What is regression?

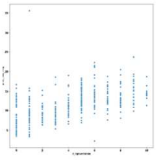
- Statistical models to explore the relationship between a response variable and some explanatory variables.
- Given values of explanatory variables, you can predict the values of the response variable.
- **Response variable** (a.k.a. dependent variable) The variable that you want to predict.
- **Explanatory variables** (a.k.a. independent variables) The variables that explain how the response variable will change.

Visualizing two numeric variables
Before you can run any statistical models, it's usually a good idea to visualize your dataset. Here, you'll look at the relationship between house price per area and the number of nearby convenience stores using the Taiwan real estate dataset.

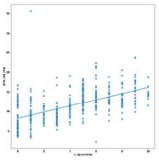
One challenge in this dataset is that the number of convenience stores contains integer data, causing points to overlap. To solve this, you will make the points transparent.

taiwan_real_estate is available from Taiwan_real_estate2.csv

- Import the seaborn and matplotlib packages
- Using taiwan_real_estate, draw a scatter plot of "price_twd_msq" (y-axis) versus "n_convenience" (x-axis).



- Draw a trend line calculated using linear regression. Omit the confidence interval ribbon. Note: The scatter_kws argument in regplot, scatter_kws={'alpha': 0.5}, makes the data points 50% transparent.



Transforming variables

- Sometimes, the relationship between the explanatory variable and the response variable may not be a straight line. To fit a linear regression model, you may need to transform the explanatory variable or the response variable, or both of them.

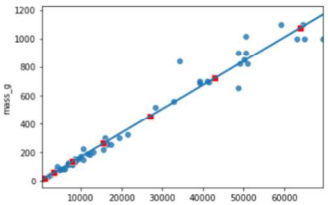
Modeling mass vs. length cubed

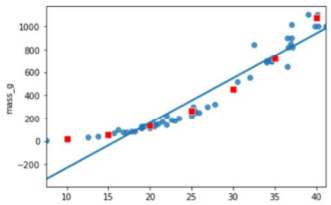
```
perch["length_cm_cubed"] = perch["length_cm"] ** 3

mdl_perch = ols("mass_g ~ length_cm_cubed", data=perch).fit()
mdl_perch.params
```

```
Intercept         -0.117478
length_cm_cubed    0.016796
dtype: float64
```

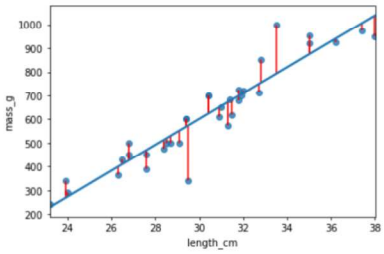
```
fig = plt.figure()
sns.regplot(x="length_cm_cubed", y="mass_g",
            data=perch, ci=None)
sns.scatterplot(data=prediction_data,
                x="length_cm_cubed", y="mass_g",
                color="red", marker="s")
```

```
fig = plt.figure()
sns.regplot(x="length_cm", y="mass_g",
            data=perch, ci=None)
sns.scatterplot(data=prediction_data,
                x="length_cm", y="mass_g",
                color="red", marker="s")
```





Residual standard error (RSE)



- A "typical" difference between a prediction and an observed response
- It has the same unit as the response variable.
- MSE = RSE²

Using scikit-learn to fit a classifier

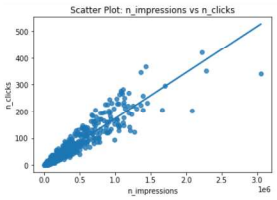
```
from sklearn.neighbors import KNeighborsClassifier
X = churn_df[["total_day_charge", "total_eve_charge"]].values
y = churn_df["churn"].values
print(X.shape, y.shape)
```

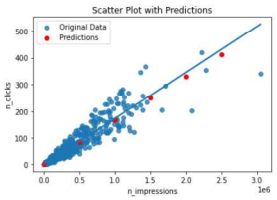
```
(3333, 2), (3333,)
```

```
knn = KNeighborsClassifier(n_neighbors=15)
knn.fit(X, y)
```

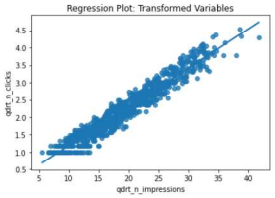
```
#Lut Lat Aung 6511163

import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from statsmodels.formula.api import ols

ad_conversion = pd.read_csv("ad_conversion.csv")

sns.regplot(x="n_impressions", y="n_clicks", data=ad_conversion, ci=None)
plt.title("Scatter Plot: n_impressions vs n_clicks")
plt.xlabel("n_impressions")
plt.ylabel("n_clicks")
plt.show()

ad_conversion["qdrt_n_impressions"] = ad_conversion["n_impressions"] ** 0.25
ad_conversion["qdrt_n_clicks"] = ad_conversion["n_clicks"] ** 0.25
sns.regplot(x="qdrt_n_impressions", y="qdrt_n_clicks", data=ad_conversion, ci=None)
plt.title("Regression Plot: Transformed Variables")
plt.xlabel("qdrt_n_impressions")
plt.ylabel("qdrt_n_clicks")
plt.show()

# Do the points track the line more closely? Yes.

mdl_click_vs_impression = ols("qdrt_n_clicks ~ qdrt_n_impressions", data=ad_conversion).fit()
print("Model Parameters:")
print(mdl_click_vs_impression.params)

explanatory_data = pd.DataFrame({
    "qdrt_n_impressions": np.arange(0, 3000000, 500000) ** 0.25,
    "n_impressions": np.arange(0, 3000000, 500000)
})
prediction_data = explanatory_data.assign(
    qdrt_n_clicks=mdl_click_vs_impression.predict(explanatory_data)
)
```

```
print("Prediction Data:")
print(prediction_data)

# Back transformation:
prediction_data["n_clicks"] = prediction_data["qdrt_n_clicks"] ** 4
sns.regplot(x="n_impressions", y="n_clicks", data=ad_conversion, ci=None, label="Original Data")
plt.scatter(
    x=prediction_data["n_impressions"],
    y=prediction_data["n_clicks"],
    color="red",
    label="Predictions",
)
plt.title("Scatter Plot with Predictions")
plt.xlabel("n_impressions")
plt.ylabel("n_clicks")
plt.legend()
plt.show()


r_squared = mdl_click_vs_impression.rsquared
print("R-squared:", r_squared)
residuals = ad_conversion["qdrt_n_clicks"] - mdl_click_vs_impression.fittedvalues
rse = np.sqrt(np.sum(residuals ** 2) / (len(ad_conversion) - 2))
print("RSE:", rse)

# Interpret the result:
# - A higher R-squared value indicates a better fit, with 1.0 being a perfect fit.
# - RSE measures the standard deviation of the residuals, indicating how well the model fits the data.
# - A lower RSE suggests a better fit, as it means the model's predictions are closer to the actual values.
```





R-squared: 0.9445272817143905
RSE: 0.19690648896875727




```
Model Parameters:
Intercept             0.071748
qdrt_n_impressions    0.111533
dtype: float64
Prediction Data:
   qdrt_n_impressions  n_impressions  qdrt_n_clicks
0            0.000000              0       0.071748
```

scikit-learn syntax

```
from sklearn.module import Model
model = Model()
model.fit(X, y)
predictions = model.predict(X_new)
print(predictions)
```

```
array([0, 0, 0, 0, 1, 0])
```

Classifying labels of unseen data

1. Build a model
2. Model learns from the labeled data we pass to it
3. Pass unlabeled data to the model as input
4. Model predicts the labels of the unseen data

- Labeled data = training data

k-Nearest Neighbors

- Predict the label of a data point by
  - Looking at the `k` closest labeled data points
  - Taking a majority vote