Python Complete Cheat Sheet: Beginner to ML Professional

Table of Contents

Python Fundamentals

Variables and Data Types

python

```python
# Variables (no declaration needed)
x = 5                    # int
y = 3.14                 # float
name = "Python"          # str
is_valid = True          # bool
nothing = None           # NoneType

# Type conversion
int("123")               # String to integer
float("3.14")            # String to float
str(123)                 # Number to string
bool(1)                  # True (0 is False)
list("abc")              # ['a', 'b', 'c']

# Multiple assignment
a, b, c = 1, 2, 3
x = y = z = 0
```

String Operations

```python
# String creation
s1 = 'single quotes'
s2 = "double quotes"
s3 = '''multi-line
        string'''
s4 = f"formatted {x}"    # f-string (Python 3.6+)

# String methods
s.upper()                # UPPERCASE
s.lower()                # lowercase
s.strip()                # Remove whitespace
s.split(',')             # Split by delimiter
s.replace('old', 'new')  # Replace substring
s.startswith('pre')      # Check prefix
s.endswith('suf')        # Check suffix
'_'.join(['a','b','c'])  # 'a_b_c'

# String formatting
"{} {}".format("Hello", "World")
"{1} {0}".format("World", "Hello")
"{name} is {age}".format(name="John", age=30)
f"{name} is {age}"       # f-string (preferred)

# String slicing
s = "Python"
s[0]                     # 'P'
s[-1]                    # 'n'
s[1:4]                   # 'yth'
s[:3]                    # 'Pyt'
s[3:]                    # 'hon'
s[::2]                   # 'Pto' (step by 2)
s[::-1]                  # 'nohtyP' (reverse)
```

Operators

```python
# Arithmetic
+, -, *, /               # Basic operations
//                       # Floor division (5//2 = 2)
%                        # Modulo (5%2 = 1)
**                       # Exponentiation (2**3 = 8)

# Comparison
==, !=                   # Equal, not equal
<, >, <=, >=            # Less/greater than
is, is not              # Identity comparison
in, not in              # Membership

# Logical
and, or, not            # Boolean operators

# Bitwise
&, |, ^                 # AND, OR, XOR
~                       # NOT
<<, >>                  # Left/right shift

# Assignment
+=, -=, *=, /=, //=, %=, **=
```

Data Structures

Lists

python

```
# Creation
lst = [1, 2, 3]
lst = list(range(5))     # [0, 1, 2, 3, 4]

# Methods
lst.append(4)            # Add to end
lst.extend([5, 6])       # Add multiple
lst.insert(0, 0)         # Insert at index
lst.remove(3)            # Remove first occurrence
lst.pop()                # Remove & return last
lst.pop(0)               # Remove & return at index
lst.clear()              # Remove all
lst.index(2)             # Find index of value
lst.count(2)             # Count occurrences
lst.sort()               # Sort in place
lst.reverse()            # Reverse in place
sorted(lst)              # Return sorted copy
len(lst)                 # Length

# List comprehension
[x**2 for x in range(5)]                    # [0, 1, 4, 9, 16]
[x for x in range(10) if x % 2 == 0]       # [0, 2, 4, 6, 8]
[x if x > 0 else 0 for x in [-1, 2, -3]]   # [0, 2, 0]
```

Tuples

python

```
# Creation (immutable)
tup = (1, 2, 3)
tup = 1, 2, 3            # Parentheses optional
single = (1,)            # Single element tuple

# Unpacking
a, b, c = tup
a, *rest = (1, 2, 3, 4)  # a=1, rest=[2, 3, 4]
```

Dictionaries

python

```python
# Creation
d = {'a': 1, 'b': 2}
d = dict(a=1, b=2)
d = {x: x**2 for x in range(5)}  # Dict comprehension

# Methods
d['key']                 # Get value (KeyError if missing)
d.get('key', default)    # Get with default
d.keys()                 # View of keys
d.values()               # View of values
d.items()                # View of (key, value) pairs
d.update({'c': 3})       # Update/add items
d.pop('key')             # Remove and return
d.popitem()              # Remove and return last item
d.clear()                # Remove all items
d.setdefault('key', 0)   # Get or set default

# Dictionary comprehension
{k: v**2 for k, v in d.items()}
{x: x**2 for x in range(5) if x % 2 == 0}
```

Sets

python

```python
# Creation
s = {1, 2, 3}
s = set([1, 2, 2, 3])    # {1, 2, 3} - duplicates removed

# Methods
s.add(4)                 # Add element
s.remove(2)              # Remove (KeyError if missing)
s.discard(2)             # Remove (no error)
s.pop()                  # Remove and return arbitrary
s.clear()                # Remove all

# Set operations
s1 | s2                  # Union
s1 & s2                  # Intersection
s1 - s2                  # Difference
s1 ^ s2                  # Symmetric difference
s1 <= s2                 # Subset
s1 >= s2                 # Superset

# Set comprehension
{x**2 for x in range(5)}
```

Control Flow

Conditionals

```python
# if-elif-else
if x > 0:
    print("positive")
elif x < 0:
    print("negative")
else:
    print("zero")

# Ternary operator
result = "positive" if x > 0 else "non-positive"

# Match statement (Python 3.10+)
match value:
    case 1:
        print("one")
    case 2 | 3:
        print("two or three")
    case _:
        print("other")
```

Loops

```python
# for loop
for i in range(5):       # 0 to 4
    print(i)

for i in range(2, 10, 2):  # 2, 4, 6, 8
    print(i)

for item in [1, 2, 3]:
    print(item)

for i, item in enumerate(['a', 'b', 'c']):
    print(i, item)       # 0 a, 1 b, 2 c

for k, v in dict.items():
    print(k, v)

# while loop
while x > 0:
    x -= 1

# Loop control
break                    # Exit loop
continue                 # Skip to next iteration
else:                    # Executed if no break
    print("completed")

# zip
for x, y in zip([1, 2, 3], ['a', 'b', 'c']):
    print(x, y)          # 1 a, 2 b, 3 c
```

Functions

Basic Functions

python

```
# Definition
def function_name(param1, param2):
    """Docstring"""
    return result

# Default parameters
def greet(name="World"):
    return f"Hello, {name}"

# Variable arguments
def sum_all(*args):      # Tuple of arguments
    return sum(args)

def print_kwargs(**kwargs):  # Dictionary of arguments
    for k, v in kwargs.items():
        print(f"{k}: {v}")

# Unpacking arguments
def func(a, b, c):
    return a + b + c

lst = [1, 2, 3]
func(*lst)               # Unpack list

d = {'a': 1, 'b': 2, 'c': 3}
func(**d)                # Unpack dict
```

Lambda Functions

python

```
# Anonymous functions
square = lambda x: x**2
add = lambda x, y: x + y

# Common use with map, filter, reduce
list(map(lambda x: x**2, [1, 2, 3]))       # [1, 4, 9]
list(filter(lambda x: x > 0, [-1, 2, -3])) # [2]

from functools import reduce
reduce(lambda x, y: x + y, [1, 2, 3])      # 6
```

Advanced Functions

python

```
# Decorators
def timer(func):
    def wrapper(*args, **kwargs):
        import time
        start = time.time()
        result = func(*args, **kwargs)
        print(f"Took {time.time() - start}s")
        return result
    return wrapper

@timer
def slow_function():
    time.sleep(1)

# Generators
def fibonacci(n):
    a, b = 0, 1
    for _ in range(n):
        yield a
        a, b = b, a + b

# Generator expression
gen = (x**2 for x in range(5))

# Closures
def outer(x):
    def inner(y):
        return x + y
    return inner

add_five = outer(5)
add_five(3)              # 8
```

Object-Oriented Programming

Classes

```
python
```

```python
# Basic class
class Person:
    # Class variable
    species = "Homo sapiens"

    # Constructor
    def __init__(self, name, age):
        self.name = name     # Instance variable
        self.age = age

    # Instance method
    def greet(self):
        return f"Hi, I'm {self.name}"

    # Class method
    @classmethod
    def from_birth_year(cls, name, birth_year):
        return cls(name, 2024 - birth_year)

    # Static method
    @staticmethod
    def is_adult(age):
        return age >= 18

    # String representation
    def __str__(self):
        return f"Person({self.name}, {self.age})"

    def __repr__(self):
        return f"Person('{self.name}', {self.age})"

# Inheritance
class Student(Person):
    def __init__(self, name, age, student_id):
        super().__init__(name, age)
        self.student_id = student_id

    def study(self, subject):
        return f"{self.name} is studying {subject}"

# Multiple inheritance
class A:
    pass

class B:
    pass
```

```python
class C(A, B):
    pass

# Properties
class Circle:
    def __init__(self, radius):
        self._radius = radius

    @property
    def radius(self):
        return self._radius

    @radius.setter
    def radius(self, value):
        if value < 0:
            raise ValueError("Radius cannot be negative")
        self._radius = value

    @property
    def area(self):
        return 3.14159 * self._radius ** 2
```

Magic Methods

python

```python
class Vector:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    # Arithmetic
    def __add__(self, other):
        return Vector(self.x + other.x, self.y + other.y)

    def __sub__(self, other):
        return Vector(self.x - other.x, self.y - other.y)

    def __mul__(self, scalar):
        return Vector(self.x * scalar, self.y * scalar)

    # Comparison
    def __eq__(self, other):
        return self.x == other.x and self.y == other.y

    def __lt__(self, other):
        return self.magnitude() < other.magnitude()

    # Container
    def __len__(self):
        return 2

    def __getitem__(self, index):
        if index == 0:
            return self.x
        elif index == 1:
            return self.y
        raise IndexError("Index out of range")

    # Context manager
    def __enter__(self):
        return self

    def __exit__(self, exc_type, exc_val, exc_tb):
        pass
```

File Handling & I/O

Reading and Writing Files

python

```
# Reading files
with open('file.txt', 'r') as f:
    content = f.read()           # Read entire file
    line = f.readline()          # Read one line
    lines = f.readlines()        # Read all lines into list

# Writing files
with open('file.txt', 'w') as f:
    f.write('Hello World')       # Write string
    f.writelines(['line1\n', 'line2\n'])  # Write list

# Append mode
with open('file.txt', 'a') as f:
    f.write('Appended text')

# Binary mode
with open('file.bin', 'rb') as f:
    data = f.read()

# CSV files
import csv

# Reading CSV
with open('data.csv', 'r') as f:
    reader = csv.reader(f)
    for row in reader:
        print(row)

# Writing CSV
with open('data.csv', 'w', newline='') as f:
    writer = csv.writer(f)
    writer.writerow(['Name', 'Age'])
    writer.writerows([['Alice', 30], ['Bob', 25]])

# JSON files
import json

# Reading JSON
with open('data.json', 'r') as f:
    data = json.load(f)

# Writing JSON
with open('data.json', 'w') as f:
    json.dump(data, f, indent=2)
```

Exception Handling

python

```python
# Basic try-except
try:
    result = 10 / 0
except ZeroDivisionError:
    print("Cannot divide by zero")

# Multiple exceptions
try:
    # code
except (ValueError, TypeError) as e:
    print(f"Error: {e}")

# Exception hierarchy
try:
    # code
except ValueError:
    # Handle ValueError
except Exception as e:
    # Handle any other exception
    print(f"Unexpected error: {e}")
else:
    # Executed if no exception
    print("Success")
finally:
    # Always executed
    print("Cleanup")

# Raising exceptions
raise ValueError("Invalid value")

# Custom exceptions
class CustomError(Exception):
    def __init__(self, message):
        self.message = message
        super().__init__(self.message)

# Assertions
assert x > 0, "x must be positive"
```

Advanced Python

Itertools

python

```python
import itertools

# Infinite iterators
itertools.count(10)              # 10, 11, 12, ...
itertools.cycle('ABC')           # A, B, C, A, B, C, ...
itertools.repeat(10, 3)          # 10, 10, 10

# Combinatoric iterators
itertools.product('AB', '12')    # A1, A2, B1, B2
itertools.permutations('ABC', 2) # AB, AC, BA, BC, CA, CB
itertools.combinations('ABC', 2) # AB, AC, BC

# Other useful functions
itertools.chain([1, 2], [3, 4])  # 1, 2, 3, 4
itertools.groupby('AAABBBCC')    # Groups consecutive elements
```

Collections

python

```python
from collections import *

# Counter
c = Counter(['a', 'b', 'c', 'a', 'b', 'b'])
c.most_common(2)                 # [('b', 3), ('a', 2)]

# defaultdict
dd = defaultdict(list)
dd['key'].append('value')        # No KeyError

# deque (double-ended queue)
d = deque([1, 2, 3])
d.appendleft(0)                  # [0, 1, 2, 3]
d.popleft()                      # 0

# namedtuple
Point = namedtuple('Point', ['x', 'y'])
p = Point(1, 2)
p.x, p.y                         # 1, 2

# OrderedDict (maintains insertion order)
od = OrderedDict()
```

Functools

python

```python
from functools import *

# lru_cache (memoization)
@lru_cache(maxsize=128)
def fibonacci(n):
    if n < 2:
        return n
    return fibonacci(n-1) + fibonacci(n-2)

# partial (partial function application)
from operator import mul
double = partial(mul, 2)
double(5)                        # 10

# reduce
reduce(lambda x, y: x + y, [1, 2, 3, 4])  # 10
```

Context Managers

python

```python
# Using contextlib
from contextlib import contextmanager

@contextmanager
def timer():
    import time
    start = time.time()
    yield
    print(f"Time: {time.time() - start}s")

with timer():
    # code to time
    pass

# Custom context manager class
class FileManager:
    def __init__(self, filename, mode):
        self.filename = filename
        self.mode = mode
        self.file = None

    def __enter__(self):
        self.file = open(self.filename, self.mode)
        return self.file

    def __exit__(self, exc_type, exc_val, exc_tb):
        self.file.close()
```

Type Hints

python

```python
from typing import List, Dict, Optional, Union, Tuple, Any

# Basic type hints
def greet(name: str) -> str:
    return f"Hello, {name}"

# Complex types
def process_items(items: List[int]) -> Dict[str, int]:
    return {"sum": sum(items), "count": len(items)}

# Optional and Union
def find_user(user_id: int) -> Optional[str]:
    # Returns str or None
    pass

def parse_value(value: Union[int, str]) -> int:
    return int(value)

# Type aliases
Vector = List[float]
Matrix = List[List[float]]

# Generic types
from typing import TypeVar, Generic

T = TypeVar('T')

class Stack(Generic[T]):
    def __init__(self) -> None:
        self._items: List[T] = []

    def push(self, item: T) -> None:
        self._items.append(item)

    def pop(self) -> T:
        return self._items.pop()
```

Essential Libraries

OS and System

python

```python
import os
import sys
import platform

# OS operations
os.getcwd()                      # Current directory
os.chdir('/path')                # Change directory
os.listdir('.')                  # List directory
os.mkdir('new_dir')              # Create directory
os.makedirs('dir/sub/dir')       # Create nested directories
os.remove('file.txt')            # Delete file
os.rmdir('empty_dir')            # Delete empty directory
os.path.join('dir', 'file.txt')  # Join paths
os.path.exists('file.txt')       # Check if exists
os.path.isfile('file.txt')       # Check if file
os.path.isdir('dir')             # Check if directory

# Environment variables
os.environ.get('PATH')
os.environ['MY_VAR'] = 'value'

# System
sys.argv                         # Command line arguments
sys.exit(0)                      # Exit program
sys.path                         # Python path
platform.system()                # OS name
platform.python_version()        # Python version
```

DateTime

python

```python
from datetime import datetime, date, time, timedelta

# Current date/time
now = datetime.now()
today = date.today()

# Creating dates
dt = datetime(2024, 1, 1, 12, 0, 0)
d = date(2024, 1, 1)
t = time(12, 0, 0)

# Formatting
dt.strftime('%Y-%m-%d %H:%M:%S')  # 2024-01-01 12:00:00
datetime.strptime('2024-01-01', '%Y-%m-%d')

# Arithmetic
tomorrow = today + timedelta(days=1)
diff = datetime.now() - dt       # timedelta object
```

Regular Expressions

python

```python
import re

# Basic patterns
re.match(r'^\d+', '123abc')      # Match at beginning
re.search(r'\d+', 'abc123def')   # Search anywhere
re.findall(r'\d+', 'a1b2c3')     # ['1', '2', '3']
re.sub(r'\d+', 'X', 'a1b2c3')    # 'aXbXcX'

# Compiled patterns
pattern = re.compile(r'\d+')
pattern.findall('a1b2c3')

# Groups
match = re.search(r'(\d+)-(\d+)', '123-456')
match.group(0)                   # '123-456'
match.group(1)                   # '123'
match.group(2)                   # '456'
match.groups()                   # ('123', '456')

# Common patterns
r'\d'                            # Digit
r'\w'                            # Word character
r'\s'                            # Whitespace
r'.'                             # Any character
r'^'                             # Start of string
r'$'                             # End of string
r'*'                             # 0 or more
r'+'                             # 1 or more
r'?'                             # 0 or 1
r'{n}'                           # Exactly n
r'{n,m}'                         # Between n and m
```

Logging

python

```python
import logging

# Basic configuration
logging.basicConfig(
    level=logging.INFO,
    format='%(asctime)s - %(levelname)s - %(message)s',
    filename='app.log'
)

# Logging levels
logging.debug('Debug message')
logging.info('Info message')
logging.warning('Warning message')
logging.error('Error message')
logging.critical('Critical message')

# Logger instance
logger = logging.getLogger(__name__)
logger.setLevel(logging.DEBUG)

# Handler
handler = logging.FileHandler('app.log')
formatter = logging.Formatter('%(asctime)s - %(name)s - %(levelname)s - %(message)s')
handler.setFormatter(formatter)
logger.addHandler(handler)
```

Data Science Libraries

NumPy

python

```
import numpy as np

# Array creation
arr = np.array([1, 2, 3])
arr = np.zeros((3, 3))           # 3x3 zeros
arr = np.ones((2, 3))            # 2x3 ones
arr = np.eye(3)                  # 3x3 identity
arr = np.arange(0, 10, 2)        # [0, 2, 4, 6, 8]
arr = np.linspace(0, 1, 5)       # 5 points from 0 to 1
arr = np.random.rand(3, 3)       # Random 3x3

# Array operations
arr.shape                        # Dimensions
arr.dtype                        # Data type
arr.ndim                         # Number of dimensions
arr.size                         # Total elements
arr.reshape(2, 3)                # Reshape
arr.flatten()                    # 1D array
arr.T                            # Transpose

# Indexing and slicing
arr[0]                           # First element
arr[-1]                          # Last element
arr[0, 1]                        # Element at (0, 1)
arr[:, 0]                        # First column
arr[0, :]                        # First row
arr[1:3, 0:2]                    # Subarray

# Mathematical operations
np.add(arr1, arr2)               # Element-wise addition
np.subtract(arr1, arr2)          # Element-wise subtraction
np.multiply(arr1, arr2)          # Element-wise multiplication
np.divide(arr1, arr2)            # Element-wise division
np.dot(arr1, arr2)               # Matrix multiplication
arr1 @ arr2                      # Matrix multiplication (Python 3.5+)

# Statistical functions
arr.mean()                       # Mean
arr.std()                        # Standard deviation
arr.var()                        # Variance
arr.min()                        # Minimum
arr.max()                        # Maximum
arr.sum()                        # Sum
arr.cumsum()                     # Cumulative sum
np.percentile(arr, 50)           # Median
```

```
# Broadcasting
arr = np.array([[1, 2], [3, 4]])
arr + 10                         # Add 10 to all elements
arr * np.array([1, 2])           # Multiply columns
```

Pandas

```
python
```

```python
import pandas as pd

# Series
s = pd.Series([1, 2, 3, 4])
s = pd.Series({'a': 1, 'b': 2})

# DataFrame
df = pd.DataFrame({
    'A': [1, 2, 3],
    'B': [4, 5, 6],
    'C': ['x', 'y', 'z']
})

# Reading data
df = pd.read_csv('file.csv')
df = pd.read_excel('file.xlsx')
df = pd.read_json('file.json')
df = pd.read_sql(query, connection)

# Basic info
df.shape                         # (rows, columns)
df.columns                       # Column names
df.index                         # Index
df.dtypes                        # Data types
df.info()                        # Summary info
df.describe()                    # Statistical summary
df.head(5)                       # First 5 rows
df.tail(5)                       # Last 5 rows

# Selection
df['A']                          # Select column
df[['A', 'B']]                   # Select multiple columns
df.loc[0]                        # Select row by label
df.iloc[0]                       # Select row by position
df.loc[0, 'A']                   # Select element
df.loc[df['A'] > 2]             # Boolean indexing

# Modification
df['D'] = df['A'] + df['B']      # New column
df.drop('D', axis=1, inplace=True)  # Drop column
df.drop(0, axis=0)               # Drop row
df.rename(columns={'A': 'a'})    # Rename columns

# Missing data
df.isnull()                      # Check for null
df.dropna()                      # Drop null rows
df.fillna(0)                     # Fill null with 0
df.interpolate()                 # Interpolate missing

# Grouping
df.groupby('A').sum()            # Group and aggregate
df.groupby(['A', 'B']).mean()    # Multiple groups
df.pivot_table(values='C', index='A', columns='B')

# Merging
pd.concat([df1, df2])            # Concatenate
pd.merge(df1, df2, on='key')     # Merge on key
df1.join(df2, on='key')          # Join

# Apply functions
df.apply(lambda x: x.max() - x.min())  # Apply to columns
df.applymap(lambda x: x**2)      # Apply to elements

# Time series
df['date'] = pd.to_datetime(df['date'])
df.set_index('date', inplace=True)
df.resample('M').mean()          # Monthly average
df.rolling(window=7).mean()      # 7-day rolling average

# Saving data
df.to_csv('output.csv', index=False)
df.to_excel('output.xlsx')
df.to_json('output.json')
```

Matplotlib

```python
import matplotlib.pyplot as plt

# Basic plot
plt.plot([1, 2, 3, 4], [1, 4, 9, 16])
plt.xlabel('X Label')
plt.ylabel('Y Label')
plt.title('Title')
plt.show()

# Subplots
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10, 5))
ax1.plot(x, y1)
ax2.plot(x, y2)

# Plot types
plt.scatter(x, y)                # Scatter plot
plt.bar(x, y)                    # Bar plot
plt.hist(data, bins=20)          # Histogram
plt.boxplot(data)                # Box plot
plt.pie(sizes, labels=labels)    # Pie chart

# Customization
plt.plot(x, y, 'r--', linewidth=2, label='Line 1')
plt.grid(True)
plt.legend()
plt.xlim(0, 10)
plt.ylim(0, 100)

# Save figure
plt.savefig('plot.png', dpi=300, bbox_inches='tight')
```

Seaborn

```python
import seaborn as sns

# Set style
sns.set_style('whitegrid')
sns.set_palette('husl')

# Basic plots
sns.lineplot(x='x', y='y', data=df)
sns.scatterplot(x='x', y='y', data=df)
sns.barplot(x='category', y='value', data=df)

# Statistical plots
sns.distplot(data)               # Distribution plot
sns.boxplot(x='category', y='value', data=df)
sns.violinplot(x='category', y='value', data=df)
sns.heatmap(df.corr(), annot=True)  # Correlation heatmap

# Regression plots
sns.regplot(x='x', y='y', data=df)
sns.lmplot(x='x', y='y', data=df, hue='category')

# Pair plots
sns.pairplot(df)
sns.pairplot(df, hue='category')

# Facet grids
g = sns.FacetGrid(df, col='category', row='group')
g.map(plt.scatter, 'x', 'y')
```

Machine Learning Libraries

Scikit-Learn

python

```python
from sklearn import *

# Data preprocessing
from sklearn.preprocessing import StandardScaler, MinMaxScaler, LabelEncoder
from sklearn.preprocessing import OneHotEncoder, PolynomialFeatures

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Train-test split
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Supervised Learning Models
from sklearn.linear_model import LinearRegression, LogisticRegression
from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor
from sklearn.ensemble import RandomForestClassifier, RandomForestRegressor
from sklearn.ensemble import GradientBoostingClassifier, GradientBoostingRegressor
from sklearn.svm import SVC, SVR
from sklearn.neighbors import KNeighborsClassifier, KNeighborsRegressor
from sklearn.naive_bayes import GaussianNB

# Model training
model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(X_train, y_train)

# Predictions
y_pred = model.predict(X_test)
y_prob = model.predict_proba(X_test)  # For classification

# Model evaluation
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

# Classification metrics
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
cm = confusion_matrix(y_test, y_pred)
print(classification_report(y_test, y_pred))

# Regression metrics
mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
```

```
r2 = r2_score(y_test, y_pred)

# Cross-validation
from sklearn.model_selection import cross_val_score, KFold
scores = cross_val_score(model, X, y, cv=5, scoring='accuracy')

# Grid search
from sklearn.model_selection import GridSearchCV
param_grid = {'n_estimators': [50, 100, 200], 'max_depth': [None, 10, 20]}
grid_search = GridSearchCV(model, param_grid, cv=5)
grid_search.fit(X_train, y_train)
best_params = grid_search.best_params_

# Pipeline
from sklearn.pipeline import Pipeline
pipe = Pipeline([
    ('scaler', StandardScaler()),
    ('model', RandomForestClassifier())
])
pipe.fit(X_train, y_train)

# Unsupervised Learning
from sklearn.cluster import KMeans, DBSCAN, AgglomerativeClustering
from sklearn.decomposition import PCA, TruncatedSVD

# Clustering
kmeans = KMeans(n_clusters=3, random_state=42)
clusters = kmeans.fit_predict(X)

# Dimensionality reduction
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X)

# Feature selection
from sklearn.feature_selection import SelectKBest, chi2, f_classif
selector = SelectKBest(chi2, k=10)
X_selected = selector.fit_transform(X, y)

# Save/Load models
import joblib
joblib.dump(model, 'model.pkl')
loaded_model = joblib.load('model.pkl')
```

XGBoost

python

```python
import xgboost as xgb

# Classification
xgb_clf = xgb.XGBClassifier(
    n_estimators=100,
    max_depth=5,
    learning_rate=0.1,
    objective='binary:logistic',
    random_state=42
)
xgb_clf.fit(X_train, y_train)

# Regression
xgb_reg = xgb.XGBRegressor(
    n_estimators=100,
    max_depth=5,
    learning_rate=0.1,
    objective='reg:squarederror',
    random_state=42
)
xgb_reg.fit(X_train, y_train)

# DMatrix (XGBoost's data structure)
dtrain = xgb.DMatrix(X_train, label=y_train)
dtest = xgb.DMatrix(X_test, label=y_test)

# Training with native API
params = {
    'max_depth': 5,
    'eta': 0.1,
    'objective': 'binary:logistic',
    'eval_metric': 'logloss'
}
model = xgb.train(params, dtrain, num_boost_round=100)

# Feature importance
xgb.plot_importance(model)
plt.show()
```

LightGBM

python

```
import lightgbm as lgb

# Classification
lgb_clf = lgb.LGBMClassifier(
    n_estimators=100,
    max_depth=5,
    learning_rate=0.1,
    objective='binary',
    random_state=42
)
lgb_clf.fit(X_train, y_train)

# Regression
lgb_reg = lgb.LGBMRegressor(
    n_estimators=100,
    max_depth=5,
    learning_rate=0.1,
    objective='regression',
    random_state=42
)
lgb_reg.fit(X_train, y_train)

# Dataset
train_data = lgb.Dataset(X_train, label=y_train)
valid_data = lgb.Dataset(X_test, label=y_test)

# Training with native API
params = {
    'objective': 'binary',
    'metric': 'binary_logloss',
    'max_depth': 5,
    'learning_rate': 0.1,
    'random_state': 42
}
model = lgb.train(params, train_data, valid_sets=[valid_data], num_boost_round=100)
```

CatBoost

```python
from catboost import CatBoostClassifier, CatBoostRegressor

# Classification
cat_clf = CatBoostClassifier(
    iterations=100,
    depth=5,
    learning_rate=0.1,
    loss_function='Logloss',
    random_state=42
)
cat_clf.fit(X_train, y_train, cat_features=categorical_features_indices)

# Regression
cat_reg = CatBoostRegressor(
    iterations=100,
    depth=5,
    learning_rate=0.1,
    loss_function='RMSE',
    random_state=42
)
cat_reg.fit(X_train, y_train)

# Feature importance
feature_importance = cat_clf.feature_importances_
```

Deep Learning

TensorFlow/Keras

```
python
```

```python
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers

# Sequential model
model = keras.Sequential([
    layers.Dense(64, activation='relu', input_shape=(input_dim,)),
    layers.Dropout(0.5),
    layers.Dense(32, activation='relu'),
    layers.Dense(1, activation='sigmoid')
])

# Functional API
inputs = keras.Input(shape=(input_dim,))
x = layers.Dense(64, activation='relu')(inputs)
x = layers.Dropout(0.5)(x)
x = layers.Dense(32, activation='relu')(x)
outputs = layers.Dense(1, activation='sigmoid')(x)
model = keras.Model(inputs=inputs, outputs=outputs)

# Compile model
model.compile(
    optimizer='adam',
    loss='binary_crossentropy',
    metrics=['accuracy']
)

# Train model
history = model.fit(
    X_train, y_train,
    epochs=50,
    batch_size=32,
    validation_split=0.2,
    callbacks=[
        keras.callbacks.EarlyStopping(patience=5),
        keras.callbacks.ModelCheckpoint('best_model.h5', save_best_only=True)
    ]
)

# Evaluate
loss, accuracy = model.evaluate(X_test, y_test)

# Predict
predictions = model.predict(X_test)

# CNN example
```

```
cnn_model = keras.Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(10, activation='softmax')
])

# RNN/LSTM example
rnn_model = keras.Sequential([
    layers.LSTM(128, return_sequences=True, input_shape=(timesteps, features)),
    layers.LSTM(64),
    layers.Dense(32, activation='relu'),
    layers.Dense(1)
])

# Save/Load model
model.save('model.h5')
loaded_model = keras.models.load_model('model.h5')
```

PyTorch

```
python
```

```python
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, TensorDataset

# Neural Network
class NeuralNet(nn.Module):
    def __init__(self, input_size, hidden_size, output_size):
        super(NeuralNet, self).__init__()
        self.fc1 = nn.Linear(input_size, hidden_size)
        self.relu = nn.ReLU()
        self.dropout = nn.Dropout(0.5)
        self.fc2 = nn.Linear(hidden_size, output_size)
        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        x = self.fc1(x)
        x = self.relu(x)
        x = self.dropout(x)
        x = self.fc2(x)
        x = self.sigmoid(x)
        return x

# Initialize model
model = NeuralNet(input_size=10, hidden_size=64, output_size=1)

# Loss and optimizer
criterion = nn.BCELoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

# Prepare data
X_tensor = torch.FloatTensor(X_train)
y_tensor = torch.FloatTensor(y_train)
dataset = TensorDataset(X_tensor, y_tensor)
dataloader = DataLoader(dataset, batch_size=32, shuffle=True)

# Training loop
for epoch in range(100):
    for batch_X, batch_y in dataloader:
        # Forward pass
        outputs = model(batch_X)
        loss = criterion(outputs, batch_y)

        # Backward pass
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

# Evaluation
model.eval()
with torch.no_grad():
    predictions = model(torch.FloatTensor(X_test))

# CNN example
class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        self.conv1 = nn.Conv2d(1, 32, kernel_size=3)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(32, 64, kernel_size=3)
        self.fc1 = nn.Linear(64 * 5 * 5, 64)
        self.fc2 = nn.Linear(64, 10)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = x.view(-1, 64 * 5 * 5)
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return x

# Save/Load model
torch.save(model.state_dict(), 'model.pth')
model.load_state_dict(torch.load('model.pth'))
```

Transformers (Hugging Face)

python

```python
from transformers import pipeline, AutoModel, AutoTokenizer

# Pipelines for quick use
classifier = pipeline("sentiment-analysis")
result = classifier("I love machine learning!")

# Text generation
generator = pipeline("text-generation", model="gpt2")
text = generator("Once upon a time", max_length=50)

# Question answering
qa_pipeline = pipeline("question-answering")
result = qa_pipeline({
    'question': 'What is machine learning?',
    'context': 'Machine learning is a subset of AI...'
})

# Custom model usage
model_name = "bert-base-uncased"
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModel.from_pretrained(model_name)

# Tokenize input
inputs = tokenizer("Hello world!", return_tensors="pt")
outputs = model(**inputs)

# Fine-tuning example
from transformers import TrainingArguments, Trainer

training_args = TrainingArguments(
    output_dir="./results",
    num_train_epochs=3,
    per_device_train_batch_size=16,
    per_device_eval_batch_size=64,
    warmup_steps=500,
    weight_decay=0.01,
    logging_dir="./logs",
)

trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=val_dataset,
)
```

```
trainer.train()
```

Best Practices

Code Style (PEP 8)

```
python
```

```python
# Naming conventions
variable_name = 1                # snake_case for variables
CONSTANT_NAME = 100              # UPPER_CASE for constants
def function_name():             # snake_case for functions
    pass
class ClassName:                 # PascalCase for classes
    pass

# Indentation
def function():
    if condition:
        # 4 spaces per indentation level
        do_something()

# Line length
# Maximum 79 characters per line
# For long strings, use parentheses for implicit line continuation
long_string = (
    "This is a very long string that would normally exceed "
    "the recommended line length limit"
)

# Imports
# Standard library imports first
import os
import sys

# Related third-party imports
import numpy as np
import pandas as pd

# Local application imports
from mymodule import myfunction

# Docstrings
def function(param1, param2):
    """
    Brief description of function.

    Args:
        param1 (type): Description of param1.
        param2 (type): Description of param2.

    Returns:
        type: Description of return value.
    Raises:
        ValueError: Description of when raised.
    """
    pass
```

Testing

python

```python
import unittest
import pytest

# unittest
class TestMyFunction(unittest.TestCase):
    def setUp(self):
        # Setup before each test
        self.data = [1, 2, 3]

    def test_sum(self):
        self.assertEqual(sum(self.data), 6)

    def test_length(self):
        self.assertEqual(len(self.data), 3)

    def tearDown(self):
        # Cleanup after each test
        self.data = None

if __name__ == '__main__':
    unittest.main()

# pytest
def test_sum():
    assert sum([1, 2, 3]) == 6

def test_length():
    assert len([1, 2, 3]) == 3

# Fixtures
@pytest.fixture
def sample_data():
    return [1, 2, 3]

def test_with_fixture(sample_data):
    assert sum(sample_data) == 6

# Parametrized tests
@pytest.mark.parametrize("input,expected", [
    ([1, 2, 3], 6),
    ([1, 1, 1], 3),
    ([], 0)
])
def test_sum_parametrized(input, expected):
    assert sum(input) == expected
```

Virtual Environments

```bash
# Create virtual environment
python -m venv myenv

# Activate (Windows)
myenv\Scripts\activate

# Activate (Linux/Mac)
source myenv/bin/activate

# Install packages
pip install package_name

# Save requirements
pip freeze > requirements.txt

# Install from requirements
pip install -r requirements.txt

# Deactivate
deactivate
```

Performance Optimization

python

```python
# Profiling
import cProfile
import timeit

# Time a function
timeit.timeit('sum([1, 2, 3])', number=1000000)

# Profile code
cProfile.run('your_function()')

# Memory profiling
from memory_profiler import profile

@profile
def memory_intensive_function():
    # Your code here
    pass

# Optimization tips
# 1. Use built-in functions (they're written in C)
# 2. Use list comprehensions instead of loops
# 3. Use generators for large datasets
# 4. Use NumPy for numerical operations
# 5. Use caching/memoization for expensive operations
# 6. Use multiprocessing for CPU-bound tasks
# 7. Use asyncio for I/O-bound tasks

# Multiprocessing example
from multiprocessing import Pool

def process_item(item):
    return item ** 2

with Pool(processes=4) as pool:
    results = pool.map(process_item, range(100))

# Async example
import asyncio

async def fetch_data(url):
    # Async operation
    await asyncio.sleep(1)
    return f"Data from {url}"

async def main():
    urls = ['url1', 'url2', 'url3']
    tasks = [fetch_data(url) for url in urls]
    results = await asyncio.gather(*tasks)
    return results

# Run async function
asyncio.run(main())
```

Documentation

python

```
"""
Module docstring describing the module's purpose.

This module provides functionality for...
"""


class MyClass:
    """
    Class docstring with description.

    Attributes:
        attr1 (type): Description of attr1.
        attr2 (type): Description of attr2.
    """

    def __init__(self, param1, param2):
        """
        Initialize MyClass.

        Args:
            param1 (type): Description.
            param2 (type): Description.
        """
        self.attr1 = param1
        self.attr2 = param2

    def method(self, arg):
        """
        Method description.

        Args:
            arg (type): Description.

        Returns:
            type: Description.

        Examples:
            >>> obj = MyClass(1, 2)
            >>> obj.method(3)
            6
        """
        return self.attr1 + self.attr2 + arg
```

Machine Learning Workflow

1. Data Preparation

python

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

# Explore data
data.info()
data.describe()
data.head()

# Handle missing values
data.dropna()
data.fillna(data.mean())

# Feature engineering
data['new_feature'] = data['feature1'] * data['feature2']

# Encode categorical variables
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
le = LabelEncoder()
data['category_encoded'] = le.fit_transform(data['category'])

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

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

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

2. Model Selection

python

```
# Try multiple models
models = {
    'Logistic Regression': LogisticRegression(),
    'Random Forest': RandomForestClassifier(),
    'XGBoost': xgb.XGBClassifier(),
    'SVM': SVC()
}

results = {}
for name, model in models.items():
    model.fit(X_train_scaled, y_train)
    y_pred = model.predict(X_test_scaled)
    accuracy = accuracy_score(y_test, y_pred)
    results[name] = accuracy
    print(f"{name}: {accuracy:.4f}")
```

3. Hyperparameter Tuning

python

```
# Grid Search
param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10]
}

grid_search = GridSearchCV(
    RandomForestClassifier(),
    param_grid,
    cv=5,
    scoring='accuracy',
    n_jobs=-1
)

grid_search.fit(X_train_scaled, y_train)
best_model = grid_search.best_estimator_
```

4. Model Evaluation

python

```python
# Predictions
y_pred = best_model.predict(X_test_scaled)
y_prob = best_model.predict_proba(X_test_scaled)[:, 1]

# Metrics
print(f"Accuracy: {accuracy_score(y_test, y_pred):.4f}")
print(f"Precision: {precision_score(y_test, y_pred):.4f}")
print(f"Recall: {recall_score(y_test, y_pred):.4f}")
print(f"F1-Score: {f1_score(y_test, y_pred):.4f}")

# Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, fmt='d')
plt.title('Confusion Matrix')
plt.show()

# ROC Curve
from sklearn.metrics import roc_curve, auc
fpr, tpr, _ = roc_curve(y_test, y_prob)
roc_auc = auc(fpr, tpr)

plt.plot(fpr, tpr, label=f'ROC curve (AUC = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], 'k--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend()
plt.show()
```

5. Model Deployment

```python
# Save model
import joblib
joblib.dump(best_model, 'model.pkl')
joblib.dump(scaler, 'scaler.pkl')

# Load and use model
loaded_model = joblib.load('model.pkl')
loaded_scaler = joblib.load('scaler.pkl')

# Make predictions on new data
new_data = pd.DataFrame({...})
new_data_scaled = loaded_scaler.transform(new_data)
predictions = loaded_model.predict(new_data_scaled)
```

Resources for Further Learning

Official Documentation

- Python: https://docs.python.org/3/
- NumPy: https://numpy.org/doc/
- Pandas: https://pandas.pydata.org/docs/
- Scikit-learn: https://scikit-learn.org/stable/
- TensorFlow: https://www.tensorflow.org/api_docs
- PyTorch: https://pytorch.org/docs/

Online Courses

- Coursera: Machine Learning by Andrew Ng
- Fast.ai: Practical Deep Learning
- Kaggle Learn: Free mini-courses

Books

- "Python for Data Analysis" by Wes McKinney
- "Hands-On Machine Learning" by Aurélien Géron
- "Deep Learning" by Ian Goodfellow

Practice Platforms

- Kaggle: Competitions and datasets
- Google Colab: Free GPU/TPU runtime

- GitHub: Open source projects

Communities

- Stack Overflow
- Reddit: r/learnpython, r/MachineLearning
- Discord/Slack ML communities

Remember: The key to mastering Python and machine learning is consistent practice and working on real projects!