# From Zero to Hero: Hands-On Machine Learning with Python 🛠️

Alex Lavaee 3/30/25

# About Me

BOEING

HARVARD MEDICAL SCHOOL

Microsoft

BOSTON UNIVERSITY

DS @ BU

# ML Ecosystem in Python*

Tabular Data Analysis

pandas

Scientific Computing

NumPy

Machine Learning

scikit learn

XGBoost

Data Visualization

matplotlib

seaborn

# Technical Note

## Python ML Libraries Low-Level Details

# Today: ML with scikit-learn (sklearn)

- Today's focus is sklearn
- Core Python machine learning library
- Contains data preprocessing routines, ML algorithms, tools to measure ML algorithm performance, etc.

# Getting Started with sklearn

# Installation

# Package Management

## Conda

- https://conda-forge.org/download/

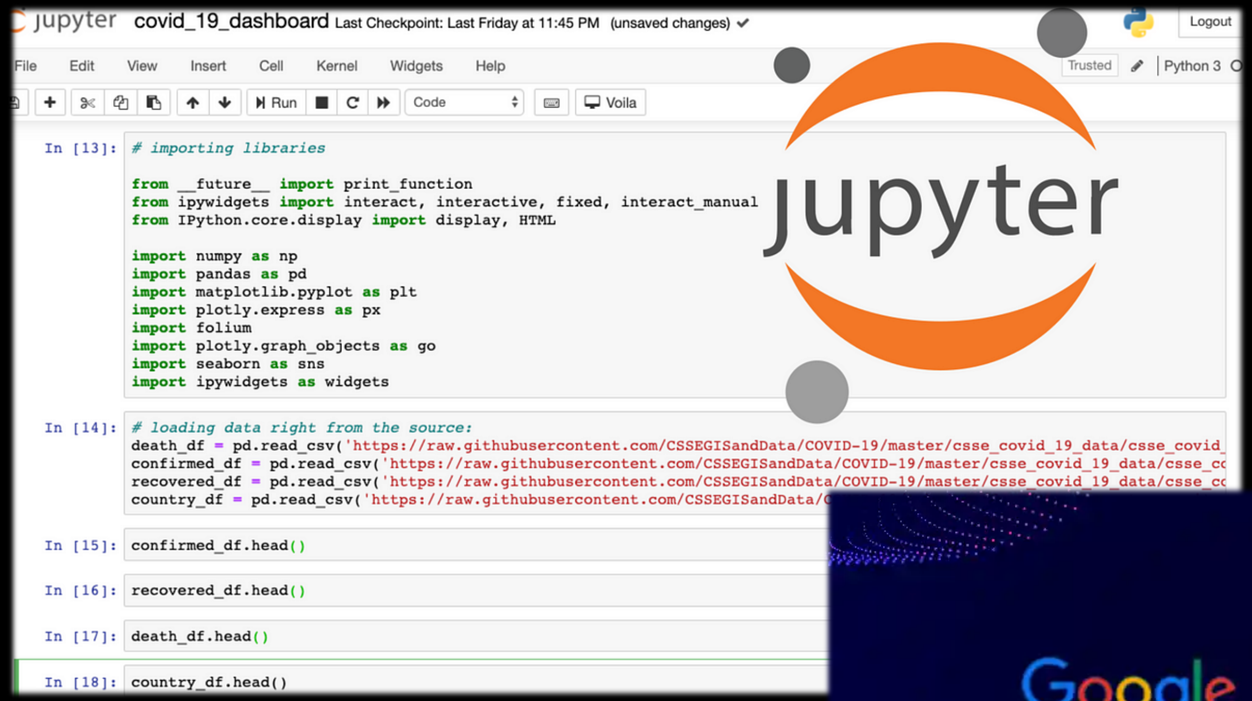
```
# ~/projects/smart-sink
$ conda activate my_env
(my_env)
# ~/projects/smart-sink
$ conda install make
(my_env)
# ~/projects/smart-sink
$ make -j 4
Building...
compiling: hello.c
```

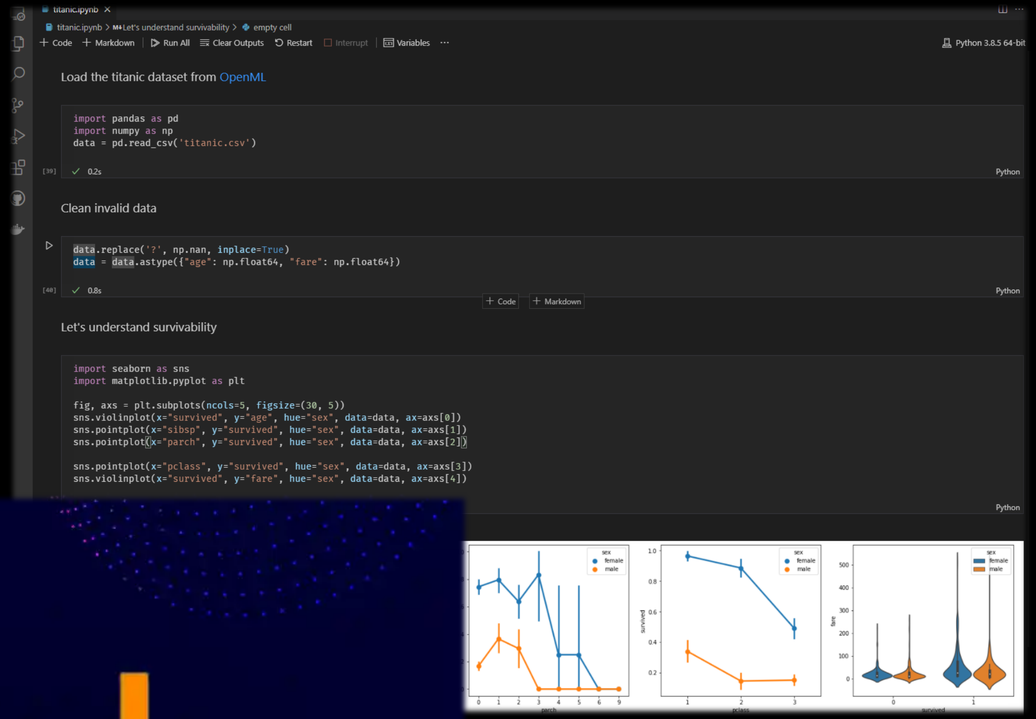
# Install Packages with Conda

```
# Create a new conda environment named 'ml' with Python 3.10 and install essential packages
conda create -n ml python=3.10 \
    pandas \
    openpyxl \
    numpy \
    scipy \
    seaborn \
    matplotlib \
    scikit-learn \
    jupyterlab \
    ipython

# Activate the environment
conda activate ml

# Later you can install other packages like this:
conda install -c conda-forge xgboost

# To deactivate the environment, run:
conda deactivate
```

# Development Environment

## Jupyter or VS Code or Google Colab



# Data Preprocessing

# Split Training and Testing Data

```python
from sklearn.model_selection import train_test_split

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

# Data Normalization

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

# Standardization (zero mean, unit variance)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Normalization (0-1 range)
normalizer = MinMaxScaler()
X_train_norm = normalizer.fit_transform(X_train)
X_test_norm = normalizer.transform(X_test)
```

# Handling Missing Data

```python
# Handle missing values
from sklearn.impute import SimpleImputer
imputer = SimpleImputer(strategy='mean')
X_imputed = imputer.fit_transform(X)
```

# Core ML Algorithms

# Supervised

# Classification

# Decision Trees

```python
from sklearn.tree import DecisionTreeClassifier
import numpy as np

# Abstract input data
X = np.array([[feature1, feature2], ...])  # Feature matrix
y = np.array([0, 1, 0, ...])  # Class labels

# Create and train model
model = DecisionTreeClassifier(max_depth=3)
model.fit(X, y)

# Make predictions
predictions = model.predict(X_new)
```

# Logistic Regression

```python
from sklearn.linear_model import LogisticRegression
import numpy as np

# Abstract input data
X = np.array([[feature1, feature2], ...])  # Features
y = np.array([0, 1, 1, 0, ...])  # Binary class labels

# Create and train model
model = LogisticRegression()
model.fit(X, y)

# Predict class and probabilities
predictions = model.predict(X_new)
probabilities = model.predict_proba(X_new)
```

# KNN (K-Nearest Neighbors)

```python
from sklearn.neighbors import KNeighborsClassifier
import numpy as np

# Abstract input data
X = np.array([[feature1, feature2], ...])  # Features
y = np.array([0, 1, 2, ...])  # Class labels

# Create and train model
model = KNeighborsClassifier(n_neighbors=5)
model.fit(X, y)

# Make predictions
predictions = model.predict(X_new)
```

# Regression

# Decision Trees

```python
from sklearn.tree import DecisionTreeRegressor
import numpy as np

# Abstract input data
X = np.array([[feature1, feature2], ...])  # Features
y = np.array([target1, target2, ...])  # Continuous targets

# Create and train model
model = DecisionTreeRegressor(max_depth=5, min_samples_split=5)
model.fit(X, y)

# Make predictions
predictions = model.predict(X_new)

```

# Linear Regression

```python
from sklearn.linear_model import LinearRegression
import numpy as np

# Abstract input data
X = np.array([[feature1], [feature2], ...])  # Features
y = np.array([target1, target2, ...])  # Continuous targets

# Create and train model
model = LinearRegression()
model.fit(X, y)

# Get coefficients and predict
print(f"Coefficient: {model.coef_[0]}, Intercept: {model.intercept_}")
predictions = model.predict(X_new)
```

# Unsupervised

# KMeans Clustering

```python
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
import numpy as np

# Abstract input data
X = np.array([[feature1, feature2], ...])  # Features (no labels needed)

# Normalize data (important for distance-based algorithms like KMeans)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Create and train model
model = KMeans(n_clusters=3, random_state=42)
model.fit(X_scaled)  # Use scaled data for clustering

# Get cluster assignments and centers
labels = model.labels_  # Cluster assignments for each sample
centers = model.cluster_centers_  # Coordinates of cluster centers

# To transform centers back to original scale (if needed):
centers_original = scaler.inverse_transform(centers)
```

# Principal Component Analysis (PCA)

```python
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
import numpy as np
import matplotlib.pyplot as plt

# Abstract input data
X = np.array([[feature1, feature2, feature3, ...], ...])  # High-dimensional data

# Standardize the data (important for PCA)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Create and fit PCA model (reduce to 2 components for visualization)
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

# Get explained variance ratio
explained_variance = pca.explained_variance_ratio_
print(f"Explained variance ratio: {explained_variance}")
print(f"Total variance explained: {sum(explained_variance):.4f}")

# Get component loadings
loadings = pca.components_
feature_names = ['feature1', 'feature2', 'feature3', ...]  # Replace with actual feature names

# For each component, print the top contributing features
for i, component in enumerate(loadings):
    sorted_idx = np.argsort(np.abs(component))[::-1]  # Sort by absolute values
    print(f"PC{i+1} top features:")
    for idx in sorted_idx[:3]:  # Print top 3 features
        print(f"  {feature_names[idx]}: {component[idx]:.4f}")
```

# Putting It All Together—Demo Notebook!

https://archive.ics.uci.edu/dataset/242/energy+efficiency

# Additional References

## Python + Data Science Resources

- https://github.com/lavaman131/PythonCheatSheet/tree/main
- https://pandas.pydata.org/docs/user_guide/10min.html#min
- https://numpy.org/doc/stable/user/absolute_beginners.html
- https://scikit-learn.org/stable/getting_started.html
- https://matplotlib.org/stable/users/explain/quick_start.html
- Use AI tools to help fill missing gaps in knowledge and love to learn!