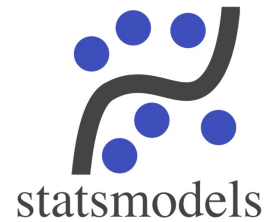




The  
Center of  
**Applied  
Data Science**



# Regression Modelling & Model Evaluation

## Content

- [Regression Nuts and Bolts](#)
  - [Understanding regression output for prediction](#)
  - [Fitted values and residuals](#)
- [Model Evaluation](#)
- [Evaluation Metrics](#)
  - [1. Mean squared errors](#)
  - [2. R-squared](#)
  - [3. Adjusted R-squared](#)
  - [4. F-test](#)
- [Model selection](#)
  - [Resampling or train-test-split Approach](#)
  - [Probabilistic Statistics Approach](#)
  - [Log-Likelihood](#)
  - [AIC: Akaike Information Criterion](#)
  - [BIC: Bayesian Information Criterion](#)
- [Linear Regression Assumptions and diagnostics](#)
- [Model Validation](#)
  - [Skewness and kurtosis](#)
  - [When does it \*not\* matter?](#)
  - [When \*might\* it matter?](#)
    - [How to deal with outliers?](#)
- [Residual diagnostics](#)
  - [Residual plots](#)
    - [Plot the Residuals Against Another Variable](#)
    - [Plot the Magnitude of the Residuals Against the Predictor](#)
    - [Plot the Distribution of the Residuals](#)
    - [QQ plot](#)
- [Solution](#)
- [Reference](#)

```
In [ ]: 1 import statsmodels.api as sm
2 import pandas as pd
3 import numpy as np
4 import statsmodels.formula.api as smf
5 import seaborn as sns
6 import matplotlib.pyplot as plt
7 from sklearn.linear_model import LinearRegression
```

```
In [ ]: 1 # `sm.datasets.get_rdataset` downloads and returns R datasets
2 iris = sm.datasets.get_rdataset("iris").data
3 mtcars = sm.datasets.get_rdataset("mtcars").data
4 ToothGrowth = sm.datasets.get_rdataset("ToothGrowth").data
5 anscombe = sns.load_dataset("anscombe")
```

```
In [ ]: 1 def lm(formula, data):
2     """
3     Specifies and fits linear model with statsmodels
4     """
5     reg = smf.ols(formula = formula, data = data)
6     return reg.fit()
```

## Regression Nuts and Bolts

Assume we want to estimate  $y$  using the predictors  $x_1, x_2, \dots, x_n$ .

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_n x_n + \epsilon$$

$x_1$	$x_2$	...	$x_n$	$y$
$x_{11}$	$x_{21}$	...	$x_{n1}$	$y_1$
$x_{12}$	$x_{22}$	...	$x_{n2}$	$y_2$
...	...	...	...	...
$x_{1m}$	$x_{2m}$	...	$x_{nm}$	$y_m$

### Example:

Predict the fuel efficiency (mpg) of a car based on its displacement (disp) in mtcars dataset and print the summary of the model.

```
In [ ]: 1 mpg_disp = lm(formula = "mpg ~ disp",
2               data = mtcars)
3 mpg_disp.summary()
```

## Understanding regression output for prediction

We use the `.summary()` method to return model statistics in a data frame.

### UNDERSTANDING REGRESSION OUTPUT FOR PREDICTION

	coef	std err	t	P> t	[0.025	0.975]
Intercept	29.5999	1.230	24.070	0.000	27.088	32.111
disp	-0.0412	0.005	-8.747	0.000	-0.051	-0.032

- **'coef'** column: Lists the coefficient estimates: Intercept relates to  $\beta_0$ , etc.
- **'std err'**: Standard deviation of the sampling distribution
- **'t'**: The t-statistic calculated for the coefficient estimates
- **'P>|t|'**: The probability that we observe an estimate as extreme as what we've observed given that the null hypothesis is true
- **'[0.025 0.975]'**: The upper and lower bounds of the 95% confidence interval

Assume  $y = \beta_0 + \beta_1 x + \epsilon$  but our model outcome estimated  $y$  as  $\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x$ .

Therefore,  $\hat{\beta}_0$  and  $\hat{\beta}_1$  are **point estimates** of  $\beta_0$  and  $\beta_1$ . The summary of the linear model helps us to find **interval estimates** of  $\beta_0$  and  $\beta_1$  as well.

- The **coef** column lists the coefficient estimates: Intercept relates to  $\beta_0$ , etc.

**std err**, **t** and **P>|t|** are used for statistical inference. As we mentioned before, we can estimate  $y$  as  $\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x$ , where:

$$\hat{\beta}_1 = \frac{\text{Cov}(x, y)}{\text{Var}(x)} = \frac{\sum_{i=0}^{N-1} (x_i - \bar{x}) \times (y_i - \bar{y})}{\sum_{i=0}^{N-1} (x_i - \bar{x})^2}$$

$$\hat{\beta}_0 = \bar{y} - \beta_1 \bar{x}$$

- **std err** is the standard deviation of the sampling distribution

**Extra:**

$$\frac{\hat{\beta}_1 - \beta_1}{\frac{\sigma}{\sqrt{\sum_{i=0}^{N-1} (x_i - \bar{x})^2}}} \sim t(N - 2)$$

$$\frac{\hat{\beta}_0 - \beta_0}{\sigma \sqrt{\frac{1}{N} + \frac{\bar{x}^2}{\sum_{i=0}^{N-1} (x_i - \bar{x})^2}}} \sim t(N - 2)$$

- $t$  here refers to the t-statistic calculated for the coefficient estimates. A general rule of thumb is that a t-statistic more than 2 in absolute value represents a significant coefficient.
- $P > |t|$  is the probability that we observe an estimate as extreme as what we've observed given that the null hypothesis is true.
- $[0.025 \quad 0.975]$  represent the upper and lower bounds of the 95% confidence interval.

```
In [ ]: 1 mpg_disp.summary()
```

#### Extra:

```
In [ ]: 1 print("\n***** t values *****")
2 print(mpg_disp.tvalues)
3 print("\n***** CI *****")
4 print(mpg_disp.conf_int(alpha=0.05))
5 print("\n***** SE *****")
6 print(mpg_disp.bse)
7 print("\n***** Coefficients *****")
8 print(mpg_disp.params)
9 print("\n***** t distribution *****")
10 from scipy.stats import t
11 tval=t.ppf(1-0.025, len(mtcars)-2)
12 print(tval)
13 print("\n***** Manual CI *****")
14 # mu = x_bar +- t(n-2)*SE
15 print(mpg_disp.params - tval * mpg_disp.bse)
16 print(mpg_disp.params + tval * mpg_disp.bse)
```

**Attention:** You should not use the *p-value* or *t-statistic* of the coefficients to select variables to include in your model. The p-value only tells you if a variable is statistically significant given the other variables in your model.

A variable may not be statistically significant but improve predictive power.

**Exercise:**

- Fit a model predicting `selling_price` with `area` using the `house_prices` dataset.
- Interpret the output (specify which elements we want them to interpret.)
- Add `living_space_size` as a predictor.

```
In [ ]: 1 house_prices = pd.read_csv("../data/house_prices.csv")
        2 sns.regplot(x = "living_space_size", y = "selling_price",
        3               data = house_prices,
        4               ci = False)
```

```
In [ ]: 1 # Your code here
```

**Fitted values and residuals**

- The **fitted value**, accessed using the `.predict()` method, is the prediction from our model - the regression's best guess of what  $y$  is based on the  $X$  observed.
- The **residual**, accessed using the `.resid` attribute, is our model's 'mistake' - the difference between our prediction and the actual value of the response variable.

`predict()` also has an `exog` argument to calculate fitted values for a new set of data.

```
In [ ]: 1 price_size.predict()
```

```
In [ ]: 1 price_size.resid
```

# Model Evaluation

We want to find a function of inputs which predicts as "good" as possible on new data. How do we measure the quality of the fitted model?

The figure below shows the summary of the linear model we designed before to estimate mpg based on disp :

## UNDERSTANDING REGRESSION OUTPUT FOR PREDICTION-Model Evaluation

R-squared:	0.718
Adj. R-squared:	0.709
F-statistic:	76.51
Prob (F-statistic):	9.38e-10
Log-Likelihood:	-82.105
AIC:	168.2
BIC:	171.1

- **R-squared**: The percentage of variation in y explained by the variation in the predictors.
- **Adj. R-squared**: Replacement for R-squared if comparing models with different numbers of predictors
- **F-statistics**: The value of f-statistics to tests whether having the model is an improvement over predicting the mean of y
- **Prob(F-statistics)**: The probability that we observe an estimate as extreme as what we've observed given that the null hypothesis is true
- **Log-Likelihood**:  $\sum_{i=0}^n \ln(P(y|\beta_i))$  - probabilistic model selection
- **AIC**: Akaike Information Criterion. A method for scoring and selecting a model. choose the model giving smallest AIC
- **BIC**: Bayesian Information Criterion. A method for scoring and selecting a model. choose the model giving smallest BIC

## Evaluation Metrics

### 1. Mean squared errors

The mean squared errors metric is appealing in part because it's the **loss function** for linear regression, i.e. how we solve for the optimal weights in linear regression. Recall that OLS minimises squared errors:

$$\sum_{i=1}^n (y_i - \hat{y})^2$$

However, it's in squared units of the response variable, and is not as interpretable.

To circumvent this, we take the square root of the mean squared error to obtain the **root mean squared error**. The root mean square error is in the same units as your response variable:

In [ ]: 1 mpg\_disp.df\_resid

```
In [ ]: 1 # d = the number of degree of freedom.
2 # The number of independent ways by which a dynamic system can
3 # without violating any constraint imposed on it,
4 # is called number of degrees of freedom
5
6 # n = mpg_disp.nobs
7 n = len(mtcars)
8 # num_params = mpg_disp.df_model + 1
9 num_params = len(mpg_disp.params)
10 # d = mpg_disp.df_resid
11 d = n - num_params
12
13 mpg_disp.mse_resid, (np.sum(mpg_disp.resid**2))/d , np.sqrt(mpg_
```

## 2. R-squared

The **R-squared**, also known as the **coefficient of determination** and often denoted  $R^2$ , is defined as the percentage of variation in  $y$  explained by the variation in the predictors. Algebraically,

$$R^2 = \frac{SSR}{SST} = 1 - \frac{SSE}{SST}$$

where:

- $SST$  Sum of Squares Total,  $\sum_{i=1}^N (y_i - \bar{y})^2$
- $SSR$ : Sum of Squares due to Regression,  $\sum_{i=1}^N (\hat{y}_i - \bar{y})^2$
- $SSE$ : Sum of Squares Error,  $\sum_{i=1}^N (\hat{y}_i - y_i)^2$

The R-squared (typically) lies within 0 and 1. An R-squared of 1 implies that the model fits the data perfectly, whereas an R-squared of 0 implies that the regression model does not explain any variation in  $y$ , or that  $\hat{y} = \bar{y}$  (the model is just predicting the mean of  $y$  for all observations, and the slope coefficients are zero).

```
In [ ]: 1 mpg_disp.rsquared
```

### 3. Adjusted R-squared

The R-squared necessarily increases for any predictor that's added to the data, even pure noise. The adjusted R-squared adjusts for this, and you should use this in place of the R-squared if comparing models with different numbers of predictors.

$$R^2_{Adj.} = 1 - \frac{(1-R^2)(N-1)}{N-p-1}$$

$N$  = size of data set

$p$  = number of predictors

Consider the following example where we add an additional predictor that's pure noise:

```
In [ ]: 1 mpg_disp.rsquared, mpg_disp.rsquared_adj
```

```
In [ ]: 1 rng = np.random.RandomState(seed = 42)
2 mtcars["noise"] = rng.rand(mtcars.shape[0])
3
4 mpg_disp_noise = lm(formula = "mpg ~ disp + noise", data = mtca
5 mpg_disp_noise.rsquared, mpg_disp_noise.rsquared_adj
```

Notice the R-squared has increased even though our additional predictor is pure noise. The adjusted R-squared measure, however, shows that the fit of our model has deteriorated.



## 4. F-test

The F-test tests for  $\hat{y} = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_p x_p$ :

**-H0:**  $\beta_1 = \beta_2 = \dots = \beta_p = 0$

**-H1:** At least one  $\beta_i$  is NOT zero  $i = 1, 2, \dots, p$

Roughly speaking, it tests whether having the model is an improvement over predicting the mean of  $y$  for all observations.

## Extra:

$$f - score = \frac{(SST - SSR)/p}{SSR/(N - p - 1)}$$

$$f - score \sim F(p, N - p - 1)$$

$N$  = size of data (number of the rows in the data)

$p$  = number of the predictors we used in the linear model

```
In [ ]: 1 print("F-score Value = {}".format(mpg_disp.fvalue))
        2
        3 print("F-test p-value = {}".format(mpg_disp.f_pvalue))
        4
        5 print ("High value of F-score and p-value < 0.05 proves 'beta0
```

## Model selection

Every model is too optimistic about how well it will actually predict.

*The Truth about Linear Regression*

## Resampling or train-test-split Approach

### Motivation

So far we have calculated only **in-sample** measures of model performance. This tells us nothing about how the model performs in practice.

A simple strategy is to split the data into a train and test (hold-out) set and then fit and tune the candidate models on train data, and eventually select a model that performs the best on the test dataset according to a chosen metric, such as Adjusted R-squared. This approach is known as the 'train-test split'. A problem with this approach is that it requires a lot of data.

```
In [ ]: 1 train = mtcars.sample(frac = .75)
        2 test = mtcars.drop(train.index)
```

```
In [ ]: 1 f"train size: {len(train)}, test_size: {len(test)}"
```

```
In [ ]: 1 mod = lm(formula = "mpg ~ disp", data = train)
        2 mod.predict(test)
```

```
In [ ]: 1 from sklearn.metrics import r2_score, mean_squared_error
        2 r2_score(mod.predict(test), test.mpg)
```

```
In [ ]: 1 mean_squared_error(mod.predict(test), test.mpg)
```

## Probabilistic Statistics Approach

One approach to model selection attempts to combine the complexity of the model with the performance of the model into a score, then select the model that minimizes (error) or maximizes the score.

Models are scored both on their performance on the training dataset and based on the complexity of the model. In regression the complexity of the model can be the number of predictors  $x_i$  s.

- **Model Performance:** How well a candidate model has performed on the training dataset.
- **Model Complexity:** How complicated the trained candidate model is after training.

Model performance may be evaluated using a probabilistic framework, such as **log-likelihood**.

In [ ]: 1 mpg\_disp.llf

## Log-Likelihood:

Assume we want to predict the value of  $y$  using the one predictor  $x$ . Therefore,  $y = \beta_0 + \beta_1 x + \epsilon$  where  $\epsilon \in N(0, \sigma)$  which means the error has normal distribution with  $mean = 0$  and standard deviation  $\sigma$ .

In Log-Likelihood technique we try to find  $\beta_0, \beta_1$  such that  $\epsilon \in N(0, \sigma)$  which means  $(y - (\beta_0 + \beta_1 x)) \in N(0, \sigma)$  or

$$P(y|x_1; \beta_0, \beta_1, \sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{-(y-(\beta_0+\beta_1 x))^2}{2\sigma^2}}$$

where  $\sigma$  is the standard deviation of the error which is unknown.

To minimize the error, we need to find  $\beta_0, \beta_1$  to maximize  $P(y|x_1; \beta_0, \beta_1, \sigma)$  for each data point. Therefore, for all the data points we need to find  $\beta_0, \beta_1$  to maximize the Likelihood function:

$$L_{data}(\beta_0, \beta_1, \sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} \prod_{(x,y) \in data} e^{\frac{-(y-(\beta_0+\beta_1 x))^2}{2\sigma^2}}.$$

To make the above equation simpler, we can take the log of the Likelihood and rewrite the it as:

$$l_{data}(\beta_0, \beta_1, \sigma) = \log\left(\frac{1}{\sqrt{2\pi\sigma^2}} \prod_{(x,y) \in data} e^{\frac{-(y-(\beta_0+\beta_1 x))^2}{2\sigma^2}}\right) = -\log(\sqrt{2\pi\sigma^2}) - \frac{1}{2\sigma^2} \sum_{(x,y) \in data}$$

If we assume  $\hat{y} = \beta_0 + \beta_1 x$ , we can rewrite the above formula as:

$$-l_{data}(\beta_0, \beta_1, \sigma) = \log(\sqrt{2\pi\sigma^2}) + \frac{1}{2\sigma^2} \sum_{(x,y) \in data} (y - \hat{y})^2.$$

Therefore, to find  $\beta_0, \beta_1$  to **maximize** Likelihood  $L_{data}(\beta_0, \beta_1, \sigma)$  we need to **minimize**:

$$-l_{data}(\beta_0, \beta_1, \sigma) = \log(\sqrt{2\pi\sigma^2}) + \frac{1}{2\sigma^2} \sum_{(x,y) \in data} (y - \hat{y})^2$$

which is equivalent to minimizing  $\sum_{(x,y) \in data} (y - \hat{y})^2$ .

In Python we can get the value of log-likelihood from the summary of the linear model using: `model.llf`

## AIC: Akaike Information Criterion

The Akaike Information Criterion, or AIC for short, is a method for scoring and selecting models which have been fitted under the maximum likelihood estimation framework. To use AIC for model selection, we simply choose the model giving smallest AIC over the set of models considered. The AIC statistic penalizes complex models but puts more emphasis on model performance on the training dataset, and, in turn, select more complex models.

$$AIC = -2\text{LogLikelihood} + 2(\text{number of predictors} + 1)$$

We can directly get AIC from the model summary: `model.aic`

```
In [ ]: 1 AIC = -2 * mpg_disp.llf + 2 * (mpg_disp.df_model+1)
        2 AIC
```

```
In [ ]: 1 mpg_disp.aic
```

## BIC: Bayesian Information Criterion

BIC is another method for scoring and selecting models under the maximum likelihood estimation framework. We choose the model with minimum BIC as the best model. Unlike the AIC, the BIC penalizes the model more for its complexity, meaning that more complex models will have a worse (larger) score and will, in turn, be less likely to be selected.

$$BIC = -2\text{LogLikelihood} + \log(\text{size of dataset}) \times (\text{number of predictors} + 1)$$

We can directly get BIC from the model summary: `model.bic`

```
In [ ]: 1 BIC = -2 * mpg_disp.llf + np.log(mpg_disp.nobs) * (mpg_disp.df_
        2 BIC
```

```
In [ ]: 1 mpg_disp.bic
```

### Exercise:

In the above exercise, we fitted two linear models to predict the `selling_price`. The first model only used `area` as the predictor, but the second model uses both `area` and `living_space_size`. Which model is better?

- Choose the best model based on the train/test split approach.
- Choose the best model based on the probabilistic statistics approach.

**Attention:** *R-squared* always increases if you increase the number of predictors. Therefore, we only use *R-squared* to check the performance of ONE model. However, to compare the performance of two models we need to compare *Mean Squared Error* on test set, or *Adjusted R-squared* or *AIC* or *BIC* on the whole data.

In [ ]: 1 # Your code here

### Exercise:

Repeat the above exercise and check if adding `n_rooms` to the list of predictors improve the model. Use both train/test split and probabilistic approaches.

In [ ]: 1 # Your code here

## Linear Regression Assumptions and diagnostics

When we use linear regression model we need to be careful about the assumptions that make our model a good fit for our data.

### Ideal conditions for regression

- **IC1** Regression assumes that your data is generated from the process

$$y_i = \beta_0 + \beta_1 x_{1i} + \dots + \beta_n x_{ni} + \epsilon_i$$

- **IC2** The errors are independent of the predictors for all observations.

These two ideal conditions imply the following assumptions:

1. *Validity*. Most importantly, the data you are analyzing should map to the research question you are trying to answer. For example, with regard to the outcome variable, a model of house age and squared area will tell you about house selling price and you may need to choose necessary predictors carefully. For data collection you also need to choose a representative sample of the population. For instance, a model of predicting salary based on the work experience and title may not be accurate if you sample data includes male employees, unless you prove gender does not affect salary.
2. *Additivity and linearity*. The most important mathematical assumption of the regression model is that:

$$y = \beta_0 + \beta_1 x_1 + \dots + \beta_n x_n + \epsilon$$

If additivity is violated, and for example,  $y = x_1 \times x_2 \times x_n + \epsilon$ , then you need to first transform you data and rewrite the above formula as:  
 $\log(y) = \log(x_1) + \log(x_2) + \log(x_n) + \epsilon$  ro you need to add interaction terms to your linear regression.

3. *Independence of errors*. Regression model assumes that the errors,  $\epsilon_i$ s from the prediction line are independent.
4. *Equal variance of errors*. This assumption states that the variance in the error is equal at each point. Here, variance refers to the data spread. Equal variance of errors is known as **homoscedasticity**. Unequal variance of errors is called **heteroscedasticity**. To check for equal variance of errors, we check to see if there's any pattern in the distribution of the residuals around the x-axis.
5. *Normality of errors*. This regression assumes that the errors are normally distributed.



## Model Validation

### Normality

One of the ideal conditions of OLS is that the errors are normally distributed. Note that  $y$  does not have to be normally distributed, just  $\epsilon$  vs.  $X$  (residuals).

Normality is linked to outliers, which often result in *fat tails* or *skewness*.

Consider the following distribution:

```
In [ ]: 1 salary_dist = np.random.randint(4000, 7000, size = 20)
        2 salary_dist = np.append(salary_dist, [-99, 30000, 20000, 25000,
        3 sns.distplot(salary_dist);
```

The value **-99** is obviously a missing value indicator, and should be removed as an outlier. The high values would be considered outliers by rules such as Tukey's method, but may be the most informative on the phenomenon that we're investigating.

## Skewness and kurtosis

Loosely speaking,

- Skewness describes how skewed the distribution is.
- Kurtosis describes how fat the tails are.

**Notice that regressing on the data with outliers results in high skew and kurtosis in the residuals.**

### Skewness Review

If a skewness value of greater than 1 is obtained in either direction (positive/negative), we say that the distribution is highly skewed. A skewness value of 0 represents a perfectly symmetric distribution, which in the case of real-world data is close to impossible to observe.

### Kurtosis Review

1. If the kurtosis is **negative** ( $< 0$ ), we say the distribution is **platykurtic**. This means its tails are **thinner** than that of a Normal distribution. Visually, platykurtic distributions appear **shorter** in height compared to a Normal distribution.
2. If the kurtosis value is **zero** ( $= 0$ ), we say the distribution is **mesokurtic**. This means the thickness of its tails are **identical/similar** to that of a Normal distribution.
3. If the kurtosis value is **positive** ( $> 0$ ), we say the distribution is **leptokurtic**. This means its tails are **thicker** than that of a Normal distribution. Visually, leptokurtic distributions appear **taller** in height compared to a Normal distribution.

```
In [ ]: 1 salary_dist
```

```
In [ ]: 1 # salary_dist is np.array so we need to use skew() and kurtosis
        2 from scipy.stats import skew, kurtosis
        3 skew(salary_dist), kurtosis(salary_dist)
```

```
In [ ]: 1 mod = lm(formula = "mpg ~ disp", data = mtcars)
        2 mod.params
```

```
In [ ]: 1 sns.distplot(mod.resid).set_title( 'mpg ~ disp; Residuals Histo
```

```
In [ ]: 1 # mod.resid is dataframe so we can use skew() and kurt() from p
        2 mod.resid.skew(), mod.resid.kurt()
```

## When does it *not* matter?

As the sample size grows, the normality assumption for the residuals is not needed.

## When *might* it matter?

Outliers can bias the slope coefficient if they do not reflect the underlying linear process.

```
In [ ]: 1 ### Residual diagnostics
        2 ### Outliers and influence on slope parameter
```

```
In [ ]: 1 mtcars_outlier = mtcars.loc[:, ["disp", "mpg"]]
        2 mtcars_outlier["outlier"] = False
        3 mtcars_outlier = mtcars_outlier.append(pd.DataFrame({"disp": [4
        4                                                                "mpg": [35
        5                                                                "outlier":
        6
        7 mod2 = lm(formula = "mpg ~ disp", data = mtcars_outlier)
        8 mod2.params
```

```
In [ ]: 1 plt.scatter(y=mod2.resid, x=mtcars_outlier.disp)
        2 plt.axhline(np.mean(mod2.resid), color='red');
```

```
In [ ]: 1 mod2.resid.skew(), mod2.resid.kurt()
```

Compare this to regressing without the outlier:



```

In [ ]: 1 ax = plt.subplot()
2 fitted_2 = lambda x: mod2.params[0] + mod2.params[1] * x
3 fitted_1 = lambda x: mod.params[0] + mod.params[1] * x
4 xmin=np.min(mtcars_outlier.disp)
5 xmax=np.max(mtcars_outlier.disp)
6 xval=np.linspace(xmin, xmax, 100)
7 yhat_2 = fitted_2(xval)
8 yhat_1 = fitted_1(xval)
9
10 plt.scatter(mtcars_outlier.disp, mtcars_outlier.mpg,
11             c = mtcars_outlier.outlier, cmap="Spectral")
12 plt.plot(xval, yhat_2, "-", c = "darkslateblue", label = 'With
13 plt.plot(xval, yhat_1, "-", c = "darkred", label = 'Without OUT
14 plt.title("The outlier makes the regression line flatter than i
15 plt.legend();

```

## How to deal with outliers?

- Deletion

We remove outliers only if we have good reasons to do that. For example, we know those outliers are wrong numbers have been recorded by mistake, or there is no way that we can fix them. For instance, if age of a patient has been recorded as zero, we are sure this value is wrong and if we do not have any other information about this patient, we cannot fix this wrong number. Also, if the outlier is from another population we are studying, we may want to delete that. Furthermore, you may want to delete a few weird data points if you already have a good model and it will be much better model if you get rid of those a few weird data points.

- Change model, e.g. functional form

Sometimes data points can look like outliers because we are looking at the wrong pattern. For instance, if a quadratic model (e.g.  $\hat{y} = \beta_0 + \beta_1 x_1 + \beta_2 x_1^2$ ) is a better fit for our data, but we insist to use linear model without quadratic terms, some part of our data points which are far from the line estimate may look like outliers. In this case, We need to change our model and add polynomial terms to the linear regression to get a better fit for the data.

## Residual diagnostics

Residuals are differences between the actual value of  $y$  and the estimated value  $\hat{y}$ .

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_n x_n + \epsilon$$

$$\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x_1 + \hat{\beta}_2 x_2 + \dots + \hat{\beta}_n x_n$$

$$residual = e = y - \hat{y}$$

We can get the list of the residuals from python by `model.resid`.

Under the linear regression assumption:

$\epsilon \sim Normal(0, \sigma)$ . For a good linear regression we need to have

$e \sim Normal(0, \hat{\sigma})$ .

## Residual plots

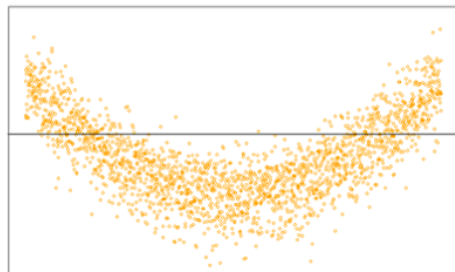
### Plot the Residuals Against the Predictor

Residual plots visualise our estimates of  $\epsilon_i$ s against each predictor.

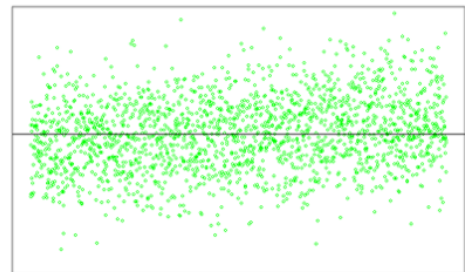
- **What to look out for?** We assumed our residuals were 'random'. Therefore, a good regression should have no pattern in the residuals; specifically:
  - flat scatter about 0
  - variation approximately constant and not change with predictors
- If there is a pattern in the residual plot (e.g. curvature), we may be able to fit the data better if we add other predictors or polynomial terms into our linear regression model.
- If the **variation** in the residuals changes with the predictor variable, we may have heteroscedasticity in our residuals perhaps due to getting the functional form of the regression wrong.

## Residual by Predictor

Curvature Pattern



No Pattern



**Example:** Plot residuals against `disp` for linear model `mpg ~ disp` in both `mtcars` and `mtcars_outlier` data sets.

```
In [ ]: 1 # plt Residual plots mpg ~ disp
2 fg, ax = plt.subplots(ncols = 2, figsize=(12,5))
3
4 ax[0].scatter(mtcars.disp, mod.resid)
5 ax[0].axhline(np.mean(mod.resid), color='red')
6 ax[1].scatter(mtcars_outlier.disp, mod2.resid, c=mtcars_outlier)
7 ax[1].axhline(np.mean(mod2.resid), color='red')
8
9
10 ax[0].set_title("Residual by disp: No Outlier")
11 ax[0].set_xlabel("disp")
12 ax[1].set_title("Residual by disp: With Outlier")
13 ax[1].set_xlabel("disp")
14
15
16 fg.tight_layout()
```

```
In [ ]: 1 # Seaborn Residual plots mpg ~ disp
2 sns.residplot('disp', mod.resid, data=mtcars)
3 plt.xlabel("disp")
4 plt.title('Residual plot')
```

**Exercise:** Plot residuals against room for linear model price ~ rooms in hprice2 data set.

```
In [ ]: 1 hprice2 = pd.read_csv("../data/hprice2.csv")
```

```
In [ ]: 1 # Your code here
```

### Plot the Residuals Against Another Variable

If the residual plot against another predictor is flat, then this predictor may not improve our model and it is not a good predictor to be added to the linear model we already have. However, if the residual plot against this new predictor shows some patterns we may want to add this new predictor to our current model.

**Example:** Plot residuals of the linear model mpg ~ disp against wt in mtcars data sets. Does adding this predictor to the model results in better performance?

```
In [ ]: 1 plt.scatter(mtcars.wt, mod.resid)
2 plt.axhline(np.mean(mod.resid), color='red');
```

```
In [ ]: 1 mod3 = lm('mpg ~ disp + wt', mtcars)
2 plt.scatter(mtcars.hp, mod3.resid)
3 plt.axhline(np.mean(mod3.resid), color='red');
```

```
In [ ]: 1 print('mpg ~ disp : MSE = {}'.format(mean_squared_error(mtcars.mpg, mod.predict())))
2         .format(mean_squared_error(mtcars.mpg, mod.predict())))
3 print('mpg ~ disp : Adjusted R-squared= {}'.format(mod.rsquared))
4 print('mpg ~ disp + wt : MSE = {}'.format(mean_squared_error(mtcars.mpg, mod3.predict())))
5         .format(mean_squared_error(mtcars.mpg, mod3.predict())))
6 print('mpg ~ disp + wt : Adjusted R-squared= {}'.format(mod3.rsquared))
```

**Example:** Plot residuals of the linear model  $\text{mpg} \sim \text{disp} + \text{wt}$  against  $\text{hp}$  in `mtcars` data sets. Does adding this predictor to the model results in better performance?

```
In [ ]: 1 plt.scatter(mtcars.hp, mod3.resid)
2 plt.axhline(np.mean(mod3.resid), color='red');
```

```
In [ ]: 1 mod4 = lm('mpg ~ disp + wt + hp', mtcars)
2 plt.scatter(mtcars.wt, mod4.resid)
3 plt.axhline(np.mean(mod4.resid), color='red');
```

```
In [ ]: 1 print('mpg ~ disp : MSE = {}'.format(mean_squared_error(mtcars.mpg, mod.predict())))
2         .format(mean_squared_error(mtcars.mpg, mod.predict())))
3 print('mpg ~ disp : Adjusted R-squared= {}'.format(mod.rsquared))
4
5 print('mpg ~ disp + wt : MSE = {}'.format(mean_squared_error(mtcars.mpg, mod3.predict())))
6         .format(mean_squared_error(mtcars.mpg, mod3.predict())))
7 print('mpg ~ disp + wt : Adjusted R-squared= {}'.format(mod3.rsquared))
8
9 print('mpg ~ disp + wt + hp : MSE = {}'.format(mean_squared_error(mtcars.mpg, mod4.predict())))
10        .format(mean_squared_error(mtcars.mpg, mod4.predict())))
11 print('mpg ~ disp + wt + hp : Adjusted R-squared= {}'.format(mod4.rsquared))
```

## Plot the Magnitude of the Residuals Against the Predictor

By plotting the squared residuals against the predictor variable we can check whether the variance of the residuals is constant. As the average of the squared residuals is MSE (Mean Squared Error), the scatter plot should normally give us a horizontal line and points around this line, whose height should be around the in-sample MSE.

**\*\*If there are Regions of the x axis (predictor) where the residuals are persistently above or below this line, then the problem is associated with the heteroskedasticity in the residuals and there should be a problem with our linear model possibly due to wrong functional form.**

**Example:** Plot residual squares against  $\text{disp}$  for  $\text{mpg} \sim \text{disp}$ ,  $\text{mpg} \sim \text{disp} + \text{wt}$  in `mtcars` data sets.

```
In [ ]: 1 fig, axes = plt.subplots(nrows=2, figsize=(10,8))
2 axes[0].scatter(mtcars.disp, mod.resid**2)
3 axes[0].axhline(np.mean(mod.resid**2), color='red')
4 axes[0].set_title('mpg ~ disp ; Against disp')
5
6 axes[1].scatter(mtcars.disp, mod3.resid**2)
7 axes[1].axhline(np.mean(mod3.resid**2), color='red')
8 axes[1].set_title('mpg ~ disp + wt ; Against disp');
```

**Exercise:** Plot residual squares against crime, rooms, and lowstat for price ~ rooms, price ~ rooms + crime and price ~ crime + rooms + lowstat in hprice2 data sets.


```
In [ ]: 1 # Your code here
```

## Plot the Distribution of the Residuals

The residuals should follow a Normal distribution with mean zero.

**Example:** Plot the histogram of the residuals for price ~ rooms + crime + lowstat in hprice2 data sets.

```
In [ ]: 1 sns.distplot(mod_p3.resid).set_title( 'price ~ rooms + crime +
```



## QQ plot

Compares the sample quantiles of the residuals against the theoretical quantiles of the normal distribution.

```
In [ ]: 1 from statsmodels.graphics.gofplots import qqplot
```

```
In [ ]: 1 fg, ax = plt.subplots(2, 2, figsize=(10,5))
        2
        3 sns.distplot(mod2.resid, ax = ax[0,1])
        4 sns.distplot(mod.resid, ax = ax[0,0])
        5 qqplot(mod.resid , fit = True, line = "45", ax=ax[1,0])
        6 qqplot(mod2.resid, fit = True, line = "45", ax= ax[1,1])
        7
        8
        9 ax[0,0].set_title("Residuals Distribution: No Outlier")
       10 ax[0,1].set_title("Residuals Distribution: With Outlier")
       11 ax[1,0].set_title("QQ Plot: No Outlier")
       12 ax[1,0].set_xlabel("disp")
       13 ax[1,1].set_title("QQ Plot of residuals")
       14 ax[1,1].set_xlabel("disp");
       15
       16 fg.tight_layout()
       17
```

## Solution

1. **Remove outliers.** This may introduce selection bias, so be cautious. Outliers that can be corrected should be corrected, e.g. data recording errors.
2. **Change the model.** Right-skew in the data is a common issue. In the case of right-skewed predictors, the logarithmic transformation of the predictor will take on a more 'normal' distribution.

**Example:**

- a- Create a linear regression model that predicts `prestige` based on `education`, `income`, and `women`.
- b- Evaluate the model performance using train-test-split approach. Use MSE and R2 as metrics
- c- Do Residual diagnostics
- d- Improve the model accuracy

Data set Prestige includes 6 columns with the following format:

- education: Average education of occupational incumbents, years, in 1971.
- income: Average income, dollars, in 1971.
- women: Percentage of incumbents who are women.
- prestige: Pineo-Porter prestige score for occupation, from a social survey conducted in the mid-1960s.
- census: Canadian Census occupational code.
- type: Type of occupation. A factor with levels (note: out of order): bc, Blue Collar; prof, Professional, Managerial, and Technical; wc, White Collar.

**Source:**

Canada (1971) Census of Canada. Vol. 3, Part 6. Statistics Canada [pp. 19-1-19-21].

```
In [ ]: 1 # import
        2 Prestige = pd.read_csv("../data/Prestige.csv")
        3 Prestige.head()
```

```
In [ ]: 1 # train-test split
        2 train = Prestige.sample(frac = .7)
        3 test = Prestige.drop(train.index)
        4
        5 # fit
        6 mod = lm(formula = "prestige ~ income + education + women",
        7           data = train)
        8 print(mod.params)
```

```
In [ ]: 1 mod_r2=r2_score(mod.predict(test), test.prestige)
        2 mod_mse=mean_squared_error(mod.predict(test), test.prestige)
        3 print('prestige ~ income + education + women; R2 = {:.2f}'.format(mod_r2))
        4 print('prestige ~ income + education + women; MSE = {:.2f}'.format(mod_mse))
```



```

In [ ]: 1 # Residual Diagnostics
2 fg, ax = plt.subplots(nrows = 2,
3                       ncols = 3,
4                       figsize = (20, 10))
5 model1 = lm(formula = "prestige ~ income + education + women",
6             data = Prestige)
7
8 i = 0
9 for col in ["income", "education", "women"]:
10     ax[0, i].scatter(Prestige[col], model1.resid)
11     ax[0, i].axhline(np.mean(model1.resid), c='red')
12     ax[0, i].set_title(f"Residuals against {col}")
13     ax[0, i].set_ylabel("Residuals")
14     ax[0, i].set_xlabel(col)
15
16     ax[1, i].scatter(Prestige[col], model1.resid ** 2)
17     ax[1, i].axhline(np.mean(model1.resid**2), c='red')
18     ax[1, i].set_title(f"$Residuals ^ 2$ against {col}")
19     ax[1, i].set_ylabel("$Residuals ^ 2$ ")
20     ax[1, i].set_xlabel(col)
21
22     i += 1
23
24 fg.tight_layout()
25 plt.show()

```

```

In [ ]: 1 model2 = lm(formula = "prestige ~ np.log(income) + education +
2             data = Prestige)
3 print('prestige ~ income + education + women; Adjusted R-square
4 print('prestige ~ income + education + women; AIC = {}'.format(
5 print('prestige ~ income + education + women; BIC = {}\n'.forma
6
7 print('prestige ~ np.log(income) + education + women; Adjusted
8 print('prestige ~ np.log(income) + education + women; AIC = {}'
9 print('prestige ~ np.log(income) + education + women; BIC = {}\n

```

```

In [ ]: 1 # Q-Q plot
2 qqplot(model2.resid, fit = True, line = "45")
3 plt.show()

```

```

In [ ]: 1 # income doesn't seem quite normally distributed - take log
2 # fit
3 mod2 = lm(formula = "prestige ~ np.log(income) + education + wo
4             data = train)
5 print(mod2.params)
6 mod2_r2=r2_score(mod2.predict(test), test.prestige)
7 mod2_mse=mean_squared_error(mod2.predict(test), test.prestige)
8 print('prestige ~ np.log(income) + education + women; R2 = {:.2
9 print('prestige ~ np.log(income) + education + women; MSE = {:.

```

**Exercise:** In advertising data set, design a linear regression model to estimate sales .

```
In [ ]: 1 ad=pd.read_csv('../data/Advertising.csv')
        2 ad.head()
```

```
In [ ]: 1 ad=ad.drop('Unnamed: 0', axis=1)
        2 ad.describe()
```

```
In [ ]: 1 # Your code here
```

# Reference

- Linear Regression General: <http://home.iitk.ac.in/~shalab/econometrics/Chapter2-Econometrics-SimpleLinearRegressionAnalysis.pdf>  
(<http://home.iitk.ac.in/~shalab/econometrics/Chapter2-Econometrics-SimpleLinearRegressionAnalysis.pdf>)
- Linear Regression General: <https://online.stat.psu.edu/stat501/lesson/6/6.1>  
(<https://online.stat.psu.edu/stat501/lesson/6/6.1>)
- Linear Regression Coefficients Sampling distribution: <https://www.econometrics-with-r.org/4-5-tsdoe.html> (<https://www.econometrics-with-r.org/4-5-tsdoe.html>)
- Linear Regression Coefficients Sampling distribution:  
<https://scholar.princeton.edu/sites/default/files/bstewart/files/lecture5handout.pdf>  
(<https://scholar.princeton.edu/sites/default/files/bstewart/files/lecture5handout.pdf>)
- Linear Regression Coefficients Sampling distribution:  
[http://www.robots.ox.ac.uk/~fwood/teaching/W4315\\_Fall2011/Lectures/lecture\\_4/lect](http://www.robots.ox.ac.uk/~fwood/teaching/W4315_Fall2011/Lectures/lecture_4/lect)  
([http://www.robots.ox.ac.uk/~fwood/teaching/W4315\\_Fall2011/Lectures/lecture\\_4/lect](http://www.robots.ox.ac.uk/~fwood/teaching/W4315_Fall2011/Lectures/lecture_4/lect))
- Assumptions Of Linear Regression Algorithm:  
<https://towardsdatascience.com/assumptions-of-linear-regression-algorithm-ed9ea32224e1> (<https://towardsdatascience.com/assumptions-of-linear-regression-algorithm-ed9ea32224e1>)
- Residual Diagnosis:  
<http://www.cs.cornell.edu/courses/cs1380/2018sp/textbook/chapters/13/5/visual-diagnostics.html>  
(<http://www.cs.cornell.edu/courses/cs1380/2018sp/textbook/chapters/13/5/visual-diagnostics.html>)
- Linear Regression & Inference Rules (Video):  
<https://www.khanacademy.org/math/ap-statistics/inference-slope-linear-regression/inference-slope/v/intro-inference-slope>  
(<https://www.khanacademy.org/math/ap-statistics/inference-slope-linear-regression/inference-slope/v/intro-inference-slope>)
- The Complete Guide to Linear Regression in Python:  
<https://towardsdatascience.com/the-complete-guide-to-linear-regression-in-python-3d3f8f06bf8> (<https://towardsdatascience.com/the-complete-guide-to-linear-regression-in-python-3d3f8f06bf8>)
- Model Selection: <https://machinelearningmastery.com/probabilistic-model-selection-measures/> (<https://machinelearningmastery.com/probabilistic-model-selection-measures/>)

