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])
```

In [ ]:

```
sns.lineplot(df.time, np.exp(regres.predict()))
plt.scatter(df.time, df.y)
```

### 4.1.2 多项式回归

#### *使用表达式方式进行模型定义*

In [ ]:

```
boston = datasets.load_boston()

dfboston = pd.DataFrame(boston.data, columns = boston.feature_names)
df2 = dfboston.copy()
df2['y'] = boston.target
df2.head()
```

In [ ]:

```
from statsmodels.formula.api import ols

# 使用公式方式定义模型
lmres = ols('y ~ CRIM + ZN + INDUS + CHAS + NOX + RM + NOX * RM',
              df2).fit()
lmres.summary()
```

In [ ]:

```
from statsmodels.formula.api import ols

# 使用公式方式定义模型
lmres = ols('y ~ CRIM + ZN + INDUS + CHAS + NOX + RM + NOX*RM + RM * RM',
              df2).fit()
lmres.summary()
```

In [ ]:

```
df2['RM2'] = df2.RM * df2.RM
lmres = ols('y ~ CRIM + ZN + INDUS + CHAS + NOX + RM + NOX * RM + RM2',
              df2).fit()
lmres.summary()
```

#### *使用sklarn的PolynomialFeatures自动生成高次项*

class sklearn.preprocessing.PolynomialFeatures(

    degree = 2 : integer，模型所考虑的高次项次方数
    interaction_only = False : boolean，是否只生成交互项，忽略变量的高次方项
    include_bias = True : boolean，模型中是否纳入常数项

)

PolynomialFeatures类的属性:

```
    powers_ : array, shape (n_output_features, n_input_features)
        powers_[i, j]代表第j个输入属性在第i个输出属性中的次方数
    n_input_features_ : int, 输入特征的总数
    n_output_features_ : int, 输出特征的总数
```

In [ ]:

```
from sklearn.preprocessing import PolynomialFeatures

poly = PolynomialFeatures(2)

IX = poly.fit_transform(boston.data)
print(len(IX[1]))
IX[:1]
```

In [ ]:

```
regres = OLS(boston.target, IX).fit()
regres.summary()
```

# 4.2 强影响点

## 4.2.1 强影响点的识别

RLM方法也可以直接用于强影响点的识别。

***计算各案例的影响力统计量***

class statsmodels.stats.outliers_influence.OLSInfluence(results)

```
    dfbetas() : 标化dfbetas
    dffits() : 标化dffits measure for influence of an observation
    cov_ratio() : covariance ratio between LOOO and original

    cooks_distance() : Cooks distance

    summary_frame() : 创建包括各种影响力统计量的DF
    summary_table([float_fmt]) : 创建包括各种影响力统计量的汇总表
```

In [ ]:

```
from statsmodels.api import OLS

regres = OLS(boston.target, sm.add_constant(boston.data)).fit()
```

In [ ]:

```
from statsmodels.stats.outliers_influence import OLSInfluence

Infstat = OLSInfluence(regres)
```

In [ ]:

```
Infstat.
```

In [ ]:

```
Infstat.cov_ratio[:10]
```

In [ ]:

```
Infstat.summary_table()
```

In [ ]:

```
tmpdf = Infstat.summary_frame()
tmpdf.head()
```

In [ ]:

```
# 另一种影响力统计量计算方式
infl = regres.get_influence()
infl.summary_frame()
```

***绘制各案例的影响力图形***

statsmodels.graphics.regressionplots.influence_plot(

```
    results : 拟合完毕的模型名称
    external = True, alpha = 0.05
    criterion = 'cooks' : {'DFFITS', 'Cooks'}, 离群值影响力展示指标
    size = 48 : 图中散点的影响力辐射半径
    plot_alpha = 0.75, ax = None, **kwargs
```

)

In [ ]:

```
from statsmodels.graphics.regressionplots import influence_plot

fig = influence_plot(regres)
```

## 4.2.2 稳健回归

class statsmodels.robust.robust_linear_model.RLM(

```
    endog, exog
    M = <statsmodels.robust.norms.HuberT object> : 用于减少离群值影响的指标
        LeastSquares, HuberT, RamsayE, AndrewWave
        TrimmedMean, Hampel, and TukeyBiweight
    missing = 'none' : 'none', 'drop', and 'raise'
```

)

In [ ]:

```
dfrlm = pd.read_excel('dmdata.xlsx', sheet_name = 'rlm')
dfrlm.head()
```

In [ ]:

```
plt.scatter(dfrlm.x1, dfrlm.y)
plt.scatter(dfrlm.x2, dfrlm.y)
```

In [ ]:

```
from statsmodels.formula.api import ols

lmres = ols('y ~ x1 + x2', dfrlm).fit()
lmres.summary()
```

In [ ]:

```
from statsmodels.formula.api import rlm

rlmres = rlm('y ~ x1 + x2', dfrlm).fit()
rlmres.summary()
```

In [ ]:

```
rlm('y ~ x1 + x2', dfrlm, M = sm.robust.norms.LeastSquares()).fit().summary()
```

# 4.3 多重共线性

## 4.3.1 共线性的识别

***共线性的表现***

案例：现测得22例胎儿的身长、头围、体重和胎儿受精周龄，具体数据见文件dmdata.xlsx的ridge表单，研究者希望能建立由前三个外形指标推测胎儿周龄的回归方程。

In [ ]:

```
dfridge = pd.read_excel('dmdata.xlsx', sheet_name = 'ridge')
dfridge = sm.add_constant(dfridge)
dfridge.head()
```

In [ ]:

```
regres = OLS(dfridge.y, dfridge.iloc[:,[0,1,2,3]]).fit()
regres.summary()
```

In [ ]:

```
plt.scatter(dfridge.touwei, dfridge.y)
```

In [ ]:

```
sns.pairplot(dfridge.iloc[:,[1,2,3]])
```

***共线性的检测***

statsmodels.stats.outliers_influence.variance_inflation_factor(

    exog : 自变量矩阵，注意必须为矩阵形式，不能使用数据框
    exog_idx : 希望计算VIF的自变量列索引值

)返回：VIF值

In [ ]:

```
from statsmodels.stats import outliers_influence

outliers_influence.variance_inflation_factor(
    dfridge.iloc[:,[0,1,2,3]].values, 1)
```

In [ ]:

```
# 直接使用模型内存储的数据矩阵
outliers_influence.variance_inflation_factor(regres.model.exog, 1)
```

In [ ]:

```
# 依次列出所有的VIF值，但跳开cons的VIF输出
[outliers_influence.variance_inflation_factor(regres.model.exog, i)
     for i in range(1, dfridge.shape[1] - 1)]
```

## 4.3.2 岭回归

***拟合岭回归模型***

class sklearn.linear_model.Ridge(

    alpha = 1.0 : 模型惩罚项的系数，正数，越大惩罚力度越强
    fit_intercept = True, normalize = False, copy_X = True

    max_iter = None : 容许的最大迭代次数
    tol = 0.001 : 收敛标准
    solver='auto' : 收敛方法
        {'auto', 'svd', 'cholesky', 'lsqr', 'sparse_cg', 'sag','saga'}
    random_state = None : 伪随机种子的数值

)

注意：岭回归也可以用于分类模型，对应的方法为sklearn.linear_model.RidgeClassifier

In [ ]:

```
dfridge = pd.read_excel("dmdata.xlsx", sheet_name = "ridge")
dfridge.head()
```

In [ ]:

```
from sklearn import linear_model

ridge = linear_model.Ridge(alpha = 0)
ridge.fit(dfridge.iloc[:,[0,1,2]], dfridge.y)
ridge.coef_
```

In [ ]:

```
ridge.score(dfridge.iloc[:,[0,1,2]], dfridge.y)
```

***使用线图协助选取最佳alpha值***

In [ ]:

```
# 对自变量做标准化，以便于图形观察
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
ridgeZX = scaler.fit_transform(dfridge.iloc[:, 0:3])
```

In [ ]:

```
# 设定用于筛选的一系列alpha值
n_alphas = 200
alphas = np.logspace(-10, 1, n_alphas)
alphas[:10]
```

In [ ]:

```
# 存储所有的模型结果
coefs = []
rsqs = []
for a in alphas:
    ridge = linear_model.Ridge(alpha = a)
    ridge.fit(ridgeZX, dfridge.y)
    coefs.append(ridge.coef_)
    rsqs.append(ridge.score(ridgeZX, dfridge.y))
coefs[:10]
```

In [ ]:

```
plt.plot(alphas, coefs)
```

In [ ]:

```
plt.plot(alphas, rsqs)
```

In [ ]:

```
# 放大关键区间进行观察
plt.plot(alphas, coefs)
plt.gca().set_xlim(0, 1.5)
ax2 = plt.gca().twinx()
ax2.plot(alphas, rsqs, 'r--', linewidth = 3)
```

In [ ]:

```
# 获取指定模型的结果
ridge = linear_model.Ridge(alpha = 0.8)
ridge.fit(ridgeZX, dfridge.y)
print(ridge.intercept_, ridge.coef_)
```

### 4.3.3 LASSO回归

class sklearn.linear_model.Lasso(

```
    alpha = 1.0, fit_intercept = True,
    normalize = False, copy_X = True

    precompute = False : 是否使用预计算的Gram矩阵加速拟合
    max_iter = 1000, tol = 0.0001
    warm_start = False : 是否使用上一次的模型拟合结果作为本次初始值
    positive = False : 系数值是否必须非负
    random_state = None
    selection = 'cyclic' : 随机更新系数还是按顺序更新，设为'random'可加速拟合
```

)

In [ ]:

```
# 存储所有的模型结果
coefs = []
rsqs = []
for a in alphas:
    lasso = linear_model.Lasso(alpha = a, max_iter = 10000)
    lasso.fit(ridgeZX, dfridge.y)
    coefs.append(lasso.coef_)
    rsqs.append(lasso.score(ridgeZX, dfridge.y))
coefs[:10]
```

In [ ]:

```
# 放大关键区间进行观察
plt.plot(alphas, coefs)
# plt.gca().set_xlim(0, 0.5)
ax2 = plt.gca().twinx()
ax2.plot(alphas, rsqs, 'r--', linewidth = 3)
```

In [ ]:

```
# 获取指定模型的结果
lasso = linear_model.Lasso(alpha = 1)
lasso.fit(ridgeZX, dfridge.y)
print(lasso.intercept_, lasso.coef_)
```

### 4.3.4 弹性网络

class sklearn.linear_model.ElasticNet(

```
    alpha = 1.0
    l1_ratio = 0.5 : L1和L2模型的混合比例
        l1_ratio = 0, 拟合Lasso回归
        l1_ratio = 1, 拟合岭回归
        0 < l1_ratio < 1, 两种模型的混合

    fit_intercept = True, normalize = False, precompute = False
    max_iter = 1000, copy_X = True, tol = 0.0001, warm_start = False
    positive = False, random_state = None, selection = 'cyclic'
```

)

In [ ]:

```
ratios = np.linspace(0, 1, 50)

coefs = []
rsqs = []
for r in ratios:
    enet = linear_model.ElasticNet(alpha = 0.8, l1_ratio = r)
    enet.fit(ridgeZX, dfridge.y)
    coefs.append(enet.coef_)
    rsqs.append(enet.score(ridgeZX, dfridge.y))
coefs[:10]
```

In [ ]:

```
# 放大关键区间进行观察
plt.plot(ratios, coefs)
# plt.gca().set_xlim(0, 0.5)
ax2 = plt.gca().twinx()
ax2.plot(ratios, rsqs, 'r--', linewidth = 3)
```

## 4.4 方差不齐

### 4.4.1 方差不齐的识别

首选的方式应当是对残差图进行观察。

Lagrange Multiplier Heteroscedasticity Test by Breusch-Pagan

```
statsmodels.stats.diagnostic.het_breuschpagan(

    resid : array-like, 回归模型的残差
    exog_het : 可能导致方差不齐的自变量矩阵
)返回:

    lm (float) - lagrange multiplier statistic
    lm_pvalue (float) - p-value of lagrange multiplier test
    fvalue (float) - 误差方差不依赖于自变量矩阵的f-statistic
    f_pvalue (float) - p-value for the f-statistic
```

Lagrange Multiplier Heteroscedasticity Test by White

```
statsmodels.stats.diagnostic.het_white(resid, exog, retres=False)
```

het_goldfeldquandt:

```
test whether variance is the same in 2 subsamples
```

注意：上述命令均假设回归模型中含有常数项

In [ ]:

```
regres = OLS(boston.target, dfboston).fit()
regres.summary()
```

In [ ]:

```
# 考察残差方差是否和因变量存在线性关系
sm.stats.diagnostic.het_breuschpagan(regres.resid,
                                     sm.add_constant(boston.target))
```

In [ ]:

```
# 考察残差方差是否和自变量NOX间存在线性关系
sm.stats.diagnostic.het_breuschpagan(regres.resid,
                                     sm.add_constant(dfboston.NOX))
```

In [ ]:

```
sm.stats.diagnostic.het_breuschpagan(regres.resid,
                                     sm.add_constant(dfboston.RM))
```

In [ ]:

```
sm.stats.diagnostic.het_white(regres.resid, sm.add_constant(boston.target))
```

### 4.4.2 加权最小二乘法

class statsmodels.regression.linear_model.WLS(

```
endog, exog
weights = 1.0 : (array-like, optional), 为每个案例提供的权重，1维数组
missing = 'none', hasconst = None, **kwargs
```

)

In [ ]:

```
dfwls = pd.read_excel('dmdata.xlsx', sheet_name = 'wls')
dfwls.head()
```

In [ ]:

```
regres = sm.OLS(dfwls.y, sm.add_constant(dfwls.x)).fit()
regres.summary()
```

In [ ]:

```
wlsres = sm.WLS(dfwls.y, sm.add_constant(dfwls.x), weights = dfwls.n).fit()

wlsres.summary()
```

In [ ]:

```
wlsres = sm.WLS(dfwls.y, sm.add_constant(dfwls.x), weights = 1).fit()

wlsres.summary()
```

# 4.5 残差非独立

## 4.5.1 残差非独立的识别

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

Durbin-Watson残差自相关检验，也可直接用summary()输出结果

statsmodels.stats.diagnostic.acorr_ljungbox(x, lags = None, boxpierce = False)

Ljung-Box残差无自相关检验，可同时输出Box-Pierce统计量

statsmodels.stats.diagnostic.acorr_breusch_godfrey(results, nlags = None, store = False)

Breusch-Pagan残差自相关检验

In [ ]:

```
dfgdp = pd.read_excel('dmdata.xlsx', sheet_name = 'gdp')
dfgdp.head()
```

In [ ]:

```
lmres = ols('gdp ~ ind1 + ind2 + ind3', dfgdp).fit()
lmres.summary()
```

In [ ]:

```
sm.stats.stattools.durbin_watson(lmres.resid)
```

In [ ]:

```
sm.stats.diagnostic.acorr_ljungbox(lmres.resid)
```

In [ ]:

```
sm.stats.diagnostic.acorr_ljungbox(lmres.resid, lags = [1,2,3,4])
```

### 4.5.2 自回归

class statsmodels.tsa.arima_model.ARIMA(

endog : 时间序列变量
order = (0, 0, 0) : (p,d,q)三个参数分别代表自回归、差分、移动平均
exog : optional, 自变量列表
dates : , optional, 日期时间变量
freq : 时间序列数据的间隔频率

)

In [ ]:

```
from statsmodels.tsa.arima_model import ARIMA

model = ARIMA(lmres.resid, order=(2, 0, 0))
results_AR = model.fit()
```

In [ ]:

```
results_AR.summary()
```

In [ ]:

```
res2 = ARIMA(lmres.resid, order=(1, 0, 0)).fit()
res2.summary()
```

In [ ]:

```
sm.stats.diagnostic.acorr_ljungbox(res2.resid)
```

In [ ]:

```
ARIMA(dfgdp.gdp, (1, 0, 0), dfgdp.iloc[:, [1,2,3]]).fit().summary()
```

## 4.6 实战练习

使用本章中学到的知识，尝试对boston数据所建立的回归方程进行全面的回归诊断。

尝试对boston数据拟合稳健回归模型、岭回归模型，比较分析结果和普通回归模型的差异。

# 5 logitsic回归