Homework 1

DAS 511, Spring 2025, 제출자: 박철준

1. Goal: 기초 Regression 분석 파이썬 실습, 모델 비교 및 선택
2. How: 질문이 있는 곳마다 빈칸 채우기.
3. Submit: 블랙보드에 ipynb 파일 모두 시행 후 pdf 파일 업로드. (PDF Compile이 잘 안되는 경우 HTML Export -> 웹브라우저에서 Open -> Print -> PDF로 저장)

In [1]:
```
# Load Packages & Moduels
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import Lasso, Ridge
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import KFold
import matplotlib.pyplot as plt
import statsmodels.api as sm
```

Q1. 전체 회귀모형의 유의성 없이 개별 계수가 유의해 보일 수 있을까?

과제 설명

100개의 독립변수를 갖는 선형 회귀 모형을 적합해보자. 이때 종속변수 Y는 X와 아무런 상관이 없도록 무작위로 생성된다. 즉, **진짜로는 모든 계수가 0**이다.

그럼에도 불구하고 우리는 통계적으로 유의해 보이는 결과를 얻을 수 있을까?

아래 지침을 따라 실험하고, 결과를 분석하시오.

지침

Test-Level: $\alpha = 0.05$

1. X ~ N(0,1), shape = (500, 100) **서로 독립**
2. Y ~ N(0,1), shape = (100,) **X와 독립**
3. statsmodels OLS로 선형회귀 적합
4. 다음을 출력하시오:
    - F-test의 p-value
    - 각 계수의 t-test p-value들 중 0.05보다 작은 것의 개수

5. 위 실험을 반복해서 1000번 반복해서 얼마나 자주 F-Test Reject이 일어나는지, t-test는 얼마나 False positive가 나오는지 확인
6. 이 실험에서 얻은 결과에 대해 결론을 작성하시오.

목표

- F-test를 통과하지 않는데도 일부 계수의 p-value가 유의해 보일 수 있음을 체험
- 전체 모형 유의성이 먼저 검토되어야 함을 인식

아래 코드들의 `pass` 부분을 완성하시오.

In [9]:
```
# 테스트 1
np.random.seed(0)

# 데이터 생성
n, p = 500, 100
X = np.random.randn(n, p)
y = np.random.randn(n)


# statsmodels 회귀
x = sm.add_constant(X)  # 상수항 추가
model = sm.OLS(y, x).fit()

# F-test p-value
f = model.fvalue
f_pvalue = model.f_pvalue
print(f"F-test p-value: {f_pvalue}")
if f_pvalue < 0.05:
    print("F-test significant at p < 0.05")
else:
    print("F-test not significant at p < 0.05")
print("=" * 50)

# 각 계수의 t-test p-values 가 0.05 보다 작은 경우 몇개가 나오는지 출력
pvals = model.pvalues[1:]  # 상수항 제외
significant = pvals[pvals < 0.05]
print(f"Number of t-tests with p < 0.05: {len(significant)}")
```

```
F-test p-value: 0.7388345502890528
F-test not significant at p < 0.05
==================================================
Number of t-tests with p < 0.05: 3
```

In [ ]:
```
# 테스트 2
import numpy as np
import statsmodels.api as sm
from tqdm import tqdm

# 설정
n, p = 500, 100
n_trials = 1000
alpha = 0.05

f_rejections = 0
t_rejection_counts = []
```

```
np.random.seed(42)

for _ in tqdm(range(n_trials)):
    # 여기에 답안 작성:
    # 위 결과 반복, F-test reject 할 때 마다 f_rejections 하나 추가
    # t_rejection_counts 리스트에 매 트라이얼마다 선택되는 변수의 수 추가
    X = np.random.randn(n, p)
    y = np.random.randn(n)
    x = sm.add_constant(X)  # 상수항 추가
    model = sm.OLS(y, x).fit()
    # F-test p-value 계산
    f_pvalue = model.f_pvalue
    if f_pvalue < alpha:
        f_rejections += 1

    # t-test p-values 계산
    t_pvalues = model.pvalues[1:]  # 상수항 제외
    # t-test에서 유의한 계수 수 계산
    t_pvalues = t_pvalues[t_pvalues < alpha]
    # 유의한 계수 수를 t_rejection_counts에 추가
    t_rejections = len(t_pvalues)
    # t_rejection_counts 리스트에 추가
    t_rejection_counts.append(t_rejections)

# 결과 요약
f_rate = f_rejections / n_trials
t_avg = np.mean(t_rejection_counts)

print(f"\n F-test가 유의했던 비율: {f_rate:.4f}")
print(f"t-test에서 평균적으로 유의했던 계수 수: {t_avg:.2f}")
```

```
100%|██████████| 1000/1000 [00:06<00:00, 150.34it/s]
 F-test가 유의했던 비율: 0.0480
t-test에서 평균적으로 유의했던 계수 수: 4.96
```

In [11]:
```
# 결론
## F-test는 모델 전체의 유의성을 평가하는 반면, t-test는 개별 계수의 유의성을 평가합니다.
## F-test가 유의하지 않더라도 t-test에서 유의한 계수가 존재할 수 있습니다. 이는 모델이 전체
## 따라서, F-test와 t-test는 서로 보완적인 역할을 합니다.
```

Q1 Conclusion

1. F-test가 유의하지 않더라도 t-test에서 유의한 계수가 존재할 수 있습니다. 이는 모델이 전체적으로 유의하지 않지만, 개별 변수는 그럴 수 있음을 나타냅니다.

2. 따라서, t-test만 진행했을 때에 전체 모델이 유의하지 않음에도 유의하다고 오판할 수 있기 때문에 F-test를 선행적으로 수행하여 전체의 유의성을 먼저 판단할 필요가 있습니다.

데이터 분석

In [12]:
```
# Load the Data
data = pd.read_csv('https://www4.stat.ncsu.edu/~boos/var.select/diabetes.
```

In [13]:
```
# Set the predictor and response
X = data.drop(columns=['Y'])
```

```
X = pd.get_dummies(X, columns=['SEX'], drop_first=True) # drop_first=True
y = data['Y']
```

Q2. `train_test_split` 함수를 사용해서 train_idx 와 test_idx 를 만드시오.

유의사항

- 데이터를 분리하는 게 아닌 index 셋 만들기. 활용예: `X_train = X.iloc[train_idx]`
- `test_size` : 0.3
- `random_state` : 0

In [36]:
```
# 여기에 Q2 답안 작성
train_idx, test_idx = train_test_split(np.arange(len(X)), test_size=0.3,
```

In [37]:
```
# train/test split
X_train, X_test, y_train, y_test = X.iloc[train_idx], X.iloc[test_idx], y
```

Q3. `Pipeline`을 이용해서 stanrdardize 후 Linear Regression 을 한 뒤 Train RMSE, Test RMSE report

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

In [20]:
```
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.metrics import mean_squared_error

LinReg = Pipeline(steps=[
    ('Standardize', StandardScaler()),
    ('LinReg', LinearRegression())
])

# 아래에 Q3 답안 작성
LinReg.fit(X_train, y_train)

LR_train_RMSE = np.sqrt(mean_squared_error(y_train, LinReg.predict(X_trai
LR_test_RMSE = np.sqrt(mean_squared_error(y_test, LinReg.predict(X_test))
```

In [21]:
```
results = pd.DataFrame()
results['Linear'] = [LR_train_RMSE, LR_test_RMSE]
results.index = ['Train RMSE', 'Test RMSE']
results
```

Out[21]:

	Linear
Train RMSE	52.954165
Test RMSE	55.651767

Q4. 5-Fold CV 를 이용해 Polynomial Regression 의 적절한 degree를 찾으시오

유의사항

- `X_train` 과 `y_train` 만을 이용해서 degree 를 찾아야 함
- `KFold` 함수 활용, Set `random_state` to be 0.
- `PolynomialFeatures` 적용 전에 Standardize 하기
- 찾아볼 Degree는 1부터 10까지
- `kfold_cv` 라는 list에 각 디그리별 CV 값 저장하기.

In [38]:
```
# 여기에 Q4 답안 작성
from sklearn.model_selection import KFold
kfold = KFold(n_splits=5, shuffle=True, random_state=0)
kfold_cv = []

for degree in range(1, 11):
    poly = PolynomialFeatures(degree=degree)
    X_poly = poly.fit_transform(X_train)

    # Standardize
    scaler = StandardScaler()
    X_poly_scaled = scaler.fit_transform(X_poly)

    # Cross-validation
    cv_scores = []
    for train_cv_idx, val_cv_idx in kfold.split(X_poly_scaled):
        X_train_cv, X_val_cv = X_poly_scaled[train_cv_idx], X_poly_scaled
        y_train_cv, y_val_cv = y_train.iloc[train_cv_idx], y_train.iloc[v

        model = LinearRegression()
        model.fit(X_train_cv, y_train_cv)
        y_pred_cv = model.predict(X_val_cv)

        rmse = np.sqrt(mean_squared_error(y_val_cv, y_pred_cv))
        cv_scores.append(rmse)

    kfold_cv.append(np.mean(cv_scores))
```

In [39]:
```
print(f"The best degree of the polynomial is {np.argmin(kfold_cv)+1}")
best_degree = np.argmin(kfold_cv)+1
np.sqrt(kfold_cv)
```

```
The best degree of the polynomial is 1
```

Out[39]:
```
array([ 7.44076806,  8.16777083, 36.29726875, 18.99901418, 18.58227245,
       18.65421381, 18.97633457, 19.47203151, 20.1110887 , 20.87883147])
```

In [40]:
```
d = best_degree
poly_d = Pipeline(steps=[
            ('Standardize', StandardScaler()),
            ('poly_d-feature', PolynomialFeatures(degree=d, include_bias=
            ('LinReg', LinearRegression())
        ])
poly_out_d = poly_d.fit(X_train,y_train)
```

```
results['Poly'] = [np.sqrt(np.mean((poly_out_d.predict(X_train)- y_train)
results
```

Out[40]:

	Linear	Poly
Train RMSE	52.954165	52.954165
Test RMSE	55.651767	55.651767

Ridge Regression

- Ridge regression 은 `sklearn.linear_model` 모듈의 `Ridge` 라는 함수를 통해 계산할 수 있다.
- penalty parameter λ 는 이 함수에서는 `alpha` 라는 변수를 통해 조절할 수 있다.
- Documentation: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Ridge.html

활용 예제는 다음과 같다. 다음의 경우 `alpha=0` 을 입력함으로서 일반 linear regression과 같은 값을 가져오게 한다.

In [41]:
```
RidgeOut = Ridge(alpha=0).fit(X_train, y_train)
np.sqrt(np.mean((RidgeOut.predict(X_test)-y_test)**2))
```

Out[41]:
```
np.float64(55.65176693892665)
```

GridSearchCV

- 여기에서부턴 `KFold` 함수가 아닌 `GridSearchCV` 라는 함수를 통해 최적값을 찾도록 한다.
- hyperparameter Grid를 설정 해 두면 K-fold CV 를 통해 Grid 중 최적의 Hyperparameter 설정을 해준다.
- 필요사항: peformance measure 를 설정해주어야 한다. (MSE 등)
- Documentation: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html

Grid for Regularization

- potential λ 에 대한 적절한 Grid로는 10^{-7} 혹은 10^{-5} 부터 10^{2} 혹은 10 까지 값을 설정한다.
- 유의사항으로 $\lambda||\beta||$ 형태의 regularization 에 대해서 Grid는 log-scale 로 한다.

In [42]:
```
# log-scale v.s. linsear-scale
print("log-scale: ",np.logspace(-5,2, num=10).round(5))
print("linear-scale: ",np.linspace(10**(-5),10**2,num=10).round(5))
```

```
log-scale:  [1.000000e-05 6.000000e-05 3.600000e-04 2.150000e-03 1.292000e
-02
 7.743000e-02 4.641600e-01 2.782560e+00 1.668101e+01 1.000000e+02]
linear-scale:  [1.000000e-05 1.111112e+01 2.222223e+01 3.333334e+01 4.4444
45e+01
 5.555556e+01 6.666667e+01 7.777778e+01 8.888889e+01 1.000000e+02]
```

Q5. log-scale 과 linear-scale grid의 차이를 간략히 설명하시오.

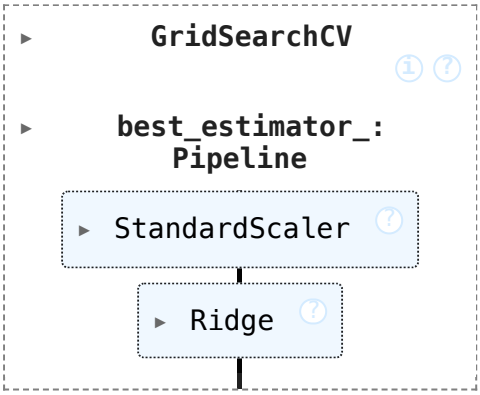
A5.

- Log-scale grid: log값이 동일한 간격으로 증가하기 때문에 실제 눈금은 동일한 배율로 증가한다.
- Linear-scale grid: 등차수열과 같이 같은 값의 차이로 grid가 증가한다.

In [43]:
```
RidgeReg = Pipeline(steps=[
    ('Standardize', StandardScaler()),
    ('Ridge_reg', Ridge())
])

# Grid Search
param_grid = {'Ridge_reg__alpha' : np.logspace(-6,2, num=15)}
RidgeCV = GridSearchCV(RidgeReg, param_grid, cv= 7, n_jobs = 15, scoring=
RidgeCV.fit(X_train,y_train)
```

Out[43]:



Q6. 위 코드는 GridSearchCV 를 활용해서 Ridge regression을 하는 예제이다. `param_grid` 를 위와 같은 dictionary 형태로 작성한 이유를 설명하시오.

Pipeline 으로 정의된 모델에 대한 추가 공부 필요

A6.

key name을 '{pipeline step name}__{parameter name}' 형식으로 지정하는 이유는 Pipeline 중 어떤 단계의 파라미터를 gridsearch 할지 지정해줄 필요가 있기 때문입니다.

그렇게 어떤 파라미터를 지정할지 key에 정의한 뒤 어떤 값들을 후보로 설정할지 np.logspace, np.linspace 등 array 형태의 데이터를 넣어 최적의 파라미터를 찾는 과정을 거치게 됩니다.

Q7. Ridge Regression의 Train RMSE, Test RMSE 를 구하시오

In [46]:
```
# 여기에 Q7 답안 작성
Ridge_train_RMSE = np.sqrt(mean_squared_error(y_train, RidgeCV.predict(X_
Ridge_test_RMSE = np.sqrt(mean_squared_error(y_test, RidgeCV.predict(X_te

# Report
results['Ridge'] = [Ridge_train_RMSE, Ridge_test_RMSE]
results
```

Out[46]:

	Linear	Poly	Ridge
Train RMSE	52.954165	52.954165	53.142049
Test RMSE	55.651767	55.651767	55.399790

In [ ]:
```

```