**Python**

**1. Importing Libraries**

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

import numpy as np

import warnings

* **pandas**: A library for data manipulation and analysis.
* **matplotlib.pyplot**: A plotting library for creating static, animated, and interactive visualizations in Python.
* **seaborn**: A data visualization library based on matplotlib, used for making statistical graphics.
* **numpy**: A library for numerical computations.
* **warnings**: A module that controls warnings during code execution.

**2. Creating the Data**

data = {

'Rank': list(range(1, 35)),

'Throw': [

'89.94m', '89.45m', '89.34m', '89.30m', '89.08m', '88.88m',

'88.77m', '88.67m', '88.44m', '88.39m',

'88.36m', '88.17m', '88.13m', '88.07m', '88.06m', '88.00m',

'87.86m', '87.80m', '87.73m', '87.66m',

'87.58m', '87.46m', '87.43m', '87.03m', '87.00m', '86.92m',

'86.84m', '86.79m', '86.69m', '86.67m',

'86.65m', '86.52m', '86.48m', '86.47m'

],

'Competition': [

'Stockholm Diamond League 2022', 'Paris 2024 Olympics - F',

'Paris 2024 Olympics - Q', 'Paavo Nurmi Games (Turku)',

'Lausanne Diamond League 2022', 'Asian Games 2023 (Hangzhou)',

'World Athletics Championships 2023 (Budapest) - Q',

'Doha Diamond League 2023', 'Zurich Diamond League Final

2022', 'World Athletics Championships 2022 (Oregon) - Q',

'Doha Diamond League 2024', 'World Athletics Championships

2023 (Budapest) - F', 'World Athletics Championships 2022 (Oregon) -

F',

'Indian Grand Prix 3 (Patiala)', 'Asian Games 2018 (Jakarta)',

'Zurich Diamond League Final 2022', 'ACNW League Meeting 1

(Potchefstroom)',

'Federation Cup (Patiala)', 'World Athletics Championships

2023 (Budapest) - F', 'Lausanne Diamond League 2023',

'Tokyo 2020 Olympics - F', 'Stockholm Diamond League 2022',

'Doha Diamond League 2018', 'Tokyo 2020 Olympics - F',

'Zurich Diamond League Final 2022', 'Paavo Nurmi Games

(Turku)', 'Stockholm Diamond League 2022', 'Kuortane Games 2021',

'Kuortane Games 2022', 'Stockholm Diamond League 2022', 'Tokyo

2020 Olympics - Q', 'Doha Diamond League 2023',

'World U20 Championships 2016 (Bydgoszcz)', 'Commonwealth

Games 2018 (Gold Coast)'

],

'Date': [

'June 30, 2022', 'August 8, 2024', 'August 6, 2024', 'June 14,

2022', 'August 26, 2022', 'October 4, 2023',

'August 25, 2023', 'May 5, 2023', 'September 8, 2022', 'July

21, 2022', 'May 10, 2024', 'August 27, 2023',

'July 23, 2022', 'March 5, 2021', 'August 27, 2018',

'September 8, 2022', 'January 28, 2020', 'March 17, 2021',

'August 27, 2023', 'June 30, 2023', 'August 7, 2021', 'June

30, 2022', 'May 4, 2018', 'August 7, 2021',

'September 8, 2022', 'June 14, 2022', 'June 30, 2022', 'June

26, 2021', 'June 18, 2022', 'June 30, 2022',

'August 4, 2021', 'May 5, 2023', 'July 23, 2016', 'April 14,

2018'

]

}

* A dictionary data is created containing three keys: Rank, Throw, Competition, and Date. Each key is associated with a list of values.
* The Rank key contains a list of integers from 1 to 34.
* The Throw key contains a list of javelin throw distances in meters.
* The Competition key contains the names of the competitions where these throws occurred.
* The Date key contains the dates of these competitions.

**3. Creating a DataFrame**

df = pd.DataFrame(data)

* Converts the dictionary data into a pandas DataFrame df, a table-like data structure that allows for easy manipulation and analysis of the data.

**4. Initial DataFrame Operations**

python

Copy code

df = df.drop('Rank', axis=1)

df.shape

df.columns

* **df.drop('Rank', axis=1)**: Removes the Rank column from the DataFrame.
* **df.shape**: Returns the dimensions of the DataFrame (rows, columns).
* **df.columns**: Returns the column names of the DataFrame.

**5. Data Cleaning and Transformation**

df['Throw'] = df['Throw'].str.replace('m', '').astype(float)

df.info()

* **df['Throw'].str.replace('m', '')**: Removes the letter 'm' (meters) from the Throw column.
* **astype(float)**: Converts the Throw column values to float data type for numerical operations.
* **df.info()**: Displays a summary of the DataFrame, including the data types and non-null counts of each column.

**6. Statistical Summary**

df.describe()

* Provides a statistical summary of the Throw column, including count, mean, standard deviation, minimum, 25th percentile, median, 75th percentile, and maximum values.

**7. Visualization: Distribution of Throw Distances**

plt.figure(figsize=(10, 6))

sns.histplot(df['Throw'], bins=10, kde=True)

plt.title('Distribution of Throw Distances')

plt.xlabel('Throw Distance (m)')

plt.ylabel('Frequency')

* **plt.figure(figsize=(10, 6))**: Sets the figure size for the plot.
* **sns.histplot(df['Throw'], bins=10, kde=True)**: Creates a histogram of the Throw distances with 10 bins and a kernel density estimate (KDE) line.
* **plt.title, plt.xlabel, plt.ylabel**: Sets the title and labels for the x and y axes.

**8. Adding Mean and Median Lines to the Plot**

mean\_throw = df['Throw'].mean()

median\_throw = df['Throw'].median()

plt.axvline(mean\_throw, color='r', linestyle='--', label=f'Mean: {mean\_throw:.2f} m')

plt.axvline(median\_throw, color='g', linestyle='--', label=f'Median: {median\_throw:.2f} m')

plt.legend()

plt.show()

* **mean\_throw = df['Throw'].mean()**: Calculates the mean throw distance.
* **median\_throw = df['Throw'].median()**: Calculates the median throw distance.
* **plt.axvline**: Adds vertical lines at the mean (red, dashed) and median (green, dashed) throw distances.
* **plt.legend()**: Adds a legend to the plot.
* **plt.show()**: Displays the plot.

**9. Visualization: Number of Top Throws by Competition**

plt.figure(figsize=(14, 8))

ax = sns.countplot(y='Competition', data=df, order=df['Competition'].value\_counts().index)

plt.title('Number of Top Throws by Competition')

plt.xlabel('Number of Throws')

plt.ylabel('Competition')

for p in ax.patches:

ax.annotate(f'{p.get\_width()}', (p.get\_width() + 0.3, p.get\_y() + p.get\_height() / 2), va='center')

plt.show()

* **plt.figure(figsize=(14, 8))**: Sets the figure size for the plot.
* **sns.countplot**: Creates a count plot showing the number of top throws by competition.
* **order=df['Competition'].value\_counts().index**: Orders the competitions by the number of top throws.
* **ax.annotate**: Adds annotations (numbers) to each bar in the plot.
* **plt.show()**: Displays the plot.

**Machine learning**

**10. Preparing the Data for Machine Learning**

Before applying any machine learning models, the data should be prepared. This involves handling missing values, encoding categorical variables, splitting the data into training and testing sets, and scaling the features.

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler, LabelEncoder

* **train\_test\_split**: Used to split the dataset into training and testing sets.
* **StandardScaler**: Standardizes features by removing the mean and scaling to unit variance.
* **LabelEncoder**: Encodes categorical variables as integers.

**Example Data Preparation:**

# Encoding the categorical variable 'Competition'

le = LabelEncoder()

df['Competition'] = le.fit\_transform(df['Competition'])

# Splitting the data

X = df.drop(columns=['Throw'])

y = df['Throw']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Feature Scaling

scaler = StandardScaler()

X\_train = scaler.fit\_transform(X\_train)

X\_test = scaler.transform(X\_test)

**11. Applying Machine Learning Models**

Several ML models can be used to predict the javelin throw distance (Throw) based on the competition and other features. Here are some common models:

**a) Linear Regression**

from sklearn.linear\_model import LinearRegression

model = LinearRegression()

model.fit(X\_train, y\_train)

y\_pred = model.predict(X\_test)

* **Concept**: Linear Regression predicts the output by finding a linear relationship between the input features and the target variable.

**b) Decision Tree Regressor**

from sklearn.tree import DecisionTreeRegressor

model = DecisionTreeRegressor()

model.fit(X\_train, y\_train)

y\_pred = model.predict(X\_test)

* **Concept**: Decision Trees model the data by splitting it into smaller and smaller subsets, based on the feature that results in the most significant information gain.

**c) Random Forest Regressor**

from sklearn.ensemble import RandomForestRegressor

model = RandomForestRegressor(n\_estimators=100, random\_state=42)

model.fit(X\_train, y\_train)

y\_pred = model.predict(X\_test)

* **Concept**: Random Forest is an ensemble learning method that creates multiple decision trees and merges them to get a more accurate and stable prediction.

**d) Support Vector Regressor (SVR)**

from sklearn.svm import SVR

model = SVR(kernel='rbf')

model.fit(X\_train, y\_train)

y\_pred = model.predict(X\_test)

* **Concept**: SVR tries to find a function that deviates from the true target values by a value no greater than a certain threshold. It's based on the concept of support vectors, which are the data points that lie closest to the decision boundary.

**12. Evaluating the Model Performance**

After fitting a model, it’s crucial to evaluate its performance using metrics like Mean Absolute Error (MAE), Mean Squared Error (MSE), and R-squared.

from sklearn.metrics import mean\_absolute\_error, mean\_squared\_error, r2\_score

print('Mean Absolute Error:', mean\_absolute\_error(y\_test, y\_pred))

print('Mean Squared Error:', mean\_squared\_error(y\_test, y\_pred))

print('R-squared:', r2\_score(y\_test, y\_pred))

* **mean\_absolute\_error**: Measures the average magnitude of the errors in a set of predictions, without considering their direction.
* **mean\_squared\_error**: Measures the average of the squares of the errors—that is, the average squared difference between the estimated values and the actual value.
* **r2\_score**: Provides an indication of how well the model explains the variance in the target variable; ranges from 0 to 1.

**13. Hyperparameter Tuning**

To improve model performance, hyperparameter tuning can be performed using techniques like Grid Search or Random Search.

from sklearn.model\_selection import GridSearchCV

param\_grid = {'n\_estimators': [50, 100, 200], 'max\_depth': [None, 10, 20]}

grid\_search = GridSearchCV(RandomForestRegressor(random\_state=42), param\_grid, cv=5)

grid\_search.fit(X\_train, y\_train)

best\_model = grid\_search.best\_estimator\_

* **Concept**: Grid Search finds the optimal hyperparameters by evaluating every combination of a given set of hyperparameter values.

**14. Cross-Validation**

To ensure the model's generalizability, cross-validation can be used.

from sklearn.model\_selection import cross\_val\_score

scores = cross\_val\_score(model, X, y, cv=5)

print('Cross-Validation Scores:', scores)

* **Concept**: Cross-Validation divides the dataset into k subsets (folds) and trains the model on k-1 folds while using the remaining fold for testing. This process is repeated k times, and the average performance is reported.

**15. Feature Importance**

In some models like Random Forest, it’s possible to extract feature importance to understand which features are most influential in predicting the target variable.

feature\_importances = model.feature\_importances\_

features = X.columns

plt.barh(features, feature\_importances)

plt.xlabel('Feature Importance')

plt.ylabel('Features')

plt.title('Feature Importance in Predicting Throw Distance')

plt.show()

* **Concept**: Feature Importance measures how much a given feature contributes to the prediction made by the model. Higher values indicate greater importance.

**16. Conclusion**

* This workflow outlines how you can extend the analysis of javelin throw distances with machine learning models. By preparing the data, applying models, evaluating their performance, and tuning hyperparameters, you can gain insights and make predictions about future performances based on historical data.

**General Terminologies**

1. **DataFrame (df)**:
   * A two-dimensional, size-mutable, and potentially heterogeneous tabular data structure with labeled axes (rows and columns) in pandas. It is similar to a table in a database or an Excel spreadsheet.
2. **CSV (Comma-Separated Values)**:
   * A file format used to store tabular data in plain text, where each line represents a data record, and each record consists of one or more fields separated by commas.
3. **Plotting**:
   * Visualization of data using graphs like line plots, bar charts, scatter plots, etc., to identify trends, patterns, and outliers.
4. **Date Parsing**:
   * The process of converting a string representation of a date into a date object in programming. It helps in handling dates in a standard format.
5. **groupby**:
   * A pandas method used to split the data into groups based on some criteria (e.g., grouping by a specific column value). Operations like aggregation or transformation can then be performed on these groups.

**Data Visualization Terminologies**

1. **Histogram**:
   * A graphical representation of the distribution of numerical data. It is an estimate of the probability distribution of a continuous variable.
2. **Line Plot**:
   * A type of chart used to show information that changes over time. It plots data points connected by a straight line.
3. **Scatter Plot**:
   * A type of plot that shows the relationship between two variables using Cartesian coordinates. Each point represents an observation in the dataset.
4. **Correlation**:
   * A statistical measure that describes the extent to which two variables are linearly related. It ranges from -1 to 1, where 1 indicates a perfect positive relationship, -1 indicates a perfect negative relationship, and 0 indicates no linear relationship.

**Python-Specific Terminologies**

1. **Import**:
   * The statement used to bring external libraries or modules into a Python script so that their functions and classes can be used.
2. **Function**:
   * A block of reusable code that performs a specific task. Functions can take input parameters and return an output.
3. **plt.show()**:
   * A command used in Matplotlib (a plotting library in Python) to display the plots.
4. **sns.set()**:
   * A command from the Seaborn library used to set the aesthetic parameters for the plots, such as background, grid lines, and color palettes.
5. **read\_csv()**:
   * A pandas function used to read a CSV file into a DataFrame.
6. **dropna()**:
   * A pandas function that removes missing values from a DataFrame.
7. **mean()**:
   * A pandas function that calculates the average value of the elements in a DataFrame column.

**Machine Learning Terminologies**

1. **Machine Learning Model**:
   * An algorithm or mathematical model that is trained on historical data to make predictions or decisions without being explicitly programmed to perform the task.
2. **Supervised Learning**:
   * A type of machine learning where the model is trained on labeled data, i.e., the data has both input features and the corresponding output (target variable). The model learns the mapping between inputs and outputs during training.
3. **Training and Testing Data**:
   * **Training Data**: The subset of the dataset used to train a model. The model learns from this data.
   * **Testing Data**: A different subset of the dataset used to evaluate the model's performance. The model has not seen this data during training.
4. **Feature**:
   * An individual measurable property or characteristic of the data, often represented as a column in a dataset.
5. **Target Variable (Label)**:
   * The output or the variable that the model is trying to predict. For example, in a regression problem, it could be the javelin throw distance.
6. **Linear Regression**:
   * A linear approach to modeling the relationship between a dependent variable (target) and one or more independent variables (features). The goal is to find the line (or hyperplane in higher dimensions) that best fits the data.
7. **Decision Tree**:
   * A non-linear model that splits the data into subsets based on the feature that gives the maximum information gain. Each node in the tree represents a feature, and the branches represent the outcome of a decision.
8. **Random Forest**:
   * An ensemble learning method that creates multiple decision trees (a "forest") and combines their predictions. It is more robust and accurate than a single decision tree.
9. **Support Vector Machine (SVM)**:
   * A supervised learning model that finds the hyperplane that best separates the classes in the data. For regression tasks, it tries to fit the data within a certain margin of tolerance.
10. **Metrics for Model Evaluation**:
    * **Mean Absolute Error (MAE)**: The average of the absolute differences between the predicted values and the actual values.
    * **Mean Squared Error (MSE)**: The average of the squared differences between the predicted values and the actual values. It penalizes larger errors more than MAE.
    * **R-squared (R²)**: A statistical measure that represents the proportion of the variance for the target variable that is explained by the features in the model.
11. **Hyperparameters**:
    * Parameters that are not learned from the data but are set before the learning process begins (e.g., the number of trees in a random forest). Hyperparameter tuning involves finding the best combination of these values to improve the model's performance.
12. **Cross-Validation**:
    * A technique used to evaluate the performance of a machine learning model by splitting the data into multiple subsets (folds). The model is trained on some folds and tested on the remaining ones. This process is repeated several times to get an average performance metric.
13. **Grid Search**:
    * A method for hyperparameter tuning where every possible combination of the provided hyperparameters is tested to find the one that results in the best model performance.
14. **Feature Importance**:
    * A technique used to measure the impact of each feature on the model's predictions. It helps in understanding which features are most influential in predicting the target variable.
15. **Label Encoding**:
    * A process of converting categorical variables into numerical values by assigning a unique integer to each category. This is necessary because most machine learning algorithms can only operate on numerical data.
16. **Scaling (Standardization)**:
    * The process of transforming the features to have a mean of zero and a standard deviation of one. This is often necessary when features have different units or scales.
17. **Overfitting**:
    * A situation where a model performs well on the training data but fails to generalize to new, unseen data. It happens when the model is too complex and captures noise in the training data.
18. **Underfitting**:
    * When a model is too simple and cannot capture the underlying patterns in the data, leading to poor performance on both training and testing data.