**Day 77 Goals: what you will make by the end of the day**

In this lesson, we're going to be looking at movie budget and revenue data. This dataset is perfect for trying out some new tools like scikit-learn to run a linear regression and seaborn, a popular data visualisation library built on top of Matplotlib.

Interfaz de usuario gráfica, Sitio web

Descripción generada automáticamente

The question we want to answer today is: Do higher film budgets lead to more revenue in the box office? In other words, should a movie studio spend more on a film to make more?

**Today you'll learn:**

* How to use a popular data visualisation library called Seaborn
* How to run and interpret a linear regression with scikit-learn
* How to plot a regression a scatter plot to visualise relationships in the data
* How to add a third dimension to a scatter plot to create a bubble chart
* How to cleverly use floor division // to convert your data

Logotipo

Descripción generada automáticamente

**Download and add the Notebook to Google Drive**

As usual, download the .zip file from this lesson and extract it. Add the .ipynb file into your Google Drive and open it as a Google Colaboratory notebook.

**Add the Data to the Notebook**

The .zip file also includes a .csv file called cost\_revenue\_dirty. This is the data for the project. Add this file to your notebook.

**Explore and Clean the Data**

As with any dataset, the first steps are going to be data exploration and data cleaning. We need to get a better understanding of what we're dealing with. Since you've gone through this process a number of times before on previous days, can you tackle the following challenges on your own?

Challenge 1

Answer these questions about how the data is structured:

1. How many rows and columns does the dataset contain?
2. Are there any NaN values present?
3. Are there any duplicate rows?
4. What are the data types of the columns?

Challenge 2

Convert the USD\_Production\_Budget, USD\_Worldwide\_Gross, and USD\_Domestic\_Gross columns to a numeric format by removing $ signs and ,.

Note that *domestic* in this context refers to the United States.

Challenge 3

Convert the Release\_Date column to a Pandas Datetime type.

.

.

..

...

..

.

.

**Solution for Challenge 1**

With any new dataset, it's a good idea to do some standard checks and conversions. I typically always first look at .shape, .head(), .tail(), .info() and .sample().  Here's what I'm spotting already:

Texto, Tabla

Descripción generada automáticamente con confianza media

There are thousands of entries in the DataFrame - one entry for each movie. We'll have some challenges formatting the data before we can do more analysis because we have non-numeric characters in our budget and revenue columns.

We can check for NaN values with this line:

1. data.isna().values.any()

And check for duplicates with this line:

1. data.duplicated().values.any()

We can see the total number of duplicates by creating a subset and looking at the length of that subset:

1. duplicated\_rows = data[data.duplicated()]
2. len(duplicated\_rows)

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

Descripción generada automáticamente

The fact that there are no duplicates or NaN (not-a-number) values in the dataset will make our job easier. We can also see if there are null values in .info(), which also shows us that we need to do some type conversion.

Tabla

Descripción generada automáticamente

**Solution for Challenge 2**

In order to convert the data in the budget and revenue columns and remove all the non-numeric characters, we can use a nested for loop. We create two Python lists: the characters to remove and the column names. Inside the nested loop we can combine .replace() and .to\_numeric() to achieve our goal.

1. chars\_to\_remove = [',', '$']
2. columns\_to\_clean = ['USD\_Production\_Budget',
3. 'USD\_Worldwide\_Gross',
4. 'USD\_Domestic\_Gross']
6. for col in columns\_to\_clean:
7. for char in chars\_to\_remove:
8. # Replace each character with an empty string
9. data[col] = data[col].astype(str).str.replace(char, "")
10. # Convert column to a numeric data type
11. data[col] = pd.to\_numeric(data[col])

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

**Solution for Challenge 3**

To convert the Release\_Date column to a DateTime object, all we need to do is call the to\_datetime() function.

1. data.Release\_Date = pd.to\_datetime(data.Release\_Date)

When we check .info() again we see that the columns now have the desired data type. This allows us to proceed with the next parts of our analysis.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

**Investigate the Films that had Zero Revenue**

Now that we've done some legwork on cleaning our data, we can investigate our data set more thoroughly.

Challenge 1

1. What is the average production budget of the films in the data set?
2. What is the average worldwide gross revenue of films?
3. What were the minimums for worldwide and domestic revenue?
4. Are the bottom 25% of films actually profitable or do they lose money?
5. What are the highest production budget and highest worldwide gross revenue of any film?
6. How much revenue did the lowest and highest budget films make?

Challenge 2

How many films grossed $0 domestically (i.e., in the United States)? What were the highest budget films that grossed nothing?

Challenge 3

How many films grossed $0 worldwide? What are the highest budget films that had no revenue internationally (i.e., the biggest flops)?

.

.

..

...

..

.

.

**Solution for Challenge 1**

We can answer many of the questions with a single command: .describe().

Tabla

Descripción generada automáticamente

The average film costs about $31m to make and makes around 3x that (or ~$89m) in worldwide revenue. So that's encouraging.

But quite a lot of films lose money too. In fact, all the films in the bottom quartile lose money, since the average cost is $5 million and they only bring in $3.8m in worldwide revenue!

The minimum domestic and worldwide revenue is $0. That makes sense. If a film never gets screened or is cancelled, then this is the number we would expect to see here.

On the other hand, the highest production budget was $425,000,000 and the highest worldwide revenue was $2,783,918,982. $2.7 Billion revenue! Holy smokes.

So which film was the lowest budget film in the dataset?

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

I've ... never heard of this film. But it looks like a real money maker. It grossed $181,041 with a measly $1,100 budget. 😮 Wow. Talk about return on investment!

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

And the highest budget film in the dataset is:

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamenteInterfaz de usuario gráfica, Aplicación, Sitio web

Descripción generada automáticamente

Sigh, I remember watching this film in the cinema with 3D glasses 🤓 and not wanting the film to ever end! I would have been quite content living with those blue people. 😊

**Solution to Challenge 2: No domestic revenue**

We see that there are 512 films in the dataset that had no revenue in the United States. However, the highest budget films with no revenue have a release date AFTER the date on which the dataset was compiled (May 1st, 2018).

1. zero\_domestic = data[data.USD\_Domestic\_Gross == 0]
2. print(f'Number of films that grossed $0 domestically {len(zero\_domestic)}')
3. zero\_domestic.sort\_values('USD\_Production\_Budget', ascending=False)

Tabla

Descripción generada automáticamente

**Solution to Challenge 3: No worldwide revenue**

When we check worldwide revenue instead, we see that there are 357 films that made no money internationally. Once again, some of the films have not been released yet at the time the data was compiled. However, 512 versus 357. Why is there a difference?

The reason some international films were never screened in the United States.  In fact, we can see an example of this in our previous screenshot. "Don Gato, el inicio de la pandilla" made about $4.5 million dollars in the box office, but nothing in the United States. Perhaps they should have screened it there too, considering it cost $80 million to make!

1. zero\_worldwide = data[data.USD\_Worldwide\_Gross == 0]
2. print(f'Number of films that grossed $0 worldwide {len(zero\_worldwide)}')
3. zero\_worldwide.sort\_values('USD\_Production\_Budget', ascending=False)

Tabla

Descripción generada automáticamente

**Filter on Multiple Conditions: International Films**

So far, we've created subsets for our DataFrames based on a single condition. But what if we want to select our data based on more than one condition? For example, which films made money internationally (i.e., data.USD\_Worldwide\_Gross != 0), but had zero box office revenue in the United States (i.e., data.USD\_Domestic\_Gross == 0)?

How would we create a filter for these two conditions? One approach is to use the .loc[] property combined with the [bitwise and](https://docs.python.org/3.4/library/operator.html" \l "mapping-operators-to-functions" \t "_blank) & operator.

1. international\_releases = data.loc[(data.USD\_Domestic\_Gross == 0) &
2. (data.USD\_Worldwide\_Gross != 0)]

Why does this work? Pandas is built on top of NumPy, which uses Python's bitwise operators. And these bitwise operators allow us to do comparisons on an element by element basis in both NumPy and Pandas! Here's an example:

Interfaz de usuario gráfica, Texto, Aplicación, Tabla

Descripción generada automáticamente con confianza media

However, we're also checking if the domestic revenue was zero and the worldwide revenue was not zero. Because the bitwise operator takes precedence, we need to include parentheses () around the comparisons we'd like to prioritise.

Tabla

Descripción generada automáticamente con confianza media

However, this is not the only technique we can use to make multiple comparisons.

Challenge

Use the Pandas [.query() function](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.query.html) to accomplish the same thing. Create a subset for international releases that had some worldwide gross revenue, but made zero revenue in the United States.

Hint: This time you'll have to use the and keyword.

.

.

..

...

..

.

.

**Solution: Using the .query() function to filter on multiple conditions**

In this case, we enclose the entire query inside a string.

1. international\_releases = data.query('USD\_Domestic\_Gross == 0 and USD\_Worldwide\_Gross != 0')
2. print(f'Number of international releases: {len(international\_releases)}')
3. international\_releases.tail()

The column names are recognised and we see the following:

Imagen que contiene Texto

Descripción generada automáticamente

Unreleased Films

Now we can turn our attention to films in the dataset that were not released at the time the data was collected. This is why films like Singularity and Aquaman had zero revenue.

Challenge

* Identify which films were not released yet as of the time of data collection (May 1st, 2018).
* How many films are included in the dataset that have not yet had a chance to be screened in the box office?
* Create another DataFrame called data\_clean that does not include these films.

.

.

..

...

..

.

.

**Solution: Removing the unreleased films**

There are a total of 7 unreleased films at the time of data collection included in the dataset.

Tabla

Descripción generada automáticamente

From this point on, we'll work with another DataFrame called data\_clean that does not include these films.

1. data\_clean = data.drop(future\_releases.index)

Bonus Challenge: Films that Lost Money

Having removed the unreleased films entirely can you calculate the percentage of films that did not break even at the box office? We already saw that more than the bottom quartile of movies appears to lose money when we ran .describe(). However, what is the true percentage of films where the costs exceed the worldwide gross revenue?

.

.

..

...

..

.

.

**Solution: Budget greater than revenue**

Again, there are different ways you could have calculated this. For example, using the .loc[] property,

1. money\_losing = data\_clean.loc[data\_clean.USD\_Production\_Budget > data\_clean.USD\_Worldwide\_Gross]
2. len(money\_losing)/len(data\_clean)

or the .query() function

1. money\_losing = data\_clean.query('USD\_Production\_Budget > USD\_Worldwide\_Gross')
2. money\_losing.shape[0]/data\_clean.shape[0]

In both cases, we see that a whopping 37.2% 😮 of films do not recoup their production budget at the box office. 💸💸💸 Who knew that film finance could be so risky! 😬

**Seaborn Data Visualisation: Bubble Charts**

We're now ready to visualise our data. Today I want to introduce you to another popular data visualisation tool that you can use alongside plotly and Matplotlib: Seaborn. [Seaborn](https://seaborn.pydata.org/" \t "_blank) is built on top of Matplotlib and it makes creating certain visualisations very convenient.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

**Import Seaborn**

The first step is adding Seaborn to our notebook. By convention we'll use the name sns.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

**Seaborn Scatter Plots**

To create a [.scatterplot()](https://seaborn.pydata.org/generated/seaborn.scatterplot.html?highlight=scatterplot#seaborn.scatterplot), all we need to do is supply our DataFrame and the column names that we'd like to see on our axes.

1. sns.scatterplot(data=data\_clean,
2. x='USD\_Production\_Budget',
3. y='USD\_Worldwide\_Gross')

Gráfico, Gráfico de dispersión

Descripción generada automáticamente

That should look familiar. 😊 Because Seaborn is built on top of Matplotlib, we can dive into the Matplotlib layer anytime to configure our chart. For example, we can increase the size of our figure:

Gráfico, Gráfico de dispersión

Descripción generada automáticamente

And to style our chart we can simply configure the Axes object that is returned from sns.scatterplot().

Logotipo

Descripción generada automáticamente

Here's how:

1. plt.figure(figsize=(8,4), dpi=200)
3. ax = sns.scatterplot(data=data\_clean,
4. x='USD\_Production\_Budget',
5. y='USD\_Worldwide\_Gross')
7. ax.set(ylim=(0, 3000000000),
8. xlim=(0, 450000000),
9. ylabel='Revenue in $ billions',
10. xlabel='Budget in $100 millions')
12. plt.show()

Here we're diving into the Matplotb layer to set the limits on the axes and change the labels.

Gráfico, Gráfico de dispersión

Descripción generada automáticamente

**From Scatter Plot to Bubble Chart**

But the reason we're using Seaborn is because of the hue and size parameters that make it very easy to create a bubble chart. These parameters allow us to colour the data and change their size according to one of the columns in our DataFrame.

1. plt.figure(figsize=(8,4), dpi=200)
2. ax = sns.scatterplot(data=data\_clean,
3. x='USD\_Production\_Budget',
4. y='USD\_Worldwide\_Gross',
5. hue='USD\_Worldwide\_Gross', # colour
6. size='USD\_Worldwide\_Gross',) # dot size
8. ax.set(ylim=(0, 3000000000),
9. xlim=(0, 450000000),
10. ylabel='Revenue in $ billions',
11. xlabel='Budget in $100 millions',)
13. plt.show()

Gráfico, Gráfico de dispersión

Descripción generada automáticamente

Now our higher grossing movies are bigger and darker on our chart. That's super handy. But Seaborn offers a number of convenient styling options as well.

To set the styling on a single chart (as opposed to all the charts in the entire notebook) we can use Python's with keyword. We've seen with used already when it comes to opening files in previous lessons.

1. plt.figure(figsize=(8,4), dpi=200)
3. # set styling on a single chart
4. with sns.axes\_style('darkgrid'):
5. ax = sns.scatterplot(data=data\_clean,
6. x='USD\_Production\_Budget',
7. y='USD\_Worldwide\_Gross',
8. hue='USD\_Worldwide\_Gross',
9. size='USD\_Worldwide\_Gross')
11. ax.set(ylim=(0, 3000000000),
12. xlim=(0, 450000000),
13. ylabel='Revenue in $ billions',
14. xlabel='Budget in $100 millions')

Gráfico, Gráfico de dispersión

Descripción generada automáticamente

In addition to 'darkgrid', Seaborn has a number of [built-in themes](https://python-graph-gallery.com/104-seaborn-themes/" \t "_blank). so you can style your chart very quickly. Try out 'whitegrid', 'dark',  or 'ticks' for example.

Challenge

Now that you've seen how to create a beautiful bubble chart in Seaborn, it's time to create your own. Can you write the code to replicate this chart? Notice how we are actually representing THREE dimensions in this chart: the budget, the release date, and the worldwide revenue. This is what makes bubble charts so awesomely informative.

Gráfico, Gráfico de dispersión

Descripción generada automáticamente

.

.

..

...

..

.

.

**Solution: Movie Budgets over Time**

Alright, I hope that was fairly straightforward. All you needed to do is change a few arguments:

1. plt.figure(figsize=(8,4), dpi=200)
3. with sns.axes\_style("darkgrid"):
4. ax = sns.scatterplot(data=data\_clean,
5. x='Release\_Date',
6. y='USD\_Production\_Budget',
7. hue='USD\_Worldwide\_Gross',
8. size='USD\_Worldwide\_Gross',)
10. ax.set(ylim=(0, 450000000),
11. xlim=(data\_clean.Release\_Date.min(), data\_clean.Release\_Date.max()),
12. xlabel='Year',
13. ylabel='Budget in $100 millions')

**Analysis**

What do we see here? What is this chart telling us? Well, first off, movie budgets have just exploded in the last 40 years or so. Up until the 1970s, the film industry appears to have been in an entirely different era. Budgets started growing fast from the 1980s onwards and continued to grow through the 2000s. Also, the industry has grown massively, producing many more films than before. The number of data points is so dense from 2000 onwards that they are overlapping.

Gráfico

Descripción generada automáticamente con confianza media

**Floor Division: A Trick to Convert Years to Decades**

In our bubble charts, we've seen how massively the industry has changed over time, especially from the 1970s onwards. This makes me think it makes sense to separate our films out by decade. Here's what I'm after:

Tabla

Descripción generada automáticamente

Challenge

Can you create a column in data\_clean that has the decade of the movie release. For example, a film released in 1992 or 1999 should have 1990 in the Decade column.

Here is one approach that you can follow:

1. Create a [DatetimeIndex object](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DatetimeIndex.html" \t "_blank) from the Release\_Date column.

Texto

Descripción generada automáticamente con confianza media

1. Grab all the years from the DatetimeIndex object using the .year property.
2. Use floor division // to convert the year data to the decades of the films.
3. Add the decades as a Decade column to the data\_clean DataFrame.

.

.

..

...

..

.

.

**Solution: Using Floor Division to Convert Years to Decades**

To create a DatetimeIndex, we just call the constructor and provide our release date column as an argument to initialise the DatetimeIndex object. Then we can extract all the years from the DatetimeIndex.

1. dt\_index = pd.DatetimeIndex(data\_clean.Release\_Date)
2. years = dt\_index.year

Now, all we need to do is convert the years to decades. For that, we will use floor division (aka integer division). The difference to regular division is that the result is effectively rounded down.

1. 5.0 / 2
2. # output: 2.5
3. 5.0 // 2
4. # output: 2.0

In our case, we will use the floor division by 10 and then multiplication by 10 to convert the release year to the release decade:

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

Descripción generada automáticamente

We can do this for all the years and then add the decades back as a column.

1. decades = years//10\*10
2. data\_clean['Decade'] = decades

Challenge

Create two new DataFrames: old\_films and new\_films

* old\_films should include all the films before 1970 (up to and including 1969)
* new\_films should include all the films from 1970 onwards
* How many of our films were released prior to 1970?
* What was the most expensive film made prior to 1970?

.

.

..

...

..

.

.

**Solution: Separate the films made before and after 1970**

Now that we have our Decades column we can use it to create subsets of our data.

1. old\_films = data\_clean[data\_clean.Decade <= 1960]
2. new\_films = data\_clean[data\_clean.Decade > 1960]

The cut-off for our calculation is 1960 in the Decade column because this will still include 1969. When we inspect our old\_films DataFrame we see that it only includes 153 films. As we saw in the bubble chart, the bulk of films in the dataset have been released in the last 30 years.

Tabla

Descripción generada automáticamente

The most expensive film prior to 1970 was Cleopatra, with a production budget of $42 million. That's some serious 1960s money, and judging by the trailer, a lot of it went into extravagant costumes, set design, and plenty of extras. Impressive.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

**Plotting Linear Regressions with Seaborn**

Let's visualise the relationship between the movie budget and the worldwide revenue using linear regression. Seaborn makes this incredibly easy with the [.regplot()](https://seaborn.pydata.org/generated/seaborn.regplot.html#seaborn.regplot) function.

1. sns.regplot(data=old\_films,
2. x='USD\_Production\_Budget',
3. y='USD\_Worldwide\_Gross')

This creates a scatter plot and draws a linear regression line together with the confidence interval at the same time.

To style the chart further, we can once again, drop into the Matplotlib layer and supply keyword arguments as dictionaries. We can customise the scatter plot (e.g., by changing the transparency of the dots) and the regression line itself (e.g., by changing the colour).

1. plt.figure(figsize=(8,4), dpi=200)
2. with sns.axes\_style("whitegrid"):
3. sns.regplot(data=old\_films,
4. x='USD\_Production\_Budget',
5. y='USD\_Worldwide\_Gross',
6. scatter\_kws = {'alpha': 0.4},
7. line\_kws = {'color': 'black'})

Gráfico, Gráfico de dispersión

Descripción generada automáticamente

What do we see here? Well, first off we can spot Cleopatra on the far right. But also, we see that many lower budget films made much more money! The relationship between the production budget and movie revenue is not very strong. Many points on the left are very far away for the line, so the line appears not to capture the relationship between budget and revenue very well at all!

But does the same hold true for the newer films?

Challenge

Use Seaborn's .regplot() to show the scatter plot and linear regression line against the new\_films.  
  
Style the chart

* Put the chart on a 'darkgrid'.
* Set limits on the axes so that they don't show negative values.
* Label the axes on the plot "Revenue in $ billions" and "Budget in $ millions".
* Provide HEX colour codes for the plot and the regression line. Make the dots dark blue (#2f4b7c) and the line orange (#ff7c43).

Interpret the chart

* Do our data points for the new films align better or worse with the linear regression than for our older films?
* Roughly how much would a film with a budget of $150 million make according to the regression line?

.

.

..

...

..

.

.

**Solution: Plotting a regression against the newer films**

To style the chart we can use the same techniques as before: providing values for the .regplot() function, as well as making use of the Matplotlib Axes object to fine-tune the limits, labels, and general style.

1. plt.figure(figsize=(8,4), dpi=200)
2. with sns.axes\_style('darkgrid'):
3. ax = sns.regplot(data=new\_films,
4. x='USD\_Production\_Budget',
5. y='USD\_Worldwide\_Gross',
6. color='#2f4b7c',
7. scatter\_kws = {'alpha': 0.3},
8. line\_kws = {'color': '#ff7c43'})
10. ax.set(ylim=(0, 3000000000),
11. xlim=(0, 450000000),
12. ylabel='Revenue in $ billions',
13. xlabel='Budget in $100 millions')

Gráfico, Gráfico de dispersión

Descripción generada automáticamente

How do we interpret our chart? This time we are getting a much better fit, compared to the old films. We can see this visually from the fact that our data points line up much better with our regression line (pun intended). Also, the confidence interval is much narrower. We also see that a film with a $150 million budget is predicted to make slightly under $500 million by our regression line.

Gráfico, Gráfico de dispersión

Descripción generada automáticamente

All in all, we can be pretty confident that there does indeed seem to be a relationship between a film's budget and that film's worldwide revenue.

But how much of the variation in revenue does the budget actually explain? And how much extra revenue can we expect for an additional $1 increase in the budget? To find out, we need to dive into the numbers underlying our regression model.

**Use scikit-learn to Run Your Own Regression**

Let's dive into our linear regression model a bit more. We are using a **univariate** regression. This is a regression with a single **explanatory variable** (our movie BUDGET). Explanatory variables are also referred to as **features** in machine learning terminology.

Using our data on budgets, the linear regression estimates the best possible line to fit our movie revenues. The regression line has the following structure:

Diagrama

Descripción generada automáticamente

To find the best possible line, our regression will estimate the y-intercept ("theta zero") and the slope ("theta one"). The line's **intercept** on the y-axis tells us how much revenue a movie would make if the budget was 0. The **slope** tells us how much extra revenue we get for a $1 increase in the movie budget.

Gráfico, Gráfico de dispersión

Descripción generada automáticamente

So how can we find out what our model's estimates are for theta-one and theta-zero? And how can we run our own regression, regardless of whether we want to visualise it on a chart? For that, we can use [scikit-learn](https://scikit-learn.org/stable/" \t "_blank).

Imagen que contiene Interfaz de usuario gráfica

Descripción generada automáticamente

**Import scikit-learn**

Let's add the LinearRegression from scikit-learn to our notebook.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Now we can run a [LinearRegression](https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html" \t "_blank). First, let's create a LinearRegression object that will do the work for us.

1. regression = LinearRegression()

Now we should specify our features and our targets (i.e., our response variable). You will often see the features named capital X and the target named lower case y:

1. # Explanatory Variable(s) or Feature(s)
2. X = pd.DataFrame(new\_films, columns=['USD\_Production\_Budget'])
4. # Response Variable or Target
5. y = pd.DataFrame(new\_films, columns=['USD\_Worldwide\_Gross'])

Our LinearRegression does not like receiving Pandas Series (e.g., new\_films.USD\_Production\_Budget), so I've created some new DataFrames here.

Now it's time to get to work and run the calculations:

1. # Find the best-fit line
2. regression.fit(X, y)

That's it. Now we can look at the values of theta-one and theta-zero from the equation above.

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

Descripción generada automáticamente

Both intercept\_ and coef\_ are simply [attributes of the LinearRegression object](https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html" \t "_blank). Don't worry about the underscores at the end, these are simply part of the attribute names that the scikit-learn developers have chosen.

How do we interpret the y-intercept? Literally, means that if a movie budget is $0, the estimated movie revenue is -$8.65 million. Hmm... so this is clearly unrealistic. Why would our model tell us such nonsense? Well, the reason is that we are specifying what the model should be ahead of time - namely a straight line - and then finding the best straight line for our data. Considering that you can't have negative revenue or a negative budget, we have to be careful about interpreting our very simple model too literally. After all, it's just an estimate and this estimate will be the most accurate on the chart where we have the most data points (rather than at the extreme left or right).

What about the slope? The slope tells us that for every extra $1 in the budget, movie revenue increases by $3.1. So, that's pretty interesting. That means the higher our budget, the higher our estimated revenue. If budgets are all that matter to make lots of money, then studio executives and film financiers should try and produce the biggest films possible, right? Maybe that's exactly why we've seen a massive increase in budgets over the past 30 years.

Tabla

Descripción generada automáticamente con confianza media

**R-Squared: Goodness of Fit**

One measure of figuring out how well our model fits our data is by looking at a metric called r-squared. This is a good number to look at in addition to eyeballing our charts.

1. # R-squared
2. regression.score(X, y)

We see that our r-squared comes in at around 0.558. This means that our model explains about 56% of the variance in movie revenue. That's actually pretty amazing, considering we've got the simplest possible model, with only one explanatory variable. The real world is *super complex*, so in many academic circles, if a researcher can build a simple model that explains over 50% or so of what is actually happening, then it's a pretty decent model.

Remember how we were quite sceptical about our regression looking at the chart for our old\_films?

Challenge

Run a linear regression for the old\_films. Calculate the intercept, slope and r-squared. How much of the variance in movie revenue does the linear model explain in this case?

.

.

..

...

..

.

.

**Solution: A bad fit**

Running the numbers this time around, we can confirm just how inappropriate the linear model is for the pre-1970 films. We still see a positive relationship between budgets and revenue, since the slope (our theta-one) is 1.6, but the r-squared is very low.

Texto

Descripción generada automáticamente

This makes sense considering how poorly our data points aligned with our line earlier.

Imagen que contiene Gráfico

Descripción generada automáticamente

Challenge

You've just estimated the intercept and slope for the Linear Regression model. Now we can use it to make a prediction! For example, how much global revenue does our model estimate for a film with a budget of $350 million?

.

.

..

...

..

.

.

**Solution: Using the model to make a prediction**

For a $350 million budget film, our model predicts a worldwide revenue of around $600 million! You can calculate this as follows:

1. 22821538 + 1.64771314 \* 350000000

Or, using the regression object, you could also work it out like this:

1. budget = 350000000
2. revenue\_estimate = regression.intercept\_[0] + regression.coef\_[0,0]\*budget
3. revenue\_estimate = round(revenue\_estimate, -6)
4. print(f'The estimated revenue for a $350 film is around ${revenue\_estimate:.10}.')

(The colon : and dot . in a print statement is quite handy for controlling the number of digits you'd like to show up in the output)

**Learning Points & Summary**

Today was a pretty packed lesson where we introduced a lot of new concepts. In this lesson we looked at how to:

* Use nested loops to remove unwanted characters from multiple columns
* Filter Pandas DataFrames based on multiple conditions using both .loc[] and .query()
* Create bubble charts using the Seaborn Library
* Style Seaborn charts using the pre-built styles and by modifying Matplotlib parameters
* Use floor division (i.e., integer division) to convert years to decades
* Use Seaborn to superimpose a linear regressions over our data
* Make a judgement if our regression is good or bad based on how well the model fits our data and the r-squared metric
* Run regressions with scikit-learn and calculate the coefficients.

You can download the completed code for today in this lesson.

Well done on completing the next step in your data science journey 👏👏👏 Upwards and onwards!