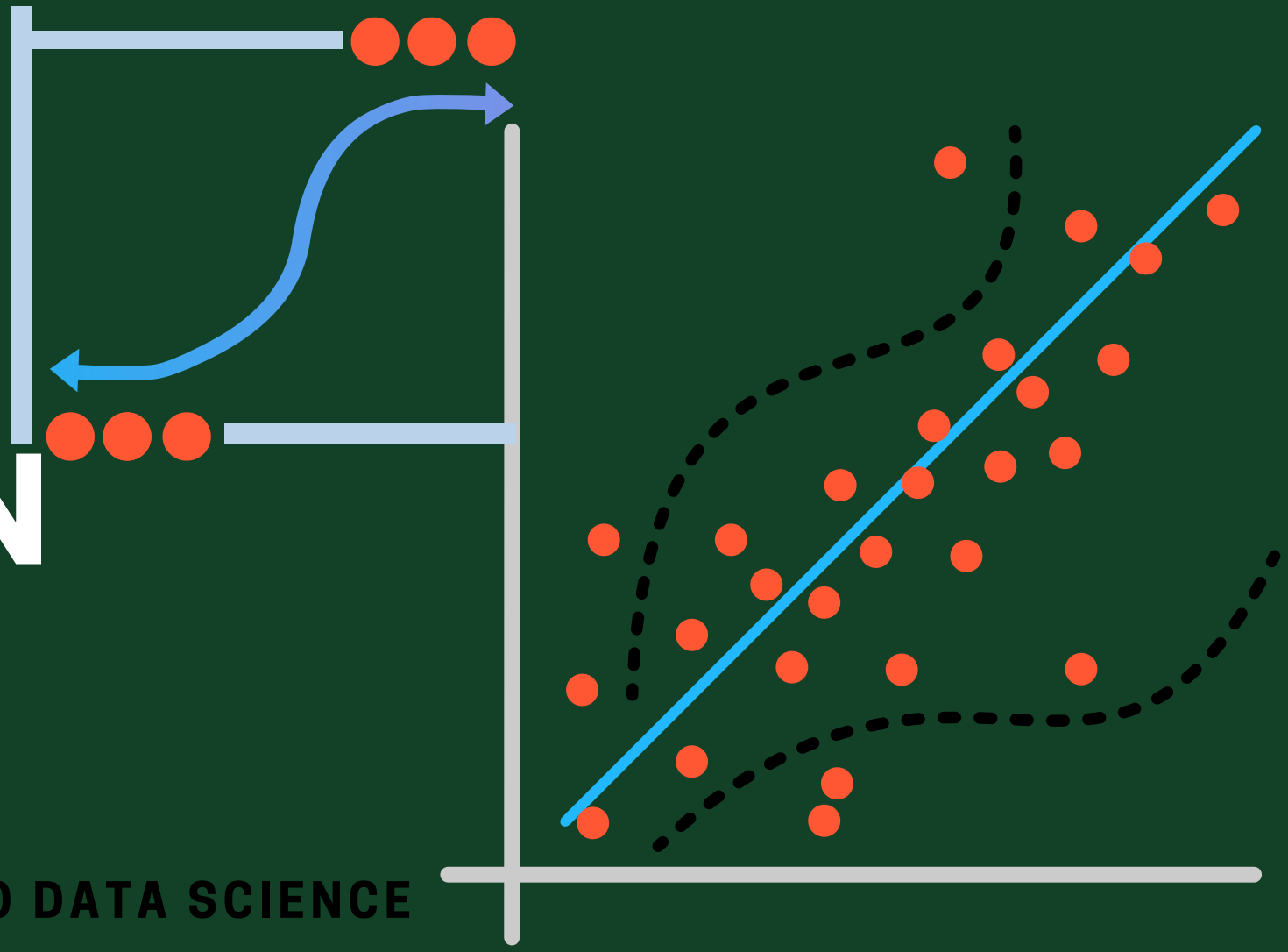
WELLCOME

REGRESSION ANALYSIS



EMBARKING ON A JOURNEY INTO DATA SCIENCE

YA MANON

ABOUT THIS SERIES

PART 1 → **PART 2** → **PART 3**

Data Prep & EDA | Regression | Classification

Recall

LINEAR RELATIONSHIPS

Linear Relationships

It's common for numeric variables to have **linear relationships** between them

- When one variable changes, so does the other
- This relationship is commonly visualized with a **scatterplot**

EXAMPLE *Relationship between age and charges*

```
data[['age', 'charges']]
```

	age	charges
0	19	16884.92400
1	18	1725.55230
2	28	4449.46200
3	33	21984.47061
4	32	3866.85520
...	...	...
1333	50	10600.54830

```
sns.scatterplot(x = data['age'], y = data['charges'], hue = data['sex'])
sns.despine()
plt.show()
```

(this is a positive relationship)

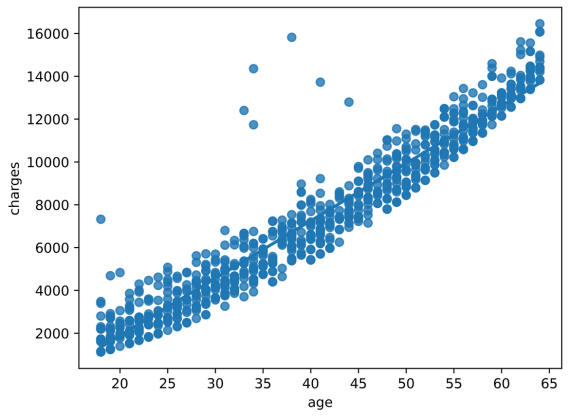
LINEAR RELATIONSHIPS

Linear Relationships

There are two possible linear relationships: **positive** & **negative**

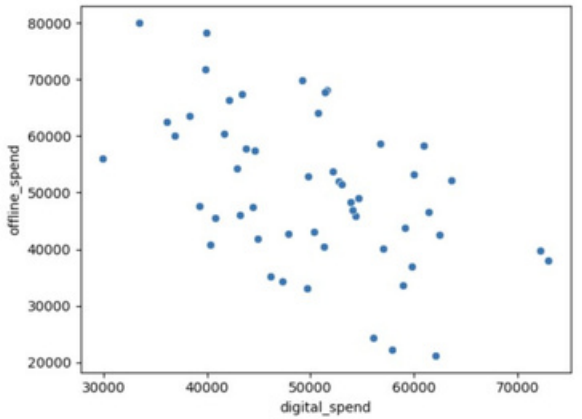
- Variables can also have **no relationship**

Positive Relationship



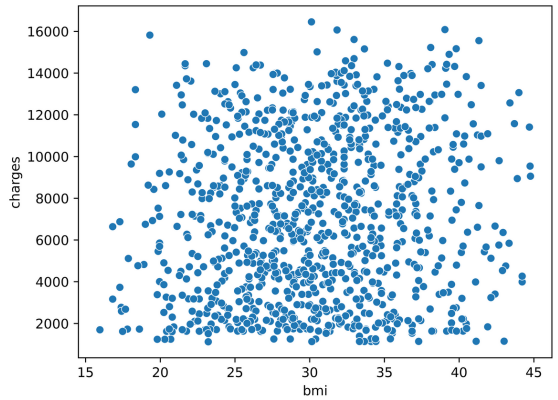
*As one changes, the other changes in the **same direction***

Negative Relationship



*As one changes, the other changes in the **opposite direction***

No Relationship



No association can be found between the changes in one variable and the other

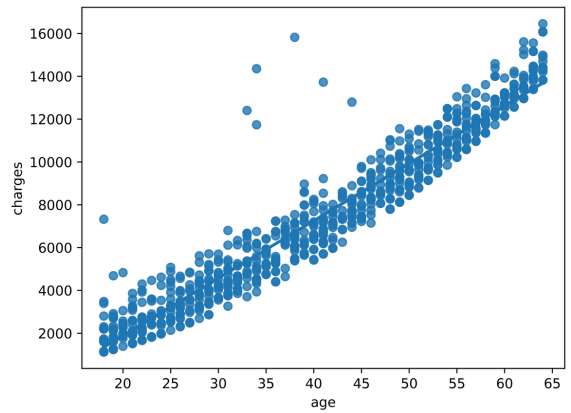
CORRELATION

The **correlation (r)** measures the strength & direction of a linear relationship *(-1 to 1)*

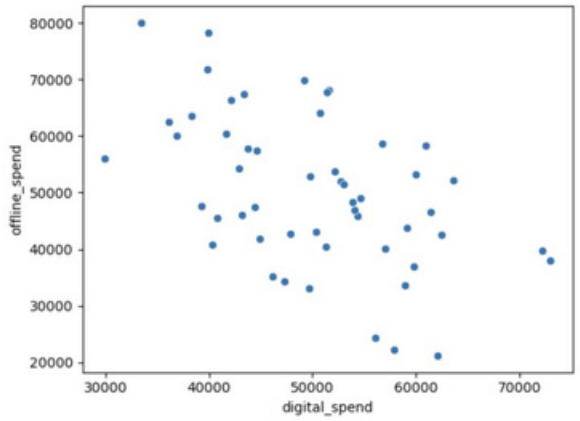
- You can use the **.corr()** method to calculate correlations in Pandas – df[“col1”].corr(df[“col2”])

Linear Relationships

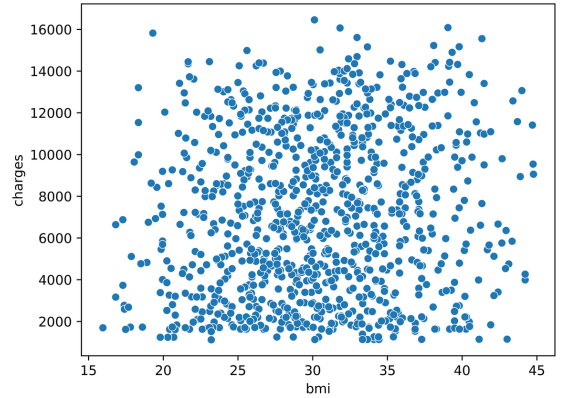
r = 0.858

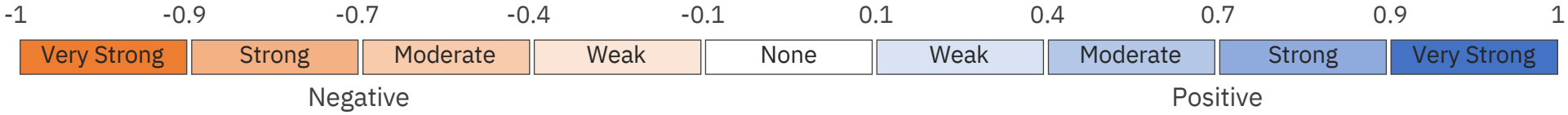


r= -0.499



r= 0.008





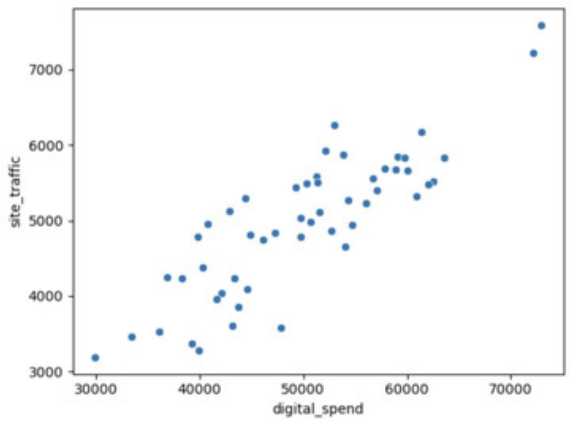
CORRELATION

Linear Relationships

The **correlation (r)** measures the strength & direction of a linear relationship *(-1 to 1)*

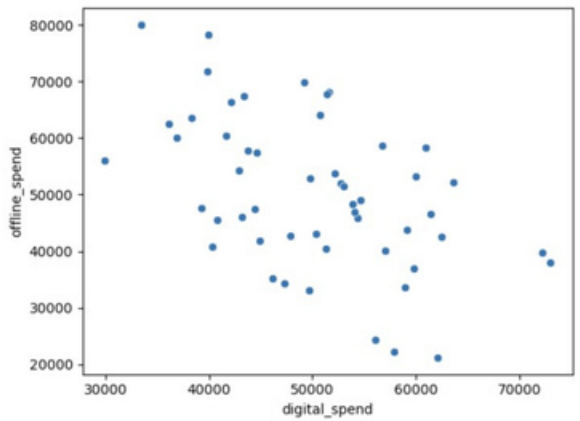
- You can use the **.corr()** method to calculate correlations in Pandas – df[“col1”].corr(df[“col2”])

r = 0.858



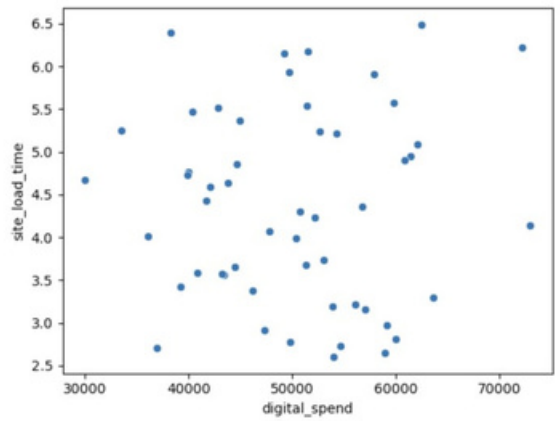
Strong positive correlation

r= -0.499



Moderate negative correlation

r= 0.008



No correlation

PRO TIP: Highly correlated variables tend to be the best candidates for your “baseline” regression but other variables can still be useful features after exploring non-linear relationships and fixing data issues

PRO TIP: CORRELATION MATRIX

A **correlation matrix** returns the correlation between each column in a DataFrame

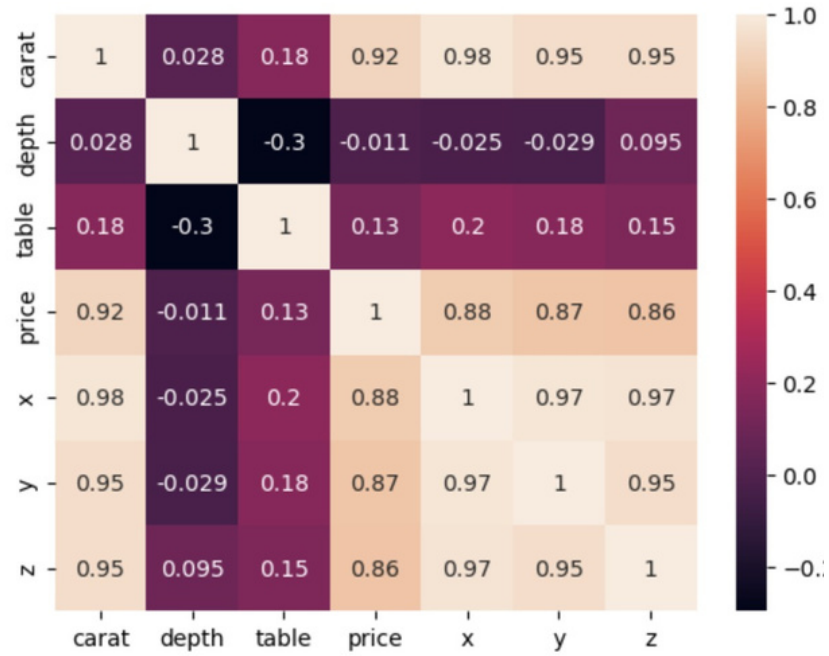
- Use **df.corr(numeric_only=True)** to only consider numeric columns in the
- DataFrame Wrap you correlation matrix in **sns.heatmap()** to make it easier to interpret

Linear Relationships

```
diamonds.corr(numeric_only=True)
```

	carat	depth	table	price	x	y	z
carat	1.000000	0.028234	0.181602	0.921591	0.975093	0.951721	0.953387
depth	0.028234	1.000000	-0.295798	-0.010630	-0.025289	-0.029340	0.094927
table	0.181602	-0.295798	1.000000	0.127118	0.195333	0.183750	0.150915
price	0.921591	-0.010630	0.127118	1.000000	0.884433	0.865419	0.861249
x	0.975093	-0.025289	0.195333	0.884433	1.000000	0.974701	0.970771
y	0.951721	-0.029340	0.183750	0.865419	0.974701	1.000000	0.952005
z	0.953387	0.094927	0.150915	0.861249	0.970771	0.952005	1.000000

```
sns.heatmap(diamonds.corr(numeric_only=True), annot=True);
```



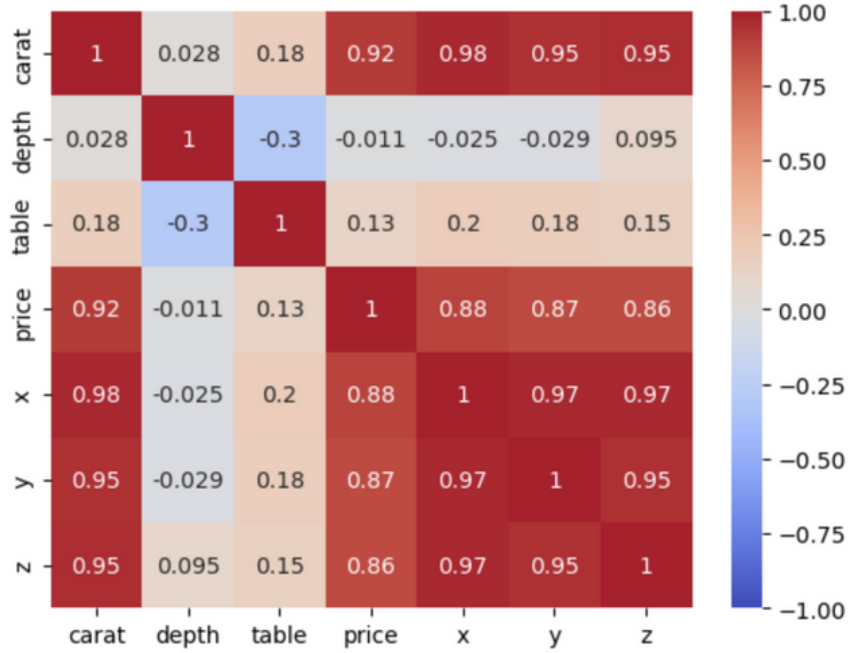
PRO TIP: CORRELATION MATRIX

A **correlation matrix** returns the correlation between each column in a DataFrame

- Use **df.corr(numeric_only=True)** to only consider numeric columns in the
- DataFrame Wrap you correlation matrix in **sns.heatmap()** to make it easier to interpret

Linear Relationships

```
sns.heatmap(diamonds.corr(numeric_only=True), annot=True, vmin=-1, vmax=1, cmap="coolwarm");
```

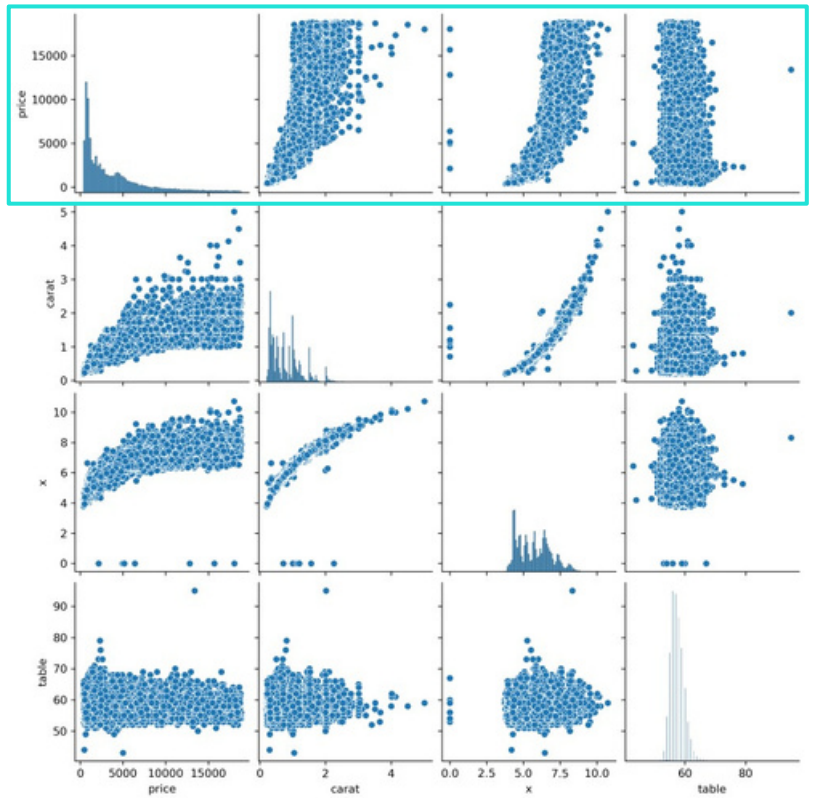


Setting the maximum and minimum values at -1 and 1 respectively, and adding a divergent color scale can help increase interpretability

TIP: PAIRPLOTS

Use **sns.pairplot()** to create a pairplot that shows all the scatterplots and histograms that can be made using the numeric variables in a DataFrame

```
sns.pairplot(diamonds);
```



*The row with your **target** can help explore numeric feature-target relationships quickly!*

Linear Relationships

PRO TIP: It can take a long time to generate pairplots for large datasets, so consider using **df.sample()** to speed up the process significantly without losing high-level insights!

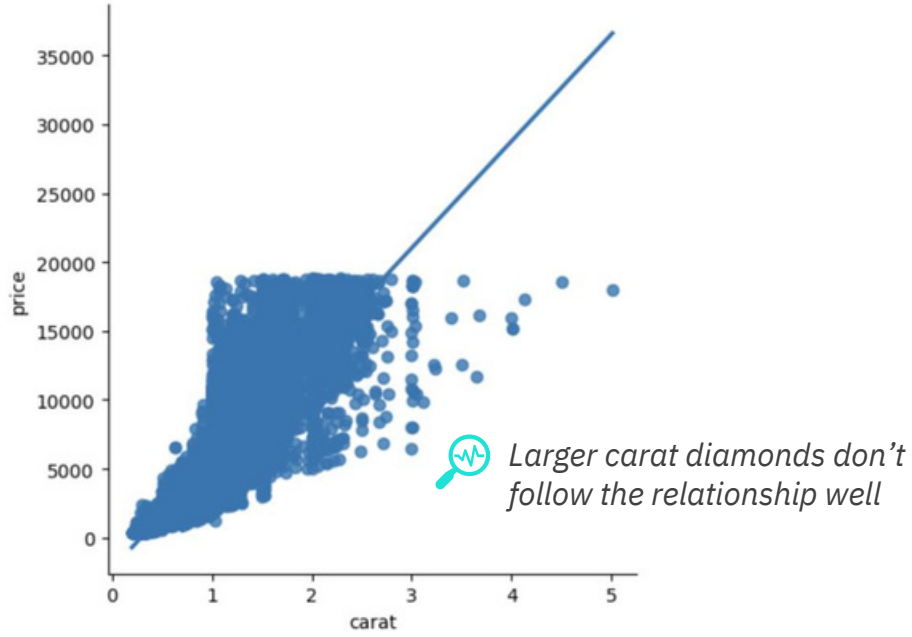
TIP: LMPLOTS

Use **sns.lmplot()** to create a scatterplot with a fitted regression line *(more soon!)*

- This is commonly used to explore the impact of other variables on a linear relationship
  - **sns.lmplot(df, x="feature", y="target", hue="categorical feature")**

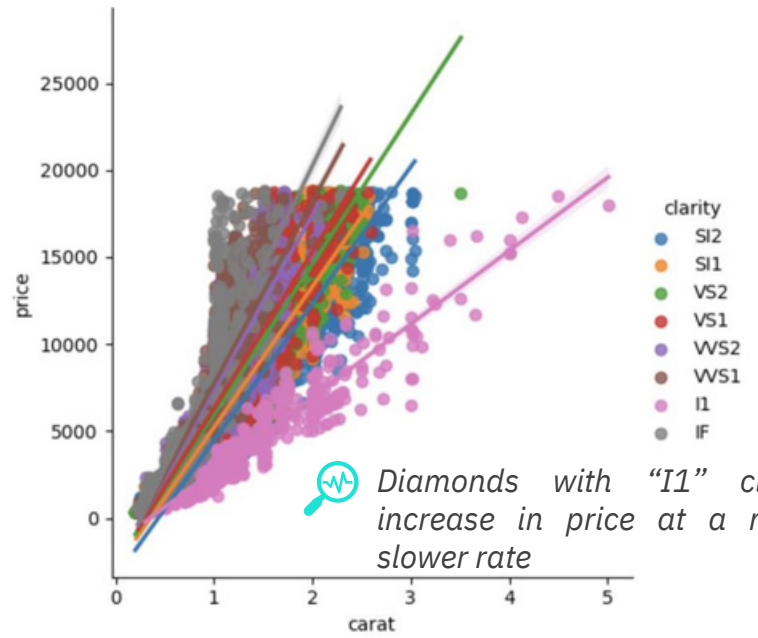
Linear Relationships

```
sns.lmplot(diamonds, x="carat", y="price");
```



Larger carat diamonds don't follow the relationship well

```
sns.lmplot(diamonds, x="carat", y="price", hue="clarity");
```



Diamonds with "I1" clarity increase in price at a much slower rate

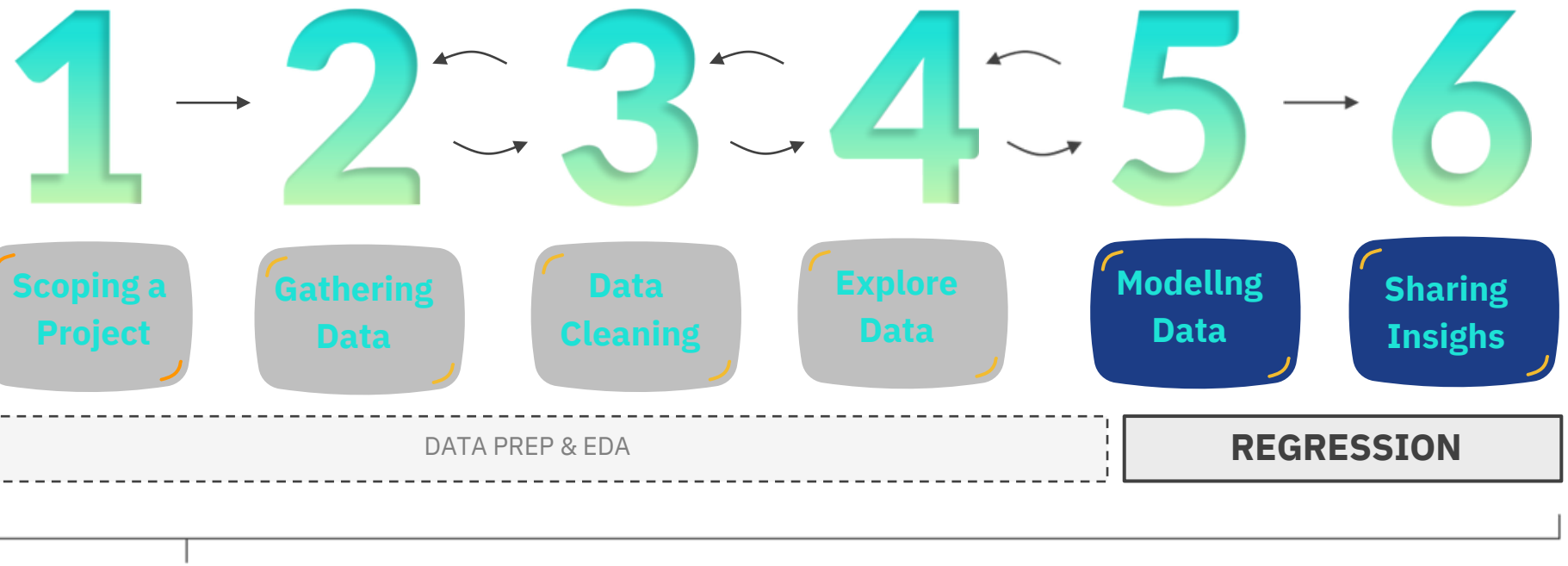
DATA SCIENCE WORKFLOW

Data Science Workflow

The **data science workflow** consists of scoping the project, gathering, cleaning and exploring the data, applying models, and sharing insights with end users

This is not a linear process! *You'll likely go back to further gather, clean and explore your data*

REGRESSION MODELING



Data Science Workflow

Regression models are used to predict the value of numeric variables

Even though regression models fall into the final two steps of the workflow, data prep & EDA should always come first to help you get the most out of your models

REGRESSION

GOALS OF REGRESSION

Goals of Regression

Regression models are used for two primary goals: **prediction** and **inference**

The goal shapes the modeling approach, including the regression algorithm used, the complexity of the model, and more

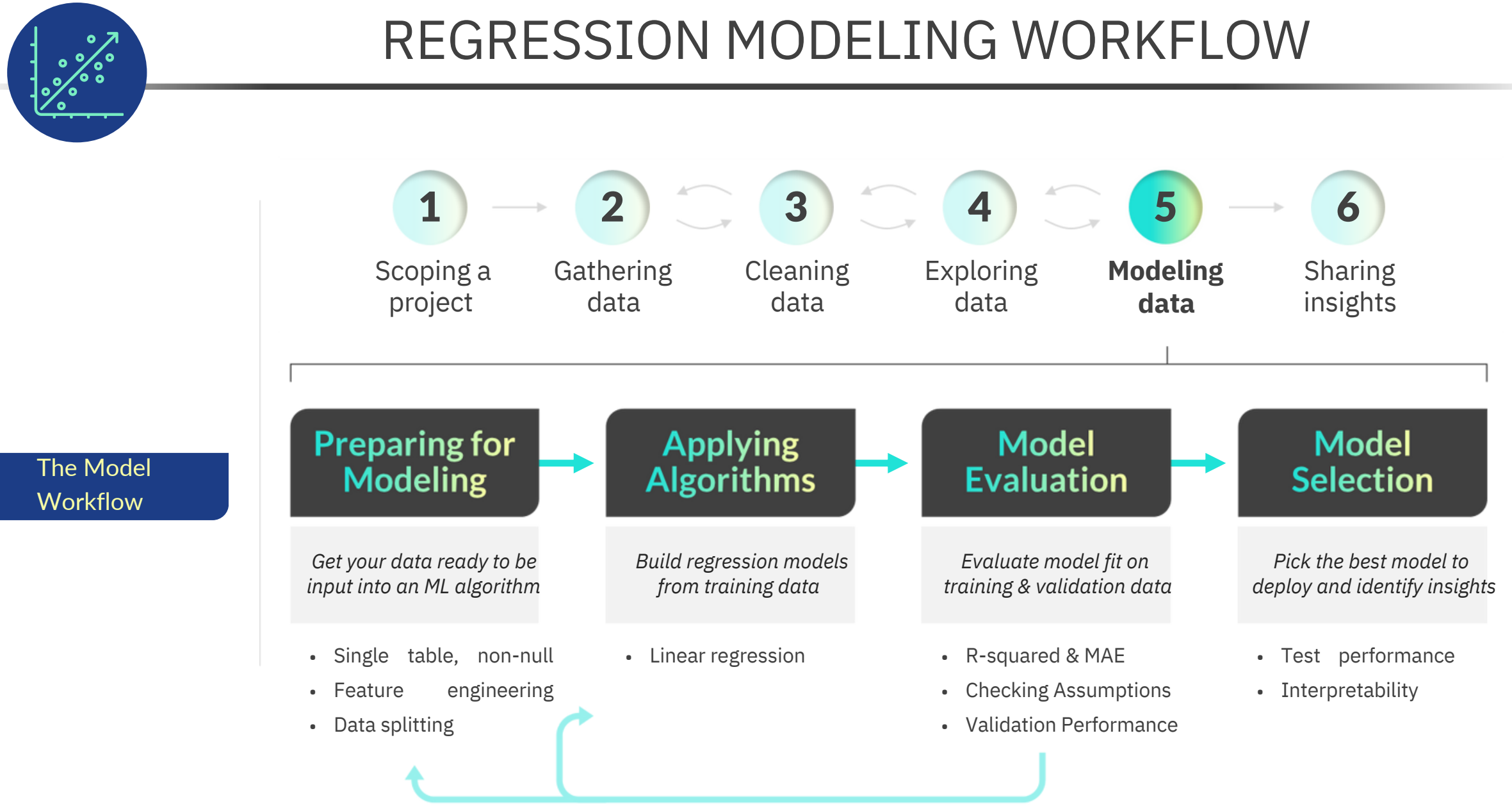
PREDICTION

- Used to **predict** the target as accurately as possible
- *"What is the predict charges for a client given their age?"*

INFERENCE

- Used to **understand the relationships** between the features and target
- *"How much do a age and smoker impact its charges?"*

REGRESSION MODELING WORKFLOW



SIMPLE LINEAR REGRESSION

LINEAR REGRESSION MODEL

- Linear Regression Model
- Least Squared Error
- Regression in Python
- Making Predictions
- Evaluation Metrics

The **linear regression model** is an equation that best describes a linear relationship

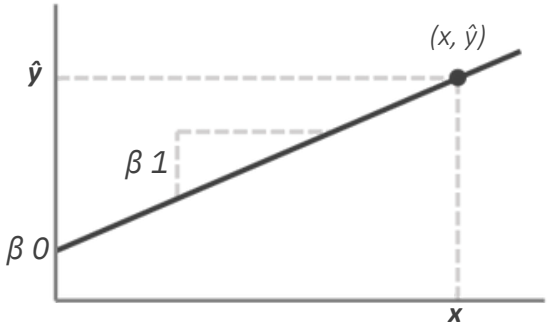
*The **predicted** value for the target*

The value for the feature

$$y = \beta_0 + \beta_1 x$$

*The **y-intercept***

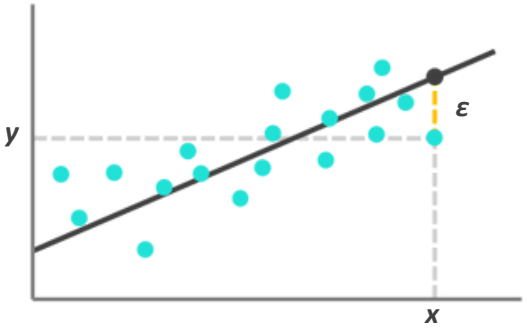
*The **slope** of the relationship*

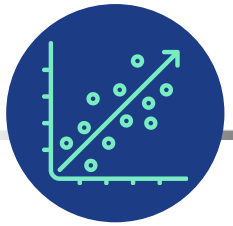


*The **actual** value for the target*

$$y = \beta_0 + \beta_1 x + \epsilon$$

*The **error**, or **residual**, caused by the difference between the actual and predicted values*





LEAST SQUARED ERROR

The **least squared error** method finds the line that best fits through the data

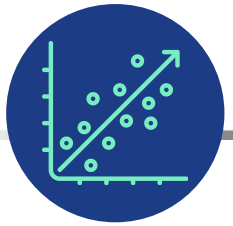
- It works by solving for the line that **minimizes the sum of squared error**
- The equation that minimizes error can be solved with **linear algebra**

Why "squared" error?

- Squaring the residuals converts them into **positive values**, and prevents positive and negative distances from cancelling each other out *(this makes the algebra to solve the line much easier, too)*
- One drawback of squared errors is that outliers can significantly impact the line *(more later!)*

Ordinary Least Squares (OLS) is another term for traditional linear regression

There are other frameworks for linear regression that don't use least squared error, but they are rarely used outside of specialized domains



REGRESSION IN PYTHON

- Linear Regression Model
- Least Squared Error
- Regression in Python
- Making Predictions
- Evaluation Metrics

These Python libraries are used fit regression models: **statsmodels** & **scikit-learn**

statsmodels

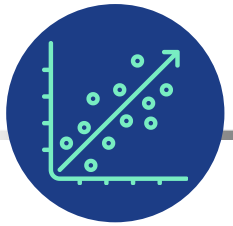
- **Ideal if your goal is inference**
- Similar output to other tools (SAS, R, Excel)
- Easy access to dozens of statistical tests
- Harder to leverage in production ML

learn

- **Ideal if your goal is prediction**
- Most popular ML library in Python
- Has various models for easy
- comparison Designed to be deployed to production

Both libraries **use the same math** and return the same regression equation!

We will begin by focusing on statsmodels, but once we have the fundamentals of regression down, we'll introduce scikit-learn



REGRESSION IN STATSMODELS

- Linear Regression Model
- Least Squared Error
- Regression in Python
- Making Predictions
- Evaluation Metrics

You can fit a **regression in statsmodels** with just a few lines of code:

```
import statsmodels.api as sm

x_train_con = sm.add_constant(x_train)
model = sm.OLS(y_train,x_train_con).fit()
model.summary()
```

*1) Import **statsmodels.api** (standard alias is **sm**)*

*2) Create an "X" DataFrame with your **feature(s)** and **add a constant***

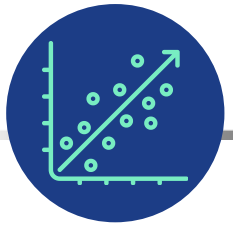
*3) Create a "y" DataFrame with your **target***

*4) Call **sm.OLS(y, X)** to set up the model, then use **.fit()** to build the model*

*5) Call **.summary()** on the model to review the model output*

Why do we need to add a constant?

- Statsmodels assumes you want to fit a model with a line that runs through the origin (0, 0)
- sm.add_constant() lets statsmodels calculate a y-intercept other than 0 for the model
- Most regression software *(like sklearn)* takes care of this step behind the scenes



REGRESSION IN STATSMODELS

- Linear Regression Model
- Least Squared Error
- Regression in Python
- Making Predictions
- Evaluation Metrics

You can fit a **regression in statsmodels** with just a few lines of code:

```
import statsmodels.api as sm

x_train_con = sm.add_constant(x_train)
model = sm.OLS(y_train,x_train_con).fit()
model.summary()
```

OLS Regression Results

Dep. Variable:	charges	**R-squared:**	0.918
Model:	OLS	**Adj. R-squared:**	0.918
Method:	Least Squares	**F-statistic:**	9714.
Date:	Sat, 30 Mar 2024	**Prob (F-statistic):**	0.00
Time:	22:43:12	**Log-Likelihood:**	-7288.4
No. Observations:	864	**AIC:**	1.458e+04
Df Residuals:	862	**BIC:**	1.459e+04
Df Model:	1		
Covariance Type:	nonrobust		

Model summary statistics

	coef	std err	t	P>|t|	[0.025	0.975]
const	-3400.3622	112.847	-30.133	0.000	-3621.849	-3178.876
age	266.8456	2.707	98.558	0.000	261.532	272.160

Variable summary statistics

Omnibus:	707.070	**Durbin-Watson:**	1.973
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	24364.367
Skew:	3.460	**Prob(JB):**	0.00
Kurtosis:	28.078	**Cond. No.**	124.

Residual (error) statistics

The model output can be intimidating the first time you see it, but we'll cover the important pieces in the next few lessons and later sections!

INTERPRETING THE MODEL

- Linear Regression Model
- Least Squared Error
- Regression in Python
- Making Predictions
- Evaluation Metrics

To **interpret the model**, use the "coef" column in the variable summary statistics

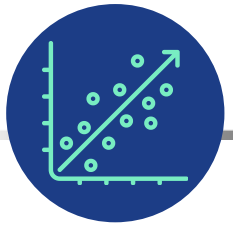
```
import statsmodels.api as sm

x_train_con = sm.add_constant(x_train)
model = sm.OLS(y_train,x_train_con).fit()
model.summary()
```

	coef	std err	t	P>|t|	[0.025	0.975]
const	-3400.3622	112.847	-30.133	0.000	-3621.849	-3178.876
age	266.8456	2.707	98.558	0.000	261.532	272.160

$\hat{y} = -3400 + 266.8456x$

How do we interpret this?

- Technically, Intercept (-3400): When x (the age of the client) is 0, the predicted outcome y_pre is -3400.



MAKING PREDICTIONS

- Linear Regression Model
- Least Squared Error
- Regression in Python
- Making Predictions
- Evaluation Metrics

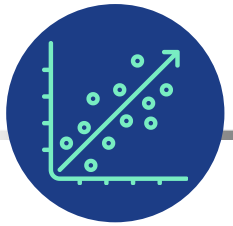
The **.predict()** method returns model predictions for single points or DataFrames

```
import statsmodels.api as sm

x_train_con = sm.add_constant(x_train)
model = sm.OLS(y_train,x_train_con).fit()
model.summary()
```

	coef	std err	t	P>|t|	[0.025	0.975]
const	-3400.3622	112.847	-30.133	0.000	-3621.849	-3178.876
age	266.8456	2.707	98.558	0.000	261.532	272.160

$$\hat{y} = -3400 + 266.8456x$$

Age 45 predicted chnarges of $?



R-SQUARED

- Linear Regression Model
- Least Squared Error
- Regression in Python
- Making Predictions
- Evaluation Metrics

R-squared, or coefficient of determination, measures how much better the model is at predicting the target than using its mean *(our best guess without using features)*

- R-squared values are bounded between 0 and 1 on training data

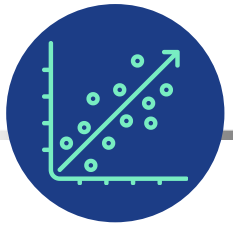
```
import statsmodels.api as sm

x_train_con = sm.add_constant(x_train)
model = sm.OLS(y_train,x_train_con).fit()
model.summary()
```

***Squaring the correlation** between the feature and target yields R2 in simple linear regression (this doesn't hold in multiple linear regression)*

OLS Regression Results

Dep. Variable:	charges	**R-squared:**	0.918
Model:	OLS	**Adj. R-squared:**	0.918
Method:	Least Squares	**F-statistic:**	9714.
Date:	Sat, 30 Mar 2024	**Prob (F-statistic):**	0.00
Time:	22:43:12	**Log-Likelihood:**	-7288.4
No. Observations:	864	**AIC:**	1.458e+04
Df Residuals:	862	**BIC:**	1.459e+04
Df Model:	1		
Covariance Type:	nonrobust		

*The model explains 91**.9%** of the variation in price not explained by the mean of charges*



R-SQUARED

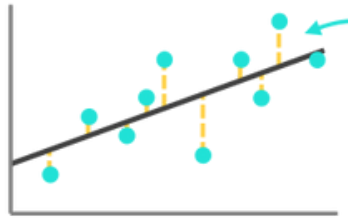
- Linear Regression Model
- Least Squared Error
- Regression in Python
- Making Predictions
- Evaluation Metrics

R-squared, or coefficient of determination, measures how much better the model is at predicting the target than using its mean *(our best guess without using features)*

R-squared values are bounded between 0 and 1 on training data

$$R2 = 1 - \frac{SSE}{SST}$$

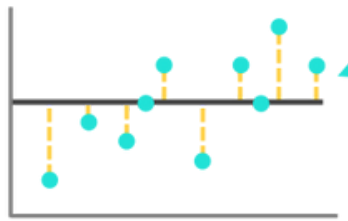
*This is the **sum of squared error** (distance between values & line)*



This is the variation of "y" not explained by "x"

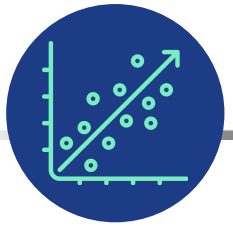
- *Reducing error will increase R2*
- *A perfect model has an error of 0, or R2 of 1*

*This is the **sum of squared total** (distance between values & mean)*



This is the variation of "y" around its mean

- *If R2 = 0, the model is no better than the mean of y*



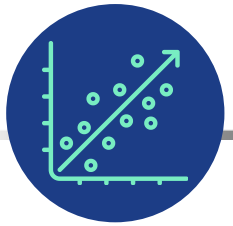
HYPOTHESIS TEST

Regression models include several **hypothesis tests**, including the F-test, that indicates whether our model is significantly better at predicting our target than using the mean of the target as the model

In other words, you're trying to find significant evidence that your model isn't useless

Steps for the hypothesis test:

1) State the **null** and **alternative hypotheses**
2) Set a **significance level** (α)
3) Calculate the **test statistic** and **pvalue**
4) Draw a **conclusion** from the test
    a) If $p \leq \alpha$, reject the null hypothesis *(you're confident the model isn't useless)*
    b) If $p > \alpha$, don't reject it *(the model is probably useless, and needs more training)*



HYPOTHESES & SIGNIFICANCE LEVEL

1)For F-Tests, the **null & alternative hypotheses** are always the same:

- **Ho: F=0** – The model has no special effect, meaning it's just as good as guessing the average.
- **Ha: F≠0** – The model does have a special effect,This means the model helps us make more accurate predictions compared to if we only used the average value as our guess.

*The hope is to **reject the null hypothesis***
(and therefore, accept the alternative)

2)The **significance level** is the threshold you set to determine when the evidence against your null hypothesis is considered "strong enough" to prove it wrong **α**

- This is set by **alpha ()**, which is the accepted probability of error
- The industry standard is **α= .05** *(this is what we'll use in the course)*

Some teams and industries set a much higher bar, such as .01 or even .001, making the null hypothesis **less likely to be rejected**

F-STATISTIC & P-VALUE

- Linear Regression Model
- Least Squared Error
- Regression in Python
- Making Predictions
- Evaluation Metrics

3)The **F-statistic** and associated **p-value** are part of the model summary and help understand the predictive power of the regression model *as a whole*

- The **F-statistic** is the ratio of variability the model explains vs the variability it doesn't
- The **p-value**, or F-significance, is the probability that your model predicts poorly

```
import statsmodels.api as sm

x_train_con = sm.add_constant(x_train)
model = sm.OLS(y_train,x_train_con).fit()
model.summary()
```

OLS Regression Results

Dep. Variable:	charges	**R-squared:**	0.918
Model:	OLS	**Adj. R-squared:**	0.918
Method:	Least Squares	**F-statistic:**	9714.
Date:	Sat, 30 Mar 2024	**Prob (F-statistic):**	0.00
Time:	22:43:12	**Log-Likelihood:**	-7288.4
No. Observations:	864	**AIC:**	1.458e+04
Df Residuals:	862	**BIC:**	1.459e+04
Df Model:	1		
Covariance Type:	nonrobust		

The F-statistic is primarily used as a stepping-stone to calculate the p-value, which is **easier to interpret** and **more commonly used** in model diagnostics

HYPOTHESIS TEST CONCLUSION

- Linear Regression Model
- Least Squared Error
- Regression in Python
- Making Predictions
- Evaluation Metrics

4)Comparing the p-value and alpha lets us draw a **conclusion** from the test **α**

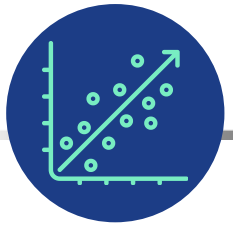
- **p≤** α reject the null hypothesis *(you're confident the model isn't useless)*
- **p>= α** don't reject it *(the model is probably useless, and needs more training)*

```
import statsmodels.api as sm

x_train_con = sm.add_constant(x_train)
model = sm.OLS(y_train,x_train_con).fit()
model.summary()
```

OLS Regression Results

Dep. Variable:	charges	**R-squared:**	0.918
Model:	OLS	**Adj. R-squared:**	0.918
Method:	Least Squares	**F-statistic:**	9714.
Date:	Sat, 30 Mar 2024	**Prob (F-statistic):**	0.00
Time:	22:43:12	**Log-Likelihood:**	-7288.4
No. Observations:	864	**AIC:**	1.458e+04
Df Residuals:	862	**BIC:**	1.459e+04
Df Model:	1		
Covariance Type:	nonrobust		

Our F-statistic is much greater than 0, and the p-value is less than 0.05, so we can reject the null hypothesis (age is a good predictor of a charges price!)



T-STATISTICS & P-VALUES

The **T-statistics** and associated **p-values** are part of the model summary and help understand the predictive power of *individual model coefficients*

- It's essentially another hypothesis test designed to find which coefficients are useful

```
import statsmodels.api as sm

x_train_con = sm.add_constant(x_train)
model = sm.OLS(y_train,x_train_con).fit()
model.summary()
```

	coef	std err	t	P>|t|	[0.025	0.975]
const	-3400.3622	112.847	-30.133	0.000	-3621.849	-3178.876
age	266.8456	2.707	98.558	0.000	261.532	272.160

Since the p-value is lower than our alpha of 0.05, we can conclude that age is a good predictor of charges

(the constant also has a p-value lower than 0.05, but we can generally ignore insignificant p-values for the intercept term)

This will become more relevant when performing **variable selection** in multiple linear regression models (up next!)

MULTIPLE LIEANR REGRESSION

MULTIPLE LINEAR REGRESSION

In this section we'll build **multiple linear regression** models using more than one feature, evaluate the model fit, perform variable selection, and compare models using error metrics

TOPICS WE'LL COVER:

- Multiple Regression
- Variable Selection
- Mean Error Metrics

GOALS FOR THIS SECTION:

- Learn how to fit and interpret multiple linear regression models in Python
- Walk through methods of performing variable selection, ensuring model variables add value
- Compare the predictive accuracy across models using mean error metrics like MAE and RMSE

MULTIPLE LINEAR REGRESSION MODEL

Multiple Regression

Variable Selection

Mean Error Metrics

Multiple linear regression models use *multiple features* to predict the target

- In other words, it's the same linear regression model, but with additional "x" variables

SIMPLE LINEAR REGRESSION MODEL:

$$y_i = \beta_0 + \beta_1 x_{i1}$$

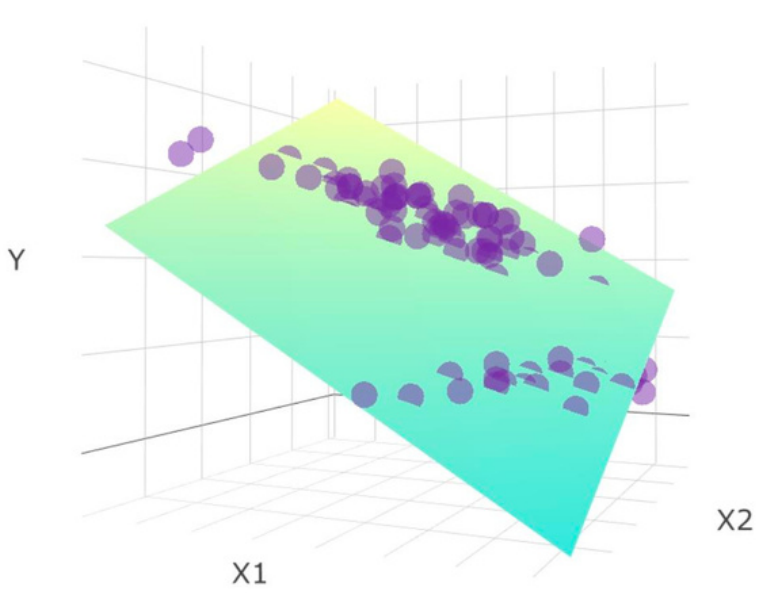
MULTIPLE LINEAR REGRESSION MODEL:

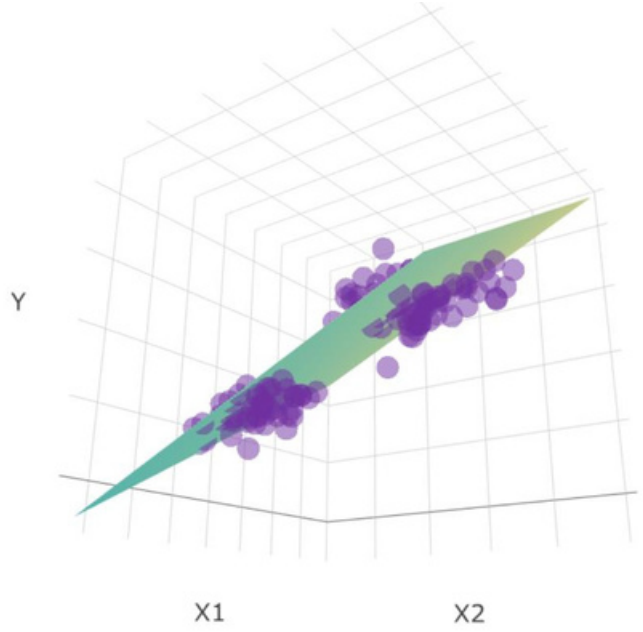
$$y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \ldots + \beta_p x_{ip} + \epsilon$$

*Instead of just one "x", we have a **whole set of features** (and associated coefficients) to help predict the target (y)*

MULTIPLE LINEAR REGRESSION MODEL

To visualize how a multiple linear regression model works with 2 features, imagine fitting a plane *(instead of a line)* through a 3D scatterplot:





Multiple Regression

Variable Selection

Mean Error Metrics

Multiple regression can scale well beyond 2 variables, but this is where visual analysis breaks down – and one reason why we need **machine learning**!

FITTING A MULTIPLE REGRESSION

To **fit a multiple regression**

Multiple Regression

Variable Selection

Mean Error Metrics

```python
from sklearn.model_selection import train_test_split
```

```python
x = df[['age']]
y = df['charges']
```

```python
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.1, random_state=90)
```

```python
df.corr()
```

	age	sex	bmi	children	smoker	charges	region_northeast	region_northwest	region_southeast	region_southwest
age	1.000000	0.018326	0.128016	0.032763	-0.012625	0.955779	0.014151	0.003251	-0.047004	0.029025
sex	0.018326	1.000000	-0.001841	-0.006287	-0.001524	0.072101	-0.008953	-0.000319	-0.002020	0.010989
bmi	0.128016	-0.001841	1.000000	0.000765	-0.087787	0.108261	-0.135730	-0.120017	0.255862	0.000401
children	0.032763	-0.006287	0.000765	1.000000	-0.002767	0.171814	-0.027998	0.009706	-0.016625	0.033845
smoker	-0.012625	-0.001524	-0.087787	-0.002767	1.000000	0.107042	0.027877	-0.026673	-0.026013	0.024803
charges	0.955779	0.072101	0.108261	0.171814	0.107042	1.000000	0.070417	0.010897	-0.085157	0.004127
region_northeast	0.014151	-0.008953	-0.135730	-0.027998	0.027877	0.070417	1.000000	-0.327673	-0.319571	-0.333977
region_northwest	0.003251	-0.000319	-0.120017	0.009706	-0.026673	0.010897	-0.327673	1.000000	-0.332356	-0.347338
region_southeast	-0.047004	-0.002020	0.255862	-0.016625	-0.026013	-0.085157	-0.319571	-0.332356	1.000000	-0.338750
region_southwest	0.029025	0.010989	0.000401	0.033845	0.024803	0.004127	-0.333977	-0.347338	-0.338750	1.000000

"age" are the most correlated features with "charges"

OLS Regression Results

Dep. Variable:	price	**R-squared:**	0.853
Model:	OLS	**Adj. R-squared:**	0.853
Method:	Least Squares	**F-statistic:**	1.570e+05
Date:	Thu, 03 Aug 2023	**Prob (F-statistic):**	0.00
Time:	10:05:44	**Log-Likelihood:**	-4.7201e+05
No. Observations:	53943	**AIC:**	9.440e+05
Df Residuals:	53940	**BIC:**	9.441e+05
Df Model:	2		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	1738.1187	103.618	16.774	0.000	1535.026	1941.211
carat	1.013e+04	62.551	161.886	0.000	1e+04	1.02e+04
x	-1026.9048	26.432	-38.851	0.000	-1078.711	-975.099

Omnibus:	14014.880	**Durbin-Watson:**	2.001
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	317574.692
Skew:	0.717	**Prob(JB):**	0.00
Kurtosis:	14.800	**Cond. No.**	112.

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

R2 increased from 0.849

$p < 0.05$, so the model isn't useless

$p < 0.05$, so all variables are useful

"x" has a negative coefficient but a positive correlation (this is a concern we'll cover shortly)

INTERPRETING MULTIPLE REGRESSION

Interpreting multiple regression is like simple regression, with a slight twist

- Multiple Regression
- Variable Selection
- Mean Error Metrics

```
X = sm.add_constant(diamonds[["carat", "x"]])
y = diamonds["price"]

model = sm.OLS(y, X).fit()

model.summary()
```

	coef	std err	t	P>|t|	[0.025	0.975]
const	1738.1187	103.618	16.774	0.000	1535.026	1941.211
carat	1.013e+04	62.551	161.886	0.000	1e+04	1.02e+04
x	-1026.9048	26.432	-38.851	0.000	-1078.711	-975.099

$$\hat{y} = 1738 + 10130(carat) - 1027(x)$$

How do we interpret this?

- Technically, a 0-carat diamond with 0 length (x) is predicted to cost $1,738 dollars
- An increase in carat by 1 corresponds to a $10,130 increase in price, **holding x constant**
- An increase in x by 1 corresponds to a $1,027 decrease in price, **holding carat constant**

VARIABLE SELECTION

Multiple Regression

Variable Selection

Mean Error Metrics

Variable selection is a critical part of the modeling process that helps reduce complexity by only including features that help meet the model's goal

- Do new variables improve model accuracy? *(critical for prediction)*
- Is each variable statistically significant? *(critical for inference)*
  - Do new variables make the model more challenging to interpret or explain to stakeholders?

EXAMPLE*Using all the possible features to predict diamond price*

OLS Regression Results

Dep. Variable:	price	**R-squared:**	0.859
Model:	OLS	**Adj. R-squared:**	0.859

	coef	std err	t	P>|t|	[0.025	0.975]
const	2.085e+04	447.545	46.584	0.000	2e+04	2.17e+04
carat	1.069e+04	63.198	169.094	0.000	1.06e+04	1.08e+04
depth	-203.1414	5.504	-36.909	0.000	-213.929	-192.354
table	-102.4407	3.084	-33.217	0.000	-108.485	-96.396
x	-1315.7397	43.069	-30.550	0.000	-1400.155	-1231.324
y	66.3300	25.522	2.599	0.009	16.306	116.354
z	41.6285	44.303	0.940	0.347	-45.207	128.464

R2 improved to .859 compared to the simple regression model (.849)
If the goal is inference, a single variable model may be a good choice!

"x" still has a negative coefficient
This helps R2 but makes the model hard to explain

"z" has a p-value greater than alpha (0.347 > 0.05) We should drop this variable from the model

ADJUSTED R-SQUARED

Multiple Regression

Variable Selection

Mean Error Metrics

A criticism of r-squared is that it will never decrease as new variables are added

Adjusted r-squared corrects this by penalizing new variables added to a model

This measure has no meaning whatsoever, but it helps as a variable selection tool

EXAMPLE | *Adding a random column to a sample of 100 diamonds*

OLS Regression Results

Dep. Variable:	price	**R-squared:**	0.782
Model:	OLS	**Adj. R-squared:**	0.780
Method:	Least Squares	**F-statistic:**	350.1

	coef	std err	t	P>|t|	[0.025	0.975]
const	-2517.2645	355.136	-7.088	0.000	-3222.020	-1812.509
carat	8631.6433	459.978	18.765	0.000	7718.831	9544.455

OLS Regression Results

Dep. Variable:	price	**R-squared:**	0.784
Model:	OLS	**Adj. R-squared:**	0.780
Method:	Least Squares	**F-statistic:**	176.2

	coef	std err	t	P>|t|	[0.025	0.975]
const	-2263.3000	452.073	-5.006	0.000	-3160.540	-1366.060
carat	8673.9033	462.726	18.745	0.000	7755.520	9592.287
random	-5.5127	6.063	-0.909	0.366	-17.547	6.521

R2 increased but adjusted R2 didn't

In this case, the p-value could also tell us to remove this variable (other times, a variable can be significant and lower adjusted R2)

MEAN ERROR METRICS

Multiple Regression

Variable Selection

Mean Error Metrics

Mean error metrics measure how well your regression model *fits in in the units of our target*, as opposed to how well it explains variance (like R-Squared)

- The most common are **Mean Absolute Error** (MAE) and **Root Mean Squared Error**
- (RMSE) They are used to compare model fit across models *(the lower the better!)*

MAE

*Average of the **absolute** distance between actual & predicted values*

MSE

*Average of the **squared** distance between actual & predicted values*

RMSE

***Square root** of Mean Squared Error, to return to the target's units (like MAE)*

RMSE is more sensitive to large outliers, so it is preferred over MAE in situations where they are particularly undesirable

MEAN ERROR METRICS

Multiple Regression

Variable Selection

Mean Error Metrics

You can use **sklearn.metrics** to calculate MAE and MSE for your model

- sklearn.metrics.**mean_absolute_error**(y_actual, y_predicted)
- sklearn.metrics.**mean_squared_error**(y_actual, y_predicted)

```
from sklearn.metrics import mean_absolute_error as mae
from sklearn.metrics import mean_squared_error as rmse

X = sm.add_constant(diamonds["carat"])
y = diamonds["price"]

model = sm.OLS(y, X).fit()

print(f"MAE: {mae(y, model.predict())}")
print(f"RMSE {rmse(y, model.predict(), squared=False)}")
```

```
MAE: 1007.4339350399421
RMSE 1548.4940983743945
```

*This returns **RMSE** instead of MSE*

The simple linear regression model has an average prediction error of ~$1,000

The outliers in the dataset are making RMSE around 50% larger than MAE

```
x = sm.add_constant(diamonds[["carat", "depth", "table", "x", "y"]])
y = diamonds["price"]

model = sm.OLS(y, x).fit()

print(f"MAE: {mae(model.predict(x), y)}")
print(f"RMSE: {rmse(model.predict(x), y, squared=False)}")
```

```
MAE: 889.2135691786077
RMSE: 1496.8311707830426
```

RMSE will always be bigger than MAE

The multiple linear regression model performs better across both metrics

KEY TAKEAWAYS

Multiple linear regression models use **multiple features** to predict the target

Each new feature comes with an associated coefficient that forms part of the regression equation

Variable selection methods help you identify valuable features for the model

- *Coefficients with p-values greater than alpha (0.05) indicate that a coefficient isn't significantly different than 0*
- *Contrary to R2, the adjusted R2 metric penalizes new variables added to a model*

Mean error metrics let you compare predictive accuracy across models

- *Mean Absolute Error (MAE) and Root Mean Square Error (RMSE) quantify a model's inaccuracy in the target's units*
- *RMSE is more sensitive to large errors, so it is preferred in situations where large errors are undesirable*

MODEL ASSUMTIONS

MODEL ASSUMPTIONS

In this section we'll cover the **assumptions of linear regression** models which should be checked and met to ensure that the model's predictions and interpretation are valid

TOPICS WE'LL COVER:

- Assumptions Overview
- Linearity
- Independence of Errors
- Normality of Errors
- No Perfect Multicollinearity
- Equal Variance of Errors
- Outliers & Influence

GOALS FOR THIS SECTION:

- Review the assumptions of linear regression
- models Learn to diagnose and fix violations to each
- assumption using Python
  Assess the influence of outliers on a regression model, and learn methods for dealing with them

ASSUMPTIONS OF LINEAR REGRESSION

There are a few key **assumptions of linear regression** models that can be violated, leading to unreliable predictions and interpretations

- If the goal is *inference*, these all need to be checked rigorously
- If the goal is *prediction*, some of these can be relaxed

1. **Linearity**: a linear relationship exists between the target and features
2. **Independence of errors**: the residuals are not correlated
3. **Normality of errors**: the residuals are approximately normally distributed
4. **No perfect multicollinearity**: the features aren't perfectly correlated with each other
5. **Equal variance of errors**: the spread of residuals is consistent across predictions

You can use the **L.I.N.N.E** acronym (like *linear* regression) to remember them

It's worth noting that you might see resources saying there are anywhere from 3 to 6 assumptions, but these 5 are the ones to focus on

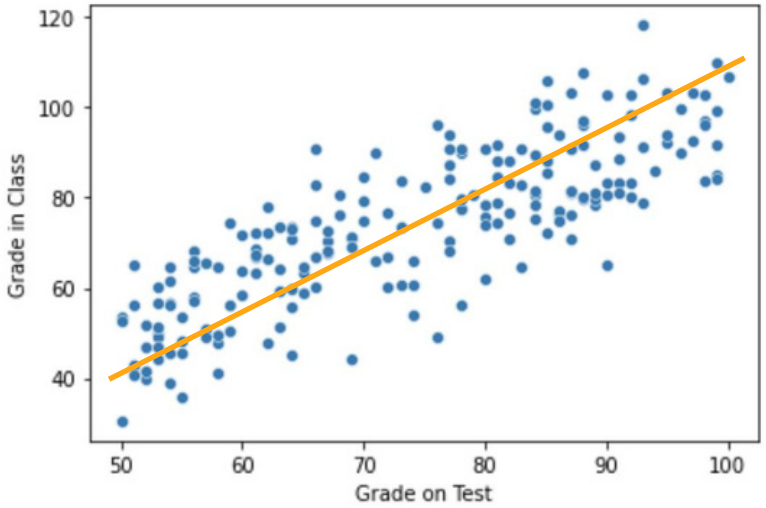
LINEARITY

Linearity assumes there's a linear relationship between the target and each feature

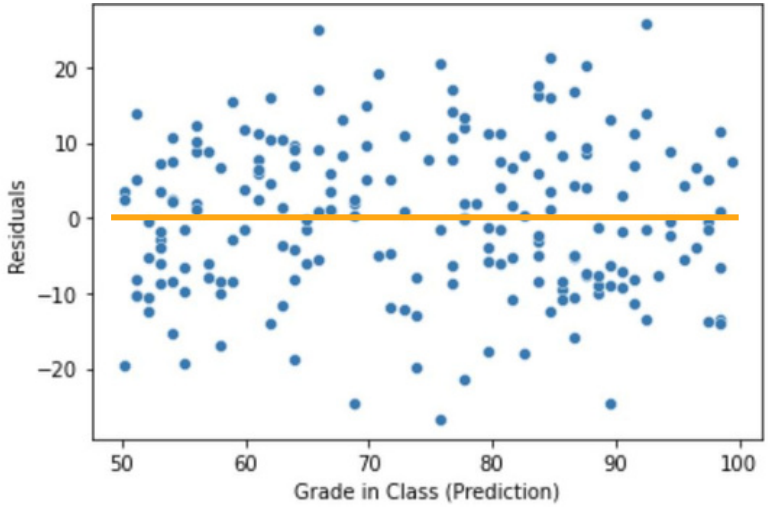
- If this assumption is violated, it means the model isn't capturing the underlying relationship between the variables, which could lead to inaccurate predictions

You can diagnose linearity by using **scatterplots** and **residual plots**:

Ideal Scatterplot



Ideal Residual Plot



INDEPENDENCE OF ERRORS

Independence of errors assumes that the residuals in your model have no patterns or relationships between them *(they aren't autocorrelated)*

In other words, it checks that you haven't fit a linear model to time series data

You can diagnose independence with the **Durbin-Watson Test**:

- **Ho: DW=2** – the errors are NOT autocorrelated
- **Ha: DW≠2** – the errors are autocorrelated
- As a rule of thumb, values between 1.5 and 2.5 are accepted

```
features = ["carat", "carat_sq", "depth", "table", "x"]

X = sm.add_constant(diamonds.loc[:, features])
y = diamonds["price"]

model = sm.OLS(y, X).fit()

model.summary()
```

Omnibus:	13855.832	**Durbin-Watson:**	1.999
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	293840.413
Skew:	0.723	**Prob(JB):**	0.00
Kurtosis:	14.342	**Cond. No.**	6.97e+03

All good!

You can fix independence issues by using a time series model *(more later!)*

INDEPENDENCE OF ERRORS

IMPORTANT: If your data is sorted by your target, or potentially an important predictor variable, it can cause this assumption to be violated

Use **df.sample(frac=1)** to fix this by randomly shuffling your dataset rows

Model (sorted by price)

```
diamonds = diamonds.sort_values("price")

X = sm.add_constant(diamonds.loc[:, features])
y = diamonds["price"]

model = sm.OLS(y, X).fit()

model.summary()
```

Omnibus:	13855.832	**Durbin-Watson:**	1.252
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	293840.413
Skew:	0.723	**Prob(JB):**	0.00
Kurtosis:	14.342	**Cond. No.**	6.97e+03

Sorting the DataFrame by the target (price) leads to a Durbin-Watson statistic outside the desired range

Model (randomized rows)

```
diamonds = diamonds.sample(frac=1).copy()

X = sm.add_constant(diamonds.loc[:, features])
y = diamonds["price"]

model = sm.OLS(y, X).fit()

model.summary()
```

Omnibus:	13855.832	**Durbin-Watson:**	2.002
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	293840.413
Skew:	0.723	**Prob(JB):**	0.00
Kurtosis:	14.342	**Cond. No.**	6.97e+03

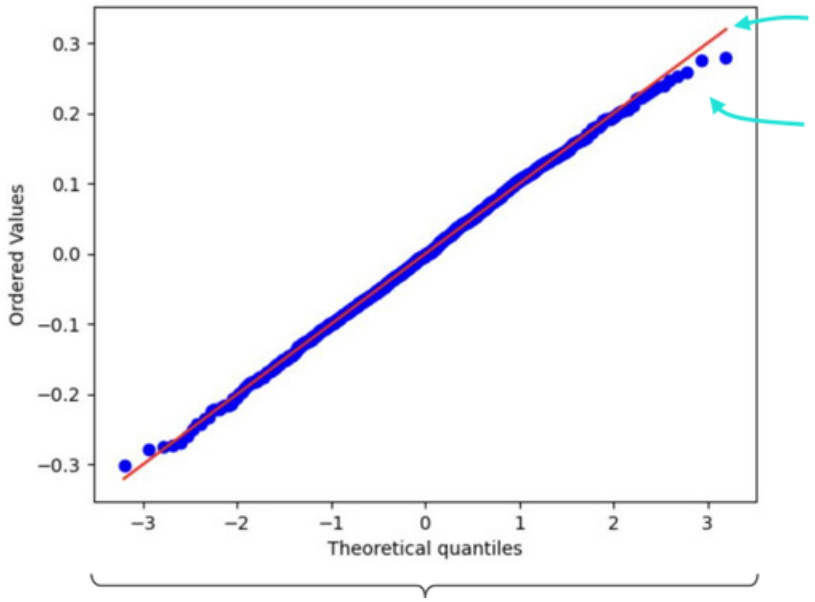
Randomizing the order brings things back to normal

NORMALITY OF ERRORS

Normality of errors assumes the *residuals* are approximately normally distributed

You can diagnose normality by using a **QQ plot** (quantile-quantile plot):

Ideal QQ Plot



*The **red line** represents a normal distribution*

It's normal for a few points to fall off the line

The x-axis represents the number of standard deviations away from the mean residuals lie

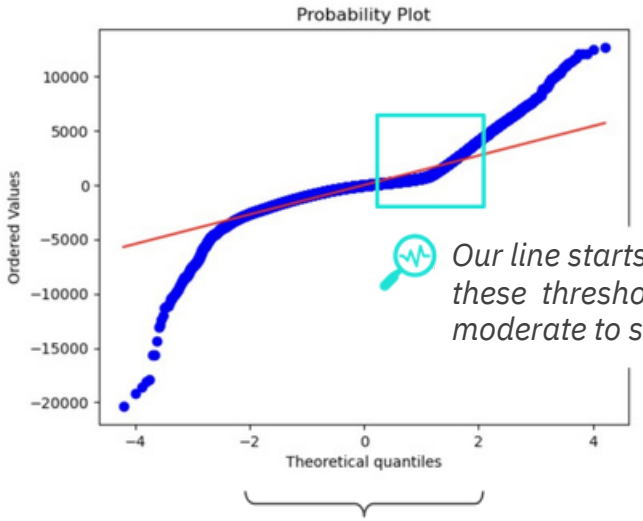
NORMALITY OF ERRORS

You can typically fix normality issues by applying a **log transform on the target**

- Other options are applying log transforms to features or simply leaving the data as is

Price ModelLog of Price Model

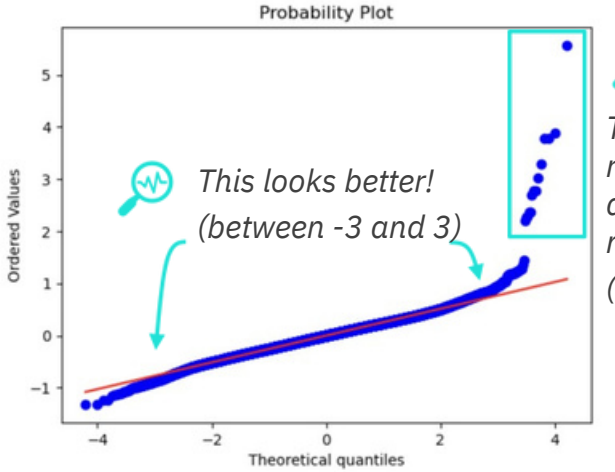
```
import scipy.stats as stats
import matplotlib.pyplot as plt

stats.probplot(model.resid, dist="norm", plot=plt);
```



Our line starts to curve in between these thresholds, which indicates moderate to severe non-normality

You generally want to see points fall along the line in between -2 and +2 standard deviations

```
import numpy as np

X = sm.add_constant(diamonds.loc[:, features])
y = np.log(diamonds["price"])

model = sm.OLS(y, X).fit()

stats.probplot(model.resid, dist="norm", plot=plt);
```



This looks better! (between -3 and 3)

There are still large residuals left, but they don't violate the normality assumption (check later)

TIP: INTERPRETING TRANSFORMED TARGETS

Transforming your target fundamentally changes the interpretation of your model, so you need to invert the transformation to understand coefficients & predictions

- Inverting a feature coefficient returns the associated multiplicative change in the target's
- value Inverting a prediction returns the target in its original units


```
X = sm.add_constant(diamonds.loc[:, features])
y = np.log(diamonds["price"])

model = sm.OLS(y, X).fit()

model.summary()
```

	coef	std err	t	P>|t|	[0.025	0.975]
const	5.4119	0.089	60.924	0.000	5.238	5.586
carat	2.3598	0.035	67.643	0.000	2.291	2.428
carat_sq	-0.6431	0.007	-92.340	0.000	-0.657	-0.629
depth	-0.0081	0.001	-8.874	0.000	-0.010	-0.006
table	-0.0159	0.001	-29.451	0.000	-0.017	-0.015
x	0.4294	0.009	46.887	0.000	0.411	0.447

*A 1-unit increase in carat is associated with a **10.6X increase in price***

(e2.3598 = 10.59)

```
#[constant, carat, carat_sq, depth, table, x]
diamond = [1, 1.5, 1.5**2, 62, 59, 7.2]

np.exp(model.predict(diamond))
```

```
array([9458.79802436])
```

*A diamond with these features is predicted to cost **$9,458***

Transformation	Inverse
y = np.sqrt(x)	x = y**2
y = np.log(x)	x = np.exp(y)
y = np.log10(x)	x = 10 ** y
y = 1/x	x = 1/y

NO PERFECT MULTICOLLINEARITY

No perfect multicollinearity assumes that features aren't perfectly correlated with each other, as that would lead to unreliable and illogical model coefficients

- If two features have a correlation (r) of 1, there are infinite ways to minimize squared error
- Even if it's not perfect, strong multicollinearity (r > 0.7) can still cause issues to a model

You can diagnose multicollinearity with the **Variance Inflation Factor** (VIF):

- Each feature is treated as the target, and R measures how well the other features
- predict it As a rule of thumb, a VIF > 5 indicates that a variable is causing multicollinearity problems

```
from statsmodels.stats.outliers_influence import variance_inflation_factor as vif

variables = sm.OLS(y, X).exog

pd.Series([vif(variables, i) for i in range(variables.shape[1])], index=X.columns)
```

```
const       6289.392199
carat        217.938040
carat_sq      43.152926
depth          1.378775
table          1.156815
x             84.132321
dtype: float64
```

We can ignore the VIF for the intercept, but most of our variables have a VIF > 5

NO PERFECT MULTICOLLINEARITY

There are several ways to fix multicollinearity issues:

- The most common is to **drop features** with a VIF > 5 *(leave at least 1 to see the impact)*

Original Model

```
const        6289.392199
carat         217.938040
carat_sq       43.152926
depth           1.378775
table           1.156815
x              84.132321
dtype: float64
```

Removing *x*

```
const        3539.505406
carat          11.182845
carat_sq       11.055436
depth           1.105012
table           1.146514
dtype: float64
```

We still have VIF > 5 for our carat terms, but we can generally ignore those since they are polynomial terms

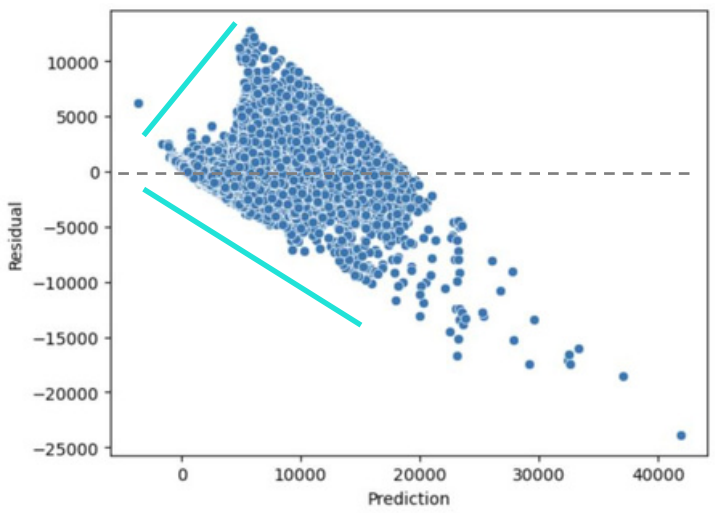
EQUAL VARIANCE OF ERRORS

Equal variance of errors assumes the residuals are consistent across predictions

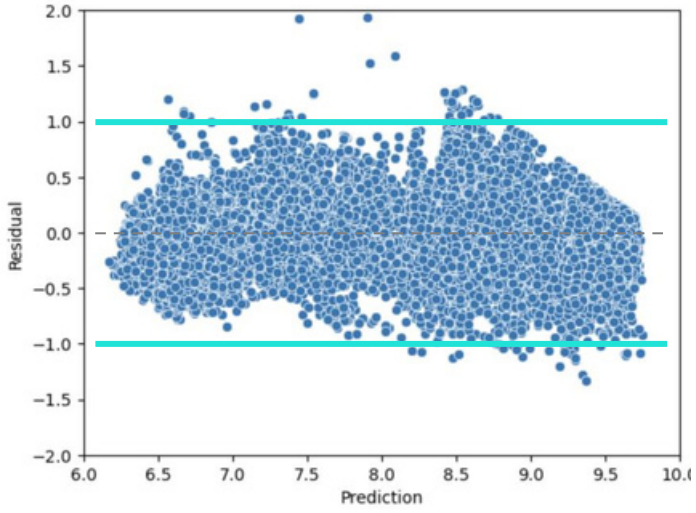
- In other words, average error should stay roughly the same across the range of the target
- Equal variance is known as **homoskedasticity**, and non-equal variance is **heteroskedasticity**

You can diagnose heteroskedasticity with **residual plots**:

Heteroskedasticity
(original regression model)



Homoskedasticity
(after fixing violated assumptions)



Our model predicts much better now!

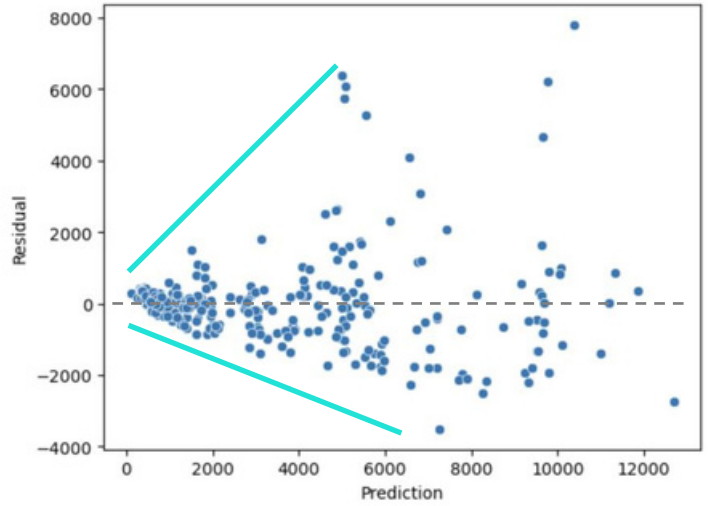
EQUAL VARIANCE OF ERRORS

You can typically fix heteroskedasticity by **applying a log transform** on the target

In other words, the average error should stay roughly the same across the target variable

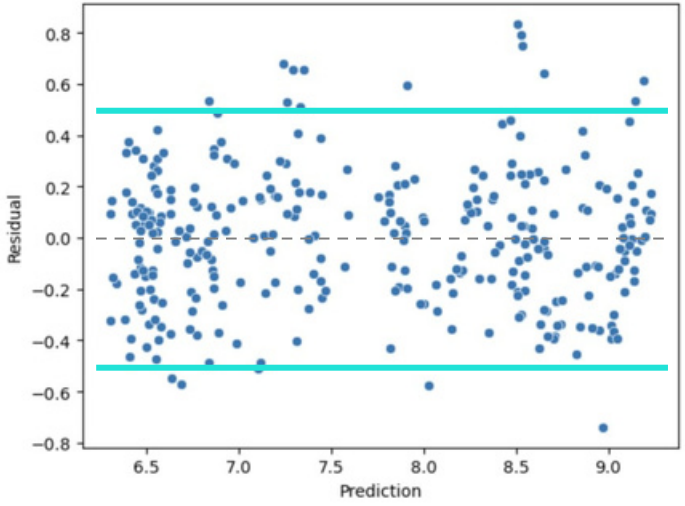
Price ModelLog of Price Model

```
X = sm.add_constant(d_sample.loc[:, features])
y = d_sample["price"]

model = sm.OLS(y, X).fit()

sns.scatterplot(x=model.predict(), y=model.resid);
```



*Errors have a **cone shape** along the x-axis*

```
X = sm.add_constant(d_sample.loc[:, features])
y = np.log(d_sample["price"])

model = sm.OLS(y, X).fit()

sns.scatterplot(x=model.predict(), y=model.resid);
```



*Errors are **spread evenly** along the x-axis*

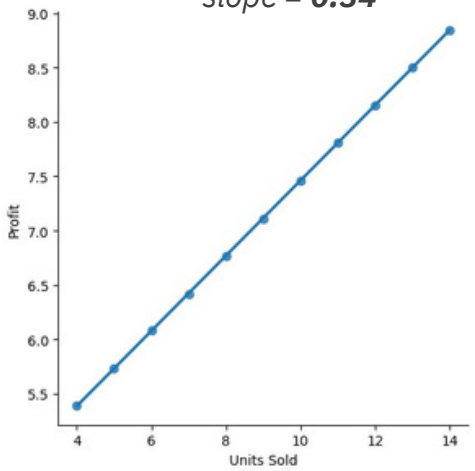
OUTLIERS & INFLUENCE

Outliers are extreme data points that fall well outside the usual pattern of data

- Some outliers have dramatic **influence** on model fit, while others won't
- Outliers that impact a regression equation significantly are called **influential points**

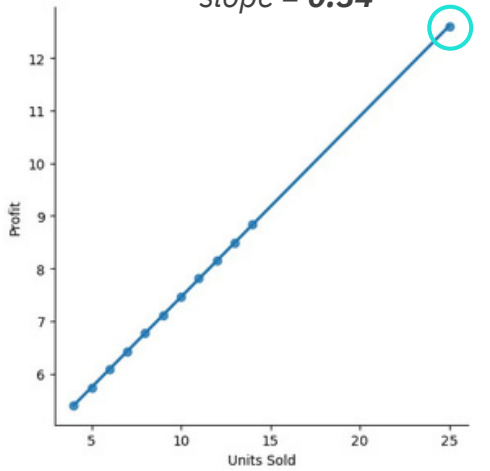
No Outliers

*slope = **0.34***



Non-influential Outlier

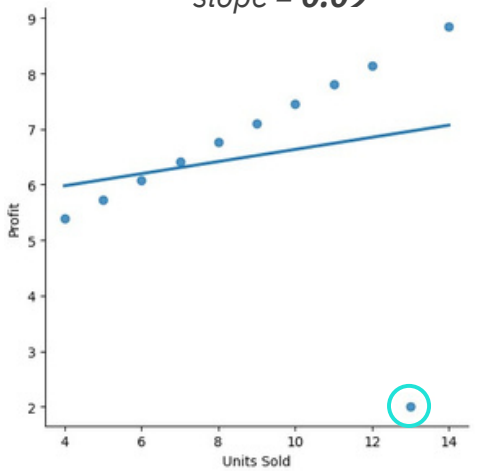
*slope = **0.34***



This is an outlier in both profit and of units sold, but it's in line with the rest the data, so it's not influential

Influential Outlier

*slope = **0.09***



This is an outlier in terms of profit that doesn't follow the same pattern as the rest of the data, so it changes the regression line

OUTLIERS & INFLUENCE

There are several ways to deal with influential points:

- You can **remove them** or **leave them in**
- You can **engineer features** to help capture their variance
- You can use **robust regression** *(outside the scope of this course)*

Model (with outliers)**Model** (without 2 outliers)

OLS Regression Results

Dep. Variable:	price	**R-squared:**	0.932
Model:	OLS	**Adj. R-squared:**	0.932
Method:	Least Squares	**F-statistic:**	1.836e+05
Date:	Wed, 09 Aug 2023	**Prob (F-statistic):**	0.00
Time:	12:03:15	**Log-Likelihood:**	-4982.5
No. Observations:	53943	**AIC:**	9975.
Df Residuals:	53938	**BIC:**	1.002e+04

	coef	std err	t	P>|t|	[0.025	0.975]
const	8.1659	0.068	120.117	0.000	8.033	8.299
carat	3.9530	0.008	490.335	0.000	3.937	3.969
carat_sq	-0.9248	0.004	-257.139	0.000	-0.932	-0.918
depth	-0.0273	0.001	-32.592	0.000	-0.029	-0.026
table	-0.0183	0.001	-33.355	0.000	-0.019	-0.017

OLS Regression Results

Dep. Variable:	price	**R-squared:**	0.933
Model:	OLS	**Adj. R-squared:**	0.933
Method:	Least Squares	**F-statistic:**	1.889e+05
Date:	Wed, 09 Aug 2023	**Prob (F-statistic):**	0.00
Time:	12:03:40	**Log-Likelihood:**	-4269.4
No. Observations:	53941	**AIC:**	8549.
Df Residuals:	53936	**BIC:**	8593.

	coef	std err	t	P>|t|	[0.025	0.975]
const	8.1992	0.067	122.200	0.000	8.068	8.331
carat	4.0257	0.008	491.983	0.000	4.010	4.042
carat_sq	-0.9610	0.004	-261.491	0.000	-0.968	-0.954
depth	-0.0280	0.001	-33.812	0.000	-0.030	-0.026
table	-0.0186	0.001	-34.428	0.000	-0.020	-0.018

R2 improved slightly and the coefficients changed a bit, but not much changed (this is a large dataset)

KEY TAKEAWAYS

Linear regression models have **5 key assumptions** that must be checked

- *Linearity, independence of errors, normality or errors, no perfect multicollinearity, and equal variance of errors*
- *If the goal is inference, they need to be checked rigorously, but for prediction some of them can be relaxed*

Diagnosing & fixing these assumptions can help improve model accuracy

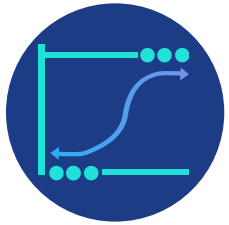
- *Use residual plots, QQ plots, the Durbin-Watson test, and the Variance Inflation Factor to diagnose*
- *Transforming the features and/or target (polynomial terms, log transforms, etc.) can typically help fix issues*

Outliers that significantly impact a model are known as **influential points**

- *Before removing influential points, consider if you expect to encounter similar data points in new data*

Logistics Regression

LOGISTICS REGRESSION



In this section we'll cover the a technique that predict a categorical dependent variable based on one or more independent variables

TOPICS WE'LL COVER:

- Logistics Regresson
- Goals of Regression
- Assumtions of Logistcs

GOALS FOR THIS SECTION:

LOGISTICS REGRESSION

Logoistics Regression

Goals of Regression

Goals of Regression

Logistic Regression is a classification technique used to predict the probability of a binary (true/false)outcome

- In its simplest form, logistic regression forms an **S-shaped curve between 0 -1**, which represents the probability of a TRUE outcome for any given value of X
- The **likelihood function** measures how accurately a model predicts outcomes, and is used to optimize the "shape" of the curve
- Although it has the word *"regression"*in its name, logistic regression is not used for predicting numeric variables

Example use cases:

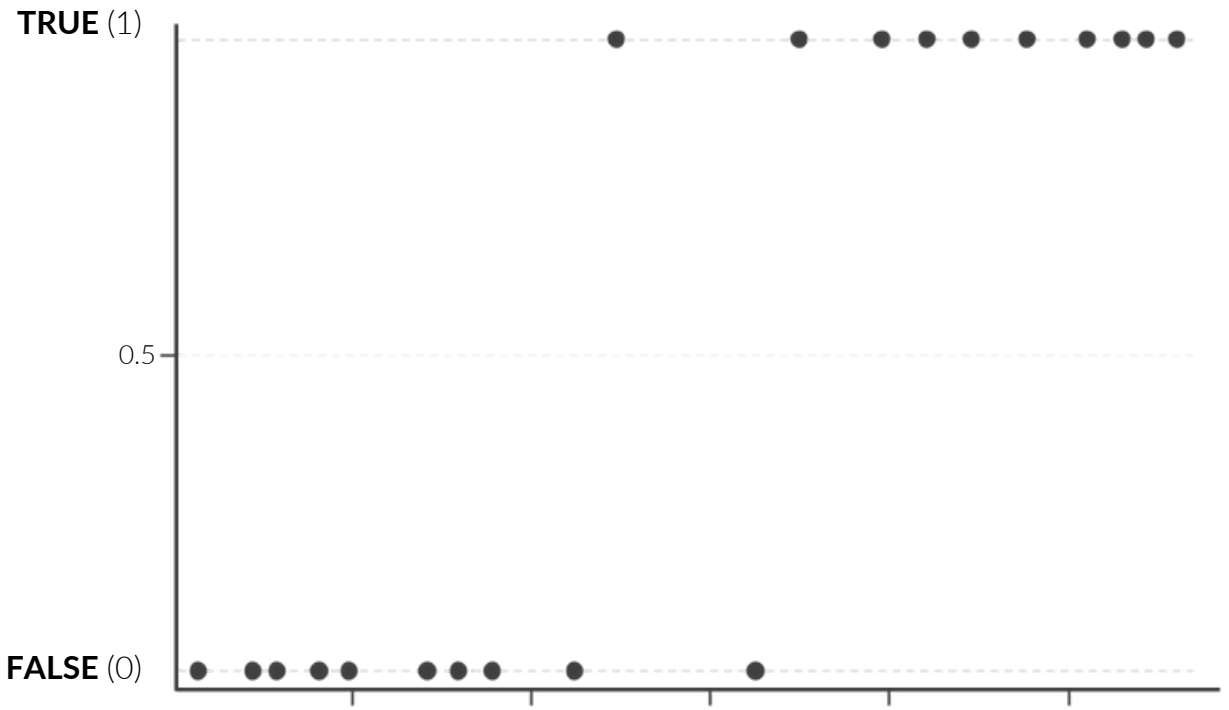
Flagging spam emails or fraudulent credit card transactions

LOGISTICS REGRESSION

Logoistics Regression

Goals of Regression

Goals of Regression



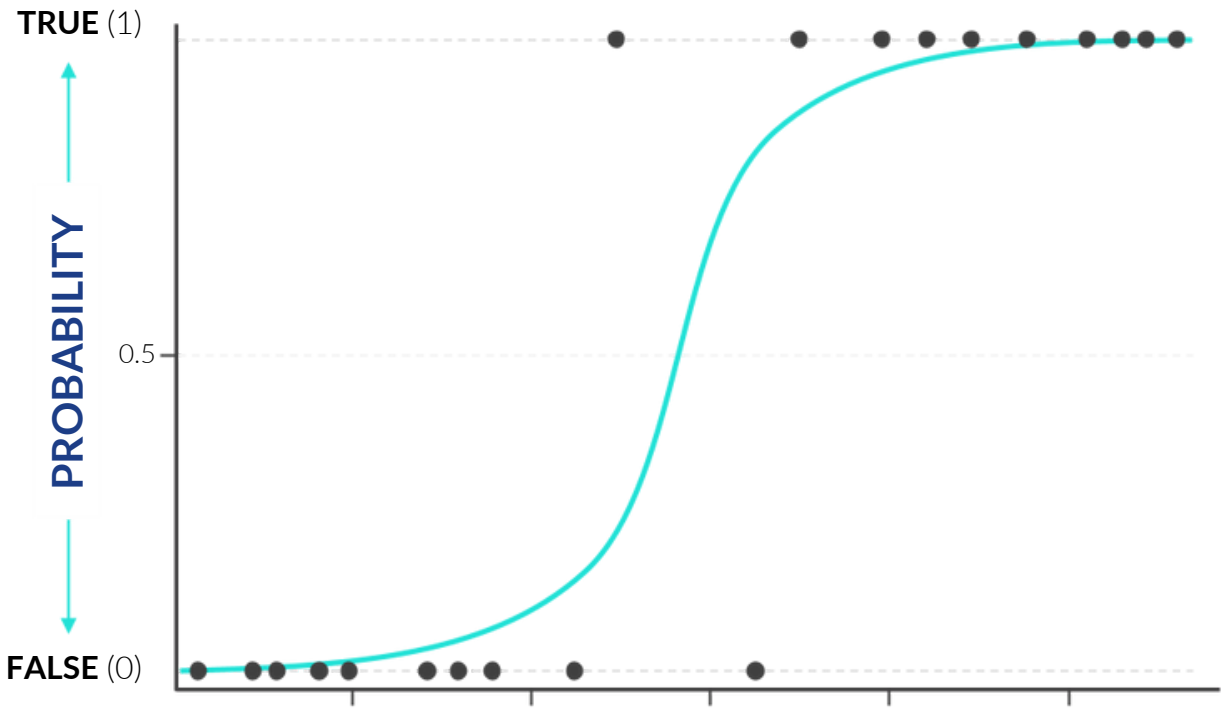
- Each dot represents an observed value, where **X is a numerical independent variable** and **Y is the binary outcome** *(true/false)*that we want to predict

LOGISTICS REGRESSION

Logoistics Regression

Goals of Regression

Goals of Regression



- Logistic regression plots the **best-fitting curve between 0 and 1**, which tells us the probability of Y being TRUE for any given value of X1

LOGISTICS REGRESSION

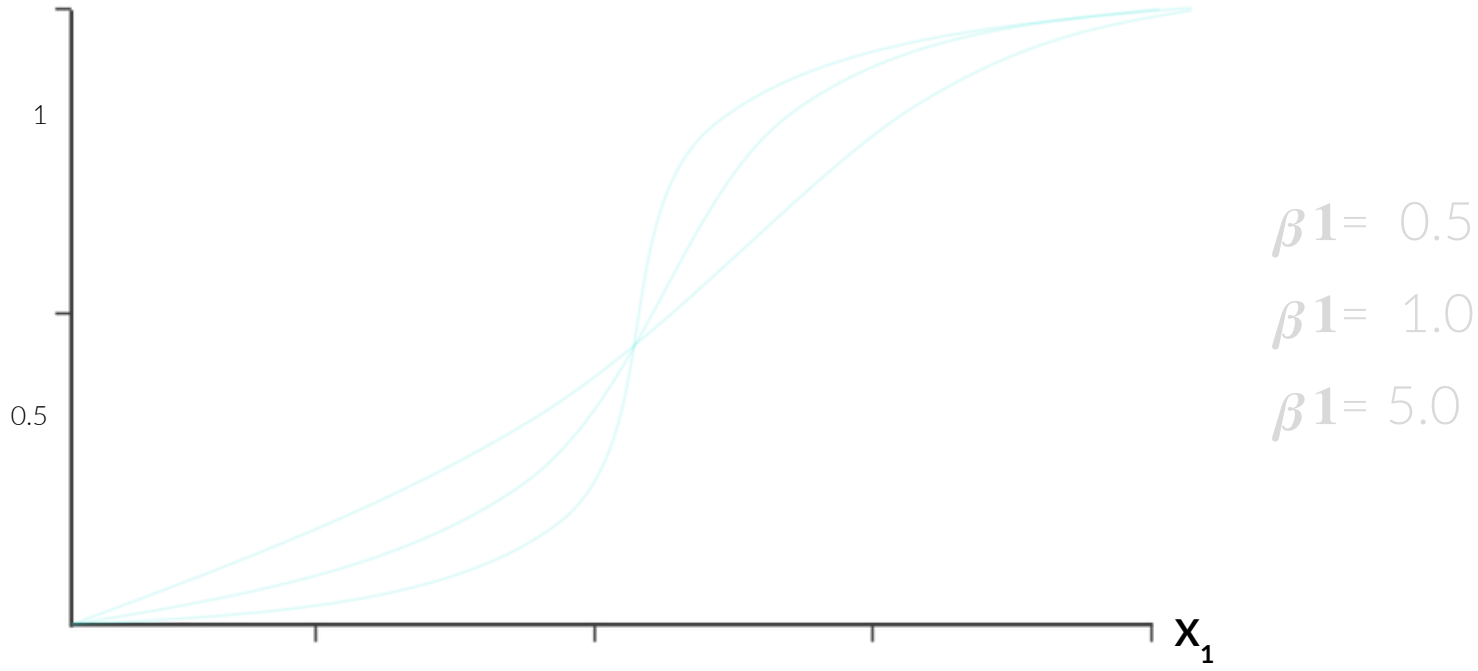
Logoistics Regression

Goals of Regression

Goals of Regression

Makes the output fall between **0** and **1**

$$\frac{1}{1 + e^{-(\beta_0 + \beta_1 x_1)}}$$



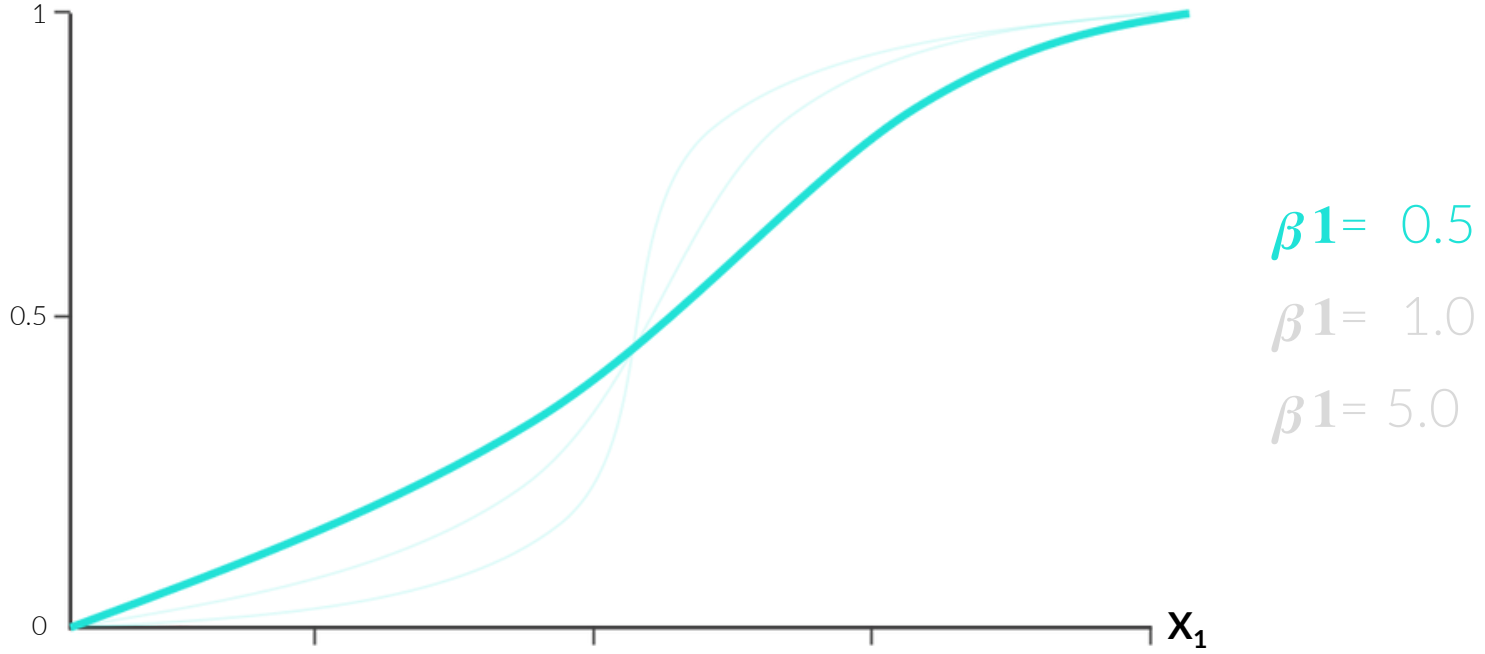
- Logistic regression plots the **best-fitting curve between 0 and 1**, which tells us the probability of Y being TRUE for any given value of X1

LOGISTICS REGRESSION

Makes the output fall between **0** and **1**

$$\frac{1}{1 + \bar{e}^{(\beta_0 + \beta_1 x_1)}}$$

- Logoistics Regression
- Goals of Regression
- Evaulation



$\beta 1 = 0.5$

$\beta 1 = 1.0$

$\beta 1 = 5.0$

- Logistic regression plots the **best-fitting curve between 0 and 1**, which tells us the probability of Y being TRUE for any given value of X1

LOGISTICS REGRESSION

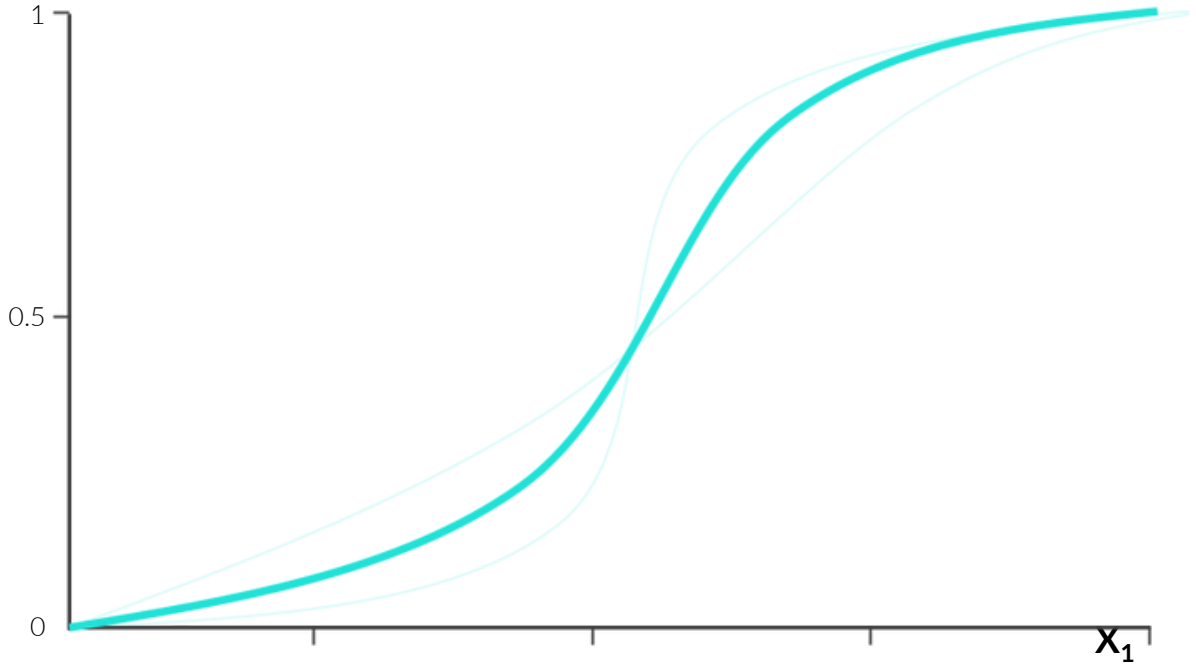
Logoistics Regression

Goals of Regression

Evaulation

Makes the output fall between **0** and **1**

$$\frac{1}{1 + e^{-(\beta_0 + \beta_1 x_1)}}$$



$\beta 1 = 0.5$

$\beta 1 = 1.0$

$\beta 1 = 5.0$

- Logistic regression plots the **best-fitting curve between 0 and 1**, which tells us the probability of Y being TRUE for any given value of X1