다변량분석 과제

2016170864 산업경영공학부 조동혁

선형회귀분석

In [28]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

In [32]:
```
data = pd.read_csv('UniversalBank.csv')
data.head()
```

Out[32]:

	ID	Age	Experience	Income	ZIP Code	Family	CCAvg	Education	Mortgage	Personal Loan	Securities Account	CD Account	Online	CreditCard
0	1	25	1	49	91107	4	1.6	1	0	0	1	0	0	0
1	2	45	19	34	90089	3	1.5	1	0	0	1	0	0	0
2	3	39	15	11	94720	1	1.0	1	0	0	0	0	0	0
3	4	35	9	100	94112	1	2.7	2	0	0	0	0	0	0
4	5	35	8	45	91330	4	1.0	2	0	0	0	0	0	1

In [35]:
```
data = data.drop(['ID','ZIP Code'], axis=1)
data.head()
```

Out[35]:

	Age	Experience	Income	Family	CCAvg	Education	Mortgage	Personal Loan	Securities Account	CD Account	Online	CreditCard
0	25	1	49	4	1.6	1	0	0	1	0	0	0
1	45	19	34	3	1.5	1	0	0	1	0	0	0
2	39	15	11	1	1.0	1	0	0	0	0	0	0
3	35	9	100	1	2.7	2	0	0	0	0	0	0
4	35	8	45	4	1.0	2	0	0	0	0	0	1

Data 는 UniversalBank.csv 파일을 사용했으며, y 변수는 income 을 잡아주었습니다. Education 카테고리형 변수는 변환을 해주지 않았고, 그 외에 범주형 변수들을 진행을 하는데 포함시키지 않았습니다. 또한, ID 및 ZIP CODE 같이 고유값을 보여주는 변수들은 미리 제거를 해주었습니다. 더미 변수들은 포함시키지 않았습니다.

In [33]:
```
#데이터크기와 결측치 확인

print(data.shape)
print(data.isnull().sum())
```

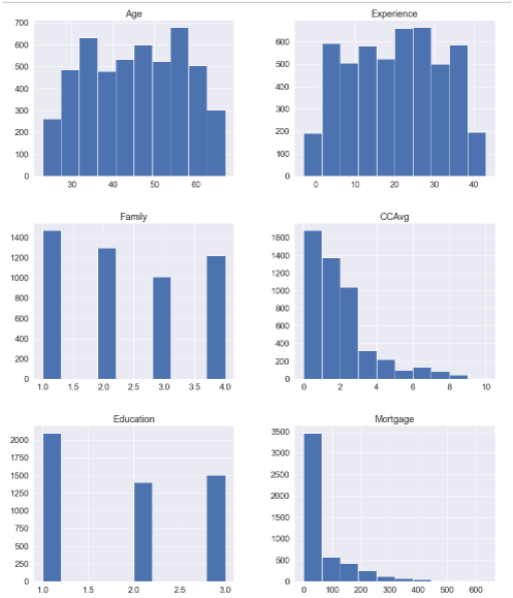
```
(5000, 14)
ID                    0
Age                   0
Experience            0
Income                0
ZIP Code              0
Family                0
CCAvg                 0
Education             0
Mortgage              0
Personal Loan         0
Securities Account    0
CD Account            0
Online                0
CreditCard            0
dtype: int64
```

In [34]:
```
data.columns
```

Out[34]:
```
Index(['ID', 'Age', 'Experience', 'Income', 'ZIP Code', 'Family', 'CCAvg',
       'Education', 'Mortgage', 'Personal Loan', 'Securities Account',
       'CD Account', 'Online', 'CreditCard'],
      dtype='object')
```

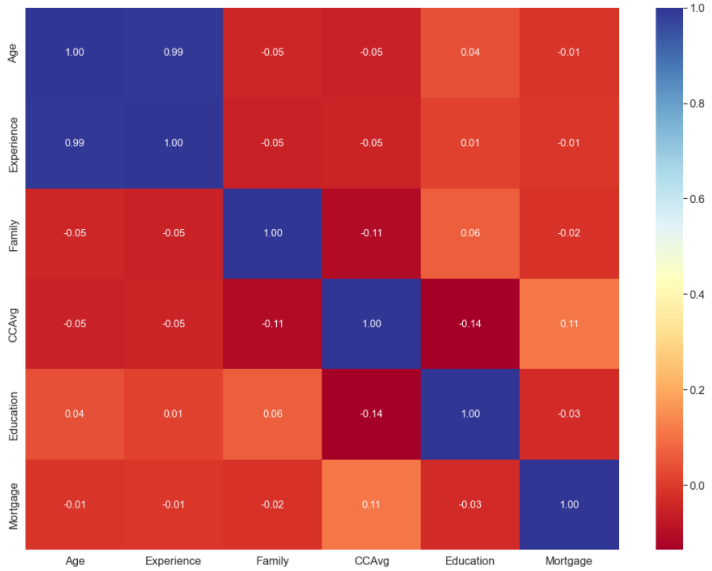
진행하기 앞서, missing value 들이 있는지 확인을 해주었지만, 없는 모습이었습니다.

In [39]:
```
#수치형 X변수의 히스토그램

numerical_columns = ['Age', 'Experience', 'Family', 'CCAvg',
       'Education', 'Mortgage']


fig = plt.figure(figsize = (16, 20))
ax = fig.gca()
data[numerical_columns].hist(ax=ax)
plt.show()
```



각각 변수들에 해당되는 히스토그램을 그려주었습니다. 또한, 변수마다 서로의 얼마나 correlation 을 가지고 있는지 확인하기 위해 heatmap 을 그려주었습니다. 파랑색은 높은, 빨강색은 낮은 모습입니다. 대각선에 해당되는 부분은, 자신과 자신의 correlation 을 나타내주기 때문에, 당연히 높은 값을 가지고 있는 모습입니다.

In [41]:
```
# heatmap (seaborn)
fig = plt.figure(figsize = (16, 12))
ax = fig.gca()

sns.set(font_scale = 1.5)  # heatmap 안의 font-size 설정
heatmap = sns.heatmap(corr.values, annot = True, fmt='.2f', annot_kws={'size':15},
                      yticklabels = cols, xticklabels = cols, ax=ax, cmap = "RdYlBu")
plt.tight_layout()
plt.show()
```



In [40]:
```
# Parson 상관계수
cols = ['Age', 'Experience', 'Family', 'CCAvg',
        'Education', 'Mortgage']

corr = data[cols].corr(method = 'pearson')
corr
```

Out[40]:

	Age	Experience	Family	CCAvg	Education	Mortgage
Age	1.000000	0.994215	-0.046418	-0.052030	0.041334	-0.012539
Experience	0.994215	1.000000	-0.052563	-0.050089	0.013152	-0.010582
Family	-0.046418	-0.052563	1.000000	-0.109285	0.064929	-0.020445
CCAvg	-0.052030	-0.050089	-0.109285	1.000000	-0.136138	0.109909
Education	0.041334	0.013152	0.064929	-0.136138	1.000000	-0.033327
Mortgage	-0.012539	-0.010582	-0.020445	0.109909	-0.033327	1.000000

또한, 위 내용들을 pearson method 를 통해 correlation 값들을 구해주었으며, 전부 높은 값이 아닌 낮은값을 가지고 있는 모습을 보여주었습니다.

In [44]:
```
#데이터 전처리
from sklearn.preprocessing import StandardScaler

# 변수 표준화

scaler = StandardScaler()  # 평균 0, 표준편차 1
scale_columns = ['Age', 'Experience', 'Family', 'CCAvg',
        'Education', 'Mortgage']
data[scale_columns] = scaler.fit_transform(data[scale_columns])
```

In [43]:
```
data.head()
```

Out[43]:

	Age	Experience	Income	Family	CCAvg	Education	Mortgage	Personal Loan	Securities Account	CD Account	Online	CreditCard
0	-1.774417	-1.666078	49	1.397414	-0.193371	-1.049078	-0.555524	0	1	0	0	0
1	-0.029524	-0.096330	34	0.525991	-0.250595	-1.049078	-0.555524	0	1	0	0	0
2	-0.552992	-0.445163	11	-1.216855	-0.536720	-1.049078	-0.555524	0	0	0	0	0
3	-0.901970	-0.968413	100	-1.216855	0.436103	0.141703	-0.555524	0	0	0	0	0
4	-0.901970	-1.055621	45	1.397414	-0.536720	0.141703	-0.555524	0	0	0	0	1

In [45]:
```
# 학습데이터 테스트데이터 나누기

from sklearn.model_selection import train_test_split

# split dataset into training & test
X = data[numerical_columns]
y = data['Income']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1)

print(X_train.shape, y_train.shape)
print(X_test.shape, y_test.shape)
```

```
(4000, 6) (4000,)
(1000, 6) (1000,)
```

단순회귀분석은 표준화를 할 필요가 없지만, 다중회귀분석에서는 변수마다 가지고 있는 scale 값이 다르기 때문에 표준화를 진행 해주었습니다. 그리고, 각 데이터를 80:20 비율로 나누어 주었습니다. 80 은 실제 데이터이며, 20 은 예측하는 값으로 사용을 해주었습니다.

In [46]:
```
#다중공선성

from statsmodels.stats.outliers_influence import variance_inflation_factor

vif = pd.DataFrame()
vif['features'] = X_train.columns
vif["VIF Factor"] = [variance_inflation_factor(X_train.values, i) for i in range(X_train.shape[1])]
vif.round(1)
```

Out[46]:

	features	VIF Factor
0	Age	95.9
1	Experience	95.8
2	Family	1.0
3	CCAvg	1.0
4	Education	1.1
5	Mortgage	1.0

다중공선성이 있는지 확인을 해보았습니다. 여기서는 Age 및 experience 는 굉장히 높은 모습을 보여주고 있습니다. 원래는 두 변수를 제거를 해주어야하지만, 진행을 한번 강행 해보았습니다.

In [48]:
```
#회귀 모델링
from sklearn import linear_model

# fit regression model in training set
lr = linear_model.LinearRegression()
model = lr.fit(X_train, y_train)

# predict in test set
pred_test = lr.predict(X_test)
```

In [50]:
```
# 회귀 계수, 회귀식의 적합도 및 모델 해석
print(lr.coef_)

# 회귀 계수 DataFrame 만들기
coefs = pd.DataFrame(zip(data[numerical_columns].columns, lr.coef_), columns = ['feature', 'coefficients'])
coefs
```

```
[-18.69189575  17.75221666  -3.84650049  28.15046621  -4.00379957
   6.27548902]
```

Out[50]:

	feature	coefficients
0	Age	-18.691896
1	Experience	17.752217
2	Family	-3.846500
3	CCAvg	28.150466
4	Education	-4.003800
5	Mortgage	6.275489

모델링을 돌린 결과, coefficient 가 나와주었습니다. 이 값들은 각자 얼마나 영향을 주고 있는지를 나타내주고 있습니다. 양수값들은 양의 영향을 주고, 음수값들은 음의 영향을 주는 모습입니다.

In [52]:
```
import statsmodels.api as sm

X_train2 = sm.add_constant(X_train)
model2 = sm.OLS(y_train, X_train2).fit()
model2.summary()
```

Out [52]:

OLS Regression Results

Dep. Variable:	Income	**R-squared:**	0.455
Model:	OLS	**Adj. R-squared:**	0.454
Method:	Least Squares	**F-statistic:**	554.7
Date:	Fri, 29 Apr 2022	**Prob (F-statistic):**	0.00
Time:	03:53:40	**Log-Likelihood:**	-19780.
No. Observations:	4000	**AIC:**	3.957e+04
Df Residuals:	3993	**BIC:**	3.962e+04
Df Model:	6		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	73.6924	0.538	136.994	0.000	72.638	74.747
Age	-18.6919	5.252	-3.559	0.000	-28.988	-8.396
Experience	17.7522	5.248	3.383	0.001	7.463	28.042
Family	-3.8465	0.543	-7.080	0.000	-4.912	-2.781
CCAvg	28.1505	0.550	51.188	0.000	27.072	29.229
Education	-4.0038	0.564	-7.094	0.000	-5.110	-2.897
Mortgage	6.2755	0.549	11.424	0.000	5.198	7.353

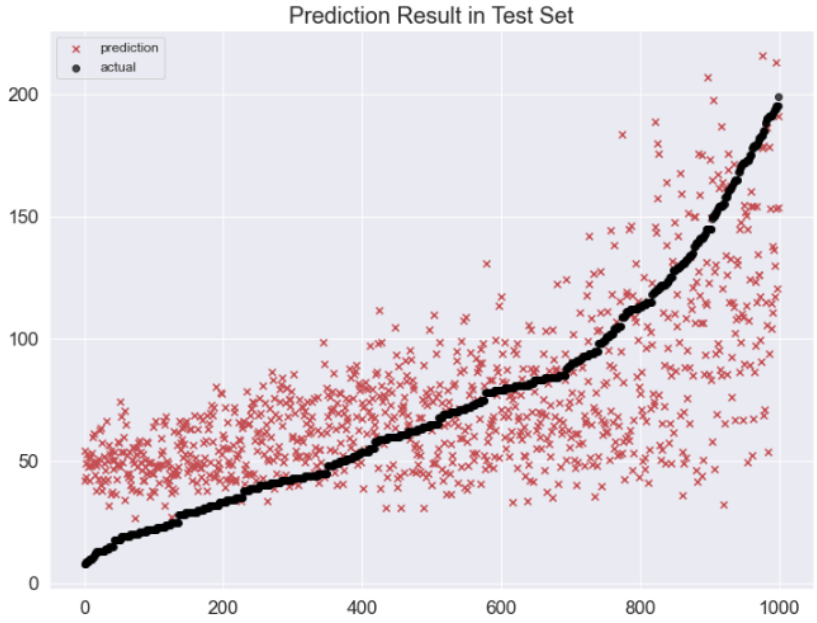
Omnibus:	695.973	**Durbin-Watson:**	1.989
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	1173.108
Skew:	1.143	**Prob(JB):**	1.83e-255
Kurtosis:	4.345	**Cond. No.**	19.6

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

조금 더 보기 편하게 결과를 도출 해주었습니다. 여기서는 R-squared 값이 낮은 모습을 보여주고 있습니다. 이 뜻은, 다른 설명변수중에 Income 을 포함한 설명력 있는 변수들이 부족한 모습입니다. P-value 는 낮게 0 이 나왔으니 귀무가설을 채택 한다는 뜻입니다. 즉, 설명변수들이 영향력이 없다는 결과가 나왔습니다. AIC 및 BIC 값들은 낮은 모습을 보니, 잘 설명을 해주는 모델이라고 하지 못한다는 결과가 나왔습니다.

Durbin Watson 의 결과에 따르면 2 에 굉장히 가까우므로, 잔차들이 자기 상관계수가 없다는 것을 보여주고 있습니다. 하지만, 여전히 모두 독립인 모습입니다.

In [59]:
```
#모델 예측 및 성능평가
# 예측 결과 시각화 (test set)
df = pd.DataFrame({'actual': y_test, 'prediction': pred_test})
df = df.sort_values(by='actual').reset_index(drop=True)

plt.figure(figsize=(12, 9))
plt.scatter(df.index, df['prediction'], marker='x', color='r')
plt.scatter(df.index, df['actual'], alpha=0.7, marker='o', color='black')
plt.title("Prediction Result in Test Set", fontsize=20)
plt.legend(['prediction', 'actual'], fontsize=12)
plt.show()
```



마지막으로 시각화를 하기 위해 prediction result를 그려주었습니다. prediction들이 초반에는 맞질 않지만, 증가할수록 그래도 어느정도는 actual data 값에 비슷하게 나오는 모습을 보여주었습니다.

이번 과제가 끝나고, 왜 제가 설계한 모델이 좋은 모델이 아닌지에 대해서 생각을 한번 해보았습니다. 아무래도, 우선 단순 회귀분석을 통해서 같은 Y 값의 R-squared 값을 구해주었으면, 비교를 할수 있었을 것 같습니다. 또한, 다중공산성 분석을 할 때, 큰 값을 가졌던 변수들을 제거를 해주지 못한 문제가 있었던거 같습니다.

로지스틱 회귀분석

```
In [2]: import numpy as np
        import pandas as pd

In [5]: from IPython.display import display

In [4]: import seaborn as sns
        import matplotlib.pyplot as plt

In [6]: # 기계학습 모델
        import statsmodels.api as sm
        from sklearn.model_selection import train_test_split
        import sklearn.preprocessing as preprocessing
        from sklearn import metrics
        from sklearn.metrics import confusion_matrix
        from sklearn.metrics import accuracy_score, roc_auc_score, roc_curve
```

```
In [23]: data = pd.read_csv('Parkinson.csv')
         data.head()
```

Out [23]:

	name	status	MDVP:Fo(Hz)	MDVP:Fhi(Hz)	MDVP:Flo(Hz)	MDVP:Jitter(%)	MDVP:Jitter(Abs)	MDVP:RAP	MDVP:PPQ	Jitter:DDP	...	MDVP:APQ
0	phon_R01_S01_1	1	-0.829300	-0.436165	-0.952037	0.334914	0.749759	0.132963	0.760800	0.131755	...	0.332985
1	phon_R01_S01_2	1	-0.770972	-0.530974	-0.057721	0.715418	1.037674	0.453892	1.276809	0.452684	...	1.159454
2	phon_R01_S01_3	1	-0.909476	-0.723168	-0.109875	0.884991	1.325589	0.720770	1.585687	0.721813	...	0.699187
3	phon_R01_S01_4	1	-0.909622	-0.649092	-0.114229	0.775389	1.325589	0.578885	1.284076	0.577677	...	0.806859
4	phon_R01_S01_5	1	-0.925657	-0.606245	-0.130608	1.368893	1.901418	1.095750	2.047187	1.096793	...	1.216839

5 rows × 24 columns

파일은 파킨슨 파일 data를 사용해서 진행을 해주었습니다. Data는 사람들의 목소리 녹음된 형태에 관한 데이터였고, status에 해당되는 column은 범주형으로, 0은 건강한 사람, 1은 파킨슨에 걸린 사람으로 구별해져 있었습니다.

이름에 해당되는 변수는 고유값이기에 제거가 필요했습니다. 제거 후 23개의 column들로 이루어졌습니다. 타케고리칼 변수는 존재하지 않았기에, 따로 dummy변수로 변환을 해주지는 않았습니다.

```
In [24]: data = data.drop(['name'], axis=1)
         data.head()
         #카테고리형 변수가 없으므로, 더미화 시키지 않았다
```

Out [24]:

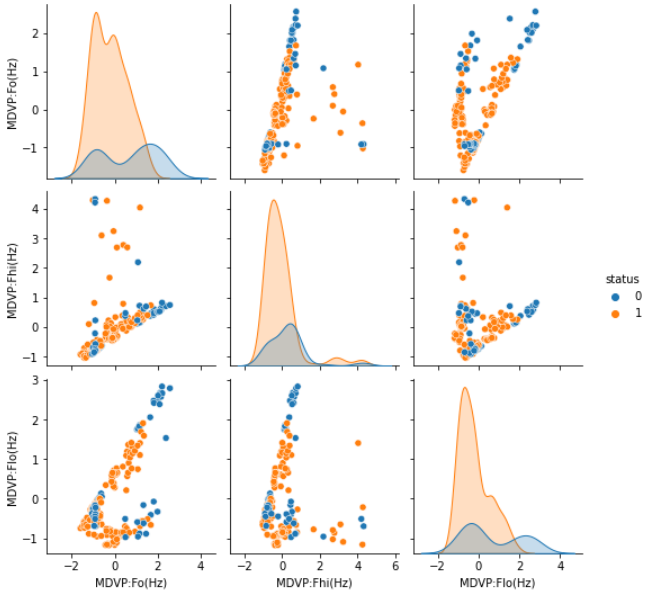
:AP	MDVP:PPQ	Jitter:DDP	MDVP:Shimmer	...	MDVP:APQ	Shimmer:DDA	NHR	HNR	RPDE	DFA	spread1	spread2	D2	PPE
963	0.760800	0.131755	0.745985	...	0.332985	0.607532	-0.067893	-0.193225	-0.807838	1.760814	0.801323	0.480477	-0.210531	0.868886
892	1.276809	0.452684	1.681731	...	1.159454	1.548254	-0.137843	-0.634508	-0.387524	1.837562	1.479853	1.311185	0.275077	1.803605
770	1.585687	0.721813	1.202693	...	0.699187	1.175323	-0.291633	-0.279760	-0.662075	1.942048	1.141445	1.017682	-0.103629	1.402661
885	1.284076	0.577677	1.340396	...	0.806859	1.340229	-0.280719	-0.281346	-0.613134	1.832380	1.440945	1.293840	0.062145	1.806954
750	2.047187	1.096793	1.836448	...	1.216839	1.899461	-0.178026	-0.506745	-0.783021	1.909364	1.780940	0.096195	-0.130026	2.267082

```
In [25]: data.shape
```

Out [25]: (195, 23)

In [43]:
```
plt.figure(figsize = (8,8))
sns.pairplot(data[['MDVP:Fo(Hz)', 'MDVP:Fhi(Hz)', 'MDVP:Flo(Hz)', 'status']], hue="status")
plt.show()
#0 for healthy and 1 for PD
#범주형 변수로는 status를 사용, 그 외 변수는 3가지를 사용했다.
```

```
<Figure size 576x576 with 0 Axes>
```



- MDVP:Fo(Hz) - Average vocal fundamental frequency
- MDVP:Fhi(Hz) - Maximum vocal fundamental frequency
- MDVP:Flo(Hz) - Minimum vocal fundamental frequency

세가지 변수들을 사용해서 파킨슨 병 유무를 판단하기 위해 Pairplot을 사용해서 찍어주었습니다. 여기서는 위에서 언급했듯이, 1은 파킨슨에 걸린 사람, 0은 병에 걸리지 않은 건강한 사람을 나타내주고 있는 모습입니다.

In [44]:
```
display(data['status'].value_counts())
```

```
1    147
0     48
Name: status, dtype: int64
```

In [45]:
```
X = data.drop('status', axis=1)
y = data['status']
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.3, random_state=20190625, stratify=y)
print('클래스별 데이터 개수: Training 데이터  ')
display(y_train.value_counts())
print('클래스별 데이터 개수: Testing')
display(y_test.value_counts())
```

```
클래스별 데이터 개수: Training 데이터

1    103
0     33
Name: status, dtype: int64

클래스별 데이터 개수: Testing

1    44
0    15
Name: status, dtype: int64
```

In [46]:
```
model = sm.Logit(y_train, X_train)
model_fitted = model.fit(method='newton')
```

```
Optimization terminated successfully.
         Current function value: 0.476728
         Iterations 10
```

데이터셋을 training:test size 7:3비율로 나누어 줬습니다. 여기서 파악할수 있었던 부분은, 147명이 파킨슨병에 걸린 status였고, 48명이 건강한 상태였습니다. Stratify를 사용하여 최대한 비율을 맞춰주려고 노력을 했습니다.

```
In [47]: model_fitted.summary()
```

Logit Regression Results

Dep. Variable:	status	**No. Observations:**	136
Model:	Logit	**Df Residuals:**	114
Method:	MLE	**Df Model:**	21
Date:	Fri, 29 Apr 2022	**Pseudo R-squ.:**	0.1397
Time:	00:30:37	**Log-Likelihood:**	-64.835
converged:	True	**LL-Null:**	-75.359
Covariance Type:	nonrobust	**LLR p-value:**	0.4560

	coef	std err	z	P>|z|	[0.025	0.975]
MDVP:Fo(Hz)	-1.1419	0.689	-1.657	0.097	-2.492	0.209
MDVP:Fhi(Hz)	-0.0762	0.246	-0.309	0.757	-0.559	0.407
MDVP:Flo(Hz)	-0.7035	0.391	-1.801	0.072	-1.469	0.062
MDVP:Jitter(%)	-2.9022	3.631	-0.799	0.424	-10.019	4.214
MDVP:Jitter(Abs)	-1.6009	1.737	-0.922	0.357	-5.006	1.804
MDVP:RAP	109.1858	275.183	0.397	0.692	-430.162	648.534
MDVP:PPQ	-2.7357	2.454	-1.115	0.265	-7.546	2.075
Jitter:DDP	-103.2639	275.055	-0.375	0.707	-642.361	435.834
MDVP:Shimmer	-0.3413	6.117	-0.056	0.956	-12.330	11.647
MDVP:Shimmer(dB)	-0.4154	2.509	-0.166	0.868	-5.333	4.502
Shimmer:APQ3	-141.9052	938.434	-0.151	0.880	-1981.202	1697.391
Shimmer:APQ5	-0.1485	2.398	-0.062	0.951	-4.848	4.551
MDVP:APQ	0.6210	1.713	0.363	0.717	-2.735	3.978
Shimmer:DDA	142.7975	938.216	0.152	0.879	-1696.071	1981.666
NHR	-0.3469	0.807	-0.430	0.667	-1.928	1.234
HNR	-0.0700	0.698	-0.100	0.920	-1.437	1.297
RPDE	-0.6461	0.450	-1.434	0.151	-1.529	0.237
DFA	0.0860	0.396	0.217	0.828	-0.690	0.862
spread1	-0.0610	0.993	-0.061	0.951	-2.008	1.886
spread2	0.6325	0.397	1.593	0.111	-0.146	1.411
D2	0.1013	0.439	0.230	0.818	-0.760	0.962
PPE	1.8626	1.242	1.500	0.134	-0.571	4.297

로지스틱 모델로 y를 status변수를 두고 돌려주었습니다. Model를 fit할때는 Newton method를 사용해주었습니다. 신뢰구간 0.05범위안에서 결과값들이 나오는걸 볼수가 있습니다. Z-test로 하지만, p-value값들이 지나치게 높은걸 봐서는, 그렇게 유의미한 data는 아니라고 생각이 들었습니다.

In [48]:
```
coef= model_fitted.params
print(coef)
```

```
MDVP:Fo(Hz)          -1.141871
MDVP:Fhi(Hz)         -0.076160
MDVP:Flo(Hz)         -0.703464
MDVP:Jitter(%)       -2.902238
MDVP:Jitter(Abs)     -1.600857
MDVP:RAP            109.185765
MDVP:PPQ             -2.735661
Jitter:DDP         -103.263898
MDVP:Shimmer         -0.341294
MDVP:Shimmer(dB)     -0.415381
Shimmer:APQ3       -141.905237
Shimmer:APQ5         -0.148506
MDVP:APQ              0.621026
Shimmer:DDA         142.797496
NHR                  -0.346913
HNR                  -0.069990
RPDE                 -0.646121
DFA                   0.085962
spread1              -0.060977
spread2               0.632540
D2                    0.101268
PPE                   1.862650
dtype: float64
```

In [49]:
```
np.exp(coef)
```

Out[49]:
```
MDVP:Fo(Hz)          3.192213e-01
MDVP:Fhi(Hz)         9.266681e-01
MDVP:Flo(Hz)         4.948682e-01
MDVP:Jitter(%)       5.490021e-02
MDVP:Jitter(Abs)     2.017236e-01
MDVP:RAP             2.622860e+47
MDVP:PPQ             6.485113e-02
Jitter:DDP           1.422522e-45
MDVP:Shimmer         7.108498e-01
MDVP:Shimmer(dB)     6.600886e-01
Shimmer:APQ3         2.351467e-62
Shimmer:APQ5         8.619948e-01
MDVP:APQ             1.860837e+00
Shimmer:DDA          1.037921e+62
NHR                  7.068670e-01
HNR                  9.324033e-01
RPDE                 5.240745e-01
DFA                  1.089765e+00
spread1              9.408453e-01
spread2              1.882386e+00
D2                   1.106574e+00
PPE                  6.440781e+00
dtype: float64
```

In [50]:
```
#성능평가
train_prob = model_fitted.predict(X_train)
train_prob.head()
```

Out[50]:
```
193    0.039306
10     0.908283
128    0.249278
106    0.315495
41     0.109234
dtype: float64
```

Odds ratio값들 역시 계산을 해주었습니다. 각 변수들이 1만큼 증가할 때, 확률이 값들만큼 증가한다는 뜻입니다.

In [52]:
```
#training 데이터 예측성능
train_prob = model_fitted.predict(X_train)
train_results = pd.concat([train_prob, y_train], axis=1)
train_results.columns = ['Predicted Probability of Class 1', 'Status']
display(train_results)
```

	Predicted Probability of Class 1	Status
193	0.039306	0
10	0.908283	1
128	0.249278	1
106	0.315495	1
41	0.109234	1
...	...	...
19	0.813061	1
35	0.029035	0
168	0.676961	0
76	0.943596	1
123	0.424323	1

136 rows × 2 columns

Training 데이터의 예측 성능을 해보았습니다. 여기서는 0 에 해당되는 확률들은 상당히 낮지만, 1 에 해당되는 확률들은 높다는 것을 알수 있었습니다.

In [53]:
```
#test 데이터 예측성능
test_prob = model_fitted.predict(X_test)
test_results = pd.concat([test_prob, y_test], axis=1)
test_results.columns = ['Predicted Probability of Class 1', 'Status']
display(test_results)
```

위와 똑같이 여기서는 test 데이터들을 예측해주었습니다. 이제는 threshold 를 정해주는 작업을 해주었습니다.

	Predicted Probability of Class 1	Status
120	0.158968	1
38	0.183355	1
145	0.640284	1
130	0.710473	1
184	0.358567	0
132	0.935616	1
160	0.841999	1
20	0.946659	1
4	0.666759	1
64	0.008061	0
107	0.305064	1
138	0.860223	1
110	0.869530	1
34	0.011399	0
154	0.754048	1
95	0.614352	1
143	0.291328	1
101	0.762226	1
126	0.319852	1
169	0.007225	0
50	0.184831	0
113	0.186993	1
77	0.912726	1
44	0.002073	0
39	0.327981	1
22	0.932334	1
67	0.747535	1
152	0.139209	1
137	0.957184	1
153	0.460112	1
52	0.230114	0

In [54]:
```
train_class = train_prob.copy()
test_class = test_prob.copy()

train_class[train_class > 0.5] = 1
train_class[train_class <= 0.5] = 0

test_class[test_class > 0.5] = 1
test_class[test_class <= 0.5] = 0
```

In [55]:
```
# Traiing Accuracy
#accuracy_score(y_train, train_class)
display(pd.DataFrame(confusion_matrix(y_train, train_class), columns=[0,1], index=[0,1]))
print('Training Accuracy: {:.3f}'.format(accuracy_score(y_train, train_class)))
```

	0	1
0	27	6
1	28	75

Training Accuracy: 0.750

In [56]:
```
# Test Accuracy
#accuracy_score(y_test, test_class)
display(pd.DataFrame(confusion_matrix(y_test, test_class), columns=[0,1], index=[0,1]))
print('Testing Accuracy: {:.3f}'.format(accuracy_score(y_test, test_class)))
```

	0	1
0	14	1
1	13	31

Testing Accuracy: 0.763

처음에는 0.5값으로, class1에 속할 확률이 0.5를 초과하면 1이라고 하고, 같거나 작으면 0으로 하겠다고 설정을 해주었습니다. 전체중에 대각선인 14,31인 accuracy를 봐주었습니다. 이때는 accuracy가 각각 0.750, 0.763이 나왔으므로, 그다지 만족스럽진 않아서 다른 값으로 threshold를 설정해주었습니다.

In [57]:
```
train_class_2 = train_prob.copy()
test_class_2 = test_prob.copy()

train_class_2[train_class_2 > 0.3] = 1
train_class_2[train_class_2 <= 0.3] = 0

test_class_2[test_class_2 > 0.3] = 1
test_class_2[test_class_2 <= 0.3] = 0

print('Train Accuracy: {:.3f}'.format(accuracy_score(y_train, train_class_2)))
display(pd.DataFrame(confusion_matrix(y_train, train_class_2), columns=[0,1], index=[0,1]))
print('Test Accuracy: {:.3f}'.format(accuracy_score(y_test, test_class_2)))
display(pd.DataFrame(confusion_matrix(y_test, test_class_2), columns=[0,1], index=[0,1]))
```

Train Accuracy: 0.860

	0	1
0	24	9
1	10	93

Test Accuracy: 0.831

	0	1
0	12	3
1	7	37

0.3값으로 해주었더니, test accuracy가 0.5보다는 올라간 모습을 볼수가 있었습니다. 하지만, 여기서도 만족을 할수 없어서, 마지막으로 다시한번 해주었습니다.

```
train_class_3 = train_prob.copy()
test_class_3 = test_prob.copy()

train_class_3[train_class_3 > 0.1] = 1
train_class_3[train_class_3 <= 0.1] = 0

test_class_3[test_class_3 > 0.1] = 1
test_class_3[test_class_3 <= 0.1] = 0

print('Train Accuracy: {:.3f}'.format(accuracy_score(y_train, train_class_3)))
display(pd.DataFrame(confusion_matrix(y_train, train_class_3), columns=[0,1], index=[0,1]))
print('Test Accuracy: {:.3f}'.format(accuracy_score(y_test, test_class_3)))
display(pd.DataFrame(confusion_matrix(y_test, test_class_3), columns=[0,1], index=[0,1]))
```

```
Train Accuracy: 0.860
```

	0	1
0	15	18
1	1	102

```
Test Accuracy: 0.915
```

	0	1
0	10	5
1	0	44

마지막으로, 0.1값으로 돌려주니, 훨씬 높은 test accuracy값이 나오는걸 볼수가 있었습니다. 다만, 이렇게 될 경우에는 data를 나눠줄 때 너무 극단적으로써, 좋은 모델로 분류하기에는 적합하지 않다고 생각이 들었습니다.