

Exercise 13

Import notebook funcs

```
from notebookfuncs import *
```

```
from IPython.display import Markdown, display, Math, Latex
def printmd(string): display(Markdown(string))
```

In this exercise you will create some simulated data and will fit simple linear regression models to it. Make sure to use the default random number generator with seed set to 1 prior to starting part (a) to ensure consistent results.

(a) Using the `normal()` method of your random number generator, create a vector, `x`, containing 100 observations drawn from a $N(0, 1)$ distribution. This represents a feature, `X`.

```
import numpy as np
import pandas as pd
import markdown

RNG = np.random.default_rng(1)

def generate_data(mean=0.0, sd=1.0, N=100):
    series = RNG.normal(size=N, loc=mean, scale=sd)
    return series

x = generate_data(mean=0.0, sd=1.0, N = 100);
```

(b) Using the `normal()` method, create a vector, `eps`, containing 100 observations drawn from a $N(0, 0.25)$ distribution—a normal distribution with mean zero and variance 0.25.

```
eps = generate_data(mean=0.0, sd = 0.25, N = 100);
```

(c) Using `x` and `eps`, generate a vector `y` according to the model

$$Y = -1 + 0.5 * X + \epsilon$$

```
beta_0 = -1  
beta_1 = 0.5  
y = -1 + 0.5 * x + eps;
```

What is the length of the vector `y`? What are the values of β_0 and β_1 in this linear model?

```
len(y)
```

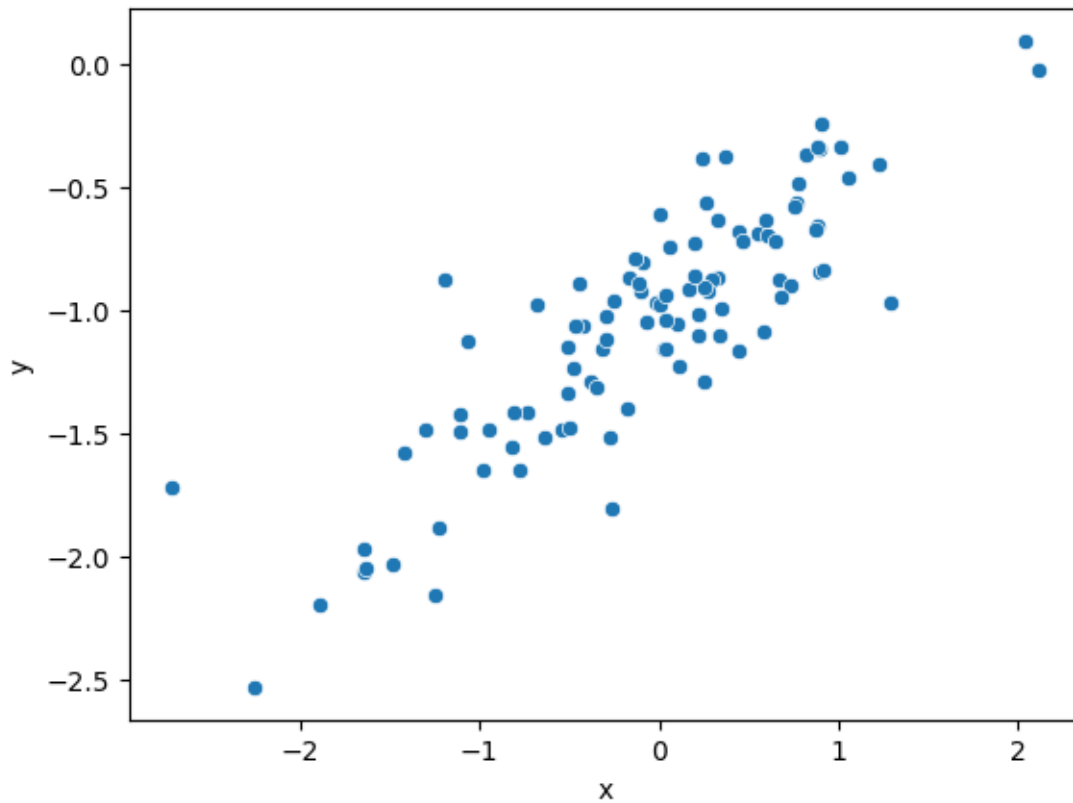
```
100
```

```
display(Math(rf"\beta_0 = {beta_0} \: and \: \beta_1 = {beta_1}"))
```

$$\beta_0 = -1 \text{ and } \beta_1 = 0.5$$

(d) Create a scatterplot displaying the relationship between `x` and `y`. Comment on what you observe.

```
import seaborn as sns  
df = pd.DataFrame({"x": x, "y": y})  
sns.scatterplot(data=df, x="x", y="y");
```



- *There appears to be a positive linear relationship between x and y when viewed visually through the scatterplot.*

(e) Fit a least squares linear model to predict y using x . Comment on the model obtained. How do $\hat{\beta}_0$ and $\hat{\beta}_1$ compare to β_0 and β_1 ?

```
import statsmodels.formula.api as smf

def get_results_df(results):
    result_df = pd.DataFrame(
        {
            "coefficients": results.params,
            "coefficients-se": results.bse,
            "tstatistic": results.tvalues,
            "p-value": results.pvalues,
            "r-squared": results.rsquared,
            "r-squared-adjusted": results.rsquared_adj,
```

```

        "pearson_coefficient": np.sqrt(results.rsquared),
        "rss": results.ssr,
        "sd_residuals": np.sqrt(results.mse_resid)
    }
)
return result_df

def regress_y_on_x(df):
    formula = "y ~ x"
    model = smf.ols(f"{formula}", df)
    results = model.fit()
    result_df = get_results_df(results)
    return results, result_df

orig_res, result_df = regress_y_on_x(df)
result_df

```

	coefficients	coefficients-se	tstatistic	p-value	r-squared	r-squared-adjusted	pearson_
Intercept	-1.019006	0.025138	-40.537258	4.843793e-63	0.74076	0.738115	0.860674
x	0.492146	0.029410	16.734055	1.738771e-30	0.74076	0.738115	0.860674

```

printmd(r"The  $\hat{\beta}_0$  = " + str(orig_res.params.iloc[0]) + r" and  $\hat{\beta}_1$  = ")

```

The $\hat{\beta}_0 = -1.0190063887490803$ and $\hat{\beta}_1 = 0.49214550728875883$ compare quite favourably to the population parameters $\beta_0 = -1$ and $\beta_1 = 0.5$.

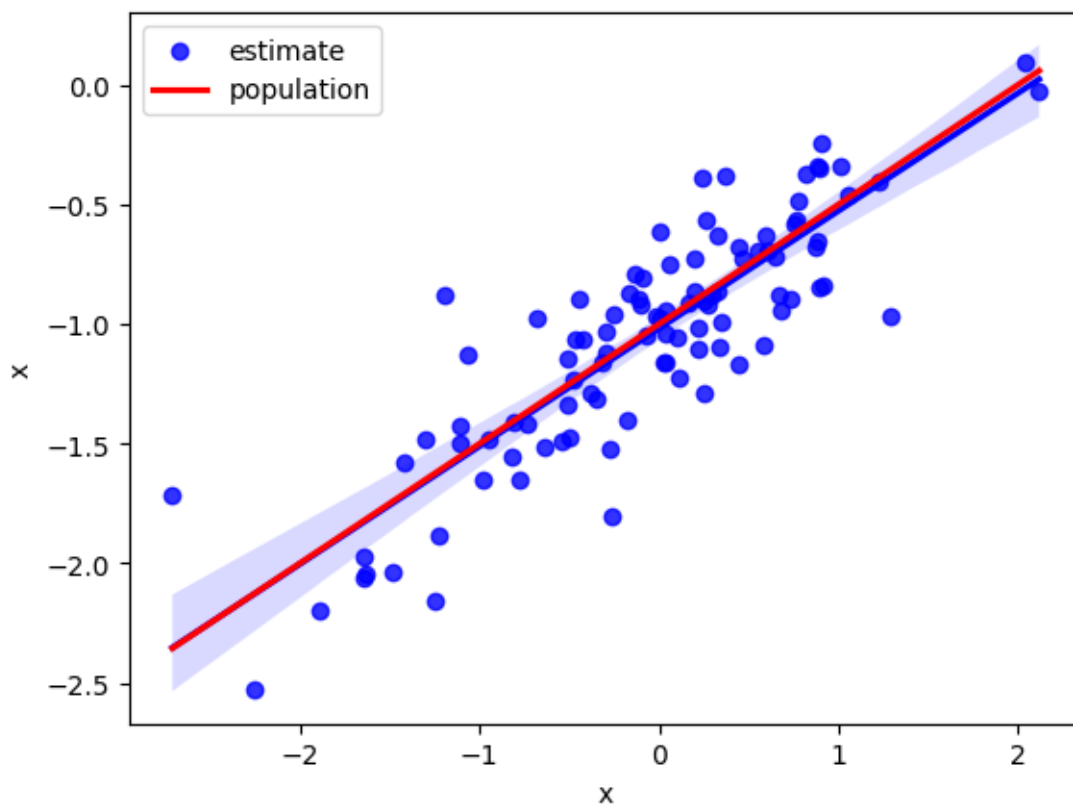
(f) Display the least squares line on the scatterplot obtained in (d). Draw the population regression line on the plot, in a different color. Use the legend() method of the axes to create an appropriate legend.

```

def draw_regplot(df):
    ax = sns.regplot(x="x",y="y",data=df,label="estimate", color="blue");
    x = df["x"]
    y_original = -1 + 0.5 * x
    sns.regplot(x=x, y=y_original, scatter=False,label="population",color="red",ax=ax);
    ax.legend();

draw_regplot(df)

```



(g) Now fit a polynomial regression model that predicts y using x and x^2 . Is there evidence that the quadratic term improves the model fit? Explain your answer.

```
simple_results = result_df
formula = "y ~ x + I(x**2)"
model = smf.ols(f"{formula}", df)
results = model.fit()
result_df = get_results_df(results)
```

	coefficients	coefficients-se	tstatistic	p-value	r-squared	r-squared-adjusted	pearson_
Intercept	-1.018189	0.029783	-34.186456	6.599982e-56	0.740768	0.735423	0.860679
x	0.491568	0.031588	15.561720	4.094544e-28	0.740768	0.735423	0.860679
I(x ** 2)	-0.001177	0.022706	-0.051847	9.587575e-01	0.740768	0.735423	0.860679

```
np.isclose(simple_results["r-squared"].iloc[0], result_df["r-squared"].iloc[0])
```

True

```
np.isclose(simple_results["r-squared-adjusted"].iloc[0], result_df["r-squared-adjusted"].iloc[0])
```

False

```
ci = results.conf_int(alpha=0.05)
ci[(ci[0] < 0) & (ci[1] > 0)]
```

	0	1
I(x ** 2)	-0.046243	0.043889

- The R^2 does not change significantly with the polynomial regression. So there is no evidence that the quadratic term improves the model fit.
- The $R^2_{adjusted}$ does increase but the coefficient for the polynomial term is not significant.
- Additionally, the confidence interval for the polynomial term spans both -ve and +ve axes i.e., zero lies in the range of the confidence interval.

(h) Repeat (a)–(f) after modifying the data generation process in such a way that there is less noise in the data. The model (3.39) should remain the same. You can do this by decreasing the variance of the normal distribution used to generate the error term ϵ in (b). Describe your results.

We decrease the standard deviation of the error terms or noise to 0.05 from 0.25.

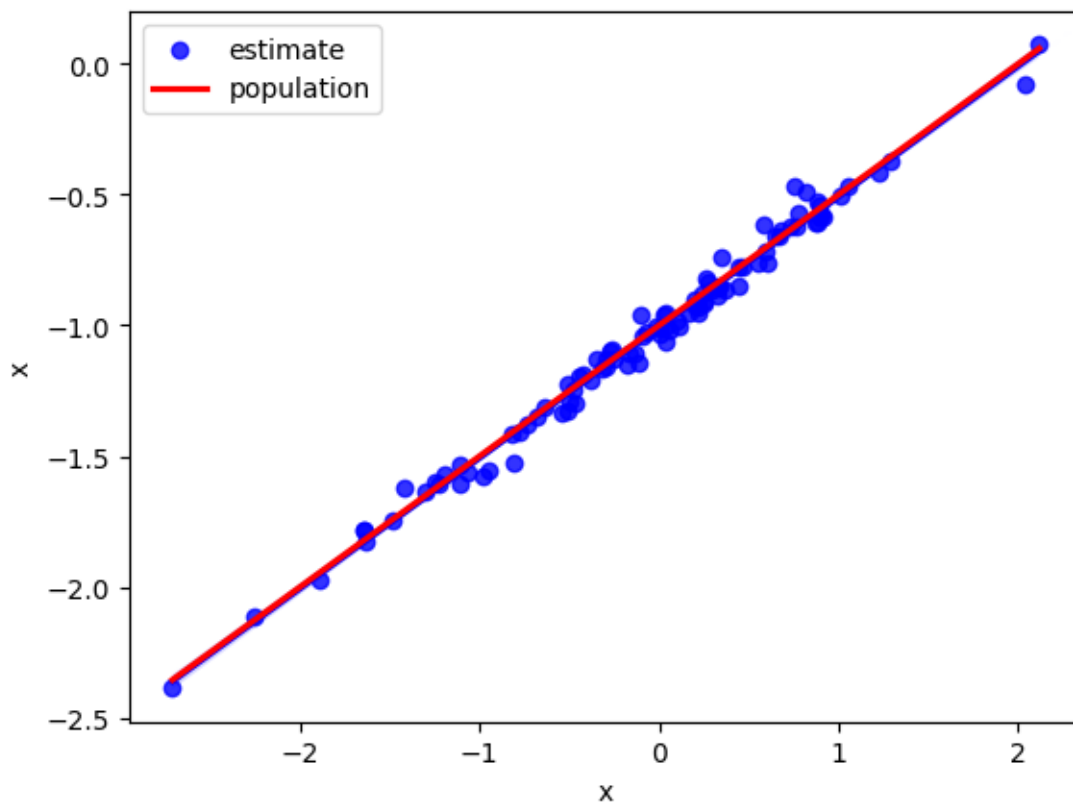
```
eps = generate_data(mean=0.0, sd = 0.05, N = 100);
```

```
y = -1 + 0.5 * x + eps;
df = pd.DataFrame({"x": x, "y": y});
```

```
less_noisier_results, result_df = regress_y_on_x(df)
result_df
```

	coefficients	coefficients-se	tstatistic	p-value	r-squared	r-squared-adjusted	pearson
Intercept	-1.008096	0.004629	-217.790607	2.013622e-133	0.988606	0.988489	0.9942
x	0.499361	0.005415	92.210827	4.844414e-97	0.988606	0.988489	0.9942

```
draw_regplot(df)
```



- We conclude that the less noiser the data, the closer the fit of the regression to the population parameters.

(i) Repeat (a)–(f) after modifying the data generation process in such a way that there is more noise in the data. The model (3.39) should remain the same. You can do this by increasing the variance of the normal distribution used to generate the error term ϵ in (b). Describe your results.

We increase the standard deviation of the error terms or noise to 1 from 0.25.

```
eps = generate_data(mean=0.0, sd = 1, N = 100);
```

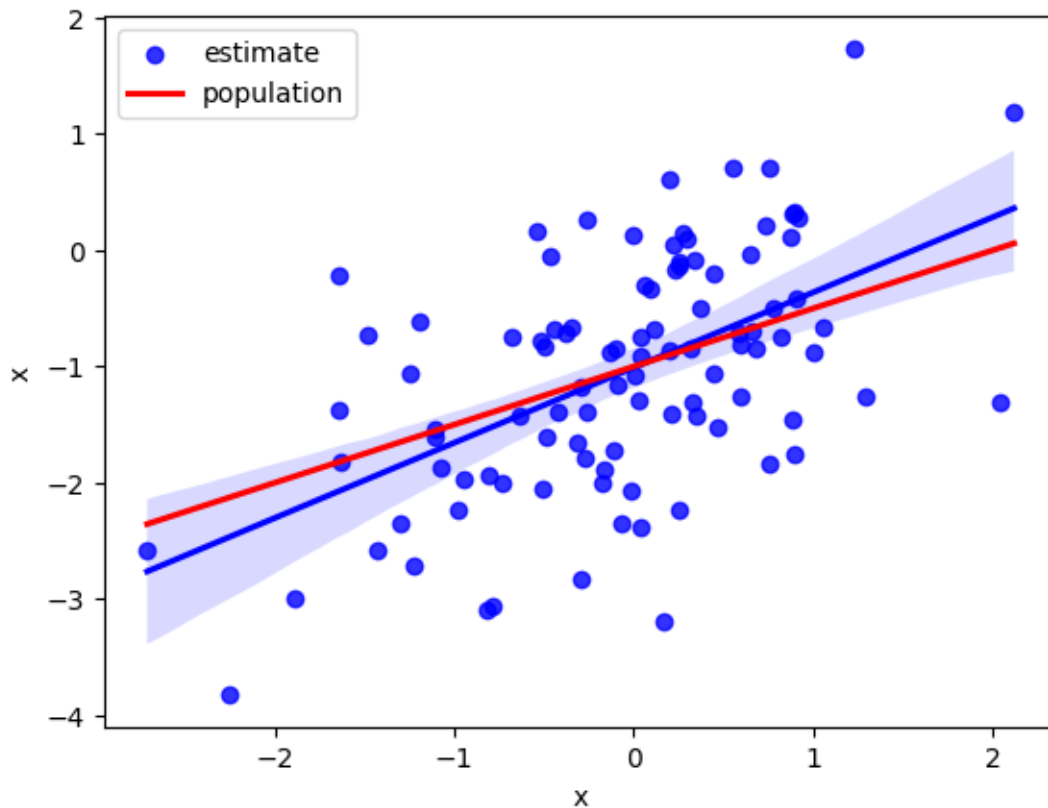
```
y = -1 + 0.5 * x + eps;
```

```
df = pd.DataFrame({"x": x, "y": y});
```

```
noisier_results, result_df = regress_y_on_x(df)
result_df
```

	coefficients	coefficients-se	tstatistic	p-value	r-squared	r-squared-adjusted	pearson_
Intercept	-1.009159	0.087455	-11.539160	5.766671e-20	0.289775	0.282528	0.538307
x	0.646994	0.102319	6.323323	7.629976e-09	0.289775	0.282528	0.538307

```
draw_regplot(df)
```



- We conclude that the noisier the data, the more the drift of the fit from the population parameters especially when it comes to estimating the slope of the fitted line.

(j) What are the confidence intervals for β_0 and β_1 based on the original data set, the noisier data set, and the less noisy data set? Comment on your results.

```
print("Original dataset")
print(orig_res.conf_int(alpha=0.05))
print()
print("Less noisy dataset")
print(less_noisier_results.conf_int(alpha=0.05))
print()
print("More noisy dataset")
print(noisier_results.conf_int(alpha=0.05))
```

Original dataset

	0	1
Intercept	-1.068891	-0.969122
x	0.433783	0.550508

Less noisy dataset

	0	1
Intercept	-1.017282	-0.998911
x	0.488614	0.510108

More noisy dataset

	0	1
Intercept	-1.182711	-0.835607
x	0.443946	0.850043

```
print("Standard errors of coefficients")
print()
print("Original dataset")
print(orig_res.bse)
print()
print("Less noisy dataset")
print(less_noisier_results.bse)
print()
print("More noisy dataset")
print(noisier_results.bse)
```

Standard errors of coefficients

```
Original dataset
Intercept    0.025138
x            0.029410
dtype: float64
```

```
Less noisy dataset
Intercept    0.004629
x            0.005415
dtype: float64
```

```
More noisy dataset
Intercept    0.087455
x            0.102319
dtype: float64
```

- We can conclude that the noisier the dataset, the more likely that the confidence intervals are wider so that the population parameters actually reside within its range.
- This is because the Standard errors of the parameters are wider when the data is noisier.

```
allDone();
```

```
<IPython.lib.display.Audio object>
```