Applied: Exercise 13

Import notebook funcs

```
from notebookfuncs import *
```

Import libraries

```
from ISLP import load_data
from ISLP import confusion_table
from ISLP.models import (ModelSpec as MS , summarize)
from summarytools import dfSummary
import numpy as np
from scipy.stats import skew
from scipy.stats import boxcox
from scipy.optimize import curve_fit
import klib
import seaborn as sns
import matplotlib.pyplot as plt
import matplotlib.ticker as mtick
import statsmodels.api as sm
from statsmodels.tools.tools import add_constant
import pandas as pd
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.metrics import ConfusionMatrixDisplay
from sklearn.discriminant_analysis import (LinearDiscriminantAnalysis as LDA
↪   , QuadraticDiscriminantAnalysis as QDA)
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
```

Exercise 13

This question should be answered using the Weekly data set, which is part of the ISLP package. This data is similar in nature to the Smarket data from this chapter's lab, except that it contains 1,089 weekly returns for 21 years, from the beginning of 1990 to the end of 2010.

```
Weekly = load_data("Weekly")
Weekly["LogVolume"] = np.log(Weekly["Volume"])
Weekly = klib.convert_datatypes(Weekly)
print(Weekly.dtypes)
Weekly.head()
```

```
Year             int16
Lag1           float32
Lag2           float32
Lag3           float32
Lag4           float32
Lag5           float32
Volume         float32
Today          float32
Direction     category
LogVolume      float32
dtype: object
```

	Year	Lag1	Lag2	Lag3	Lag4	Lag5	Volume	Today	Direction	LogVolume
0	1990	0.816	1.572	-3.936	-0.229	-3.484	0.154976	-0.270	Down	-1.864485
1	1990	-0.270	0.816	1.572	-3.936	-0.229	0.148574	-2.576	Down	-1.906672
2	1990	-2.576	-0.270	0.816	1.572	-3.936	0.159837	3.514	Up	-1.833598
3	1990	3.514	-2.576	-0.270	0.816	1.572	0.161630	0.712	Up	-1.822446
4	1990	0.712	3.514	-2.576	-0.270	0.816	0.153728	1.178	Up	-1.872571

```
# Calculate skew for Volume, LogVolume, sqrt(Volume), sqrt4(Volume), Volume ^
↪ 2
print("Skew for Volume: ", skew(Weekly["Volume"], axis=0, bias=True))
print("Skew for LogVolume: ", skew(Weekly["LogVolume"], axis=0, bias=True))
print("Skew for Sqrt(Volume): ", skew(np.sqrt(Weekly["Volume"]), axis=0,
↪ bias=True))
print("Skew for Sqrt4(Volume): ", skew(np.sqrt(np.sqrt(Weekly["Volume"])),
↪ axis=0, bias=True))
print("Skew for Volume ** 2: ", skew(Weekly["Volume"] ** 2, axis=0,
↪ bias=True))
```

```
Skew for Volume:  1.6181865242606417
Skew for LogVolume:  0.05204035343976238
Skew for Sqrt(Volume):  0.8527756846731206
Skew for Sqrt4(Volume):  0.4482314175369111
Skew for Volume ** 2:  3.03324753476614
```

We transform column Volume to LogVolume since this is the most symmetrical among the transformations sqrt, sqrt4 and log (as evidenced by its low skew value).

Alternatively, we could use the Box-Cox series of transformations to convert Volume to a normally distributed variable.

```
Weekly["BoxCoxVolume"], lambda_vol = boxcox(Weekly["Volume"])
print("lambda: :",lambda_vol)
print("Skew for BoxCoxVolume: ", skew(Weekly["BoxCoxVolume"], axis=0,
 ↪  bias=True))
```

```
lambda: : -0.02702330888725645
Skew for BoxCoxVolume:  0.010558441792967627
```

Here, we see that the value of lambda_vol is almost zero and hence, the BoxCox transformation is the log transformation approximately.

```
Weekly.shape
```

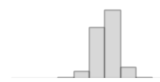
```
(1089, 11)
```

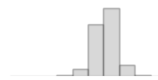
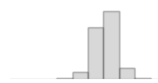
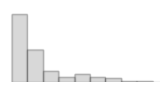
(a)

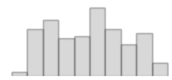
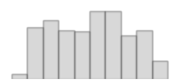
Produce some numerical and graphical summaries of the Weekly data. Do there appear to be any patterns?

```
dfSummary(Weekly)
```

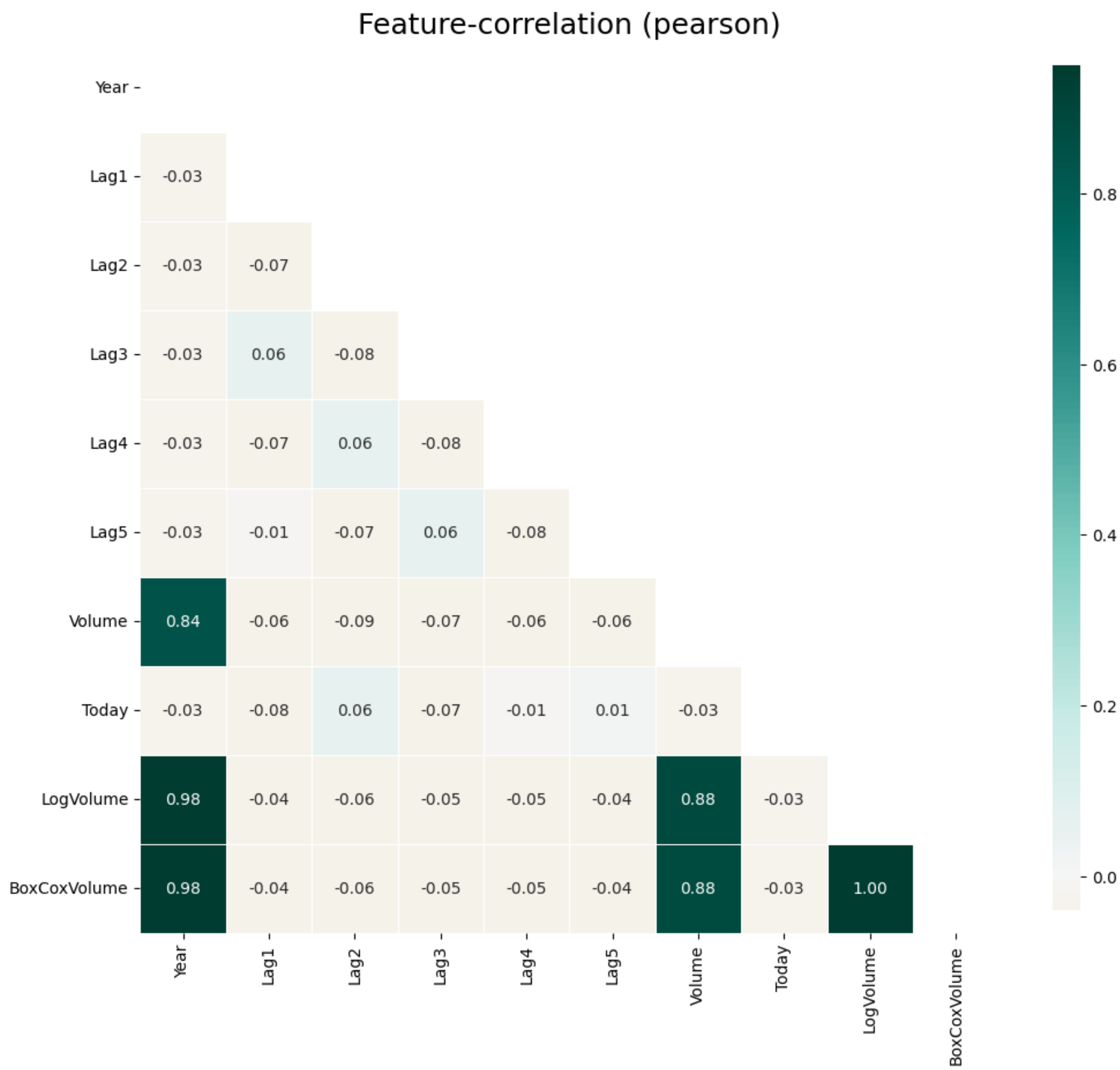
Table 2: **Data Frame Summary**
Weekly
Dimensions: 1,089 x 11
Duplicates: 0

No	Variable	Stats / Values	Freqs / (% of Valid)	Graph	Missing
1	**Year** [int16]	Mean (sd) : 2000.0 (6.0) min < med < max: 1990.0 < 2000.0 < 2010.0 IQR (CV) : 10.0 (331.5)	21 distinct values		0 (0.0%)
2	**Lag1** [float32]	Mean (sd) : 0.2 (2.4) min < med < max: -18.2 < 0.2 < 12.0 IQR (CV) : 2.6 (0.1)	1,004 distinct values		0 (0.0%)
3	**Lag2** [float32]	Mean (sd) : 0.2 (2.4) min < med < max: -18.2 < 0.2 < 12.0 IQR (CV) : 2.6 (0.1)	1,005 distinct values		0 (0.0%)

No	Variable	Stats / Values	Freqs / (% of Valid)	Graph	Missing
4	**Lag3** [float32]	Mean (sd) : 0.1 (2.4) min < med < max: -18.2 < 0.2 < 12.0 IQR (CV) : 2.6 (0.1)	1,005 distinct values		0 (0.0%)
5	**Lag4** [float32]	Mean (sd) : 0.1 (2.4) min < med < max: -18.2 < 0.2 < 12.0 IQR (CV) : 2.6 (0.1)	1,005 distinct values		0 (0.0%)
6	**Lag5** [float32]	Mean (sd) : 0.1 (2.4) min < med < max: -18.2 < 0.2 < 12.0 IQR (CV) : 2.6 (0.1)	1,005 distinct values		0 (0.0%)
7	**Volume** [float32]	Mean (sd) : 1.6 (1.7) min < med < max: 0.1 < 1.0 < 9.3 IQR (CV) : 1.7 (0.9)	1,089 distinct values		0 (0.0%)

No	Variable	Stats / Values	Freqs / (% of Valid)	Graph	Missing
8	**Today** [float32]	Mean (sd) : 0.1 (2.4) min < med < max: -18.2 < 0.2 < 12.0 IQR (CV) : 2.6 (0.1)	1,003 distinct values		0 (0.0%)
9	**Direction** [category]	1. Up 2. Down	605 (55.6%) 484 (44.4%)		0 (0.0%)
10	**LogVolume** [float32]	Mean (sd) : -0.1 (1.1) min < med < max: -2.4 < 0.0 < 2.2 IQR (CV) : 1.8 (-0.1)	1,089 distinct values		0 (0.0%)
11	**BoxCoxVolume** [float32]	Mean (sd) : -0.1 (1.1) min < med < max: -2.5 < 0.0 < 2.2 IQR (CV) : 1.8 (-0.1)	1,089 distinct values		0 (0.0%)

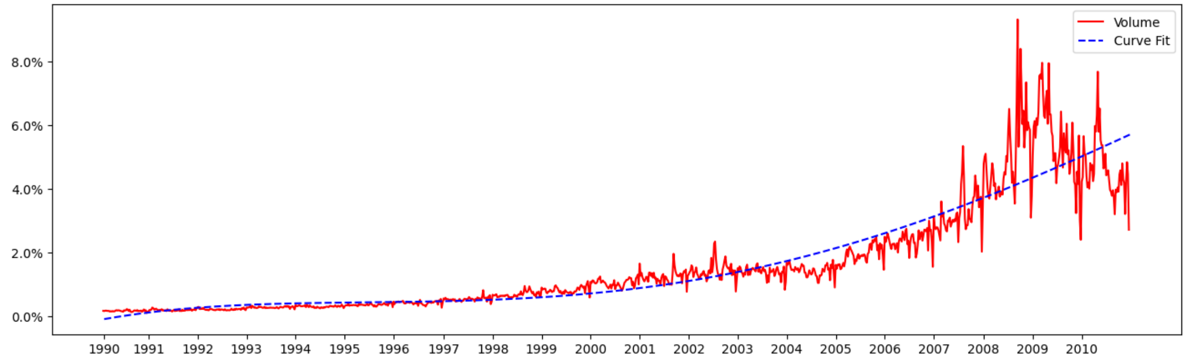
```
klib.corr_plot(Weekly);
```

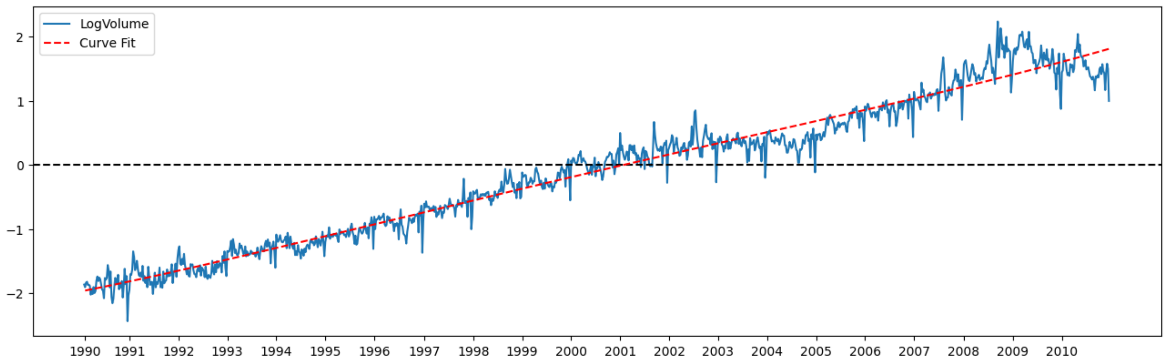


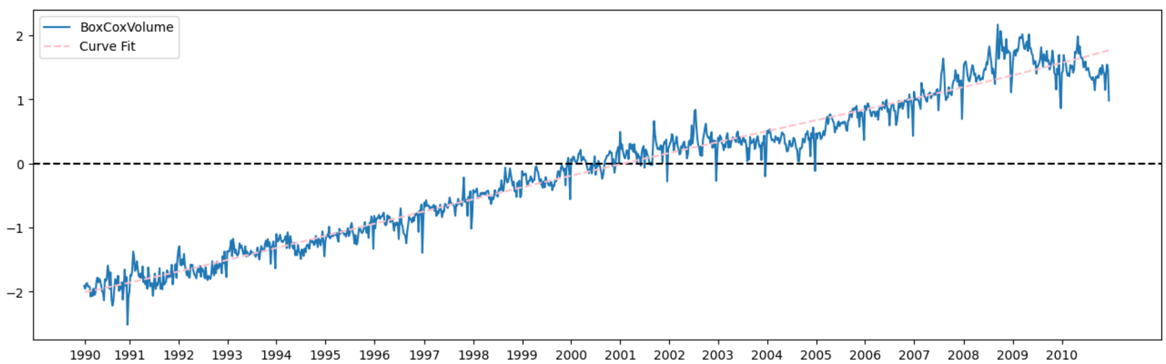
We can see that the correlation between Year and LogVolume is 0.98 which is much higher than the correlation between Year and Volume which is 0.84. That's because log transformation is non-linear and the original relation was non-linear as seen from the plot below. The same applies for BoxCoxVolume since it is approximately the log transformation of Volume.

```
Weekly["Week"] = np.arange(1, Weekly.shape[0] + 1)
Years_Break = Weekly.groupby("Year").first()
plt.figure(figsize=(16, 16))
plt.subplot(3,1,1)
plt.plot(Weekly["Volume"], label="Volume", c="r");
plt.xticks(ticks=Years_Break.Week,labels=Years_Break.index);
```

```
plt.gca().yaxis.set_major_formatter(mtick.PercentFormatter())

# objective function
def objective(x, a, b, c, d, e):
    return a * x + b * x ** 2 + c * x ** 3 + d * x ** 4 + e

# fit curve
popt, _ = curve_fit(objective, Weekly["Week"],Weekly["Volume"])
a, b, c, d, e = popt
y_new = objective(Weekly["Week"], a , b, c, d, e)
plt.plot(Weekly["Week"], y_new, "--", color="blue", label="Curve Fit")
plt.legend();

plt.subplot(3,1,2)
plt.plot(Weekly["LogVolume"], label="LogVolume");
plt.xticks(ticks=Years_Break.Week,labels=Years_Break.index);
plt.axhline(y=0, color="black", linestyle="--")
popt, _ = curve_fit(objective, Weekly["Week"],Weekly["LogVolume"])
a, b, c, d, e = popt
y_new = objective(Weekly["Week"], a , b, c, d, e)
plt.plot(Weekly["Week"], y_new, "--", color="red", label="Curve Fit")
plt.legend();

plt.subplot(3,1,3)
plt.plot(Weekly["BoxCoxVolume"], label="BoxCoxVolume");
plt.xticks(ticks=Years_Break.Week,labels=Years_Break.index);
plt.axhline(y=0, color="black", linestyle="--")
popt, _ = curve_fit(objective, Weekly["Week"],Weekly["BoxCoxVolume"])
a, b, c, d, e = popt
y_new = objective(Weekly["Week"], a , b, c, d, e)
plt.plot(Weekly["Week"], y_new, "--", color="pink", label="Curve Fit")
plt.legend();
```

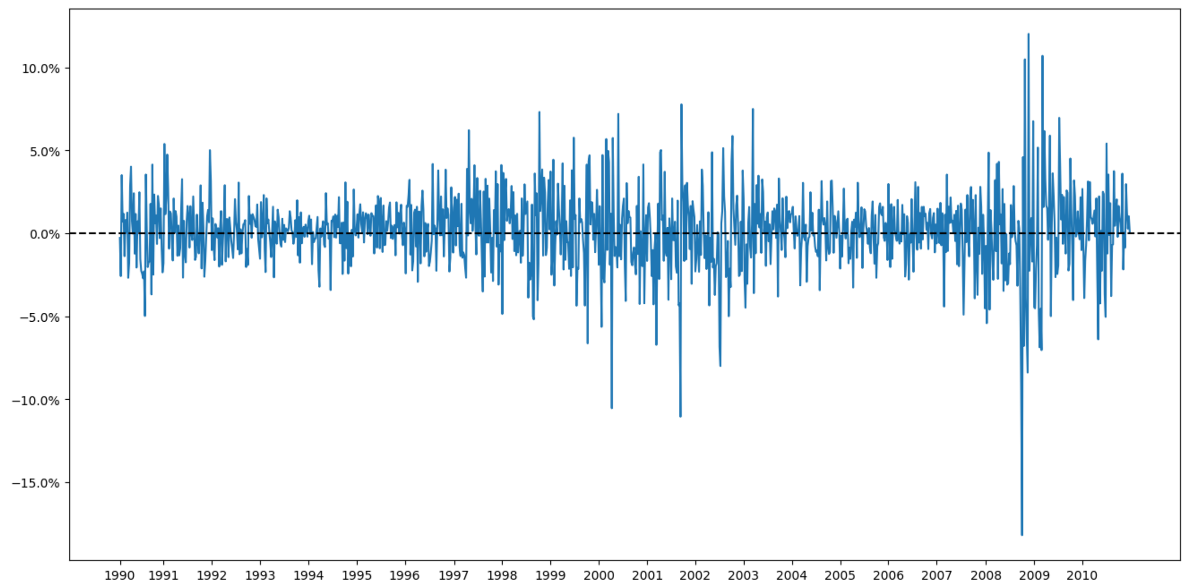






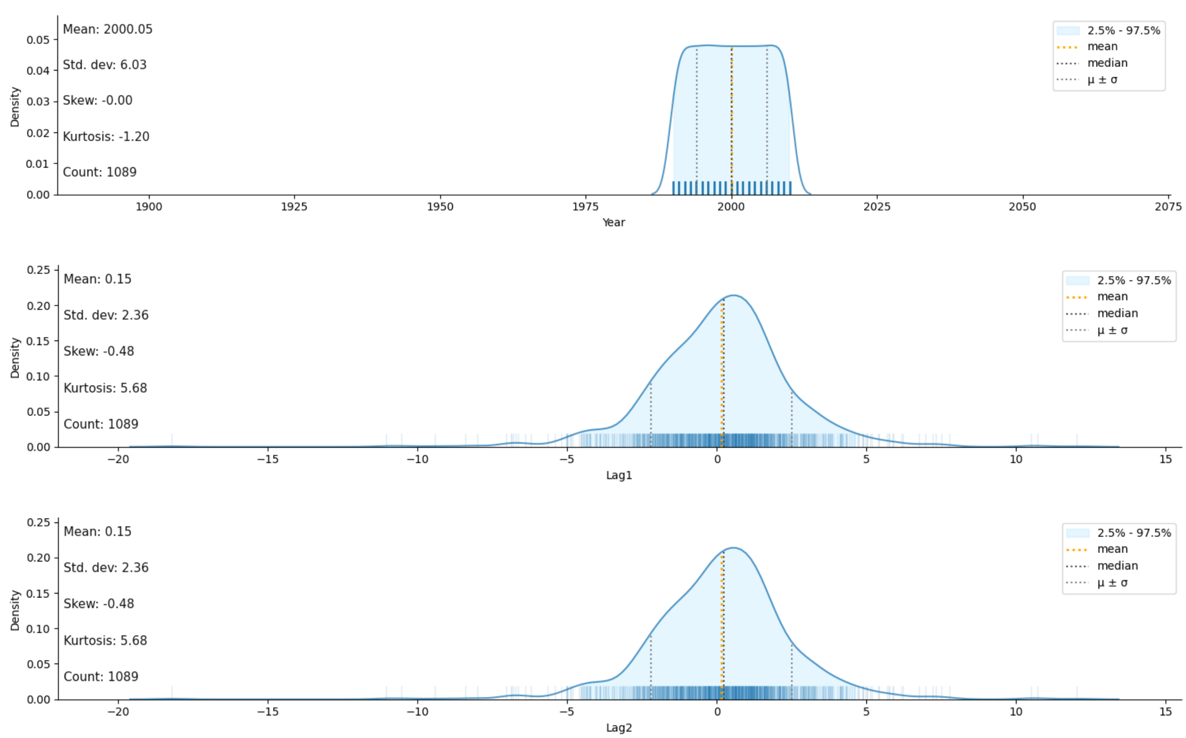
Here, we can see from the curve fit where we specified a cubic objective function, the Volume chart displays non-linearity but the LogVolume and BoxCoxVolume fits are straight lines.

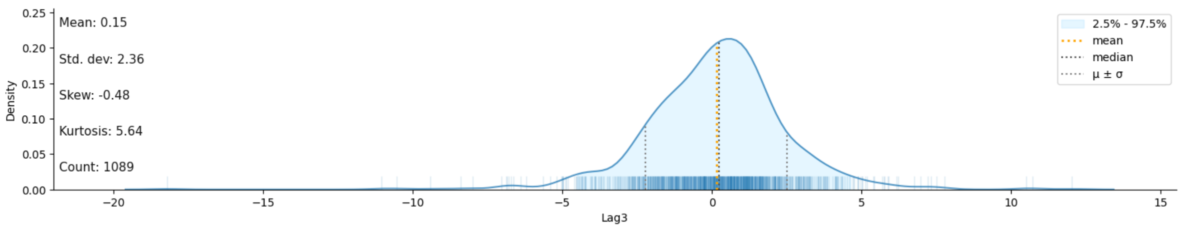
```
plt.figure(figsize=(16, 8))
plt.plot(Weekly["Week"],Weekly["Today"])
plt.xticks(ticks=Years_Break.Week,labels=Years_Break.index)
plt.gca().yaxis.set_major_formatter(mtick.PercentFormatter())
plt.axhline(y=0, color="k", linestyle="--");
```

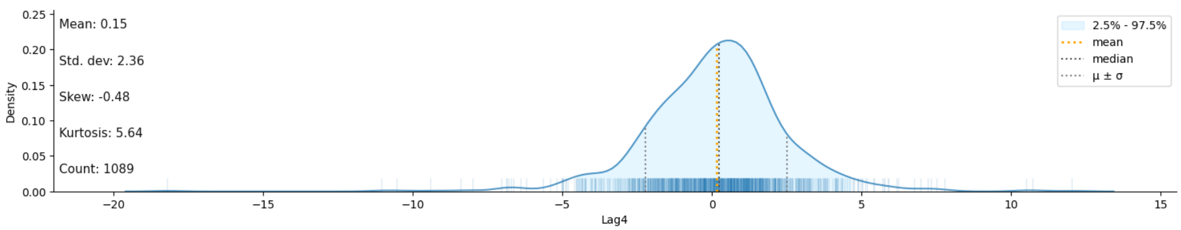


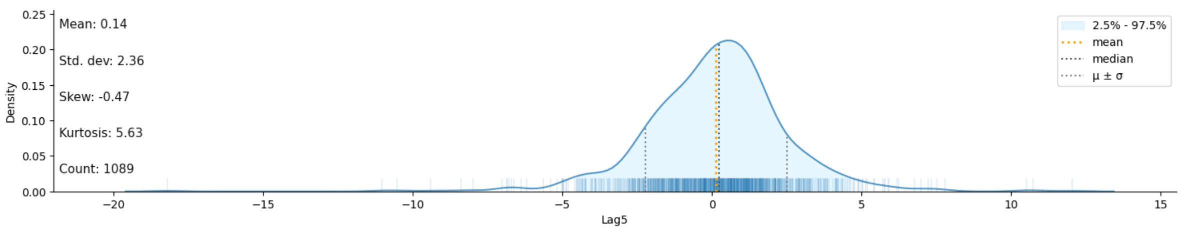
Here, we can see that the market go through periods of low and high volatility. Events such as market crashes exhibit high variance/volatility.

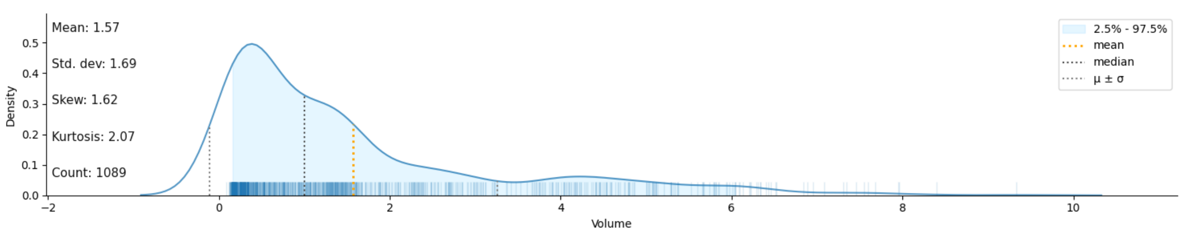
```
klib.dist_plot(Weekly);
```

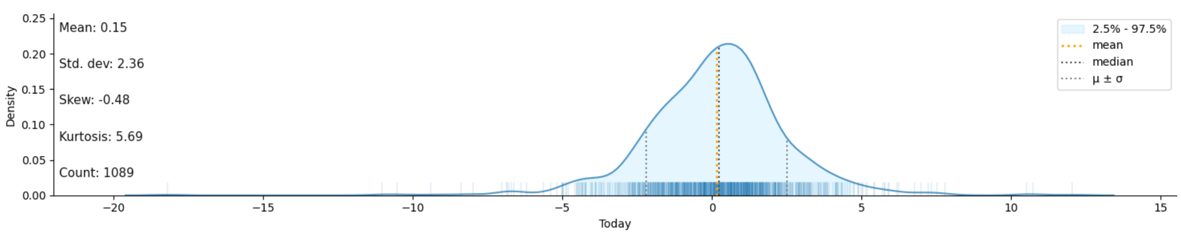


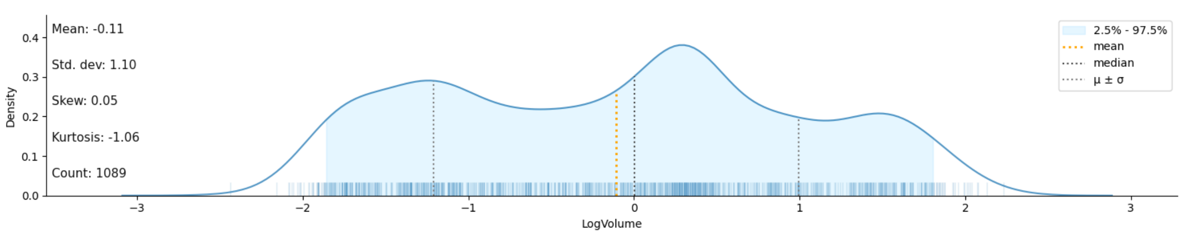

Mean: 0.15
Std. dev: 2.36
Skew: -0.48
Kurtosis: 5.64
Count: 1089
2.5% - 97.5%
mean
median
μ ± σ
Density
Lag3

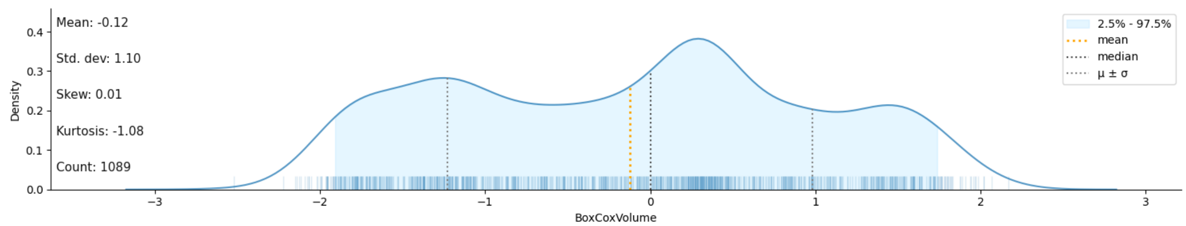

Mean: 0.15
Std. dev: 2.36
Skew: -0.48
Kurtosis: 5.64
Count: 1089
2.5% - 97.5%
mean
median
μ ± σ
Density
Lag4

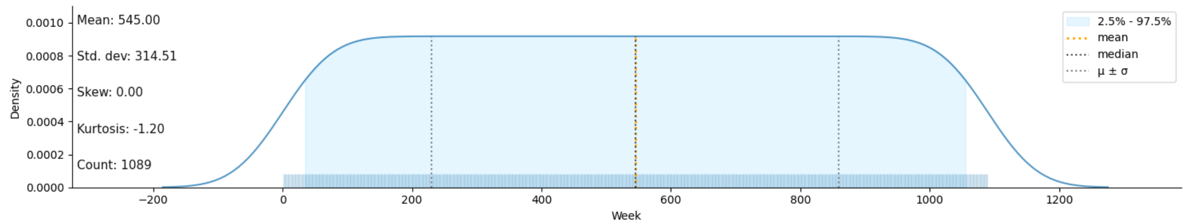

Mean: 0.14
Std. dev: 2.36
Skew: -0.47
Kurtosis: 5.63
Count: 1089
2.5% - 97.5%
mean
median
μ ± σ
Density
Lag5


Mean: 1.57
Std. dev: 1.69
Skew: 1.62
Kurtosis: 2.07
Count: 1089
2.5% - 97.5%
mean
median
μ ± σ
Density
Volume


Mean: 0.15
Std. dev: 2.36
Skew: -0.48
Kurtosis: 5.69
Count: 1089
2.5% - 97.5%
mean
median
μ ± σ
Density
Today


Mean: -0.11
Std. dev: 1.10
Skew: 0.05
Kurtosis: -1.06
Count: 1089
2.5% - 97.5%
mean
median
μ ± σ
Density
LogVolume





Skewness

Skewness is a measure of asymmetry or distortion of symmetric distribution. It measures the deviation of the given distribution of a random variable from a symmetric distribution, such as normal distribution. A normal distribution is without any skewness, as it is symmetrical on both sides.

Kurtosis

Negative kurtosis, also known as platykurtic, is a measure of a distribution's thin tails, meaning that outliers are infrequent:

Explanation

Kurtosis is a statistical measure that describes the shape of a distribution's tails in relation to its overall shape. It measures how often outliers occur, or the "tailedness" of the distribution.

Kurtosis types

A distribution with a kurtosis of 3 is considered mesokurtic, meaning it has a medium tail. A distribution with a kurtosis greater than 3 is leptokurtic, meaning it has a fat tail and a lot of outliers. A distribution with a kurtosis less than 3 is platykurtic, meaning it has a thin tail and infrequent outliers.

Kurtosis vs peakedness

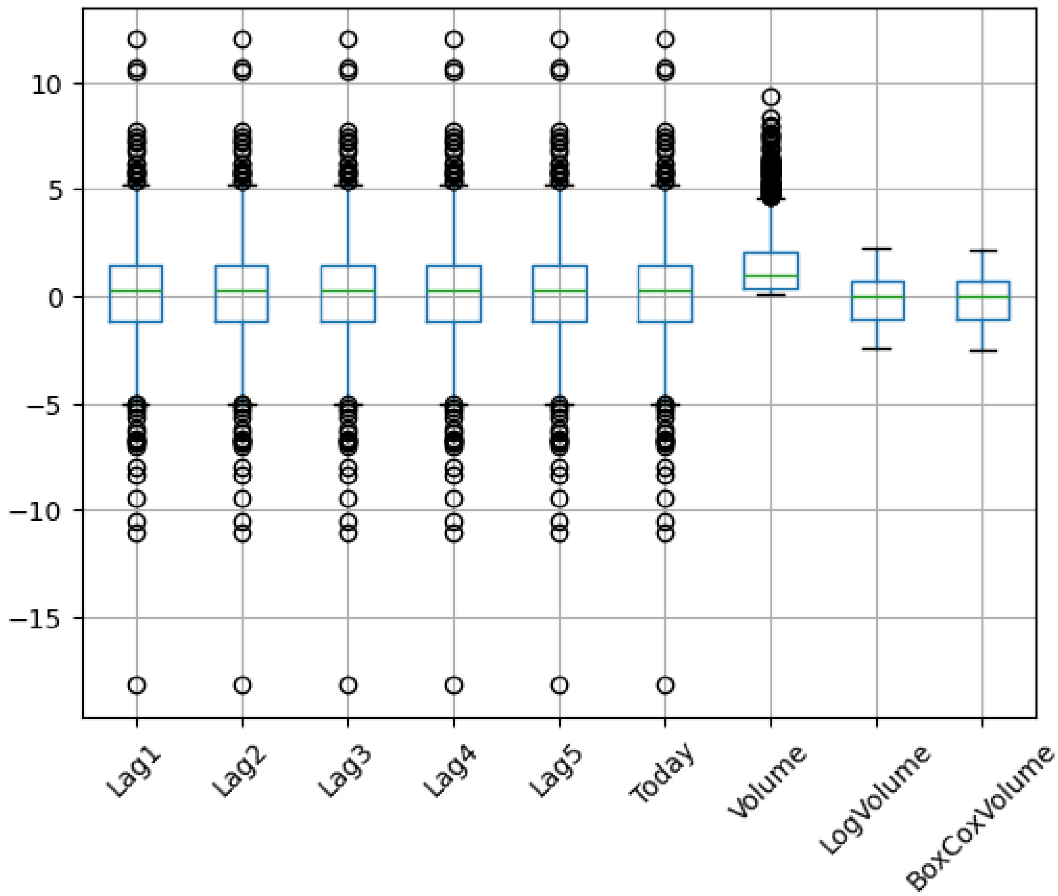
Kurtosis measures "tailedness," not "peakedness". A distribution can have a lower peak with high kurtosis, or a sharply peaked distribution with low kurtosis.

Calculating kurtosis

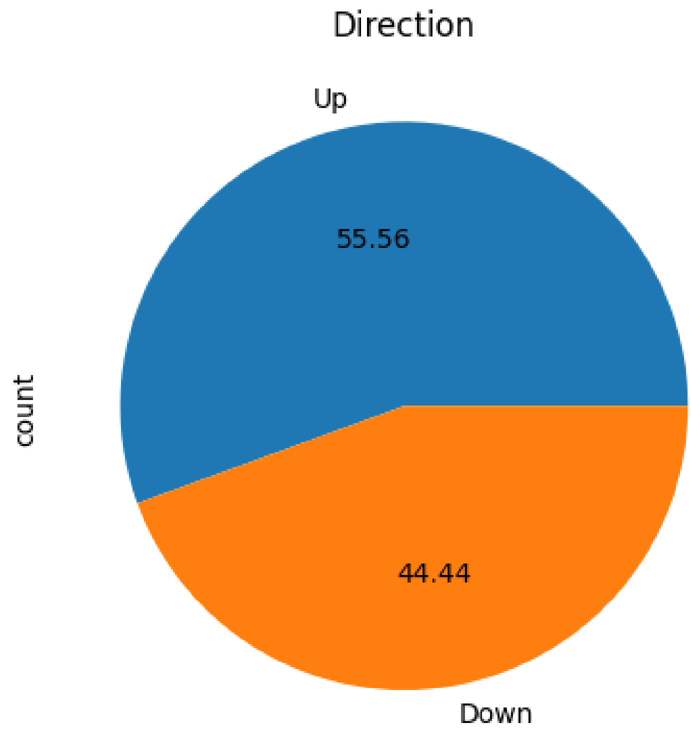
Kurtosis is calculated mathematically as the standardized fourth moment of a distribution.

We can verify the above conclusion from kurtosis definition by plotting the boxplots for the continuous variables, Lag1 - Lag5, Today and LogVolume.

```
Weekly.boxplot(column=["Lag1","Lag2","Lag3", "Lag4", "Lag5", "Today",
↪  "Volume" ,"LogVolume", "BoxCoxVolume"], rot=45);
```




```
Weekly["Direction"].value_counts().plot(kind="pie",autopct="%.2f",title="Direction");
```




```
nic_classifier_pct = Weekly["Direction"].value_counts()[0] / len(Weekly)
```

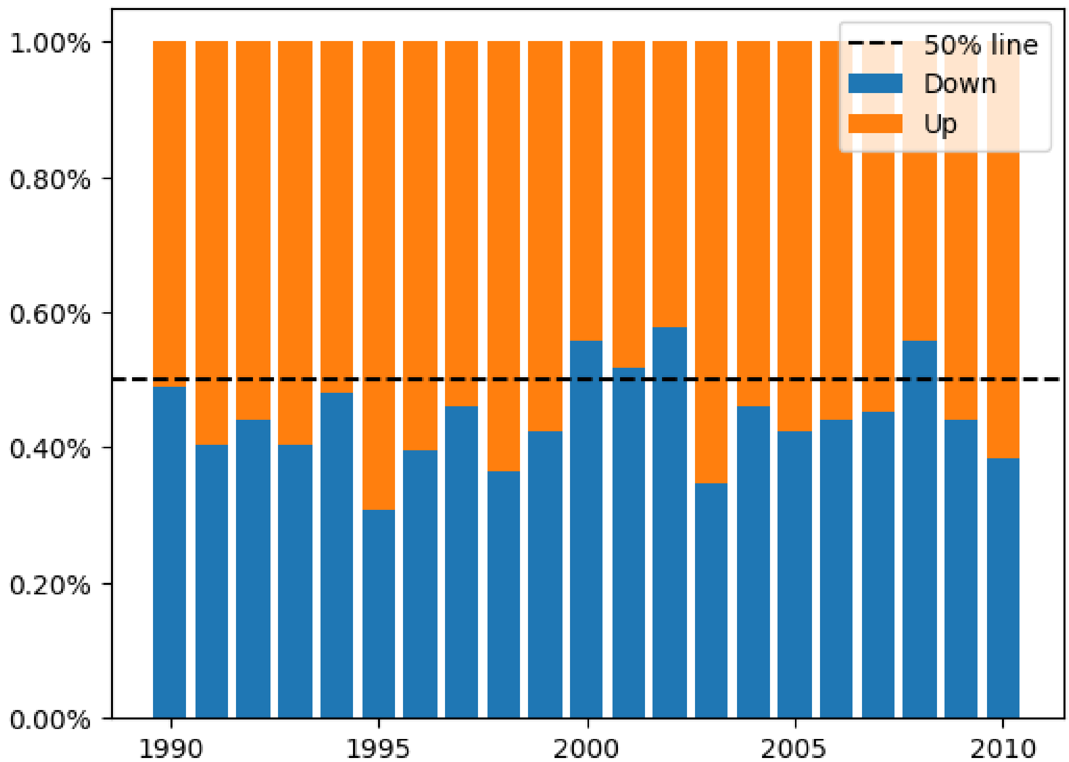
```
/tmp/ipykernel_35629/3312589537.py:1: FutureWarning: Series.__getitem__ treating keys as posi
  nic_classifier_pct = Weekly["Direction"].value_counts()[0] / len(Weekly)
```

```
0.5555555555555556
```

- Thus, we see from the pie-chart, that if we classify all responses as 'Up', we would still achieve an accuracy level of 55.56%. This is the base level which we have to improve upon.

```
bar_df = Weekly.groupby(["Year", "Direction"],
 ↪  observed=False).size().reset_index(name="Counts")
downs = bar_df[bar_df.Direction == "Down"].Counts.values
ups = bar_df[bar_df.Direction == "Up"].Counts.values
downs_pct = np.divide(downs,np.add(downs,ups))
ups_pct = np.divide(ups,np.add(downs,ups))
years = bar_df["Year"].unique();
```

```
fig, ax = plt.subplots()
ax.bar(years, downs_pct, label="Down")
ax.bar(years,ups_pct, bottom=downs_pct, label="Up");
ax.axhline(y=0.5, color="k", linestyle="--", label="50% line");
ax.yaxis.set_major_formatter(mtick.PercentFormatter())
ax.legend();
```



The bar-chart displays the percentage of Ups and Downs in a year from 1990 – 2010. The Ups dominate for most years except four.

(b)

Use the full data set to perform a logistic regression with Direction as the response and the five lag variables plus Volume as predictors. Use the summary function to print the results. Do any of the predictors appear to be statistically significant? If so, which ones?

```
# Try to avoid using ISLP classes for anything else but to load data since it
↪  may not transfer well to
# actual usage in data analysis projects
# drop columns Today, Direction, Year, Week , Volume, LogVolume
# We use BoxCoxVolume to regress against since it has the lowest skew value
↪  amongst all the transformations of the feature Volume
allvars = Weekly[Weekly.columns.difference(['Today', 'Direction', 'Year',
↪  "Week", "Volume", "LogVolume"])]
# add constant term of 1s
X = add_constant(allvars)
# Convert 'Down' and 'Up' to 0s and 1s respectively
y = Weekly.Direction == 'Up'
# Use Binomial family for Logistic Regression
family = sm.families.Binomial()
glm = sm.GLM(y, X, family=family)
results = glm.fit()
summarize(results)
```

	coef	std err	z	P>|z|
const	0.2247	0.062	3.606	0.000
BoxCoxVolume	-0.0515	0.056	-0.920	0.358
Lag1	-0.0413	0.026	-1.565	0.118
Lag2	0.0584	0.027	2.178	0.029
Lag3	-0.0161	0.027	-0.603	0.547
Lag4	-0.0279	0.026	-1.055	0.291
Lag5	-0.0146	0.026	-0.552	0.581

```
results.summary()
```

Dep. Variable:	Direction	**No. Observations:**	1089
Model:	GLM	**Df Residuals:**	1082
Model Family:	Binomial	**Df Model:**	6
Link Function:	Logit	**Scale:**	1.0000
Method:	IRLS	**Log-Likelihood:**	-742.94
Date:	Tue, 11 Feb 2025	**Deviance:**	1485.9
Time:	14:08:50	**Pearson chi2:**	1.09e+03
No. Iterations:	4	**Pseudo R-squ. (CS):**	0.009426
Covariance Type:	nonrobust		

	coef	**std err**	**z**	**P> |z|**	**[0.025**	**0.975]**
const	0.2247	0.062	3.606	0.000	0.103	0.347
BoxCoxVolume	-0.0515	0.056	-0.920	0.358	-0.161	0.058
Lag1	-0.0413	0.026	-1.565	0.118	-0.093	0.010
Lag2	0.0584	0.027	2.178	0.029	0.006	0.111
Lag3	-0.0161	0.027	-0.603	0.547	-0.068	0.036
Lag4	-0.0279	0.026	-1.055	0.291	-0.080	0.024
Lag5	-0.0146	0.026	-0.552	0.581	-0.066	0.037

```
results.model.endog_names
```

```
'Direction'
```

```
results.params
```

```
const           0.224750
BoxCoxVolume   -0.051454
Lag1           -0.041254
Lag2            0.058360
Lag3           -0.016059
Lag4           -0.027884
Lag5           -0.014559
dtype: float64
```

```
results.pvalues[results.pvalues < 0.05]
```

```
const    0.000311
Lag2     0.029378
dtype: float64
```

From the above, it can be deduced that `Lag2` is the only significant variable that predicts `Direction`. The positive coefficient for `Lag2` suggests that if the market had a positive return today, it is more likely that the market will rise once more in two days and vice versa. We can also see that the confidence intervals of the other parameters `Lag1, Lag3, Lag4, Lag5 and LogVolume` span the value 0 and thus are not significant.

(c)

Compute the confusion matrix and overall fraction of correct predictions. Explain what the confusion matrix is telling you about the types of mistakes made by logistic regression.

```
predictions = results.predict()
predictions.shape
```

```
(1089,)
```

```
labels = np.array(['Down']*len(Weekly))
print(labels.shape)
labels[predictions > 0.5] = "Up"
```

```
(1089,)
```

```
ct = confusion_table(Weekly["Direction"],
                     labels)
print(ct)
```

```
Truth      Down   Up
Predicted
Down         61  423
Up           52  553
```

The diagonal elements of the confusion matrix indicate correct predictions, while the off-diagonals represent incorrect predictions.

```python
correct_up_preds = ct["Up"]["Up"]
correct_down_preds = ct["Down"]["Down"]
print(f"Hence our model correctly predicted that the market would go up on
↪ {correct_up_preds} days and that it would go down on {correct_down_preds}
↪ days, for a total of {ct["Up"]["Up"]} + {ct["Down"]["Down"]} =
↪ {ct["Up"]["Up"] + ct["Down"]["Down"]} correct predictions. The np.mean()
↪ function can be used to compute the fraction of days for which the
↪ prediction was correct.")
correct_pred_perc = (ct["Up"]["Up"] + ct["Down"]["Down"]) * 100 /
↪ (ct["Up"]["Up"] + ct["Down"]["Down"] + ct["Down"]["Up"] +
↪ ct["Up"]["Down"])
print(f"In this case, logistic regression correctly predicted the movement of
↪ the market {correct_pred_perc:.1f}% of the time.")
```

```
Hence our model correctly predicted that the market would go up on 553 days and that it woul
In this case, logistic regression correctly predicted the movement of the market 56.4% of th
```

```python
accuracy = np.mean(labels == Weekly.Direction),
↪ (correct_up_preds+correct_down_preds) /len(Weekly)
```

```
(0.5638200183654729, 0.5638200183654729)
```

```python
print(f"This accuracy of {accuracy[0]*100:.2f}% is not much better than the
↪ no information classifier's (NIC) accuracy of {nic_classifier_pct
↪ *100:.2f}% when we just guess that the market will go up all the time and
↪ achieve an accuracy level of {nic_classifier_pct*100:.2f}%.")
```

```
This accuracy of 56.38% is not much better than the no information classifier's (NIC) accura
```

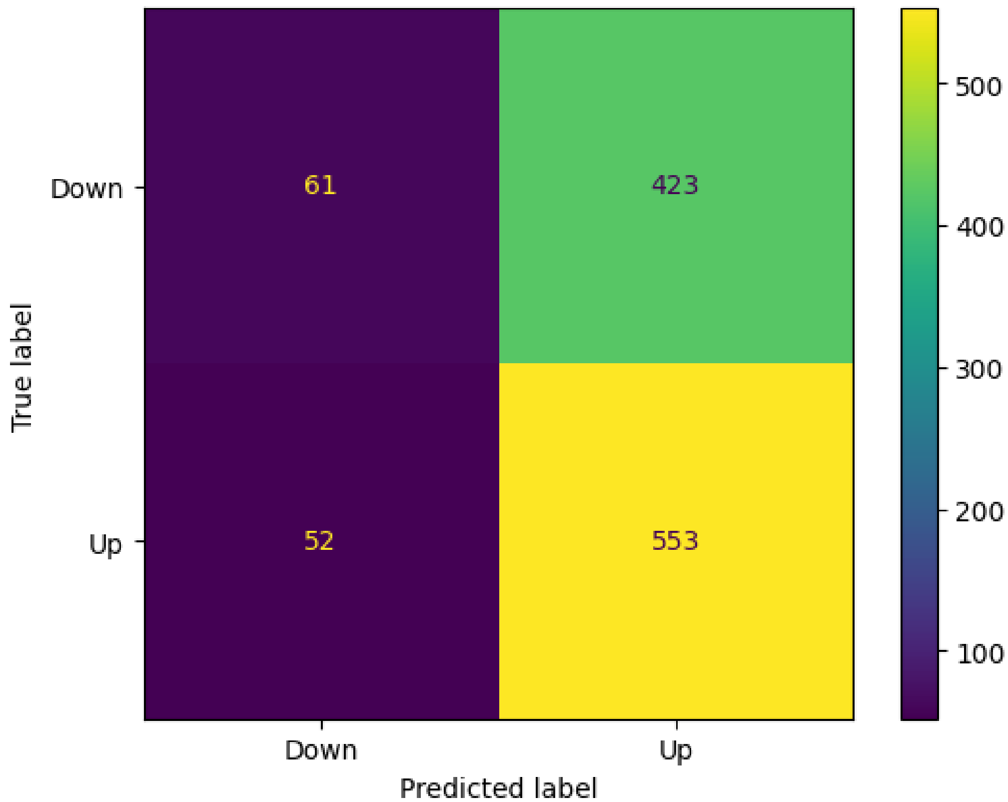
```python
print(f"100 - {accuracy[0]*100:.1f} = {100 - accuracy[0]*100:.1f}% is the
↪ training error rate.")
```

```
100 - 56.4 = 43.6% is the training error rate.
```

As we have seen previously, the training error rate is often overly optimistic — it tends to underestimate the test error rate. In order to better assess the accuracy of the logistic regression model in this setting, we can fit the model using part of the data, and then examine how well it predicts the held out data. This will yield a more realistic error rate, in the sense that in practice we will be interested in our model's performance not on the data that we used to fit the model, but rather on days in the future for which the market's movements are unknown.

```
print(classification_report(Weekly["Direction"],
                            labels,
                            digits = 3))
cm = confusion_matrix(Weekly["Direction"],
                      labels)

disp = ConfusionMatrixDisplay(confusion_matrix=cm,
                              display_labels=["Down", "Up"])
disp.plot();
```

```
              precision    recall  f1-score   support

        Down      0.540     0.126     0.204       484
          Up      0.567     0.914     0.700       605

    accuracy                          0.564      1089
   macro avg      0.553     0.520     0.452      1089
weighted avg      0.555     0.564     0.479      1089
```




```
# Getting individual values for
true_negatives, false_positives, false_negatives, true_positives = cm.ravel()
```

```
support_up = np.sum(Weekly["Direction"] == "Up")
support_down = Weekly.shape[0] - support_up
predicted_up = np.sum(labels == "Up")
predicted_down = np.sum(labels == "Down")
predicted_correctly_up = true_positives
precision_up = predicted_correctly_up / predicted_up
predicted_correctly_down = true_negatives
precision_down = predicted_correctly_down / predicted_down
print(f"Support (Up, Down): {support_up}, {support_down}")
print(f"Precision (Up, Down): {precision_up:.3f}, {precision_down:.3f}")
print(f"Precision Average (Macro, Weighted): {(precision_up +
↪  precision_down)/2:.3f}, {(precision_up * support_up + precision_down *
↪  support_down)/(support_up + support_down):.3f}")
recall_up = true_positives / support_up
recall_down = true_negatives / support_down
```

```
print(f"Recall (Up, Down): {recall_up:.3f}, {recall_down:.3f}")
print(f"Recall Average (Macro, Weighted): {(recall_up +
↪  recall_down)/2:.3f},{(recall_up * support_up + recall_down *
↪  support_down)/(support_up + support_down):.3f}")
f1_score_up = 2 * precision_up * recall_up/ (precision_up + recall_up)
f1_score_down = 2 * precision_down * recall_down/ (precision_down +
↪  recall_down)
print(f"F1 score (Up, Down): {f1_score_up:.3f}, {f1_score_down:.3f}")
print(f"F1 score Average (Macro, Weighted): {(f1_score_up +
↪  f1_score_down)/2:.3f},{(f1_score_up * support_up + f1_score_down *
↪  support_down)/(support_up + support_down):.3f}")
```

```
Support (Up, Down): 605, 484
Precision (Up, Down): 0.567, 0.540
Precision Average (Macro, Weighted): 0.553, 0.555
Recall (Up, Down): 0.914, 0.126
Recall Average (Macro, Weighted): 0.520,0.564
F1 score (Up, Down): 0.700, 0.204
F1 score Average (Macro, Weighted): 0.452,0.479
```

(d)

Now fit the logistic regression model using a training data period from 1990 to 2008, with Lag2 as the only predictor. Compute the confusion matrix and the overall fraction of correct predictions for the held out data (that is, the data from 2009 and 2010).

```
train = (Weekly.Year <= 2008)
Weekly_train = Weekly.loc[train]
Weekly_test = Weekly.loc[~train];
```

```
Weekly_train.shape
```

```
(985, 12)
```

```
Weekly_test.shape
```

```
(104, 12)
```

```python
X_train , X_test = Weekly_train["Lag2"], Weekly_test["Lag2"]
# add constant term of 1s
X_train = add_constant(X_train)
X_test = add_constant(X_test)
y_train , y_test = Weekly_train["Direction"] == "Up",
 ↪ Weekly_test["Direction"] == "Up"
glm_train = sm.GLM(y_train , X_train, family=sm.families.Binomial())
results = glm_train.fit()
probs = results.predict(exog=X_test);
```

```python
D = Weekly.Direction
L_train , L_test = D.loc[train], D.loc[~train]
```

```python
labels = np.array(['Down']*L_test.shape[0])
labels[probs > 0.5] = 'Up'
confusion_table(labels , L_test)
```

Truth Predicted	Down	Up
Down	9	5
Up	34	56

```python
accuracy = np.mean(labels == L_test)
```

```
0.625
```

```python
test_error = np.mean(labels != L_test)
```

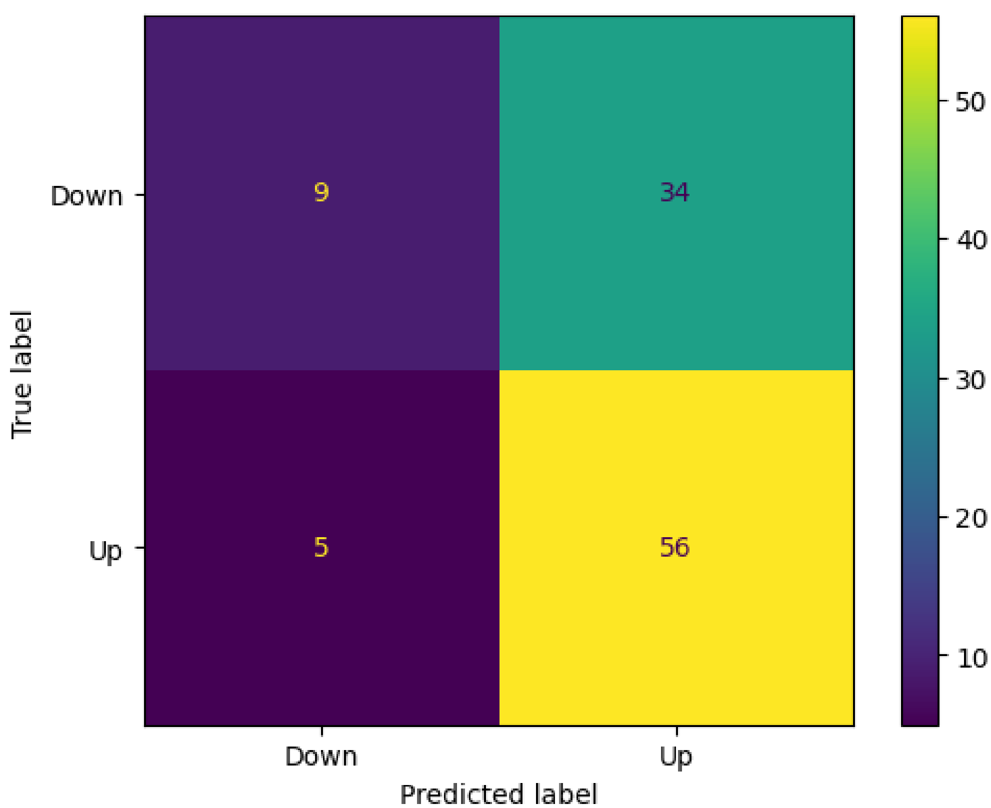
```
0.375
```

```python
print(f"Here we see the accuracy is {accuracy * 100:g}%.")
print(f"The test error is {test_error * 100:g}%.")
```

```
Here we see the accuracy is 62.5%.
The test error is 37.5%.
```

```
print(classification_report(Weekly_test["Direction"],
                            labels,
                            digits = 3, output_dict=False))
cm = confusion_matrix(Weekly_test["Direction"],
                      labels)
disp = ConfusionMatrixDisplay(confusion_matrix=cm,
                    display_labels=["Down", "Up"])
disp.plot();
```

```
              precision    recall  f1-score   support

        Down      0.643     0.209     0.316        43
          Up      0.622     0.918     0.742        61

    accuracy                          0.625       104
   macro avg      0.633     0.564     0.529       104
weighted avg      0.631     0.625     0.566       104
```



(e)

Repeat (d) using LDA.

```python
lda = LDA(store_covariance=True)
```

```
LinearDiscriminantAnalysis(store_covariance=True)
```

Since the LDA estimator automatically adds an intercept, we should remove the column corresponding to the intercept in both X_train and X_test. We can also directly use the labels rather than the Boolean vectors y_train.

```python
X_lda_train , X_lda_test = [M.drop(columns =['const']) for M in [X_train ,
↪  X_test ]]
```

```python
lda.fit(X_lda_train , L_train)
lda.means_
```

```
array([[-0.03568254],
       [ 0.2603659 ]], dtype=float32)
```

The above means indicate that when `Lag2` is negative, the market direction is Down two days later and vice versa.

The estimated prior probabilities are stored in the priors_ attribute. The package `sklearn` typically uses this trailing _ to denote a quantity estimated when using the fit() method. We can be sure of which entry corresponds to which label by looking at the classes_ attribute.

```python
lda.classes_
```

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

```python
priors = lda.priors_
```

```
array([0.44771573, 0.55228424], dtype=float32)
```

```python
str_down = f"{priors[0]:.3f}"
str_up = f"{priors[1]:.3f}"
```

```
'0.552'
```

```python
printmd("The LDA output indicates that $\\hat{\\pi}_{Down}$ = " + str_down +
↪  " and $\\hat{\\pi}_{Up}$ = " + str_up)
```

The LDA output indicates that $\hat{\pi}_{Down} = 0.448$ and $\hat{\pi}_{Up} = 0.552$

The linear discriminant vectors can be found in the scalings_ attribute:

```python
lda.scalings_
```

```
array([[0.44141617]], dtype=float32)
```

These values provide the linear combination of `Lag2` that are used to form the LDA decision rule. In other words, these are the multipliers of the elements of X = x in (4.24).

$$\delta_k = x^T \Sigma^{-1} \mu_k + \frac{\mu_k^T \Sigma^{-1} \mu_k}{2} + log(\pi_k)$$

If $-0.44 \times$ Lag2 is large, then the LDA classifier will predict a market increase, and if it is small, then the LDA classifier will predict a market decline.

```python
lda.xbar_
```

```
array([0.12782034], dtype=float32)
```

```python
lda_pred = lda.predict(X_lda_test)
```

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

```
np.all(lda_pred == labels)
```

```
True
```

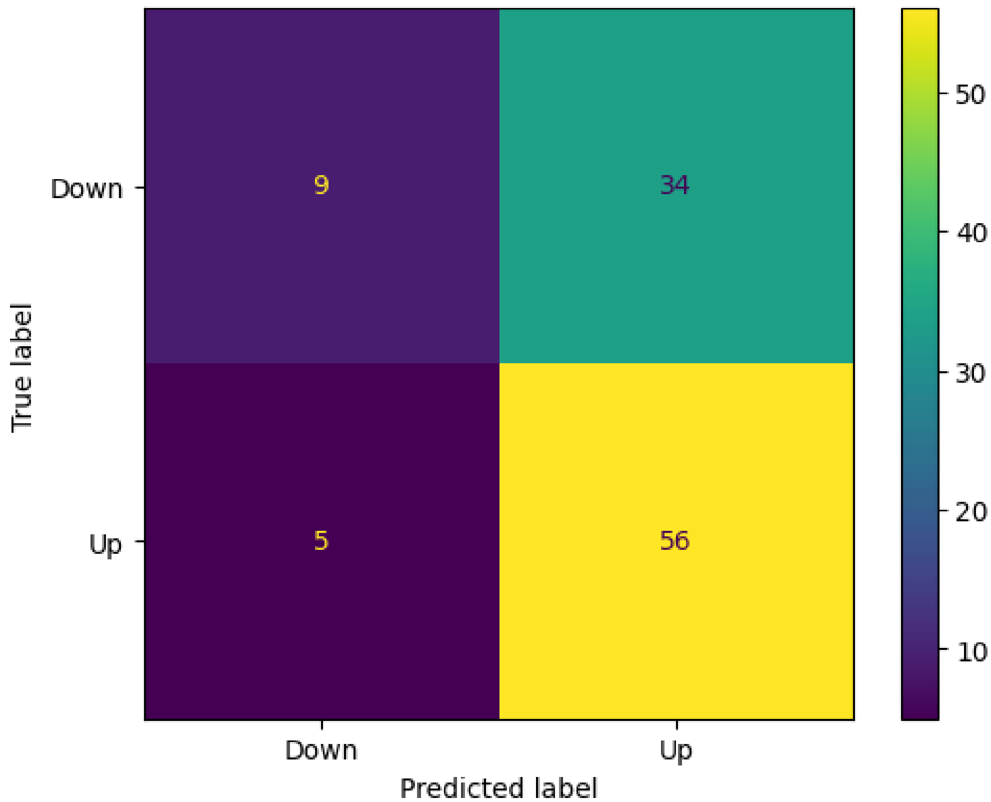
As we observed in our comparison of classification methods (Section 4.5), the LDA and logistic regression predictions are almost identical.

```
confusion_table(L_test, lda_pred)
```

Truth Predicted	Down	Up
Down	9	34
Up	5	56

```
print(classification_report(L_test,
                            lda_pred,
                            digits = 3, output_dict=False))
cm = confusion_matrix(L_test,
                      lda_pred)
ConfusionMatrixDisplay(confusion_matrix=cm,
                       display_labels=["Down", "Up"]).plot();
```

```
              precision    recall  f1-score   support

        Down      0.643     0.209     0.316        43
          Up      0.622     0.918     0.742        61

    accuracy                          0.625       104
   macro avg      0.633     0.564     0.529       104
weighted avg      0.631     0.625     0.566       104
```



(f)

Repeat (d) using QDA.

(g)

Repeat (d) using KNN with K = 1.

(h)

Repeat (d) using naive Bayes.

(i)

Which of these methods appears to provide the best results on this data?

(j)

Experiment with different combinations of predictors, including possible transformations and interactions, for each of the methods. Report the variables, method, and associated confusion matrix that appears to provide the best results on the held out data. Note that you should also experiment with values for K in the KNN classifier.

```
allDone();
```

```
<IPython.lib.display.Audio object>
```