

```
In [2]: import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import statsmodels.api as sm
import statsmodels.formula.api as smf
```

### Example: automobile data

<https://www.kaggle.com/toramky/automobile-dataset>

```
In [4]: carsdata=pd.read_csv('.../data/Automobile_data.csv')
carsdata.sample(5)
```

Out[4]:

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engi-local
48	0	?	jaguar	gas	std	four	sedan	rwd	fi
51	1	104	mazda	gas	std	two	hatchback	fwd	fi
92	1	122	nissan	gas	std	four	sedan	fwd	fi
31	2	137	honda	gas	std	two	hatchback	fwd	fi
49	0	?	jaguar	gas	std	two	sedan	rwd	fi

5 rows × 26 columns

To fit a model to find important predictors of `price`, let's start with some predictors to fit an initial model.

```
In [6]: cars=carsdata[['engine-size','horsepower','city-mpg','price']].copy()
#names have been giving me troubles..
cars.columns = ['enginesize', 'horsepower', 'citympg', 'price']
```

```
In [7]: #deal with missing values: the missing values here are "?"
cars['enginesize'].replace('?', np.nan, inplace= True)
cars['horsepower'].replace('?', np.nan, inplace= True)
cars['citympg'].replace('?', np.nan, inplace= True)
cars['price'].replace('?', np.nan, inplace= True)
cars.isnull().sum()
```

```
Out[7]: enginesize      0
horsepower      2
citympg        0
price          4
dtype: int64
```

```
In [8]: #Not too many missing values, for now let's drop them
#Recall how we could impute missing values from EDA
carsnew=cars.dropna()
```

```

carsnew.info()
carsnew.isnull().sum()

<class 'pandas.core.frame.DataFrame'>
Index: 199 entries, 0 to 204
Data columns (total 4 columns):
 #   Column      Non-Null Count  Dtype  
--- 
 0   enginesize   199 non-null    int64  
 1   horsepower   199 non-null    object  
 2   citympg     199 non-null    int64  
 3   price        199 non-null    object  
dtypes: int64(2), object(2)
memory usage: 7.8+ KB

```

```

Out[8]: enginesize      0
         horsepower     0
         citympg       0
         price          0
         dtype: int64

```

```

In [9]: #one more problem: price is a object, not int64(numeric), so is horsepower!
carsnew['price']=carsnew['price'].astype('int64')
carsnew['horsepower']=carsnew['horsepower'].astype('int64')
carsnew.info()

```

```

<class 'pandas.core.frame.DataFrame'>
Index: 199 entries, 0 to 204
Data columns (total 4 columns):
 #   Column      Non-Null Count  Dtype  
--- 
 0   enginesize   199 non-null    int64  
 1   horsepower   199 non-null    int64  
 2   citympg     199 non-null    int64  
 3   price        199 non-null    int64  
dtypes: int64(4)
memory usage: 7.8 KB

```

```

/var/folders/xp/wk0n_cc925g6bz7szq91rxhxpdk82n/T/ipykernel_26953/2633004782.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

```

See the caveats in the documentation: [https://pandas.pydata.org/pandas-docs/stable/user\\_guide/indexing.html#returning-a-view-versus-a-copy](https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)

```

carsnew['price']=carsnew['price'].astype('int64')
/var/folders/xp/wk0n_cc925g6bz7szq91rxhxpdk82n/T/ipykernel_26953/2633004782.py:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

```

See the caveats in the documentation: [https://pandas.pydata.org/pandas-docs/stable/user\\_guide/indexing.html#returning-a-view-versus-a-copy](https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)

```

carsnew['horsepower']=carsnew['horsepower'].astype('int64')

```

Now we can fit mlr: price~enginesize + citympg+ horsepower

```
In [11]: reg = smf.ols('price~enginesize+citympg+horsepower', data=carsnew).fit()

#print(dir(reg)) #members of the object provided by the modelling
```

```
In [12]: #Model summary
reg.summary()
```

Out[12]: OLS Regression Results

<b>Dep. Variable:</b>	price	<b>R-squared:</b>	0.794			
<b>Model:</b>	OLS	<b>Adj. R-squared:</b>	0.791			
<b>Method:</b>	Least Squares	<b>F-statistic:</b>	251.2			
<b>Date:</b>	Mon, 16 Sep 2024	<b>Prob (F-statistic):</b>	1.03e-66			
<b>Time:</b>	17:07:18	<b>Log-Likelihood:</b>	-1912.4			
<b>No. Observations:</b>	199	<b>AIC:</b>	3833.			
<b>Df Residuals:</b>	195	<b>BIC:</b>	3846.			
<b>Df Model:</b>	3					
<b>Covariance Type:</b>	nonrobust					
	coef	std err	t	P> t	[0.025	0.975]
<b>Intercept</b>	-2547.0951	2914.668	-0.874	0.383	-8295.415	3201.225
<b>enginesize</b>	124.3388	10.951	11.355	0.000	102.742	145.935
<b>citympg</b>	-151.3184	70.849	-2.136	0.034	-291.047	-11.590
<b>horsepower</b>	37.0876	16.262	2.281	0.024	5.016	69.159
<b>Omnibus:</b>	10.990	<b>Durbin-Watson:</b>	0.771			
<b>Prob(Omnibus):</b>	0.004	<b>Jarque-Bera (JB):</b>	17.130			
<b>Skew:</b>	0.312	<b>Prob(JB):</b>	0.000191			
<b>Kurtosis:</b>	4.295	<b>Cond. No.</b>	1.96e+03			

### Notes:

- [1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
- [2] The condition number is large, 1.96e+03. This might indicate that there are strong multicollinearity or other numerical problems.

### ANOVA / F test

- from global anova: the model is significant compared to the null model
- from t test: enginesize, citympg, horsepower are all significant predictors of price

But recall that when we extract `anova_lm` separately, different `type` can give us the results of either **sequential F test** results or **partial F test** results.

In [14]: `# type1 and type2 in anova_lm gives different values in SS and F test  
sm.stats.anova_lm(reg, typ=1)`

*SST* Out [14]:

	df	sum_sq	mean_sq	F	PR(>F)	H <sub>0</sub>	H <sub>1</sub>
enginesize	1.0	9.625888e+09	9.625888e+09	724.441229	1.387785e-67	$H_0: \gamma_1 = 0$	$H_1: \gamma_1 \neq 0$
citympg	1.0	3.186064e+08	3.186064e+08	23.978216	2.039654e-06	$H_0: \gamma_{eng} = \gamma_{citympg} = 0$	$H_1: \gamma_{eng} \neq 0 \text{ or } \gamma_{citympg} \neq 0$
horsepower	1.0	6.911228e+07	6.911228e+07	5.201368	2.365037e-02	$H_0: \gamma_{eng} + \gamma_{citympg} = 0$	$H_1: \gamma_{eng} + \gamma_{citympg} \neq 0$
Residual	195.0	2.591029e+09	1.328733e+07	NaN	NaN		Full model

VS

In [15]: `sm.stats.anova_lm(reg, typ=2)`

*SST* Out [15]:

	sum_sq	df	F	PR(>F)
enginesize	1.713088e+09	1.0	128.926426	2.904006e-23
citympg	6.061127e+07	1.0	4.561585	3.394397e-02
horsepower	6.911228e+07	1.0	5.201368	2.365037e-02
Residual	2.591029e+09	195.0	NaN	NaN

$$2.9 \times 10^{-23} = 0.000\ldots 029$$

$$3.3 \times 10^{-2} = .03394$$

In [16]:

`# type-3 is the same as typ=2, just added a test on intercept  
sm.stats.anova_lm(reg, typ= 3)`

Out [16]:

	sum_sq	df	F	PR(>F)
Intercept	1.014728e+07	1.0	0.763681	3.832538e-01
enginesize	1.713088e+09	1.0	128.926426	2.904006e-23
citympg	6.061127e+07	1.0	4.561585	3.394397e-02
horsepower	6.911228e+07	1.0	5.201368	2.365037e-02
Residual	2.591029e+09	195.0	NaN	NaN

In [17]:

`#Take a closer look at typ=1: reduced model without citympg  
reg_reduced_1 = smf.ols('price~enginesize', data=carsnew).fit()  
sm.stats.anova_lm(reg_reduced_1, typ=1)`

Out [17]:

	df	sum_sq	mean_sq	F	PR(>F)
enginesize	1.0	9.625888e+09	9.625888e+09	636.609809	1.265067e-63
Residual	197.0	2.978748e+09	1.512055e+07	NaN	NaN

In [18]:

`#Take a closer look at typ=1: full model with citympg  
reg_full_1 = smf.ols('price~enginesize+citympg', data=carsnew).fit()`

```
sm.stats.anova_lm(reg_full_1, typ=1)
```

	df	sum_sq	mean_sq	F	PR(>F)
<b>enginesize</b>	1.0	9.625888e+09	9.625888e+09	709.238315	4.852522e-67
<b>citympg</b>	1.0	3.186064e+08	3.186064e+08	23.475016	2.566218e-06
<b>Residual</b>	196.0	2.660141e+09	1.357215e+07	NaN	NaN

The difference in SSE is the same as the SS for citympg in  
 $y \sim \text{enginesize} + \text{citympg} + \text{horsepower}$

```
In [20]: sm.stats.anova_lm(reg, typ=1)
```

	df	sum_sq	mean_sq	F	PR(>F)
<b>enginesize</b>	1.0	9.625888e+09	9.625888e+09	724.441229	1.387785e-67
<b>citympg</b>	1.0	3.186064e+08	3.186064e+08	23.978216	2.039654e-06
<b>horsepower</b>	1.0	6.911228e+07	6.911228e+07	5.201368	2.365037e-02
<b>Residual</b>	195.0	2.591029e+09	1.328733e+07	NaN	NaN

Change orders in typ=1 changes the result because you are comparing different reduced and full model now

```
In [22]: reg_order = smf.ols('price~citympg+horsepower+enginesize', data=carsnew).fit()
sm.stats.anova_lm(reg_order, typ=1)
```

	df	sum_sq	mean_sq	F	PR(>F)
<b>citympg</b>	1.0	5.988084e+09	5.988084e+09	450.661312	1.369003e-52
<b>horsepower</b>	1.0	2.312434e+09	2.312434e+09	174.033076	8.065939e-29
<b>enginesize</b>	1.0	1.713088e+09	1.713088e+09	128.926426	2.904006e-23
<b>Residual</b>	195.0	2.591029e+09	1.328733e+07	NaN	NaN

### From sequential F tests

- We can kind of "rank" the importance of the predictors.
- Since in the above example all predictors are significant, we will leave this to the next example.

```
In [24]: #Take a closer look at type2: reduced model without citympg
reg_reduced_nomp = smf.ols('price~enginesize+horsepower', data=carsnew).fit()
sm.stats.anova_lm(reg_reduced_nomp, typ = 2)
#the difference in SSE is the SS of citympg in the full model
```

Out[24]:

	sum_sq	df	F	PR(>F)
<b>enginesize</b>	1.672205e+09	1.0	123.603577	1.403756e-22
<b>horsepower</b>	3.271074e+08	1.0	24.178641	1.852652e-06
<b>Residual</b>	2.651640e+09	196.0	NaN	NaN

In [25]: sm.stats.anova\_lm(reg, typ=2)

Out[25]:

	sum_sq	df	F	PR(>F)
<b>enginesize</b>	1.713088e+09	1.0	128.926426	2.904006e-23
<del>* citympg</del>	6.061127e+07	1.0	4.561585	3.394397e-02
<b>horsepower</b>	6.911228e+07	1.0	5.201368	2.365037e-02
<b>Residual</b>	2.591029e+09	195.0	NaN	NaN

In [26]: #citympg 6.061127e+07 = Residual(red) 2.651640e+09 - Residual(full)

In [27]: 2.651640e+09 - 2.591029e+09 #approx bc of sigfig issue

Out[27]: 60611000.0

In [28]: reg\_reduced\_noengine = smf.ols('price~citympg+horsepower', data=carsnew).fit()  
sm.stats.anova\_lm(reg\_reduced\_noengine, typ=2)

Out[28]:

	sum_sq	df	F	PR(>F)
<b>citympg</b>	1.972876e+07	1.0	0.898404	3.443771e-01
<b>horsepower</b>	2.312434e+09	1.0	105.303180	4.767363e-20
<b>Residual</b>	4.304117e+09	196.0	NaN	NaN

In [29]: 4.304117e+09 - 2.591029e+09 #enginesize 1.713088e+09

Out[29]: 1713088000.0

In [30]: reg\_reduced\_nohorse = smf.ols('price~citympg+enginesize', data=carsnew).fit()  
sm.stats.anova\_lm(reg\_reduced\_nohorse, typ=2)

Out[30]:

	sum_sq	df	F	PR(>F)
<b>citympg</b>	3.186064e+08	1.0	23.475016	2.566218e-06
<b>enginesize</b>	3.956410e+09	1.0	291.509467	1.213994e-40
<b>Residual</b>	2.660141e+09	196.0	NaN	NaN

In [31]: 2.660141e+09 - 2.591029e+09 #horsepower ~6.911228e+07

Out[31]: 69112000.0

### This tells us Type 2 is doing partial F tests

- Every reduced model is compared to the same full model:

For example, the F test for `enginesize` is comparing

- Reduced model:  $\text{price} \sim \text{citympg} + \text{horsepower}$
- Full model:  $\text{price} \sim \text{enginesize} + \text{citympg} + \text{horsepower}$

So it will suggest the significance of one predictor given ALL OTHER predictors are already in the model; we can use it to

- judge the "conditional" importance of each predictor
- pick some reduced models with one less predictor that's better than the full model.

### Example: Credit data

```
In [34]: # example: Credit data
credit = pd.read_csv("../data/Credit.csv")
credit.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 400 entries, 0 to 399
Data columns (total 12 columns):
 #   Column      Non-Null Count  Dtype  
 ---  --          --          --      
 0   Unnamed: 0  400 non-null    int64  
 1   Income       400 non-null    float64 
 2   Limit        400 non-null    int64  
 3   Rating       400 non-null    int64  
 4   Cards        400 non-null    int64  
 5   Age          400 non-null    int64  
 6   Education    400 non-null    int64  
 7   Gender       400 non-null    object  
 8   Student      400 non-null    object  
 9   Married      400 non-null    object  
 10  Ethnicity    400 non-null    object  
 11  Balance      400 non-null    int64  
dtypes: float64(1), int64(7), object(4)
memory usage: 37.6+ KB
```

```
In [35]: credit['Income'] = pd.to_numeric(credit['Income'])
```

```
In [36]: model = smf.ols('Balance~Income + Limit + Rating + Age', data=credit).fit()
model.summary()
```

Out [36]:

## OLS Regression Results

<b>Dep. Variable:</b>	Balance	<b>R-squared:</b>	0.877			
<b>Model:</b>	OLS	<b>Adj. R-squared:</b>	0.876			
<b>Method:</b>	Least Squares	<b>F-statistic:</b>	705.6			
<b>Date:</b>	Mon, 16 Sep 2024	<b>Prob (F-statistic):</b>	2.16e-178			
<b>Time:</b>	17:07:18	<b>Log-Likelihood:</b>	-2599.9			
<b>No. Observations:</b>	400	<b>AIC:</b>	5210.			
<b>Df Residuals:</b>	395	<b>BIC:</b>	5230.			
<b>Df Model:</b>	4					
<b>Covariance Type:</b>	nonrobust					
	<b>coef</b>	<b>std err</b>	<b>t</b>	<b>P&gt; t </b>	<b>[0.025</b>	<b>0.975]</b>
<b>Intercept</b>	-445.1048	40.576	-10.970	0.000	-524.877	-365.332
<b>Income</b>	-7.6127	0.382	-19.945	0.000	-8.363	-6.862
<b>Limit</b>	0.0818	0.045	1.834	0.067	-0.006	0.170
<b>Rating</b>	2.7314	0.664	4.111	0.000	1.425	4.038
<b>Age</b>	-0.8561	0.478	-1.789	0.074	-1.797	0.084
<b>Omnibus:</b>	94.733	<b>Durbin-Watson:</b>	1.906			
<b>Prob(Omnibus):</b>	0.000	<b>Jarque-Bera (JB):</b>	165.919			
<b>Skew:</b>	1.374	<b>Prob(JB):</b>	9.36e-37			
<b>Kurtosis:</b>	4.550	<b>Cond. No.</b>	2.65e+04			

Notes:

- [1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
- [2] The condition number is large, 2.65e+04. This might indicate that there are strong multicollinearity or other numerical problems.

From individual t test, Income and Rating are significant, Limit is on the boundary. So t test suggests the model:  $y \sim \text{Income} + \text{Rating}$

In [38]: # try sequential anova with the same order

sm.stats.anova\_lm(model, typ=1)

	<b>df</b>	<b>sum_sq</b>	<b>mean_sq</b>	<b>F</b>	<b>PR(&gt;F)</b>
<b>Income</b>	1.0	1.813117e+07	1.813117e+07	691.691426	7.902872e-89
<b>Limit</b>	1.0	5.533791e+07	5.533791e+07	2111.102870	1.465723e-160
<b>Rating</b>	1.0	4.328357e+05	4.328357e+05	16.512381	5.832806e-05
<b>Age</b>	1.0	8.394134e+04	8.394134e+04	3.202304	7.430058e-02
<b>Residual</b>	395.0	1.035406e+07	2.621280e+04		NaN
					NaN

- p-values disagree with t tests
- suggest  $y \sim \text{Income} + \text{Limit}$  is sig. better than just  $y \sim \text{Income}$
- $y \sim \text{Income} + \text{Limit} + \text{Rating}$  is the best one (adding age is not significant)

```
In [40]: #Put the questionable one (Limit) in the end
model_2 = smf.ols('Balance~Income + Rating +Limit', data=credit).fit()
sm.stats.anova_lm(model_2, typ=1)
```

	<b>df</b>	<b>sum_sq</b>	<b>mean_sq</b>	<b>F</b>	<b>PR(&gt;F)</b>
<b>Income</b>	1.0	1.813117e+07	1.813117e+07	687.865947	1.316473e-88
<b>Rating</b>	1.0	5.567620e+07	5.567620e+07	2112.261356	8.086517e-161
<b>Limit</b>	1.0	9.454487e+04	9.454487e+04	3.586873	5.896627e-02
<b>Residual</b>	396.0	1.043800e+07	2.635858e+04		NaN
					NaN

Now it suggest  $y \sim \text{Income} + \text{Rating}$  might be enough, adding Limit and Age doesn't significantly improve it anymore.

- limit might be on the boundary

**Can you try type2?** What does that tell us?

```
In [43]: model.summary()
```

Out [43]:

## OLS Regression Results

<b>Dep. Variable:</b>	Balance	<b>R-squared:</b>	0.877			
<b>Model:</b>	OLS	<b>Adj. R-squared:</b>	0.876			
<b>Method:</b>	Least Squares	<b>F-statistic:</b>	705.6			
<b>Date:</b>	Mon, 16 Sep 2024	<b>Prob (F-statistic):</b>	2.16e-178			
<b>Time:</b>	17:07:18	<b>Log-Likelihood:</b>	-2599.9			
<b>No. Observations:</b>	400	<b>AIC:</b>	5210.			
<b>Df Residuals:</b>	395	<b>BIC:</b>	5230.			
<b>Df Model:</b>	4					
<b>Covariance Type:</b>	nonrobust					
	<b>coef</b>	<b>std err</b>	<b>t</b>	<b>P&gt; t </b>	<b>[0.025</b>	<b>0.975]</b>
<b>Intercept</b>	-445.1048	40.576	-10.970	0.000	-524.877	-365.332
<b>Income</b>	-7.6127	0.382	-19.945	0.000	-8.363	-6.862
<b>Limit</b>	0.0818	0.045	1.834	0.067	-0.006	0.170
<b>Rating</b>	2.7314	0.664	4.111	0.000	1.425	4.038
<b>Age</b>	-0.8561	0.478	-1.789	0.074	-1.797	0.084
<b>Omnibus:</b>	94.733	<b>Durbin-Watson:</b>	1.906			
<b>Prob(Omnibus):</b>	0.000	<b>Jarque-Bera (JB):</b>	165.919			
<b>Skew:</b>	1.374	<b>Prob(JB):</b>	9.36e-37			
<b>Kurtosis:</b>	4.550	<b>Cond. No.</b>	2.65e+04			

Notes:

- [1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
- [2] The condition number is large, 2.65e+04. This might indicate that there are strong multicollinearity or other numerical problems.

In [44]: `anova = sm.stats.anova_lm(model, typ=2)`  
`anova`

	sum_sq	df	F	PR(>F)
Income	1.042717e+07	1.0	397.789351	9.915136e-62
Limit	8.820013e+04	1.0	3.364774	6.735675e-02
Rating	4.430959e+05	1.0	16.903801	4.785510e-05
Age	8.394134e+04	1.0	3.202304	7.430058e-02
Residual	1.035406e+07	395.0	NaN	NaN

Now p-vals agree ^.^

In [46]: `model.pvalues[1:]`

Out[46]:

Income	9.915136e-62
Limit	6.735675e-02
Rating	4.785510e-05
Age	7.430058e-02
dtype:	float64

In [47]: `anova.values[:,3]`

Out[47]:

```
array([9.91513625e-62, 6.73567507e-02, 4.78551005e-05, 7.43005824e-02,
       nan])
```

For the two candidate models, let's look at R^2 and adj-R^2

In [49]:

```
model_c1 = smf.ols('Balance~Income + Rating', data=credit).fit() #from t test
model_c2 = smf.ols('Balance~Income + Rating + Limit', data=credit).fit()#from f test
#model_c3 = smf.ols('Balance~Income + Rating +Age',data=credit).fit() #not t or f test
```

In [50]:

```
print(model_c1.rsquared, model_c1.rsquared_adj)
print(model_c2.rsquared, model_c2.rsquared_adj)
#print(model_c3.rsquared, model_c3.rsquared_adj)
```

0.8751179476994355 0.8744888189724805  
0.8762389456262863 0.8753013618810309

Out of possible candidates , based on adjusted R^2, model\_c2 is the best choice;

the analysis during the way suggests the possible ranking of importance: Rating ~ Income ~ Rating > Limit > Age

## Remark

You can already see that different methods lead to different conclusions. After we learn more about model diagnostics and several model selection methods, we need to consider all the methods, and choose one that you feel best in terms of interpretability

or prediction, depending on your purpose. If it's still hard to decide, you can try to evaluate prediction error on a test set and choose the one with better performance.

In [ ]:

In [ ]:

