The Startup

Multi-Linear Regression Using Python

Rafi Atha · Follow

Published in The Startup · 9 min read · Nov 21, 2020

Hi! It's been a while since the last time I write an article here. In today's article I want to talk about how to do a multi-linear regression analysis using Python. Most of the writing in this article is directly taken from my assignment at Telkom Digital Talent Incubator 2020 a few weeks ago. You can check the notebook here and try to follow along. So without further ado, let's start!

Introduction

Regression analysis itself is a tool for building statistical models that characterize relationships among a dependent variable and one or more independent variables. **Simple Linear Regression** refers to the method used when there is only one independent variable, while **Multi-Linear Regression**

refers to the method used when there is more than one independent variable. Multi-Linear Regression can be written as below:

$$\hat{y} = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_k X_k$$

where:

$\hat{y}$: *dependent variable (predicted value)*

β_0 : *estimated intercept*

$\beta_k X_k$: *estimated slope coefficient*

In this example we will try to use multi-linear regression to analyze the relationship of a product's price, advertisement cost, and the product sales number. We will also try to predict how much products will be sold given specific product's price and advertisement cost.

Preparation

In the first code cell we will load some Python libraries we will be using, such as Pandas, NumPy, matplotlib, sklearn, etc. We will also load our dataset from my GitHub repos into a dataframe called `df_pie` by using the Pandas library.

```python
# Import libraries
## Basic libs
import pandas as pd
import numpy as np
import warnings
## Building Model
from sklearn import linear_model
from scipy import stats
import statsmodels
import statsmodels.api as sm
import statsmodels.formula.api as smf
import statsmodels.stats.api as sms
from statsmodels.compat import lzip
## Data Visualization
import seaborn as sns
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

warnings.filterwarnings('ignore')
plt.rcParams['figure.figsize'] = (7, 7)
plt.style.use('ggplot')

# Load dataset
df_pie = pd.read_csv('https://raw.githubusercontent.com/rafiag/DTI-Linear-Regression/main/data/pi
                     index_col='week', sep=';')

print(df_pie.shape)
df_pie
```

Multi-Linear Regression_1.py hosted with ❤ by GitHub view raw

week	pie_sales	price	advertising
1	350	5.5	3.3
2	460	7.5	3.3
3	350	8.0	3.0
4	430	8.0	4.5
5	350	6.8	3.0

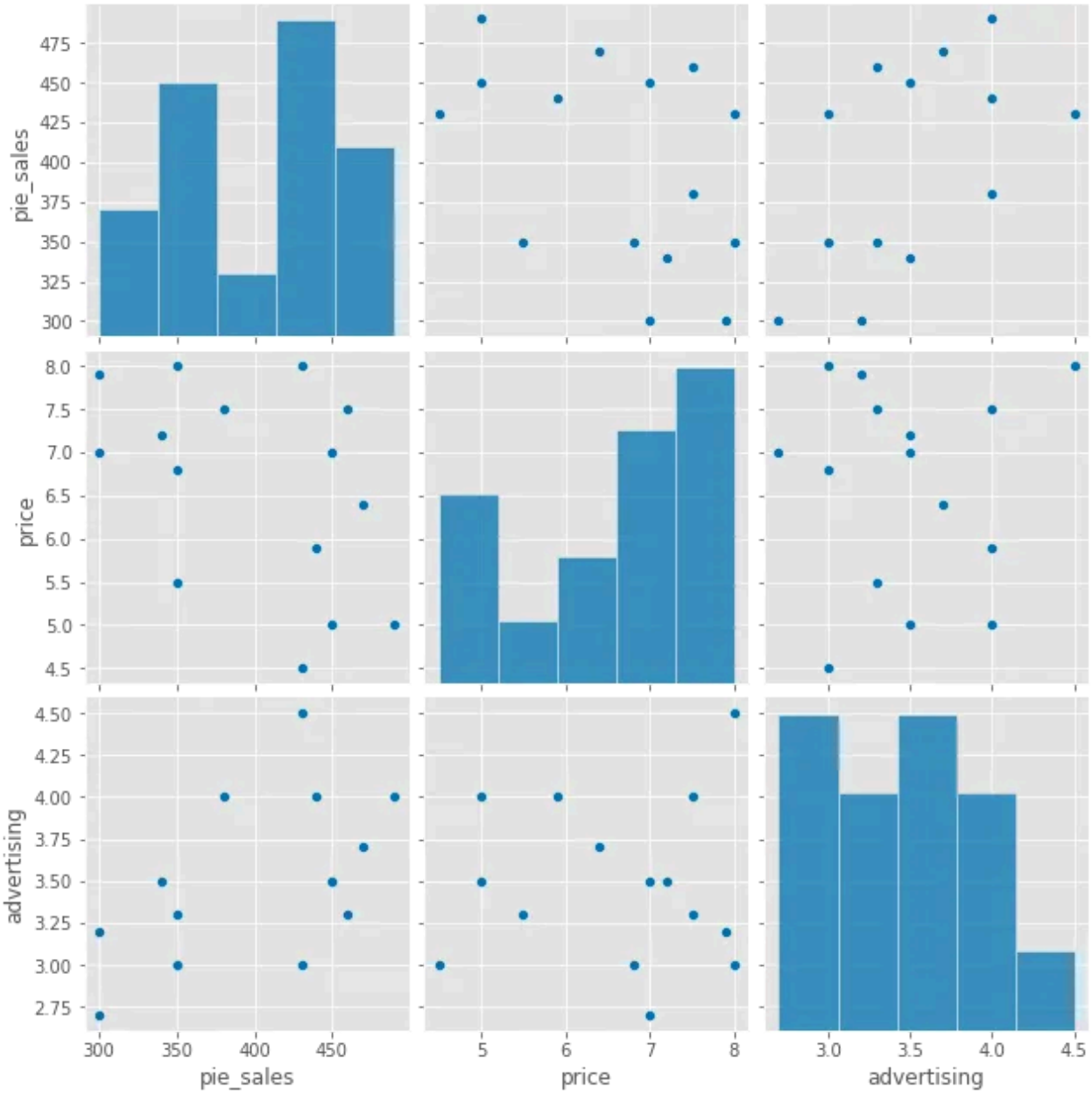
As seen above our dataset consist of 3 columns (pie_sales, price, and advertising) and 15 rows. We will try to predict how much pie will be sold depending on its price and advertisement cost.

Descriptive Analysis

Before going deeper into using multi-linear regression, it's always a good idea to simply visualize our data to understand it better and see if there are any relationship between each variable. To do this we will use the `pairplot()` function from the Seaborn library. The function will output a figure containing histogram and scatter plot between each variable.

```
# Visualize the data using scatter plot and histogram
sns.set_palette('colorblind')
sns.pairplot(data=df_pie, height=3)
```

Multi-Linear Regression_2.py hosted with ❤ by GitHub view raw



Looking at first row in the figures we can see that there might be relations between price, advertising, and pie_sales. The scatter plot between pie sales and price display pattern of negative relation, which means the higher the price the lower the sales will be. In the other hand the scatter plot between advertising and pie sales display a positive relation, the more money we spent on advertising the more pie we will sells.

Building Regression Model

Since we already see that there might be relations between our independent and dependent variables, let's continue to building our regression model. We will use the `LinearRegression()` function from the sklearn library to build our models.

```
# Set independent and dependent variables
X = df_pie[['price', 'advertising']]
y = df_pie['pie_sales']

# Initialize model from sklearn and fit it into our data
regr = linear_model.LinearRegression()
model = regr.fit(X, y)

print('Intercept:', model.intercept_)
print('Coefficients:', model.coef_)
```

Multi-Linear Regression_3.py hosted with ❤ by GitHub view raw

```
Intercept: 306.5261932837436
Coefficients: [-24.97508952  74.13095749]
```

The code above printed few important values from our model. Those values are the intercept and coefficients values of the models which can be put in mathematic equation as below:

$$\hat{y} = 306.5261 - 24.975X_1 + 74.1309X_2$$

Let's breakdown what each of those number means:

- The intercept value is the estimated average value of our dependent variable when all of our independent variables values is 0. In our case this means that in the case we sell our pie at price of 0 and spent advertising expense of 0 we will sell about 306 pies.
- For the coefficients we have 2 values for the price and advertising variables respectively. This value represents the relation of our independent variable to the dependent variable, where a change of exactly 1 at the independent variable will change the value of our dependent variable the same amount as the coefficient. For example, if we increase our advertising expense by 10, we will also increase our sales by about 741 pies (74.1309 * 10).

Now, let's try to predict our pie sales by inputing our own data below...

```
# Values to predict
price = input('What is the price of the pie? \n')
advertising = input('How much money are you going to spend for advertising? \n')

try:
    print('We predict {:.0f} pies will be sold if we sold the pie at ${} and spend ${} at adverti
        model.predict([[float(price), float(advertising)]])[0],
        price,
        advertising))
except ValueError:
    print('Please only input either:\n- whole number e.g. 1, 4, 7\n- decimal/float number e.g. 3.
```

Multi-Linear Regression_4.py hosted with ❤ by GitHub view raw

```
What is the price of the pie?
3.4

How much money are you going to spend for advertising?
5
```

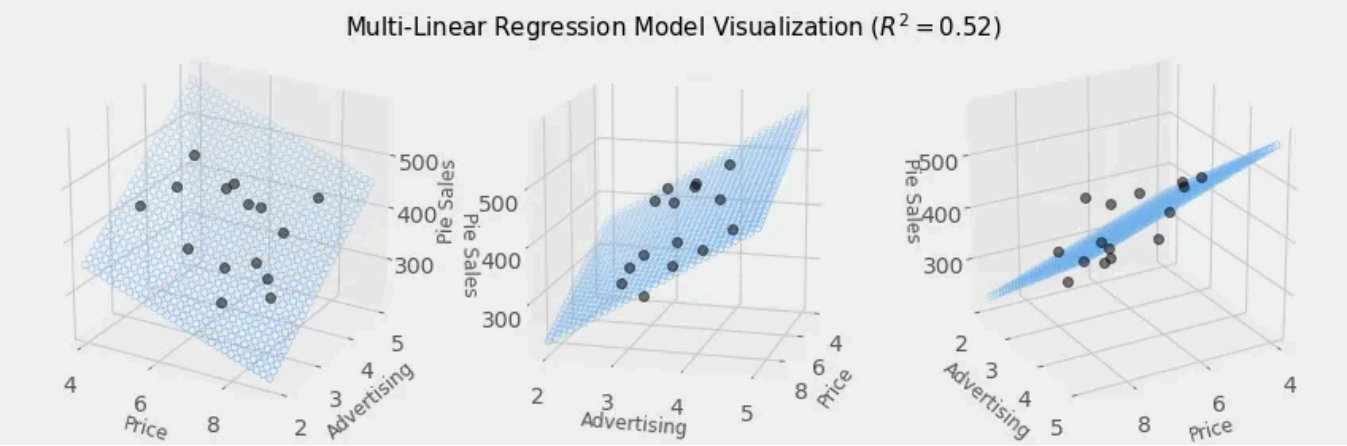
```
We predict 592 pies will be sold if we sold the pie at $3.4 and spend
$5 at advertising.
```

Before going into the next step we will try to visualize our model into 3D graph with the code cell below. We will draw the linear model as a blue plane and we will plot our data point in the graph as grey dot.

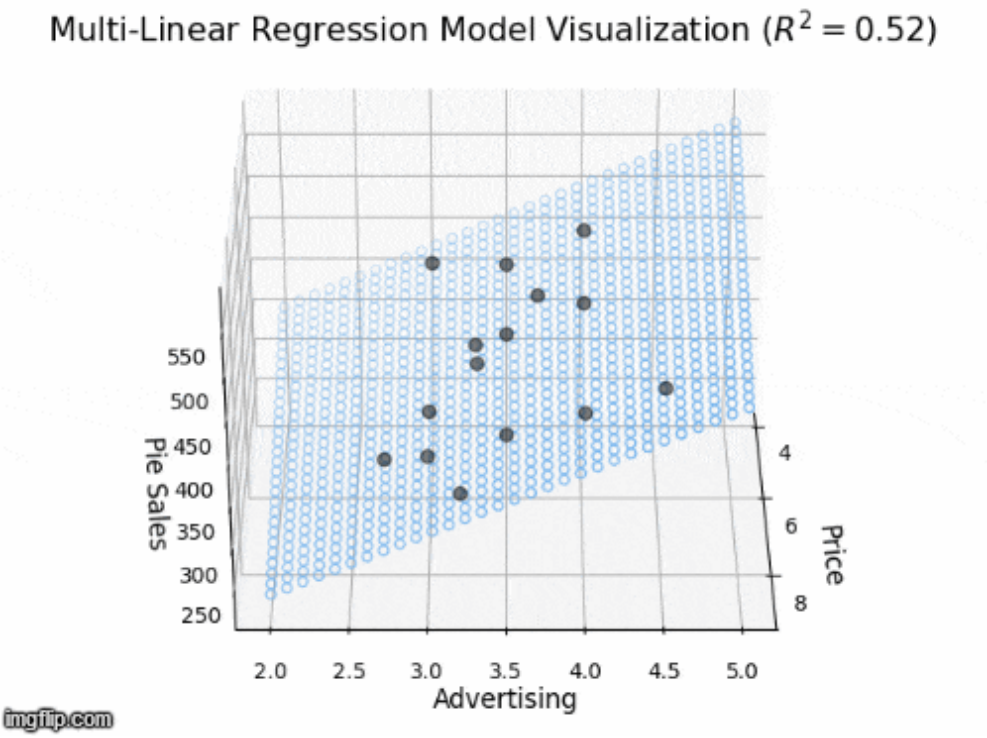
```
# Prepare data
X = df_pie[['price', 'advertising']].values.reshape(-1,2)
Y = df_pie['pie_sales']

# Create range for each dimension
x = X[:, 0]
y = X[:, 1]
z = Y

xx_pred = np.linspace(4, 9, 30)  # range of price values
yy_pred = np.linspace(2, 5, 30)  # range of advertising values
xx_pred, yy_pred = np.meshgrid(xx_pred, yy_pred)
model_viz = np.array([xx_pred.flatten(), yy_pred.flatten()]).T

# Predict using model built on previous step
# ols = linear_model.LinearRegression()
# model = ols.fit(X, Y)
predicted = model.predict(model_viz)

# Evaluate model by using it's R^2 score
r2 = model.score(X, Y)

# Plot model visualization
plt.style.use('fivethirtyeight')

fig = plt.figure(figsize=(12, 4))

ax1 = fig.add_subplot(131, projection='3d')
ax2 = fig.add_subplot(132, projection='3d')
ax3 = fig.add_subplot(133, projection='3d')

axes = [ax1, ax2, ax3]

for ax in axes:
    ax.plot(x, y, z, color='k', zorder=15, linestyle='none', marker='o', alpha=0.5)
    ax.scatter(xx_pred.flatten(), yy_pred.flatten(), predicted, facecolor=(0,0,0,0), s=20, edgecc
    ax.set_xlabel('Price', fontsize=12)
    ax.set_ylabel('Advertising', fontsize=12)
    ax.set_zlabel('Pie Sales', fontsize=12)
    ax.locator_params(nbins=4, axis='x')
    ax.locator_params(nbins=5, axis='x')

ax1.view_init(elev=25, azim=-60)
ax2.view_init(elev=15, azim=15)
ax3.view_init(elev=25, azim=60)
```

```
46
47  fig.suptitle('Multi-Linear Regression Model Visualization ($R^2 = %.2f$)' % r2, fontsize=15, colo
48
49  fig.tight_layout()
```

Multi-Linear Regression_5.py hosted with ❤ by GitHub view raw



Here is the full 360° view of the model visualization:



Click here if the image failed to load.

Model Validation

After building the model it is important for us to validate it's performance. We can evaluate a model by looking at it's coefficient of determination (R^2), F-test, t-test, and also residuals. Before we continue we will rebuild our model using the statsmodel library with the `OLS()` function. Then we will print the model summary using the `summary()` function on the model. The model summary contains lots of important value we can use to evaluate our model.

```
X = df_pie[['price', 'advertising']]
X = sm.add_constant(X) # adding a constant

olsmod = sm.OLS(df_pie['pie_sales'], X).fit()
print(olsmod.summary())
```

Multi-Linear Regression_6.py hosted with ❤ by GitHub view raw

```
                            OLS Regression Results
==============================================================================
Dep. Variable:              pie_sales   R-squared:                       0.521
Model:                            OLS   Adj. R-squared:                  0.442
Method:                 Least Squares   F-statistic:                     6.539
Date:                Thu, 19 Nov 2020   Prob (F-statistic):            0.0120
Time:                        12:33:06   Log-Likelihood:                -77.510
No. Observations:                  15   AIC:                             161.0
Df Residuals:                      12   BIC:                             163.1
Df Model:                           2
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const        306.5262    114.254      2.683      0.020      57.588     555.464
price        -24.9751     10.832     -2.306      0.040     -48.576      -1.374
advertising   74.1310     25.967      2.855      0.014      17.553     130.709
==============================================================================
Omnibus:                        1.505   Durbin-Watson:                   1.683
Prob(Omnibus):                  0.471   Jarque-Bera (JB):                0.937
Skew:                           0.595   Prob(JB):                        0.626
Kurtosis:                       2.709   Cond. No.                         72.2
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

Coefficient of Determination (R^2)

Coefficient of determination is the portion of the total variation in the dependent variable that is explained by variation in the independent variable. R^2 scores are calculated as below:

$$R^2 = \frac{\sum(\hat{Y}_i - \bar{Y})^2}{\sum(Y_i - \bar{Y})^2}$$

In statsmodel we can obtain the R^2 value of our model by accesing the `.rsquared` attribute of the our model.

```
1   print('R2 score:', olsmod.rsquared)
```

Multi-Linear Regression_7.py hosted with ❤ by GitHub view raw

```
R2 score: 0.5214779360292285
```

R^2 range between 0 and 1, where R^2=0 means there are no linear relationship between the variables and R^2=1 shows a perfect linear relationship. In our case, we got R^2 score about 0.5214 which means 52.14% of our dependent variable can be explained using our independent variables.

F-Test (ANOVA)

F-test or ANOVA (Analysis of variance) in multi-linear regression can be used to determine whether our complex model perform better than a simpler model (e.g. model with only one independent variable). With F-test we can evaluate the significance of our model by calculating the probability of observing an F-statistic that is at least as high as the value that our model obtained. Similar to R^2 score, we can easily get the F-statistic and probability

of said F-statistic by accessing the `.fvalues` and `.f_pvalues` attribute of our model as below.

```
print('F-statistic:', olsmod.fvalue)
print('Probability of observing value at least as high as F-statistic:', olsmod.f_pvalue)
```

Multi-Linear Regression_8.py hosted with ❤ by GitHub view raw

```
F-statistic: 6.538606789020464
Probability of observing value at least as high as F-statistic:
0.01200637223318641
```

Because our f_pvalue is lower than 0.05 **we can conclude that our model performs better than other simpler model.**

T-test

The t-statistic is the coefficient divided by its standard error. The standard error is an estimate of the standard deviation of the coefficient, the amount it varies across cases. It can be thought of as a measure of the precision with which the regression coefficient is measured. Same as the F-test, the p-value show the probability of seeing a result as extreme as the one our model have. We can also get the p-value for all of our variables by calling the `.pvalues` attribute on the model.

```
print(olsmod.pvalues)
```

Multi-Linear Regression_9.py hosted with ❤ by GitHub view raw

```
const          0.019932
price          0.039788
```

```
advertising    0.014494
dtype: float64
```

Both of our independent variables, price and advertising, have p-value less than 0.05 which shows that **there is sufficient evidence that price and advertising affects our pie sales.**

Assumption Testing

Next, we will validate our model by doing residual analysis, below are the list of test or assumption we will do to check on our model validity:

- Linearity
- Normality
- Multicollinearity
- Autocorrelation
- Homoscedasticity

Residual is the difference between the observed value and predicted value from our dataset. With statsmodel we can easily get the residual value of our model by simply accesing the `.resid` attribute of the model and then we can keep it in a new column called 'residual' in our `df_pie` dataframe.

```
1  df_pie['pie_sales_pred'] = olsmod.predict(X)
2  df_pie['residual'] = olsmod.resid
3  df_pie.head()
```

Multi-Linear Regression_10.py hosted with ❤ by GitHub view raw

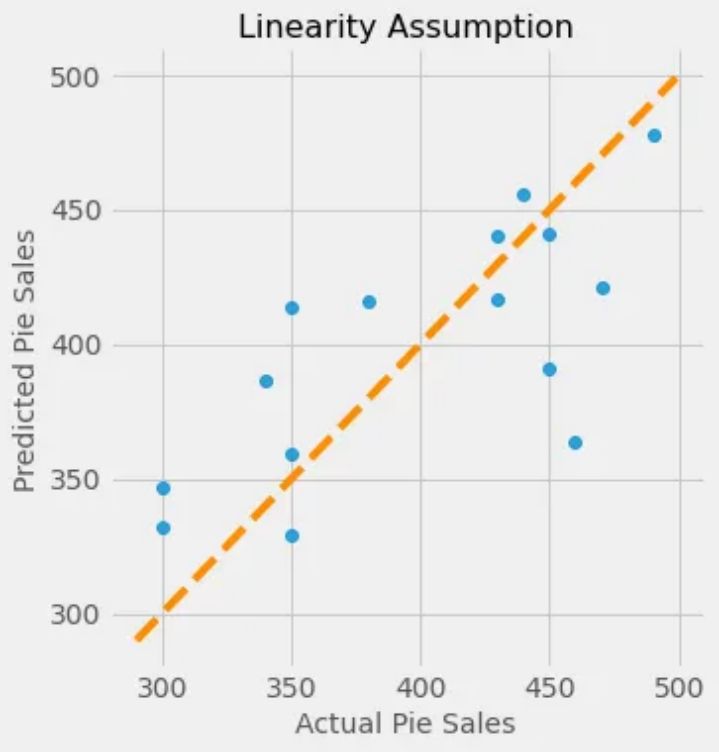
week	pie_sales	price	advertising	pie_sales_pred	residual
1	350	5.5	3.3	413.795361	-63.795361
2	460	7.5	3.3	363.845182	96.154818
3	350	8.0	3.0	329.118350	20.881650
4	430	8.0	4.5	440.314786	-10.314786
5	350	6.8	3.0	359.088457	-9.088457

Linearity

This assumes that there is a linear relationship between the independent variables and the dependent variable. In our case since we have multiple independent variables, we can do this by using a scatter plot to see our predicted values versus the actual values.

```
# Plotting the observed vs predicted values
sns.lmplot(x='pie_sales', y='pie_sales_pred', data=df_pie, fit_reg=False, size=5)

# Plotting the diagonal line
line_coords = np.arange(df_pie[['pie_sales', 'pie_sales_pred']].min().min()-10,
                        df_pie[['pie_sales', 'pie_sales_pred']].max().max()+10)
plt.plot(line_coords, line_coords,  # X and y points
         color='darkorange', linestyle='--')

plt.ylabel('Predicted Pie Sales', fontsize=14)
plt.xlabel('Actual Pie Sales', fontsize=14)
plt.title('Linearity Assumption', fontsize=16)
plt.show()
```

Multi-Linear Regression_11.py hosted with ❤ by GitHub view raw



The scatter plots show residual point evenly spread around the diagonal line, so we can assume that **there is linear relationship between our independent and dependent variables.**

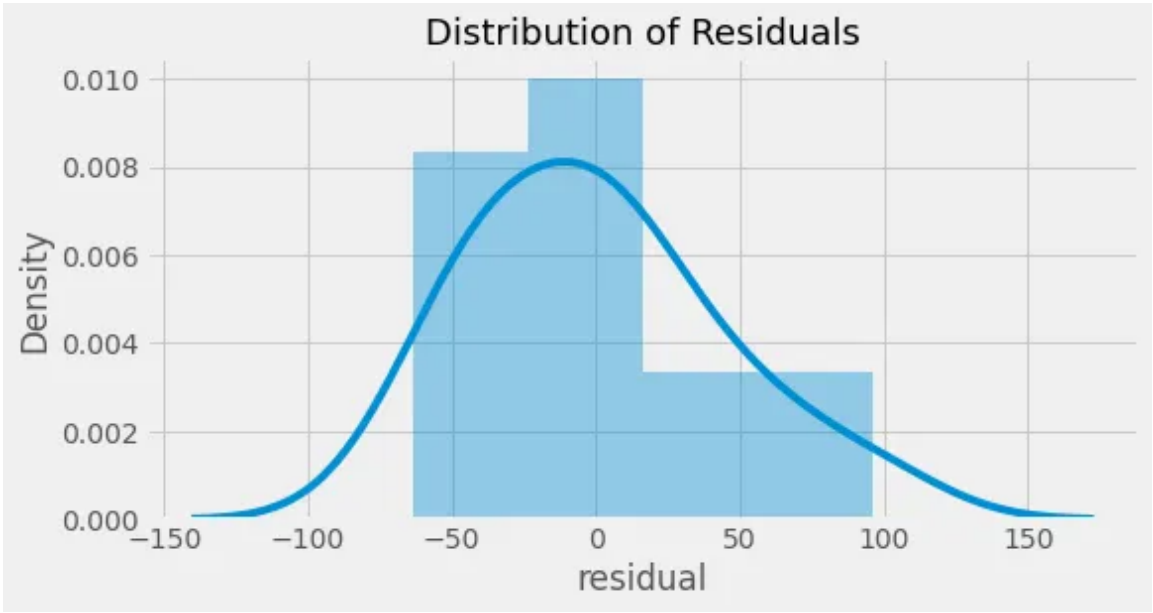
Normality

This assumes that the error terms of the model are normally distributed. We will examine the normality of the residuals by plotting it into histogram and looking at the p-value from the Anderson-Darling test for normality. We will use the `normal_ad()` function from statsmodel to calculate our p-value and then compare it to threshold of 0.05, if the p-value we get is higher than the threshold then we can assume that our residual is normally distributed.

```python
from statsmodels.stats.diagnostic import normal_ad

# Performing the test on the residuals
p_value = normal_ad(df_pie['residual'])[1]
print('p-value from the test Anderson-Darling test below 0.05 generally means non-normal:', p_val

# Plotting the residuals distribution
plt.subplots(figsize=(8, 4))
plt.title('Distribution of Residuals', fontsize=18)
sns.distplot(df_pie['residual'])
plt.show()

# Reporting the normality of the residuals
if p_value < 0.05:
    print('Residuals are not normally distributed')
else:
    print('Residuals are normally distributed')
```

Multi-Linear Regression_12.py hosted with ❤ by GitHub view raw

```
p-value from the test Anderson-Darling test below 0.05 generally
means non-normal: 0.6655438857701688
```



```
Residuals are normally distributed
```

From the code above we got our p-value of 0.6644 which can be considered normal because it's above the 0.05 threshold. The histogram plot also show a normal distribution (despite it might be looking a little skewed because we only have 15 observation in our dataset). **From both of those result we can assume that our residual are normally distributed.**

Multicollinearity

This assumes that the predictors used in the regression are not correlated with each other. To identify if there are any correlation between our predictors we can calculate the Pearson correlation coefficient between each column in our data using the `corr()` function from Pandas dataframe. Then we can display it as a heatmap using `heatmap()` function from Seaborn.

```
corr = df_pie[['pie_sales', 'price', 'advertising']].corr()
print('Pearson correlation coefficient matrix of each variables:\n', corr)

# Generate a mask for the diagonal cell
mask = np.zeros_like(corr, dtype=np.bool)
np.fill_diagonal(mask, val=True)

# Initialize matplotlib figure
fig, ax = plt.subplots(figsize=(4, 3))

# Generate a custom diverging colormap
cmap = sns.diverging_palette(220, 10, as_cmap=True, sep=100)
cmap.set_bad('grey')

# Draw the heatmap with the mask and correct aspect ratio
sns.heatmap(corr, mask=mask, cmap=cmap, vmin=-1, vmax=1, center=0, linewidths=.5)
fig.suptitle('Pearson correlation coefficient matrix', fontsize=14)
ax.tick_params(axis='both', which='major', labelsize=10)
# fig.tight_layout()
```

Multi-Linear Regression_13.py hosted with ❤ by GitHub view raw

```
Pearson correlation coefficient matrix of each variables:
              pie_sales     price  advertising
pie_sales      1.000000 -0.443273     0.556320
price         -0.443273  1.000000     0.030438
advertising    0.556320  0.030438     1.000000
```

The image shows that there are some positive relationship between advertising and pie_sales and a negative relationship between price and pie_sales. Both of this result support our resulting model from before. Most importanly, notice how the price and advertising have almost 0 correlation coefficient. **This means both of our independent variable are not affecting each other and that there is no multicollinearity in our data.**

Autocorrelation

Autocorrelation is correlation of the errors (residuals) over time. Used when data are collected over time to detect if autocorrelation is present. Autocorrelation exists if residuals in one time period are related to residuals in another period. We can detect autocorrelation by performing Durbin-Watson test to determine if either positive or negative correlation is present. In this step we will use the `durbin_watson ()` function from statsmodel to calculate our Durbin-Watson score and then assess the value with the following condition:

- If the Durbin-Watson score is less than 1.5 then there is a positive autocorrelation and the assumption is not satisfied
- If the Durbin-Watson score is between 1.5 and 2.5 then there is no autocorrelation and the assumption is satisfied
- If the Durbin-Watson score is more than 2.5 then there is a negative autocorrelation and the assumption is not satisfied

```
from statsmodels.stats.stattools import durbin_watson

durbinWatson = durbin_watson(df_pie['residual'])

print('Durbin-Watson:', durbinWatson)
if durbinWatson < 1.5:
    print('Signs of positive autocorrelation', '\n')
    print('Assumption not satisfied')
elif durbinWatson > 2.5:
    print('Signs of negative autocorrelation', '\n')
    print('Assumption not satisfied')
else:
    print('Little to no autocorrelation', '\n')
    print('Assumption satisfied')
```

Multi-Linear Regression_14.py hosted with ❤ by GitHub view raw

```
Durbin-Watson: 1.6831203020921253
Little to no autocorrelation

Assumption satisfied
```

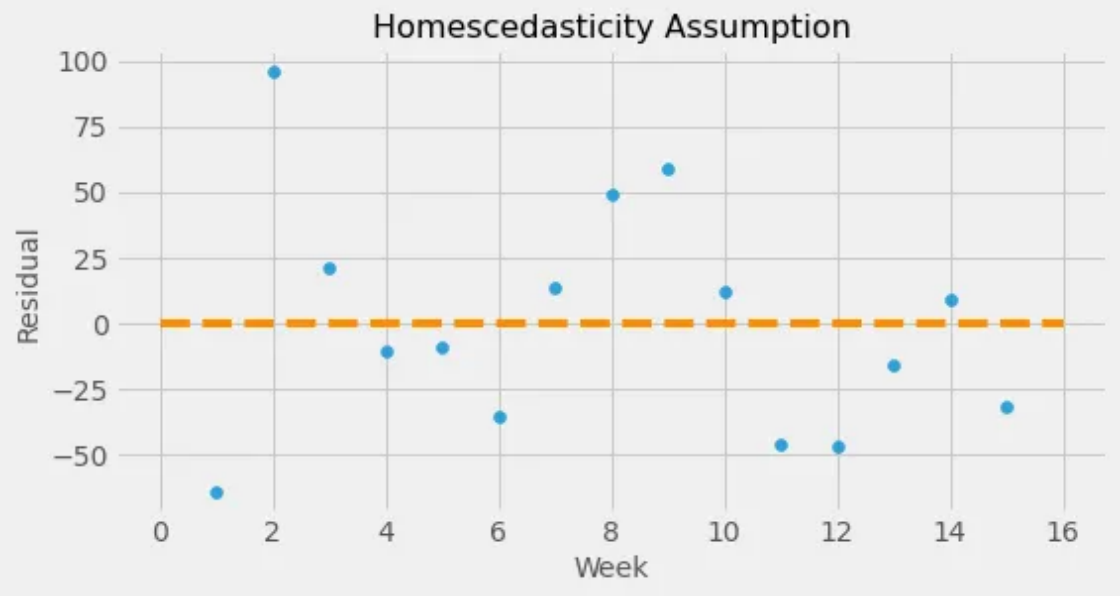
Our model got a Durbin-Watson score of about 1.6831 which is between 1.5 and 2.5, **so we can assume that there is no autocorrelation in our residual.**

Homoscedasticity

This assumes homoscedasticity, which is the same variance within our error terms. Heteroscedasticity, the violation of homoscedasticity, occurs when we don't have an even variance across the error terms. To detect homoscedasticity, we can plot our residual and see if the variance appears to be uniform.

```
# Plotting the residuals
plt.subplots(figsize=(8, 4))
plt.scatter(x=df_pie.index, y=df_pie.residual, alpha=0.8)
plt.plot(np.repeat(0, len(df_pie.index)+2), color='darkorange', linestyle='--')

plt.ylabel('Residual', fontsize=14)
plt.xlabel('Week', fontsize=14)
plt.title('Homescedasticity Assumption', fontsize=16)
plt.show()
```

Multi-Linear Regression_15.py hosted with ❤ by GitHub view raw



Despite having only 15 data points our residual seems to have constant and uniform variance, so **we can assume that it satisfied the homoscedasticity assumption.**

Conclusion

Our models succesfuly passed all the tests in the model validation steps, so we can conclude that our model can perform well to predict future pie sales by using the two independent variables, price and advertising. But still, our model only has R^2 score of 52.14%, which means that there is still about 48% unknown factors that are affecting our pie sales.

References

1. Telkom Digital Talent Incubator — Data Scientist Module 4 (Regression)
2. Multiple Linear Regression and Visualization in Python
3. Testing Linear Regression Assumptions in Python

Python | Data Science | Statistics | Data Visualization | Linear Regression

Data enthusiast. I use Python and R (mostly Python) to do stuff with data.

Responses (4)

Write a response

What are your thoughts?

TheNewSheppard
Aug 24, 2022

Big plus for assumptions testing at the end ! Great article

Reply

Justin B.
Aug 14, 2022

This is literally fantastic. Thank you for the excellent, excellent info!

Reply

Bhogayatachandrakant
Feb 26, 2022

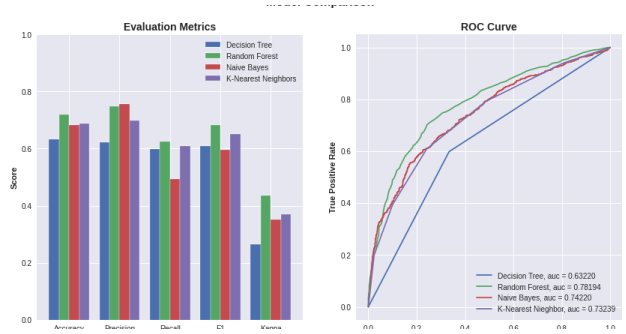
Hi, Rafi!

Extra-ordinarily written your article is highly relevant and concretely useful. Thanks a lot!

Reply

See all responses

More from Rafi Atha and The Startup



In Analytics Vidhya by Rafi Atha

Building Classification Model with Python

Guide on how-to solve classification problem using Python. This article covers the basic...

Jan 29, 2021 60 1

In The Startup by Divad Sanders

4 Boring Startup Ideas Screaming to Be Built (and How to Build Them)

Build the unsexy thing. Charge for outcomes. Repeat.

Apr 7 3.3K 94

In The Startup by Jano le Roux

The Toothbrush That Cured My Bad Breath — And Made Me Believe in...

I finally discovered the Apple of toothbrushes.

Apr 1 4.7K 115

Rafi Atha

Data Analytics in Baseball — As told in Moneyball (2011)

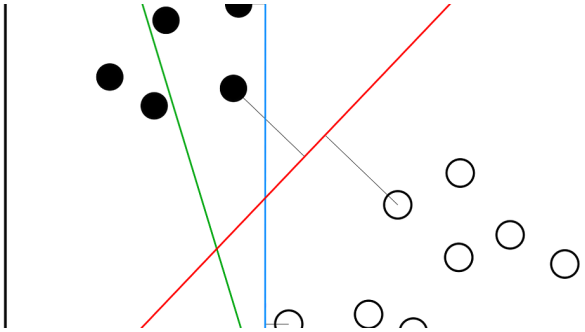
How data and statistics plays part in writing a new MLB record.

Sep 21, 2020

See all from Rafi Atha

See all from The Startup

Recommended from Medium



In Python in Plain English by Kiran Maan

Just Stop Writing Python Functions Like This!!!

I just reviewed someone else's code and I was just shocked.

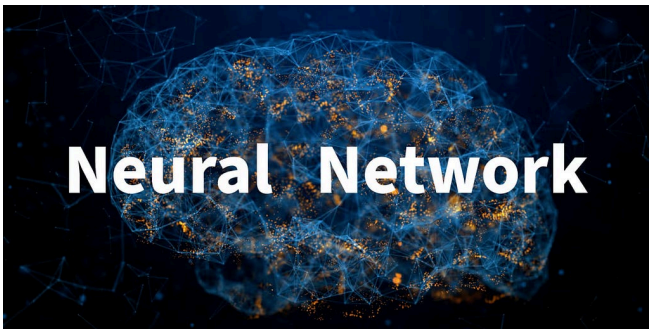
Jan 19 2.9K 72

In Artificial Intelligence in Plain En... by Ritesh Gu...

Data Science All Algorithm Cheatsheet 2025

Stories, strategies, and secrets to choosing the perfect algorithm.

Jan 5 1.6K 42



Garvit Sapra

Understanding Deep learning Neural Networks: Key Concepts and...

Neural Networks: The Building Blocks of Modern AI

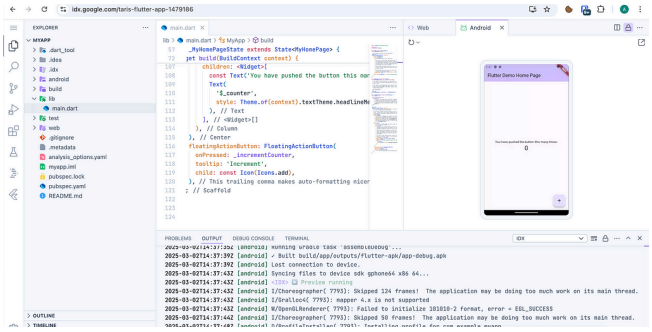
Dec 22, 2024 16

In Pythonic AF by Aysha R

I Tried Writing Python in VS Code and PyCharm — Here's What I Found

One felt like a smart coding companion. The other felt like assembling IKEA furniture...

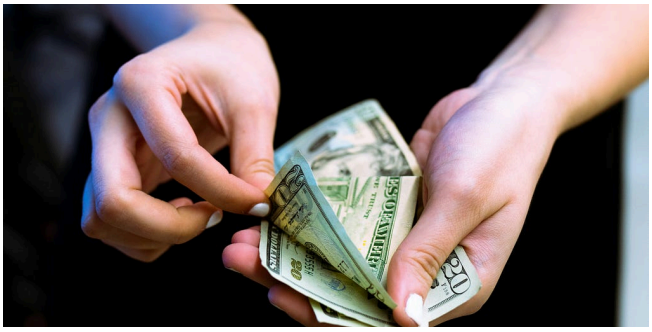
Apr 15 1.1K 92



In Coding Beauty by Tari Ibaba

This new IDE from Google is an absolute game changer

This new IDE from Google is seriously revolutionary.



In Learn AI for Profit by Nipuna Maduranga

You Can Make Money With AI Without Quitting Your Job

I'm doing it, 2 hours a day

Mar 11 4.9K 282 Mar 24 7.7K 362

See more recommendations