notebook

March 9, 2023

1 What did Adam do to AMC?

1.0.1 Let's use data science to find out.

- First we will do a little exploratory analysis and visualizations about earnings.
- We will construct a predictive model to predict the 2023 EOY earnings.
- In addition, two other predictive models will be constructed for comparative analysis.
- We will evaluate, and visualize the AMC float dilution history.
- Finally we will determine whether dilution has ever caused AMC share prices to increase.

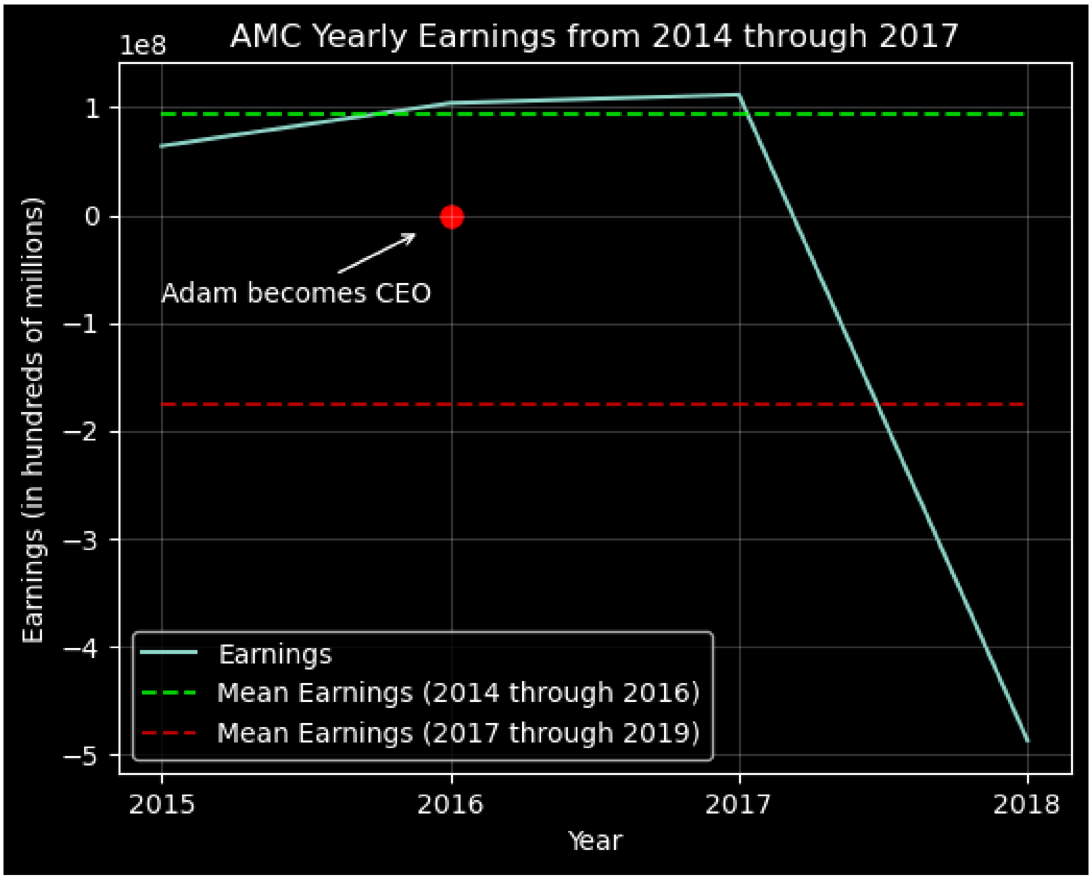
[ ]:
```
# Import statements.

import matplotlib.pyplot as plt
from datetime import datetime
import pandas as pd
import numpy as np
import math

import sys
if sys.path[0] != "scripts": sys.path.insert(0, "scripts")
from scripts.comparative import comparative
from scripts.residuals import presiduals
from scripts.datasets import pdatasets
from scripts.prediction import predict
from scripts.backtest import backtest
from scripts.dilution import dilute
from scripts.dresiduals import dred
from scripts.earnings import eplot
from scripts.pfloat import pfloat
from scripts.epies import epies
from scripts.dhist import hplot
from scripts.preg import rplot
from scripts.dbo import dplot

plt.style.use("dark_background")
```

[ ]:
```
# Load the data.
df = pd.read_csv("data/data.csv", index_col = 0)
```

2 Consider AMC Earnings.

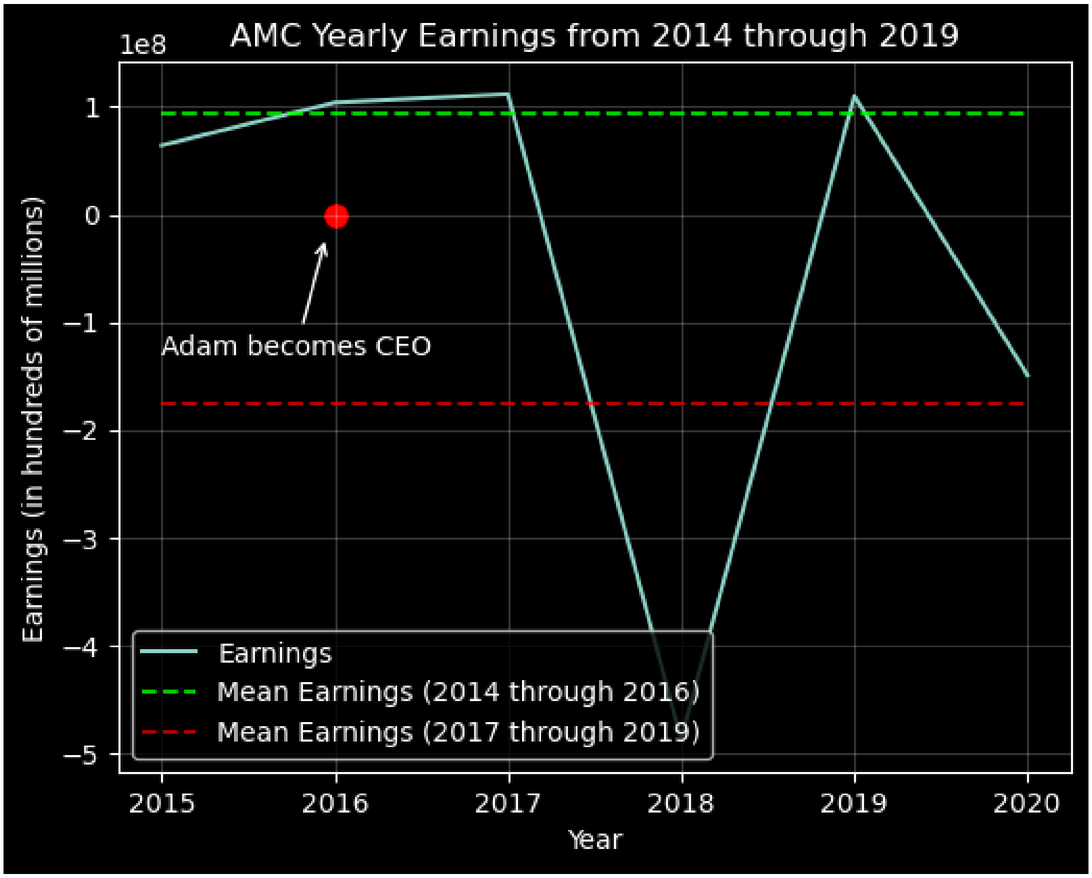
- Adam became CEO in 2016
- Adam purchased Odeon in Q3 2016, and Nordic in Q1 2017

[ ]:
```
# See: analysis/scripts/earnings.py
eplot(4, 2014, 2017, 2014, -0.8, 2014.9, -0.15, 8, plt, df)
```



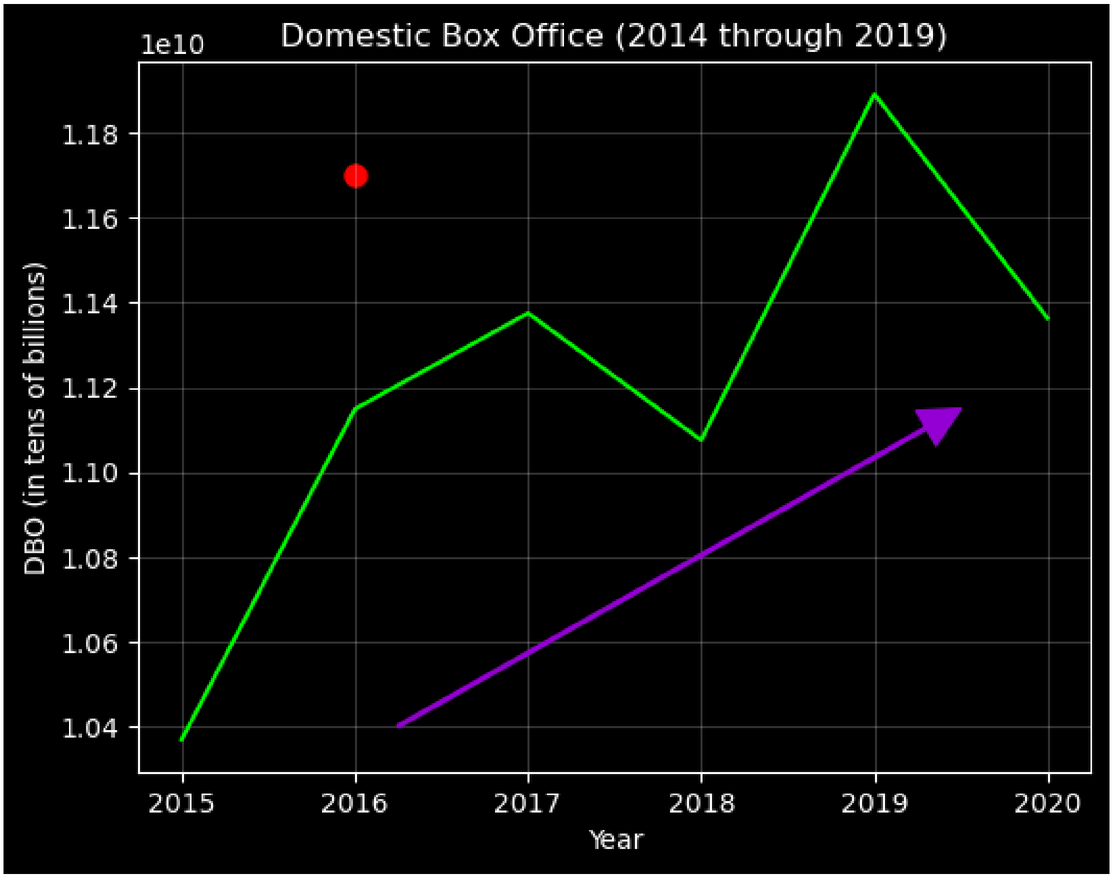
Let's zoom out for some context.

[ ]:
```
# See: analysis/scripts/earnings.py
eplot(6, 2014, 2019, 2014, -1.3, 2014.95, -0.2, 8, plt, df)
```



Compare that to the domestic box office for the same time period.

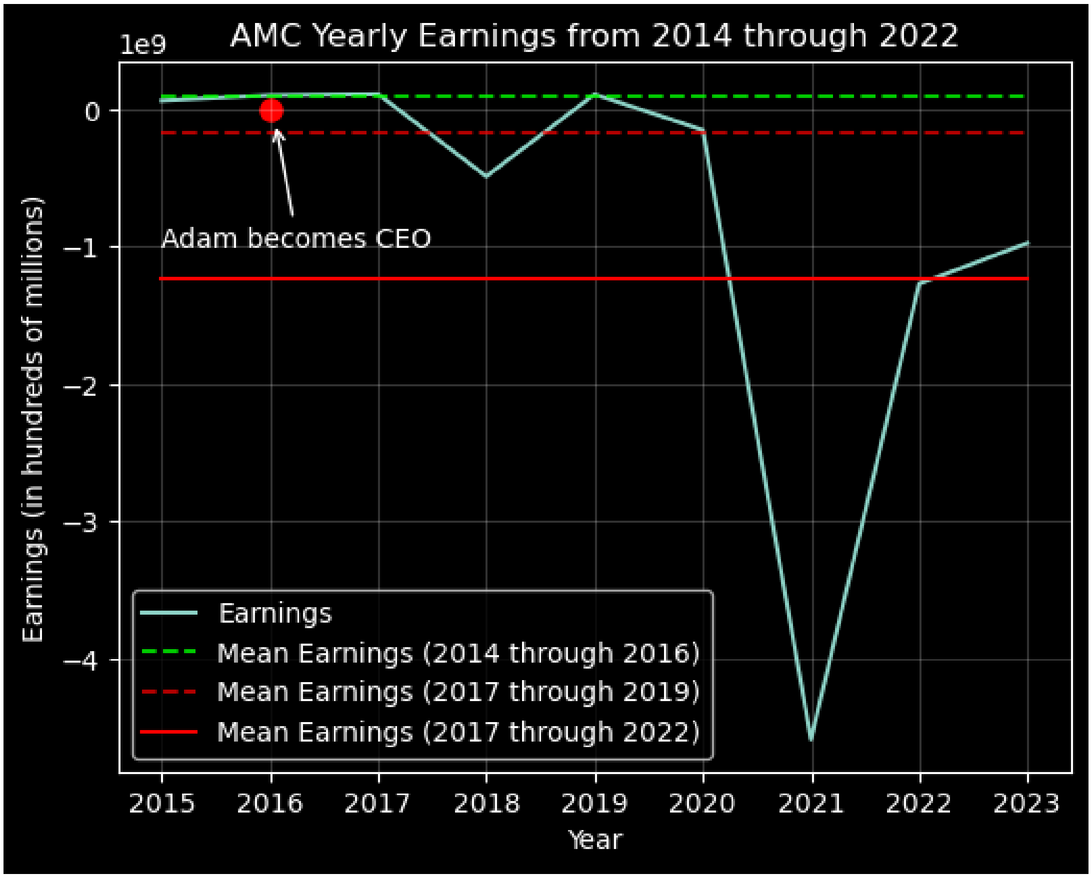
[ ]:
```
# See: analysis/scripts/dbo.py
dplot(plt, df)
```



After Adam becoming CEO, and purchasings Odeon & Nordic it took domestic box office reaching all time highs in 2019 to push AMC past a net loss back up to just slightly above it's mean net earnings from 2014 through 2017. That can't be good.

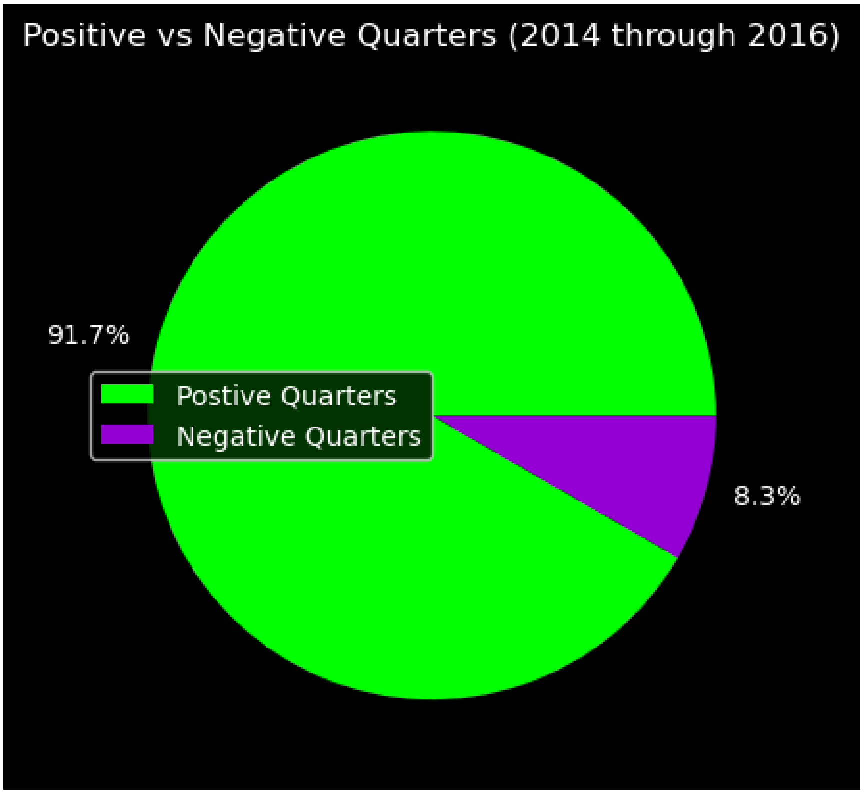
Let's look at earnings for the entire dataset, that is, from 2014 to 2023.

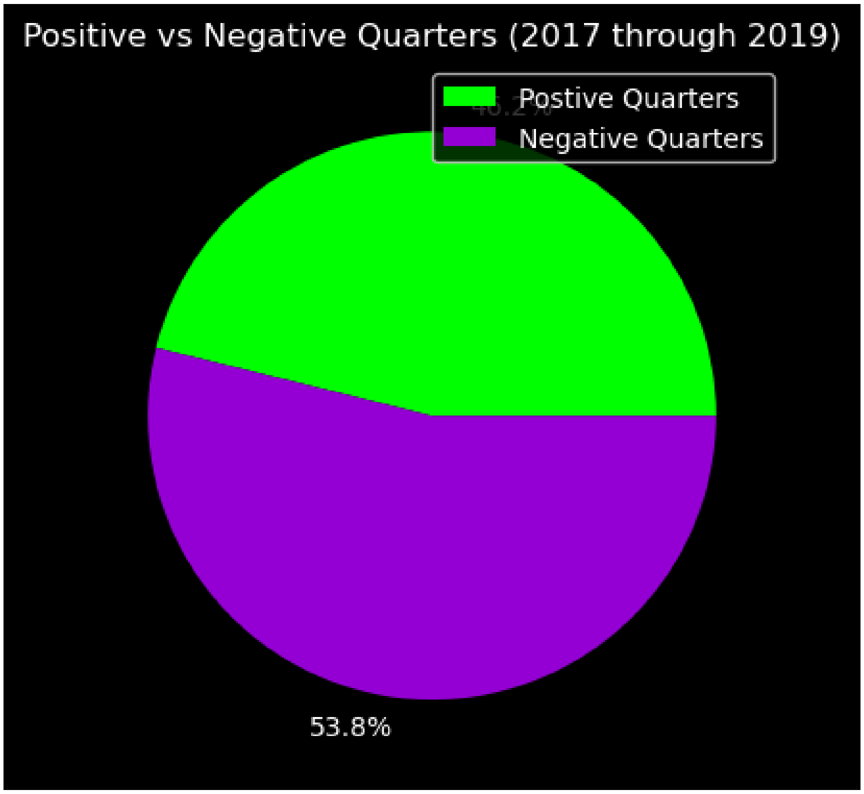
[ ]:
```
# See: analysis/scripts/earnings.py
eplot(9, 2014, 2022, 2014, -1, 2015.05, -0.09, 9, plt, df)
```

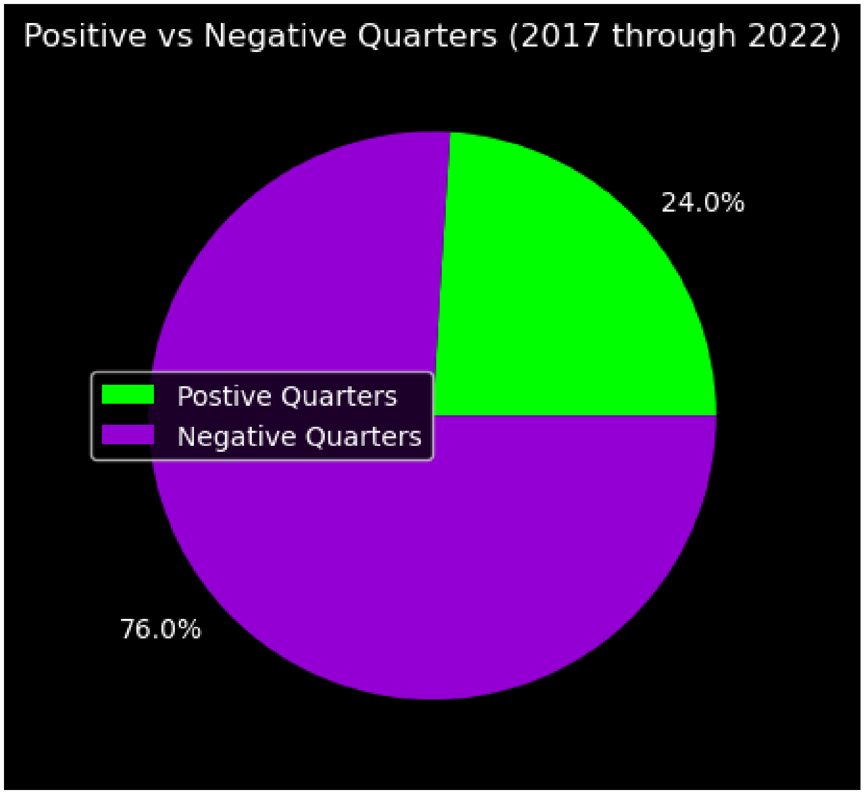


2.0.1 Ouch! Let's really see how much quarterly earnings have dropped off during Adams tenure.

[ ]:
```
# See: analysis/scripts/epies.py
epies(plt, df)
```


Positive vs Negative Quarters (2014 through 2016)
91.7%
8.3%
Postive Quarters
Negative Quarters


Positive vs Negative Quarters (2017 through 2019)
Postive Quarters
Negative Quarters
53.8%



2.0.2 Wow!

2.0.3 Summary about these exploratory analytics

AMC started trading publicly in 2014. Over that 2-year period, from 2014 through 2015, AMC made a total of $167,936,000 in profit. Adam became CEO in 2016. Between 2016 through 2019 AMC lost $-414,533,000. Despite the domestic box office increasing well above its 2014 through 2016 levels. In fact, the mean yearly DBO *(from 2016 through 2019)* increased by $462,244,118 compared to mean yearly DBO from 2014 through 2016. Later we see earnings is a function of DBO so earnings should have increased in this time period under Adam, but they decreased. That decrease was from mean yearly earnings of $93,201,000 *(from 2014 through 2016,)* to $-103,633,250 *(from 2016 through 2019.)* From 2020 Q2 through 2022, Adam managed to multiply his pre-pandemic losses by over an entire order of magintude, to $-4,656,500,000. Excluding 2020 Q1, during Adams full tenure he lost a total of $-5,071,033,000. Including 2020 Q1, altogether, Mr. Aron managed to lose the total astronomical amount of $-7,247,333,000. That is over a billion dollars per year loss, on average.

- Mean quarterly earnings *(2014 through 2016)*: 20,992,000
- Mean quarterly earnings *(2016 through 2019)*: -25,908,312
- Mean quarterly earnings *(2016 through 2022)*: -258,833,321

- Mean quarterly earnings *(2014 through 2022)*: -196,649,917

[ ]:
```
# Calculate P/L for key features, and summarize them.

pl1 = df.loc[:7].earnings.sum()
pl2 = df.loc[8:23].earnings.sum()
pl3 = df.loc[25:].earnings.sum()
pl4 = df.loc[8:].drop(24).earnings.sum()
pl5 = df.loc[8:].earnings.sum()

means = df.groupby("year").agg("sum").drop(columns=["Q"])
m1 = means.loc[2016:2019]["DBO gross"].mean()
m2 = means.loc[2014:2016]["DBO gross"].mean()
e1 = means.loc[2016:2019]["earnings"].mean()
e2 = means.loc[2014:2016]["earnings"].mean()
difference = m1 - m2
del means

print(f"""
AMC started trading publicly in 2014. Over that 2-year period, from 2014␣
↪through 2015, AMC made a total of
${pl1:,.0f} in profit. Adam became CEO in 2016. Between 2016 through 2019 AMC␣
↪lost ${pl2:,.0f}. Despite the
domestic box office increasing well above its 2014 through 2016 levels. In␣
↪fact, the mean yearly DBO (from
2016 through 2019) increased by ${difference:,.0f} compared to mean yearly DBO␣
↪(from 2014 through 2016.) Later
we see earnings is a function of DBO, so earnings should have increased in this␣
↪time period under Adam, but
they decreased from mean yearly earnings of ${e2:,.0f} (from 2014 through␣
↪2016,) to ${e1:,.0f} (from 2016
through 2019.) From 2020 Q2 through 2022, Adam managed to multiply his␣
↪pre-pandemic losses by over an entire
order of magintude, to ${pl3:,.0f}. Excluding 2020 Q1, during Adams full tenure␣
↪he lost a total of ${pl4:,.0f}.
Including 2020 Q1, altogether, Mr. Aron managed to lose the total astronomical␣
↪amount
of ${pl5:,.0f}. That is over a billion dollars per year loss, on average.""")

print(f"""\n\n
    Mean quarterly earnings (2014 through 2016): {df.loc[:7].earnings.mean():,.
↪0f}

    Mean quarterly earnings (2016 through 2019): {df.loc[8:23].earnings.mean():
↪,.0f}
```

```
    Mean quarterly earnings (2016 through 2022): {df.loc[8:].earnings.mean():,.
 ↪0f}

    Mean quarterly earnings (2014 through 2022): {df.earnings.mean():,.0f}
""")
```

```
AMC started trading publicly in 2014. Over that 2-year period, from 2014 through
2015, AMC made a total of
$167,936,000 in profit. Adam became CEO in 2016. Between 2016 through 2019 AMC
lost $-414,533,000. Despite the
domestic box office increasing well above its 2014 through 2016 levels. In fact,
the mean yearly DBO (from
2016 through 2019) increased by $462,244,118 compared to mean yearly DBO (from
2014 through 2016.) Later
we see earnings is a function of DBO, so earnings should have increased in this
time period under Adam, but
they decreased from mean yearly earnings of $93,201,000 (from 2014 through
2016,) to $-103,633,250 (from 2016
through 2019.) From 2020 Q2 through 2022, Adam managed to multiply his pre-
pandemic losses by over an entire
order of magintude, to $-4,656,500,000. Excluding 2020 Q1, during Adams full
tenure he lost a total of $-5,071,033,000.
Including 2020 Q1, altogether, Mr. Aron managed to lose the total astronomical
amount
of $-7,247,333,000. That is over a billion dollars per year loss, on average.



    Mean quarterly earnings (2014 through 2016): 20,992,000

    Mean quarterly earnings (2016 through 2019): -25,908,312

    Mean quarterly earnings (2016 through 2022): -258,833,321

    Mean quarterly earnings (2014 through 2022): -196,649,917
```

[ ]:
```
# Drop 2020 Q1 outlier, and superfluous columns. Extrapolate quarterly to␣
 ↪yearly values.
columns = ["DBO gross", "earnings"]
try:
    df = df.drop(24)
    df["DBO gross"] = df["DBO gross"] * 4
    df["earnings"]  = df["earnings"]  * 4
except KeyError: pass
df = df[columns]
```

```
x  = df["DBO gross"]
y  = df["earnings"]

# We need this for later when talking about dilution
num_quarters = len(df.iloc[8:]) + 1
```

3 Linear Regressions Predictive Models

We are going to predict 2023 earnings based on linear regression analysis. Moreover, we will compare how well AMC might have otherwise performed, in terms of profits (or losses) by constructing:

1. A model based on prior to the Odeon & Nordic purchases. In other words, before Adam disrupted the past profitability of AMC by capital misallocation.
2. A model based on after the Odeon & Nordic purchases, but before Covid. So we can fairly compare Adams performance against the prior performance of AMC, in terms of profitability, by excluding the pandemic.
3. The third model will be used to actually predict future earnings, and also can be compared to either of the other two models for the purpose of contrasting performance based on the different business structurings.

Descriptive summary about the dataset used for regression analysis.

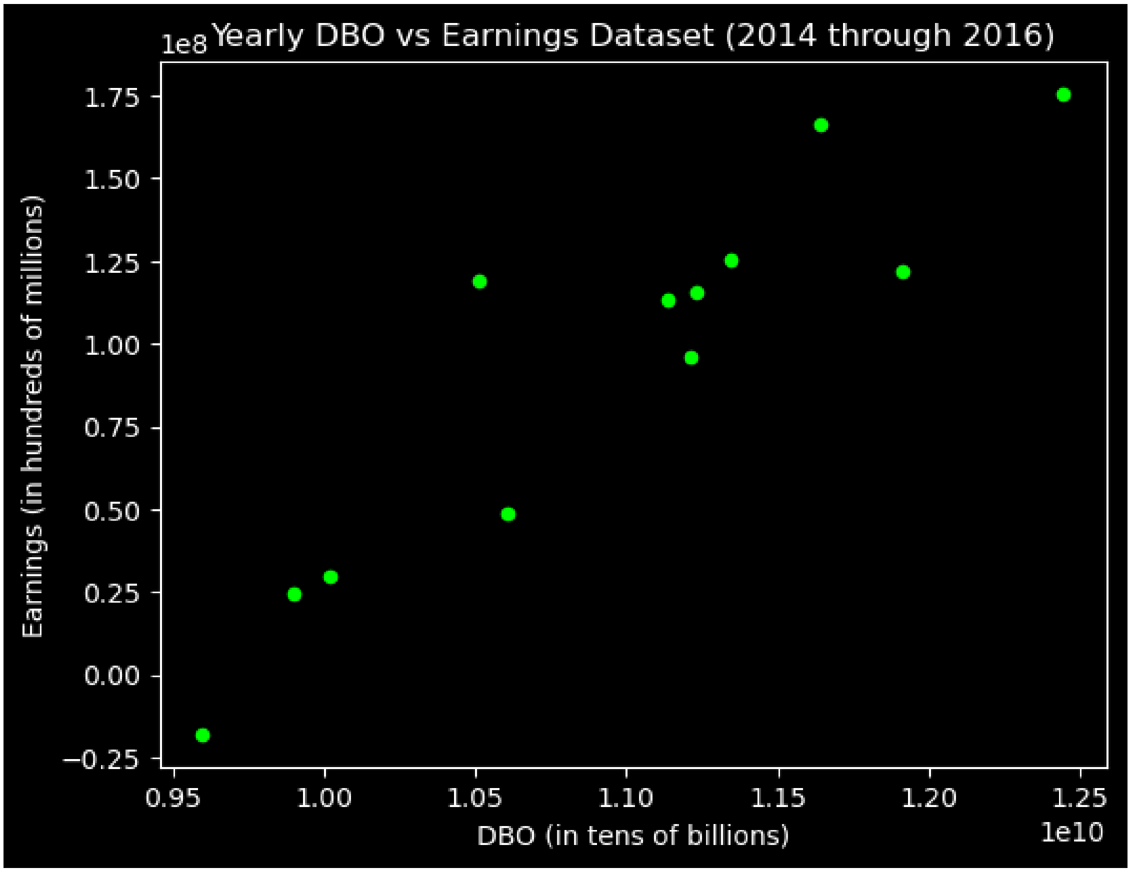
[ ]:
```
print(df.describe())
```

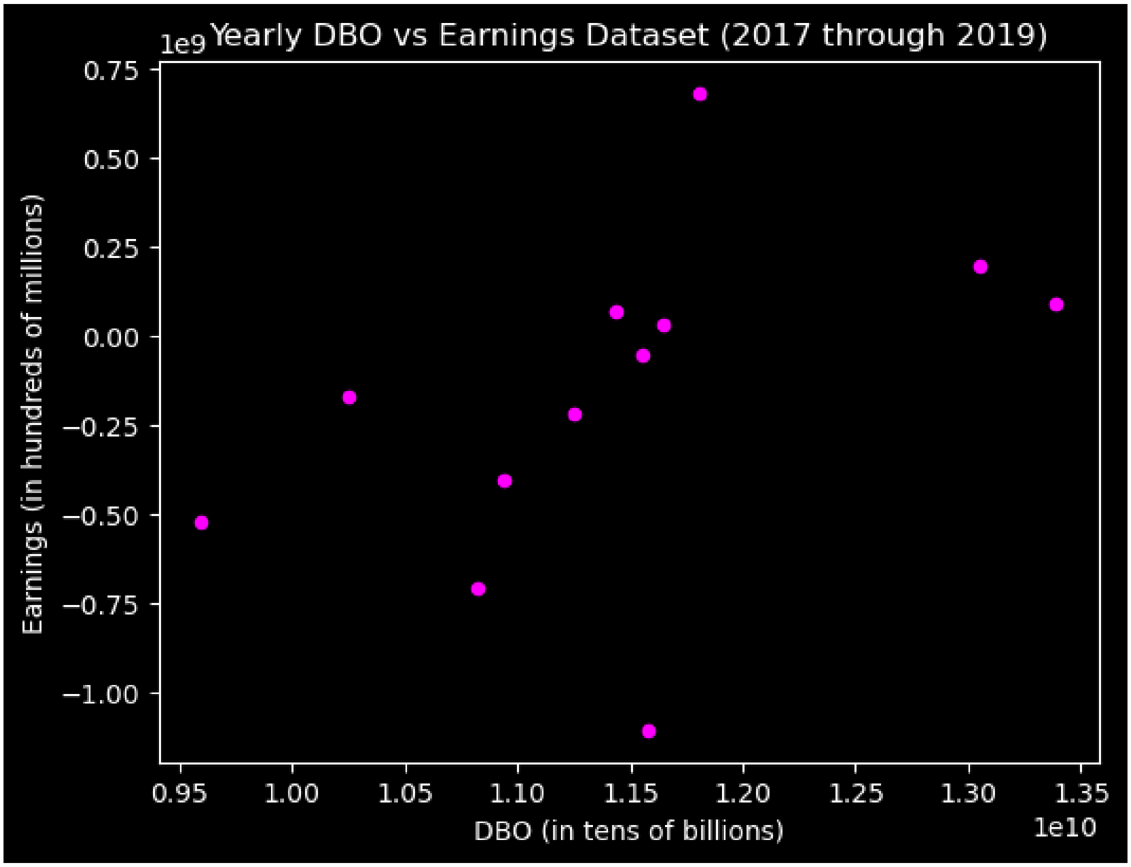
```
          DBO gross      earnings
count  3.500000e+01  3.500000e+01
mean   9.074312e+09 -5.603539e+08
std    3.788194e+09  1.039427e+09
min    1.890192e+07 -3.784400e+09
25%    7.982850e+09 -9.022000e+08
50%    1.060793e+10 -5.400000e+07
75%    1.149479e+10  1.045160e+08
max    1.338965e+10  6.824000e+08
```

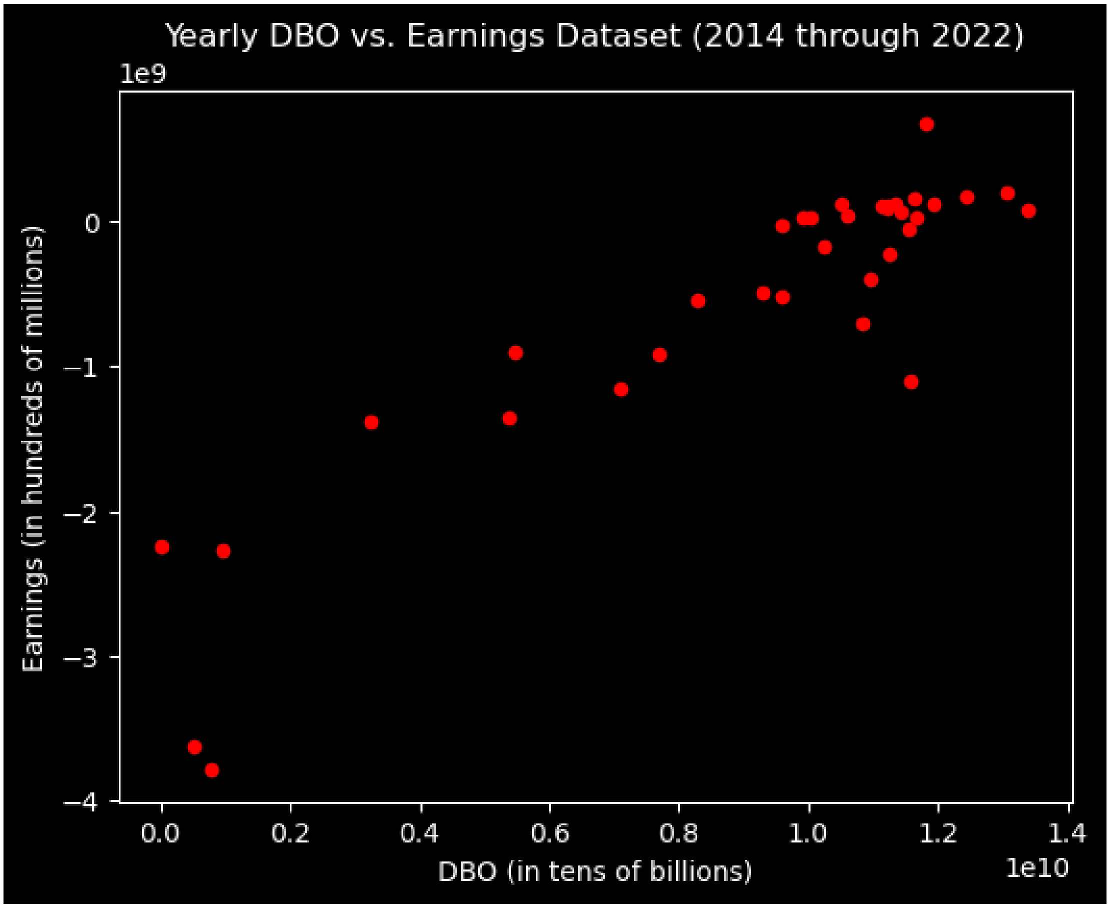
3.0.1 Data visualization of the datasets used for modeling

- We are visualizing how DBO and earnings might be correlated, and looking for potential linearity in each plot.

[ ]:
```
# See: analysis/plots/datasets.py
pdatasets(plt, df)
```


Yearly DBO vs Earnings Dataset (2014 through 2016)
1e8
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
−0.25
Earnings (in hundreds of millions)
0.95
1.00
1.05
1.10
1.15
1.20
1.25
1e10
DBO (in tens of billions)


Yearly DBO vs Earnings Dataset (2017 through 2019)
1e9
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
Earnings (in hundreds of millions)
0.95
1.00
1.05
1.10
1.15
1.20
1.25
1.30
1.35
1e10
DBO (in tens of billions)



3.0.2 Pearsons Correlation Coefficients

Here we find the z-scores for each model, so that we can evaluate whether a correlation exists, the nature about that correlation, and how strong any such correlation may be.

z-scores:

[ ]:
```
%%latex
$$ \zeta = \frac{x - \mu}{\sigma} $$
```

$$\zeta = \frac{x - \mu}{\sigma}$$

[ ]:
```
# Calculate the z-scores.
zeta = lambda x: (x - x.mean()) / x.std()

df["z-scoreX1"], df["z-scoreY1"] = zeta(x.iloc[:12]), zeta(y.iloc[:12])
df["z-scoreX2"], df["z-scoreY2"] = zeta(x.iloc[12:24]), zeta(y.iloc[12:24])
df["z-scoreX3"], df["z-scoreY3"] = zeta(x), zeta(y)
```

Pearsons Correlation Coefficient:

[ ]:
```
%%latex
$$ \rho = \frac{ \sum{\zeta_x \zeta_y} }{n - 1} $$
```

$$\rho = \frac{\sum \zeta_x \zeta_y}{n - 1}$$

[ ]:
```
# Calculate Pearsons Correlation Coefficients.
r1 = (df["z-scoreX1"] * df["z-scoreY1"]).sum() / (len(df.loc[:11]) - 1)
r2 = (df["z-scoreX2"] * df["z-scoreY2"]).sum() / (len(df.loc[12:23]) - 1)
r3 = (df["z-scoreX3"] * df["z-scoreY3"]).sum() / (len(df) - 1)

print(f"""
For 2014 through 2016, Pearsons Correlation Coefficient is: {r1}
For 2017 through 2019, Pearsons Correlation Coefficient is: {r2}
For 2014 through 2022, Pearsons Correlation Coefficient is: {r3}
""")
```

```
For 2014 through 2016, Pearsons Correlation Coefficient is: 0.9118679351959351
For 2017 through 2019, Pearsons Correlation Coefficient is: 0.45365018248882316
For 2014 through 2022, Pearsons Correlation Coefficient is: 0.9066831913698027
```

3.0.3 Summary about Pearsons Correlation Coefficients

We correlate DBO with earnings for three seperate time periods: 1) 2014 through 2016, 2) 2017 through 2019, and 3) 2014 through 2022. Each model is able to predict earnings from DBO, within the range of its DBO domain.

1. This is a very strong positive correlation of approximately: 0.91. This model will likely make very accurate predictions.
2. This correlation is a relatively weak positive correlation of approximately: 0.45, but we move forward with this model in an attempt to give Adam the benefit of the doubt. A more powerful model would be ideal, but it is better than nothing. Certainly it would not be fair to include Covid in the model. I assume in 2016 AMC was operationally much the same compared to 2014, and 2015. The major effect Adam had immediately after compeleting his first year as CEO was purchasing Odeon, so that is the rational basis of the time period.
3. Like (1), this correlation is a very strong positive correlation of approximately: 0.91

Next we will determine the best-fit functions for these datasets.

3.0.4 Line of Best-fit

The slope of the least-squares line:

[ ]:
```
%%latex
$$ \beta = \frac{ \sum{ \langle x - \bar{x} \rangle \langle y - \bar{y} \rangle
↪}
}{ \sum{ \langle x - \bar{x} \rangle^{2} } } $$
```

$$\beta = \frac{\sum \langle x - \bar{x} \rangle \langle y - \bar{y} \rangle}{\sum \langle x - \bar{x} \rangle^{2}}$$

[ ]:
```
# Calculate the slope of the least-squares line.
slope = lambda x, y: ((x - x.mean()) * (y - y.mean())).sum() / ((x - x.
↪mean())**2).sum()
df["b1"] = b1 = slope(x.iloc[:12], y.iloc[:12])
df["b2"] = b2 = slope(x.iloc[12:24], y.iloc[12:24])
df["b3"] = b3 = slope(x, y)
```

The y-intercept of the least-squares line:

[ ]:
```
%%latex
$$ \alpha = \bar{y} - \beta \bar{x} $$
```

$$\alpha = \bar{y} - \beta \bar{x}$$

[ ]:
```
# Calculate the y-intercept of the least-squares line.

intercept = lambda x, y, b: y.mean() - b * x.mean()
df["a1"] = a1 = intercept(x.iloc[:12], y.iloc[:12], b1)
df["a2"] = a2 = intercept(x.iloc[12:24], y.iloc[12:24], b2)
df["a3"] = a3 = intercept(x, y, b3)
```

- For 2014 through 2016:
  $\alpha = -598,645,528$
  $\beta \approx 0.06309998813676994$
- For 2017 through 2019:
  $\alpha = -2,482,569,804$
  $\beta \approx 0.20161160108675694$
- For 2014 through 2022:
  $\alpha = -2,817,870,262$
  $\beta \approx 0.24878099226102765$

[ ]:
```
# Display a and b.

print(f"""
For 2014 through 2016:
 a = {a1:,.0f}
 b = {b1}
```

```
For 2017 through 2019:
 a = {a2:,.0f}
 b = {b2}

For 2014 through 2022:
 a = {a3:,.0f}
 b = {b3}
""")
```

```
For 2014 through 2016:
 a = -598,645,528
 b = 0.06309998813676994

For 2017 through 2019:
 a = -2,482,569,804
 b = 0.20161160108675694

For 2014 through 2022:
 a = -2,817,870,262
 b = 0.24878099226102765
```

The function for the least-squares line

[ ]:
```
%%latex
$$ \hat{y} = f \langle x \rangle = \alpha + \beta x $$
```

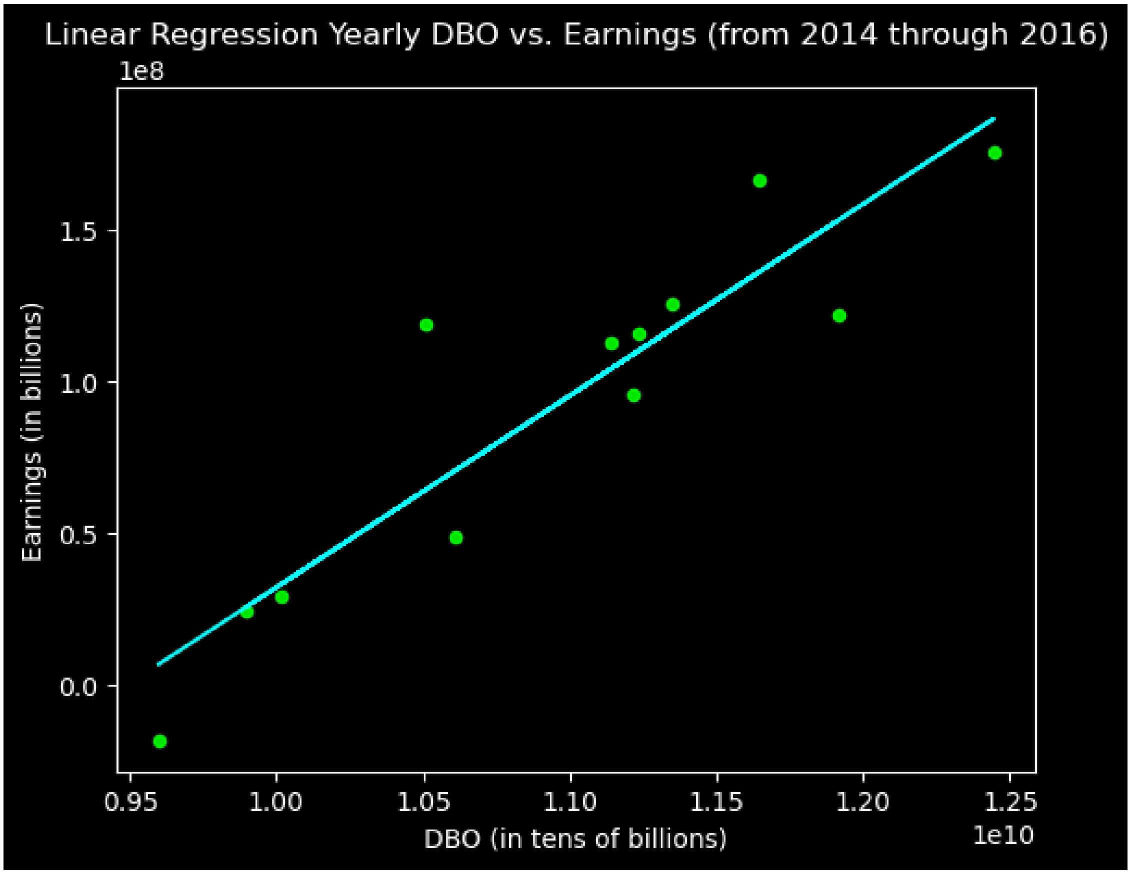
$$ \hat{y} = f \langle x \rangle = \alpha + \beta x $$

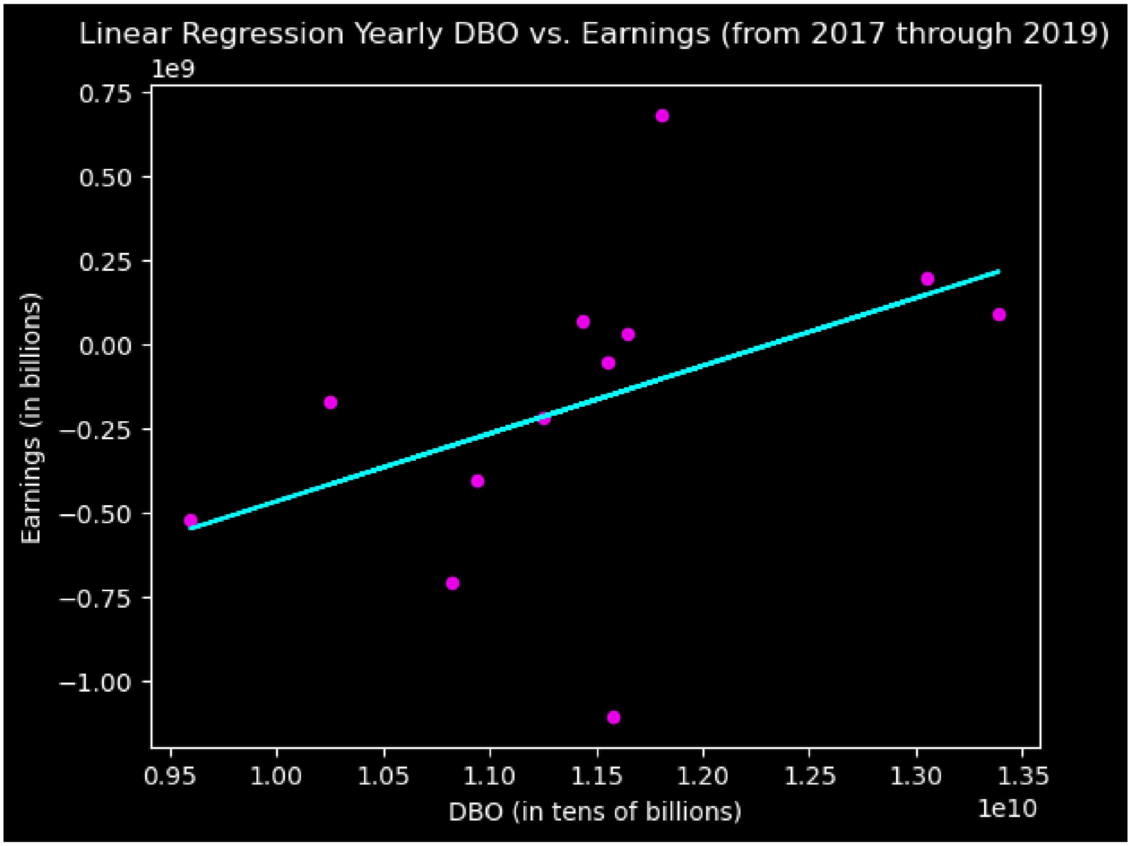
[ ]:
```
# Calculate y-hats.

hat = lambda a, b, x: a + b * x

df["y_hat1"] = y_hat1 = hat(a1, b1, x.iloc[:12])
df["y_hat2"] = y_hat2 = hat(a2, b2, x.iloc[12:24])
df["y_hat3"] = y_hat3 = hat(a3, b3, x)
```

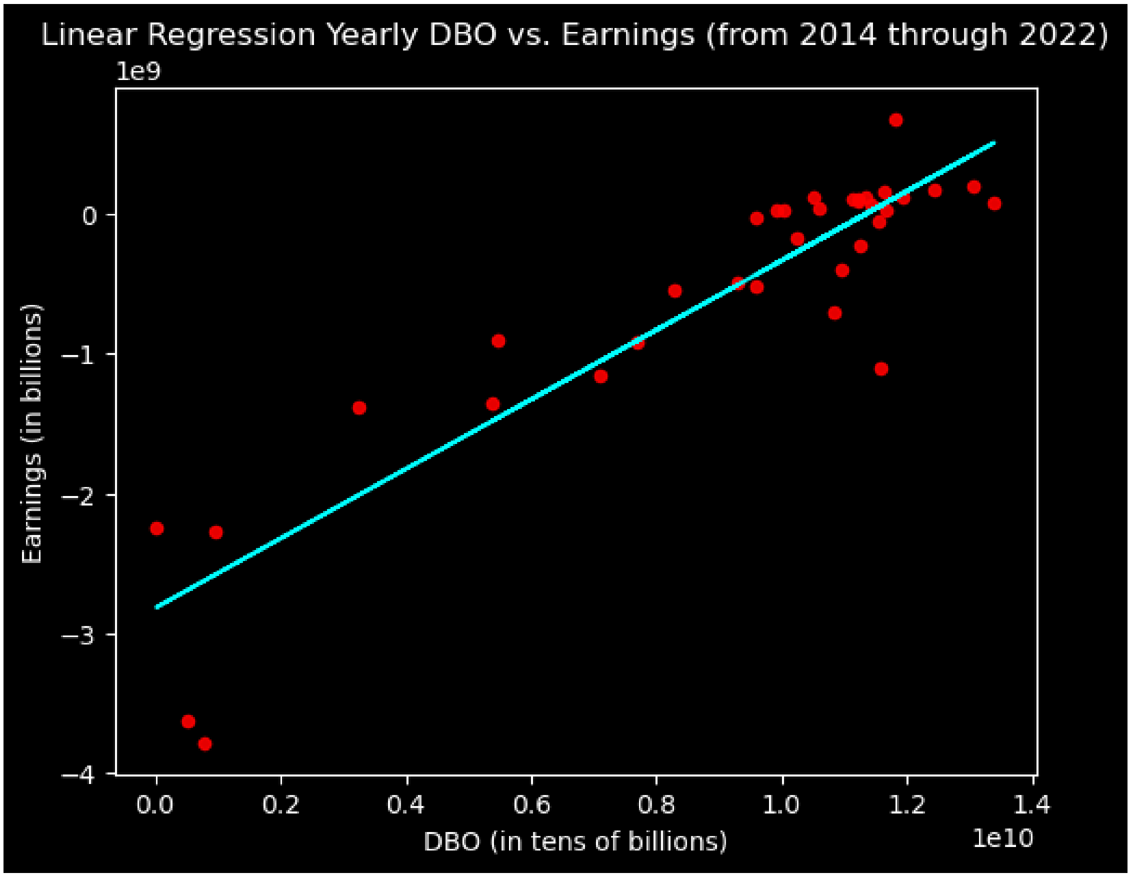
3.0.5 Linear regression plots

- Here we plot the linear function that predicts earnings against the dataset used to derive those predictions.

[ ]:
```
# See: analysis/scripts/preg.py
rplot(x, y_hat1, y_hat2, y_hat3, plt, df)
```


Linear Regression Yearly DBO vs. Earnings (from 2014 through 2016)
1e8
1.5
1.0
0.5
0.0
Earnings (in billions)
0.95
1.00
1.05
1.10
1.15
1.20
1.25
1e10
DBO (in tens of billions)


Linear Regression Yearly DBO vs. Earnings (from 2017 through 2019)
1e9
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
Earnings (in billions)
0.95
1.00
1.05
1.10
1.15
1.20
1.25
1.30
1.35
1e10
DBO (in tens of billions)



3.0.6 Evaluating Residuals to Assess the Regression Lines

Now, we construct the residual plots. In these plots we shall look for patterns. Especially such as strong curvatures, clusters, influential observations, or outliers. We see none so these plots look pretty good!

Residual:

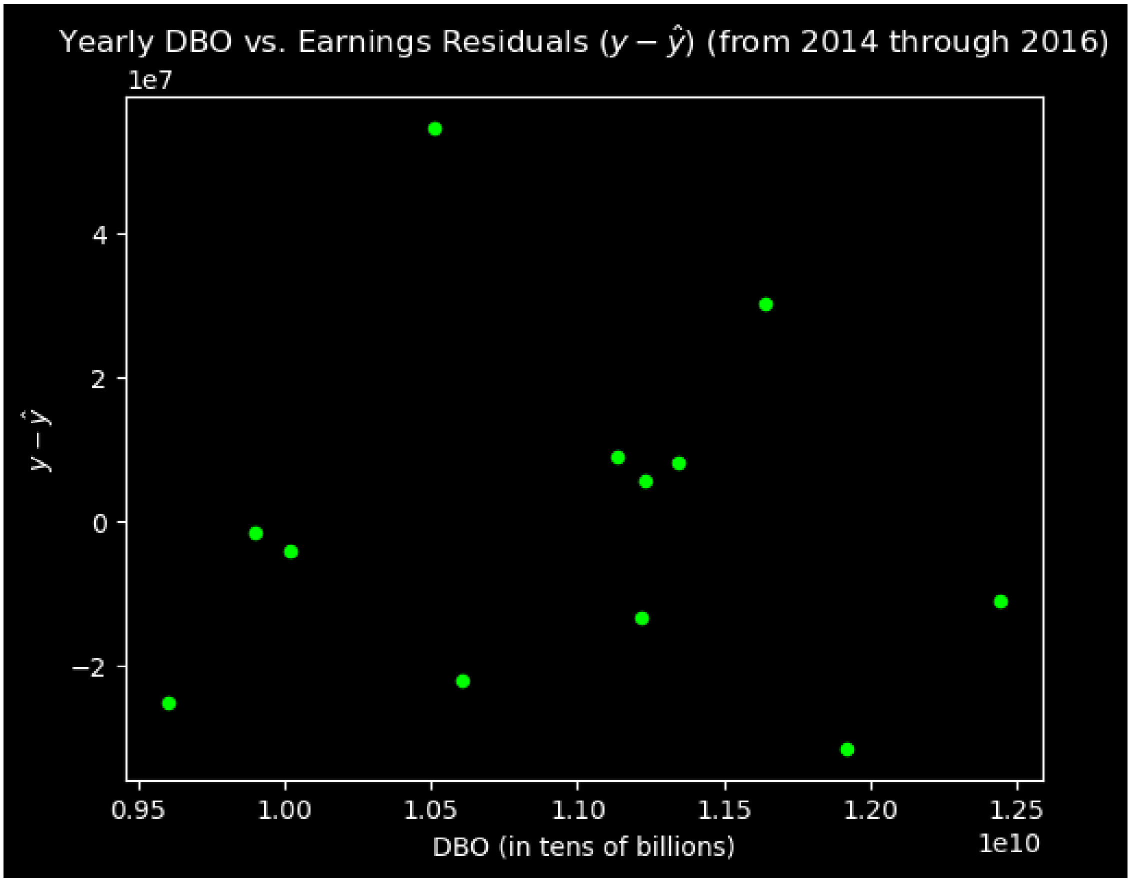
[ ]:
```
%%latex
$$ y - \hat{y} $$
```

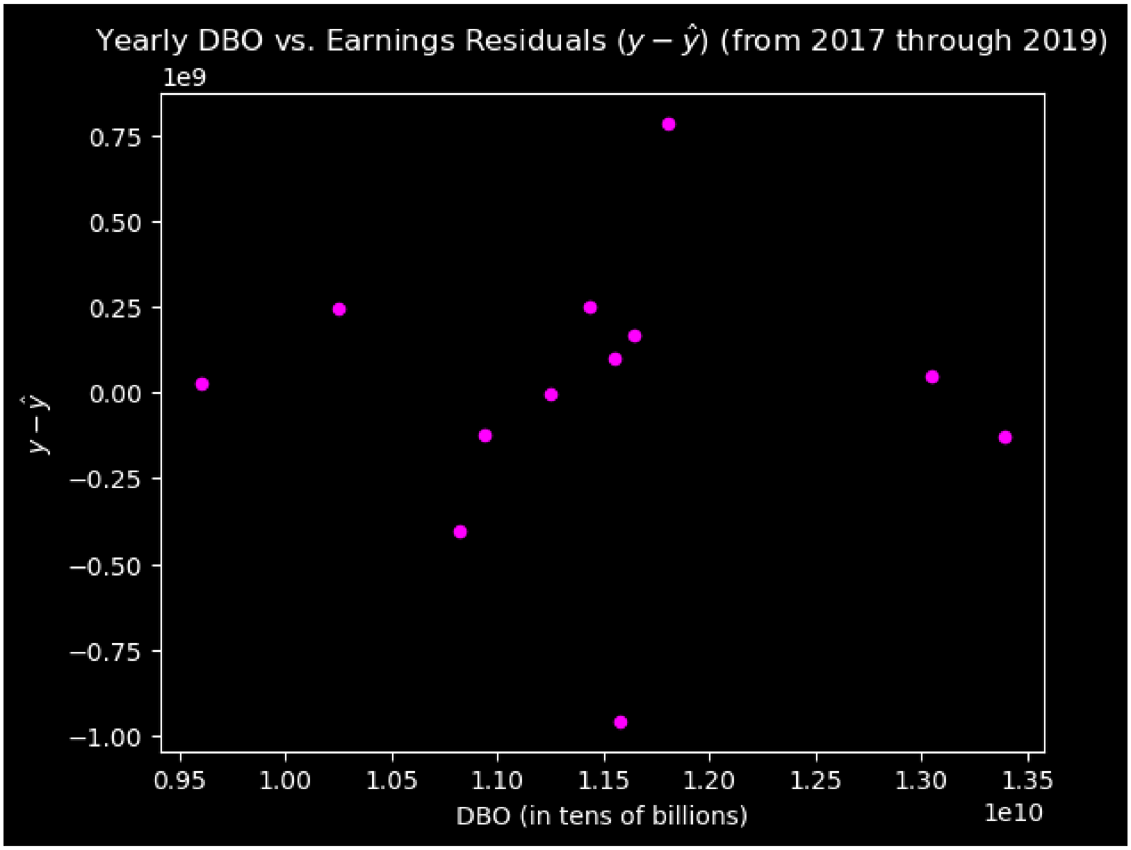
$$y - \hat{y}$$

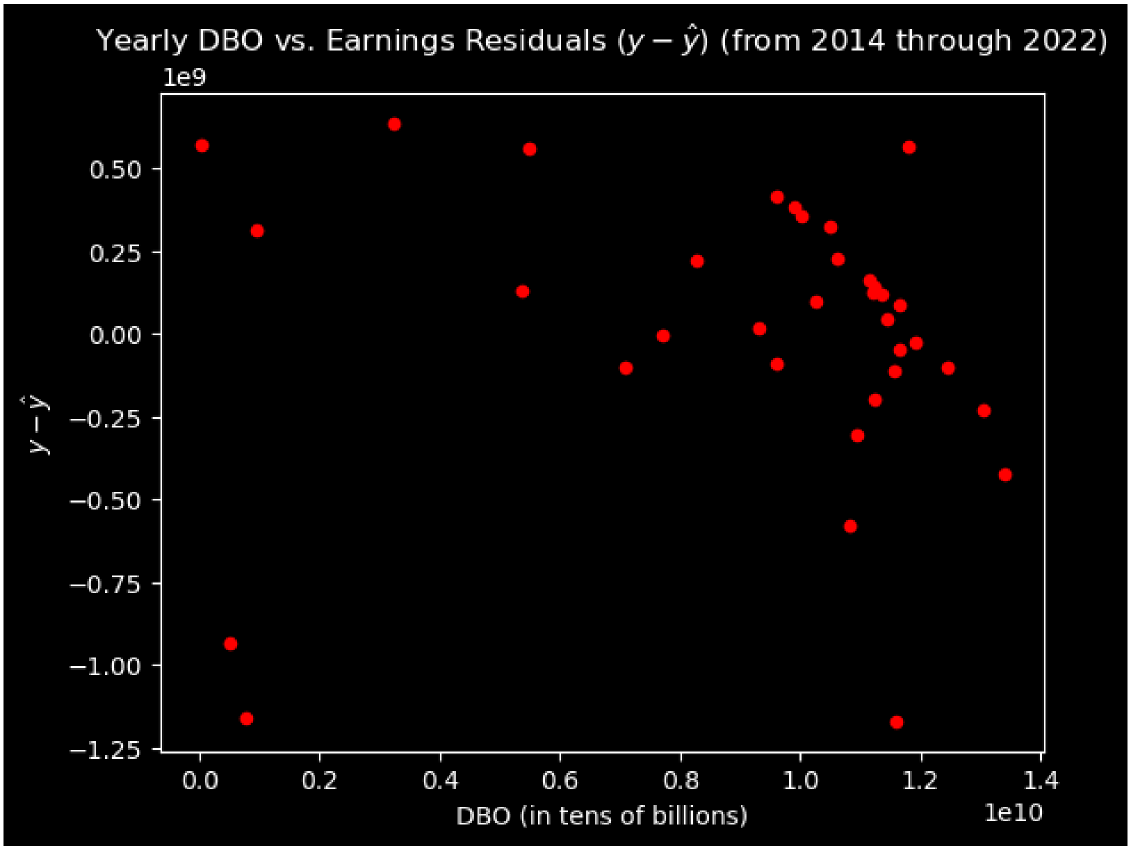
[ ]:
```
# Calculate the residuals.

df["residuals1"] = residuals1 = y.iloc[:12] - y_hat1
df["residuals2"] = residuals2 = y.iloc[12:24] - y_hat2
df["residuals3"] = residuals3 = y - y_hat3
```

3.0.7 Residual Plots

```
[ ]: # See: analysis/scripts/presiduals.py
     presiduals(plt, df)
```




Yearly DBO vs. Earnings Residuals ($y - \hat{y}$) (from 2017 through 2019)
1e9
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
$y - \hat{y}$
0.95
1.00
1.05
1.10
1.15
1.20
1.25
1.30
1.35
1e10
DBO (in tens of billions)



3.0.8 Coefficient of Determination.

This will tell us how well the models explain earnings as a consequence of DBO. We find the Coefficient of Determination by:

1. Finding the total sum of squares
2. Finding the residual sum of squares
3. Subtracting the quotient of (1) divided by (2) from the integer 1

The total sum of squares:

[ ]:
```
%%latex
$$ \tau = \sum{ \langle y - \bar{y} \rangle^{2}} $$
```

$$\tau = \sum \langle y - \bar{y} \rangle^2$$

[ ]:
```
# Calculate the total sum of squares.

squares = lambda y: (y - y.mean())**2

df["squares1"] = squares1 = squares(y.iloc[:12])
```

```
df["squares2"] = squares2 = squares(y.iloc[12:24])
df["squares3"] = squares3 = squares(y)

t1 = squares1.sum()
t2 = squares2.sum()
t3 = squares3.sum()
```

The residual sum of squares:

[ ]:
```
%%latex
$$ \epsilon = \sum{ \langle y - \hat{y} \rangle^{2}} $$
```

$$\epsilon = \sum \langle y - \hat{y} \rangle^{2}$$

[ ]:
```
# Calculate the residual sum of squares.

df["residual squares1"] = df["residuals1"]**2
df["residual squares2"] = df["residuals2"]**2
df["residual squares3"] = df["residuals3"]**2

e1 = df["residual squares1"].sum()
e2 = df["residual squares2"].sum()
e3 = df["residual squares3"].sum()
```

3.0.9 Coefficient of Determination

[ ]:
```
%%latex
$$ r^{2} = 1 - \frac{\epsilon}{\tau} $$
```

$$r^{2} = 1 - \frac{\epsilon}{\tau}$$

Possible values for the coefficient of determination are in the real-interval [0, 1]. 1 meanings the model is very good at making reliable predictions, and 0 meaning that the model essentially doesn't work.

For 2014 through 2016, the Coefficient of Determination is 0.8315031312384981

For 2017 through 2019, the Coefficient of Determination is 0.2057984880721423

For 2014 through 2022, the Coefficient of Determination is 0.8220744095125301

The second model leaves a lot to be desired, but we need it for making a comparison, and getting some idea about the facts. A coefficient of determination of .83 says: 83% of the variation in earnings can be explained by domestic box office. The first and third models have very high predictive accuracy.

[ ]:
```
# Calculate the Coefficient of Determination.
```

```
r_2_1 = 1 - (e1 / t1)
r_2_2 = 1 - (e2 / t2)
r_2_3 = 1 - (e3 / t3)

print(f"""
For 2014 through 2016:
 The Coefficient of Determination is: {r_2_1}

For 2017 through 2019:
 The Coefficient of Determination is: {r_2_2}

For 2014 through 2022:
 The Coefficient of Determination is: {r_2_3}
""")
```

```
For 2014 through 2016:
 The Coefficient of Determination is: 0.8315031312384981

For 2017 through 2019:
 The Coefficient of Determination is: 0.2057984880721423

For 2014 through 2022:
 The Coefficient of Determination is: 0.8220744095125301
```

3.0.10 The standard deviation about the least-squares line σ_ϵ

This will tell us the range of error in the prediction.

We want $\sigma < \sigma_\epsilon$. In all three models, such is the case.

[ ]:
```
%%latex
$$ \sigma_{\epsilon} = \sqrt{ \frac{\epsilon}{n - 2} } $$
```

$$\sigma_\epsilon = \sqrt{\frac{\epsilon}{n-2}}$$

For 2014 through 2016, the Standard deviation about y is 59,186,777.5202432. The Standard deviation about the least-squares line is 25,554,998.54506667

For 2017 through 2019, the Standard deviation about y is 465,442,163.1875729. The Standard deviation about the least-squares line is 435,038,226.38255721

For 2014 through 2022, the Standard deviation about y is: 1,039,426,652.4898386. The Standard deviation about the least-squares line: 445,036,411.52865213

[ ]:
```
# Calculate the standard deviation about the least-squares line.
```

```python
rssd = lambda e, df: math.sqrt((e / (len(df) - 2)))

sd1  = y.loc[:12].std()
sd2  = y.loc[12:24].std()
sd3  = y.std()

r1   = rssd(e1, df.iloc[:12])
r2   = rssd(e2, df.iloc[12:24])
r3   = rssd(e3, df)

print(f"""
For 2014 through 2016:
 The Standard deviation about y is: {sd1:,.7f}
 The Standard deviation about the least-squares line: {r1:,.8f}

For 2017 through 2019:
 The Standard deviation about y is: {sd2:,.7f}
 The Standard deviation about the least-squares line: {r2:,.8f}

For 2014 through 2022:
 The Standard deviation about y is: {sd3:,.7f}
 The Standard deviation about the least-squares line: {r3:,.8f}
""")
```

```
For 2014 through 2016:
 The Standard deviation about y is: 59,186,777.5202432
 The Standard deviation about the least-squares line: 25,554,998.54506667

For 2017 through 2019:
 The Standard deviation about y is: 465,442,163.1875729
 The Standard deviation about the least-squares line: 435,038,226.38255721

For 2014 through 2022:
 The Standard deviation about y is: 1,039,426,652.4898386
 The Standard deviation about the least-squares line: 445,036,411.52865213
```

4 Predictions

Earnings is a function of DBO. So that means we can predict what the earnings will be based on DBO as input. We will input various DBO into the predictive functions to predict earnings.

4.0.1 First we compare the 3 predictive models side by side

Look at how each model differs between each column. Something fundamentally changed for the worse since Adam took over AMC. Even though the second model *(2017 - 2019)* isn't very powerful, together with third model we get a pretty good idea of the difference. Earnings were robust in *(2014 - 2016)*, and depended on far less DBO to be net postive. Adam became CEO in 2016, and bought Odeon in Q3 2016 & Nordic in Q1 2017. AMC earnings have been down hill ever since. In fact, other than being a set back, we can't even say Covid made it much worse. It almost appears to not even be a contributing factor to any fundamental issues in any significant way.

Clearly AMC was fundamentally in a much better position prior to Adam Aron becoming CEO because far less DBO was required to turn a profit, before Adam took over.

[ ]:
```python
# Compute the lower and upper bound for prediction.

m1 = x.loc[:12].min()
m2 = x.loc[12:24].min()
m3 = x.min()
minimum = max([m1, m2, m3])

m1 = x.loc[:12].max()
m2 = x.loc[12:24].max()
m3 = x.max()
maximum = min([m1, m2, m3])

print("The DBO minimum and maximum predictive range for all 3 models together is:\n"
      f" minimum = {minimum:,.0f}\n maximum = {maximum:,.0f}")
```

```
The DBO minimum and maximum predictive range for all 3 models together is:
 minimum = 9,600,109,748
 maximum = 12,445,502,384
```

[ ]:
```python
# See: analysis/scripts/comparative.py
mY1, mY2, mY3 = comparative(a1, a2, a3, b1, b2, b3)
```

All Predictions

DBO	Earnings (2014 - 2016)	Earnings (2017 - 2019)	Earnings (2014 - 2022)
9,600,000,000	7,114,358	-547,098,434	-429,572,737
9,850,000,000	22,889,355	-496,695,534	-367,377,489
10,100,000,000	38,664,352	-446,292,633	-305,182,240
10,350,000,000	54,439,349	-395,889,733	-242,986,992
10,600,000,000	70,214,346	-345,486,833	-180,791,744
10,850,000,000	85,989,343	-295,083,933	-118,596,496
11,100,000,000	101,764,340	-244,681,032	-56,401,248
11,350,000,000	117,539,337	-194,278,132	5,794,000
11,600,000,000	133,314,334	-143,875,232	67,989,248
11,850,000,000	149,089,331	-93,472,332	130,184,496
12,100,000,000	164,864,329	-43,069,431	192,379,744
12,350,000,000	180,639,326	7,333,469	254,574,992

```
[ ]: # Calculate summary statistics about each prediction.

print(f"""
Approximate descriptive statistics of the predicted earnings.

 For 2014 through 2016:

    mean:   {mY1[0]:,.0f}
    min:    {mY1[1]:,.0f}
    max:    {mY1[2]:,.0f}

 For 2017 through 2019:

    mean:   {mY2[0]:,.0f}
    min:    {mY2[1]:,.0f}
    max:    {mY2[2]:,.0f}

 For 2014 through 2022:

    mean:   {mY3[0]:,.0f}
    min:    {mY3[1]:,.0f}
    max:    {mY3[2]:,.0f}

""")
```

```
Approximate descriptive statistics of the predicted earnings.

 For 2014 through 2016:

    mean:   93,876,842
    min:    7,114,358
    max:    180,639,326

 For 2017 through 2019:

    mean:   -269,882,483
    min:    -547,098,434
    max:    7,333,469

 For 2014 through 2022:

    mean:   -87,498,872
    min:    -429,572,737
    max:    254,574,992
```

4.0.2 We can also see the fundamental differences by the summary statistics on the prediction output.

Approximated descriptive statistics about the predicted earnings.

For 2014 through 2016:

```
mean:   93,876,842
min:    7,114,358
max:    180,639,326
```

For 2017 through 2019:

```
mean:   -269,882,483
min:    -547,098,434
max:    7,333,469
```

For 2014 through 2022:

```
mean:   -87,498,872
min:    -429,572,737
max:    254,574,992
```

5 2023 Yearly Earnings Prediction

[ ]:
```
# See: analysis/scripts/prediction.py
predict(a3, b3, r3, df)
```

2023 Earnings from DBO Prediction

DBO	Lowest	Low	Earnings	High	Highest
7,500,000,000	-1,842,085,643	-1,397,049,232	-952,012,820	-506,976,409	-61,939,997
8,000,000,000	-1,717,695,147	-1,272,658,736	-827,622,324	-382,585,913	62,450,499
8,500,000,000	-1,593,304,651	-1,148,268,240	-703,231,828	-258,195,417	186,840,995
9,000,000,000	-1,468,914,155	-1,023,877,743	-578,841,332	-133,804,920	311,231,491
9,500,000,000	-1,344,523,659	-899,487,247	-454,450,836	-9,414,424	435,621,987
10,000,000,000	-1,220,133,163	-775,096,751	-330,060,340	114,976,072	560,012,483
10,500,000,000	-1,095,742,667	-650,706,255	-205,669,844	239,366,568	684,402,980
11,000,000,000	-971,352,170	-526,315,759	-81,279,347	363,757,064	808,793,476
11,500,000,000	-846,961,674	-401,925,263	43,111,149	488,147,560	933,183,972
12,000,000,000	-722,571,178	-277,534,767	167,501,645	612,538,056	1,057,574,468
12,500,000,000	-598,180,682	-153,144,271	291,892,141	736,928,553	1,181,964,964
13,000,000,000	-473,790,186	-28,753,774	416,282,637	861,319,049	1,306,355,460
13,500,000,000	-349,399,690	95,636,722	540,673,133	985,709,545	1,430,745,956

- The projected 2023 domestic box office is expected to be $9B
- The 'Earnings' column is the prediction
- There is an approximate 68% chance that the actual earnings will fall within the range 'Low' to 'High'

- There is an approximate 98% chance that the actual earnings will fall within the range 'Lowest' to 'Highest'
- Notice that the model successfully predicted 2022 AMC earnings from a $7.5 billion DBO with extreme accuaracy!

This model has incredibly strong predictive power.

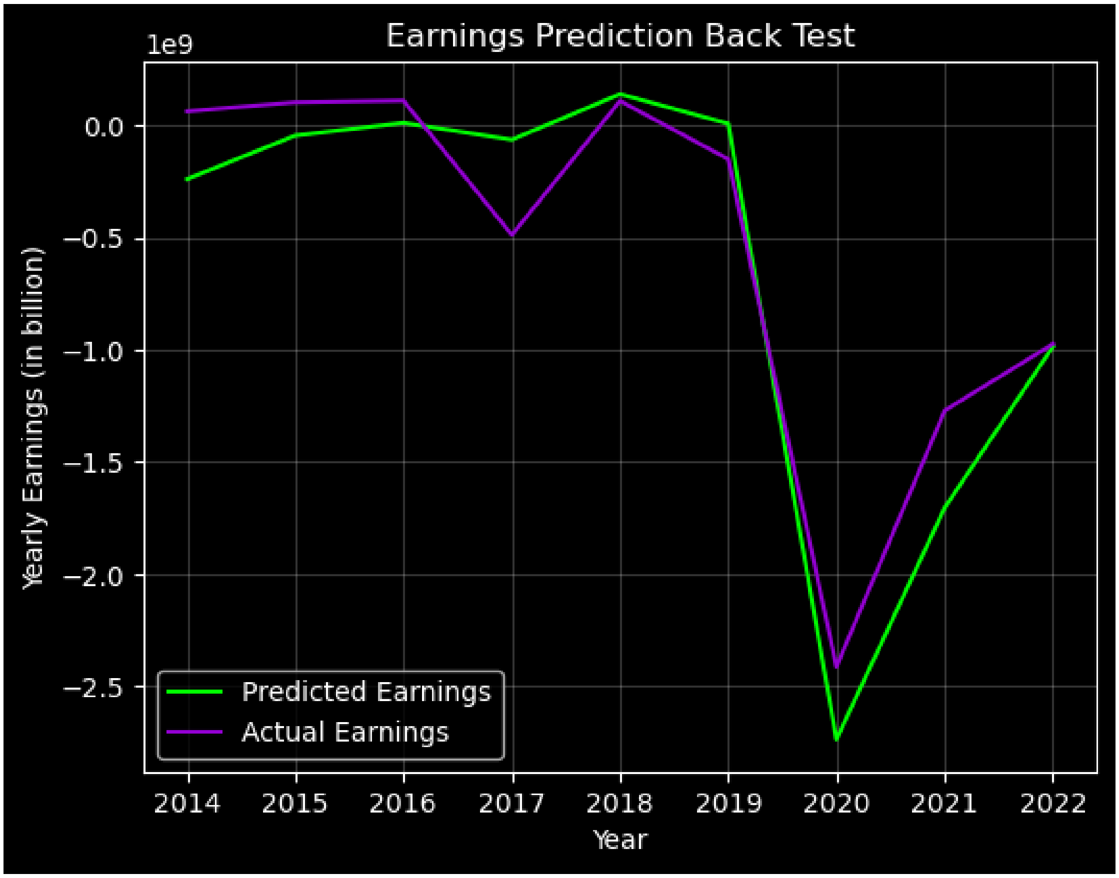
5.0.1 How well does the predictive model actually perform?

Let's back test it. We will take actual yearly DBO from 2014 through 2022 as input to the predictive model, and compare the predicted values to the actual values. Then we shall graphically visualize the results.

[ ]:
```
# Save the models.
df.to_csv("data/model.csv", index=True)
```

[ ]:
```
# Load data.

df = pd.read_csv("data/data.csv", index_col = 0)
df = df.drop(24).groupby("year").agg("sum").drop(columns = ["Q", "movies",␣
  ↪"revenue"])
df.index = [datetime.strptime(str(int(i)), "%Y") for i in df.index]
```

[ ]:
```
# See: analysis/scripts/backtest.py
backtest(a3, b3, plt, df)
```



5.0.2 Amazing predictive capability!

[ ]:
```
# Load data.

df = pd.read_csv("data/dilution_history.csv", index_col=0)
```

[ ]:
```
# Calculate and report float size percentage increase.

# The size of the float before Adam became CEO
f0 = df.loc[0].float

# 2016 to 2020 (excluding 2020) float size increase
f1 = df.loc[3].float / df.loc[0].float

# 2020 to 2023 float size increase
f2 = df.loc[11].float / df.loc[3].float

# Since 2016 float size increase
f3 = df.loc[11].float / df.loc[0].float
```

```
# If reverse split passes
f4 = 1032 / df.loc[0].float

print(f"\nFrom 2016 through 2019 Adam increased the float {f1 * 100:,.2f}%")
print(f"From 2020 to 2023, only, Adam increased the float {f2 * 100:,.2f}%")
print(f"Adams total float increase is {f3 * 100:,.2f}%")
print(f"If rs-c passed the total float increase would be {f4 * 100:,.2f}%␣
↪before the split")
print(f"\nThat is nearly {47.82:,.2f} times the size of the float "
       "before Adam took over AMC.")
```

```
From 2016 through 2019 Adam increased the float 255.24%
From 2020 to 2023, only, Adam increased the float 938.31%
Adams total float increase is 2,394.90%
If rs-c passed the total float increase would be 4,782.21% before the split

That is nearly 47.82 times the size of the float before Adam took over AMC.
```

6 Float Size History Since 2016

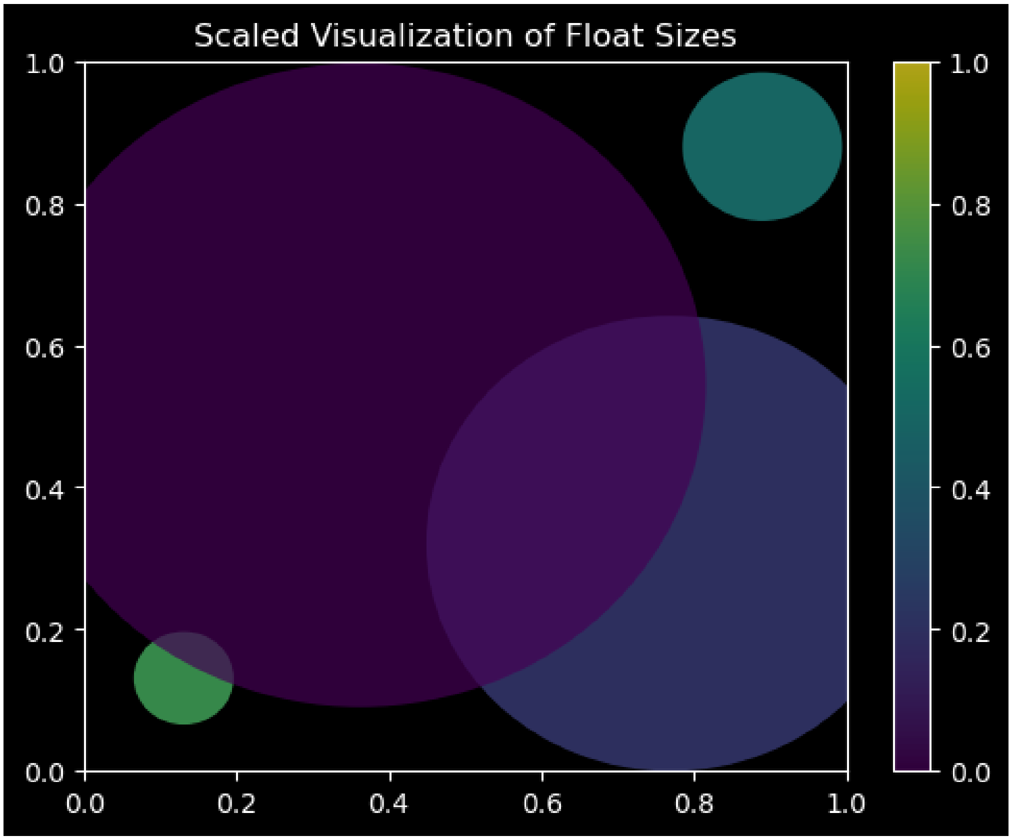
- From 2016 through 2019 Adam increased the float 255.24%
- From 2020 to 2023, only, Adam increased the float 938.31%
- Adams total float increase is 2,394.90%
- If the reverse split were passed the total float increase would be 4,782.21%, before the split

That is nearly 47.82 times the size of the float before Adam took over AMC.

6.0.1 Float Increase Scaled Visualization

Each bubble in the plot below shows the size of the float at the end of a time period. In increasing sizes: the smallest is the size of the float when Adam became CEO, followed by how much the float increased by the end of 2019, and then the size of the float by the end of 2022. The largest bubble is the size of the float after the APE conversion, but before the reverse split.

[ ]:
```
# See: analysis/scripts/pfloat.py
pfloat(f0, f1, f2, f3, f4, 40, plt, df)
```



6.0.2 Interesting random facts about the float:

- There's a 39% chance Adam would dilute AMC, each quarter.
- There's an 157% chance, each year. In other words, Adam dilutes more than 3 times every 2 years.
- From 2016 through 2022 is 7 years, and Adam has diluted AMC 11 times out of those 7 years.

[ ]:
```
# Calculate dilution frequency.
events = len(df) - 1
quarterly_freq = events / num_quarters
yearly_freq    = quarterly_freq * 4
print(f"There's a {quarterly_freq * 100:,.0f}% chance Adam would dilute AMC,␣
↪each quarter.")
print(f"There's an {yearly_freq * 100:,.0f}% chance Adam would dilute AMC, each␣
↪year.")
print(f"From 2016 through 2022 is {num_quarters / 4:,.0f} years.")
print(f"Adam has diluted AMC {events} times out of those {num_quarters / 4:,.
↪0f} years.")
```

```
There's a 39% chance Adam would dilute AMC, each quarter.
There's an 157% chance Adam would dilute AMC, each year.
```

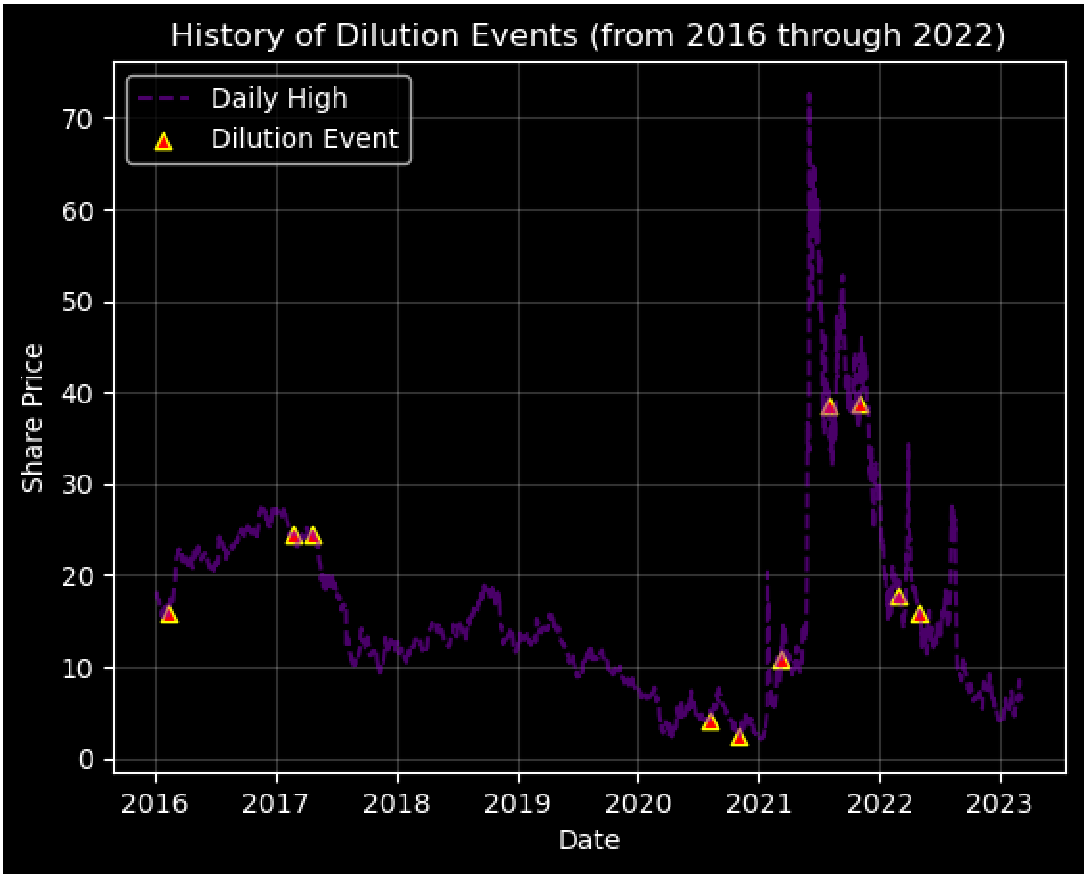
```
From 2016 through 2022 is 7 years.
Adam has diluted AMC 11 times out of those 7 years.
```

[ ]:
```python
# Load data.

events = df
df = pd.read_csv("data/dilution.csv", index_col=0)[474:]
df.dilution = np.nan

for e, row in enumerate(events.iterrows()):
    datum = df[df.date == row[1].date]
    try: index = datum.index[0]
    except IndexError: continue
    high  = datum.high.values[0]
    df.at[index, "dilution"] = high

for e, row in enumerate(df.iterrows()):
    date = row[1].date
    date = datetime.strptime(date, "%Y-%m-%d")
    df.at[474 + e, "date"] = date

del events
```

6.0.3 Visualization of the AMC Dilution History From 2016 Through 2022

- Note: in the below graph 2 dilution events are overlapping in 2021, making 1 of them imperceptible.

[ ]:
```python
# See: analysis/scripts/dhist.py
hplot(plt, df)
```



6.1 Has Dilution Ever Caused AMC Price Increases Since 2014?

Again we turn to regression analysis to see if dilution has ever caused positive price action in AMC since Adam has been the CEO. A dataset is constructed from float size history, and daily high price history. We look for strong evidence of a correlation, such that we could reasonabily posit causality. The following are the major points of the analytic process:

- The x- and y-values are extracted and cleaned from the tables containing the raw data.
- A visualization of the raw data, such that we can look for linearity, is given.
- Pearsons Correlation Coefficients are calculated.
- Rigorous treatment is given to outliers and influential observations.
- The residuals are calculated of the model with the highest correlation, and visualized.
- On the best predictive model the Coefficient of Determination is calculated.
- The Standard Deviation about the Least-Squares Line is calculated.
- Finally, the conclusion about all the evidence is summarized.

6.1.1 Data Extraction Algorithm

The x-values are the change in float size as a percentage of that float size. The y-values are the maximum daily high within the following 15-day period of a dilution event. Both x and y are normalized so that they are neither over nor underweighted.

[ ]:
```python
# Load data.
df = pd.read_csv("data/dilution.csv", index_col = 0)
```

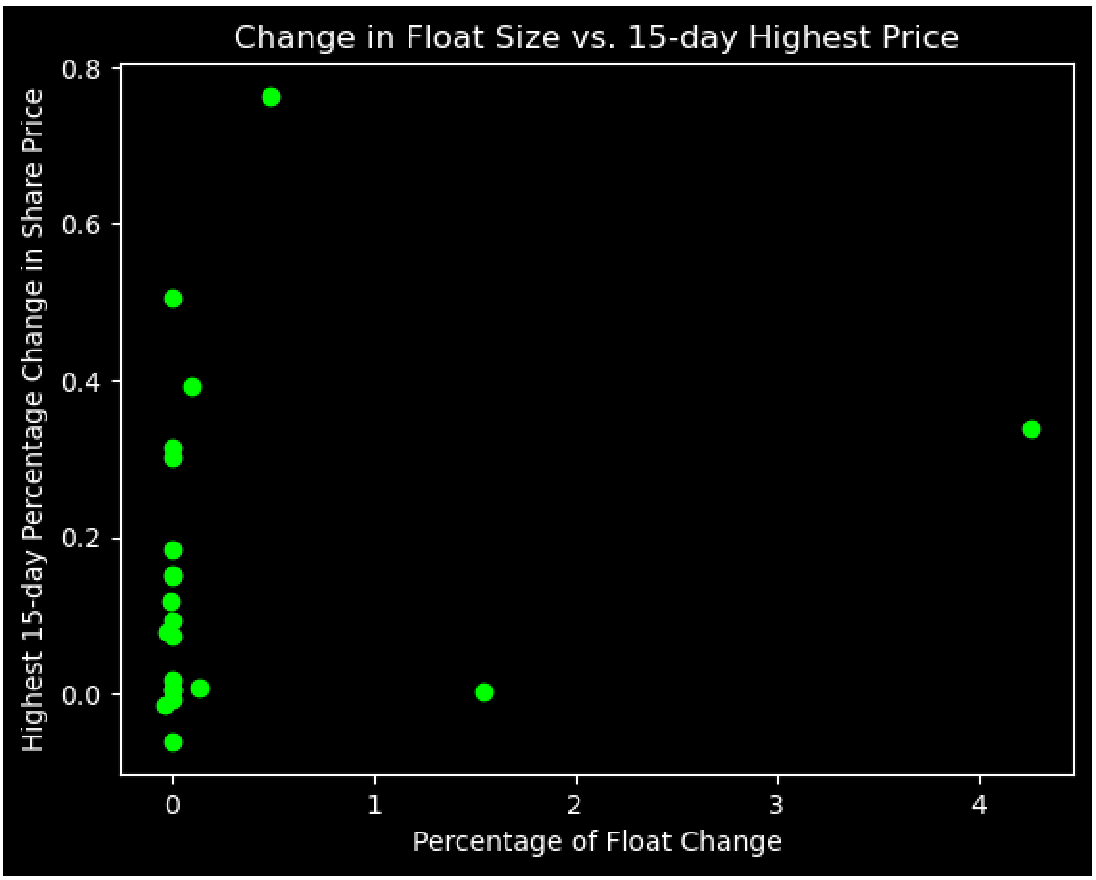
[ ]:
```python
# the algorithm for constructing the data.

def extract(df):

    X, Y = [], []

    for e, row in enumerate(df.iterrows()):

        # Test to see if the float size has increased.
        # If not continue searching for dilution events.
        dilution = row[1].dilution
        if not dilution: continue

        # Find the number of shares by which the float was diluted
        prev_float = df.loc[e-1].FF
        curr_float = row[1].FF
        difference = curr_float - prev_float

        # How many shares out of the float (before dilution) were diluted?
        # That is to say, by which percentage of the float was the float␣
↪diluted?
        # For normalization.
        fpercent = difference / prev_float

        # Get the daily high share price at the time dilution occurred.
        baseline = row[1].high

        # Get the highest daily high from the next 15 days that follow.
        cohort = df[e+1:e+15].high

        # Get the difference from the highest 15-day share price from the␣
↪baseline
        delta = max(cohort) - baseline

        # Get the delta price as a percentage of the baseline share price.
        # For normalization.
        dpercent = delta / baseline

        # Store the data points in memory.
        X.append(fpercent)
        Y.append(dpercent)

    return X, Y
```

```
X, Y = extract(df)
```

```
[ ]: # Load the x y values into a new dataframe.

     df = pd.DataFrame(np.array([X, Y]).T, columns = ["x", "y"])
```

6.1.2 The Dataset Visualization

```
[ ]: # See: analysis/scripts/dilution.py
     dilute(plt, df, color="lime", fname="raw")
```




```
[ ]: # Calculate Pearsons Correlation Coefficient.

     # z-scores
     df["z-scoreX"] = zeta(df.x)
     df["z-scoreY"] = zeta(df.y)

     # Pearsons Correlation Coefficient
     r = (df["z-scoreX"] * df["z-scoreY"]).sum() / (len(df) - 1)
     print(r)
```

```
0.19848990989791165
```

Upon visual inspection it appears as if, in terms of searching for a correlation between float size changes via dilution or float shrink, and price movements, no correlation is evident. Calculating Pearsons Correlation Coefficient for this data results in approximately 0.1985. That is to say, a very weak positive correlation. Removing outliers makes the correlation decrease even further with tight bounds defined for outliers such that y is within $1 \pm \sigma$ of $\bar{y}$. Removing influential observations (defined as x values within $1 \pm \sigma$ of $\bar{x}$) does improve the Pearsons Correlation coefficient to approximately 0.6779.

Note that the order in which outliers is removed consistently produces undesirable results. Removing outliers from the raw data, or first removing influencial observations, and then removing outliers. Neither result in an improvement in the correlation greater than removing influential observations only.

[ ]:
```
# Remove influencial observations.

df = df[df.x < df.x.mean() + df.x.std()]
df = df[df.x > df.x.mean() - df.x.std()]
```

[ ]:
```
# Calculate Pearsons Correlation Coefficient.

# z-scores
df["z-scoreX"] = zeta(df.x)
df["z-scoreY"] = zeta(df.y)

# Pearsons Correlation Coefficient
r = (df["z-scoreX"] * df["z-scoreY"]).sum() / (len(df) - 1)
print(r)
```

```
0.6778825337279126
```

[ ]:
```
# See if there are still influencial observations.

df[df.x >= df.x.mean() + df.x.std()]
```

[ ]:

	x	y	z-scoreX	z-scoreY
14	0.48775	0.763052	3.881232	2.851477

By removing influencial observations we are able to reach a Pearsons Correlation Coefficient of approximately 0.6679. Notice however that this means we cannot find a correlation for large increases in float sizes from this data. By that, I mean: one of the influential observations that was removed consisted of a 400% increase in the float. This means we can not say large increases in the float are correlated with increases in price. In fact, it is quite the opposite since including that in the data significantly eroded the correlation. The largest float size increase remaining in the dataset is about a 50% increase in float size. Although that is substantial, it is a redundant influencial observation relative to the modifed data. However, removing it reduces the potential predictive range, and significantly reduces Pearsons Correlation Coefficient. So we shall proceed with the analysis using this data.

[ ]:
```
# Calculate the slope of the least-squares line.

b = ((df.x - df.x.mean()) * (df.y - df.y.mean())).sum() / ((df.x - df.x.
↪mean())**2).sum()
```

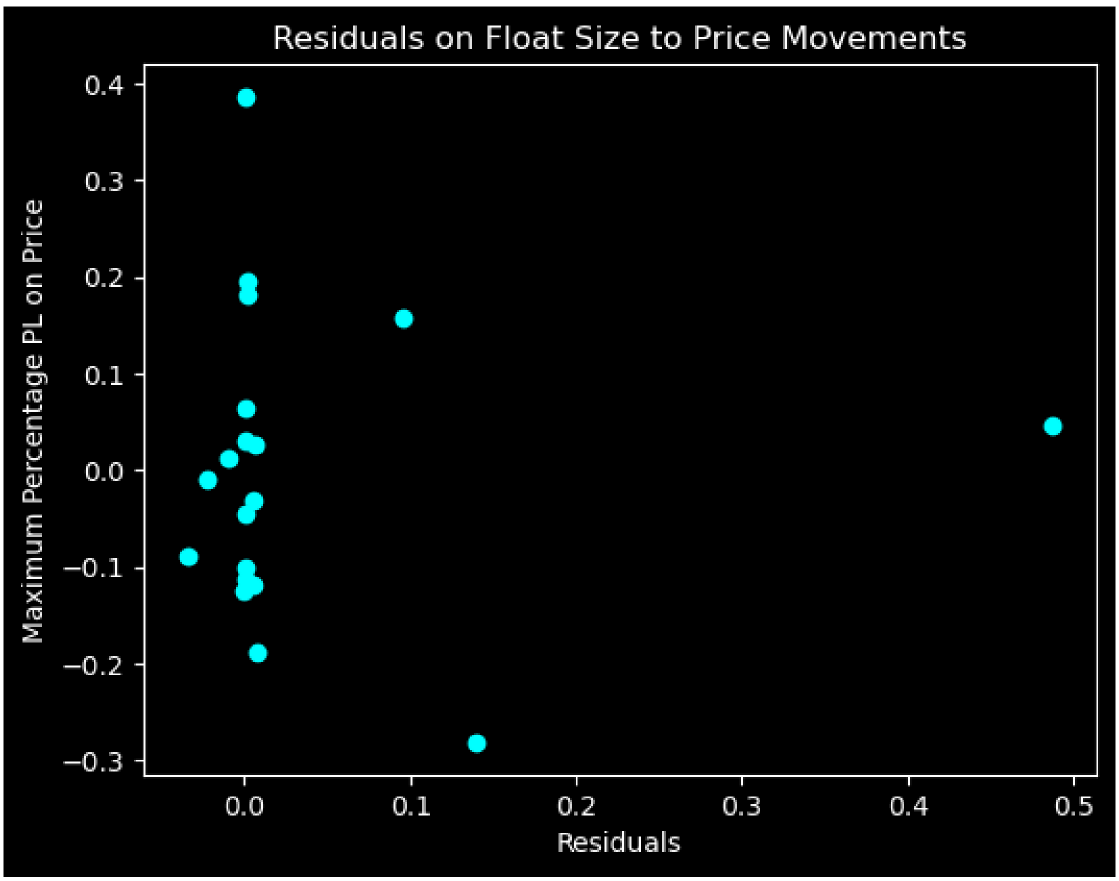
[ ]:
```
# Calculate the y-intercept of the least-squares line.

a = df.y.mean() - b * df.x.mean()
```

[ ]:
```
# Calculate y-hats.

df["y_hat"] = a + b * df.x
```

6.1.3 The Residuals Plot

[ ]:
```
# Compute the residuals.

df["residuals"] = df.y - df.y_hat
```

[ ]:
```
# See: analysis/scripts/dresiduals.py
dred(plt, df)
```



This residual plot looks horrible, as expected. Moreover, the Coefficient of Determination is approximately 0.46. Which means the best we could say is that 46% of the positive price movement following a dilution event could be explained by this model. That is not what we wanted. Moreover, we could say that, but ONLY for dilution events within the range of 0 to 50% increase in float size. The Correlation Coefficient is only barely strong, and the residuals plot contains clustering about the 0% x-value. The non-clustered datapoints are haphazard at best. Although the standard deviation about the least-squares line is sufficiently small relative to the standard deviation about y, it seems it would be jumping to wild conclusions to say that dilution and positive price movements were somehow significantly correlated.

To put this in context, the raw data had almost no correlation whatsoever. Removing both influential observations and outliers together in an unbiased way also resulted in extremely weak correlations. So it's almost grasping for straws in making the claim that any correlation actually exists.

Although we are able to arrive at a 67% correlation by cherry picking which data to leave out of the initial data, the weak positive correlation of approximately 20% is far more likely. Linearity is being pulled up by influential observations in the raw data. The clear clustering about the value 0% in the x-axis is more likely caused by traders trading just as business as usual.

After evaluating all of the evidence, for all intents and purposes, I am confident in saying that no significant correlation exits in terms of dilution verses postive price movements. The evidence would be flimsy at best. Furthermore, claiming that causality, from all evidence, were even plausible would just be grasping for straws. Generally, while correlation does not imply causation, causation does imply correlation. That means, absent correlation, causation is impossible. Therefore, conclusively: we can confidently sat that dilution has never caused AMC share prices to increase.

[ ]:
```python
# Calculate the Coefficient of Determination.

t = ((df.y - df.y.mean())**2).sum()

e = (df.residuals**2).sum()

r = 1 - (e / t)

print(r)
```

```
0.45952472953337486
```

[ ]:
```python
# Calculate the Standard Deviation about the Least-Squares Line.

s = df.y.std()

s_e = math.sqrt( ( e / (len(df) - 2) ) )

print(f"The standard deviation about y: {s}")
print(f"The standard deviation about the least squares line: {s_e}")
```

```
The standard deviation about y: 0.2107527457909997
The standard deviation about the least squares line: 0.15943106498849335
```

7 Final Summary

- Since Adam became CEO of AMC the company has been under performing compared to its performance history. This strongly appears to be the result of Adams capital misallocations.
- The 2023 projected DBO is $9B. That means the model predicts a net loss in earnings of approximately $-578,841,332 in 2023.
- Adams best solution to his habitual capital misallocation is to dilute his shareholders. The amount by which Adam has diluted shareholders during his tenure at AMC is not acceptable.
- The often repeated claim that dilution would cause positive price action in AMC is not supported by evidence. In fact, we can't even say a reasonable correlation exists.

7.0.1 Final Note About This Work:

the model for predicting earnings follows from testing on 2 other models not demonstrated here. Although they are available in the repository deprecation directory. The model demonstrated here out performed all other models, although those precluded models were used to back test the model used here.

Special thanks to Alexandar Holland for reviewing this work, and currating some of the data sets used in these analytics!

7.0.2 Data sources:

- investor.amctheatres.com/financial-performance/
- boxofficemojo.com/
- the-numbers.com/market/
- sharesoutstandinghistory.com/amc/
- finance.yahoo.com/quote/AMC/history?p=AMC