plt.figure(figsize=(8,8))

plt.scatter(x=df["Principal Components 1"], y=df["Principal Components 2"])

plt.xlabel("Principal Components 1")
plt.ylabel("Principal Components 2")
plt.title("Scatter Plot for PCA v/s Species")
colors = ["r", "g", "b"]
targets = ["Iris-setosa", "Iris-versicolor", "Iris-virginica"]
for target, color in zip(targets, colors):
index = y == target
plt.scatter(df.loc[index, df["Principal Component 1"]], df.loc[index, df["Principal Component 2"]], c=color, s=50)
plt.grid(True)
plt.legend(targets)
plt.show()

Correct the code

Certainly! Here's a corrected and improved version of your code for plotting a PCA scatter plot by species using matplotlib and pandas. I'll address the following issues:

1. **Column Names:** Your dataframe columns are inconsistently named (`"Principal Components 1"` vs `"Principal Component 1"`). Choose one convention.
2. **Scatter Function:** The correct way to access dataframe columns is `df["column_name"]`, not `df.loc[index, df["column_name"]]`.
3. **Legend Placement:** Pass the scatter plot handles to the legend for better labeling.
4. **Boolean Indexing:** Ensure `y` is a pandas Series or numpy array with species names.

Here's the corrected code:

```
import matplotlib.pyplot as plt

plt.figure(figsize=(8, 8))
plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")
plt.title("Scatter Plot for PCA vs Species")
colors = ["r", "g", "b"]
targets = ["Iris-setosa", "Iris-versicolor", "Iris-virginica"]

for target, color in zip(targets, colors):
    index = (y == target)
    plt.scatter(
```

```
        df.loc[index, "Principal Component 1"],
        df.loc[index, "Principal Component 2"],
        c=color,
        s=50,
        label=target
    )

plt.grid(True)
plt.legend()
plt.show()
```

Key Points:

- Make sure your dataframe columns are named `"Principal Component 1"` and `"Principal Component 2"`.
- The `label=target` argument in `plt.scatter` automatically creates the legend.
- `index = (y == target)` assumes `y` is a pandas Series or numpy array containing species names.

Let me know if you need help with the PCA code or data preparation!

what is s here

In Matplotlib's `plt.scatter()` function, the parameter `s` specifies the **size of the markers** (the dots) in the scatter plot. The value of `s` is given in **points squared** (points^2), where one point is 1/72 of an inch.

- If you set `s` to a single number (e.g., `s=50`), all markers will have the same size.
- If you provide an array or list to `s`, each marker can have a different size, corresponding to each data point [1] [2] [3] [4].

Example:

```
plt.scatter(x, y, s=100)  # All markers have size 100 points^2
plt.scatter(x, y, s=[20, 40, 60, 80, 100])  # Each marker has a different size
```

Summary:
`s` controls the marker size in a scatter plot, and its value is in points squared [3] [4].

⁂

I have an AI Lab exam tomorrow for which I will have to write a single piece of code depending on what topic I get, from the following set of what all we have done over the past 12 labs. Give me a cheatsheet of essential codes, etc. and make a separate heading for each of them

Absolutely! Here is a **comprehensive Python AI Lab Cheatsheet** built to cover all essential algorithms, code templates, and explanations matching your lab syllabus. You can quickly refer to these snippets and modify them as needed during your exam.

1. Data Visualization and Descriptive Analytics

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Load data
df = pd.read_csv('data.csv')

# Descriptive statistics
print(df.describe())

# Histogram
df['column_name'].hist()
plt.xlabel('column_name')
plt.ylabel('Frequency')
plt.title('Histogram of column_name')
plt.show()

# Boxplot
sns.boxplot(x='label_column', y='feature_column', data=df)
plt.title('Boxplot')
plt.show()

# Correlation matrix
sns.heatmap(df.corr(), annot=True, cmap='coolwarm')
plt.title('Correlation Matrix')
plt.show()
```

2. Data Preprocessing: Handling Missing Values

```
# Check for missing values
print(df.isnull().sum())

# Fill missing values with mean/median/mode
df['column'] = df['column'].fillna(df['column'].mean())
```

```
# Drop rows with missing values
df = df.dropna()
```

3. Outlier Detection, Normalization, and Standardization

```
from sklearn.preprocessing import MinMaxScaler, StandardScaler

# Detect outliers (Z-score method)
from scipy import stats
z_scores = stats.zscore(df['column'])
print(df[(z_scores &gt; 3) | (z_scores &lt; -3)])

# Remove outliers
df = df[(z_scores &lt;= 3) &amp; (z_scores &gt;= -3)]

# Normalization
scaler = MinMaxScaler()
df[['col1','col2']] = scaler.fit_transform(df[['col1','col2']])

# Standardization
scaler = StandardScaler()
df[['col1','col2']] = scaler.fit_transform(df[['col1','col2']])
```

4. Dimensionality Reduction: PCA

```
from sklearn.decomposition import PCA

X = df.drop('target', axis=1)
pca = PCA(n_components=2)
principalComponents = pca.fit_transform(X)
df_pca = pd.DataFrame(data=principalComponents, columns=['PC1', 'PC2'])

# Plot
plt.scatter(df_pca['PC1'], df_pca['PC2'], c=df['target'].map({'Iris-setosa':0, 'Iris-vers
plt.xlabel('PC1')
plt.ylabel('PC2')
plt.title('PCA Scatter Plot')
plt.show()
```

5. Classification: Train-Test Split, Model Training, and Prediction

```
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report

# Prepare data
X = df.drop('target', axis=1)
y = df['target']
```

```
# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# KNN Classifier
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)

print('Accuracy:', accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

6. Classification: SVM and Decision Tree

```
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier

# Support Vector Machine
svm = SVC()
svm.fit(X_train, y_train)
print("SVM Accuracy:", svm.score(X_test, y_test))

# Decision Tree
dt = DecisionTreeClassifier()
dt.fit(X_train, y_train)
print("Decision Tree Accuracy:", dt.score(X_test, y_test))
```

7. Regression: Linear and Polynomial

```
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures

# Linear Regression
linreg = LinearRegression()
linreg.fit(X_train, y_train)
y_pred = linreg.predict(X_test)

# Polynomial Regression
poly = PolynomialFeatures(degree=2)
X_poly = poly.fit_transform(X_train)
linreg_poly = LinearRegression()
linreg_poly.fit(X_poly, y_train)
y_poly_pred = linreg_poly.predict(poly.transform(X_test))
```

8. Regression: Metrics

```
from sklearn.metrics import mean_squared_error, r2_score

print("MSE:", mean_squared_error(y_test, y_pred))
print("R^2 Score:", r2_score(y_test, y_pred))
```

9. Clustering: KMeans

```
from sklearn.cluster import KMeans

kmeans = KMeans(n_clusters=3)
kmeans.fit(X)
labels = kmeans.labels_

# Plot clusters
plt.scatter(X.iloc[:,0], X.iloc[:,1], c=labels)
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.title('KMeans Clustering')
plt.show()
```

10. Clustering: Hierarchical (Agglomerative)

```
from sklearn.cluster import AgglomerativeClustering

agg = AgglomerativeClustering(n_clusters=3)
labels = agg.fit_predict(X)

plt.scatter(X.iloc[:,0], X.iloc[:,1], c=labels)
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.title('Agglomerative Clustering')
plt.show()
```

11. General: Model Evaluation and Cross-Validation

```
from sklearn.model_selection import cross_val_score
scores = cross_val_score(knn, X, y, cv=5)
print("Cross-validation scores:", scores)
print("Average CV score:", scores.mean())
```

12. Tips for Exam

- Always **import the required libraries** at the top.
- Use `df.head()` and `df.info()` to inspect data quickly.
- For classification/regression, always split data and check accuracy/metrics.
- For visualizations, `plt.show()` at the end.
- For **clustering**, remember to plot with predicted labels.
- Don't forget to **label axes and title your plots**!

Good luck! You got this!
Need quick help with any specific topic? Just ask!

⁂

This is good, thanks, now I want a cheatsheet for using matplotlib.pyplot and plotting various stuff especially something like classification and all, which would require for loops, etc in their graphs

Here's a **targeted Matplotlib cheatsheet** focused on classification/regression/clustering visualization with loops, color-coding, and common plot types you'll need for your AI lab exam:

1. Basic Plot Setup

```
import matplotlib.pyplot as plt
plt.figure(figsize=(8, 6))  # Set figure size
plt.xlabel("Feature X")      # Always label axes!
plt.ylabel("Feature Y")
plt.title("Meaningful Title")
plt.grid(True)
plt.legend()                # Add legend when multiple series
plt.show()                  # Display plot
```

2. Classification Plots

2.1 Scatter Plot with Class Colors (For-Loop)

```
# Assume X1, X2 are features, y contains class labels (0,1,2)
classes = ['Class 0', 'Class 1', 'Class 2']
colors = ['red', 'green', 'blue']
markers = ['o', 's', '^']

for cls, color, marker in zip(classes, colors, markers):
    plt.scatter(
```

```
        X1[y == cls],
        X2[y == cls],
        c=color,
        marker=marker,
        label=cls,
        s=60                # Marker size
    )

plt.legend()
plt.show()
```

2.2 Decision Boundary Plot

```
# Create meshgrid for decision boundaries
xx, yy = np.meshgrid(
    np.linspace(X1.min()-1, X1.max()+1, 100),
    np.linspace(X2.min()-1, X2.max()+1, 100)
)

# Predict for all grid points
Z = model.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

# Plot contour
plt.contourf(xx, yy, Z, alpha=0.3, cmap='Paired')
plt.scatter(X1, X2, c=y, cmap='viridis', edgecolor='k')
plt.show()
```

3. Clustering Visualization

3.1 KMeans Clusters with Centroids

```
# After fitting KMeans
plt.scatter(X[:,0], X[:,1], c=labels, cmap='tab10')
plt.scatter(
    kmeans.cluster_centers_[:,0],
    kmeans.cluster_centers_[:,1],
    s=200,                   # Size of centroids
    marker='*',              # Star shape
    c='black',
    label='Centroids'
)
plt.legend()
plt.show()
```

3.2 Hierarchical Clustering (Dendrogram)

```
from scipy.cluster.hierarchy import dendrogram, linkage

Z = linkage(X, method='ward')
plt.figure(figsize=(10, 5))
dendrogram(Z)
plt.axhline(y=threshold, color='r', linestyle='--')  # Cutoff line
plt.show()
```

4. Regression Plots

4.1 Linear Regression Line

```
plt.scatter(X, y, color='blue', label='Data')
plt.plot(
    X,
    lin_reg.predict(X),
    color='red',
    linewidth=2,
    label='Regression Line'
)
plt.legend()
plt.show()
```

4.2 Polynomial Regression Curve

```
X_grid = np.arange(min(X), max(X), 0.1).reshape(-1,1)
plt.scatter(X, y, color='blue')
plt.plot(
    X_grid,
    poly_reg.predict(poly.fit_transform(X_grid)),
    color='green'
)
plt.show()
```

5. Advanced Styling Tips

```
# Customize markers/colors per class in loop
for i in range(n_classes):
    plt.scatter(
        X_features[y == i, 0],
        X_features[y == i, 1],
        alpha=0.5,          # Transparency (0-1)
        edgecolors='w',     # Marker border color
        linewidths=0.5,     # Border thickness
        label=f'Class {i}'
```

```
    )

# Add annotations
plt.text(
    x=0.5, y=0.5,
    s='Important Note',
    fontsize=12,
    bbox=dict(facecolor='yellow', alpha=0.5)
)

# Save high-res image
plt.savefig('plot.png', dpi=300, bbox_inches='tight')
```

6. Essential Parameters Cheat Sheet

Component	Key Parameters
`plt.scatter()`	`c`, `s`, `marker`, `alpha`, `label`
`plt.plot()`	`linestyle`, `linewidth`, `color`
`plt.legend()`	`loc='best'`, `fontsize`, `shadow=True`
`plt.title()`	`fontsize=14`, `pad=20` (title padding)

Exam-Ready Checklist:

1. Always set `figsize` before plotting
2. Use loops to handle multiple classes/clusters
3. Add `label` in plot commands for auto-legend
4. Set `alpha` for overlapping points
5. Include axis labels and title
6. Use `plt.tight_layout()` if elements overlap

Need a specific visualization example? Let me know!

⁂

Next, there is this topic of implementing a neural network, give me a cheat sheet for that using pytorch.

Here’s a concise **PyTorch Neural Network Cheatsheet** tailored for your AI Lab exam, covering model setup, training loops, and key components:

1. Basic Neural Network Setup

```
import torch
import torch.nn as nn
import torch.optim as optim

# Define a neural network
class NeuralNetwork(nn.Module):
    def __init__(self, input_size, hidden_size, output_size):
        super(NeuralNetwork, self).__init__()
        self.layer1 = nn.Linear(input_size, hidden_size)
        self.relu = nn.ReLU()
        self.layer2 = nn.Linear(hidden_size, output_size)

    def forward(self, x):
        x = self.layer1(x)
        x = self.relu(x)
        x = self.layer2(x)
        return x

# Initialize model (example: 4 input features, 5 hidden neurons, 3 output classes)
model = NeuralNetwork(input_size=4, hidden_size=5, output_size=3)
```

2. Loss Function & Optimizer

```
# For classification (e.g., CrossEntropyLoss)
criterion = nn.CrossEntropyLoss()
# For regression (e.g., MSELoss)
# criterion = nn.MSELoss()

optimizer = optim.Adam(model.parameters(), lr=0.001)
```

3. Training Loop Template

```
epochs = 100

for epoch in range(epochs):
    # Forward pass
    outputs = model(X_train_tensor)  # X_train_tensor: input data tensor
    loss = criterion(outputs, y_train_tensor)  # y_train_tensor: labels tensor

    # Backward pass and optimize
    optimizer.zero_grad()  # Clear existing gradients
    loss.backward()        # Backpropagation
    optimizer.step()       # Update weights

    # Print progress
    if (epoch+1) % 10 == 0:
        print(f'Epoch [{epoch+1}/{epochs}], Loss: {loss.item():.4f}')
```

4. Evaluation (Classification Example)

```
with torch.no_grad():
    outputs = model(X_test_tensor)
    _, predicted = torch.max(outputs.data, 1)  # Get class labels
    accuracy = (predicted == y_test_tensor).sum().item() / y_test_tensor.size(0)
    print(f'Test Accuracy: {accuracy * 100:.2f}%')
```

5. Data Handling

Convert Pandas DataFrame/Numpy to Tensor:

```
X_train_tensor = torch.tensor(X_train.values, dtype=torch.float32)
y_train_tensor = torch.tensor(y_train.values, dtype=torch.long)  # long for CrossEntropy
```

DataLoader for Batches:

```
from torch.utils.data import DataLoader, TensorDataset

dataset = TensorDataset(X_train_tensor, y_train_tensor)
dataloader = DataLoader(dataset, batch_size=32, shuffle=True)

# Usage in training loop:
for inputs, labels in dataloader:
    outputs = model(inputs)
    # ... rest of training steps ...
```

6. Common Activation Functions

```
nn.ReLU()     # Most common
nn.Sigmoid()  # For binary classification
nn.Softmax()  # For multi-class output (use dim=1)
```

7. Tips for Exam

1. **Device Configuration** (CPU/GPU):

   ```
   device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
   model = NeuralNetwork(...).to(device)
   ```

2. **Save/Load Model**:

   ```
   torch.save(model.state_dict(), 'model.pth')
   model.load_state_dict(torch.load('model.pth'))
   ```

3. **Debugging**:
   - Check tensor shapes with `.shape`
   - Use `print(model)` to see layer architecture

Example: Regression Network

```
class RegressionNN(nn.Module):
    def __init__(self):
        super().__init__()
        self.layer1 = nn.Linear(1, 10)  # 1 input feature
        self.layer2 = nn.Linear(10, 1)  # 1 output

    def forward(self, x):
        x = torch.relu(self.layer1(x))
        x = self.layer2(x)
        return x

# Loss: nn.MSELoss()
```

Good luck! Let me know if you need help with specific architectures (CNNs, RNNs) or advanced topics like dropout or batch normalization!

⁂

1. https://www.codecademy.com/resources/docs/matplotlib/pyplot/scatter
2. https://www.tutorialspoint.com/how-to-adjust-marker-size-in-matplotlib
3. https://www.geeksforgeeks.org/customizing-marker-size-in-pyplot-scatter-plots/
4. https://www.altcademy.com/blog/how-to-change-marker-size-in-scatter-plot-matplotlib/