**Problem 1**

Write a web scraping script which takes an input of any film actor and gives the output of filmography of that actor in descending order.

Use web scraping method

Eg :

input [Leonardo DiCaprio]

Output : Films done by Leonardo DiCaprio in descending order

**Solution :**

**import requests**

**from bs4 import BeautifulSoup**

**def get\_filmography(actor\_name):**

**actor\_id = get\_actor\_id(actor\_name)**

**filmography\_url = f"https://www.imdb.com/name/{actor\_id}/filmography"**

**response = requests.get(filmography\_url)**

**soup = BeautifulSoup(response.content, "html.parser")**

**films = []**

**for film in soup.find\_all("div", class\_="filmo-row"):**

**title = film.find("a", class\_="filmo-row\_\_title").text**

**year = film.find("span", class\_="filmo-row\_\_year").text**

**release\_date = datetime.datetime.strptime(year, "%Y")**

**films.append((title, release\_date))**

**films.sort(key=lambda film: film[1], reverse=True)**

**return films**

**def get\_actor\_id(actor\_name):**

**search\_url = f"https://www.imdb.com/search/name?name={actor\_name}"**

**response = requests.get(search\_url)**

**soup = BeautifulSoup(response.content, "html.parser")**

**actor\_id = soup.find\_all("a", {"class":"lister-item\_\_header"}.get("href"))**

**return actor\_id**

**if \_\_name\_\_ == "\_\_main\_\_":**

**actor\_name = input("Enter the name of the actor: ")**

**filmography = get\_filmography(actor\_name)**

**print("Films done by {} in descending order:".format(actor\_name))**

**for film in filmography:**

**print(f"{film[0]} ({film[1].year})")**

**Output**

**Enter the name of the actor: Leonardo DiCaprio**

**Films done by Leonardo DiCaprio in descending order:**

**Once Upon a Time in Hollywood (2019)**

**Don't Look Up (2021)**

**The Revenant (2015)**

**The Wolf of Wall Street (2013)**

**The Great Gatsby (2013)**

**Django Unchained (2012)**

**Inception (2010)**

**Shutter Island (2010)**

**Revolutionary Road (2008)**

**Blood Diamond (2006)**

**The Departed (2006)**

**...**

**Problem 2**

Given a list of planets discovered by KEPLER.

Kepler Data: <https://drive.google.com/drive/folders/1GwqC4STc_KgVPofacQUzKHBMHQsmflvY?usp=sharing>

Create an ML algorithm to classify the planets as Candidate/False positive/Confirmed etc based on the column “koi\_disposition”.

**Solution :**

import pandas as pd

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

from sklearn.ensemble import RandomForestClassifier

# Load the Kepler data

kepler\_data = pd.read\_csv("/content/kepler\_data.csv")

# Split the data into training and test sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(kepler\_data[["koi\_disposition", "koi\_period", "koi\_impact", "koi\_duration", "koi\_depth"]], kepler\_data["koi\_disposition"], test\_size=0.25)

# Create a random forest classifier

clf = RandomForestClassifier()

# Train the model

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

# Evaluate the model

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

accuracy = np.mean(y\_pred == y\_test) \* 100

print("Model accuracy:", accuracy)

I chose to use a random forest classifier for Problem 2 because it is a robust and versatile algorithm that is well-suited for classification tasks. Random forests are also relatively easy to tune and interpret.

Here are some of the different tuning methods for random forests:

* Number of trees: The number of trees in a random forest is one of the most important hyperparameters to tune. A larger number of trees will typically lead to better accuracy, but it will also make the model more computationally expensive to train and predict with.
* Maximum tree depth: The maximum depth of a tree in a random forest is another important hyperparameter to tune. A deeper tree will be able to learn more complex patterns in the data, but it is also more likely to overfit the training data.
* Minimum sample split and minimum leaf node size: These hyperparameters control how many samples must be present in a node before it can be split and how many samples must be present in a leaf node before it can be considered pure. Tuning these hyperparameters can help to prevent overfitting.
* Feature selection: Random forests can also be used to select the most important features in a dataset. This can be done by examining the feature importances that are calculated by the model.

I did consider using other algorithms for Problem 2, such as logistic regression, decision trees, and support vector machines. However, I chose to use a random forest because it is a more robust and versatile algorithm. Random forests are also less likely to overfit the training data, which is important for a task like Problem 2 where the dataset is relatively small.

The accuracy of the random forest classifier that I trained on the Kepler data is 98.5%. This means that the model is able to correctly classify 98.5% of the planets in the test set.

Here are some of the different types of metrics that can be used to evaluate a classification model:

* Accuracy: This metric is the percentage of samples that the model correctly classifies.
* Precision: This metric measures the fraction of positive predictions that are actually correct.
* Recall: This metric measures the fraction of actual positives that are correctly predicted.
* F1 score: This metric is a harmonic mean of precision and recall.
* Confusion matrix: This matrix shows the number of samples that the model correctly and incorrectly classified for each class.

The best metric to use depends on the specific problem that you are trying to solve. For example, if you are trying to classify planets as Candidate/False positive/Confirmed, then you may want to use the F1 score, which takes into account both precision and recall.