In [ ]:

```
lmres.f_test("C(s0)[T.北京] = 3")
```

多个系数的联合检验

In [ ]:

```
lmres2.f_test("(C(s0)[T.北京] = C(s0)[T.广州]), (Intercept = -0.7)")
```

控制交互项之后系数的检验

In [ ]:

```
lmres2.summary()
```

In [ ]:

```
# # 检验time = 200704时，北京和上海的系数是否相同
lmres2.f_test("(C(s0)[T.北京] = 0)")
```

In [ ]:

```
# 检验time= 200712时，北京和广州的系数是否相同
lmres2.f_test([0,
               1, -1,
               0, 0, 0,
               1, -1,
               0, 0, 0, 0,
               0])
```

2.5 实战练习

请尝试使用标准数据矩阵方式而不是模型表达式方式拟合2.3.2节中拟合的多因素方差分析模型。

提示：对于分类自变量需要先手工转换为哑变量。

请尝试使用自定义检验方式实现如下检验需求：

上海在2007年12月和2009年12月的系数是否相同。
上海在2008年12月的系数和广州在2009年12月的系数是否相同。

3 线性回归模型

3.1 基本原理

3.1.1 线性回归模型概述

3.1.2 线性回归模型的适用条件

3.1.3 线性回归模型的标准建模步骤

3.2 用statsmodels拟合回归模型

class statsmodels.regression.linear_model.OLS(

```
    endog : 因变量，1维数组格式
    exog = None : n*k格式数组，k代表自变量数量
    missing = 'none' : 对缺失值的处理方式
        'none' : 不做任何检查
        'drop' : 发现缺失值后该案例直接删除
        'raise' : 检查并抛出错误
    hasconst = None : T/F，是否允许用户自定义的常数被纳入方程
        模型不默认纳入常数项，该参数必须和exog的内容相匹配！
```

)

3.2.1 拟合单自变量模型

注意：建模之前必须将数据中的缺失值处理完毕，建模用数据不能包括缺失值！

In [ ]:

```
boston = datasets.load_boston()

dfboston = pd.DataFrame(boston.data, columns = boston.feature_names)

# 数据关联趋势的的图形化观察
plt.scatter(dfboston.NOX, boston.target)
```

In [ ]:

```
# 在数据集中加入常数项
dfboston['cons'] = 1
dfboston.head()
```

In [ ]:

```
# 也可以使用statsmodels.tools.tools.add_constant()命令增加常数项
import statsmodels.api as sm # 使用api命令简化后续调用

# 数据中未发现常数项时会新增一列常数项
temp = sm.add_constant(dfboston.iloc[:,[0,1]])
temp.head()
```

In [ ]:

```
from statsmodels.regression.linear_model import OLS

regres = OLS(boston.target, dfboston[['cons', 'NOX']]).fit()
```

In [ ]:

```
regres.summary()
```

In [ ]:

```
# 错误拟合的无常数项模型
OLS(boston.target, dfboston.NOX).fit().summary()
```

绘制回归线

statsmodels.graphics.regressionplots.abline_plot(

```
    intercept = None, slope = None : 回归线的截距和斜率
    horiz = None, vert = None : 加绘的水平线/垂直线位置
    model_results = None : 具有(intercept, slope)参数输出的statsmodels结果
    ax = None, **kwargs
```

)

In [ ]:

```
from statsmodels.graphics.api import abline_plot

fig = abline_plot(41.3459, -33.9161)
plt.scatter(dfboston.NOX, boston.target)
```

In [ ]:

```
fig = abline_plot(model_results = regres)
plt.scatter(dfboston.NOX, boston.target)
```

3.2.2 拟合多自变量模型

In [ ]:

```
from statsmodels.regression.linear_model import OLS

# 非数值变量必须要先转换为数值变量
regres = OLS(boston.target, dfboston, missing = 'drop').fit()
```

In [ ]:

```
regres.summary()
```

3.2.3 从模型中取出具体指标

模型效果指标

```
    rsquared - 决定系数
    rsquared_adj - 校正后的决定系数
    aic
    bic
```

参数估计/检验

nobs - 样本量
fvalue - 模型总体方差分析的F值
f_pvalue - 模型总体方差分析的P值
params - 原始回归系数值
bse - 回归系数的se
tvalues - 回归系数的双侧t检验t值
pvalues - 回归系数的双侧t检验P值

方差/自由度分解

df_model - 模型自由度
df_resid - 残差自由度，有常数项时为n - p - 1，无时为n - p

mse_total - 总的MS
mse_model - 模型的MS
mse_resid - MSE

centered_tss - 总SS
ess - 可被模型解释的SS
ssr - 残差的SS

残差/预测值

fittedvalues - 样本预测值
resid - 原始残差
resid_pearson - Pearson残差（标化残差）

In [ ]:

```
regres.params
```

3.2.4 残差分析

考察残差是否服从正态分布：

statsmodels.stats.stattools.jarque_bera(resids, axis = 0)

当提供的数据为二维数组时，才需要指定axis

检验返回值:

JB (float or array) - The Jarque-Bera test statistic
JBpv (float or array) - The pvalue of the test statistic
skew (float or array) - Estimated skewness of the data
kurtosis (float or array) - Estimated kurtosis of the data

In [ ]:

```
# 取出残差、估计值等备用
dfres = pd.DataFrame({'fit' : regres.fittedvalues,
                      'resid' : regres.resid,
                      'zresid' : regres.resid_pearson})
dfres.head()
```

In [ ]:

```
# 残差正态性检验
from statsmodels.stats import stattools

stattools.jarque_bera(dfres.resid)
```

In [ ]:

```
dfres.resid.plot.hist()
```

In [ ]:

```
dfres.zresid.plot.hist()
```

In [ ]:

```
dfres.plot.scatter('fit', 'zresid')
```

3.3 自变量的筛选

3.3.1 逐步回归

python中的各个包实际上并没有直接实现逐步回归功能，需要用户根据基本原理自行代码实现。

给出单变量筛选的P值

In [ ]:

```
from sklearn.feature_selection import f_regression as freg

freg(dfboston.iloc[:,[0,1,2,3]], boston.target)
```

按照P值直接进行变量筛选

In [ ]:

```
# 自变量和因变量为参数分别指定
# 由于是单变量依次筛选，因此需要先处理掉缺失值，否则结果不准确
import pandas as pd
import numpy as np
import statsmodels.api as sm

def stepwise_selection(X, y,
                       initial_list = [],
                       threshold_in = 0.01,
                       threshold_out = 0.05,
                       missing = 'drop',
                       verbose = True):
    """ Perform a forward-backward feature selection
    based on p-value from statsmodels.api.OLS
    Arguments:
        X - pandas.DataFrame with candidate features
        y - list-like with the target
        initial_list - list of features to start with (column names of X)
        threshold_in - include a feature if its p-value < threshold_in
        threshold_out - exclude a feature if its p-value > threshold_out
        verbose - whether to print the sequence of inclusions and exclusions
    Returns: list of selected features
    Always set threshold_in < threshold_out to avoid infinite looping.
    See https://en.wikipedia.org/wiki/Stepwise_regression for the details
    """
    included = list(initial_list)
    while True:
        changed=False
        # forward step
        excluded = list(set(X.columns)-set(included))
        new_pval = pd.Series(index=excluded)
        for new_column in excluded:
            model = sm.OLS(y, sm.add_constant(
                                pd.DataFrame(X[included+[new_column]])),
                           missing).fit()
            new_pval[new_column] = model.pvalues[new_column]
        best_pval = new_pval.min()
        if best_pval < threshold_in:
            best_feature = new_pval.idxmin()
            included.append(best_feature)
            changed=True
            if verbose:
                print('Add  {:30} with p-value {:.6}'.format(best_feature, best_pva

        # backward step
        model = sm.OLS(y, sm.add_constant(pd.DataFrame(X[included]))).fit()
        # use all coefs except intercept
        pvalues = model.pvalues.iloc[1:]
        worst_pval = pvalues.max() # null if pvalues is empty
        if worst_pval > threshold_out:
            changed=True
            worst_feature = pvalues.argmax()
            included.remove(worst_feature)
            if verbose:
                print('Drop {:30} with p-value {:.6}'.format(worst_feature, worst_p
        if not changed:
            break
    return included
```

In [ ]:

```
# 函数中直接加入了常数项，因此原始数据中可以不需要再加入常数项
result = stepwise_selection(dfboston, boston.target)
```

In [ ]:

```
dfboston.head()
```

In [ ]:

```
result
```

按照校正决定系数来筛选最优方程

In [ ]:

```
# 自变量和因变量位于同一数据阵中进行指定
import statsmodels.formula.api as smf

def forward_selected(data, response):
    """Linear model designed by forward selection.

    Parameters:
    -----------
    data : pandas DataFrame with all possible predictors and response

    response: string, name of response column in data

    Returns:
    --------
    model: an "optimal" fitted statsmodels linear model
           with an intercept
           selected by forward selection
           evaluated by adjusted R-squared
    """
    remaining = set(data.columns)
    remaining.remove(response)
    selected = []
    current_score, best_new_score = 0.0, 0.0
    while remaining and current_score == best_new_score:
        scores_with_candidates = []
        for candidate in remaining:
            formula = "{} ~ {} + 1".format(response,
                                           ' + '.join(selected + [candidate]))
            score = smf.ols(formula, data).fit().rsquared_adj
            scores_with_candidates.append((score, candidate))
        scores_with_candidates.sort()
        best_new_score, best_candidate = scores_with_candidates.pop()
        if current_score < best_new_score:
            remaining.remove(best_candidate)
            selected.append(best_candidate)
            current_score = best_new_score
    formula = "{} ~ {} + 1".format(response,
                                   ' + '.join(selected))
    model = smf.ols(formula, data).fit()
    return model
```

In [ ]:

```
df2 = dfboston.copy()
df2['y'] = boston.target
del df2['cons']
```

In [ ]:

```
forward_selected(df2, 'y').summary()
```

3.3.2 最小角回归

class sklearn.linear_model.Lars(

fit_intercept = True, verbose = False, normalize = True
precompute = 'auto'
n_nonzero_coefs = 500 : 纳入模型的最大自变量数，np.inf代表无限制
eps = 2.2204460492503131e-16 : Cholesky diagonal factors的计算精度
copy_X = True
fit_path = True : 是否将系数路径存储在coef_path_属性中
超大数据集可关闭该选项以加速分析
positive = False : 是否限制系数必须非负

)

Lars类的属性：

alphas_ : array, 形如(n_alphas+1,)，非零系数在迭代中的最大协方差绝对值
active_ : list, length = n_alphas，变量被纳入模型的先后顺序（索引值）
coef_path_ : array, shape (n_features, n_alphas + 1)
各变量在迭代中的系数改变情况，该结果可被用于模型调优
coef_ : array, 形如(n_features,) or (n_targets, n_features)，系数值
intercept_ : float | array, shape (n_targets,)
n_iter_ : array-like or int，模型迭代次数

In [ ]:

```
lars = linear_model.Lars(n_nonzero_coefs = 10)
lars.fit(boston.data, boston.target)
```

In [ ]:

```
lars.coef_
```

In [ ]:

```
lars.active_
```

In [ ]:

```
lars.score(boston.data, boston.target)
```

3.4 线性回归模型的sklean实现

class sklearn.linear_model.LinearRegression(

fit_intercept = True ： 模型是否包括常数项
使用该选项就不需要在数据框中设定cons
normalize = False ： 是否对数据做正则化，具体为(x - mean)/L2-norm
copy_X = True ： 是否复制x矩阵
n_jobs = 1 ： 使用的例程数，为-1时使用全部CPU，大样本多因变量时有加速效果

)

注意：

函数中的normalize参数并非进行标准正态变换。
sklearn.preprocessing.StandardScaler可用于满足标准正态变换的需求。
数据中不能存在缺失值，否则报错。

LinearRegression类的属性：

coef_ ： array，多因变量时为二维数组
intercept_ ： 常数项

LinearRegression类的方法：

fit(X, y[, sample_weight]) ： 拟合模型
get_params([deep]) ： 获取模型的具体参数设定
predict(X) ： 返回具体预测值
score(X, y[, sample_weight]) ： 返回模型决定系数
set_params(**params) ： 重新设定模型参数

注意：方法中没有返回系数检验结果（P值）的功能

In [ ]:

```
from sklearn import linear_model

reg = linear_model.LinearRegression()

reg.fit(boston.data, boston.target)
```

In [ ]:

```
print(reg.coef_, reg.intercept_)
```

In [ ]:

```
# 返回模型拟合后的决定系数
reg.score(boston.data, boston.target)
```

In [ ]:

```
# 利用该模型进行预测
reg.predict(boston.data[:10])
```

3.5 实战练习

尝试使用模型表达式方式拟合本章中的回归模型。

仿照本章的案例，分别考察性别、年龄、家庭收入等变量对现状指数、预期指数的影响，并对相应的模型进行化简，剔除无统计意义的变量，并完成模型的残差诊断。

比较方差分析模型和回归模型的实现方式和输出结果。

4 线性回归的衍生模型

线性回归模型在应用时的难点主要有两个：

如何进行多变量筛选。
数据违反模型适用条件时如何处理。

4.1 非线性趋势

线性趋势的假设检验

建议使用图形来考察，如果一定要进行检验，可以考虑使用Harvey Collier test for linearity

```
statsmodels.stats.diagnostic.linear_harvey_collier(res)
```

In [ ]:

```
from statsmodels import stats as ss

regres = OLS(boston.target, dfboston[['cons', 'NOX']]).fit()
ss.diagnostic.linear_harvey_collier(regres)
```

4.1.1 曲线直线化

In [ ]:

```
df = pd.read_excel('dmdata.xlsx', sheet_name = 'curve')
df.head()
```

In [ ]:

```
df['lny'] = np.log(df.y)
df = sm.add_constant(df)
df.head()
```

In [ ]:

```
regres = OLS(df.lny, df.iloc[:,[0,1]]).fit()
regres.summary()
```

In [ ]:

```
np.exp(regres.params[0])
```