

Random Forest Heatmap.



palmerpenguins

Unable to display visualization

I am new to Python. I would like to design a random forest classifier for this dataset to predict the species of bird. Please show me the code, and also show me the resulting heatmap that visualizes the confusion matrix of this classifier. Thanks!



python

```
import pandas as pd

# Load the dataset
file_path = '/mnt/data/palmerpenguins.csv'
data = pd.read_csv(file_path)

# Display the first few rows of the dataset to understand its structure
data.head()
```

Result

	species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	\
0	Adelie	Torgersen	39.1	18.7	181	
1	Adelie	Torgersen	39.5	17.4	186	
2	Adelie	Torgersen	40.3	18.0	195	
3	Adelie	Torgersen	36.7	19.3	193	
4	Adelie	Torgersen	39.3	20.6	190	

	body_mass_g	sex	year
0	3750	male	2007
1	3800	female	2007
2	3250	female	2007
3	3450	female	2007
4	3650	male	2007

Here's a step-by-step guide to creating a random forest classifier using this dataset and visualizing the confusion matrix as a heatmap.

1. **Preprocess the Data:** Handle missing values, encode categorical variables, and split the data into features and target.
2. **Train the Random Forest Classifier.**
3. **Evaluate the Classifier:** Generate a confusion matrix.
4. **Visualize the Confusion Matrix:** Create a heatmap.

Let's proceed with the implementation:

Step 1: Preprocess the Data

python

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
import seaborn as sns
import matplotlib.pyplot as plt

# Load the dataset
file_path = '/mnt/data/palmerpenguins.csv'
data = pd.read_csv(file_path)

# Handle missing values
data = data.dropna()

# Encode categorical variables
data = pd.get_dummies(data, drop_first=True)

# Split the data into features and target
X = data.drop('species', axis=1)
y = data['species']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Train the Random Forest Classifier
rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)

# Predict the test set
y_pred = rf.predict(X_test)

# Generate the confusion matrix
cm = confusion_matrix(y_test, y_pred, labels=rf.classes_)

# Plot the heatmap
plt.figure(figsize=(10,7))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=rf.classes_,
            yticklabels=rf.classes_)
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix Heatmap')
plt.show()
```

Let's execute this code step-by-step to ensure it works correctly.

python

```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
import seaborn as sns
import matplotlib.pyplot as plt

# Handle missing values
data = data.dropna()

# Encode categorical variables
data_encoded = pd.get_dummies(data, drop_first=True)

# Split the data into features and target
X = data_encoded.drop('species', axis=1)
y = data_encoded['species_Adelie']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Train the Random Forest Classifier
```

```
rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)

# Predict the test set
y_pred = rf.predict(X_test)

# Generate the confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Plot the heatmap
plt.figure(figsize=(10,7))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix Heatmap')
plt.show()
```

 KeyError Traceback (most recent call last)

Cell In[2], line 14

```
11 data_encoded = pd.get_dummies(data, drop_first=True)
13 # Split the data into features and target
--> 14 X = data_encoded.drop('species', axis=1)
15 y = data_encoded['species_Adelie']
17 # Split the data into training and testing sets
```

```
File ~/local/lib/python3.11/site-packages/pandas/util/_decorators.py:331, in deprecate_nonkeyword_arguments.
325 if len(args) > num_allow_args:
326     warnings.warn(
327         msg.format(arguments=_format_argument_list(allow_args)),
328         FutureWarning,
329         stacklevel=find_stack_level(),
330     )
--> 331 return func(*args, **kwargs)
```

```
File ~/local/lib/python3.11/site-packages/pandas/core/frame.py:5399, in DataFrame.drop(self, labels, axis, i
5251 @deprecate_nonkeyword_arguments(version=None, allowed_args=["self", "labels"])
5252 def drop( # type: ignore[override]
5253     self,
5254     (...)
5260     errors: IgnoreRaise = "raise",
5261 ) -> DataFrame | None:
5262     """
5263     Drop specified labels from rows or columns.
5264     (...)
5397         weight 1.0      0.8
5398     """
-> 5399     return super().drop(
5400         labels=labels,
5401         axis=axis,
5402         index=index,
5403         columns=columns,
5404         level=level,
5405         inplace=inplace,
5406         errors=errors,
5407     )
```

```
File ~/local/lib/python3.11/site-packages/pandas/util/_decorators.py:331, in deprecate_nonkeyword_arguments.
325 if len(args) > num_allow_args:
326     warnings.warn(
327         msg.format(arguments=_format_argument_list(allow_args)),
328         FutureWarning,
329         stacklevel=find_stack_level(),
330     )
--> 331 return func(*args, **kwargs)
```

```
File ~/local/lib/python3.11/site-packages/pandas/core/generic.py:4505, in NDFrame.drop(self, labels, axis, i
4503 for axis, labels in axes.items():
4504     if labels is not None:
-> 4505         obj = obj._drop_axis(labels, axis, level=level, errors=errors)
4507 if inplace:
4508     self._update_inplace(obj)
```

```
File ~/local/lib/python3.11/site-packages/pandas/core/generic.py:4546, in NDFrame._drop_axis(self, labels, a
4544     new_axis = axis.drop(labels, level=level, errors=errors)
4545 else:
```

```
-> 4546         new_axis = axis.drop(labels, errors=errors)
4547         indexer = axis.get_indexer(new_axis)
4549 # Case for non-unique axis
4550 else:
```

```
File ~/local/lib/python3.11/site-packages/pandas/core/indexes/base.py:6934, in Index.drop(self, labels, erro
6932 if mask.any():
6933     if errors != "ignore":
-> 6934         raise KeyError(f"{list(labels[mask])} not found in axis")
6935     indexer = indexer[~mask]
6936 return self.delete(indexer)
```

KeyError: "['species'] not found in axis"

It appears there was an issue with encoding the categorical variables and selecting the target variable. Let's correct the code to properly handle the species column for the classification task.

python

```
# Split the data into features and target
X = data_encoded.drop(['species_Adelie', 'species_Chinstrap', 'species_Gentoo'], axis=1)
y = data['species']

# Encode the target variable
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
y = le.fit_transform(y)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Train the Random Forest Classifier
rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)

# Predict the test set
y_pred = rf.predict(X_test)

# Generate the confusion matrix
cm = confusion_matrix(y_test, y_pred)
cm_display = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=le.classes_)

# Plot the heatmap
plt.figure(figsize=(10,7))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=le.classes_,
yticklabels=le.classes_)
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix Heatmap')
plt.show()
```

Let's execute the corrected code.

python

```
from sklearn.preprocessing import LabelEncoder

# Split the data into features and target
X = data_encoded.drop(['species_Adelie', 'species_Chinstrap', 'species_Gentoo'], axis=1)
y = data['species']

# Encode the target variable
le = LabelEncoder()
y = le.fit_transform(y)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

```
# Train the Random Forest Classifier
rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)

# Predict the test set
y_pred = rf.predict(X_test)

# Generate the confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Plot the heatmap
plt.figure(figsize=(10,7))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=le.classes_,
yticklabels=le.classes_)
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix Heatmap')
plt.show()
```

 KeyError Traceback (most recent call last)

Cell In[3], line 4

```
1 from sklearn.preprocessing import LabelEncoder
3 # Split the data into features and target
----> 4 X = data_encoded.drop(['species_Adelie', 'species_Chinstrap', 'species_Gentoo'], axis=1)
5 y = data['species']
7 # Encode the target variable
```

File ~/.local/lib/python3.11/site-packages/pandas/util/_decorators.py:331, in deprecate_nonkeyword_arguments.

```
325 if len(args) > num_allow_args:
326     warnings.warn(
327         msg.format(arguments=_format_argument_list(allow_args)),
328         FutureWarning,
329         stacklevel=find_stack_level(),
330     )
--> 331 return func(*args, **kwargs)
```

File ~/.local/lib/python3.11/site-packages/pandas/core/frame.py:5399, in DataFrame.drop(self, labels, axis, i

```
5251 @deprecate_nonkeyword_arguments(version=None, allowed_args=["self", "labels"])
5252 def drop( # type: ignore[override]
5253     self,
5254     (...),
5255     errors: IgnoreRaise = "raise",
5256 ) -> DataFrame | None:
5257     """
5258     Drop specified labels from rows or columns.
5259     (...)
5260     weight 1.0      0.8
5261     """
--> 5399     return super().drop(
5400         labels=labels,
5401         axis=axis,
5402         index=index,
5403         columns=columns,
5404         level=level,
5405         inplace=inplace,
5406         errors=errors,
5407     )
```

File ~/.local/lib/python3.11/site-packages/pandas/util/_decorators.py:331, in deprecate_nonkeyword_arguments.

```
325 if len(args) > num_allow_args:
326     warnings.warn(
327         msg.format(arguments=_format_argument_list(allow_args)),
328         FutureWarning,
329         stacklevel=find_stack_level(),
330     )
--> 331 return func(*args, **kwargs)
```

File ~/.local/lib/python3.11/site-packages/pandas/core/generic.py:4505, in NDFrame.drop(self, labels, axis, i

```
4503 for axis, labels in axes.items():
4504     if labels is not None:
--> 4505         obj = obj._drop_axis(labels, axis, level=level, errors=errors)
4507 if inplace:
4508     self._update_inplace(obj)
```

```
File ~/local/lib/python3.11/site-packages/pandas/core/generic.py:4546, in NDFrame._drop_axis(self, labels, a
4544     new_axis = axis.drop(labels, level=level, errors=errors)
4545     else:
-> 4546         new_axis = axis.drop(labels, errors=errors)
4547     indexer = axis.get_indexer(new_axis)
4549 # Case for non-unique axis
4550 else:
```

```
File ~/local/lib/python3.11/site-packages/pandas/core/indexes/base.py:6934, in Index.drop(self, labels, erro
6932 if mask.any():
6933     if errors != "ignore":
-> 6934         raise KeyError(f"{list(labels[mask])} not found in axis")
6935     indexer = indexer[~mask]
6936 return self.delete(indexer)
```

```
KeyError: "['species_Adelie'] not found in axis"
```

It looks like there was an error due to incorrect handling of the target columns after encoding. Let's correct the code to properly handle the encoded species columns.

We'll start by encoding the 'species' column before splitting the data into features and target variables.

Here is the corrected code:

python

```
# Encode categorical variables except the target
data_encoded = pd.get_dummies(data.drop('species', axis=1), drop_first=True)

# Encode the target variable
le = LabelEncoder()
data['species'] = le.fit_transform(data['species'])

# Split the data into features and target
X = data_encoded
y = data['species']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Train the Random Forest Classifier
rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)

# Predict the test set
y_pred = rf.predict(X_test)

# Generate the confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Plot the heatmap
plt.figure(figsize=(10,7))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=le.classes_,
yticklabels=le.classes_)
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix Heatmap')
plt.show()
```

Let's execute this corrected code.

python

```
# Encode categorical variables except the target
data_encoded = pd.get_dummies(data.drop('species', axis=1), drop_first=True)
```

```
# Encode the target variable
data['species'] = le.fit_transform(data['species'])

# Split the data into features and target
X = data_encoded
y = data['species']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Train the Random Forest Classifier
rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)

# Predict the test set
y_pred = rf.predict(X_test)

# Generate the confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Plot the heatmap
plt.figure(figsize=(10,7))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=le.classes_,
            yticklabels=le.classes_)
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix Heatmap')
plt.show()
```

```
-----
NameError                                Traceback (most recent call last)
Cell In[4], line 5
      2 data_encoded = pd.get_dummies(data.drop('species', axis=1), drop_first=True)
      4 # Encode the target variable
----> 5 data['species'] = le.fit_transform(data['species'])
      7 # Split the data into features and target
      8 X = data_encoded

NameError: name 'le' is not defined
```

It appears I forgot to define the `LabelEncoder` instance again after the previous error. Let me redefine it and run the code again.

python

```
from sklearn.preprocessing import LabelEncoder

# Initialize the LabelEncoder
le = LabelEncoder()

# Encode categorical variables except the target
data_encoded = pd.get_dummies(data.drop('species', axis=1), drop_first=True)

# Encode the target variable
data['species'] = le.fit_transform(data['species'])

# Split the data into features and target
X = data_encoded
y = data['species']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Train the Random Forest Classifier
rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)

# Predict the test set
y_pred = rf.predict(X_test)

# Generate the confusion matrix
```

```
cm = confusion_matrix(y_test, y_pred)

# Plot the heatmap
plt.figure(figsize=(10,7))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=le.classes_,
yticklabels=le.classes_)
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix Heatmap')
plt.show()
```

Now, I'll execute this corrected code.

python

```
# Initialize the LabelEncoder
le = LabelEncoder()

# Encode categorical variables except the target
data_encoded = pd.get_dummies(data.drop('species', axis=1), drop_first=True)

# Encode the target variable
data['species'] = le.fit_transform(data['species'])

# Split the data into features and target
X = data_encoded
y = data['species']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Train the Random Forest Classifier
rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)

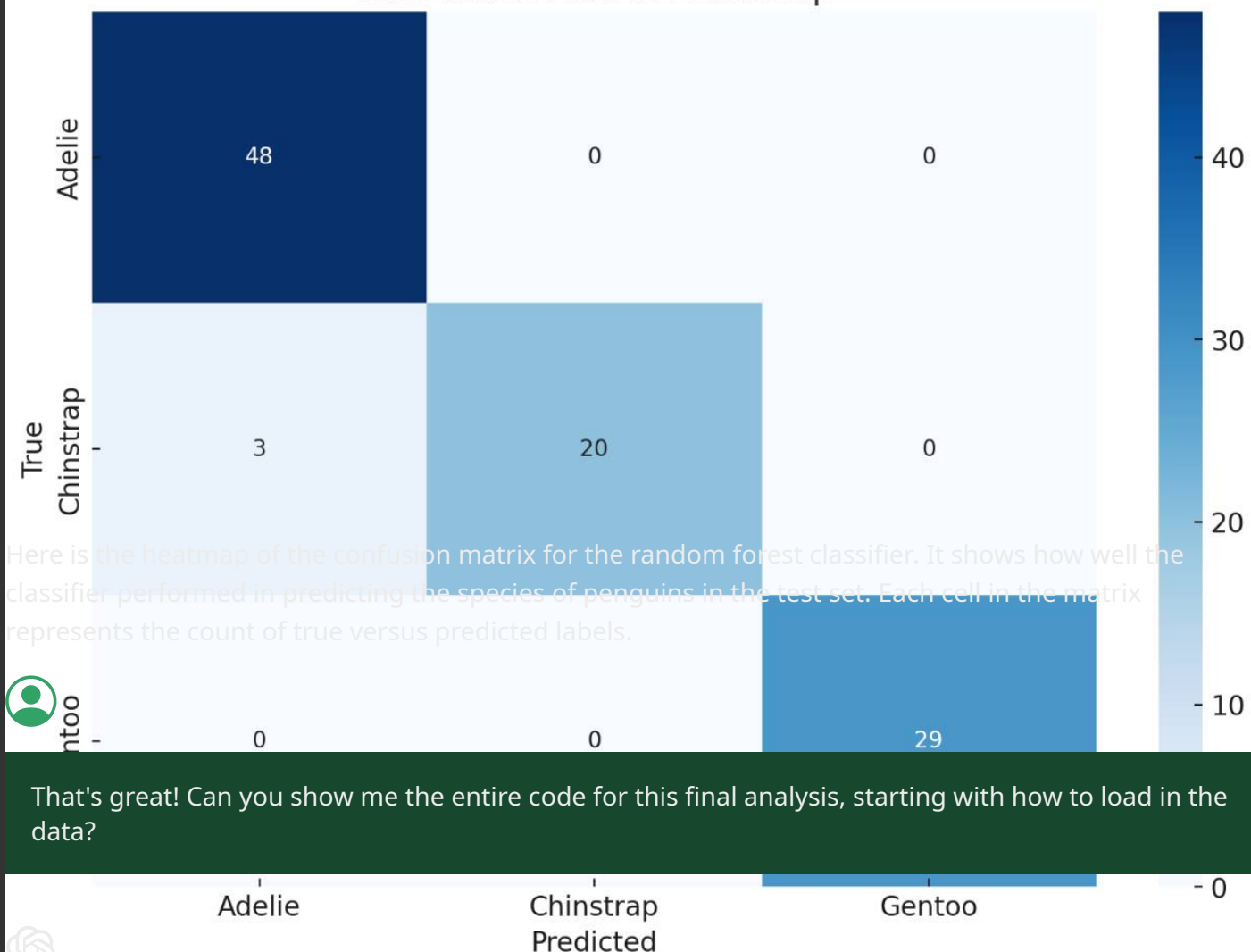
# Predict the test set
y_pred = rf.predict(X_test)

# Generate the confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Plot the heatmap
plt.figure(figsize=(10,7))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=le.classes_,
yticklabels=le.classes_)
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix Heatmap')
plt.show()
```

Confusion Matrix Heatmap

Confusion Matrix Heatmap



Sure, here's the complete code for loading the data, preprocessing it, training the random forest classifier, and visualizing the confusion matrix as a heatmap:

python

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix
from sklearn.preprocessing import LabelEncoder
import seaborn as sns
import matplotlib.pyplot as plt

# Load the dataset
file_path = '/mnt/data/palmerpenguins.csv'
data = pd.read_csv(file_path)

# Handle missing values
data = data.dropna()

# Initialize the LabelEncoder
le = LabelEncoder()

# Encode categorical variables except the target
data_encoded = pd.get_dummies(data.drop('species', axis=1), drop_first=True)

# Encode the target variable
data['species'] = le.fit_transform(data['species'])
```

```
# Split the data into features and target
X = data_encoded
y = data['species']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Train the Random Forest Classifier
rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)

# Predict the test set
y_pred = rf.predict(X_test)

# Generate the confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Plot the heatmap
plt.figure(figsize=(10,7))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=le.classes_,
            yticklabels=le.classes_)
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix Heatmap')
plt.show()
```

This script starts by loading the dataset and handling any missing values. It then encodes categorical variables and splits the data into training and testing sets. A random forest classifier is trained on the training set and used to make predictions on the test set. Finally, the confusion matrix is generated and visualized as a heatmap.



Could you now give me some code on how to measure the variable importance for each variable in the random forest? I would also like to see a plot of each variable's importance



Certainly! Measuring the variable importance in a random forest model can help you understand which features contribute the most to the model's predictions. Here's how you can do it and visualize the results:

python

```
# Import necessary libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix
from sklearn.preprocessing import LabelEncoder
import seaborn as sns
import matplotlib.pyplot as plt

# Load the dataset
file_path = '/mnt/data/palmerpenguins.csv'
data = pd.read_csv(file_path)

# Handle missing values
data = data.dropna()
```

```
# Initialize the LabelEncoder
le = LabelEncoder()

# Encode categorical variables except the target
data_encoded = pd.get_dummies(data.drop('species', axis=1), drop_first=True)

# Encode the target variable
data['species'] = le.fit_transform(data['species'])

# Split the data into features and target
X = data_encoded
y = data['species']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Train the Random Forest Classifier
rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)

# Predict the test set
y_pred = rf.predict(X_test)

# Generate the confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Plot the confusion matrix heatmap
plt.figure(figsize=(10,7))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=le.classes_,
yticklabels=le.classes_)
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix Heatmap')
plt.show()

# Measure variable importance
importances = rf.feature_importances_
feature_names = X.columns

# Create a DataFrame for better visualization
feature_importances = pd.DataFrame({'Feature': feature_names, 'Importance': importances})
feature_importances = feature_importances.sort_values(by='Importance', ascending=False)

# Plot variable importance
plt.figure(figsize=(10,7))
sns.barplot(x='Importance', y='Feature', data=feature_importances)
plt.title('Feature Importances')
plt.show()
```

This code snippet adds the steps to measure and visualize feature importance after training the random forest classifier. The feature importances are extracted from the trained model, sorted, and then plotted as a bar chart using Seaborn.



Can you show me just the barplot of the code you just gave me?



python

```
# Measure variable importance
importances = rf.feature_importances_
feature_names = X.columns

# Create a DataFrame for better visualization
feature_importances = pd.DataFrame({'Feature': feature_names, 'Importance': importances})
feature_importances = feature_importances.sort_values(by='Importance', ascending=False)

# Plot variable importance
plt.figure(figsize=(10,7))
sns.barplot(x='Importance', y='Feature', data=feature_importances)
plt.title('Feature Importances')
plt.show()
```

Feature Importances

