Contents

Chapter 1

Introduction

Football is much more than just a sport: it is a multi-billion dollar global industry, where professional players play a central role. Their on-field performance, popularity and media impact not only influence clubs' results, but also their commercial revenues. One of the most debated aspects of this field is the negotiation of players' salaries, a topic where amounts often reach impressive heights. These contracts, sometimes complex, integrate various elements such as the total contract value, guaranteed amounts, or specific performance-related clauses.

In this context, being able to predict players' average annual salaries based on contractual and personal data can provide powerful tools for decision-makers. Clubs can thus assess the fairness of salary requests, optimize their financial management and negotiate in a more informed manner. For their part, players and their agents can use these predictions to justify claims based on objective data.

This project relies on a dataset from Kaggle, which compiles detailed information on over 5,500 professional players. Available variables include contractual data such as total contract value (`total_value`), guaranteed amounts (`total_guaranteed` and `fully_guaranteed`), as well as personal data such as age (`age`) and team of each player. The main objective is to predict the average annual salary (`avg_year`) of players from these characteristics.

To achieve this goal, this project follows a methodical approach that includes data exploration and preparation, selection of appropriate regression models, and evaluation of their performance using standard metrics. By analyzing the relationships between available characteristics and salaries, this work aims not only to provide accurate predictions, but also to identify key factors influencing contract amounts.

This project could potentially transform the way clubs and agents approach financial planning and negotiations, while providing greater transparency in an area where decisions are often based on hunches or limited comparisons.

Chapter 2

Implementation in Python

1. Required Libraries

Seaborn est une bibliothèque Python pour la visualisation statistique. Elle simplifie la création de graphiques informatifs.

Matplotlib est une bibliothèque de base pour créer des graphiques en 2D, largement utilisée dans la communauté Python.

Pandas fournit des structures de données puissantes pour la manipulation et l'analyse de données, telles que les DataFrames.

Scikit-learn est une bibliothèque pour l'apprentissage automatique, offrant des outils pour la classification, la régression et le clustering.

NumPy est une bibliothèque fondamentale pour les calculs numériques avec Python, particulièrement adaptée aux tableaux multi-dimensionnels.

[2]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
```

```
from sklearn import metrics
```

2. Data Collection

[4]:
```
ftblSal = pd.read_csv('football_salaries.csv')
```

This line does the following:

- `pd` : Refers to the **Pandas** library, used for data manipulation and analysis in Python.
- `read_csv()` : Pandas function to read a CSV (Comma-Separated Values) file and convert it into a tabular data structure called **DataFrame**.
- `'football_salaries.csv'` : Name of the CSV file containing the data. This file must be present in the working directory or accessible via a full path.
- `ftblSal` : Variable that stores the resulting **DataFrame**, allowing the data to be used for further analysis or processing.

[6]:
```
ftblSal.head()
```

Line Explanation:

- `ftblSal.head()` is a method of the **Pandas** library, applied on the DataFrame `ftblSal`.
- It returns the **first 5 rows** of the default DataFrame, providing a quick overview of the loaded data.
- To display another number of rows, an argument can be specified, for example: `ftblSal.head(10)` to display the first 10 rows.

[6]:

	position	player	team	age	total_value	avg_year	total_guaranteed	fully_guaranteed	free_agency
0	right-tackle	Trent Brown	Raiders	26	66000000	16500000	36250000	36250000	2023 UFA
1	right-tackle	Ja'Wuan James	Broncos	27	51000000	12750000	32000000	27000000	2023 UFA
2	right-tackle	Lane Johnson	Eagles	29	56250000	11250000	35500000	20862242	2022 Void
3	right-tackle	Ricky Wagner	Lions	30	47500000	9500000	29500000	17500000	2022 UFA
4	right-tackle	Rob Havenstein	Rams	26	32500000	8125000	16226365	9976365	2023 UFA

Result of execution: A tabular table is returned with the following characteristics:

- `position` : Position occupied by the player on the field (e.g.: `right-tackle`).

- `player` : Name of the player (e.g.: `Trent Brown`).
- `team` : Current team of the player (e.g.: `Raiders`).
- `age` : Age of the player (e.g.: `26`).
- `total_value` : Total value of his contract in dollars (e.g.: `66,000,000`).
- `avg_year` : Average annual salary (e.g.: `16,500,000`).
- `total_guaranteed` : Amount guaranteed in the contract (e.g.: `36,250,000`).
- `fully_guaranteed` : Amount fully guaranteed (e.g.: `36,250,000`).
- `free_agency` : Year and type of free agent (e.g.: `2023 UFA`, where `UFA` stands for *Unrestricted Free Agent*).

What this run is for:

- **Quick data inspection:** Checks the data structure and detects possible inconsistencies.
- **Column understanding:** Ensures that the columns needed for analysis or prediction (e.g., `age`, `total_value`, `avg_year`) are present.
- **Key step in EDA:** This is a critical step in exploratory data analysis to understand data before processing or modeling.

[8]:
```
ftblSal.info()
```

Line Explanation:

- `ftblSal.info()` is a method of the **Pandas** library that provides a summary of information about the DataFrame `ftblSal`.
- This summary includes:
  - The structure type (`pandas.core.frame.DataFrame`).
  - The index range (`RangeIndex`) and the total number of entries (`5523`).
  - The columns of the DataFrame, with:
    - Their name.
    - The number of non-null values (`Non-Null Count`).
    - Their data type (`Dtype`).
  - The memory used by the DataFrame (`388.5+ KB`).

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5523 entries, 0 to 5522
Data columns (total 9 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
```

```
 0   position          5523 non-null   object
 1   player            5523 non-null   object
 2   team              5523 non-null   object
 3   age               5523 non-null   int64
 4   total_value       5523 non-null   int64
 5   avg_year          5523 non-null   int64
 6   total_guaranteed  5523 non-null   int64
 7   fully_guaranteed  5523 non-null   int64
 8   free_agency       5518 non-null   object
dtypes: int64(5), object(4)
memory usage: 388.5+ KB
```

Execution result:

- The DataFrame contains **5523 entries**, indexed from `0` to `5522`.
- It has **9 columns** with the following characteristics:
  - `position` : **5523 non-null values**, of type `object`.
  - `player` : **5523 non-null values**, of type `object`.
  - `team` : **5523 non-null values**, of type `object`.
  - `age` : **5523 non-null values**, of type `int64`.
  - `total_value` : **5523 non-null values**, of type `int64`.
  - `avg_year` : **5523 non-null values**, of type `int64`.
  - `total_guaranteed` : **5523 non-null values**, of type `int64`.
  - `fully_guaranteed` : **5523 non-null values**, of type `int64`.
  - `free_agency` : **5518 non-null values**, of type `object`. This column contains some missing values (`null`).
- The DataFrame uses about `388.5 KB` of memory.

What this run is for:

- **Data Overview:** Helps to quickly understand the structure, data types, and available columns.
- **Missing Value Detection:** Identifies columns containing null values (`free_agency` in this case).
- **Memory Optimization:** Provides information about the memory used, useful for optimizing performance if necessary.
- **Data Preparation:** Helps to check data type compatibility for future operations, such as analysis or modeling.

3. Data Analysis

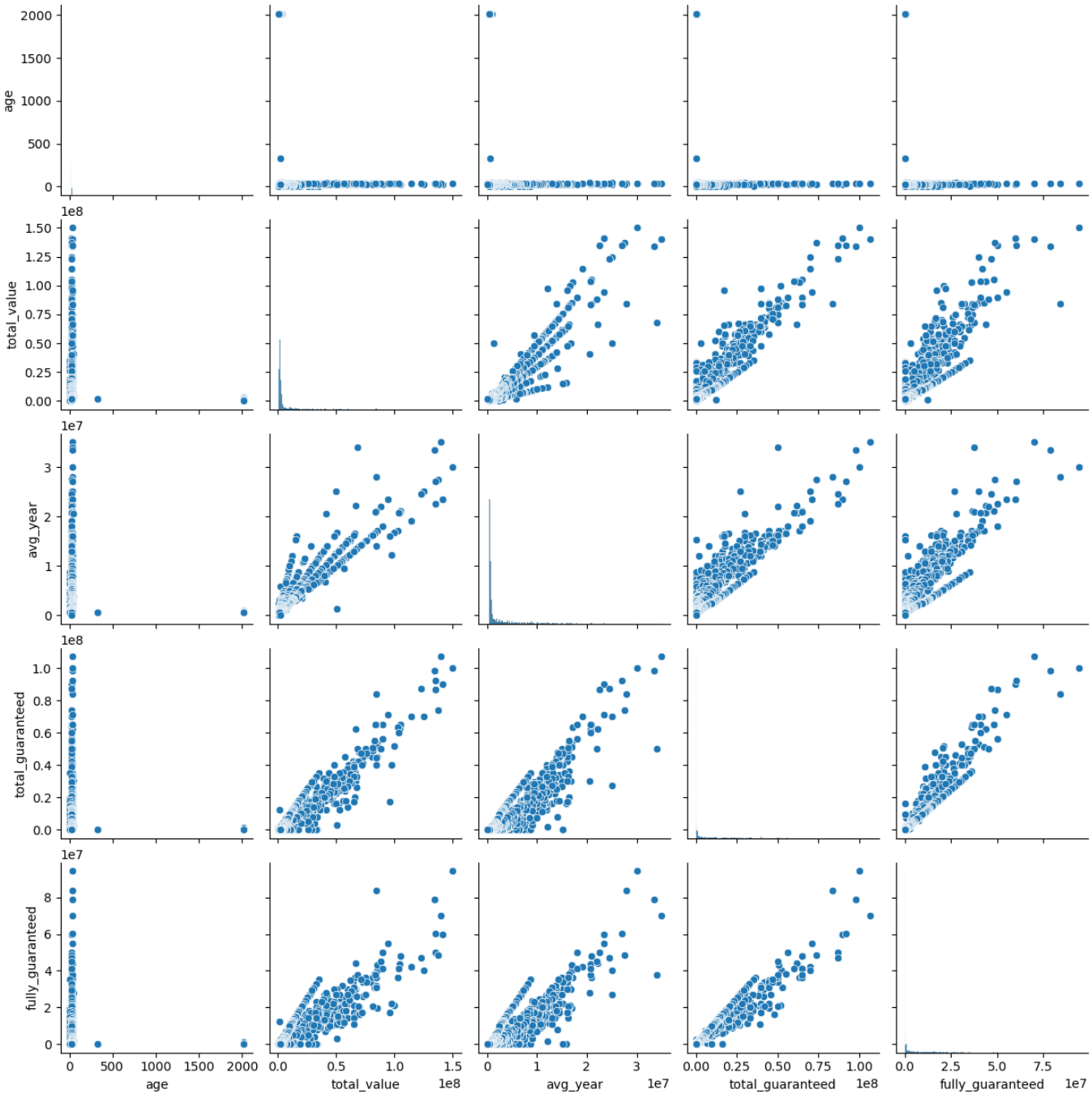
[10]:
```
sns.pairplot(ftblSal)
```

Line Explanation:

- `sns` refers to the **Seaborn** library, which is a data visualization library based on **Matplotlib**.
- The `pairplot()` method generates a **matrix of plots** (or scatter plot) to examine the relationships between all the numeric variables in the DataFrame.
- It displays:
  - Scatter plots for each pair of numeric variables.
  - Distribution plots (histograms or density plots) along the diagonal, representing the distribution of each variable.
- `ftblSal`: The DataFrame containing the football player salary data, on which the method is applied.

Result of execution:

- A **set of graphs** is generated, allowing to visualize the relationships between the different numeric variables of the DataFrame, such as `age`, `total_value`, `avg_year`, `total_guaranteed`, and `fully_guaranteed`.
- Scatter plots allow to observe possible correlations between variables.
- Diagonal distribution plots allow to understand the individual distribution of each variable.
- This method is useful to visually detect relationships or patterns between numeric variables in the dataset.


age
total_value
avg_year
total_guaranteed
fully_guaranteed
1e8
1e7

4. Data Preparation

[12]:
```
X = ftblSal[['age', 'total_value', 'total_guaranteed',␣
 ↪'fully_guaranteed']]
Y = ftblSal['avg_year']
```

Explanation of the line `X`:

- `X` is defined as a **subset of columns** extracted from the DataFrame `ftblSal`.
- The selected columns are:
  - `age`: The age of the players.
  - `total_value`: The total value of the contract.
  - `total_guaranteed`: The amount guaranteed in the contract.
  - `fully_guaranteed`: The fully guaranteed amount.
- These columns are considered as **independent variables** or **features**, which will be used as inputs in a predictive model.
- The result is a `DataFrame` containing only these 4 columns, which constitutes the predictor matrix.

Explanation of the line `Y` :

- `Y` is defined as the column `avg_year` extracted from the DataFrame `ftblSal`.
- This column represents the **average annual salary** of the players and constitutes the **target variable** or **dependent** for the prediction.
- The result is an object of type `Series` containing the values of the variable `avg_year`.

Usefulness of this code:

- **Data preparation:** Selects only the columns needed for training and prediction.
- **Model construction:** `X` serves as input and `Y` as output to train a machine learning model.
- **Simplicity and clarity:** Allows to work on a relevant subset of the data, ignoring columns that are not useful for this specific task.

[14]:
```
X_train, X_test, Y_train, Y_test = train_test_split(X,Y, test_size=0.
 ↪3,random_state=101)
```

Code explanation:

- The `train_test_split` function is imported from the `scikit-learn` library. It splits the data into training and test sets.
- `X` and `Y` :

- `X` : The independent variables (or predictors).
- `Y` : The dependent variable (or target).

- Function parameters:
  - `test_size=0.3` : 30% of the data is allocated to the test set, while the remaining 70% is used for training.
  - `random_state=101` : Ensures that the data split is reproducible. Any subsequent execution with the same value of `random_state` will produce the same training and test sets.

- Function output:
  - `X_train` : The training data for the predictors.
  - `X_test` : The test data for the predictors.
  - `Y_train` : The training data for the target.
  - `Y_test` : The test data for the target.

What this code is for:

- **Model training and evaluation:** Allows you to separate the data to train the model (`train`) and evaluate its performance on never-before-seen data (`test`).
- **Overfitting Prevention:** Ensures that the model is tested on a set that is independent of the data used for training.
- **Balanced Distribution:** Ensures that the data is randomly distributed across the sets, while maintaining reproducibility using `random_state`.

5. Model Building

[16]:
```
regressor = LinearRegression()
```

Explanation:

- This line initializes an object of the `LinearRegression()` class from the `scikit-learn` library.
- The `regressor` object represents an empty linear regression model, ready to be trained on data.
- By default, the `LinearRegression()` class uses the following parameters:
  - `fit_intercept=True`: The model calculates and includes a constant (or intercept) in the regression equation.
  - `normalize=False`: The input data is not normalized before training.

- `copy_X=True`: A copy of the data is used, leaving the original data unchanged.
- `n_jobs=None` : Training uses a single processor.

- The model thus created can be trained on a dataset to learn the relationships between the independent variables (`features`) and the dependent variable (`target`).

Usefulness:

- This step is a preparation for training a linear regression model.
- The model will fit an equation of the form:

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_n X_n,$$

where β_0 is the intercept and $\beta_1, \beta_2, \ldots, \beta_n$ are the coefficients associated with the explanatory variables.

- Once trained, the model will be able to predict values of the target variable (`Y`) for new data.

6. Model Training

[18]:
```
regressor.fit(X_train,Y_train)
```

[18]:
```
LinearRegression()
```

Explanation:

- The `fit()` method is used to train the `regressor` object, which is a linear regression model.
- `X_train` : Features matrix used as input for the model. It contains the independent variables (`age`, `total_value`, `total_guaranteed`, `fully_guaranteed`) in the training data.
- `Y_train` : Vector of targets, representing the dependent variable (`avg_year`) in the training data.
- The goal of training is to find the coefficients ($\beta_1, \beta_2, \ldots$) and the intercept (β_0) that minimize the mean squared error between the values predicted by the model and the actual values of `Y_train`.
- After calling this method, the attributes `coef_` (coefficients of the variables) and `intercept_` (intercept) of the `regressor` object are calculated.

Result:

- Running this line returns the `LinearRegression()` object, showing that training has completed successfully.

- The calculated coefficients can be viewed via `regressor.coef_`, and the intercept via `regressor.intercept_`.

Usefulness:

- Training allows the model to capture linear relationships between the explanatory variables and the target variable.
- The model is now ready to make predictions on new data or to evaluate its performance on a test set (`X_test`).

7. Model Evaluation

[20]:
```
print(regressor.intercept_)
```

```
603136.7559387339
```

Explanation:

- The `intercept_` property of the `regressor` object gives the intercept of the trained linear regression model.
- This value represents the predicted **average annual salary** (`avg_year`) when all independent variables (`age`, `total_value`, `total_guaranteed`, `fully_guaranteed`) are equal to zero.
- In this context, it is a fixed component that is added to the predictions made by the model.
- Using `print()` displays the intercept value.

Result of execution:

- The displayed value is `603136.7559387339`.
- This means that, according to the model, a player with all explanatory variables equal to zero would have an average annual salary of approximately `603136.76` (unit corresponding to that of the data).

Usefulness of `intercept_`:

- Allows to understand the fixed contribution to the predicted average salary, independent of other factors.
- Helps to interpret the model and assess the impact of the explanatory variables.
- Plays a vital role in the linear prediction equation:

$$\text{avg\_year} = (\text{coefficients} \cdot \text{features}) + \text{intercept\_}.$$

[22]:
```
print(regressor.coef_)
```

```
[-1.59256722e+02  1.83617429e-01 -7.58119018e-02  1.73035891e-01]
```

Explanation:

- The `coef_` property of the `regressor` object gives the coefficients associated with the independent variables in the linear regression model.
- These coefficients represent the impact or weight of each explanatory variable on the dependent variable (`avg_year`), while keeping the other variables constant.
- Each coefficient corresponds to an independent variable in the order of their inclusion in `X_train`.

Result of the execution:

- The coefficients obtained are as follows:

$$[-159.256722, 0.183617429, -0.0758119018, 0.173035891].$$

- These values refer to the variables `age`, `total_value`, `total_guaranteed`, and `fully_guaranteed`, respectively.

Interpretation of coefficients:

- `-159.256722 (age)`: A one-year increase in player age is associated with an average decrease of 159.26 units in average annual salary (`avg_year`), all else being equal.
- `0.183617429 (total_value)`: A one-unit increase in total contract value (`total_value`) is associated with an average increase of 0.18 units in average annual salary.
- `-0.0758119018 (total_guaranteed)` : A one-unit increase in the total guaranteed amount (`total_guaranteed`) is associated with a small decrease of 0.08 units in the average annual salary.
- `0.173035891 (fully_guaranteed)` : A one-unit increase in the fully guaranteed amount (`fully_guaranteed`) is associated with an average increase of 0.17 units in the average annual salary.

Usefulness of `coef_` :

- Allows to evaluate the importance and direction (positive or negative) of the impact of each variable on the prediction.
- Provides essential information to interpret the model and identify the most influential variables.
- Plays a key role in the linear regression equation:

$$\text{avg\_year} = (\text{coef\_} \cdot \text{features}) + \text{intercept\_}.$$

[24]:
```
coeff_df = pd.DataFrame(regressor.coef_,X.columns,columns=['Coefficient'])
coeff_df
```

[24]:

	Coefficient
age	-159.256722
total_value	0.183617
total_guaranteed	-0.075812
fully_guaranteed	0.173036

Explanation:

- The `pd.DataFrame()` function of the `pandas` library is used to create a structured table of the coefficients calculated by the model.
- `regressor.coef_`: Table of coefficients associated with the explanatory variables (`age`, `total_value`, `total_guaranteed`, `fully_guaranteed`).
- `X.columns`: List of the names of the columns used as explanatory variables.
- `columns=['Coefficient']`: Gives a title to the column containing the coefficients.
- The resulting table associates each explanatory variable with its calculated coefficient.

Result:

Variable	Coefficient
age	−159.256722
total_value	0.183617
total_guaranteed	−0.075812
fully_guaranteed	0.173036

Interpretation:

- This table allows you to easily visualize the impact of each explanatory variable on the target variable (`avg_year`).
- Positive values of the coefficients indicate a direct relationship with `avg_year` (an increase in the target variable when the explanatory variable increases), while negative values indicate an inverse relationship.
- For example:
  - `age` : Each additional year results in an average decrease of 159.26 units in `avg_year`.
  - `total_value` : An increase of one unit in `total_value` results in an average increase of 0.18 units in `avg_year`.

Usefulness:

- This table is essential for interpreting the regression model and identifying the variables that have the most influence on the predictions.

- It also makes it easier to communicate the results to a non-technical audience.

8. Model Predictions

[26]:
```
Y_predict = regressor.predict(X_test)
```

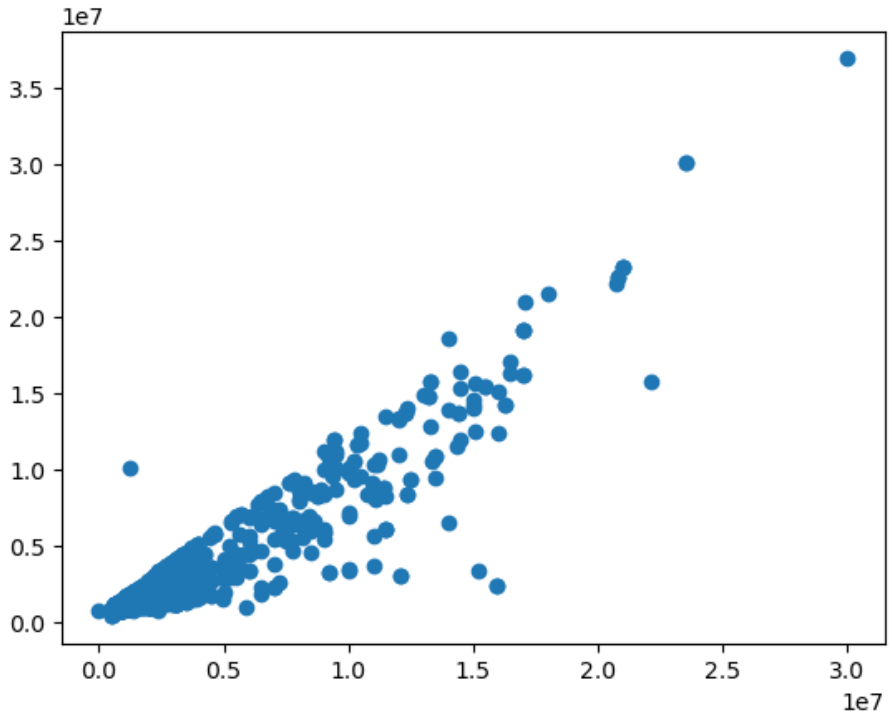
Explanation:

- `regressor.predict()` : This method is used to predict the values of the target variable (Y) based on the test data (`X_test`) from the trained linear regression model.
- `X_test` : Dataset containing the values of the explanatory variables, not used for training, but reserved for model evaluation.
- `Y_predict` : Array containing the predictions generated by the model for the test data.

Usefulness:

- This step is crucial to evaluate the performance of the model by comparing the predictions (`Y_predict`) with the actual values (`Y_test`).
- Metrics such as mean square error (MSE), coefficient of determination (R^2), or mean absolute error (MAE) are calculated from these predictions to measure the accuracy of the model.

[28]:
```
plt.scatter(Y_test,Y_predict)
```



9. Regression Metrics

[30]:
```
print('MAE:', metrics.mean_absolute_error(Y_test, Y_predict))
print('MSE:', metrics.mean_squared_error(Y_test, Y_predict))
print('RMSE:', np.sqrt(metrics.mean_squared_error(Y_test, Y_predict)))
print('R²:', metrics.r2_score(Y_test, Y_predict))
```

Explanation:

- **MAE (Mean Absolute Error)**: The average of the absolute errors between the predicted values ($Y\_predict$) and the actual values (Y_test).
  - This gives a general idea of the mean absolute error of the model, in terms of the target variable unit.
  - The lower the value, the better.
- **MSE (Mean Squared Error)**: The average of the squared errors between the predicted and actual values.
  - This penalizes large errors more heavily than MAE, because they are squared.
  - Like MAE, a lower value is desirable.
- **RMSE (Root Mean Squared Error)** : The square root of the MSE.
  - This metric gives an idea of the error in terms of the target variable's unit.
  - This is a more interpretable version of the MSE.
- **R^2 (Coefficient of Determination)** : A measure of the proportion of the data variance that is explained by the model.
  - An R^2 of 1 indicates that the model explains the data variance perfectly, while an R^2 of 0 indicates that it explains nothing.
  - An R^2 close to 1 is desirable.

```
MAE: 632992.004179555
MSE: 1529064126650.6885
RMSE: 1236553.3254375602
R²: 0.8780029173443924
```

Interpretation:

- **MAE**: The mean absolute error of the model is 632992.00, which means that, on average, the prediction is wrong by 632992 units.

- **MSE**: The mean squared error is 1.53×10^{12}, which is relatively high, but it is a measure sensitive to large errors.
- **RMSE**: The root mean square error is 1236553.33, which shows the magnitude of the model's errors in terms of the target variable unit.
- $\mathbf{R^2}$: The coefficient of determination is 0.878, indicating that the model explains about 87.8% of the variance in the data. This means that the model performs quite well.