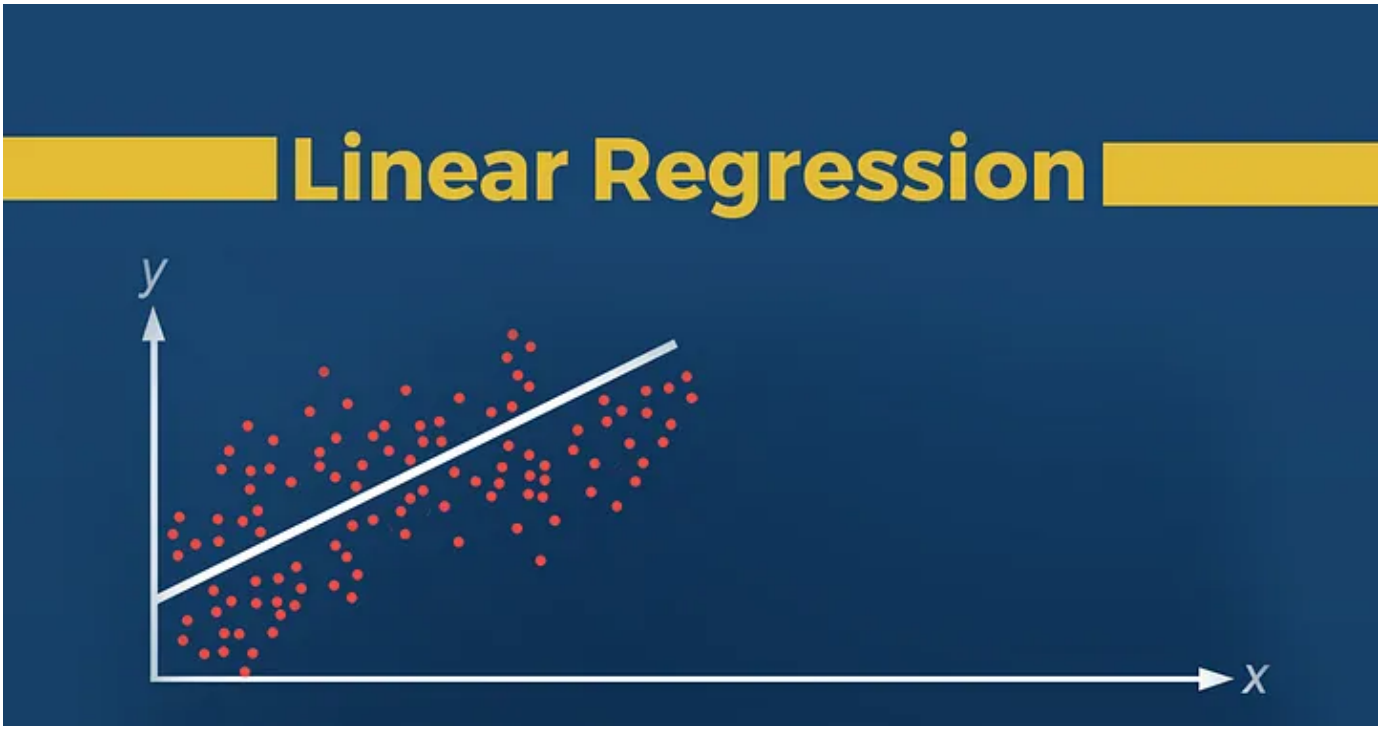
Mastering Linear Regression with Statsmodels

Luís Fernando Torres · Follow

15 min read · Just now



Source: https://www.digitalvidya.com/blog/linear-regression/

Note: This article is based on my Kaggle Notebook: 📒📈 Mastering Linear Regression with Statsmodels

Introduction

Linear Regression is one of the most essential techniques used in Data Science and Machine Learning to **predict** the value of a certain variable based on the value of another variable.

The goal of this article is to explore in-depth how Linear Regression works and the math behind it, explaining the relationship between dependent variable and independent variables and the main **conditions** for a successful Linear Regression.

For this notebook, we are going to use Sklearn's **diabetes dataset**, which is particularly useful for regression tasks.

We start by loading all the necessary libraries.

```
# Basic libraries
import pandas as pd
import numpy as np

# Data Visualization
import plotly.express as px
import plotly.graph_objs as go
import plotly.subplots as sp
from plotly.subplots import make_subplots
import plotly.figure_factory as ff
import plotly.io as pio
from IPython.display import display
from plotly.offline import init_notebook_mode
init_notebook_mode(connected=True)

# Diabetes dataset
from sklearn.datasets import load_diabetes

# For splitting the dataset into training and testing sets
from sklearn.model_selection import train_test_split

# Metric for evaluation
```

```
from sklearn.metrics import mean_squared_error

# Statsmodels for Linear Regression
import statsmodels.api as sm

# Hiding warnings
import warnings
warnings.filterwarnings("ignore")
```

What is Linear Regression?

Linear Regression is a statistical method widely used in various fields, including economics, biology, engineering, and many more for **predictive modeling** and **hypothesis testing**.

It is based on modeling the **relationship** between and dependent variable and one or more independent variable by fitting a linear equation to the observed data. The primary objective of a linear regression is finding the **best-fitting line** that can accurately predict the output of the dependent variable given the values of the independent variable(s).

The line is given by the following equation:

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_3 + \ldots + \epsilon$$

Where:

Y is the dependent variable.

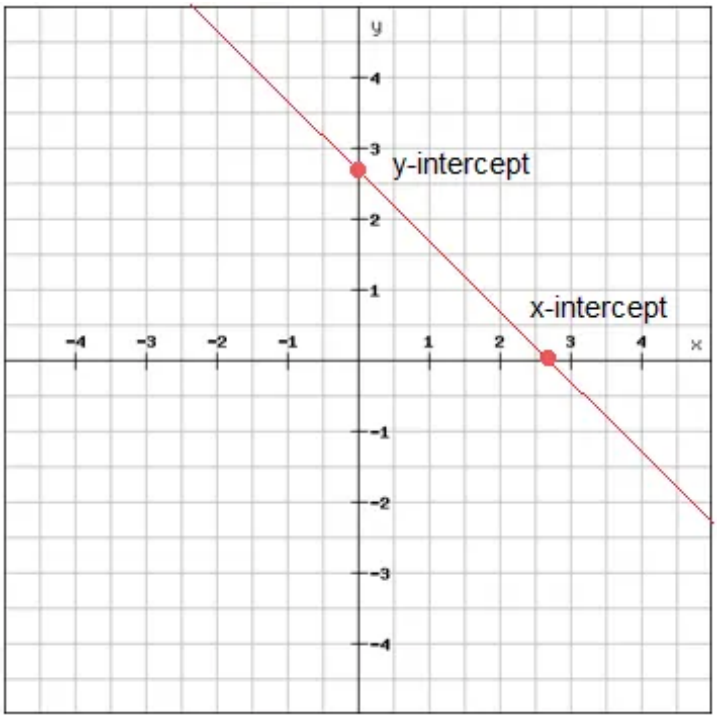
β_0 is the intercept.

β_1, β_2, $\beta 3_3$, ... are the coefficients for each independent variable.

X_1, X_2, X_3, ... represent each one independent variable.

ϵ is the error term, which captures the **random noise** in the dependent variable. It is the **stochastic** component.

In this equation, each component serves a specific purpose. The *Intercept*(β_0), for instance, is the **constant term** that represents the value of Y (the dependent variable) when all the independent variables X_1, X_2, ... are equal to **zero**. It locates the line up or down the y-axis on the **Cartesian coordinate system**, as you can see in the image below.



When **X = 0**, **Y = β_0**

In the case of **one single** independent variable, the intercept is calculated as the following:

$$\beta_0 = \bar{Y} - \beta\bar{X}$$

Where:

$\bar{Y}$ is the **mean** value of the dependent variable Y.

$\bar{X}$ is the **mean** value of the independent variable X.

The *Coefficients*(β_1, β_2, $\beta 3_3$, ...) are the values that measure the **magnitude of impact** of each independent variable on the dependent variable. A one-unit change in X_1, for instance, will result in a β_1 change in Y if all the other X_s are **constant.**

The equation for finding the value of β_1 in a simple linear regression model with only **one single** independent variable is given by the following:

$$\beta_1 = \frac{\sum_{i=1}^{n}(X_i - \bar{X})(Y_i - \bar{Y})}{\sum_{i=1}^{n}(X_i - \bar{X})^2}$$

Where:

X_i and Y_i represent the i^{th} sample of both X and Y.

$\bar{X}$ and $\bar{Y}$ represent the mean values for both X and Y.

$\sum_{i=1}^{n}$ represent the sum starting from the first element of the series up until the n^{th} element of the series.

In this equation, the coefficient β_1 is the result of the ratio of the **covariance** between X and Y, represented by the product of the deviations between each X and Y from their respective means, and the variance of X, given by the sum of the squares of the deviations of each X from its mean. The effect of the independent variable on the dependent variable can be summed up as the following:

• $\beta_1 > 0$: Y increases as X increases.

• $\beta_1 < 0$: Y decreases as X increases.

• $\beta_1 = 0$: X has no effect on Y.

In the case of a linear regression with **multiple** independent variables, the coefficients and the intercept are obtained through **matrix algebra**:

$$\beta = (X^T X)^{-1} X^T Y$$

Where:

β is a **vector** containing the intercept and coefficients.

X is a **matrix** where the first column is all 1_s and the other columns each represent a different independent variable.

X^T is the **transpose** of the X matrix.

$(X^T X)^{-1}$ is the **inverse** of the product of the X matrix and its transpose.

Y is a **vector** of values of the dependent variable.

The goal of this operation is to obtain estimates of the parameters in the β vector, which are the ***coefficients*** and the ***intercept***.

The linear regression with multiple independent variables can be then given by the following equation:

$$Y = X\beta + \epsilon$$

The image below can correctly display the matrix form of the equation above.

$$\begin{bmatrix} Y_1 \\ Y_2 \\ \vdots \\ \vdots \\ Y_n \end{bmatrix}_{n\times 1} = \begin{bmatrix} 1 & X_{11} & X_{21} & \dots & X_{k1} \\ 1 & X_{12} & X_{22} & \dots & X_{k2} \\ \vdots & \vdots & \vdots & \dots & \vdots \\ \vdots & \vdots & \vdots & \dots & \vdots \\ 1 & X_{1n} & X_{2n} & \dots & X_{kn} \end{bmatrix}_{n\times k} \begin{bmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \vdots \\ \beta_n \end{bmatrix}_{k\times 1} + \begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ \vdots \\ \vdots \\ \epsilon_n \end{bmatrix}_{n\times 1}$$

The first column of the **X** matrix is populated by **1**s for the intercept, which is represented by the **β1** in the **β** vector.

To demonstrate how Linear Regression works, let's load the diabetes dataset and obtain our X and Y values.

```
# Loading dataset
X, y = load_diabetes(return_X_y=True)
```

```
# Creating a DataFrame for the Independent variables
X_df = pd.DataFrame(X, columns = [f'Feature_{i+1}' for i in range(X.shape[1])])
X_df
```

	Feature_1	Feature_2	Feature_3	Feature_4	Feature_5	Feature_6	Feature_7	Feature_8	Feature_9	Feature_10
0	0.038076	0.050680	0.061696	0.021872	-0.044223	-0.034821	-0.043401	-0.002592	0.019907	-0.017646
1	-0.001882	-0.044642	-0.051474	-0.026328	-0.008449	-0.019163	0.074412	-0.039493	-0.068332	-0.092204
2	0.085299	0.050680	0.044451	-0.005670	-0.045599	-0.034194	-0.032356	-0.002592	0.002861	-0.025930
3	-0.089063	-0.044642	-0.011595	-0.036656	0.012191	0.024991	-0.036038	0.034309	0.022688	-0.009362
4	0.005383	-0.044642	-0.036385	0.021872	0.003935	0.015596	0.008142	-0.002592	-0.031988	-0.046641
...	...	...	...	...	...	...	...	...	...	...
437	0.041708	0.050680	0.019662	0.059744	-0.005697	-0.002566	-0.028674	-0.002592	0.031193	0.007207
438	-0.005515	0.050680	-0.015906	-0.067642	0.049341	0.079165	-0.028674	0.034309	-0.018114	0.044485
439	0.041708	0.050680	-0.015906	0.017293	-0.037344	-0.013840	-0.024993	-0.011080	-0.046883	0.015491
440	-0.045472	-0.044642	0.039062	0.001215	0.016318	0.015283	-0.028674	0.026560	0.044529	-0.025930
441	-0.045472	-0.044642	-0.073030	-0.081413	0.083740	0.027809	0.173816	-0.039493	-0.004222	0.003064

442 rows × 10 columns

Features from 1 to 10 are our independent variables X_1, X_2, X_3, ..., X_{10}.

```
# Creating a DataFrame for the Dependent variable
y_df = pd.DataFrame(y, columns = ['Target'])
y_df
```

	Target
0	151.0
1	75.0
2	141.0
3	206.0
4	135.0
...	...
437	178.0
438	104.0
439	132.0
440	220.0
441	57.0

442 rows × 1 columns

We have above our dependent variable, also called the target variable.

We can also combine both dataframes.

```
# Concatenating both X and Y dataframes into one single dataframe
df = pd.concat([X_df, y_df], axis = 1)
df
```

	Feature_1	Feature_2	Feature_3	Feature_4	Feature_5	Feature_6	Feature_7	Feature_8	Feature_9	Feature_10	Target
0	0.038076	0.050680	0.061696	0.021872	-0.044223	-0.034821	-0.043401	-0.002592	0.019907	-0.017646	151.0
1	-0.001882	-0.044642	-0.051474	-0.026328	-0.008449	-0.019163	0.074412	-0.039493	-0.068332	-0.092204	75.0
2	0.085299	0.050680	0.044451	-0.005670	-0.045599	-0.034194	-0.032356	-0.002592	0.002861	-0.025930	141.0
3	-0.089063	-0.044642	-0.011595	-0.036656	0.012191	0.024991	-0.036038	0.034309	0.022688	-0.009362	206.0
4	0.005383	-0.044642	-0.036385	0.021872	0.003935	0.015596	0.008142	-0.002592	-0.031988	-0.046641	135.0
...	...	...	...	...	...	...	...	...	...	...	...
437	0.041708	0.050680	0.019662	0.059744	-0.005697	-0.002566	-0.028674	-0.002592	0.031193	0.007207	178.0
438	-0.005515	0.050680	-0.015906	-0.067642	0.049341	0.079165	-0.028674	0.034309	-0.018114	0.044485	104.0
439	0.041708	0.050680	-0.015906	0.017293	-0.037344	-0.013840	-0.024993	-0.011080	-0.046883	0.015491	132.0
440	-0.045472	-0.044642	0.039062	0.001215	0.016318	0.015283	-0.028674	0.026560	0.044529	-0.025930	220.0
441	-0.045472	-0.044642	-0.073030	-0.081413	0.083740	0.027809	0.173816	-0.039493	-0.004222	0.003064	57.0

442 rows × 11 columns

Assumptions for Linear Regression

For a linear regression model to be considered significant and efficient, there are some **key assumptions** that need to be met. These are:

1. There must be a linear relationship between the independent variable and the dependent variable.

2. There should be little to no multicollinearity in the data. That is, the independent variables should not be too highly correlated with each other.

3. The observations should be independent of each other.

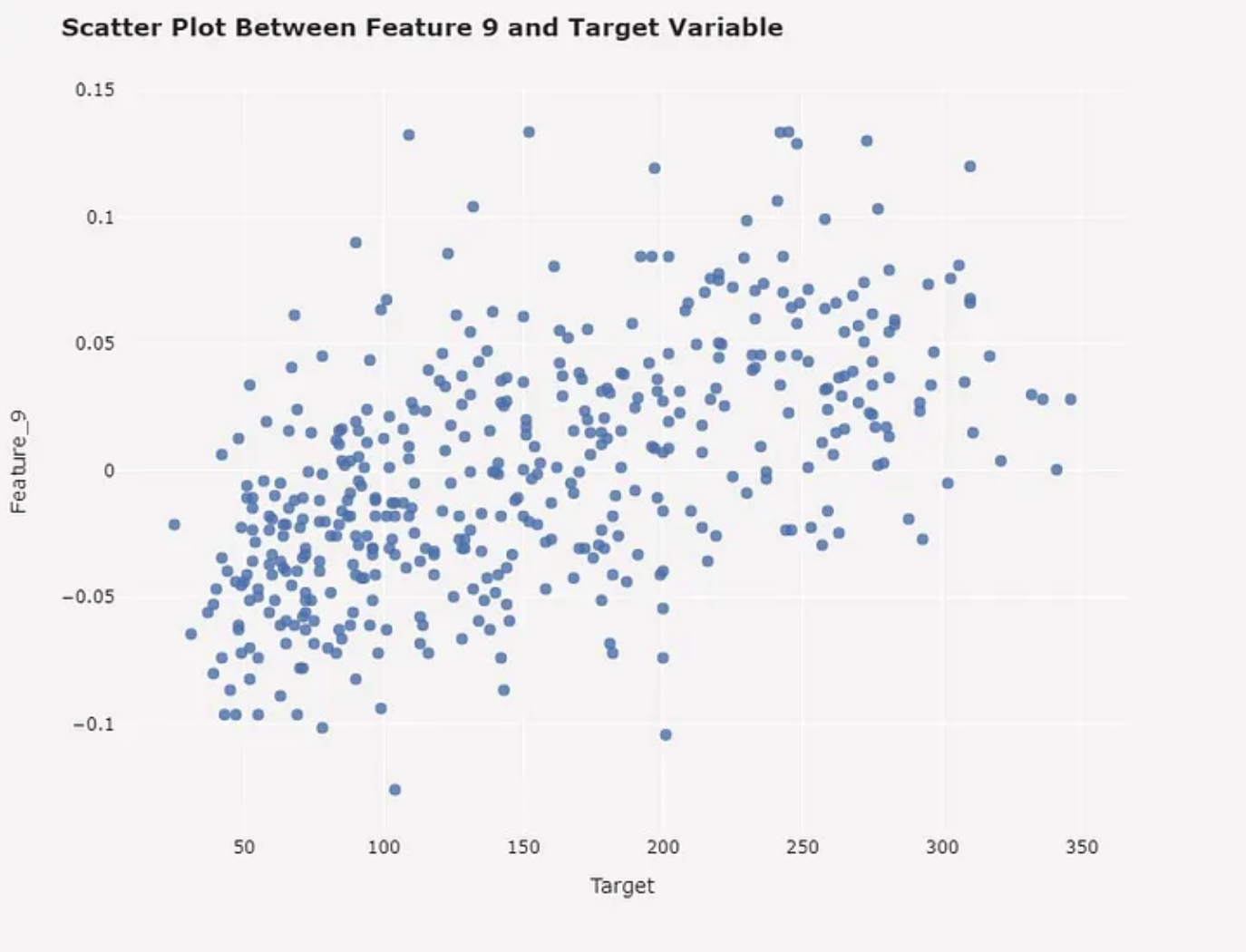
4. The residuals — errors between the predicted values and the true values — should be normally distributed.

5. The variance of residuals should be constant across all levels of the dependent variable, this is described as homoscedasticity.

Failing to meet the assumptions listed above can have many implications, among them is having a model with **poor predictive** power, unstable coefficient estimates, inaccurate hypothesis tests, and biased errors.

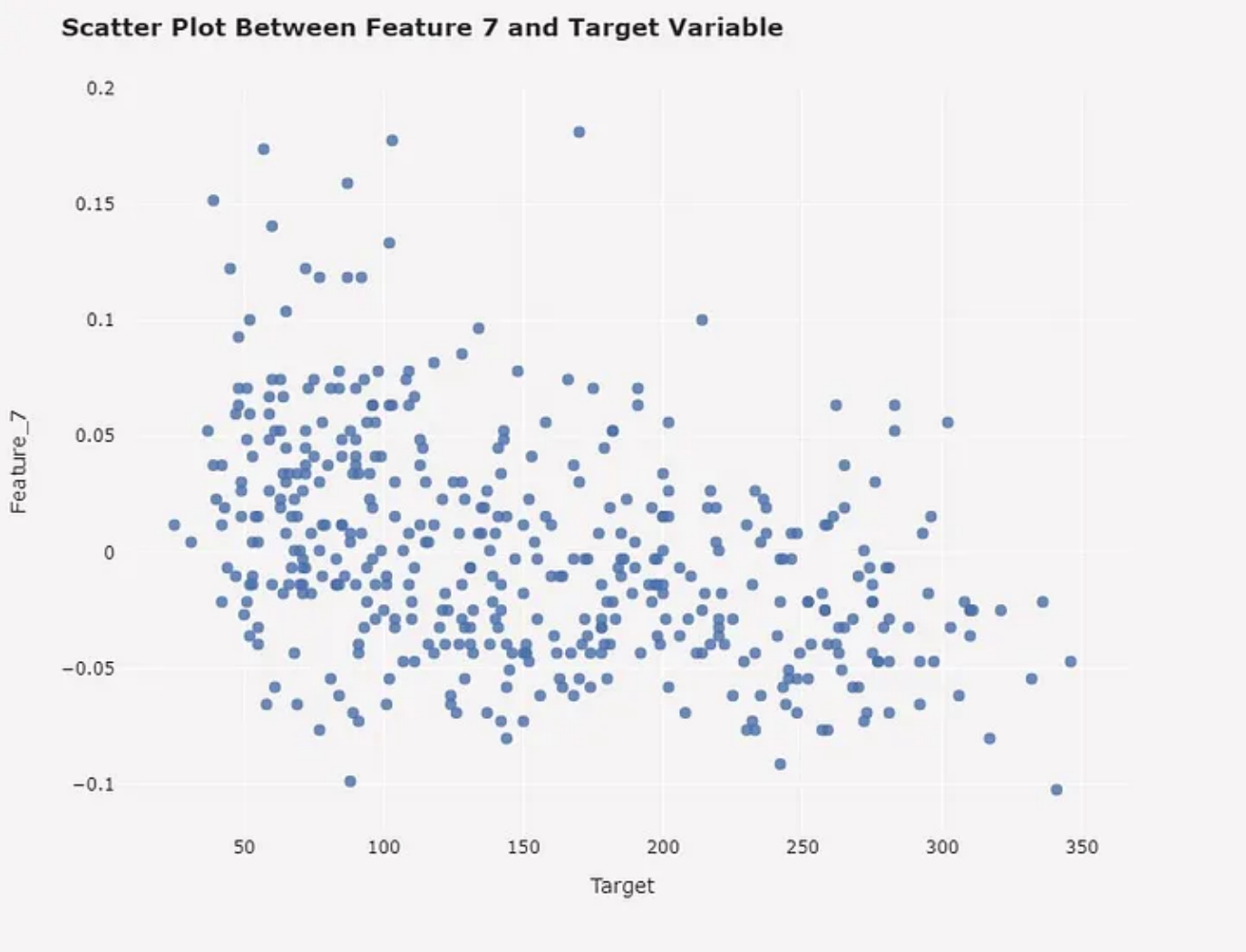
We can use Exploratory Data Analysis and data visualization tools to identify whether our data meets the assumptions listed above.

We can start by identifying whether there is or isn't a linear relationship between Feature 9 and the Target variable by plotting a scatter plot.



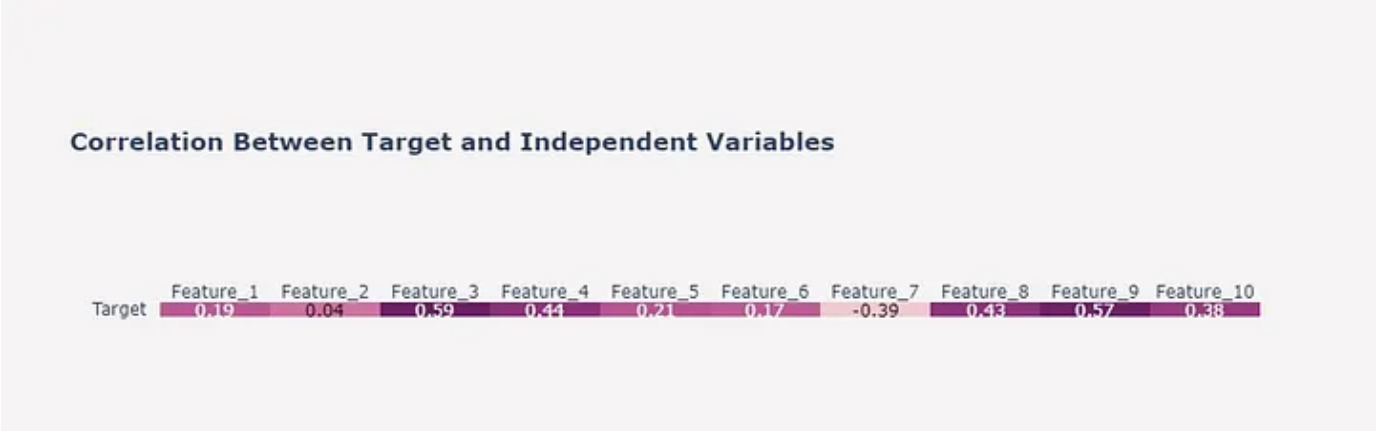
We can clearly identify a positive linear relationship between `Feature_9` and the target variable, as we can see that as values on the X-axis increase, the values on the Y-axis also increase.

A similar linear relationship, although negative, can be seen among `Feature_7` and the target variable, where a decrease in the Y-axis leads to an increase in the X-axis.



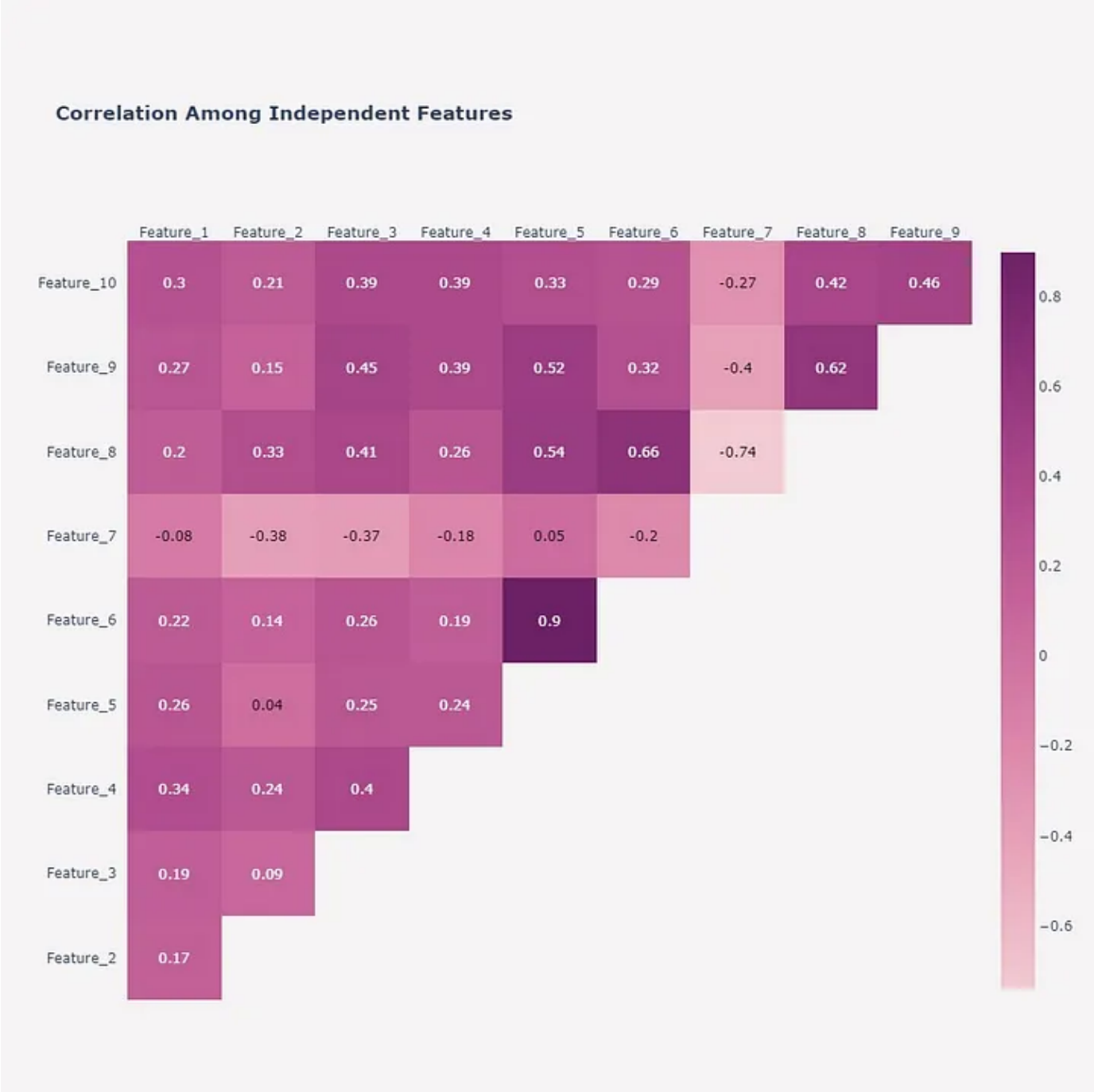
We can also plot a correlation plot containing Pearson's R values between the target variable and the independent variables.

In the plot, values will range from -1 to 1, where -1 indicates a perfect negative relationship and 1 indicates a perfect positive relationship. Values closer to 0 suggest no linear relationship.



We can see that the target variable has the strongest positive linear relationship with `Feature_3`, the strongest negative linear relationship with `Feature_7`, and there is possibly no real relationship between the target variable and `Feature_2`.

We can also use correlation plots to check for multicollinearity, to see if there is any strong correlation between the independent variables.



It’s possible to see a very strong correlation between `Feature_6` and `Feature_5`. Let's see how they relate to each other on the scatter plot.

We can observe a very strong trend where values increase in both the X and Y axes.

With the plots above, we have been able to obtain some valuable insights on the data we are working with. First, we already identified features that have a stronger linear relationship with the target variable. Second, we discovered that there is a high correlation between two distinct independent features, which we are going to address during the modeling phase.

Modeling

For the modeling step, we are going to use `statsmodel.OLS` to fit the linear regression model to the training data. This will also allow us to print a

summary describing key metrics for a better understanding on how well the model fits the data.

I am also going to use a testing set to predict the values for the dependent variable, and use the ***Root Mean Squared Error*** to evaluate performance. The RMSE is given by the following formula:

$$\mathrm{RMSE} = \sqrt{\frac{\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2}{n}}$$

Where:

Y_i is the true value for Y in the i^{th} sample.

$\hat{Y}_i$ is the predicted value for Y in the i^{th} sample.

The RMSE is simply the square root of the average of the squared differences between the actual and predicted values. It is expressed in the same units as the target variable, making it very intuitive to comprehend. The closer to 0, the most accurate are the predictions.

To start the modeling process, we are going to once more split the dataframe into independent variables X and dependent variable Y.

```
X = df.drop('Target', axis = 1) # Selecting independent features
y = df.Target # Selecting target variable
```

We must also split the data into training and testing set, for fitting and evaluating model's performance.

```
# Creating training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size = .3, # 30% of data
                                                    shuffle = True, # Shuffling v
                                                    random_state = 42)
```

We are also going to add a constant to the X dataframes. This is basically the first column of 1_s we have seen before, used for the intercept, which is the β_1 in the β vector.

```
# Adding a Constant term for the Intercept
X_train = sm.add_constant(X_train)
X_test = sm.add_constant(X_test)
```

```
X_train # Visualizing X dataframe with the constant
```

	const	Feature_1	Feature_2	Feature_3	Feature_4	Feature_5	Feature_6	Feature_7	Feature_8	Feature_9	Feature_10
225	1.0	0.030811	0.050680	0.032595	0.049415	-0.040096	-0.043589	-0.069172	0.034309	0.063015	0.003064
412	1.0	0.074401	-0.044642	0.085408	0.063187	0.014942	0.013091	0.015505	-0.002592	0.006207	0.085907
118	1.0	-0.056370	0.050680	-0.010517	0.025315	0.023198	0.040022	-0.039719	0.034309	0.020609	0.056912
114	1.0	0.023546	-0.044642	0.110198	0.063187	0.013567	-0.032942	-0.024993	0.020655	0.099241	0.023775
364	1.0	0.001751	0.050680	-0.006206	-0.019442	-0.009825	0.004949	-0.039719	0.034309	0.014821	0.098333
...	...	...	...	...	...	...	...	...	...	...	...
106	1.0	-0.096328	-0.044642	-0.076264	-0.043542	-0.045599	-0.034821	0.008142	-0.039493	-0.059471	-0.083920
270	1.0	0.005383	0.050680	0.030440	0.083844	-0.037344	-0.047347	0.015505	-0.039493	0.008641	0.015491
348	1.0	0.030811	-0.044642	-0.020218	-0.005670	-0.004321	-0.029497	0.078093	-0.039493	-0.010903	-0.001078
435	1.0	-0.012780	-0.044642	-0.023451	-0.040099	-0.016704	0.004636	-0.017629	-0.002592	-0.038460	-0.038357
102	1.0	-0.092695	-0.044642	0.028284	-0.015999	0.036958	0.024991	0.056003	-0.039493	-0.005142	-0.001078

309 rows × 11 columns

Now we may fit the model to the data and draw some conclusions by looking at the summary.

```
# Fitting model
model = sm.OLS(y_train, X_train).fit()
print(model.summary(alpha = 0.05))
```

OLS Regression Results

Dep. Variable:	Target	R-squared:	0.524
Model:	OLS	Adj. R-squared:	0.508
Method:	Least Squares	F-statistic:	32.86
Date:	Sun, 17 Sep 2023	Prob (F-statistic):	1.37e-42
Time:	14:33:09	Log-Likelihood:	-1671.5
No. Observations:	309	AIC:	3365.
Df Residuals:	298	BIC:	3406.
Df Model:	10		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	151.0082	3.143	48.050	0.000	144.823	157.193
Feature_1	29.2540	74.280	0.394	0.694	-116.926	175.434
Feature_2	-261.7065	73.301	-3.570	0.000	-405.961	-117.452
Feature_3	546.2997	81.326	6.717	0.000	386.255	706.345
Feature_4	388.3983	77.707	4.998	0.000	235.474	541.322
Feature_5	-901.9597	479.190	-1.882	0.061	-1844.985	41.066
Feature_6	506.7632	385.746	1.314	0.190	-252.368	1265.895
Feature_7	121.1544	245.611	0.493	0.622	-362.197	604.506
Feature_8	288.0353	196.608	1.465	0.144	-98.881	674.951
Feature_9	659.2690	207.907	3.171	0.002	250.118	1068.420
Feature_10	41.3767	79.783	0.519	0.604	-115.632	198.386

Omnibus:	1.511	Durbin-Watson:	1.764
Prob(Omnibus):	0.470	Jarque-Bera (JB):	1.417
Skew:	0.056	Prob(JB):	0.492
Kurtosis:	2.688	Cond. No.	215.

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

There are many insights that can be extracted from the summary above. Let's start by observing the features and their coefficients.

Remember the linear regression equation:

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \epsilon$$

The ***Intercept(β_0)*** is equal to 151.0082. It means that the predicted value of ***Y*** will be 151.0082 when all the other features are equal to 0.

The β_1 coefficient is the average change in ***Y*** for a one-unit change in X_1, which is the `Feature_1` variable. In this case, a one-unit increase in `Feature_1` is going to represent an increase of about 29.2540 in the predicted value for ***Y*** when all other features are constant.

We can also see negative coefficients, which is the case of the coefficient of `Feature_5`. This suggests that for one-unit increase in this feature we may expect a decrease of about 901.9597 in the predicted value of ***Y***, as long as all other features are held constant.

We also have in the summary the standard errors ***std err*** values. When looking at this metric, we want values to be as low as possible, indicating a more reliable estimate for coefficient values. The ***P***>|***t***| tells us if the feature is statistically significant or not. We want these values to be **below** 0.050.05 to deem a feature statistically significant for predicting the outcome of ***Y***.

With that in mind, we can notice that the most reliable independent variables were **`Feature_2`**, **`Feature_3`**, **`Feature_4`**, and **`Feature_9`**.

We also have other important metrics to look at. The R-Squared metric at 0.524 suggests that me model is able to explain approximately 52.452.4% of

the variance in the target variable.

The ***Durbin-Watson*** statistic is used to detect autocorrelation between the residuals. Values for the ***DW*** range between 0 to 4 and, as a rule of thumb, values between 1.5 to 2.5 suggest that there is no autocorrelation between the residuals and our samples are independent of each other. Considering our ***DW*** value is equal to 1.764, this ticks one of the key assumptions of Linear Regression.

Another relevant metric is the ***Jarque-Bera*** statistic, which tells us whether the residuals are normally-distributed or not. As a rule of thumb, if ***JB*** is too far from zero, it indicates that the data is not-normally distributed. In this case, a ***JB*** equal to 1.417 suggests that our residuals may be normally distributed. The values for the ***Kurtosis***, close to 3.0, and ***Skew***, close to 0, are also relevant metrics to suggest our residuals are indeed normally distributed. This is another positive sign when talking about the assumptions regarding Linear Regression models.

To confirm if the residuals are normally-distributed or not, we can use a Histogram plot to visualize how they're distributed.

It's possible to see that the curve do resemble a bell curve, which is a sign of the normal distribution.

Other metrics, such as ***Log-Likelihood***, ***AIC***, and ***BIC*** are only relevant when comparing two different models to each other. We will approach them later on.

Let's move forward and predict values for our dependent variable Y in the testing set.

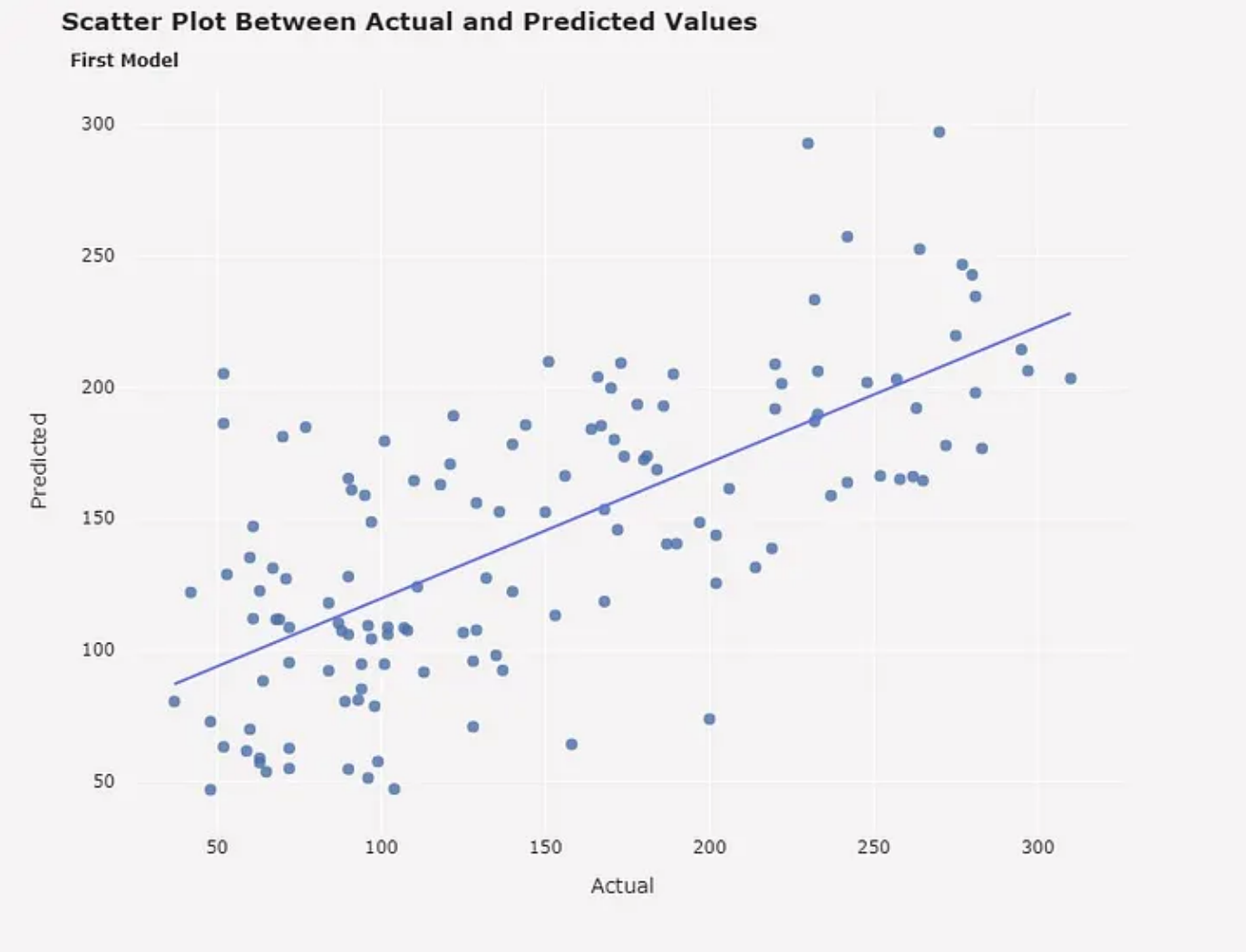
```
y_pred = model.predict(X_test) # Running predictions
rmse = mean_squared_error(y_test, y_pred,squared = False) # Computing RMSE

print(f'\nRoot Mean Squared Error for Baseline Model: {rmse:.2f}')
```

```
Root Mean Squared Error for Baseline Model: 53.12
```

The RMSE shows us that, on average, our model is getting the value of Y wrong by 53.12 units.

We can also plot a scatter plot between actual and predicted values to visualize how they relate to each other.



We can take the first model above as a baseline. Below, we can try changing our data to see if we can improve performance.

Even though `Feature_2` was deemed relevant in the summary above, I want to test how well we perform without it. I want to remove this feature due to its low Pearson's R value when we checked its correlation with the target variable. I'm also going to remove `Feature_6` due to its high correlation with `Feature_5` and its lower correlation to the target variable.

```
# Selecting feature to remove
features_to_remove = ['Feature_2',
                      'Feature_6']
```

```
# Removing them
new_X_train = X_train.drop(features_to_remove, axis =1)
new_X_test = X_test.drop(features_to_remove, axis =1)
```

	const	Feature_1	Feature_3	Feature_4	Feature_5	Feature_7	Feature_8	Feature_9	Feature_10
225	1.0	0.030811	0.032595	0.049415	-0.040096	-0.069172	0.034309	0.063015	0.003064
412	1.0	0.074401	0.085408	0.063187	0.014942	0.015505	-0.002592	0.006207	0.085907
118	1.0	-0.056370	-0.010517	0.025315	0.023198	-0.039719	0.034309	0.020609	0.056912
114	1.0	0.023546	0.110198	0.063187	0.013567	-0.024993	0.020655	0.099241	0.023775
364	1.0	0.001751	-0.006206	-0.019442	-0.009825	-0.039719	0.034309	0.014821	0.098333
...	...	...	...	...	...	...	...	...	...
106	1.0	-0.096328	-0.076264	-0.043542	-0.045599	0.008142	-0.039493	-0.059471	-0.083920
270	1.0	0.005383	0.030440	0.083844	-0.037344	0.015505	-0.039493	0.008641	0.015491
348	1.0	0.030811	-0.020218	-0.005670	-0.004321	0.078093	-0.039493	-0.010903	-0.001078
435	1.0	-0.012780	-0.023451	-0.040099	-0.016704	-0.017629	-0.002592	-0.038460	-0.038357
102	1.0	-0.092695	0.028284	-0.015999	0.036958	0.056003	-0.039493	-0.005142	-0.001078

309 rows × 9 columns

```
# Fitting model
model = sm.OLS(y_train, new_X_train).fit()
print(model.summary())
```

OLS Regression Results

Dep. Variable:	Target	R-squared:	0.502
Model:	OLS	Adj. R-squared:	0.488
Method:	Least Squares	F-statistic:	37.75
Date:	Sun, 17 Sep 2023	Prob (F-statistic):	3.19e-41
Time:	14:33:10	Log-Likelihood:	-1678.7
No. Observations:	309	AIC:	3375.
Df Residuals:	300	BIC:	3409.
Df Model:	8		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	150.8524	3.206	47.060	0.000	144.544	157.161
Feature_1	5.6966	75.092	0.076	0.940	-142.077	153.470
Feature_3	607.5941	81.062	7.495	0.000	448.073	767.115
Feature_4	338.9592	78.137	4.338	0.000	185.193	492.725
Feature_5	-258.9041	140.745	-1.840	0.067	-535.876	18.068
Feature_7	-84.5802	168.822	-0.501	0.617	-416.806	247.646
Feature_8	149.7657	196.500	0.762	0.447	-236.928	536.460
Feature_9	448.7356	96.500	4.650	0.000	258.832	638.639
Feature_10	40.7414	81.207	0.502	0.616	-119.066	200.548

Omnibus:	6.486	Durbin-Watson:	1.807
Prob(Omnibus):	0.039	Jarque-Bera (JB):	4.729
Skew:	0.176	Prob(JB):	0.0940
Kurtosis:	2.507	Cond. No.	87.3

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

We can start it out by noticing that R-Squared, at 0.502, is lower than the previous one. This may indicate that this model is able to explain a lesser percentage of the variance in Y than the previous model.

We may also use the **Log-Likelihood**, **AIC**, and **BIC** metrics to compare both models. Overall, a Log-Likelihood closer to 0 and lower AIC and BIC values are preferred. In this case, this model performs worse than the previous model, although not by much.

Values for ***Kurtosis, Skew,*** and the ***Durbin-Watson*** statistic also place the distribution of residuals in this case a bit further from the normal distribution compared to the previous model.

Let's predict on the testing set to find out how this model performs.

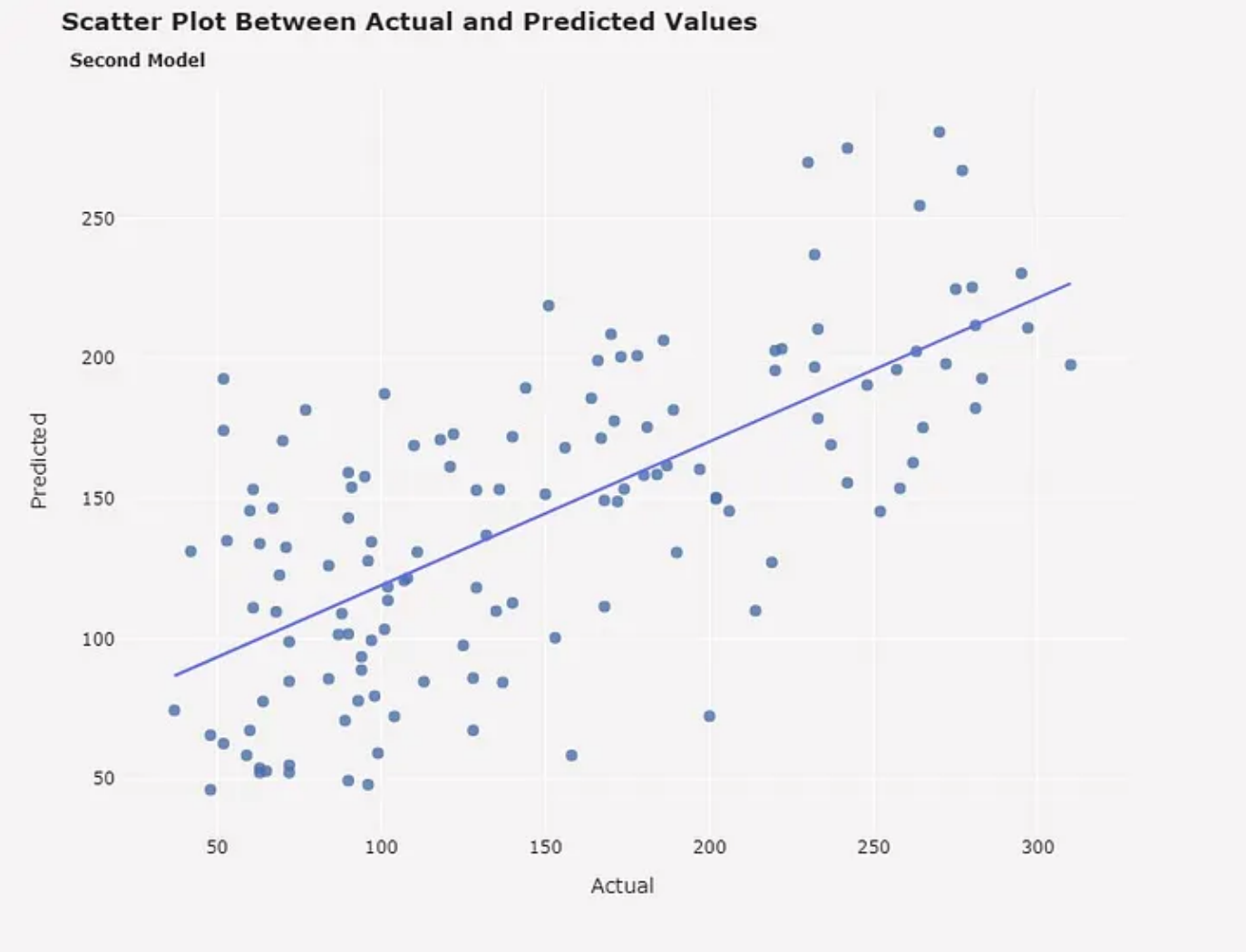
```
y_pred = model.predict(new_X_test)
rmse = mean_squared_error(y_test, y_pred,squared = False)

print(f'\nRoot Mean Squared Error for Baseline Model: {rmse:.2f}')
```

```
Root Mean Squared Error for Baseline Model: 53.64
```

We have a slighlty **worse** performance compared to the first model, with a Root Mean Squared Error sligthly larger than the first one.

We can once again plot a scatter plot displaying the difference data points for actual and predicted values.



The second model, then, performed slightly worse than the first one, although the changes weren't really drastic.

Let's go back to the inferences we made when analyzing the $P>|t|$ values for the independent variables of the first model. We could observe that features with larger p-values, such as `Feature_7` with 0.622, were deemed not very statistically significant. In the cell below, we are going to select all these features with high ***p–values*** and remove them from our X sets.

```
# Selecting feature to remove
features_to_remove = ['Feature_1',
                      'Feature_5',
```

```
                        'Feature_6',
                        'Feature_7',
                        'Feature_8',
                        'Feature_10']

# Removing features
new_X_train = X_train.drop(features_to_remove, axis =1)
new_X_test = X_test.drop(features_to_remove, axis =1)
```

	const	Feature_2	Feature_3	Feature_4	Feature_9
225	1.0	0.050680	0.032595	0.049415	0.063015
412	1.0	-0.044642	0.085408	0.063187	0.006207
118	1.0	0.050680	-0.010517	0.025315	0.020609
114	1.0	-0.044642	0.110198	0.063187	0.099241
364	1.0	0.050680	-0.006206	-0.019442	0.014821
...	...	...	...	...	...
106	1.0	-0.044642	-0.076264	-0.043542	-0.059471
270	1.0	0.050680	0.030440	0.083844	0.008641
348	1.0	-0.044642	-0.020218	-0.005670	-0.010903
435	1.0	-0.044642	-0.023451	-0.040099	-0.038460
102	1.0	-0.044642	0.028284	-0.015999	-0.005142

309 rows × 5 columns

We now have a more compact X sets with 4 features plus the constant.

```
# Fitting model
model = sm.OLS(y_train, new_X_train).fit()
print(model.summary())
```

OLS Regression Results

Dep. Variable:	Target	R-squared:	0.481
Model:	OLS	Adj. R-squared:	0.474
Method:	Least Squares	F-statistic:	70.44
Date:	Sun, 17 Sep 2023	Prob (F-statistic):	3.76e-42
Time:	14:33:10	Log-Likelihood:	-1685.0
No. Observations:	309	AIC:	3380.
Df Residuals:	304	BIC:	3399.
Df Model:	4		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	150.7472	3.246	46.439	0.000	144.360	157.135
Feature_2	-134.9760	69.262	-1.949	0.052	-271.270	1.318
Feature_3	663.2952	78.484	8.451	0.000	508.855	817.735
Feature_4	333.5608	75.990	4.390	0.000	184.028	483.094
Feature_9	453.2164	78.029	5.808	0.000	299.672	606.761

Omnibus:	5.146	Durbin-Watson:	1.762
Prob(Omnibus):	0.076	Jarque-Bera (JB):	4.192
Skew:	0.187	Prob(JB):	0.123
Kurtosis:	2.569	Cond. No.	28.4

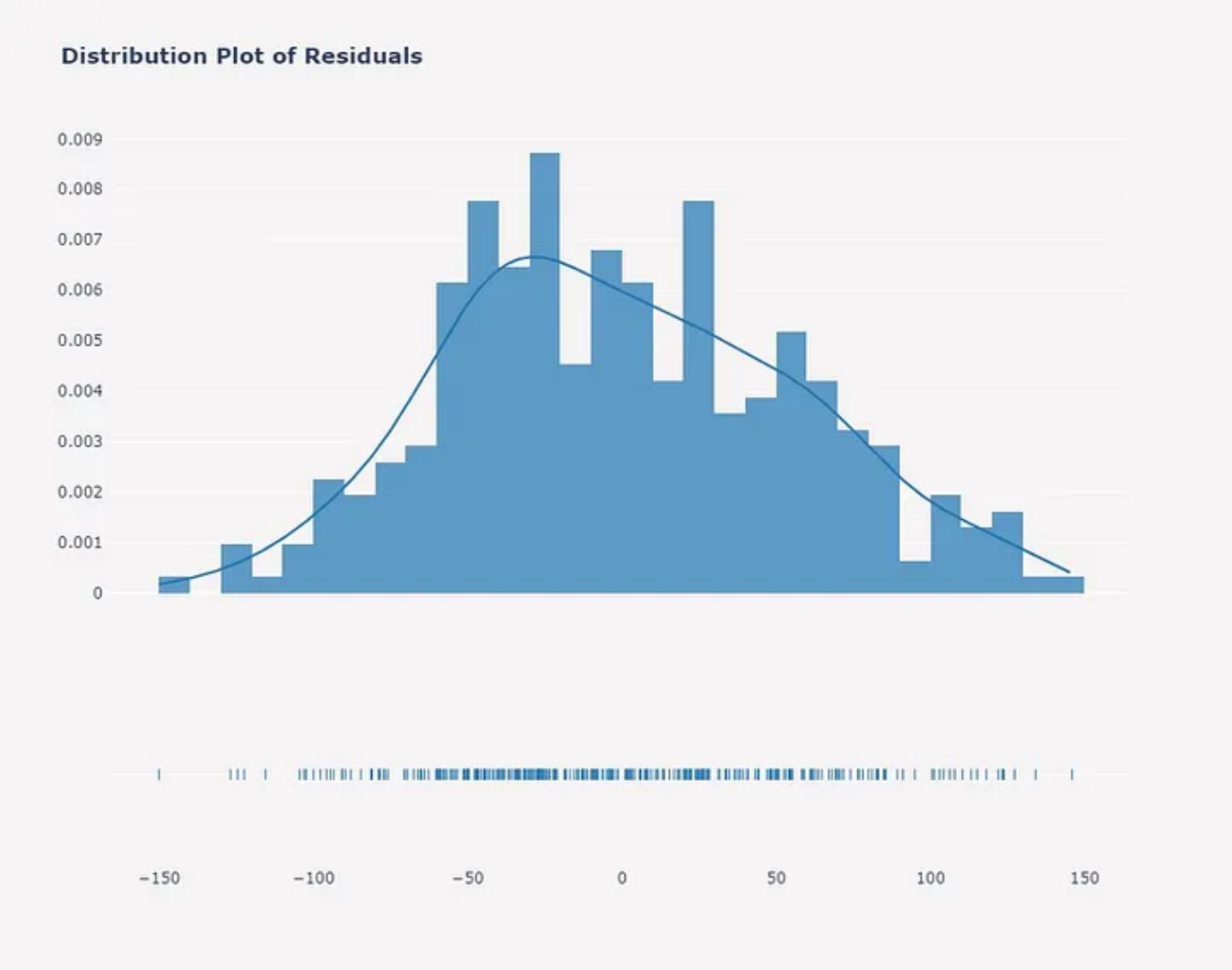
Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

Compared to the first model, we once more have inferior values for the Log-Likelihood and AIC. For the BIC, however, in this case we have a lower value, considered better than the more slightly higher value of the first model.

This model also has a lower R-Squared, explaining 48.1% of the variance of the target variable, a bit lower than that of the first model.

When looking at the independent variables, it seems that all are statistically significant, given that $P>|t|$ are below 0.05. `Feature_2` is the only one that is slightly above that threshold and it's also the only one with a negative coefficient.

It’s also noticeable that the Jarque-Bera statistic indicate that residuals may not be normally-distributed. Let’s plot the distribution to see how it is shaped.



The curve doesn’t really resemble a bell curve, which suggests the non normality of the distribution of residuals in this case.

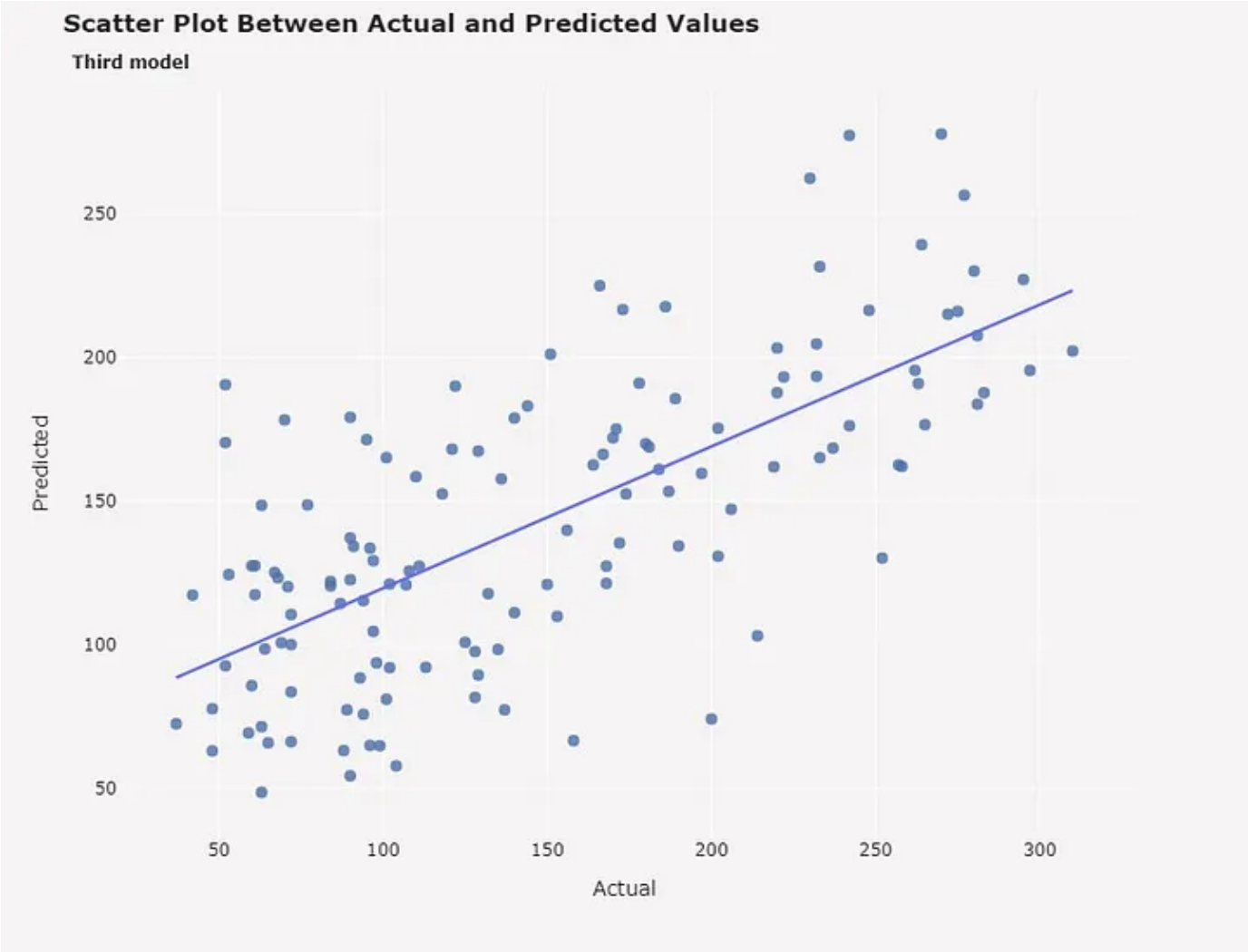
Let’s predict the *Y* values on the testing set to see how the Root Mean Squared Error compares to the former models.

```
y_pred = model.predict(new_X_test)
rmse = mean_squared_error(y_test, y_pred,squared = False)

print(f'\nRoot Mean Squared Error for Baseline Model: {rmse:.2f}')
```

```
Root Mean Squared Error for Baseline Model: 52.68
```

This model achieves a better RMSE score, by scoring a value lower than the other two models. We can also plot the scatter plot between actual and predicted values.



The third model generalized better to the testing set than the other models. However, according to the metrics, it seems like the first model may best meet the assumptions of linear regression, providing a more trustworthy estimates and predictions.

It is still relevant to point out that all three models are statistically significant, according to the **F-Statistic** and their very low **Prob (F-statistic)** values, indicating that our independent variables ***Xs*** are indeed useful for predicting the values of ***Y***.

Conclusion

In this notebook, we've taken a deep dive into key concepts of Linear Regression, exploring its mathematical foundations, assumptions, and practical applications. We've worked with the Sklearn's diabetes dataset to fit multiple models, each with its own set of features, and evaluated their performance on a hold-out testing set using metrics like Root Mean Squared Error, as well as other statistical significance metrics.

While the first model seemed to best meet the assumptions of linear regression, which might translate into more trustworthy estimates, the third model ended up generalizing better to the hold-out set, achieving the lowest Root Mean Squared Error score.

Additional tests could further elucidate the generalization capabilities of these models. Cross-validation, for instance, could be highly effective in this context. And this is where I leave my call to action for you, the reader: pick up where I left off and run your own tests. After all, **science thrives on collaboration and the collective refinement of our findings.**

If you liked this article, make sure to leave your upvote. Also, engage in the comment section by leaving your suggestions and insights. These are all great incentives to keep going and publishing these. 😊

Thank you so much for your support!

Further Reading...

During the analyses of the different models, I have mentioned several metrics that I didn't cover with that much extension. For this reason, I am leaving below a number of sources where you can read more about them.

- **F-Statistic:**
- F Statistic / F Value: Simple Definition and Interpretation
- A Simple Guide to Understanding the F-Test of Overall Significance in Regression
- **Log-Likelihood:**
- Log Likelihood Function
- How to Interpret Log-Likelihood Values (With Examples)
- **Durbin Watson Test:**
- Durbin Watson Test: What It Is in Statistics, With Examples
- Understanding Durbin-Watson Test
- **Jarque-Bera Test:**
- Jarque-Bera Test
- Wikipedia: Jarque–Bera test

- **AIC (Akaike Information Criterion):**
- Akaike's Information Criterion: Definition, Formulas
- **BIC (Bayesian Information Criterion):**
- Bayesian Information Criterion (BIC) / Schwarz Criterion
- **R-Squared:**
- Coefficient of Determination (R Squared): Definition, Calculation

Stay curious!

Luis Fernando Torres

Let's connect! 🔗
LinkedIn • Kaggle • HuggingFace

Statistics | Regression | Data Science | Machine Learning | AI

Written by Luís Fernando Torres

689 Followers

Data Scientist | Machine Learning | Commodities Trader & Investor

More from Luís Fernando Torres

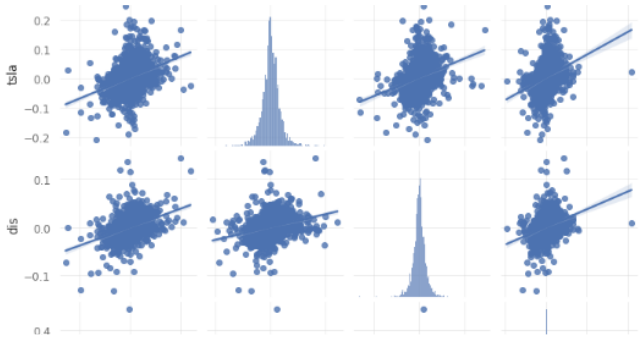
Luís Fernando Torres in LatinXinAI

How I Deployed a Machine Learning Model for the First Time

Going beyond Jupyter Notebooks 🚀

13 min read · Jul 18

637 6



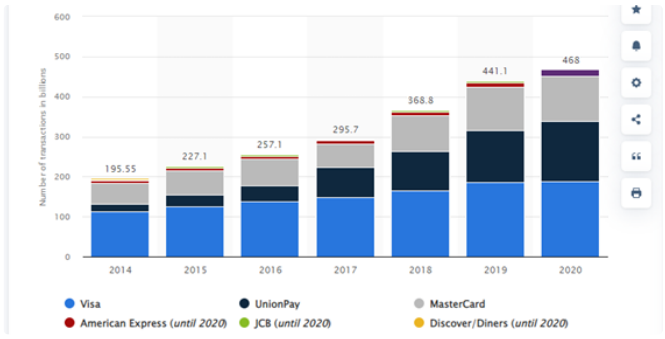
Luís Fernando Torres in InsiderFinance Wire

Introduction to Quant Investing with Python

Introduction

15 min read · Mar 30

897 9



Luís Fernando Torres

Machine Learning for Credit Card Fraud Detection

Introduction

Luís Fernando Torres in LatinXinAI

Logistic Regression in Credit Risk: The Role of Weight of Evidence an...

Optimizing Performance and Simplicity in White Box Models

12 min read · Mar 30

127 1

10 min read · Aug 27

34

See all from Luís Fernando Torres

Recommended from Medium

Brandon Wohlwend

Three Regression Models for Data Science: Linear Regression, Lasso...

In the ever-evolving field of data science, the art of making accurate predictions by...

21 min read · Jul 14

37 1

Erkan Hatipoğlu in Towards AI

Multivariate Linear Regression From Scratch

With Code in Python

12 min read · Mar 21

60