lab4

November 28, 2023

0.1 Lab 4: Classification

Jack Krebsbach Math 313

Imports

[ ]:
```
import numpy as np
import pandas as pd
from matplotlib.pyplot import subplots
import statsmodels.api as sm
from ISLP import load_data
from ISLP.models import (ModelSpec as MS,
summarize)
```

New imports needed

[ ]:
```
from ISLP import confusion_table
from ISLP.models import contrast
from sklearn.discriminant_analysis import \
(LinearDiscriminantAnalysis as LDA, QuadraticDiscriminantAnalysis as QDA)
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
```

[ ]:
```
# Load the data
Smarket = load_data('Smarket')
Smarket
```

[ ]:

	Year	Lag1	Lag2	Lag3	Lag4	Lag5	Volume	Today	Direction
0	2001	0.381	-0.192	-2.624	-1.055	5.010	1.19130	0.959	Up
1	2001	0.959	0.381	-0.192	-2.624	-1.055	1.29650	1.032	Up
2	2001	1.032	0.959	0.381	-0.192	-2.624	1.41120	-0.623	Down
3	2001	-0.623	1.032	0.959	0.381	-0.192	1.27600	0.614	Up
4	2001	0.614	-0.623	1.032	0.959	0.381	1.20570	0.213	Up
…	…	…	…	…	…	…	…	…	…
1245	2005	0.422	0.252	-0.024	-0.584	-0.285	1.88850	0.043	Up
1246	2005	0.043	0.422	0.252	-0.024	-0.584	1.28581	-0.955	Down

```
1247  2005 -0.955  0.043  0.422  0.252 -0.024  1.54047  0.130        Up
1248  2005  0.130 -0.955  0.043  0.422  0.252  1.42236 -0.298      Down
1249  2005 -0.298  0.130 -0.955  0.043  0.422  1.38254 -0.489      Down

[1250 rows x 9 columns]
```

[ ]:
```python
# Columns of data set
Smarket.columns
```

[ ]:
```
Index(['Year', 'Lag1', 'Lag2', 'Lag3', 'Lag4', 'Lag5', 'Volume', 'Today',
       'Direction'],
      dtype='object')
```

[ ]:
```python
Smarket.corr()
```

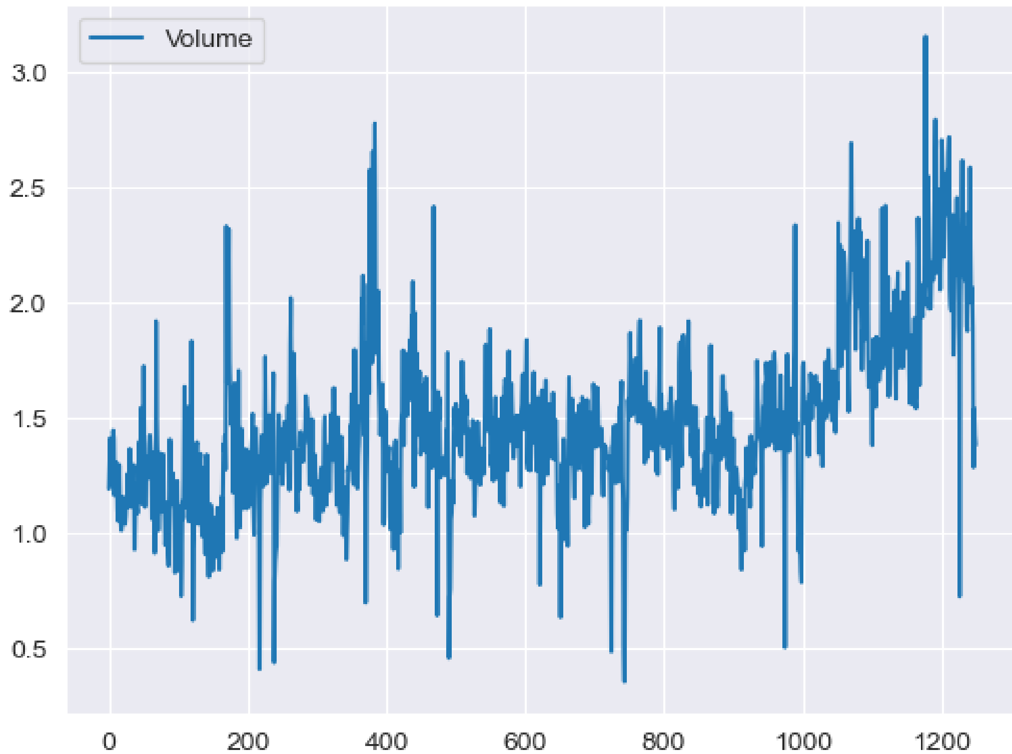
```
/var/folders/gf/bt25hkv172n_bttx0h72_6340000gn/T/ipykernel_12419/1422385858.py:1
: FutureWarning: The default value of numeric_only in DataFrame.corr is
deprecated. In a future version, it will default to False. Select only valid
columns or specify the value of numeric_only to silence this warning.
  Smarket.corr()
```

[ ]:
```
            Year      Lag1      Lag2      Lag3      Lag4      Lag5    Volume  \
Year    1.000000  0.029700  0.030596  0.033195  0.035689  0.029788  0.539006
Lag1    0.029700  1.000000 -0.026294 -0.010803 -0.002986 -0.005675  0.040910
Lag2    0.030596 -0.026294  1.000000 -0.025897 -0.010854 -0.003558 -0.043383
Lag3    0.033195 -0.010803 -0.025897  1.000000 -0.024051 -0.018808 -0.041824
Lag4    0.035689 -0.002986 -0.010854 -0.024051  1.000000 -0.027084 -0.048414
Lag5    0.029788 -0.005675 -0.003558 -0.018808 -0.027084  1.000000 -0.022002
Volume  0.539006  0.040910 -0.043383 -0.041824 -0.048414 -0.022002  1.000000
Today   0.030095 -0.026155 -0.010250 -0.002448 -0.006900 -0.034860  0.014592

           Today
Year    0.030095
Lag1   -0.026155
Lag2   -0.010250
Lag3   -0.002448
Lag4   -0.006900
Lag5   -0.034860
Volume  0.014592
Today   1.000000
```

[ ]:
```python
# Volume is increasing over time.
Smarket.plot(y='Volume')
```

[ ]: <Axes: >



Logistic Regression We will fit a logistic regression models to predict `Direction` using `Lag1` through `Lag5`.

[ ]:
```
# To run LR we use family = sm.families.Binomial()
allvars = Smarket.columns.drop(['Today', 'Direction', 'Year'])
design = MS(allvars)
X = design.fit_transform(Smarket)
y = Smarket.Direction == 'Up'
glm = sm.GLM(y, X, family=sm.families.Binomial())
results = glm.fit()
summarize(results)
```

[ ]:

	coef	std err	z	P>|z|
intercept	-0.1260	0.241	-0.523	0.601
Lag1	-0.0731	0.050	-1.457	0.145
Lag2	-0.0423	0.050	-0.845	0.398
Lag3	0.0111	0.050	0.222	0.824
Lag4	0.0094	0.050	0.187	0.851
Lag5	0.0103	0.050	0.208	0.835
Volume	0.1354	0.158	0.855	0.392

The smallest p-value here is associated with Lag1, but it is still not enough to provide clear evidence of an association between `Lag1` and `Direction`. Because it is a negative coefficient that means a positive increase in market in the previous day suggests it is less likely to go up today.

[ ]:
```
# Grab the coefficients
results.params
```

[ ]:
```
intercept   -0.126000
Lag1        -0.073074
Lag2        -0.042301
Lag3         0.011085
Lag4         0.009359
Lag5         0.010313
Volume       0.135441
dtype: float64
```

[ ]:
```
# Grab the p-values
results.pvalues
```

[ ]:
```
intercept    0.600700
Lag1         0.145232
Lag2         0.398352
Lag3         0.824334
Lag4         0.851445
Lag5         0.834998
Volume       0.392404
dtype: float64
```

[ ]:
```
probs = results.predict()
probs [:10]
```

[ ]:
```
array([0.50708413, 0.48146788, 0.48113883, 0.51522236, 0.51078116,
       0.50695646, 0.49265087, 0.50922916, 0.51761353, 0.48883778])
```

To predict a binary response – up or down – we must first convert these probabilities to class labels based on if the probability is greater or less than 0.5.

[ ]:
```
# Create array of length 1250 with each element Down
labels = np.array(['Down']*1250)
labels[probs>0.5] = "Up"
```

[ ]:
```
confusion_table(labels, Smarket.Direction)
```

[ ]:
```
Truth      Down   Up
Predicted
Down        145  141
Up          457  507
```

[ ]:
```
(507+145)/1250, np.mean(labels == Smarket.Direction)
```

[ ]: (0.5216, 0.5216)

The training error rate here is 100 - 52.2 = 47.8, which is overly optimistic. To get a more clear idea of the true error rate we create a training and testing set.

```
[ ]: train = (Smarket.Year < 2005)
Smarket_train = Smarket.loc[train]
Smarket_test = Smarket.loc[~train]
Smarket_test.shape
```

[ ]: (252, 9)

```
[ ]: X_train, X_test = X.loc[train], X.loc[~train]
y_train, y_test = y.loc[train], y.loc[~train]
glm_train = sm.GLM(y_train, X_train , family=sm.families.Binomial())
results = glm_train.fit()
probs = results.predict(exog=X_test)
```

```
[ ]: D = Smarket.Direction
L_train, L_test = D.loc[train], D.loc[~train]
```

```
[ ]: # Create array for all observations
labels = np.array(['Down']*252)
# Classify with decision rule
labels[probs>0.5] = 'Up'
# Create confusion matrix
confusion_table(labels, L_test)
```

```
[ ]: Truth      Down  Up
     Predicted
     Down         77  97
     Up           34  44
```

```
[ ]: np.mean(labels == L_test), np.mean(labels != L_test)
```

[ ]: (0.4801587301587302, 0.5198412698412699)

- The diagonals of the decision matrix are the correct values.
- The test accuracy is ~48%
- The test error rate is ~52%

```
[ ]: # We build a model with just Lag1 and Lag2
model = MS(['Lag1', 'Lag2']).fit(Smarket)
X = model.transform(Smarket)
X_train, X_test = X.loc[train], X.loc[~train]
glm_train = sm.GLM(y_train, X_train , family=sm.families.Binomial())
results = glm_train.fit()
probs = results.predict(exog=X_test)
```

```
labels = np.array(['Down']*252)
labels[probs>0.5] = 'Up'
confusion_table(labels, L_test)
```

```
[ ]: Truth      Down   Up
     Predicted
     Down         35   35
     Up           76  106
```

```
[ ]: (35+106) /252 ,106/(106+76)
```

```
[ ]: (0.5595238095238095, 0.5824175824175825)
```

```
[ ]: newdata = pd.DataFrame({'Lag1':[1.2, 1.5], 'Lag2':[1.1, -0.8]});
```

```
[ ]: newX = model.transform(newdata)
     results.predict(newX)
```

```
[ ]: 0    0.479146
     1    0.496094
     dtype: float64
```

We see that when in this new model the overall accuracy goes up. This logistic regression model has a 58% accuracy rate predicting increases in the market.

Linear Discriminant Analysis We perform Linear Discriminant Analysis on the Smarket data.

```
[ ]: # Create instance of LDA
     lda = LDA(store_covariance=True)
```

```
[ ]: # Split the train ant test set
     X_train, X_test = [M.drop(columns=['intercept']) for M in [X_train, X_test]]
     lda.fit(X_train, L_train)
```

```
[ ]: LinearDiscriminantAnalysis(store_covariance=True)
```

We can extract the means of the two classes with the means attribute on the `lda` object.

```
[ ]: lda.means_
```

```
[ ]: array([[ 0.04279022,  0.03389409],
            [-0.03954635, -0.03132544]])
```

To insure we know which class is corresponding to which label with `lda.classes_`.

```
[ ]: lda.classes_
```

```
[ ]: array(['Down', 'Up'], dtype='<U4')
```

Extracting the priors yeilds

[ ]:
```
lda.priors_
```

[ ]:
```
array([0.49198397, 0.50801603])
```

[ ]:
```
# Make predictions on the test set
lda_pred = lda.predict(X_test)
```

[ ]:
```
# The linear discriminant vectors
lda.scalings_
```

[ ]:
```
array([[-0.64201904],
       [-0.51352928]])
```

[ ]:
```
# Make predictions
lda_pred = lda.predict(X_test)
```

[ ]:
```
# Create confusing matrix
confusion_table(lda_pred, L_test)
```

[ ]:
```
Truth      Down   Up
Predicted
Down         35   35
Up           76  106
```

[ ]:
```
lda_prob = lda.predict_proba(X_test)
np.all(
np.where(lda_prob[:,1] >= 0.5, 'Up','Down') == lda_pred )
```

[ ]:
```
True
```

We can also use the posterior probabilities with a 50% threshold to re-create the predictions from the fitted lda instance.

[ ]:
```
np.all(
[lda.classes_[i] for i in np.argmax(lda_prob, 1)] ==
lda_pred )
```

[ ]:
```
True
```

[ ]:
```
np.sum(lda_prob[:,0] > 0.9)
```

[ ]:
```
0
```

Interestingly, there are no days that meet the threshold of 90%!

Quadratic Discriminant Analysis Fit a QDA model on the Smarket data

[ ]:
```
# Instantiate the model
qda = QDA(store_covariance=True)
qda.fit(X_train, L_train)
```

[ ]:
```
QuadraticDiscriminantAnalysis(store_covariance=True)
```

[ ]:
```
# Get the means and priors
qda.means_, qda.priors_
```

[ ]:
```
(array([[ 0.04279022,  0.03389409],
        [-0.03954635, -0.03132544]]),
 array([0.49198397, 0.50801603]))
```

The QDA() function will compute each a covariance matrix for each class.

[ ]:
```
# Extract the covariance matrix for the first class.
qda.covariance_[0]
```

[ ]:
```
array([[ 1.50662277, -0.03924806],
       [-0.03924806,  1.53559498]])
```

Just like before we can make predictions on the test set using the trained classifier.

[ ]:
```
qda_pred = qda.predict(X_test)
confusion_table(qda_pred, L_test)
```

[ ]:
```
Truth      Down   Up
Predicted
Down         30   20
Up           81  121
```

[ ]:
```
np.mean(qda_pred == L_test)
```

[ ]:
```
0.5992063492063492
```

An accuracy this high for stock market data suggests that QDA has the possibility of capturing the true relationship more than the linear forms.

Naive Bayes

[ ]:
```
# Instantiate object
NB = GaussianNB()
NB.fit(X_train, L_train)
```

[ ]:
```
GaussianNB()
```

[ ]:
```
# Get the classes
NB.classes_
```

[ ]: array(['Down', 'Up'], dtype='<U4')

[ ]:
```
# Extract the prior probabilities
NB.class_prior_
```

[ ]: array([0.49198397, 0.50801603])

[ ]:
```
# These are the means for each fitted model by each class and feature
NB.theta_
```

[ ]:
```
array([[ 0.04279022,  0.03389409],
       [-0.03954635, -0.03132544]])
```

[ ]:
```
# Similarly here are the variances. 2 classes * 2 features= 4 variances
NB.var_
```

[ ]:
```
array([[1.50355429, 1.53246749],
       [1.51401364, 1.48732877]])
```

[ ]:
```
mean= X_train[L_train == 'Down'].mean()
variance = X_train[L_train == 'Down'].var(ddof=0)
print(f'Mean: {mean}')
print(f'Variance: {variance}')
```

```
Mean: Lag1    0.042790
Lag2    0.033894
dtype: float64
Variance: Lag1    1.503554
Lag2    1.532467
dtype: float64
```

[ ]:
```
# Extract confusion matrix
nb_labels = NB.predict(X_test)
confusion_table(nb_labels, L_test)
```

[ ]:
```
Truth      Down   Up
Predicted
Down         29   20
Up           82  121
```

[ ]:
```
# Predict on new data points
NB.predict_proba(X_test)[:5]
```

[ ]:
```
array([[0.4873288 , 0.5126712 ],
       [0.47623584, 0.52376416],
       [0.46529531, 0.53470469],
       [0.47484469, 0.52515531],
```

```
[0.49020587, 0.50979413]])
```

Using `NB.predict_proba` we can estimate the probability that each observation belongs to a particular class.

K-Nearest Neighbors We can also use K-Nearest Neighbors to create a classifier

```
[ ]: knn1 = KNeighborsClassifier(n_neighbors=1)
     knn1.fit(X_train, L_train)
     knn1_pred = knn1.predict(X_test)
     confusion_table(knn1_pred, L_test)
```

```
[ ]: Truth      Down  Up
     Predicted
     Down         43  58
     Up           68  83
```

```
[ ]: # Not a very good fit with 50 percent accuracy
     (83+43)/252, np.mean(knn1_pred == L_test)
```

```
[ ]: (0.5, 0.5)
```

Since we did not get a good accuracy with 2 nearest neighbors we can try with 3.

```
[ ]: knn3 = KNeighborsClassifier(n_neighbors=3)
     knn3_pred = knn3.fit(X_train, L_train).predict(X_test)
     np.mean(knn3_pred == L_test)
```

```
[ ]: 0.5317460317460317
```

We only did a little bit better, now with 53% accuracy. Thus, KNN does not do well on the `Smarket` data set. We can see its utility using the `Caravan` data set.

```
[ ]: Caravan = load_data('Caravan')
     Purchase = Caravan.Purchase
     Purchase.value_counts()
```

```
[ ]: No     5474
     Yes     348
     Name: Purchase, dtype: int64
```

```
[ ]: # Proportion of individuals that purchase a caravan insurance policy.
     348 / 5822
```

```
[ ]: 0.05977327378907592
```

```
[ ]: # Create the feature data frame
     feature_df = Caravan.drop(columns=['Purchase'])
```

[ ]:
```
# Create scaler
scaler = StandardScaler(with_mean=True, with_std=True,
copy=True)
```

When argument `with_mean` is True the means are subtracted off.

[ ]:
```
# Scale the data
scaler.fit(feature_df)
X_std = scaler.transform(feature_df)
```

[ ]:
```
feature_std = pd.DataFrame( X_std , columns=feature_df.columns);
feature_std.std()
```

[ ]:
```
MOSTYPE     1.000086
MAANTHUI    1.000086
MGEMOMV     1.000086
MGEMLEEF    1.000086
MOSHOOFD    1.000086
              …
AZEILPL     1.000086
APLEZIER    1.000086
AFIETS      1.000086
AINBOED     1.000086
ABYSTAND    1.000086
Length: 85, dtype: float64
```

[ ]:
```
(X_train, X_test , y_train , y_test) = train_test_split(feature_std, Purchase ,␣
↪test_size=1000, random_state=0)
```

[ ]:
```
# Instantiate classifier
knn1 = KNeighborsClassifier(n_neighbors=1)
# Fit the model
knn1_pred = knn1.fit(X_train, y_train).predict(X_test)
# Get the fitted values.
np.mean(y_test != knn1_pred), np.mean(y_test != "No")
```

[ ]:
```
(0.111, 0.067)
```

[ ]:
```
# Confusion matrix
confusion_table(knn1_pred, y_test)
```

[ ]:
```
Truth        No  Yes
Predicted
No          880   58
Yes          53    9
```

[ ]:
```
9/(53+9)
```

[ ]: 0.14516129032258066

The KNN model with K=1 does a good job at predicting customers that buy insurance. This is double what you would get if you were just guessing.

Tuning Parameters

[ ]:
```
for K in range(1,6):
    knn = KNeighborsClassifier(n_neighbors=K)
    knn_pred = knn.fit(X_train, y_train).predict(X_test)
    C = confusion_table(knn_pred, y_test)
    templ = ('K={0:d}: # predicted to rent: {1:>2},' +
' # who did rent {2:d}, accuracy {3:.1%}')
    pred = C.loc['Yes'].sum()
    did_rent = C.loc['Yes','Yes']
    print(templ.format(K, pred , did_rent ,did_rent / pred))
```

```
K=1: # predicted to rent: 62, # who did rent 9, accuracy 14.5%
K=2: # predicted to rent:  6, # who did rent 1, accuracy 16.7%
K=3: # predicted to rent: 20, # who did rent 3, accuracy 15.0%
K=4: # predicted to rent:  4, # who did rent 0, accuracy 0.0%
K=5: # predicted to rent:  7, # who did rent 1, accuracy 14.3%
```

Here we see that at K=4 the accuracy and predictions is very different than the rest.

Comparison to Logistic Regression

[ ]:
```
logit = LogisticRegression(C=1e10, solver='liblinear')
logit.fit(X_train, y_train)
logit_pred = logit.predict_proba(X_test)
logit_labels = np.where(logit_pred[:,1] > 5, 'Yes', 'No')
confusion_table(logit_labels, y_test)
```

[ ]:
```
Truth       No  Yes
Predicted
No         933   67
Yes          0    0
```

[ ]:
```
logit_labels = np.where(logit_pred[:,1]>0.25, 'Yes', 'No')
confusion_table(logit_labels, y_test)
```

[ ]:
```
Truth       No  Yes
Predicted
No         913   58
Yes         20    9
```

[ ]:
```
9/(20+9)
```

[ ]: 0.3103448275862069

Linear and Poisson Regression on Bikeshare Data

[ ]:
```
# Load the data
Bike = load_data('Bikeshare')
```

First we fit a linear regression model to the data.

[ ]:
```
X = MS(['mnth', 'hr',
'workingday', 'temp',
'weathersit']).fit_transform(Bike)
Y = Bike['bikers']
M_lm = sm.OLS(Y, X).fit()
summarize(M_lm)
```

[ ]:

	coef	std err	t	P>|t|
intercept	-68.6317	5.307	-12.932	0.000
mnth[Feb]	6.8452	4.287	1.597	0.110
mnth[March]	16.5514	4.301	3.848	0.000
mnth[April]	41.4249	4.972	8.331	0.000
mnth[May]	72.5571	5.641	12.862	0.000
mnth[June]	67.8187	6.544	10.364	0.000
mnth[July]	45.3245	7.081	6.401	0.000
mnth[Aug]	53.2430	6.640	8.019	0.000
mnth[Sept]	66.6783	5.925	11.254	0.000
mnth[Oct]	75.8343	4.950	15.319	0.000
mnth[Nov]	60.3100	4.610	13.083	0.000
mnth[Dec]	46.4577	4.271	10.878	0.000
hr[1]	-14.5793	5.699	-2.558	0.011
hr[2]	-21.5791	5.733	-3.764	0.000
hr[3]	-31.1408	5.778	-5.389	0.000
hr[4]	-36.9075	5.802	-6.361	0.000
hr[5]	-24.1355	5.737	-4.207	0.000
hr[6]	20.5997	5.704	3.612	0.000
hr[7]	120.0931	5.693	21.095	0.000
hr[8]	223.6619	5.690	39.310	0.000
hr[9]	120.5819	5.693	21.182	0.000
hr[10]	83.8013	5.705	14.689	0.000
hr[11]	105.4234	5.722	18.424	0.000
hr[12]	137.2837	5.740	23.916	0.000
hr[13]	136.0359	5.760	23.617	0.000
hr[14]	126.6361	5.776	21.923	0.000
hr[15]	132.0865	5.780	22.852	0.000
hr[16]	178.5206	5.772	30.927	0.000
hr[17]	296.2670	5.749	51.537	0.000
hr[18]	269.4409	5.736	46.976	0.000
hr[19]	186.2558	5.714	32.596	0.000
hr[20]	125.5492	5.704	22.012	0.000
hr[21]	87.5537	5.693	15.378	0.000

```
hr[22]                         59.1226    5.689  10.392  0.000
hr[23]                         26.8376    5.688   4.719  0.000
workingday                      1.2696    1.784   0.711  0.477
temp                          157.2094   10.261  15.321  0.000
weathersit[cloudy/misty]      -12.8903    1.964  -6.562  0.000
weathersit[heavy rain/snow] -109.7446   76.667  -1.431  0.152
weathersit[light rain/snow]  -66.4944    2.965 -22.425  0.000
```

We see that there are 24 levels and 40 observations.

[ ]:
```python
# We change the encoding of the variables hr and mnth
hr_encode = contrast('hr', 'sum')
mnth_encode = contrast('mnth', 'sum')
```

[ ]:
```python
X2 = MS([mnth_encode, hr_encode , 'workingday', 'temp', 'weathersit']).
 ↪fit_transform(Bike)
M2_lm = sm.OLS(Y, X2).fit()
S2 = summarize(M2_lm)
S2
```

[ ]:
```
                                coef  std err       t  P>|t|
intercept                    73.5974    5.132  14.340  0.000
mnth[Jan]                   -46.0871    4.085 -11.281  0.000
mnth[Feb]                   -39.2419    3.539 -11.088  0.000
mnth[March]                 -29.5357    3.155  -9.361  0.000
mnth[April]                  -4.6622    2.741  -1.701  0.089
mnth[May]                    26.4700    2.851   9.285  0.000
mnth[June]                   21.7317    3.465   6.272  0.000
mnth[July]                   -0.7626    3.908  -0.195  0.845
mnth[Aug]                     7.1560    3.535   2.024  0.043
mnth[Sept]                   20.5912    3.046   6.761  0.000
mnth[Oct]                    29.7472    2.700  11.019  0.000
mnth[Nov]                    14.2229    2.860   4.972  0.000
hr[0]                       -96.1420    3.955 -24.307  0.000
hr[1]                      -110.7213    3.966 -27.916  0.000
hr[2]                      -117.7212    4.016 -29.310  0.000
hr[3]                      -127.2828    4.081 -31.191  0.000
hr[4]                      -133.0495    4.117 -32.319  0.000
hr[5]                      -120.2775    4.037 -29.794  0.000
hr[6]                       -75.5424    3.992 -18.925  0.000
hr[7]                        23.9511    3.969   6.035  0.000
hr[8]                       127.5199    3.950  32.284  0.000
hr[9]                        24.4399    3.936   6.209  0.000
hr[10]                      -12.3407    3.936  -3.135  0.002
hr[11]                        9.2814    3.945   2.353  0.019
hr[12]                       41.1417    3.957  10.397  0.000
hr[13]                       39.8939    3.975  10.036  0.000
```

```
hr[14]                        30.4940    3.991   7.641  0.000
hr[15]                        35.9445    3.995   8.998  0.000
hr[16]                        82.3786    3.988  20.655  0.000
hr[17]                       200.1249    3.964  50.488  0.000
hr[18]                       173.2989    3.956  43.806  0.000
hr[19]                        90.1138    3.940  22.872  0.000
hr[20]                        29.4071    3.936   7.471  0.000
hr[21]                        -8.5883    3.933  -2.184  0.029
hr[22]                       -37.0194    3.934  -9.409  0.000
workingday                     1.2696    1.784   0.711  0.477
temp                         157.2094   10.261  15.321  0.000
weathersit[cloudy/misty]     -12.8903    1.964  -6.562  0.000
weathersit[heavy rain/snow] -109.7446   76.667  -1.431  0.152
weathersit[light rain/snow]  -66.4944    2.965 -22.425  0.000
```

Overall, we see that the choice of coding does not really matter, as along as we interpret the model output correctly.

[ ]:
```python
# The sum of the squared differences is zero
np.sum((M_lm.fittedvalues - M2_lm.fittedvalues)**2)
```

[ ]:
```
5.006155854534554e-20
```

[ ]:
```python
np.allclose(M_lm.fittedvalues, M2_lm.fittedvalues)
```

[ ]:
```
True
```

[ ]:
```python
# Extract the coefficients for month
coef_month = S2[S2.index.str.contains('mnth')]['coef']
coef_month
```

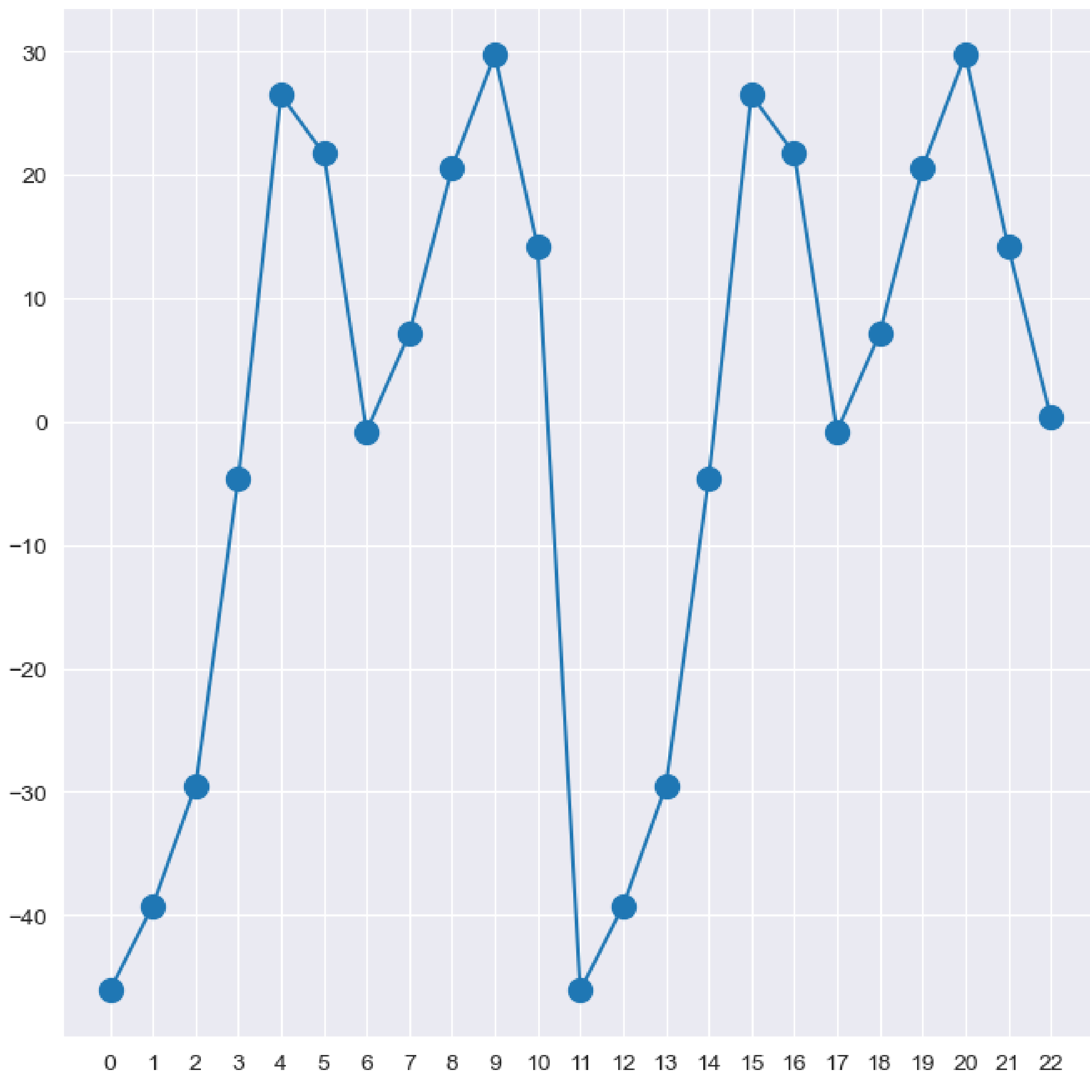
[ ]:
```
mnth[Jan]     -46.0871
mnth[Feb]     -39.2419
mnth[March]   -29.5357
mnth[April]    -4.6622
mnth[May]      26.4700
mnth[June]     21.7317
mnth[July]     -0.7626
mnth[Aug]       7.1560
mnth[Sept]     20.5912
mnth[Oct]      29.7472
mnth[Nov]      14.2229
Name: coef, dtype: float64
```

[ ]:
```python
# Append Dec as the negative of the sum of other months
months = Bike['mnth'].dtype.categories
coef_month = pd.concat([
```

```
coef_month, coef_month , pd.Series([-coef_month.sum()])
])
```

[ ]:
```
try:
    fig_month, ax_month = subplots(figsize=(8,8))
    x_month = np.arange(coef_month.shape[0])
    ax_month.plot(x_month, coef_month, marker='o', ms=10)
    ax_month.set_xticks(x_month)
    ax_month.set_xticklabels([l[5] for l in coef_month.index], fontsize
    =20)
    ax_month.set_xlabel('Month', fontsize=20)
    ax_month.set_ylabel('Coefficient', fontsize=20);
except Exception as e:
    print(e)
```

```
'int' object is not subscriptable
```



[ ]:
```
coef_hr = S2[S2.index.str.contains('hr')]['coef']
coef_hr = coef_hr.reindex(['hr[{0}]'.format(h) for h in range(23)])
coef_hr = pd.concat([coef_hr, pd.Series([-coef_hr.sum()], index=['hr[23]']) ])
```

[ ]:
```
fig_hr, ax_hr = subplots(figsize=(8,8))
x_hr = np.arange(coef_hr.shape[0])
ax_hr.plot(x_hr, coef_hr, marker='o', ms=10)
ax_hr.set_xticks(x_hr[::2])
ax_hr.set_xticklabels(range(24)[::2], fontsize=20)
ax_hr.set_xlabel('Hour', fontsize=20)
ax_hr.set_ylabel('Coefficient', fontsize=20);
```

Poisson Regression Instead of a linear regression we use poission regression, which is more suitable for this data.

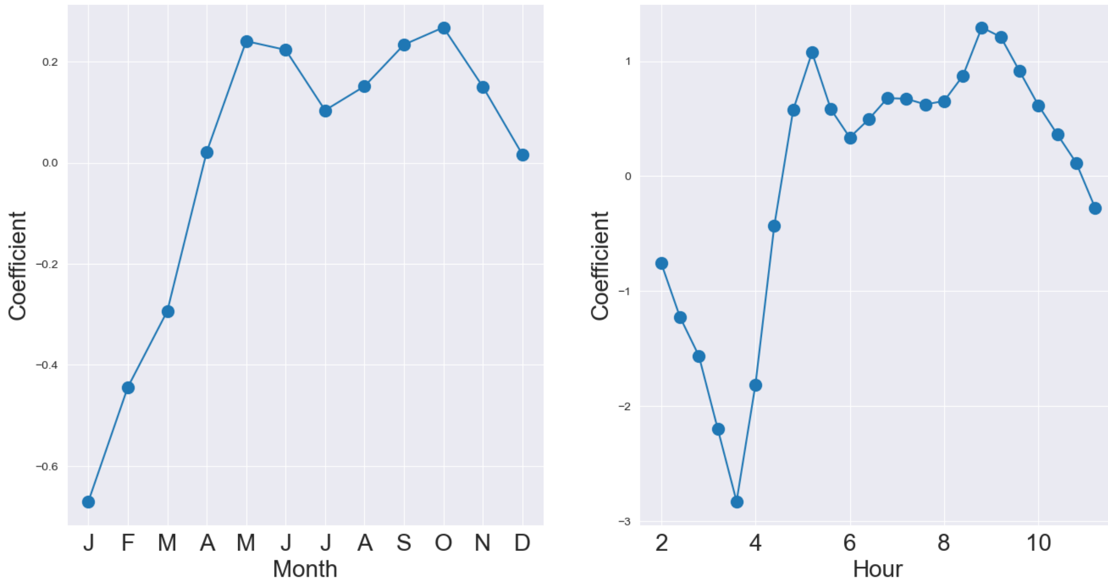
[ ]:
```
M_pois = sm.GLM(Y, X2, family=sm.families.Poisson()).fit()
```

[ ]:
```
S_pois = summarize(M_pois)
coef_month = S_pois[S_pois.index.str.contains('mnth')]['coef']
coef_month = pd.concat([coef_month,
pd.Series([-coef_month.sum()], index=['mnth[Dec]'])])
coef_hr = S_pois[S_pois.index.str.contains('hr')]['coef']
coef_hr = pd.concat([coef_hr, pd.Series([-coef_hr.sum()],
index=['hr[23]'])])
```

The following plots are of the coefficients of `mnth` and `hr`.

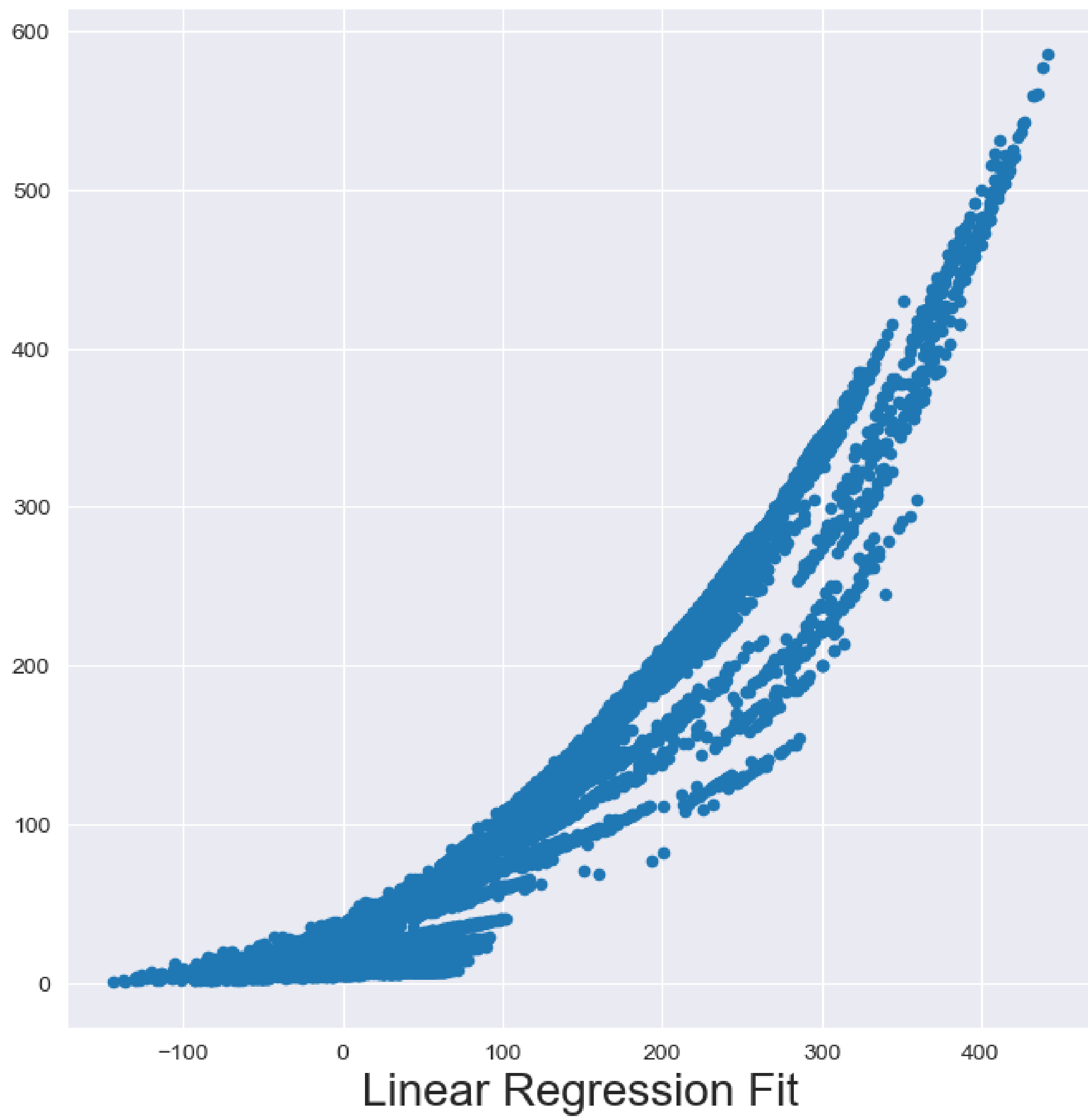
[ ]:
```
fig_pois, (ax_month, ax_hr) = subplots(1, 2, figsize=(16,8))
x_month = np.arange(coef_month.shape[0])
x_hr = np.arange(coef_hr.shape[0])
ax_month.plot(x_month, coef_month, marker='o', ms=10)
ax_month.set_xticks(x_month)
ax_month.set_xticklabels([l[5] for l in coef_month.index], fontsize
=20)
ax_month.set_xlabel('Month', fontsize=20)
ax_month.set_ylabel('Coefficient', fontsize=20)
ax_hr.plot(x_hr, coef_hr, marker='o', ms=10)
ax_hr.set_xticklabels(range(24)[::2], fontsize=20)
ax_hr.set_xlabel('Hour', fontsize=20)
ax_hr.set_ylabel('Coefficient', fontsize=20);
```

```
/var/folders/gf/bt25hkv172n_bttx0h72_6340000gn/T/ipykernel_12419/824882860.py:11
: UserWarning: FixedFormatter should only be used together with FixedLocator
  ax_hr.set_xticklabels(range(24)[::2], fontsize=20)
```



[ ]:
```
fig, ax = subplots(figsize=(8, 8))
ax.scatter(M2_lm.fittedvalues , M_pois.fittedvalues ,
s=20)
ax.set_xlabel('Linear Regression Fit', fontsize=20)
```

[ ]: Text(0.5, 0, 'Linear Regression Fit')



This shows that the prediction from the poisson regressions are correlated predictions from the linear model.