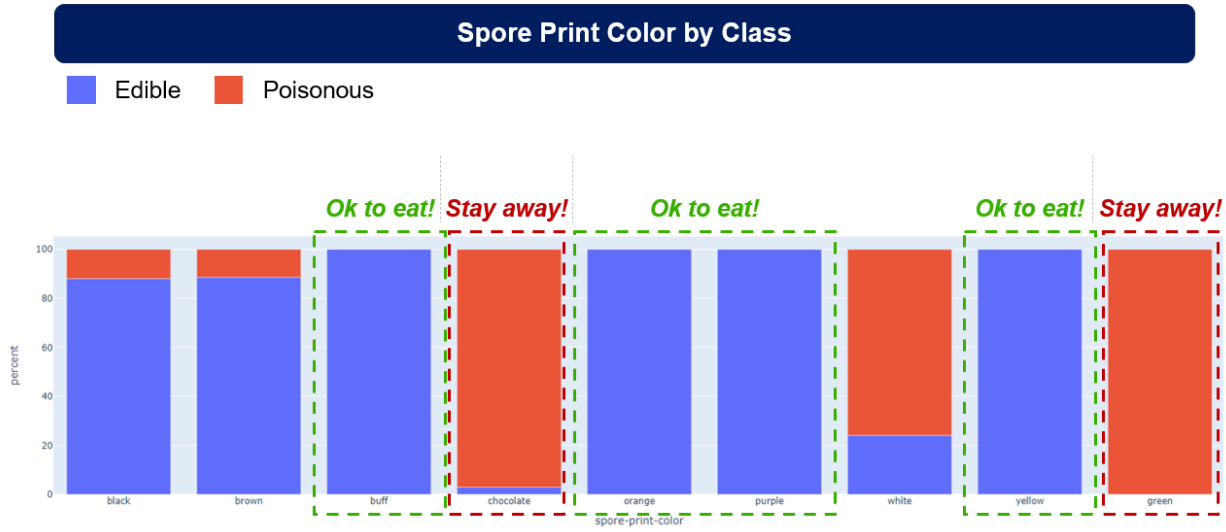
Characteristics such as smell, color, habitat, and spore color serve as pivotal points, steering the classification towards either poisonous or edible.

In machine learning, a decision tree using these attributes would segment the data to separate purely poisonous or edible groups at every node. This natural decision-making process is notably straightforward yet efficient, reflecting in machine learning models' ability to classify with high precision. Mushroom classification in the wild can essentially be achieved through if/else statements, and Evan will explain a few things to look for.

2) If you are only going to remember 2-3 things from today's presentation to help you identify poisonous mushrooms out in the wild, these are those 2-3 things (Evan)







3) Dangers of relying on AI image classification models (Matt)

LAST SLIDE

Dangers of AI Misidentifying Mushrooms / Shortfall of Visual Models [Matt]

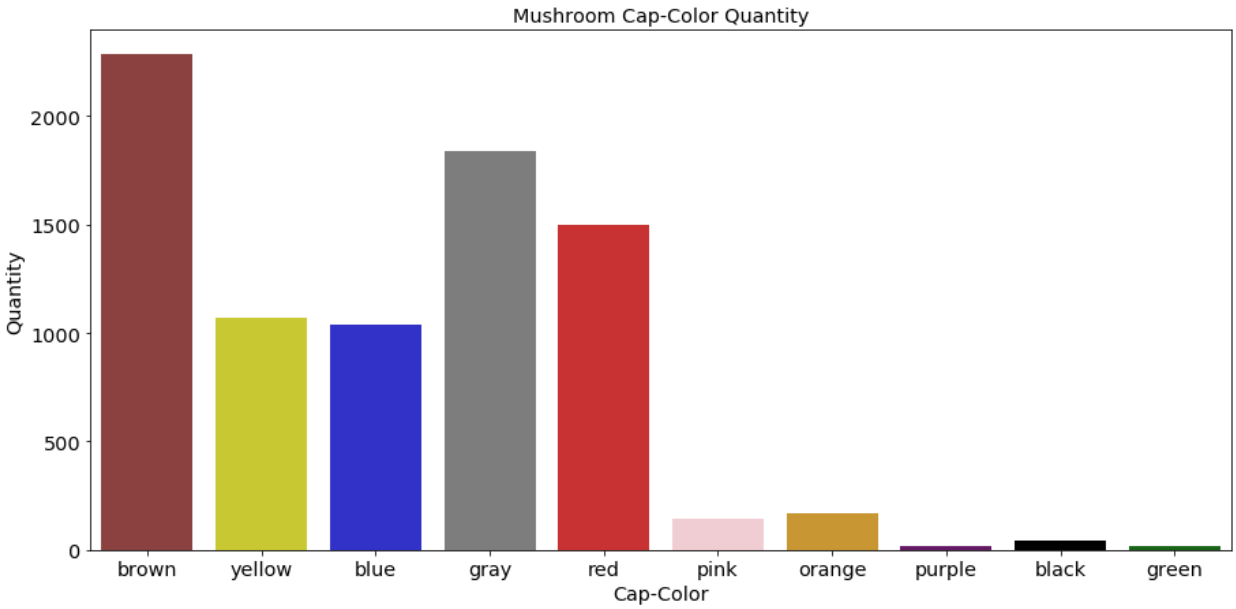
From: https://www.citizen.org/article/mushroom-risk-ai-app-misinformation/

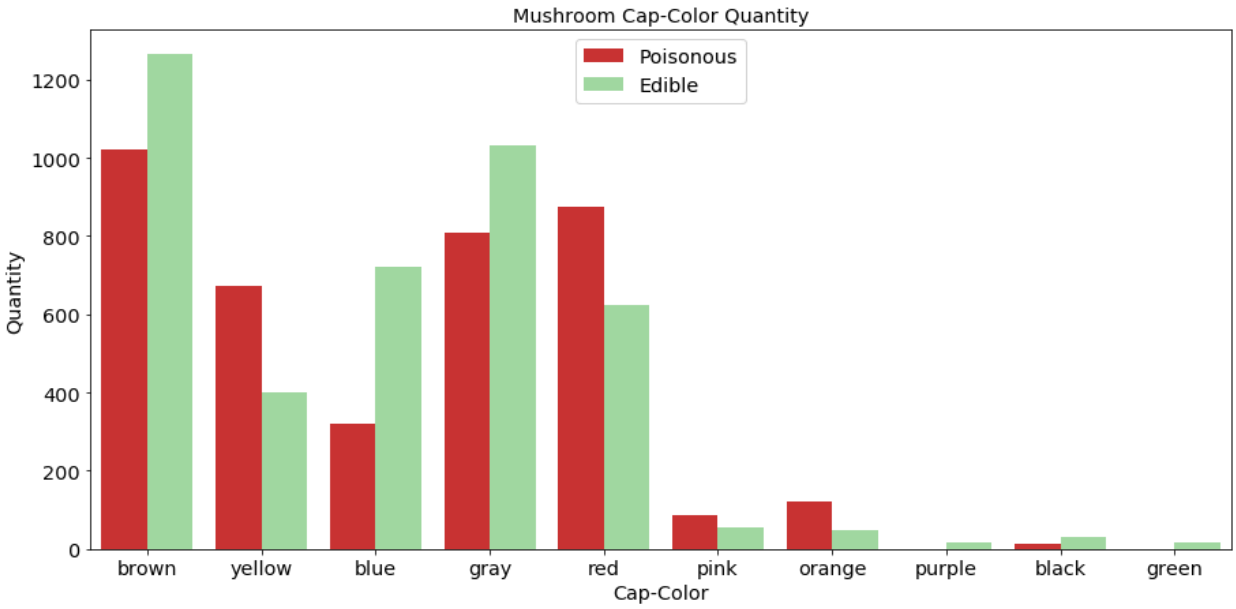
“Following a rise in wild mushroom poisonings, Australian poison researchers in 2022 tested three applications that use A.I. and which foragers often use to identify wild mushrooms. The researchers were interested in assessing the apps’ utility for assisting emergency medical workers in mushroom poisoning triage situations – situations where identifying a mushroom based on digital photographs – and rapidly transporting and administering medicinal treatments when needed – can have literal life-or-death consequences. The apps were tested on digital photos of 78 mushrooms in 2020 and 2021. The researchers collaborated with an expert mycologist who consults with Australian authorities to identify mushrooms implicated in possible poisonings. They found the following:

- **The best-performing app (Picture Mushroom) provided accurate identifications from digital photos less than half (49%) of the time, and identified toxic mushrooms 44% of the time;**
- The runners up, Mushroom Identificator and iNaturalist, offered accurate identifications about a third (35%) of the time, and correctly identified toxic mushrooms 30% and 40% of the time, respectively;
- In terms of which app was most successful at identifying the death cap (*Amanita phalloides*), Mushroom Identificator performed the best, identifying 67% of the specimens, compared to Picture Mushroom (60%) and iNaturalist (27%);
- In some of the apps’ misidentification errors, toxic mushrooms were misidentified as edible mushrooms;
- Regional bias appears to have played a role in some of the misidentification errors, as apps misidentified toxic Australian species as similar-looking edible North American species.

We compared the accuracy of three iPhone™ and Android™ mushroom identification applications: Picture Mushroom (Next Vision Limited©), Mushroom Identificator (Pierre Semedard©), and iNaturalist (iNaturalist, California Academy of Sciences©). Each app was tested independently by three researchers using digital photographs of 78 specimens sent to the Victorian Poisons Information Centre and Royal Botanic Gardens Victoria over a two-year period, 2020-2021. Mushroom identification was confirmed by an expert mycologist. For each app, individual and combined results were compared.

4) Any additional questions that surfaced, fun visuals, what your group might research next if more time was available, or share a plan for future development (James)





—Because the dataset is so straightforward, we were able to visualize it in a multitude of ways. In these graphs for we take cap color by quantity in one and then we can even delineate poison and edibility by color.

—While researching for this we learned that poisonous in this context means you can't eat it. That doesn't mean that there isn't some way for it to benefit humans.

—So, outside of survival scenarios following an apocalypse, for a future project that can benefit humanity more generally, we would be interested in creating a broader classification between poisonous, edible, medicinal, useful for natural dye creation, and viability for biodegradable material synthesis.

—When it comes to prediction, we could organize a citizen-science type project around collecting photos from people around the world. They can use direction from our mycologist Matt on how and what parts specifically to photograph—requiring more than just a picture of the cap, and the metadata from the pictures already contain the latitude and longitude needed to map the location data. We can use the flow of mushroom growth round the planet to see the ecological impact of human expansion and urban development.

-From there we can take ML to predict which species might be going extinct and draw inferences from there on why

—Lastly we can replace mushrooms with other plants and insects to try and replicate this process to glean the same benefits

> If we created an app how we would improve on the failed models
Future projects
Unanswered questions if more time
–like regional bias maybe
Accuracy still 100.00% using gradient boosted

	Feature	Importance
8	gill-color	0.443210
20	population	0.150325
19	spore-print-color	0.070936
9	stalk-shape	0.069119
7	gill-size	0.064978
13	stalk-color-above-ring	0.053445
3	bruises	0.049981
4	odor	0.024492
10	stalk-root	0.017891
21	habitat	0.016402
1	cap-surface	0.011290
14	stalk-color-below-ring	0.008060
12	stalk-surface-below-ring	0.006961
17	ring-number	0.005097
2	cap-color	0.003233
18	ring-type	0.003175
11	stalk-surface-above-ring	0.000765
0	cap-shape	0.000464
6	gill-spacing	0.000177
16	veil-color	0.000000
15	veil-type	0.000000
5	gill-attachment	0.000000

Where do we want to show the decision tree (either question 1 or 2)?

Conclusion (maybe Ken?)

OPPENHEIMER NUKE SCENE / APOCALYPTIC]

```
Accuracy: 100.00%
                 Feature  Importance
8              gill-color    0.443210
20              population    0.150325
19       spore-print-color    0.070936
9              stalk-shape    0.069119
7              gill-size    0.064978
13   stalk-color-above-ring    0.053445
3                bruises    0.049981
4                  odor    0.024492
10             stalk-root    0.017891
21               habitat    0.016402
1             cap-surface    0.011290
14   stalk-color-below-ring    0.008060
12  stalk-surface-below-ring    0.006961
17             ring-number    0.005097
2              cap-color    0.003233
18              ring-type    0.003175
11  stalk-surface-above-ring    0.000765
0              cap-shape    0.000464
6            gill-spacing    0.000177
16             veil-color    0.000000
15              veil-type    0.000000
5          gill-attachment    0.000000


        ```

        import pandas as pd
        from sklearn.model_selection import train_test_split
        from sklearn.preprocessing import LabelEncoder
        from sklearn.metrics import accuracy_score
        import xgboost as xgb

        # Load the dataset
        df = pd.read_csv('mushrooms.csv') # Update this path to your dataset location

        # Encode the categorical data
        label_encoders = {}
        for column in df.columns:
            if df[column].dtype == type(object): # Encoding if the column is categorical
                le = LabelEncoder()
                df[column] = le.fit_transform(df[column])
                label_encoders[column] = le
```
```

```
# Separate features and target
X = df.drop('class', axis=1)
y = df['class']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Instantiate an XGBoost classifier
# 'objective': binary classification
# 'use_label_encoder': False to avoid warning since XGBoost now uses its own
label encoder
model = xgb.XGBClassifier(objective='binary:logistic', use_label_encoder=False,
eval_metric='logloss')

# Train the model
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Calculate the accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy * 100:.2f}%')

# Feature importance
importance_df = pd.DataFrame({
    'Feature': X.columns,
    'Importance': model.feature_importances_
}).sort_values(by='Importance', ascending=False)

print(importance_df)

```
```
```