Python Programming for Machine Learning

NumPy, Pandas, Matplotlib, Scikit-learn Introduction

Sarwan Ali

Department of Computer Science
Georgia State University

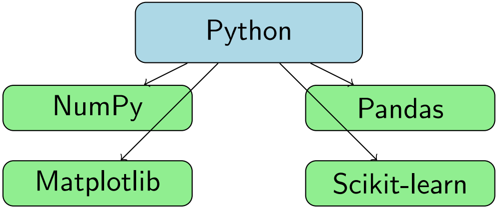
Building the Foundation for ML Programming

Today's Programming Journey

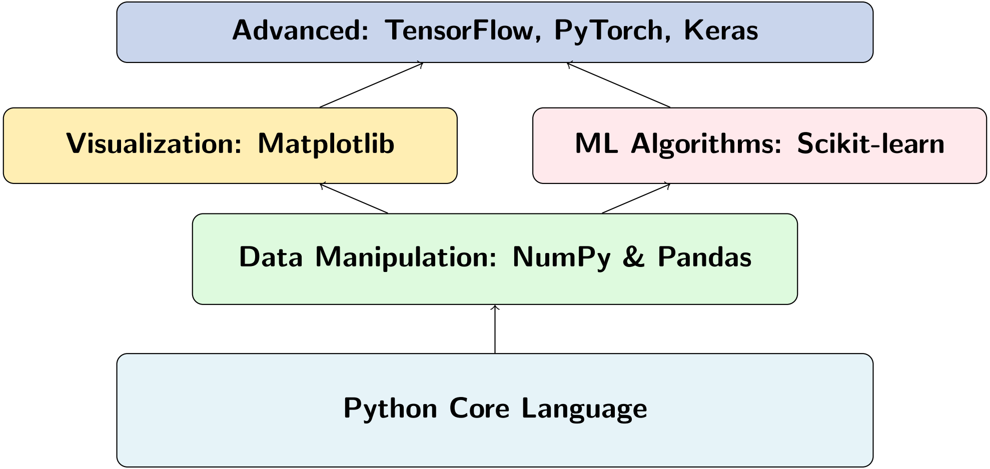
Why Python for Machine Learning?

Key Advantages:

- Simple & Readable Syntax
- Rich Ecosystem of ML libraries
- Interactive Development (Jupyter)
- Strong Community support
- Cross-platform compatibility



ML Python Stack Overview



NumPy: The Foundation of Scientific Computing

What is NumPy?

- Numerical Python - fundamental package for scientific computing
- Provides support for multi-dimensional arrays and matrices
- High-performance mathematical functions
- Foundation for most ML libraries

Key Features:

- N-dimensional arrays (ndarray)
- Broadcasting capabilities
- Linear algebra operations
- Random number generation
- Integration with C/C++/Fortran
- Memory efficient operations

NumPy Arrays: Creation and Basic Operations

Array Creation:

```
import numpy as np

# Creating arrays
arr1 = np.array([1, 2, 3, 4, 5])
arr2 = np.zeros((3, 4))              # 3x4 array of zeros
arr3 = np.ones((2, 3))               # 2x3 array of ones
arr4 = np.arange(0, 10, 2)           # [0, 2, 4, 6, 8]
arr5 = np.linspace(0, 1, 5)          # 5 points from 0 to 1
arr6 = np.random.random((2, 3))       # Random 2x3 array
```

Array Properties:

```
print(arr1.shape)      # (5,)
print(arr2.dtype)      # float64
print(arr2.ndim)       # 2
print(arr2.size)       # 12
```

NumPy: Mathematical Operations

Element-wise Operations:

```
a = np.array([1, 2, 3, 4])
b = np.array([5, 6, 7, 8])

# Arithmetic operations
c = a + b                # [6, 8, 10, 12]
d = a * b                # [5, 12, 21, 32]
e = np.sqrt(a)           # Square root
f = np.exp(a)            # Exponential
```

NumPy: Mathematical Operations (Continued)

Matrix Operations:

```
A = np.array([[1, 2], [3, 4]])
B = np.array([[5, 6], [7, 8]])

# Matrix multiplication
C = np.dot(A, B)   # or A @ B
# Transpose
A_T = A.T
# Inverse
A_inv = np.linalg.inv(A)
```

NumPy: Advanced Operations for ML

Statistical Operations:

```
data = np.random.normal(0, 1, (100, 5))  # 100x5 normal data

# Statistical measures
mean = np.mean(data, axis=0)     # Mean along columns
std = np.std(data, axis=0)       # Standard deviation
corr = np.corrcoef(data.T)       # Correlation matrix
```

Array Manipulation:

```
# Reshaping
arr = np.arange(12).reshape(3, 4)
# Indexing and slicing
subset = arr[1:3, 0:2]
# Boolean indexing
mask = arr > 5
filtered = arr[mask]
```

Pandas: Data Analysis Made Easy

What is Pandas?

- Python Data Analysis Library
- Built on top of NumPy
- Provides DataFrame and Series data structures
- Excel for Python programmers

Key Data Structures:

- Series: 1D labeled array
- DataFrame: 2D labeled data structure

Core Capabilities:

- Data cleaning & preparation
- Missing data handling
- Group operations
- Data merging & joining
- Time series analysis

Pandas: DataFrame Creation and Basic Operations

Creating DataFrames:

```
import pandas as pd

# From dictionary
data = {'Name': ['Alice', 'Bob', 'Charlie'],
        'Age': [25, 30, 35],
        'City': ['NY', 'LA', 'Chicago']}
df = pd.DataFrame(data)

# From CSV file
df = pd.read_csv('data.csv')

# From NumPy array
arr = np.random.randn(100, 4)
df = pd.DataFrame(arr, columns=['A', 'B', 'C', 'D'])
```

Pandas: DataFrame Creation and Basic Operations (Continued)

Basic Information:

```
df.head()          # First 5 rows
df.info()          # DataFrame info
df.describe()      # Statistical summary
df.shape           # Dimensions
```

Pandas: Data Selection and Filtering

Data Selection:

```
# Column selection
ages = df['Age']              # Single column (Series)
subset = df[['Name', 'Age']]  # Multiple columns
# Row selection
first_row = df.iloc[0]        # By position
alice = df.loc[0]             # By label
# Conditional filtering
adults = df[df['Age'] > 30]
ny_people = df[df['City'] == 'NY']
complex_filter = df[(df['Age'] > 25) & (df['City'] == 'LA')]
```

Pandas: Data Selection and Filtering (Continued)

Data Modification:

```
# Adding new column
df['Age_Group'] = df['Age'].apply(lambda x: 'Young' if x < 30 else '
    Old')
# Modifying existing data
df.loc[df['City'] == 'NY', 'City'] = 'New York'
```

Pandas: Data Cleaning and Preparation

Handling Missing Data:

```
# Check for missing values
df.isnull().sum()
# Drop missing values
df_clean = df.dropna()
# Fill missing values
df_filled = df.fillna(df.mean())  # Fill with mean
df_filled = df.fillna(method='forward')  # Forward fill
```

Data Transformation:

```
# Group operations
grouped = df.groupby('City')['Age'].mean()
# Pivot tables
pivot = df.pivot_table(values='Age', index='City', aggfunc='mean')
# Data merging
merged = pd.merge(df1, df2, on='common_column')
```

Matplotlib: Bringing Data to Life

What is Matplotlib?

- Comprehensive plotting library for Python
- MATLAB-like interface through pyplot
- Creates publication-quality figures
- Foundation for other plotting libraries (Seaborn, Plotly)

Plot Types:

- Line plots
- Scatter plots
- Bar charts
- Histograms
- Box plots
- Heatmaps

Key Features:

- Customizable styling
- Multiple subplot support
- Interactive capabilities
- Export to various formats
- Integration with Pandas

Simple Line Plot:

```python
import matplotlib.pyplot as plt

# Basic line plot
x = np.linspace(0, 10, 100)
y = np.sin(x)

plt.figure(figsize=(8, 6))
plt.plot(x, y, label='sin(x)')
plt.xlabel('X values')
plt.ylabel('Y values')
plt.title('Sine Wave')
plt.legend()
plt.grid(True)
plt.show()
```

Matplotlib: Basic Plotting (Continued)

Scatter Plot:

```
plt.figure(figsize=(8, 6))
plt.scatter(x_data, y_data, c=colors, alpha=0.7)
plt.colorbar()
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.title('Data Distribution')
```

Matplotlib: Multiple Plots and Subplots

Subplots:

```
fig, axes = plt.subplots(2, 2, figsize=(12, 8))

# Plot 1: Line plot
axes[0, 0].plot(x, y1)
axes[0, 0].set_title('Line Plot')
# Plot 2: Scatter plot
axes[0, 1].scatter(x, y2)
axes[0, 1].set_title('Scatter Plot')
# Plot 3: Histogram
axes[1, 0].hist(data, bins=20)
axes[1, 0].set_title('Histogram')
# Plot 4: Bar plot
axes[1, 1].bar(categories, values)
axes[1, 1].set_title('Bar Plot')
plt.tight_layout()
plt.show()
```

Matplotlib: Statistical Visualizations

Box Plot for Distribution Analysis:

```
# Box plot
plt.figure(figsize=(10, 6))
plt.boxplot([group1, group2, group3],
            labels=['Group 1', 'Group 2', 'Group 3'])
plt.ylabel('Values')
plt.title('Distribution Comparison')
```

Heatmap for Correlation Matrix:

```
# Correlation heatmap
correlation_matrix = df.corr()
plt.figure(figsize=(8, 6))
plt.imshow(correlation_matrix, cmap='coolwarm', aspect='auto')
plt.colorbar()
plt.xticks(range(len(df.columns)), df.columns, rotation=45)
plt.yticks(range(len(df.columns)), df.columns)
plt.title('Feature Correlation Matrix')
```

Scikit-learn: Your ML Toolkit

What is Scikit-learn?

- Machine Learning library built on NumPy, SciPy, and Matplotlib
- Simple and efficient tools for data mining and analysis
- Consistent API across all algorithms
- Extensive documentation and examples

ML Categories:

- Supervised Learning
- Unsupervised Learning
- Model Selection
- Preprocessing

Key Modules:

- Classification & Regression
- Clustering
- Dimensionality Reduction
- Model Evaluation
- Feature Selection

Scikit-learn: Algorithm Categories

Supervised Learning: Linear Regression, Logistic Regression, Decision Trees, Random Forest

Unsupervised Learning: K-Means, Hierarchical, DBSCAN, PCA, t-SNE

Model Evaluation: Cross-validation, Metrics, Grid Search

Preprocessing: Scaling, Encoding, Feature Selection

Scikit-learn: Basic Workflow

Standard ML Workflow:

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report

# 1. Load and prepare data
X, y = load_data()  # Features and target

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

# 3. Scale features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Scikit-learn: Basic Workflow (Continued)

```
# 4. Train model
model = LogisticRegression()
model.fit(X_train_scaled, y_train)

# 5. Make predictions and evaluate
y_pred = model.predict(X_test_scaled)
accuracy = accuracy_score(y_test, y_pred)
```

Scikit-learn: Data Preprocessing

Feature Scaling:

```python
from sklearn.preprocessing import StandardScaler, MinMaxScaler

# Standardization (z-score normalization)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Min-Max scaling (0-1 range)
minmax_scaler = MinMaxScaler()
X_minmax = minmax_scaler.fit_transform(X)
```

Scikit-learn: Data Preprocessing (Continued)

Encoding Categorical Variables:

```
from sklearn.preprocessing import LabelEncoder, OneHotEncoder

# Label encoding
label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y_categorical)

# One-hot encoding
onehot_encoder = OneHotEncoder(sparse=False)
X_onehot = onehot_encoder.fit_transform(X_categorical)
```

Cross-Validation:

```
from sklearn.model_selection import cross_val_score, GridSearchCV

# K-fold cross-validation
scores = cross_val_score(model, X, y, cv=5, scoring='accuracy')
print(f"CV Accuracy: {scores.mean():.3f} (+/- {scores.std()*2:.3f})"
    )
```

Hyperparameter Tuning:

```
# Grid search for best parameters
param_grid = {'C': [0.1, 1, 10], 'gamma': [0.001, 0.01, 0.1]}
grid_search = GridSearchCV(SVC(), param_grid, cv=5)
grid_search.fit(X_train, y_train)

print(f"Best parameters: {grid_search.best_params_}")
print(f"Best score: {grid_search.best_score_:.3f}")
```

Scikit-learn: Model Selection and Evaluation (Continued)

Performance Metrics:

```
from sklearn.metrics import accuracy_score, precision_score,
    recall_score

accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')
```

Complete ML Pipeline Example

End-to-End Example:

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix

# 1. Load data using pandas
iris = load_iris()
df = pd.DataFrame(iris.data, columns=iris.feature_names)
df['target'] = iris.target
```

Complete ML Pipeline Example (Continued)

End-to-End Example:

```
# 2. Explore data
print(df.head())
print(df.describe())

# 3. Visualize data
plt.figure(figsize=(10, 6))
for i, target in enumerate(iris.target_names):
    plt.scatter(df[df['target']==i]['sepal length (cm)'],
                df[df['target']==i]['sepal width (cm)'],
                label=target)
plt.xlabel('Sepal Length')
plt.ylabel('Sepal Width')
plt.legend()
plt.title('Iris Dataset Visualization')
plt.show()
```

Complete ML Pipeline Example (Continued)

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

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

# 6. Scale features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# 7. Train model
model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(X_train_scaled, y_train)
```

Complete ML Pipeline Example (Continued)

```
# 8. Evaluate model
y_pred = model.predict(X_test_scaled)
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.3f}")

# 9. Visualize results
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8, 6))
plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)
plt.title('Confusion Matrix')
plt.colorbar()
plt.show()
```

Best Practices and Tips

Code Organization:

- Use Jupyter notebooks for exploration
- Create functions for reusable code
- Follow PEP 8 style guidelines
- Document your code thoroughly

Data Handling:

- Always explore data first
- Handle missing values appropriately
- Validate data types and ranges
- Keep raw data unchanged

ML Pipeline:

- Use train/validation/test splits
- Always scale features for ML
- Use cross-validation
- Monitor for overfitting

Performance:

- Vectorize operations with NumPy
- Use built-in pandas functions
- Profile code for bottlenecks
- Consider memory usage

Data Leakage:

```
# WRONG: Scaling before splitting
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
X_train, X_test = train_test_split(X_scaled, y)

# CORRECT: Scaling after splitting
X_train, X_test, y_train, y_test = train_test_split(X, y)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)  # Only transform!
```

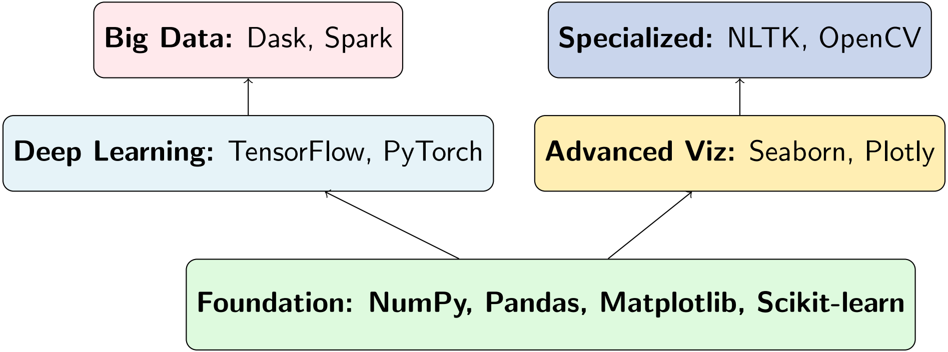
Common Pitfalls and How to Avoid Them (2/2)

Memory Issues with Large Datasets:

```
# Use chunking for large CSV files
chunk_size = 10000
for chunk in pd.read_csv('large_file.csv', chunksize=chunk_size):
    process_chunk(chunk)

# Use efficient data types
df['category'] = df['category'].astype('category')
df['int_col'] = pd.to_numeric(df['int_col'], downcast='integer')
```

Next Steps: Advanced Topics



Recommended Learning Path:

- Master the basics: NumPy $\rightarrow$ Pandas $\rightarrow$ Matplotlib $\rightarrow$ Scikit-learn
- Practice with real datasets (Kaggle, UCI ML Repository)
- Learn specialized libraries based on your domain
- Contribute to open-source projects

Hands-on Exercise: Your First ML Project

Project: Predicting House Prices

Tasks:

1. Load data using pandas from 'housing.csv'
2. Explore the dataset:
   - Check data types and missing values
   - Create summary statistics
   - Visualize key relationships
3. Preprocess the data:
   - Handle missing values
   - Scale numerical features
   - Encode categorical variables
4. Build and evaluate models:
   - Split data into train/test sets
   - Train Linear Regression and Random Forest
   - Compare performance using RMSE
5. Visualize results and feature importance

Time: 45 minutes — **Tools:** Jupyter Notebook

Resources for Continued Learning

Official Documentation:

- NumPy: numpy.org
- Pandas: pandas.pydata.org
- Matplotlib: matplotlib.org
- Scikit-learn: scikit-learn.org

Books:

- "Python for Data Analysis" by Wes McKinney
- "Hands-On Machine Learning" by Aurélien Géron
- "Python Machine Learning" by Sebastian Raschka

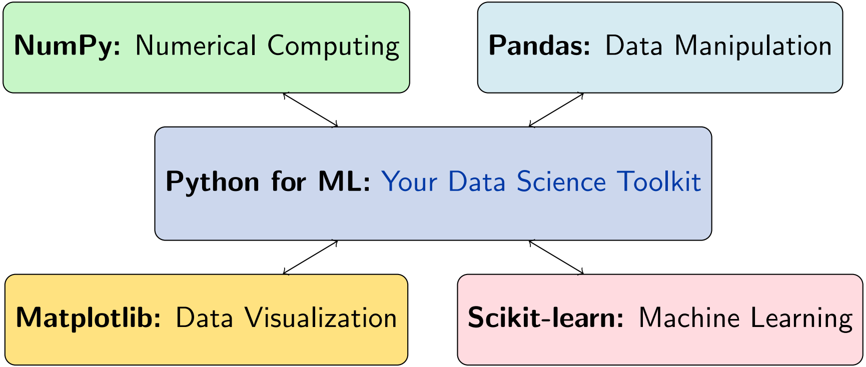
Online Platforms:

- Kaggle: Competitions and datasets
- Google Colab: Free Jupyter environment
- GitHub: Open-source projects
- Stack Overflow: Community help

Practice Datasets:

- UCI ML Repository
- Seaborn built-in datasets
- Scikit-learn toy datasets
- Kaggle Learn micro-courses

Summary: Python for Machine Learning



Key Takeaways:

- Python provides a comprehensive ecosystem for ML
- These libraries work seamlessly together
- Master the basics before moving to advanced topics
- Practice with real projects to solidify learning

Thank You!

Questions & Discussion

Happy Coding with Python!

Next Session: Data Preprocessing: Data cleaning, handling missing values