Homework 2

20200639 Chae Woo Jin

Chapter 4: 1, 4, 10, 12, 14

Chapter 5: 1, 2, 5, 6, 7

Chapter 4

#1.

$$(4.2)\ p(X) = \frac{e^{\beta_0+\beta_1 X}}{1+e^{\beta_0+\beta_1 X}},\ \frac{p(X)}{1-p(X)} = \frac{\frac{e^{\beta_0+\beta_1 X}}{1+e^{\beta_0+\beta_1 X}}}{\frac{1}{1+e^{\beta_0+\beta_1 X}}} = e^{\beta_0+\beta_1 X}\ (4.3)$$

#4.

(a) On average, $\frac{1}{10}$ will be used to make the precision, because observations are uniformly distribution.

(b) On average, when $p = 2$, $\frac{1}{10} \times \frac{1}{10} = \frac{1}{100}$ will be used.

(c) On average, when $p = 100$, $\left(\frac{1}{10}\right)^{100}$ will be used.

(d) As p increases linear, the proportion of observations against whole data set which are used to predict the real value decreases exponentially.

(e) The length of p-dimensional hypercube which contains 10% of the training observation is $\sqrt[p]{\frac{1}{10}}$, so that the volume of this hypercube is $\frac{1}{10}$, which is a proportion that it contains its training observations.

#10.

$$\log\left(\frac{\Pr(Y=k|X=x)}{\Pr(Y=K|X=x)}\right)=\log\left(\frac{\pi_k f_k(x)}{\pi_K f_K(x)}\right)$$

$$=\log\left(\frac{\pi_k \exp\left(-\frac{1}{2}(x-\mu_k)^2/\sigma^2\right)}{\pi_K \exp\left(-\frac{1}{2}(x-\mu_K)^2/\sigma^2\right)}\right)$$

$$=\log\left(\frac{\pi_k}{\pi_K}\right)-\frac{1}{2}\frac{(x-\mu_k)^2}{\sigma^2}+\frac{1}{2}\frac{(x-\mu_K)^2}{\sigma^2}$$

$$=\log\left(\frac{\pi_k}{\pi_K}\right)-\frac{1}{2\sigma^2}\{(x-\mu_k)^2-(x-\mu_K)^2\}$$

$$=\log\left(\frac{\pi_k}{\pi_K}\right)-\frac{1}{2\sigma^2}\{2(\mu_K-\mu_k)x-(\mu_K^2-\mu_k^2)\}$$

$$=\log\left(\frac{\pi_k}{\pi_K}\right)+\frac{\mu_K^2-\mu_k^2}{2\sigma^2}+\frac{(\mu_k-\mu_K)x}{\sigma^2}$$

$$=a_k+b_kx$$

$$\therefore a_k=\log\left(\frac{\pi_k}{\pi_K}\right)+\frac{\mu_K^2-\mu_k^2}{2\sigma^2},\ b_k=\frac{\mu_k-\mu_K}{\sigma^2}$$

#12.

(a) $\log\left(\frac{\widehat{Pr}(Y=orange|X)}{1-\widehat{Pr}(Y=orange|X)}\right)=\widehat{\beta_0}+\widehat{\beta_1}X$. Therefore, log odds of orange versus apple in my model is $\widehat{\beta_0}+\widehat{\beta_1}X$

(b) $\log\left(\frac{\widehat{Pr}(Y=orange|X)}{1-\widehat{Pr}(Y=orange|X)}\right)=(\hat{\alpha}_{orange0}-\hat{\alpha}_{apple0})+(\hat{\alpha}_{orange1}-\hat{\alpha}_{apple1})X$. Therefore, log odds of orange versus apple in friend's model is $(\hat{\alpha}_{orange0}-\hat{\alpha}_{apple0})+(\hat{\alpha}_{orange1}-\hat{\alpha}_{apple1})X$

(c) $\begin{cases}\hat{\alpha}_{orange0}-\hat{\alpha}_{apple0}=2\\ \hat{\alpha}_{orange1}-\hat{\alpha}_{apple1}=-1\end{cases}$. However, finding a solution of 4 parameters with 2 equations is impossible.

(d) $\begin{cases}\hat{\alpha}_{orange0}-\hat{\alpha}_{apple0}=1.2-3=-1.8=\widehat{\beta_0}\\ \hat{\alpha}_{orange1}-\hat{\alpha}_{apple1}=-2-0.6=-2.6=\widehat{\beta_1}\end{cases}$.

(e) The models are identical with different parameterization so they should perfectly agree.

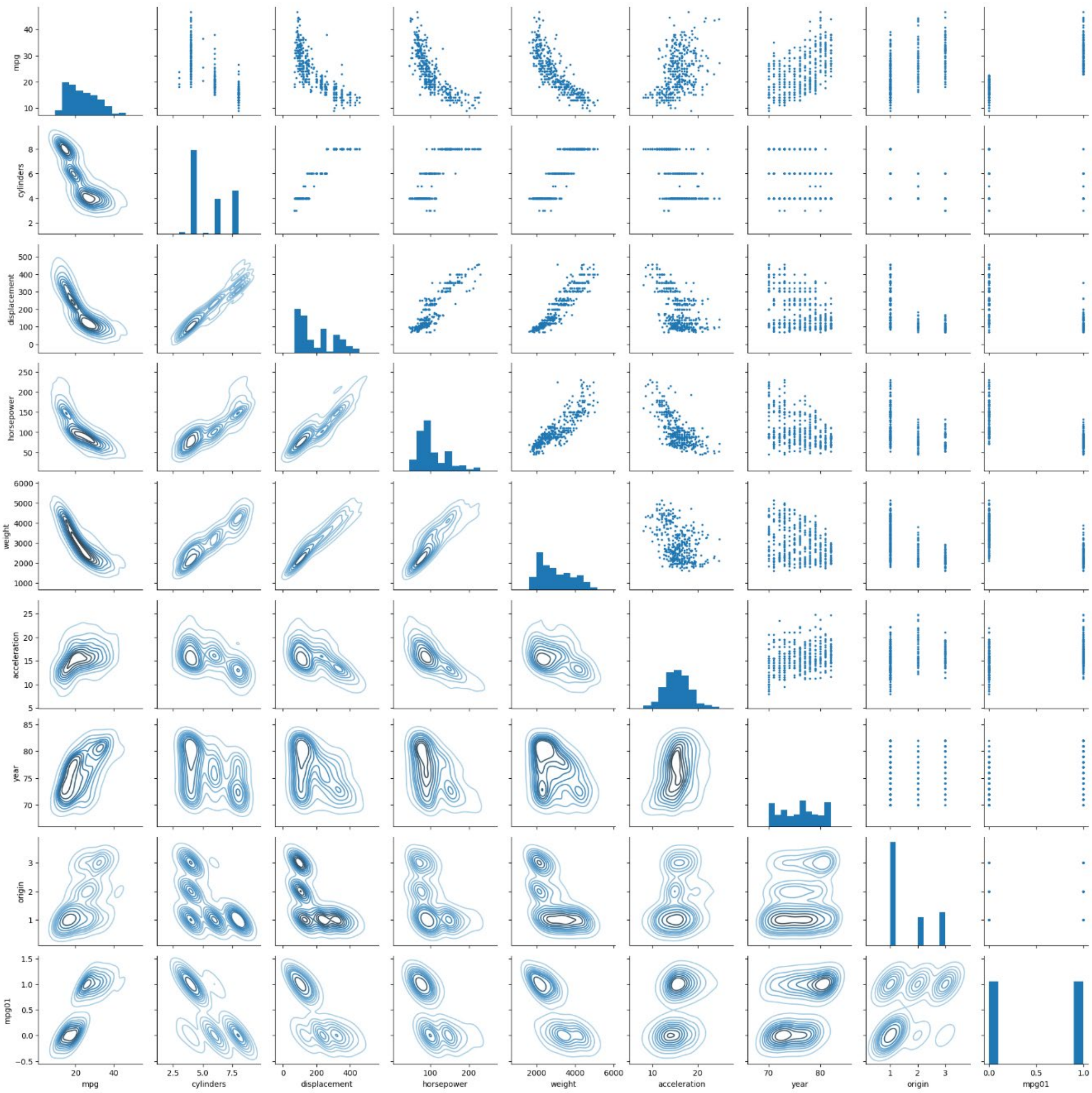
#14.

(a)

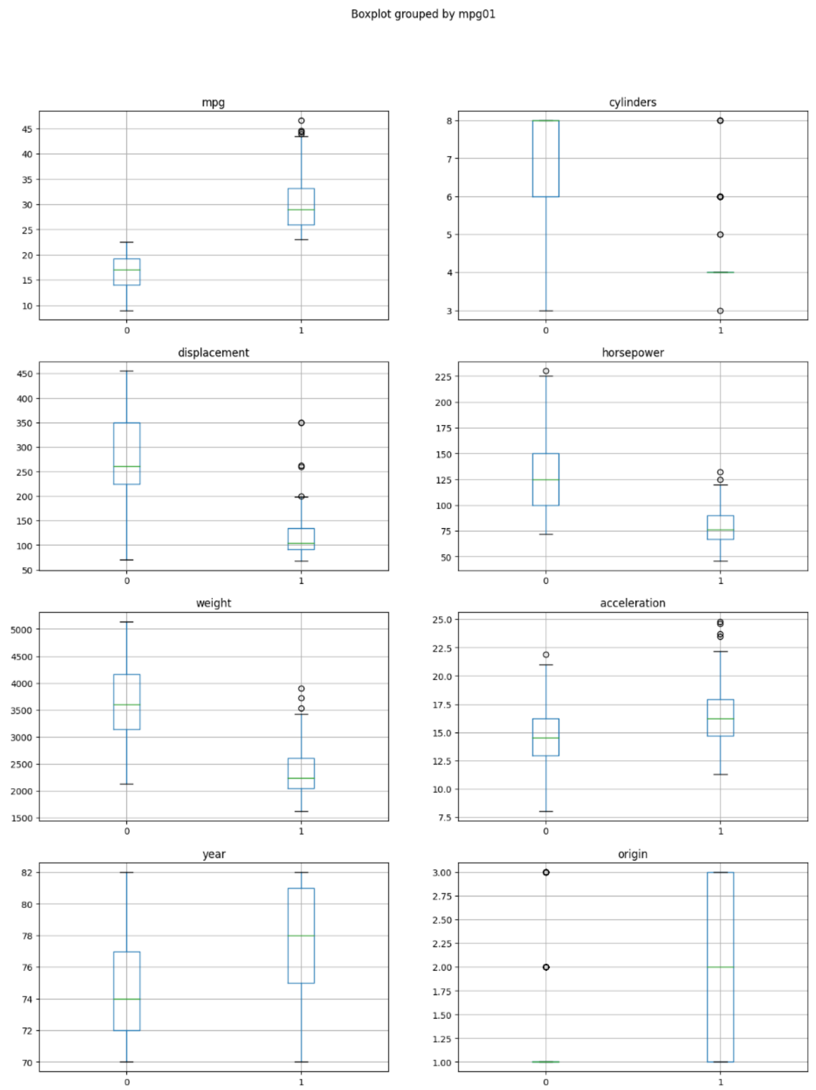
```
[92] med = df['mpg'].median()

     df['mpg01'] = df['mpg'].apply(lambda x: 1 if x > med else 0)
```

(b)



[Scatterplots between features]



[Boxplots grouped by mpg01]

'cylinders', 'displacement', 'horsepower', 'weight', 'acceleration', and 'year' seems well to fit mpg01

(c)

```
[98] x = np.array(df[['cylinders', 'displacement', 'horsepower', 'weight', 'acceleration', 'year']])
     y = np.array(df['mpg01'])
```

```
[99] # i.
     x_train, x_test, y_train, y_test = train_test_split(
         x, y, test_size=0.33, random_state=1
     )
     print(len(x_train))
     print(len(x_test))
     print(len(y_train))
     print(len(y_test))

     262
     130
     262
     130
```

※ **From (d) to (h), 'cylinders', 'displacement', 'horsepower', 'weight', 'acceleration', and 'year' will be used as predictors of 'mpg01'.**

(d)

```
[101] lda = LinearDiscriminantAnalysis()
      lda.fit(x_train, y_train)

      print('Test Error Rate is {0}'.format(1-lda.score(x_test, y_test)))

      Test Error Rate is 0.08461538461538465
```

```
[102] display(confusion_table(confusion_matrix(y_test, lda.predict(x_test))))
```

	y_pred=0	y_pred=1	Total
y=0	58	10	68
y=1	1	61	62
Total	59	71	

Test error (LDA): 0.084615

(e)

```
[103] qda = QuadraticDiscriminantAnalysis()
      qda.fit(x_train, y_train)

      print('Test Error Rate is {0}'.format(1-qda.score(x_test, y_test)))

      Test Error Rate is 0.0692307692307692
```

```
[104] display(confusion_table(confusion_matrix(y_test, qda.predict(x_test))))
```

	y_pred=0	y_pred=1	Total
y=0	62	6	68
y=1	3	59	62
Total	65	65	

Test error (QDA): 0.069230

(f)

```
[105] lgr = LogisticRegression(max_iter=5000)
      lgr.fit(x_train, y_train)

      print('Test Error Rate is {0}'.format(1 - lgr.score(x_test, y_test)))
```

```
Test Error Rate is 0.06153846153846154
```

```
[106] display(confusion_table(confusion_matrix(y_test, lgr.predict(x_test))))
```

	y_pred=0	y_pred=1	Total
y=0	63	5	68
y=1	3	59	62
Total	66	64	

Test error (Logistic Regression): 0.061538

(g)

```
[107] gnb = GaussianNB()
      gnb.fit(x_train, y_train)

      print('Test Error Rate is {0}'.format(1 - gnb.score(x_test, y_test)))
```

```
Test Error Rate is 0.07692307692307687
```

```
[108] display(confusion_table(confusion_matrix(y_test, gnb.predict(x_test))))
```

	y_pred=0	y_pred=1	Total
y=0	62	6	68
y=1	4	58	62
Total	66	64	

Test error (Naïve Bayes): 0.076923

(h)

```
[109] testErrList = []
      for k in range(1, 21):
        knn = KNeighborsClassifier(n_neighbors=k)
        knn.fit(x_train, y_train)

        print('Test Error Rate(k={0}) is {1}'.format(k, 1 - knn.score(x_test, y_test)))
        testErrList.append(1 - knn.score(x_test, y_test))
```

```
Test Error Rate(k=1) is 0.10769230769230764
Test Error Rate(k=2) is 0.13076923076923075
Test Error Rate(k=3) is 0.13076923076923075
Test Error Rate(k=4) is 0.11538461538461542
Test Error Rate(k=5) is 0.13076923076923075
Test Error Rate(k=6) is 0.13076923076923075
Test Error Rate(k=7) is 0.09999999999999998
Test Error Rate(k=8) is 0.08461538461538465
Test Error Rate(k=9) is 0.09999999999999998
Test Error Rate(k=10) is 0.09999999999999998
Test Error Rate(k=11) is 0.12307692307692308
Test Error Rate(k=12) is 0.11538461538461542
Test Error Rate(k=13) is 0.12307692307692308
Test Error Rate(k=14) is 0.11538461538461542
Test Error Rate(k=15) is 0.11538461538461542
Test Error Rate(k=16) is 0.11538461538461542
Test Error Rate(k=17) is 0.12307692307692308
Test Error Rate(k=18) is 0.08461538461538465
Test Error Rate(k=19) is 0.10769230769230764
Test Error Rate(k=20) is 0.09999999999999998
```

When we consider only overall test error rate, it seems **k=8** perform the best on the data set.

Chapter 5

#1.

$$f(\alpha) = Var(\alpha X + (1-\alpha)Y) = \alpha^2 Var(X) + (1-\alpha)^2 Var(Y) + 2\alpha(1-\alpha)Cov(X,Y)$$

To find α that minimizes $f(\alpha)$, lets differentiate it with regard to α.

$$\frac{d}{d\alpha}f(\alpha) = 2\alpha Var(X) - 2(1-\alpha)Var(Y) + 2(1-2\alpha)Cov(X,Y) = 0$$

$$\therefore \alpha = \frac{2Var(Y)-2Cov(X,Y)}{2Var(Y)+2Var(Y)-4Cov(X,Y)} = \frac{\sigma_Y^2-\sigma_{XY}}{\sigma_X^2+\sigma_Y^2-2\sigma_{XY}} \quad (5.6)$$

Examining $f(x)$ near $x = \alpha$ tells that $f(x)$ is minimized at $x = \alpha$

#2.

(a) $1-\frac{1}{n}$

(b) As each bootstrap sample is a random sample, this probability is the same. $1-\frac{1}{n}$

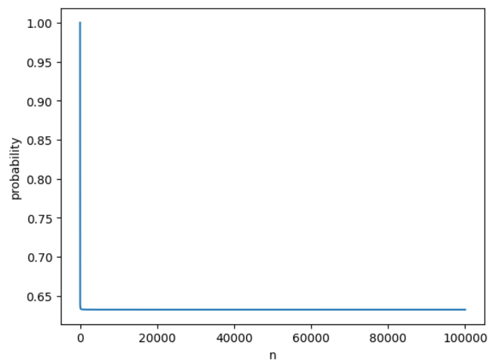
(c) As every sample is independent, it is $\left(1-\frac{1}{n}\right)^n$

(d) It is $1-\left(1-\frac{1}{5}\right)^5 = 0.67232$

(e) It is $1-\left(1-\frac{1}{100}\right)^{100} = 0.63397$

(f) It is $1-\left(1-\frac{1}{10000}\right)^{10000} = 0.63212$

(g) The probability likely to converge in $1-e^{-1}$ as j increases, since $\lim_{j\to\infty}\left(1-\frac{1}{j}\right)^j = \frac{1}{e}$.



(h) 0.6398639863986398, result from bootstrapping resembles theoretical probability.

```python
store = []
for i in np.arange(1, 10000):
    store += [np.sum((np.random.randint(low=1, high=101, size=100) == 4)) > 0]

np.mean(store)
```

0.6398639863986398

#5.

(a)

```
[157] x = np.array(df[['balance', 'income']])
      y = np.array(df.default)
```

```
[159] logit = LogisticRegression()
      logit.fit(x, y)

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

```
[-11.54047812] [[5.64710797e-03 2.08091985e-05]]
```

(b)

i.

```
[118] # i.
      x_train, x_test, y_train, y_test = train_test_split(
          x, y, test_size=0.33, random_state=1
      )
      print(len(x_train))
      print(len(x_test))
      print(len(y_train))
      print(len(y_test))
```

```
6700
3300
6700
3300
```

ii.

```
[161] # ii.
      logit = LogisticRegression()
      logit.fit(x_train, y_train)

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

```
[-1.18250162e-06] [[ 0.00039328 -0.00012161]]
```

iii.

```
[121] # iii.

      probs = logit.predict_proba(x_test)
      predict = (probs[:, 1] > 0.5).astype(int)
      result = ['Yes' if item==1 else 'No' for item in predict]
```

```
[122] display(confusion_table(confusion_matrix(y_test, result)))
```

	y_pred=0	y_pred=1	Total
y=0	3200	0	3200
y=1	100	0	100
Total	3300	0	

iv.

Validation set overall error rate is 100/3300 = 0.030303

(c)

random_state: 2

	y_pred=0	y_pred=1	Total
y=0	3212	2	3214
y=1	86	0	86
Total	3298	2	

random_state: 3

	y_pred=0	y_pred=1	Total
y=0	3179	8	3187
y=1	74	39	113
Total	3253	47	

random_state: 4

	y_pred=0	y_pred=1	Total
y=0	3180	12	3192
y=1	69	39	108
Total	3249	51	

[Validation Set Overall Error Rate]

- random_state (2) => (86+2)/3300 = 0.026667

- random_state (3) => (74+8)/3300 = 0.024848

- random_state (4) => (69+12)/3300 = 0.024545

The results obtained are variable and depend on the samples allocated to training vs. test.

(d)

random_state: 2

	Pred: No	Pred: Yes	Total
No	3212	2	3214
Yes	86	0	86
Total	3298	2	

random_state: 3

	Pred: No	Pred: Yes	Total
No	3173	14	3187
Yes	91	22	113
Total	3264	36	

random_state: 4

	Pred: No	Pred: Yes	Total
No	3176	16	3192
Yes	87	21	108
Total	3263	37	

[Validation Set Overall Error Rate (Including 'Student')]

- random_state (2) => (86+2)/3300 = 0.026667
- random_state (3) => (91+14)/3300 = 0.031818
- random_state (4) => (87+16)/3300 = 0.031212

Including student does not seem to make a substantial improvement to the test error.

#6.

(a)

```
smf.logit('default ~ income + balance', data=temp_df).fit().summary()
```

```
Optimization terminated successfully.
         Current function value: 0.078948
         Iterations 10
```

Logit Regression Results

Dep. Variable:	default	**No. Observations:**	10000
Model:	Logit	**Df Residuals:**	9997
Method:	MLE	**Df Model:**	2
Date:	Fri, 29 Sep 2023	**Pseudo R-squ.:**	0.4594
Time:	08:34:34	**Log-Likelihood:**	-789.48
converged:	True	**LL-Null:**	-1460.3
Covariance Type:	nonrobust	**LLR p-value:**	4.541e-292

	coef	**std err**	**z**	**P>|z|**	**[0.025**	**0.975]**
Intercept	-11.5405	0.435	-26.544	0.000	-12.393	-10.688
income	2.081e-05	4.99e-06	4.174	0.000	1.1e-05	3.06e-05
balance	0.0056	0.000	24.835	0.000	0.005	0.006

Possibly complete quasi-separation: A fraction 0.14 of observations can be perfectly predicted. This might indicate that there is complete quasi-separation. In this case some parameters will not be identified.

standard error of estimated coefficient(income): 4.99e-06

standard error of estimated coefficient(balance): 0.000

(b)

```
[298] def boot_fn(default):
          mod1 = smf.logit('default ~ income + balance', data=default).fit(maxiter=10000)
          coef_income = mod1.params[1]
          coef_balance = mod1.params[2]
          return [coef_income, coef_balance]
```

```
[299] boot_fn(temp_df)
```

```
Optimization terminated successfully.
         Current function value: 0.078948
         Iterations 10
[2.0808975528986992e-05, 0.005647102950316493]
```

(c)

```
[300] def bootstrap_sample(df, num_samples=None):
        n = df.shape[0]

        if num_samples == None:
          num_samples = df.shape[0]

        bootstrap_samples = np.random.choice(n, num_samples, replace=True)
        bootstrap_df = df.iloc[bootstrap_samples, :]
        return bootstrap_df
```

```
[301] coef_list = []
      for _ in tqdm(range(100)):
        try:
          sample_coefs = boot_fn(bootstrap_sample(temp_df))
          coef_list.append(sample_coefs)
        except Exception as e:
          continue
```

(d)

```
[302] pd.DataFrame(coef_list, columns=['income', 'balance']).describe()
```

	income	balance
count	100.000000	100.000000
mean	0.000020	0.005652
std	0.000004	0.000225
min	0.000010	0.005201
25%	0.000018	0.005506
50%	0.000021	0.005649
75%	0.000023	0.005800
max	0.000031	0.006193

From (a), estimated coefficients and standard errors of these coefficients for logistic regression of income and balance was [2.081e-05, 0.0056], and [4.99e-06, 0.000] respectively. According to the bootstrapping method, we obtained coefficients [0.000020, 0.005639], standard errors [0.000006, 0.000228] for these coefficients. From this result, one can conclude that result from bootstrapping resembles well with the original one.

#7.

(a)

```
[308] # make dummy variables

      temp_df = df.copy()
      temp_df['Direction_Up'] = temp_df['Direction'].apply(lambda x: 1 if x=='Up' else 0)
```

```
[309] logit = LogisticRegression()
      logit.fit(np.array(temp_df[['Lag1', 'Lag2']]), np.array(temp_df['Direction_Up']))
```

```
▾ LogisticRegression
LogisticRegression()
```

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

[0.22122502] [[-0.03869814  0.06020749]]
```

estimated coefficient (Lag1): -0.03869814

estimated coefficient (Lag2): 0.06020749

(b)

```
[311] logit = LogisticRegression()

      # predicts Direction Using Lag1 and Lag2 using all but the first observation
      x_train = np.array(temp_df.loc[1:, ['Lag1', 'Lag2']])
      y_train = np.array(temp_df.loc[1:, 'Direction'])

      x_test = np.array(temp_df.loc[0, ['Lag1', 'Lag2']])
      y_test = np.array(temp_df.loc[0, 'Direction'])

      logit.fit(x_train, y_train)
      print(logit.intercept_, logit.coef_)

      [0.22324404] [[-0.03840931  0.06080633]]
```

estimated coefficient (Lag1): -0.03840931

estimated coefficient (Lag2): 0.06080633

(c)

```
[335] y_test
```

```
array('Down', dtype='<U4')
```

```
prob = logit.predict_proba(x_test.reshape(-1, 2))
predict = (prob[:, 1] > 0.5)
print(('Up' if predict[0] == True else 'Down'))
```

```
Up
```

It isn't correctly classified.

(d)

```
[348] loocv = LeaveOneOut()
      real_y_list = []
      pred_y_list = []
      errors = np.zeros(temp_df.shape[0])

      for i, (train_index, test_index) in enumerate(loocv.split(np.array(temp_df))):
        # i.
        train_data = temp_df.iloc[train_index, :]
        test_datum = temp_df.iloc[test_index, :]

        x_train = np.array(train_data[['Lag1', 'Lag2']])
        y_train = np.array(train_data['Direction'])
        x_test = np.array(test_datum[['Lag1', 'Lag2']])
        y_test = np.array(test_datum['Direction'])

        logit = LogisticRegression()
        logit.fit(x_train, y_train)
        # ii.
        prob = logit.predict_proba(x_test)

        # iii.
        predict = (prob[:, 1] > 0.5).astype(int)
        predict = ['Up' if item==1 else 'Down' for item in predict]

        # iv.
        if (predict[0] != y_test.reshape(-1, 1)[0][0]):
          errors[i] = 1

        real_y_list.append(y_test[0])
        pred_y_list.append(predict[0])
```

```
[349] display(confusion_table(confusion_matrix(real_y_list, pred_y_list)))
```

	Pred: Down	Pred: Up	Total
Down	34	450	484
Up	40	565	605
Total	74	1015	

(e)

```
np.mean(errors)
```

```
0.44995408631772266
```

The LOOCV test error rate is 45% which implies that our predictions are marginally more often correct than not.