វិទ្យាស្ថានសាន់រ៉ាយស៍

SUNRISE INSTITUTE

ASUMPTIONS OF LINEAR REGRESSION

EMBARKING ON A JOURNEY INTO DATA SCIENCE

YA MANON

You can have data without information but you cannot have information without data.

-Daniel Keys Maran

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 Variable 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

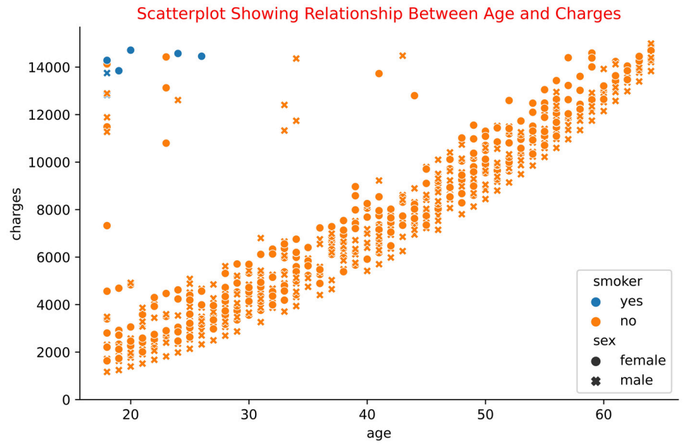
LINEARITY

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

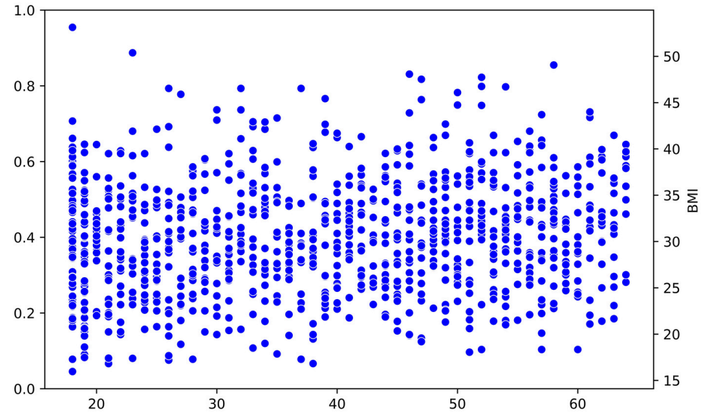
- If this assumption is violated, it means the model isn't capturing the underlying relationship between the variables, which could lead to inaccurate predictions

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

Ideal Scatterplot



Ideal Scatterplot



INDEPENDENCE OF ERRORS

ទិន្នាស្ថានសាន់រ៉ាយស៍
SUNRISE INSTITUTE

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

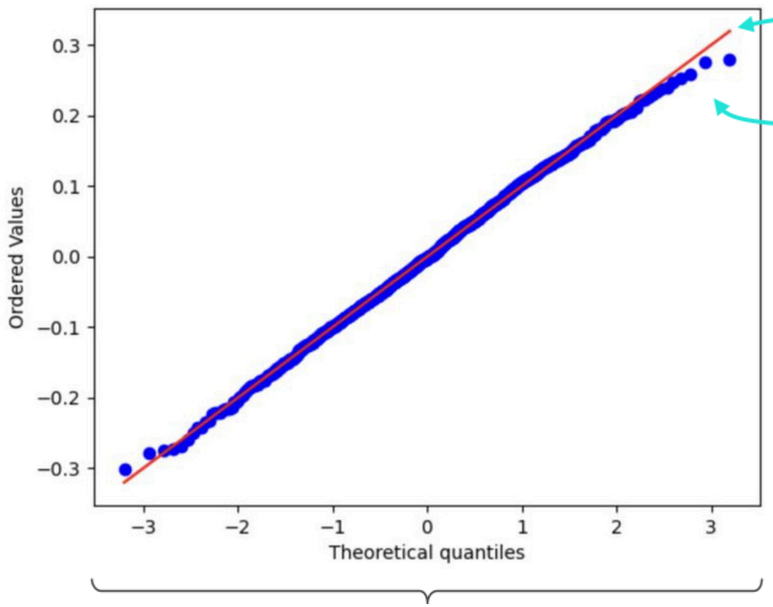
Randomizing the order brings things back to normal

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

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

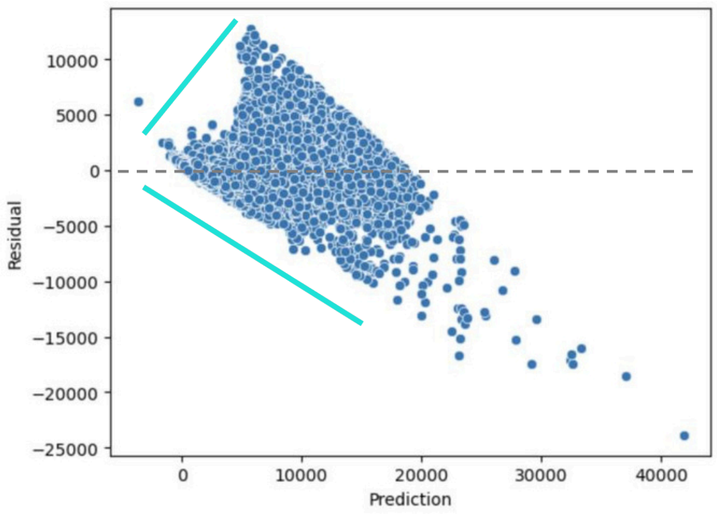
EQUAL VARIANCE OF ERRORS

Equal variance of errors assumes the residuals are consistent across predictions

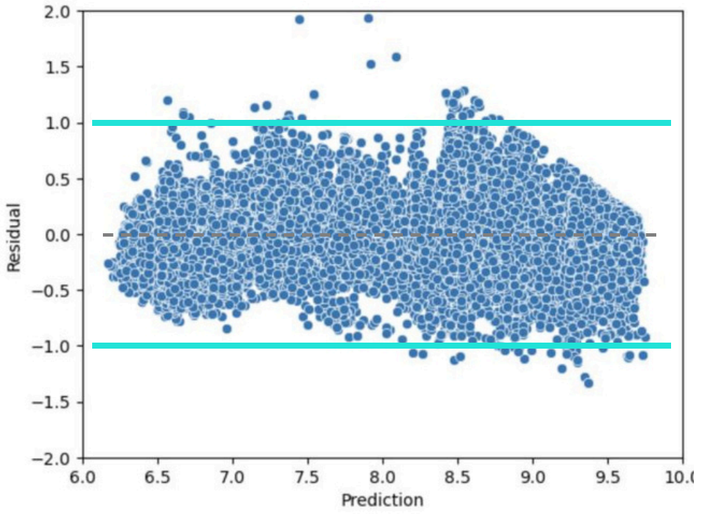
- In other words, average error should stay roughly the same across the range of the target
- Equal variance is known as **homoskedasticity**, and non-equal variance is **heteroskedasticity**

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

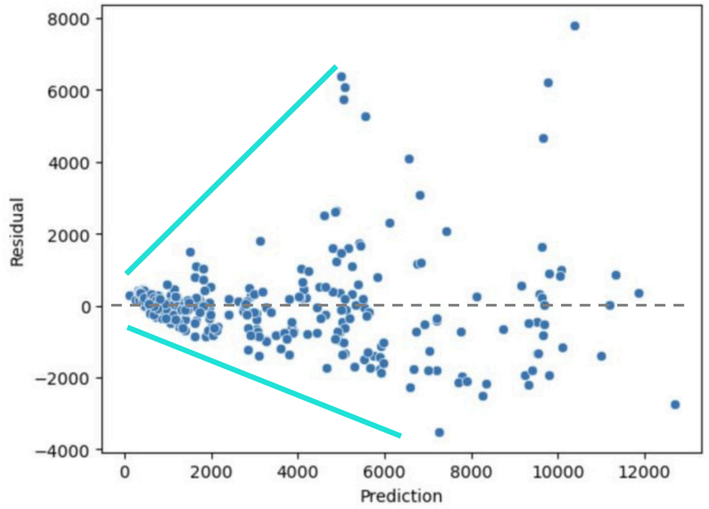
In other words, the average error should stay roughly the same across the target variable

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);
```

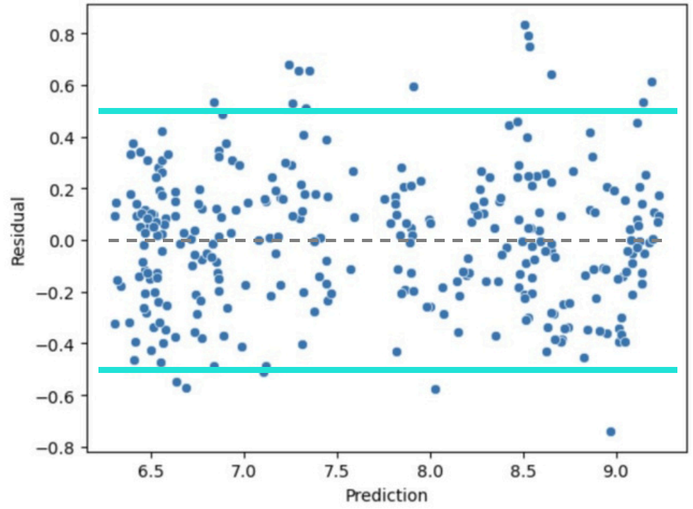


*Errors have a **cone shape** along the x-axis*

```
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*

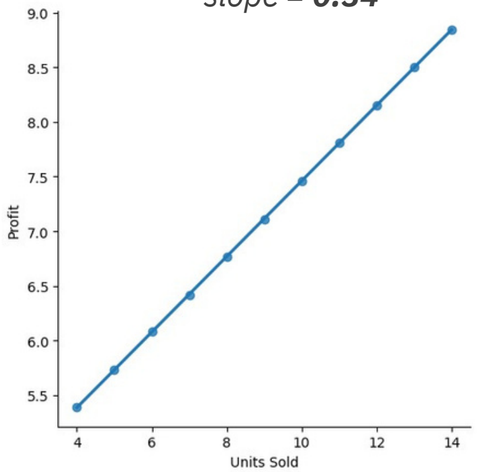
OUTLIERS & INFLUENCE

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**

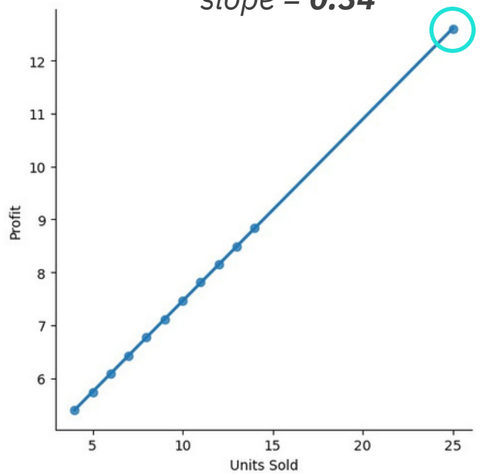
No Outliers

*slope = **0.34***



Non-influential Outlier

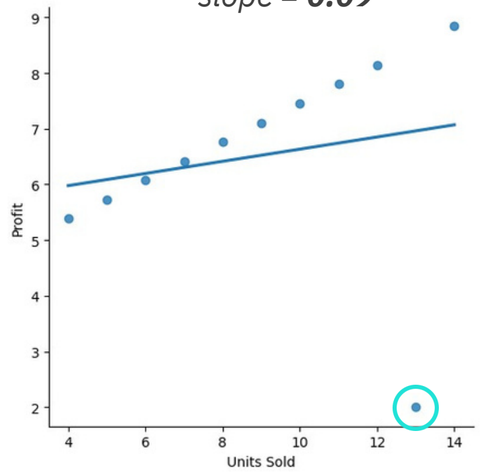
*slope = **0.34***



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)