WiDs Final Report

Deepankar Hiwarale
24bB128

Jan 2026

1 Python Basics

Learning Python and its different libraries has been the first step in this project, and towards understanding how to store and manipulate data. Unlike other languages that are syntax heavy and difficult for beginners, **python** relies heavily on indentation and simple english like keywords, which makes it easy to learn, understand and code in.

1.1 Data Types and Data Structures

This project started with understanding and using simple data types that are avaiable in python like **integers**, **floats**, **strings** and **booleans**. The major takeaway from this section was that integers are unbounded in pyhton, meaning that there is no restriction on the size of numbers it can store.

Next part was understanding some important data structures, how to access it and manipulate it:

- **Lists:** Lists are mutable data structures, meaning that a list can be edited after its created without making another list. Elements can be added using `append()`, elements can be removed using `remove()` lists can be sliced using range indexing (e.g., `L[2:4]`).
- **Tuples:** In contrast to lists, tuples are immutable. Once a tuple is set, it cannot be modified, which is useful for data integrity. A tuple can be edited by converting it first to list than making desired changes and converting back to a tuple.
- **Dictionaries:** Dictionaries act as Hashmaps to store key-value pairings. Learned how to access values via keys and how to dynamically modify the dictionary.

1.2 Control Structures and Operators

In the section various operators were introduced like `+` for addition, `*` for multiplication. Key point is that division always returns a float, and floor division (\\\\) returns the integer part. Some of the logical operators like `and` which returns `true` only when all the values are `true`, and `or` which returns `true` when one of the value is `true`.

Also got to know about:

- **Conditionals:** Learned how to use `if`, `elif` and `else` to impose some condition to do some work it the code.
- **Loops:** Utilized `while` loops for boolean conditions and `for` loops to iterate over sequences using iterators like `range()`.
- In all the above conditionals and loops like `if`, `elif`, `while` and `for` identation is used.

1.3 Functions and Modules

To define functions in python use the `def` keyword. Learned about arguments for a function, including positional arguments, keyword arguments, and arbitrary arguments (using `*` for tuples and `**` for dictionaries). **Lambda functions**, are small anonymous functions defined using the keyword `lambda`.

Learned to manage code using **Modules** and **Packages**. Imported specific functions using aliases (e.g., `import numpy as np`) to keep the namespace clean and efficient.

1.4 Classes and Objects

Learned about Object-Oriented Programming (OOP) in Python. Learned to define a `class`, use the `__init__` constructor to initialize attributes, and use the `self` parameter to access instance-specific methods and variables.

2 Python Libraries

2.1 NumPy

NumPy is the fundamental and open source library for numerical computing in Python. The primary function of this library is while working with arrays. Developed basic understanding about this library. Its primary object is ndarray.

Key Concepts Learned:

- **Array Creation:** To create arrays in numpy use built-in functions like `np.zeros()` to create arrays with all elements 0, `np.ones()`to create arrays with all elements 0, `np.eye()` (identity matrix), and `np.random.random()` array with random elements.
- **Dimensions and Shape:** To check the number of dimensions use `ndim` and to check the structure of the array use `shape`. Higher dimensional arrays can also be created using `np.array()`.
- **Indexing and Slicing:** Similar to lists, integer indexing and slicing is possible to access subarrays. Slicing creates a "view" of the data, meaning modifications made in the slice affect the original array.
- **Mathematical Operations:** Element-wise arithmetic (addition, subtraction, multiplication) can be performed on arrays. And the `dot()` function can be used for matrix multiplication. Aggregation functions like `np.sum()`, `np.mean()`, and `np.std()` are used to perform statistical analysis on specific axes.

2.2 Pandas

Pandas is essential for data manipulation and analysis. It provides two primary data structures: the **Series** (1D) and the **DataFrame** (2D).

Key Concepts Learned:

- **Pandas Series:** A Series is a one-dimensional labeled array. Series can be created from lists and dictionaries, noting that dictionary keys automatically become the index labels. Custom labels can be created using `index` argument.
- **Pandas DataFrames:** Created DataFrames as 2-dimensional structures similar to Excel tables. Created DataFrames from dictionaries and learned to locate specific rows using the `loc` attribute.

- **File Handling:** Datasets from external storage can be loaded , specifically reading CSV and JSON files into DataFrames. Use `to_string()` to print the entire DataFrame.
- **Statistical Analysis:** Similar to NumPy, built-in Pandas functions can be used to quickly calculate the Mean, Standard Deviation, and Variance of data columns.
- **Data Cleaning:** Data cleaning is used in removing incorrect, corrupt or empty data to prepare standardized datasets for analysis, by doing this we improve the dataset.

2.3 Matplotlib

Matplotlib is a comprehensive, open-source data visualization library for Python used to create static, animated, and interactive plots and graphs. To import Matplotlib in Python, one can use the standard code `(import matplotlib,pyplot as plt)`.

Key Concepts Covered:

- **Plot Types:** Covered the implementation of various plot types for data visualisation and distribution:
  - **Line Plot:** Can be created using `plt.plot()` to visualize trends over a range.
  - **Bar Chart:** Can be generated using `plt.bar()` to compare categorical data.
  - **Histogram:** Can be implemented via `plt.hist()` to display the frequency distribution of a dataset.
  - **Scatter Plot:** Can be created using `plt.scatter()` to observe relationships between two numerical variables.
  - **Pie Chart:** Can be generated using `plt.pie()`, utilizing the `explode` parameter to highlight specific sectors.
- **Customization:** `plt.title()` can be used to add title to the graph to add context, and used `plt.xlabel()` and `plt.ylabel()`0 to name the axis.

3 Introduction to Stock Markets

Understanding of financial markets is necessary for applying data science tools and techniques to financial data to analyse and work on it. This section introduced stock market functioning.

3.1 Market Structure and Intermediaries

The market is divided into the **Primary Market**, where companies issue new securities (IPOs), and the **Secondary Market**. Key regulatory oversight is provided by the **Securities and Exchange Board of India (SEBI)**, which was established to protect investor interests and regulate the securities market.

3.2 Stock Exchanges and Indices

Two major stock exchanges in the Indian financial markets are:

- **Bombay Stock Exchange (BSE):** Established in 1875, The Bombay Stock Exchange is Asia's largest and oldest stock exchange, serving as a platform for trading various financial instruments like stocks, currencies, and derivatives.

- **National Stock Exchange (NSE):** The National Stock Exchange, India's leading financial exchange, was founded in 1992. Its benchmark index is the **NIFTY**, which consists of the top 50 companies.

3.3 Key Financial Terminology

Financial terminologies: that were defined:

- **Market Capitalization:** Market capitalization is the total market value of a publicly traded company's equity, reflecting the stock market's perception of its worth at a specific time.
- **Dividend:** Dividend in finance is a distribution of a portion of a company's earnings, decided by the board of directors, to its shareholders as a return on investment, typically paid in cash, but sometimes as additional shares or other assets.
- **Market Trends:** A market trend is a perceived tendency of the financial markets to move in a particular direction over time.

4 Machine Learning Fundamentals

4.1 Introduction to Machine Learning

A market trend is a perceived tendency of the financial markets to move in a particular direction over time. The different types of Machine learning are:

- **Supervised Learning:** Supervised Learning is a type of Machine Learning where a model learns from labelled data.
- **Unsupervised Learning:** Unsupervised Learning is a type of machine learning where the model works without labelled data. It is used for tasks like clustering.
- **Reinforcement Learning:** In reinforcement learning the model learns through reward and punishment system by trial and error method.

4.2 Supervised Learning

Supervised learning involves learning or finding out the function that maps an input to an output based on the training data provided . The training set of data is where the correct answers are already known.

Core Concepts:

- **Goal:** The objective is to approximate the mapping function (f) so accurately that when new input data (x) is given to the model, the system can predict the output variable (Y) with a good precision($Y = f(x)$).
- **Input vs. Output:** Input variables (x) are commonly called features, while the output variable (Y) is called the target or label.
- **Learning Process:** The algorithm iteratively makes predictions on the training data and make adjustments based on the feedback from the known labels. The learning stops when the algorithm achieves an acceptable level of precision.

Problem Types:

- **Classification:** A problem is a classification problem when the output variable is a category, such as "spam" or "not spam", or "Disease" and "No Disease".

- **Regression:** A problem is a regression when the output variable is a real, continuous value, such as "dollars" or "weight".

4.3 Simple Linear Regression (Theory)

Simple Linear Regression (SLR) is a statistical method used to model the relationship between a single independent variable (input) and a dependent variable (output).

Mathematical Formulation: The relationship is expressed as:

$$Y = \beta_0 + \beta_1 X + \epsilon$$

Where:

- Y is called the dependent variable (Target).
- X is the called independent variable (Predictor).
- β_0 is called the Intercept.
- β_1 is called the Slope (coefficient).
- ϵ is the error term.

Simple Linear regression finds the "Best Fit Line" that minimizes the error between the actual values and the predicted values using the **Ordinary Least Squares (OLS)** method.

4.4 Linear Regression Implementation in Python

The practical implementation of Linear Regression in Python involves utilizing libraries like `numpy`, `pandas`, `matplotlib`, and `scikit-learn`.

Implementation Pipeline:

1. **Importing Libraries:** The essential libraries are imported: `numpy` and `matplotlib.pyplot` for calculation and plotting, `pandas` for data management, and the `LinearRegression` class from `sklearn.linear_model`.
2. **Data Loading and Analysis:** Data is loaded using `pd.read_csv()`. Before modeling, it is critical to understand the data structure using:
    - `dataset.shape` to view dimensions.
    - `dataset.head()` to inspect the first few rows.
    - `dataset.describe()` to view statistical details like percentile, mean, and std.
    - `dataset.plot()` to visualize relationships in 2D space.
3. **Data Preparation:** The data is divided into attributes (inputs) and labels (outputs).
    - The `iloc` function is used to extract specific columns for X (attributes) and y (labels).
    - The data is split into training and test sets using the `train_test_split` method from `sklearn.model_selection`, typically with a test size of 20%.
4. **Model Training:** The algorithm is initialized using `regressor = LinearRegression()`. The `regressor.fit(X_train, y_train)` method is called to train the model, which finds the optimal intercept and slope.
5. **Parameter Retrieval:** After training, the model parameters can be retrieved:
    - `regressor.intercept_` retrieves the y-intercept.

- `regressor.coef_` retrieves the slope coefficient, indicating the impact of a unit change in X on y.

6. **Prediction and Evaluation:** `y_pred = regressor.predict(X_test)` are used to make predictions. The model's performance can be evaluated by comparing `y_test` (actual) vs `y_pred` (predicted) using metrics such as:
   - **Mean Absolute Error (MAE):** Mean ansolute error is the the average of the absolute errors.
   - **Mean Squared Error (MSE):** Mean squared error is the average of squared errors.
   - **Root Mean Squared Error (RMSE):** Root Mean SSquared Error is the square root of the mean of squared errors.

4.5 Multiple Linear Regression

Multiple Linear Regression (MLR) include multiple independent input variables $(x_1, x_2, \dots x_n)$ to predict a single dependent output variable y.

The Equation:

$$y = b_0 + b_1x_1 + b_2x_2 + \cdots + b_nx_n$$

Key Assumptions: For the model to be reliable, five assumptions must be met:

1. **Linearity:** The relationship between variables is linear.
2. **Homoscedasticity:** Constant error variance.
3. **Multivariate Normality:** Normally distrbuted residuals.
4. **Independence of Errors:** Ther should be no correlation between error terms.
5. **Lack of Multicollinearity:** The independent variables should not be influenced by other independent variables highly.

Handling Categorical Data The Dummy Variable Trap: Since equations require numbers, categorical text data (like "State") must be converted into numbers using **Dunmy Variables** (0s and 1s). However, we must be careful of the **Dummy Variable Trap**, where variables become highly correlated. To avoid this, we always include $n-1$ dummy variables for n categories (e.g., if there are 2 states, we only keep 1 column).

4.6 Implementing Multiple Linear Regression

The implementation is like simple regression but this includes preprocessing for categorical data.

1. **Encoding Categorical Data:** We use `OneHotEncoder` combined with `ColumnTransformer` to transform categorical columns (like "State") into binary vectors. We use `remainder='passthrough'` to keep the numerical columns intact.
2. **Avoiding the Trap:** To satisfy the n-1 rule and prevent multicollinearity we mannualy remove one of the dummy variables.
3. **Training and Prediction:** We split the data and train it using the `LinearRegression` class, which can handle multiple features. Model evaluation is done by comparing the predicted and actual values.

4.7 Classification (Logistic Regression)

Classification is a type of supervised learning where the aim is to predict separable discrete class labels instead of continuous values.

Concept: Classification predicts a category rather than a value like a numberr.

- **Logistic Regression:** Despite its name, this is a classification algorithm. It estimates the probability that an instance belongs to a specific class.

Mechanism:

- **Sigmoid Function:** The core of logistic regression is the sigmoid function, which maps any real value into a range between 0 and 1 ($\sigma(x) = \frac{1}{1+e^{-x}}$).
- **Decision Boundary:** If the predicted probability is greater than 50% ($P > 0.5$), the instance is classified as positive (1); otherwise, it is classified as negative (0).

4.8 Implementing Classification in Python

The workflow for implementing classification mirrors regression but utilizes different classes and evaluation metrics.

1. **Data Preprocessing:** The dataset is loaded and split into training and test sets. Since classification algorithms are sensitive to the scale of data, we perform **Feature Scaling** using `StandardScaler` to normalize the range of independent variables.
2. **Training the Model:** We use the `LogisticRegression` class from `sklearn.linear_model`. We create an object (`classifier`) and fit it to the training data using `classifier.fit(X_train, y_train)`.
3. **Prediction:** We predict the class labels for the test set using `y_pred = classifier.predict(X_test)`.
4. **Evaluation (Confusion Matrix):** To evaluate performance, we use the **Confusion Matrix**, a table that summarizes correct and incorrect predictions.
   - We import `confusion_matrix` from `sklearn.metrics`.
   - The matrix displays True Positives (TP), True Negatives (TN), False Positives (FP), and False Negatives (FN), allowing us to calculate accuracy.

5 Model Performance and Optimization

Building a model is not only about feeding it data but ensuring it generalizes well to new, unseen information. We explored two critical concepts that affect model performance: Overfitting and Regularization.

5.1 Overfitting and Underfitting

Overfitting occurs when a model learns the training data too closely, capturing not just the underlying patterns (Signal) but also the random fluctuations and inaccuracies (Noise).

- **Symptoms:** The model performs exceptionally well on training data (high accuracy) but fails to predict accurately on test data (low accuracy). It is analgous to "fitting into oversized apparel"—it fits the specific shape but is not practical for general use.
- **Causes:** Using a model that is too complex or training with too many features relative to the amount of data.

Underfitting is the opposite scenario, where the model is too simple to capture the underluing structure of the data, resulting in poor performance on both training and test sets.

5.2 Regularization

Regularization is a technique used to prevent overfitting by discouraging the learning of a model that is too complex or flexible. Mathematically, it works by adding a penalty term to the cost function that "shrinks" the coefficient estimates towards zero.

We examined two primary types of regularization:

- **Ridge Regression (L2 Regularization):** This adds a penalty equivalent to the *square* of the magnitude of coeficients. It shrinks coefficients close to zero but rarely makes them exactly zero. It is usful when we want to retain all features but reduce their impact.
- **Lasso Regression (L1 Regularization):** This adds a penalty equivalent to the *absolute value* of the magnitude of coefficients. A key characteristic of Lasso is that it can shrink coefficients *exactly* to zero. This effectively performs feature selection by removing irelevant variables entirely, leading to "sparse" solutions.

5.3 Unsupervised Learning

Unsupervised Learning involves training a machine using data that is neither classified nor labeled. The algorithm is left to act on the data without guidance.

Key Characteristics:

- **Unlabelled Dataset:** The system inputs raw data without knowing the correct output in advance.
- **Self-Learning:** Unlike supervised learning where a "teacher" corrects the model, here the algorithm autonomously discovers hidden patterns and structures within the data.

Primary Types:

- **Clustering:** In clustering objects of similar types are grouped(clustered) together.
- **Association Mining:** The task of finding interesting relationships or rules between variables in large databases (e.g., "People who buy Bread also buy Butter").

5.4 K-Means Clustering

K-Means learning is one of the simplest machine learning algorithms. It is used to solve the clustering problem. It is done by dividing a dataset into k distinct, non overlapping subgroups.

How it Works: The algorithm follows an iterative process to assign data points to clusters:

1. **Initialization:** Specify the number of clusters k and randomly initialize k centroids (center points).
2. **Assignment:** Calculate the distance between each data point and the centroids. Assign each point to the cluster of the nearest centroid.
3. **Update:** Recalculate the new centroids by taking the mean of all data points assigned to that cluster.
4. **Repeat:** Steps 2 and 3 are repeated until the centroids no longer change (convergence).

This section details advanced methodologies utilized in financial market analysis and alternative classification algorithms.

5.5 K-Nearest Neighbors (KNN)

K-Nearest Neighbors (KNN) is a simple, supervised learning algorithm that can be used for both classification and regression. It operates on the principle that similar data points exist in close proximity to each other.

[Image of K-nearest neighbors classification]

Working Principle: KNN is often described as a "lazy learner" because it does not learn a discriminative function from the training data but instead stores the entire dataset. When a prediction is required:

1. **Distance Calculation:** The algorithm calculates the distance between the new data point and all points in the training set, commonly using Euclidan distance.
2. **Finding Neighbors:** It identifies the k nearest data points (neghbors).
3. **Voting:** For classification, it assigns the class that is most common among the k neighbors. For regression, it calculates the average of the neighbors' values.

5.6 Technical Analysis

To define trade opportunities by analyzing market activty, specifically price and volume, traders use a popular technique called Technical Analysis(TA). Technical Analysis is used to develop directional point of view with respect to entry, exit and risk.

Key Indicator: VWAP The **Volume Weighted Average Price (VWAP)** is a crucial intraday indicator. It calculates the average price a stock has tradd at throughout the day, weighted by volume. It is used to gauge the eficiency of order execution and the trend direction.

- If the current price is **below** VWAP, the intraday trend is generally considered down.
- If the current price is **above** VWAP, the trend is generally considered up.

5.7 Financial Data Acquisition (yfinance)

Accessing real-time and historical financial data is critical for market analysis. We explored `yfinance`, a Python library that retrieves data directly from Yahoo Finance.

Setup and Usage: The library is installed via `pip install yfinance` and typically imported with the alias `yf`. The core class, `Ticker`, allows retrieval of market data for specific symbols (e.g., "MSFT", "TSLA").

- `yf.Ticker("Symbol")`: Initializes the object.
- `.history(period="...")`: Fetches historical data (Open, High, Low, Close, Volume) for a specified duration (e.g., "1mo", "6mo").

Data Visualization Application: Pandas is used to process the data that was fetched. Methods like `resample()` to aggregate metrics (e.g., summing trading volume by month) can be used. To identify trends like monthly trading volume distributions, from the processed data Matplotlib can be used to visualise it.

6 Recurrent Neural Networks and LSTMs

Standard neural networks and feedforward networks have a major limitation: they do not have persistence. They treat each input as independent, which is problematic for sequential data like time-series financial data or natural language. Recurrent Neural Networks (RNNs) and Long Short-Term Memory (LSTMs) address this issue.

6.1 Recurrent Neural Networks (RNNs)

RNNs are networks with loops in them, allowing information to persist. A loop allows information to be passed from one step of the network to the next.

The Problem of Long-Term Dependencies: While RNNs work well when the gap between the relevant information and the point where it is needed is small, they struggle as that gap grows. For example, predicting the last word in "The clouds are in the *sky*" is easy, but predicting the language in "I grew up in France... [long context] ... I speak *French*" is difficult for standard RNNs due to the vanishing gradient problem.

6.2 Long Short-Term Memory (LSTM) Networks

LSTMs are a special kind of RNN, explicitly designed to avoid the long-term dependency problem. Remembering information for long periods of time is their default behavior, not something they struggle to learn.

The Core Idea: The Cell State The key to LSTMs is the **cell state**, the horizontal line running through the top of the diagram. It acts like a conveyor belt, allowing information to flow down the chain with only minor linear interactions. This makes it easy for information to flow unchanged.

6.3 Step-by-Step LSTM Walkthrough

The LSTM removes or adds information to the cell state, carefully regulated by structures called **gates**.

1. **Forget Gate Layer:** The first step is to decide what information to throw away from the cell state. This decision is made by a sigmoid layer called the "forget gate layer."

   $$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$

   It outputs a number between 0 and 1. A 1 represents "completely keep this" while a 0 represents "completely get rid of this."

2. **Input Gate Layer:** The next step is to decide what new information to store in the cell state. This has two parts:

   - A sigmoid layer (input gate) decides which values to update (i_t).
   - A tanh layer creates a vector of new candidate values ($\tilde{C}_t$) that could be added to the state.

3. **Updating the Cell State:** We update the old cell state (C_{t-1}) into the new cell state (C_t). We multiply the old state by f_t (forgetting things we decided to forget) and add $i_t * \tilde{C}_t$ (adding the new candidate values, scaled by how much we decided to update each state value).

   $$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

4. **Output Gate Layer:** Finally, we decide what we are going to output. The output is a filtered version of the cell state.

   - First, a sigmoid layer decides what parts of the cell state we are going to output (o_t).

- Then, the cell state is put through a **tanh** function (to push values between -1 and 1) and multiplied by the output of the sigmoid gate.

$$h_t = o_t * \tanh(C_t)$$