Linear Regression on a Simulated Dataset

Table of contents

1 About

This PDF shows applying a linear regression model on a simulated datset of 3 features and 1 target variable. The purpose is to demonstrate the conceptual understanding of a linear algebra interpretation of the linear regression model.

2 Data

2.1 simulation and exporting

The data is simulated using the following code:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

```
feature_1 = np.random.normal(5, 3, 1000)
feature_2 = np.random.normal(10, 5, 1000)
feature_3 = np.random.normal(-7, 1.5, 1000)
```

Note

Food for thought: what if the features are not normally distributed?

```
beta_0, beta_1, beta_2, beta_3 = 10, 5, 3, 1
```

```
feature_1 = feature_1 * beta_1
feature_2 = feature_2 * beta_2
feature_3 = feature_3 * beta_3

y = beta_0 + feature_1 + feature_2 + feature_3

## add some error to y
y = y + np.random.normal(2, 1, 1000)

df = pd.DataFrame({'feature_1': feature_1, 'feature_2': feature_2, 'feature_3': feature_3,
```

```
## taking a peak at the data
df.sample(5)
```

	feature_1	feature_2	feature_3	y
862	44.052109	28.330858	-5.268885	79.335520
797	15.986749	49.982219	-6.878771	70.228390
891	35.490441	48.741265	-7.273572	88.401455
293	43.588368	25.356133	-9.072219	72.496816
235	3.579450	34.210834	-6.481955	44.264155

```
## export the dataset
df.to_csv('dataset.csv', index=False)
```

2.2 importing the dataset

```python
## read the dataset
data = pd.read_csv('dataset.csv')
```

```python
## rename columns as X_1, X_2, X_3
data.columns = ['X_1', 'X_2', 'X_3', 'y']
```

2.3 splitting the data

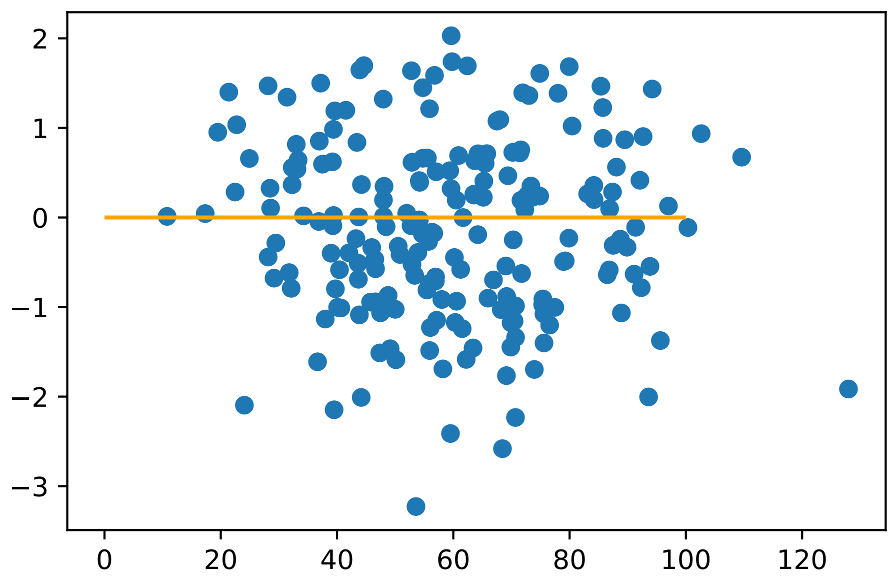
```python
## split the data into training and testing sets
from sklearn.model_selection import train_test_split

X = data[['X_1', 'X_2', 'X_3']]
y = data['y']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

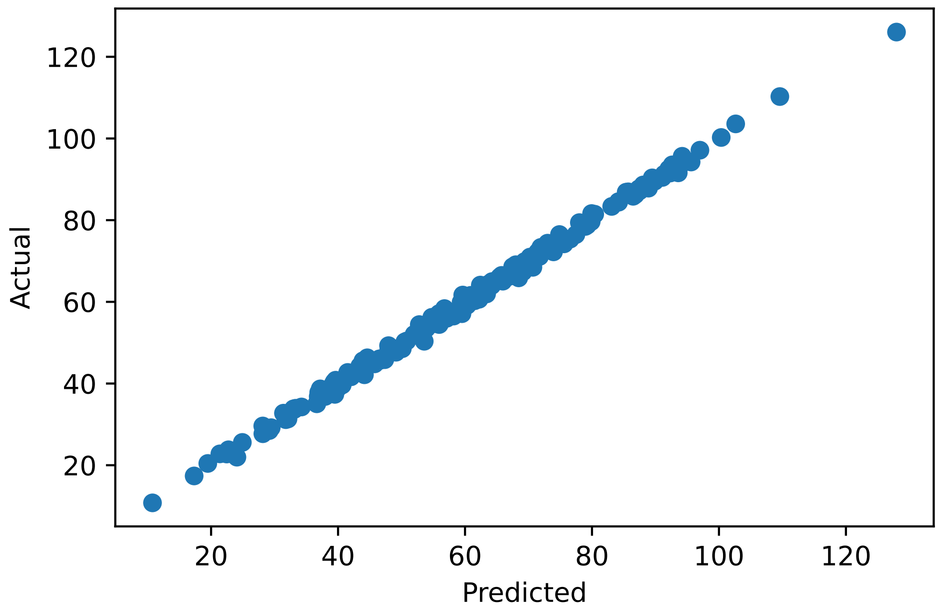
2.4 applying the Linear Regression model and evaluating the model

```python
## fit the model
from sklearn.linear_model import LinearRegression

model = LinearRegression()
model.fit(X_train, y_train)

## print the coefficients and intercept
print(model.coef_)
print(model.intercept_)

## predict on the test data
y_pred = model.predict(X_test)

## calculate the mean squared error
from sklearn.metrics import mean_squared_error

print(mean_squared_error(y_test, y_pred))
```

```
## calculate the R-squared
from sklearn.metrics import r2_score

print(r2_score(y_test, y_pred))

## plot the residuals

plt.scatter(y_pred, y_test - y_pred)

plt.hlines(y=0, xmin=0, xmax=100, color='orange')

plt.show()

## plot predictions vs actual

plt.scatter(y_pred, y_test)

plt.xlabel('Predicted')

plt.ylabel('Actual')

plt.show()
```

```
[0.9974904  0.99844136 0.98251561]
12.066560941672698
1.012775090210041
0.9974856529554142
```





3 Further actions:

1. The data can be from distributions other than the normal distribution.
2. I can omit the noise data and see what coefficients are estimated.