In-Class Assignment 6 - Resampling

- Model Assessment - evaluating a model's performance
- Model Selection - selecting the proper level of flexibility
  - *flexibility* can be defined in many ways, as we have seen
    - number of features in the regression model
    - generally number of parameters estimated
    - how much a fitted model can vary with the training data
- Train, Test, Validate
  - We have seen in the class so far the idea of splitting data into Train and Test
    - This allowed us to measure how well our model *generalizes* to non-training data
    - We have learned to evaluate the chosen model's performance on training and test data (and importantly how to *compare* models and their trade-offs)
    - What we have really been doing is *Validating*, because we don't have a true test set
      - We want to use validation to get the best model we can, then it will be tested
    - Can we use model testing in Assessment and Model Selection?

Cross-Validation (*CV*)

- Select a percentage of the dataset to use for validation, then repeatedly take samples with replacement to create many train/validation pairs.
- *Leave-one-out cross validation* occurs when the number of data points used for validation is n, and therefore the number of train/validation pairs is n.
  - This is a special case of generalized *CV*

In [417…

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import statsmodels.api as sm
from statsmodels.formula.api import ols, logit
from sklearn.model_selection import train_test_split
from statsmodels.stats.outliers_influence import variance_inflation_factor
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
from sklearn.model_selection import ShuffleSplit
import scipy.stats as ss
import seaborn as sns
from itertools import product
from sklearn.metrics import roc_auc_score, roc_curve, accuracy_score
import warnings

# Turn off pandas warnings
warnings.filterwarnings("ignore", category=pd.errors.SettingWithCopyWarning)
```

```
from ISLP import load_data
Smarket = load_data('Smarket')
Smarket
```

Out[417…

	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 × 9 columns

The ``Smarket'' dataset contains a 1250 days of S&P 500 index percentage returns. The data dictionary is as follows:

- Year: year of the return
- Lag1 - Lag5: 1 to 5 days lagged returns
- Volume: the number of shares traded on the previous day (in billions)
- Today: the percentage return on the day in question
- Direction: whether the return was positive or negative on the day in question

Create a training dataset of all days from 2004 and earlier. The test set is data from 2005.

In [418…

```
Smarket = pd.get_dummies(Smarket, columns=['Direction'], drop_first=True)
Smarket['Direction_Up'] = Smarket['Direction_Up'].astype(int)
Smarket
```

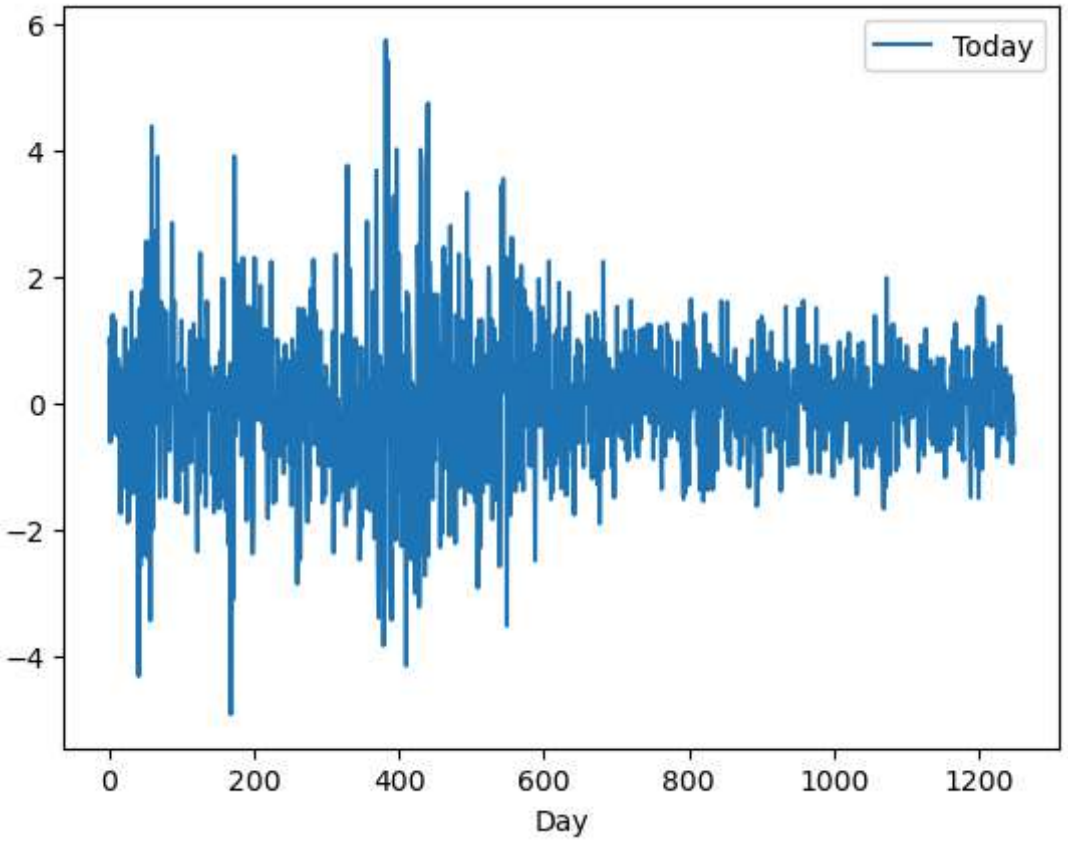
Out[418…

	Year	Lag1	Lag2	Lag3	Lag4	Lag5	Volume	Today	Direction_Up
0	2001	0.381	-0.192	-2.624	-1.055	5.010	1.19130	0.959	1
1	2001	0.959	0.381	-0.192	-2.624	-1.055	1.29650	1.032	1
2	2001	1.032	0.959	0.381	-0.192	-2.624	1.41120	-0.623	0
3	2001	-0.623	1.032	0.959	0.381	-0.192	1.27600	0.614	1
4	2001	0.614	-0.623	1.032	0.959	0.381	1.20570	0.213	1
…	…	…	…	…	…	…	…	…	…
1245	2005	0.422	0.252	-0.024	-0.584	-0.285	1.88850	0.043	1
1246	2005	0.043	0.422	0.252	-0.024	-0.584	1.28581	-0.955	0
1247	2005	-0.955	0.043	0.422	0.252	-0.024	1.54047	0.130	1
1248	2005	0.130	-0.955	0.043	0.422	0.252	1.42236	-0.298	0
1249	2005	-0.298	0.130	-0.955	0.043	0.422	1.38254	-0.489	0

1250 rows × 9 columns

In [419…
```
Smarket['Day'] = Smarket.index.copy()
Smarket.plot(x='Day', y='Today')
```

Out[419… `<Axes: xlabel='Day'>`



In [420…
```
year_day_max_dict = Smarket.groupby('Year')['Day'].max().to_dict()
Smarket['Year_Day'] = Smarket.apply(lambda x: x['Year'] + x['Day'] / year_day_ma
Smarket
```

Out[420…

	Year	Lag1	Lag2	Lag3	Lag4	Lag5	Volume	Today	Direction_Up	Day
0	2001	0.381	-0.192	-2.624	-1.055	5.010	1.19130	0.959	1	0
1	2001	0.959	0.381	-0.192	-2.624	-1.055	1.29650	1.032	1	1
2	2001	1.032	0.959	0.381	-0.192	-2.624	1.41120	-0.623	0	2
3	2001	-0.623	1.032	0.959	0.381	-0.192	1.27600	0.614	1	3
4	2001	0.614	-0.623	1.032	0.959	0.381	1.20570	0.213	1	4
…	…	…	…	…	…	…	…	…	…	…
1245	2005	0.422	0.252	-0.024	-0.584	-0.285	1.88850	0.043	1	1245
1246	2005	0.043	0.422	0.252	-0.024	-0.584	1.28581	-0.955	0	1246
1247	2005	-0.955	0.043	0.422	0.252	-0.024	1.54047	0.130	1	1247
1248	2005	0.130	-0.955	0.043	0.422	0.252	1.42236	-0.298	0	1248
1249	2005	-0.298	0.130	-0.955	0.043	0.422	1.38254	-0.489	0	1249

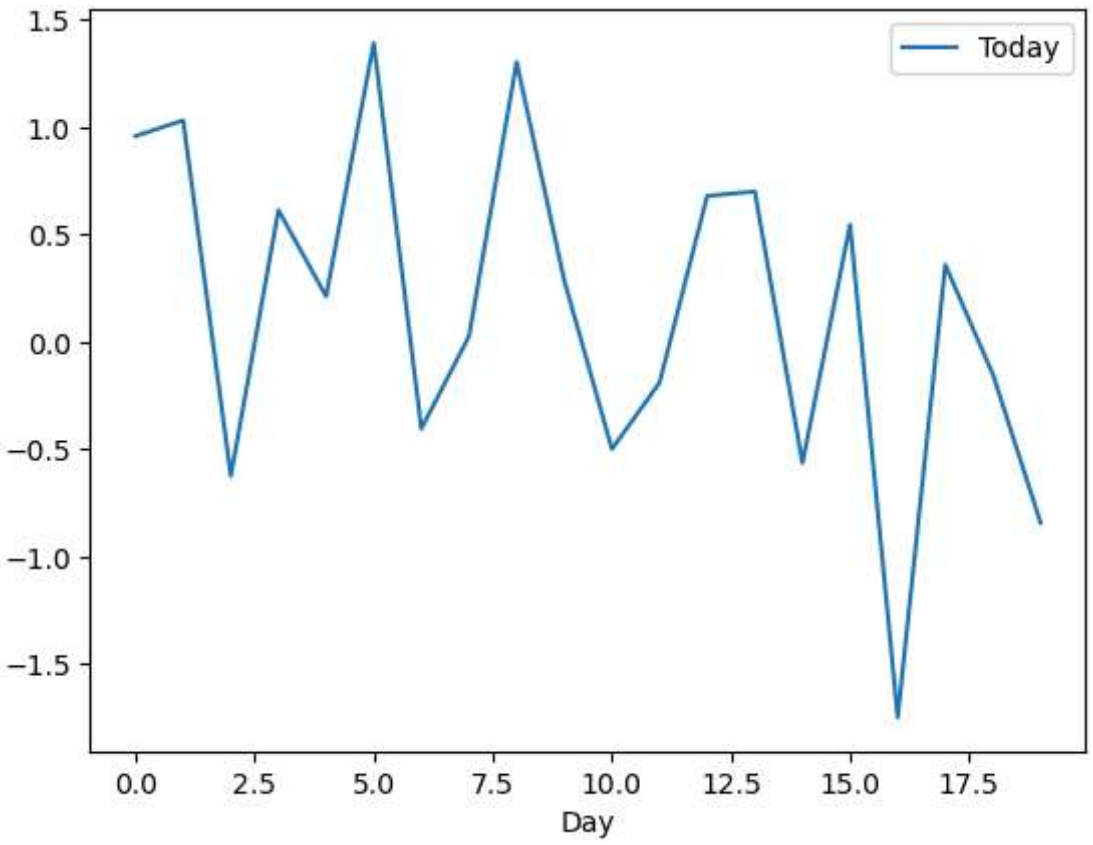
1250 rows × 11 columns

In [421…

```
Smarket[0:20].plot(x='Day', y='Today')
```

Out[421…

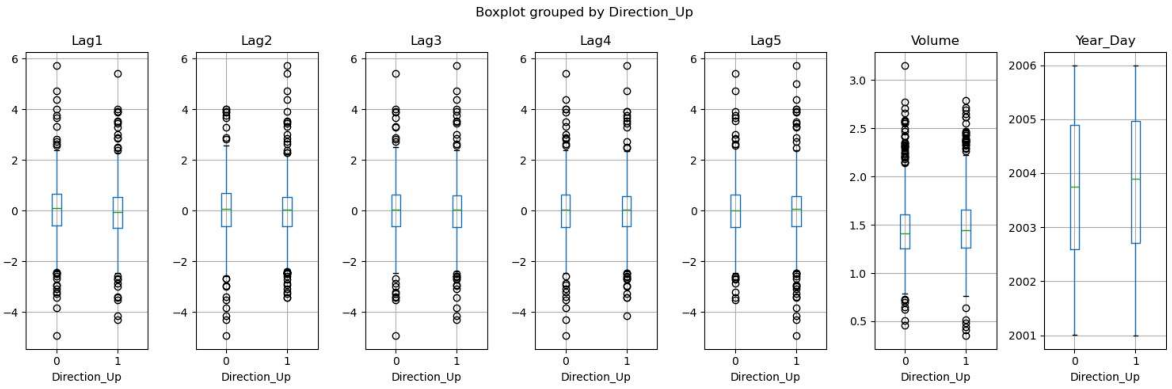
```
<Axes: xlabel='Day'>
```



In [422…

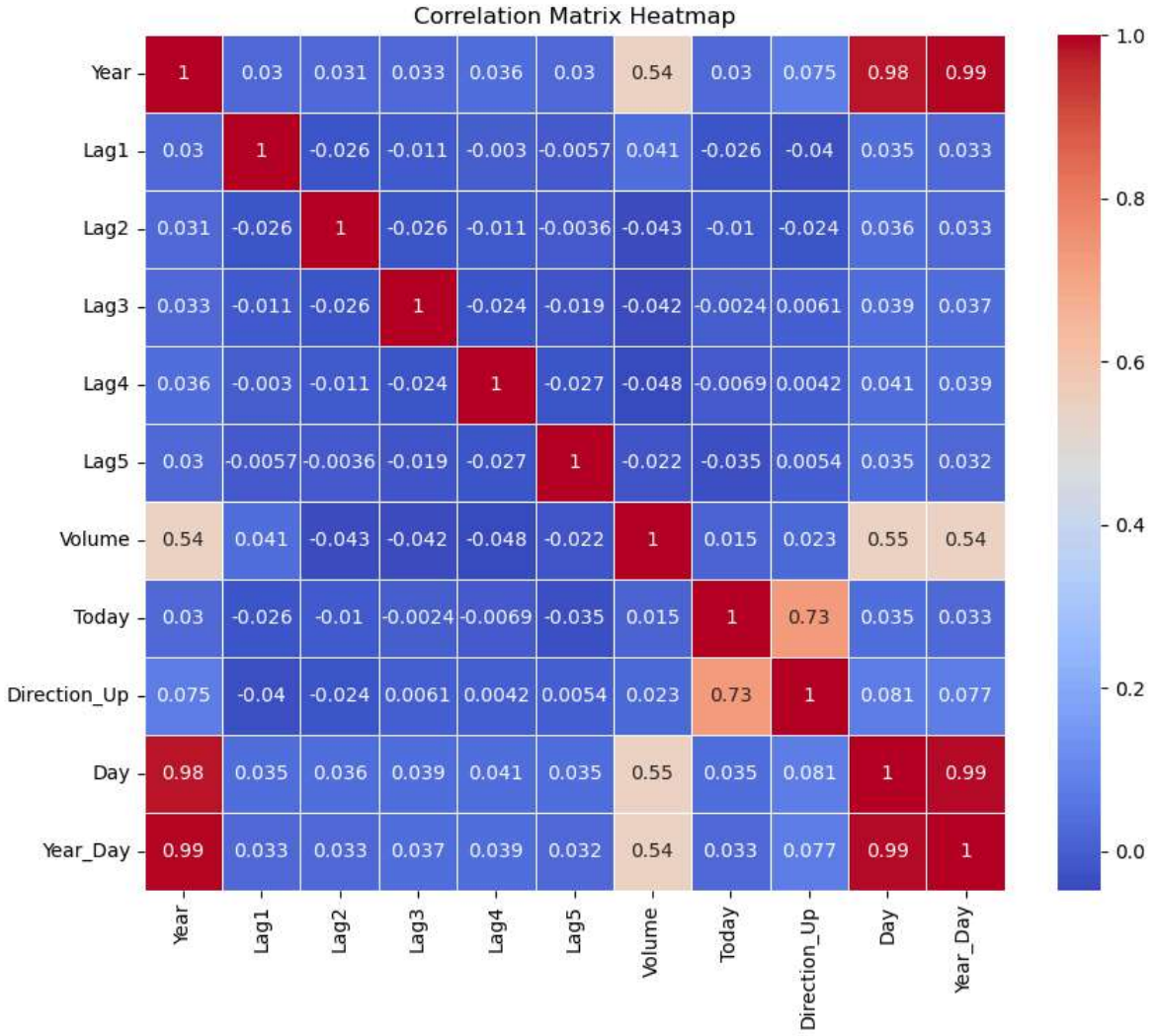
```
fig, ax = plt.subplots(1, 7, figsize=(15, 5))
for (i, col) in enumerate(['Lag1', 'Lag2', 'Lag3', 'Lag4', 'Lag5', 'Volume', 'Ye
```

```
    Smarket.boxplot(column=[col], by='Direction_Up', ax=ax[i])
plt.tight_layout()
```



In [423…

```
corr_matrix = Smarket.corr(numeric_only=True)
plt.figure(figsize=(10, 8))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', linewidths=0.5)
plt.title('Correlation Matrix Heatmap')
plt.show()
```



K Fold Cross Validation

- First we'll take out our test set.
- Then we'll use the training set to get many different *folds* or samples of train and validation data

- How many *folds* and what percentage for each validation set?
  - There is a tradeoff here:
    - *Computational Expense*: for each fold, we'll need to fit our model
    - For example with Leave One Out Cross Validation, we have to fit our model n times - THIS WILL MINIMIZE BIAS, THIS ESTIMATE WILL HAVE HIGH VARIANCE, because regardless of the dataset our estimates will be very similar accross all n folds, therefore from dataset to dataset the differences will be great
    - Empirical analysis has shown that 5 to 10 folds perform well accross a wide range of problem settings
- What is this estimate we talk of?
  - Usually we are talking of some model assessment metric, e.g. MSE, accuracy, AUC
  - Let the metric be M_i for fold i, then we have that the cross validation matric with k folds is just the average of the estimate over the folds:

$$CV_k = \frac{1}{k}\sum_{i=1}^{k} M_i$$

In [424…
```
train = Smarket.loc[Smarket['Year'] <= 2004]
test = Smarket.loc[Smarket['Year'] == 2005]
```

In [425…
```
training_set, validation_set = train_test_split(train, test_size=.2, shuffle=Tru
training_set, validation_set
```

Out[425…

```
(     Year   Lag1   Lag2   Lag3   Lag4   Lag5  Volume  Today  Direction_Up  \
 471  2002 -2.097  0.249 -0.344  2.145  1.941  1.5436  2.799             1
 803  2004 -0.239 -0.132 -1.296 -1.117 -0.127  1.5278  1.637             1
 94   2001  0.320 -1.553 -0.263  1.615  0.269  1.1007 -1.182             0
 329  2002 -0.302 -1.934 -1.026 -0.175  0.886  1.3547  3.750             1
 413  2002 -1.595  1.752 -4.154 -0.188 -0.008  1.4013  1.680             1
 ..    ...    ...    ...    ...    ...    ...     ...    ...           ...
 220  2001 -1.825 -0.684  0.615  1.171 -0.493  1.4237  1.035             1
 609  2003  0.091  2.238 -0.991  0.103  1.283  1.4797 -0.155             0
 208  2001  0.158  0.246 -0.273  1.453  1.439  1.0938 -0.177             0
 39   2001  0.645  0.998  0.586 -0.568  0.104  1.1322  0.226             1
 895  2004 -1.632 -0.096 -0.626  0.445  0.117  1.3974 -1.548             0

      Day     Year_Day
 471  471  2002.955375
 803  803  2004.805416
 94    94  2001.390041
 329  329  2002.667343
 413  413  2002.837728
 ..   ...          ...
 220  220  2001.912863
 609  609  2003.817450
 208  208  2001.863071
 39    39  2001.161826
 895  895  2004.897693

 [798 rows x 11 columns],
      Year   Lag1   Lag2   Lag3   Lag4   Lag5   Volume  Today  Direction_Up  \
 657  2003  0.065 -1.018  0.297 -0.205  0.261  0.97170  0.304             1
 916  2004 -0.418  1.121  0.151  0.463 -0.778  0.92417  0.689             1
 272  2002  0.994 -0.399  1.434  1.486 -0.308  1.21590 -0.181             0
 217  2001  1.171 -0.493 -0.730  1.090 -0.314  0.41030  0.615             1
 119  2001 -0.148  1.249 -0.468 -0.151 -0.551  1.83236  1.008             1
 ..    ...    ...    ...    ...    ...    ...      ...    ...           ...
 114  2001 -0.945  1.136  0.871  0.343 -0.488  1.18920 -0.551             0
 86   2001  0.261 -0.758 -0.029 -0.449 -0.183  0.85820  0.042             1
 302  2002  0.246  0.535  0.585 -1.465 -0.424  1.14760 -0.074             0
 879  2004 -0.329  0.071  0.138  0.334 -0.824  1.46200 -0.430             0
 463  2002  2.463 -0.048  0.772 -2.073 -0.876  1.51900  0.615             1

      Day     Year_Day
 657  657  2003.881879
 916  916  2004.918756
 272  272  2002.551724
 217  217  2001.900415
 119  119  2001.493776
 ..   ...          ...
 114  114  2001.473029
 86    86  2001.356846
 302  302  2002.612576
 879  879  2004.881645
 463  463  2002.939148

 [200 rows x 11 columns])
```

In [426…

```python
def get_metrics(validation_set):
    accuracy = accuracy_score(validation_set[y_var], validation_set['pred'])

    fpr = ((validation_set['pred'] == 1) & (validation_set[y_var] == 0)).mean()
    tpr = ((validation_set['pred'] == 1) & (validation_set[y_var] == 1)).mean()
```

```
    fnr = ((validation_set['pred'] == 0) & (validation_set[y_var] == 1)).mean()
    tnr = ((validation_set['pred'] == 0) & (validation_set[y_var] == 0)).mean()

    auc_logistic = roc_auc_score(validation_set[y_var], validation_set['score'])

    return accuracy, fpr, tpr, fnr, tnr, auc_logistic

def cross_validate(model_formula, threshold=.5, folds=10):
    llr_pvalues = []
    accuracies = []
    fprs = []
    tprs = []
    fnrs = []
    tnrs = []
    aucs = []

    for i in range(10):
        training_set, validation_set = train_test_split(train, test_size=.2, shu
        model = logit(model_formula, data=training_set)
        results = model.fit(disp=False)
        results.summary()

        validation_set['score'] = results.predict(validation_set)
        validation_set['pred'] = (validation_set['score'] > threshold).astype(in

        accuracy, fpr, tpr, fnr, tnr, auc_logistic = get_metrics(validation_set)

        llr_pvalues.append(results.llr_pvalue)
        accuracies.append(accuracy)
        fprs.append(fpr)
        tprs.append(tpr)
        fnrs.append(fnr)
        tnrs.append(tnr)
        aucs.append(auc_logistic)

    print(model_formula)
    print("Average LLR p-value: ", round(np.average(llr_pvalues), 3))
    print("Average Accuracy: ", round(np.average(accuracies), 3))
    print("Average FPR: ", round(np.average(fprs), 3))
    print("Average TPR: ", round(np.average(tprs), 3))
    print("Average FNR: ", round(np.average(fnrs), 3))
    print("Average TNR: ", round(np.average(tnrs), 3))
    print("Average AUC: ", round(np.average(aucs), 3))


y_var = 'Direction_Up'
x_vars_list = ['Lag1 + Lag2 + Lag3 + Lag4 + Lag5 + Volume', 'Lag1 + Lag2 + Lag1*
threshold = .5

for x_vars in x_vars_list:
    model_formula = f"{y_var} ~ {x_vars}"
    cross_validate(model_formula, threshold=threshold)
    print()
```

```
Direction_Up ~ Lag1 + Lag2 + Lag3 + Lag4 + Lag5 + Volume
Average LLR p-value:  0.759
Average Accuracy:  0.493
Average FPR:  0.334
Average TPR:  0.326
Average FNR:  0.173
Average TNR:  0.167
Average AUC:  0.489

Direction_Up ~ Lag1 + Lag2 + Lag1*Lag2
Average LLR p-value:  0.544
Average Accuracy:  0.495
Average FPR:  0.279
Average TPR:  0.297
Average FNR:  0.226
Average TNR:  0.198
Average AUC:  0.493

Direction_Up ~ Lag1 + Year_Day + Year_Day * Lag1
Average LLR p-value:  0.116
Average Accuracy:  0.526
Average FPR:  0.26
Average TPR:  0.297
Average FNR:  0.214
Average TNR:  0.228
Average AUC:  0.534
```

Testing

In [427…

```python
for x_vars in x_vars_list:
    model_formula = f"{y_var} ~ {x_vars}"
    model = logit(model_formula, data=train)
    results = model.fit(disp=False)
    results.summary()

    test['score'] = results.predict(test)
    test['pred'] = (test['score'] > threshold).astype(int)

    accuracy, fpr, tpr, fnr, tnr, auc_logistic = get_metrics(test)
    print(model_formula)
    print("Average Accuracy: ", round(accuracy, 3))
    print("Average FPR: ", round(fpr, 3))
    print("Average TPR: ", round(tpr, 3))
    print("Average FNR: ", round(fnr, 3))
    print("Average TNR: ", round(tnr, 3))
    print("Average AUC: ", round(auc_logistic, 3))
    print()
```

```
Direction_Up ~ Lag1 + Lag2 + Lag3 + Lag4 + Lag5 + Volume
Average Accuracy:  0.48
Average FPR:  0.135
Average TPR:  0.175
Average FNR:  0.385
Average TNR:  0.306
Average AUC:  0.52

Direction_Up ~ Lag1 + Lag2 + Lag1*Lag2
Average Accuracy:  0.56
Average FPR:  0.302
Average TPR:  0.421
Average FNR:  0.139
Average TNR:  0.139
Average AUC:  0.559

Direction_Up ~ Lag1 + Year_Day + Year_Day * Lag1
Average Accuracy:  0.563
Average FPR:  0.437
Average TPR:  0.56
Average FNR:  0.0
Average TNR:  0.004
Average AUC:  0.554
```

In [428…

```
model_formula = "Direction_Up ~ Lag1 + Year_Day + Year_Day * Lag1"
model = logit(model_formula, data=train)
results = model.fit(disp=False)
test['score'] = results.predict(test)
test['pred'] = (test['score'] > threshold).astype(int)
test['Direction_Up'].mean(), test['pred'].mean()
```

Out[428…

```
(0.5595238095238095, 0.996031746031746)
```

In [429…

```
model_formula = "Direction_Up ~ Lag1 + Lag2 + Lag1*Lag2"
model = logit(model_formula, data=train)
results = model.fit(disp=False)
test['score'] = results.predict(test)
test['pred'] = (test['score'] > threshold).astype(int)

money = 100

def get_return(money, sample):
    for i in sample.index:
        if sample.loc[i]['pred'] == 1:
            money *= (1 + sample.loc[i]['Today'] / 100)
    return money


print("Return: ", get_return(money, test))
```

```
Return:  107.70639260599118
```

Bootstrapping

- When we learn about the result above, it is too late to change anything. Can we get an estimate of our performance before testing?

- In bootstrapping, we sample from the data available to generate estimates of metrics of importance. Just as before, we can calculate statistics on the bootstrap samples...

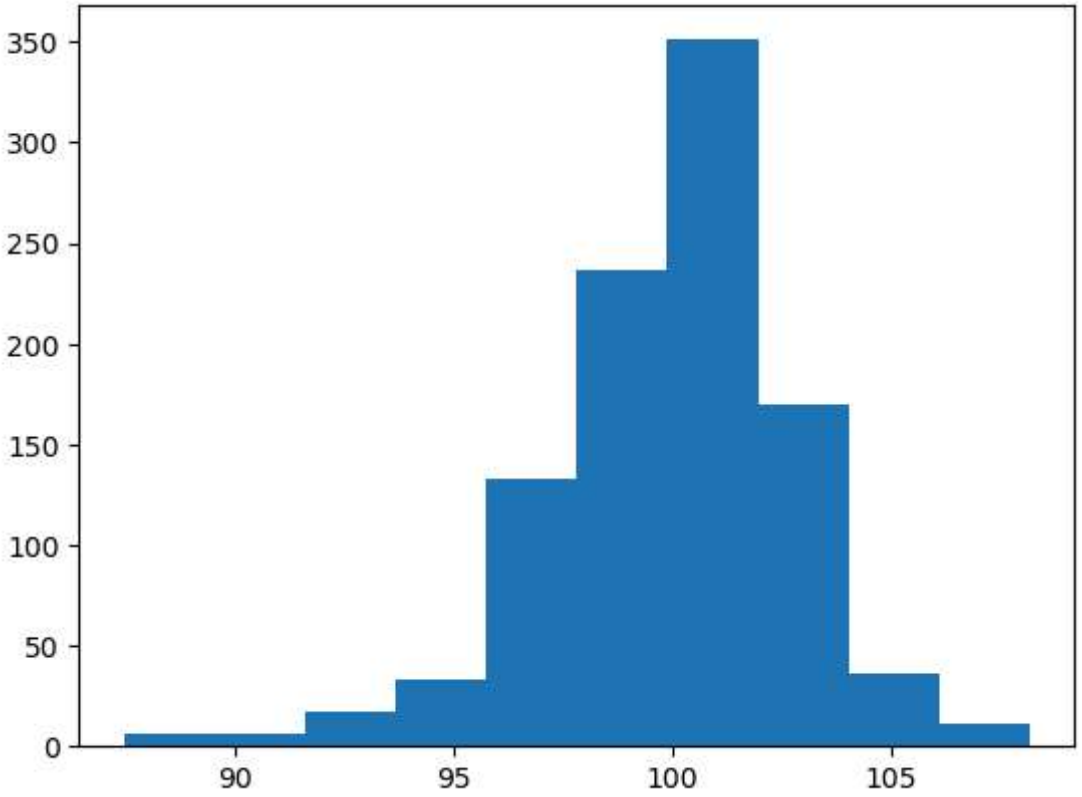
In [447…

```
def bootstrap_return(money, investment_period, model_formula, sample_type="time"

    model = logit(model_formula, data=train)
    results = model.fit(disp=False)
    train['score'] = results.predict(train)
    train['pred'] = (train['score'] > threshold).astype(int)
    returns = []

    for i in range(1000):
        if sample_type == "time":
            sample_start = np.random.randint(train.index.min(), train.index.max(
            sample = train.loc[sample_start: sample_start + investment_period]
        else:
            sample = train.sample(investment_period)
        returns.append(get_return(money, sample))
    return returns

model_formula = "Direction_Up ~ Lag1 + Lag2 + Lag1*Lag2"
threshold = .5
money = 100
investment_period = 10

returns = bootstrap_return(money, investment_period, model_formula)
print("Average return: ", np.average(returns))
print("\tStandard deviation: ", np.sqrt(np.var(returns)))

plt.hist(returns)
```

```
Average return:  99.97041799019988
        Standard deviation:  2.7727094318244205
```

Out[447…

```
(array([  6.,   6.,  17.,  33., 133., 237., 351., 170.,  36.,  11.]),
 array([ 87.4819284 ,  89.54839956,  91.61487071,  93.68134187,
         95.74781302,  97.81428418,  99.88075534, 101.94722649,
        104.01369765, 106.0801688 , 108.14663996]),
 <BarContainer object of 10 artists>)
```



Questions

1. How does your model compare to the best model discussed in this notebook?
    - Answer: Best model in this notebook performs better overall due to its higher accuracy (0.56 vs. 0.48), better true positive rate (0.421 vs. 0.28), and lower false negative rate (0.139 vs. 0.72). It also has a higher AUC (0.559 vs. 0.505), indicating better ability to distinguish between positive and negative classes.

While our model has a slightly lower false positive rate and better true negative rate, these advantages are outweighed by its significantly lower ability to correctly identify positive cases. Overall, the best model of this notebook provides more balanced and effective performance, especially when correctly identifying positives is crucial.