Data Loading

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import xgboost as xgb

sns.set_theme(style="whitegrid")
```

In [2]:

```
raw = pd.read_csv("data/train_data.csv")
```

Data Cleaning

Let's check out the dataset to see how it looks.

In [3]:

```
raw.head()
```

Out[3]:

	symbol	open	high	low	close	average	time	day
0	B	101.72	101.72	101.72	101.72	101.72	06:00:00	0
1	B	101.72	101.72	101.72	101.72	101.72	06:00:05	0
2	B	101.72	101.72	101.72	101.72	101.72	06:00:10	0
3	B	101.72	101.72	101.72	101.72	101.72	06:00:15	0
4	B	101.72	101.72	101.72	101.72	101.72	06:00:20	0

In [4]:

```
raw.isna().sum()
```

Out[4]:

```
symbol     0
open       0
high       0
low        0
close      0
average    0
time       0
day        0
dtype: int64
```

In [5]:

```
raw.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4330254 entries, 0 to 4330253
Data columns (total 8 columns):
 #   Column   Dtype  
---  ------   -----  
 0   symbol   object 
 1   open     float64
 2   high     float64
 3   low      float64
 4   close    float64
 5   average  float64
 6   time     object 
 7   day      int64  
dtypes: float64(5), int64(1), object(2)
memory usage: 264.3+ MB
```

Looks our dataframe is in good shape with no missing values, but the `time` column could to be converted to a better datatype. We'll adjust `time` to a datetime variable. After that, I'd like to combine `day` and `time` into a single column so I don't need two columns to reference the time. Finally, I want to re-index the dataframe by time so that the time is automatically included as an index.

In [6]:
```
# change `time` to datetime variable
raw.time = pd.to_datetime(raw.time)
```

In [7]:
```
# combine `day` and `time` variables
raw.time = pd.Series([time + pd.DateOffset(days=day) for time, day in zip(raw
```

In [8]:
```
# re-index the dataframe and drop `time` and `day` variables
raw = pd.DataFrame(data    =raw.drop(columns=['time', 'day'], axis=1).values,
                   index   =raw.time,
                   columns=raw.drop(columns=['time', 'day'], axis=1).columns)
```

In [9]:
```
raw.head()
```

Out[9]:

time	symbol	open	high	low	close	average
2021-08-27 06:00:00	B	101.72	101.72	101.72	101.72	101.72
2021-08-27 06:00:05	B	101.72	101.72	101.72	101.72	101.72
2021-08-27 06:00:10	B	101.72	101.72	101.72	101.72	101.72
2021-08-27 06:00:15	B	101.72	101.72	101.72	101.72	101.72
2021-08-27 06:00:20	B	101.72	101.72	101.72	101.72	101.72

Our dataframe is now in excellent condition.

Let's collect the symbols and set up a dictionary over symbols that maps to the dataframe for just that symbol. I think this will be easier to work with.

In [10]:
```
symbols = np.sort(raw.symbol.unique())

stocks  = {symbol : raw[raw.symbol == symbol].drop("symbol", axis=1).astype('
```

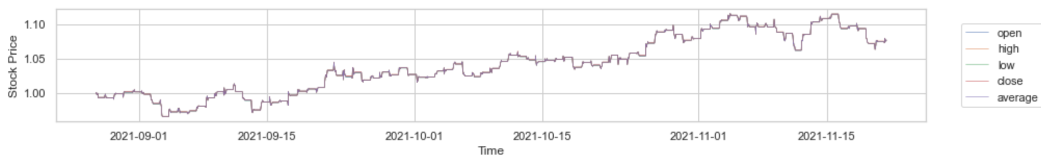
In [11]:
```
stocks["A"].info()
```

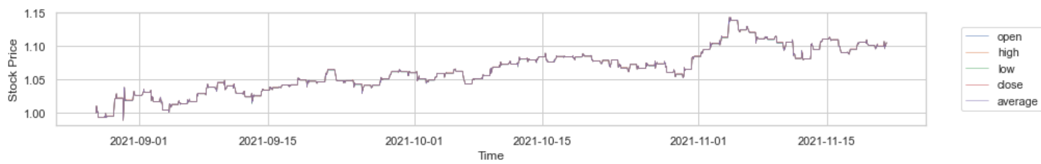
```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 426897 entries, 2021-08-27 06:29:50 to 2021-11-21 12:59:55
Data columns (total 5 columns):
 #   Column   Non-Null Count   Dtype  
---  ------   --------------   -----  
 0   open     426897 non-null  float64
 1   high     426897 non-null  float64
 2   low      426897 non-null  float64
 3   close    426897 non-null  float64
 4   average  426897 non-null  float64
dtypes: float64(5)
memory usage: 19.5 MB
```

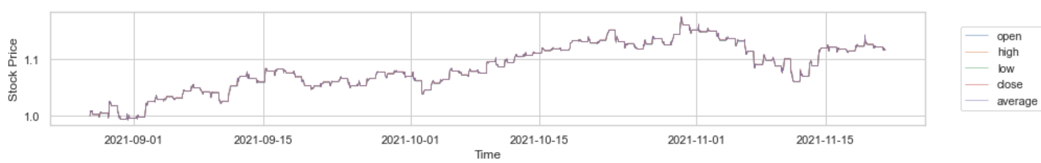
Data Analysis and Visualization

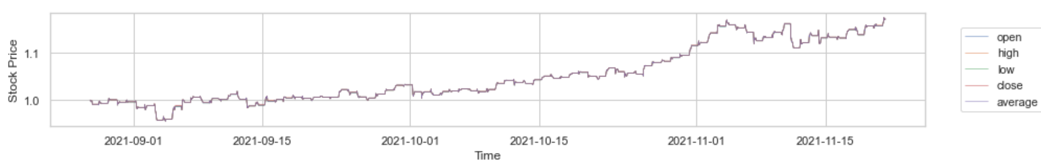
There is a lot of data here. Let's plot each of the features fully to see if anything stands out. I plan to simplify to consider only open prices, but I want to make sure. I will also normalize by the first open price so that we can compare across tickers more easily.

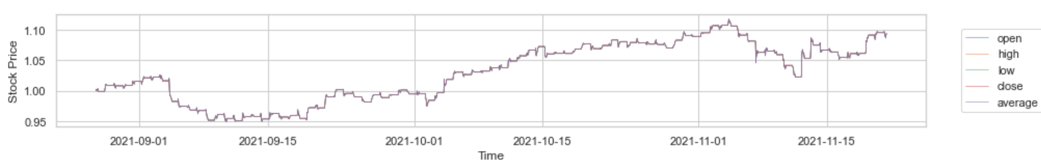
In [12]:
```
for symbol in symbols:

    plt.figure(figsize=(15, 2))

    for column in stocks["A"].columns:

        sns.lineplot(data=stocks[symbol][column].resample('1H').ffill() \
                    /stocks[symbol].open.iloc[0], label=column, linewidth=0.

    plt.xlabel("Time")
    plt.ylabel("Stock Price")
    plt.legend(bbox_to_anchor=(1.04,0.5), loc="center left", borderaxespad=0)
```

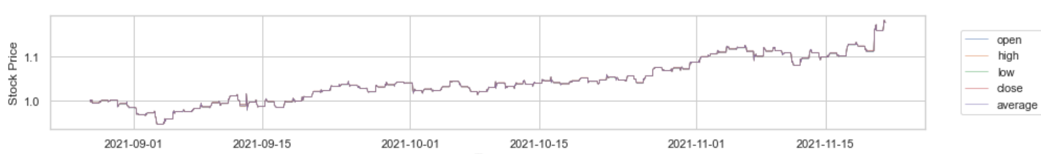


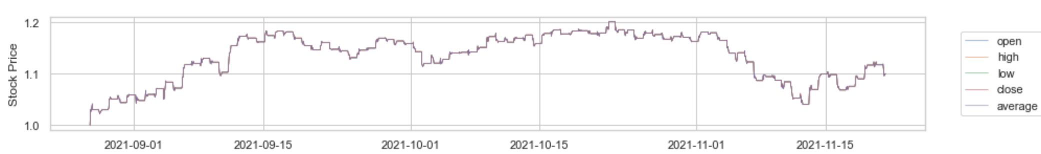


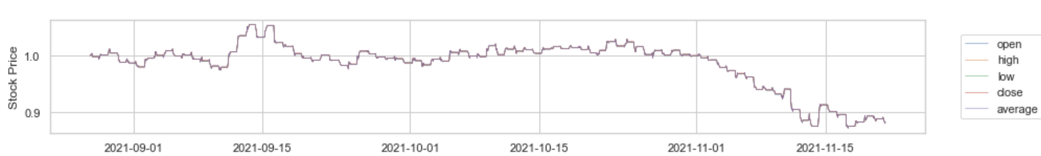


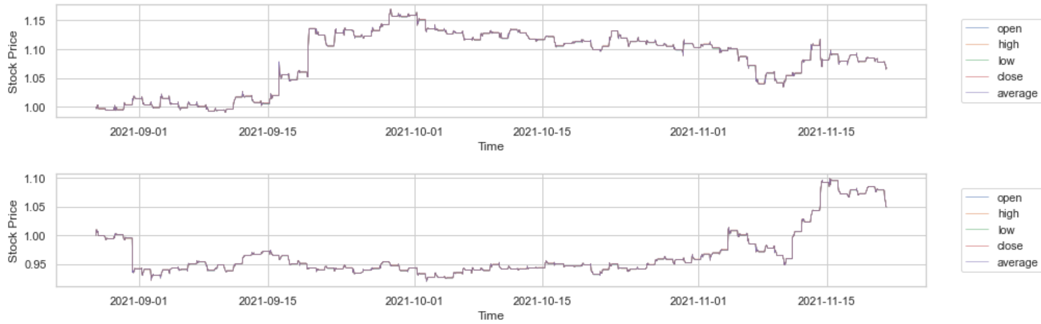








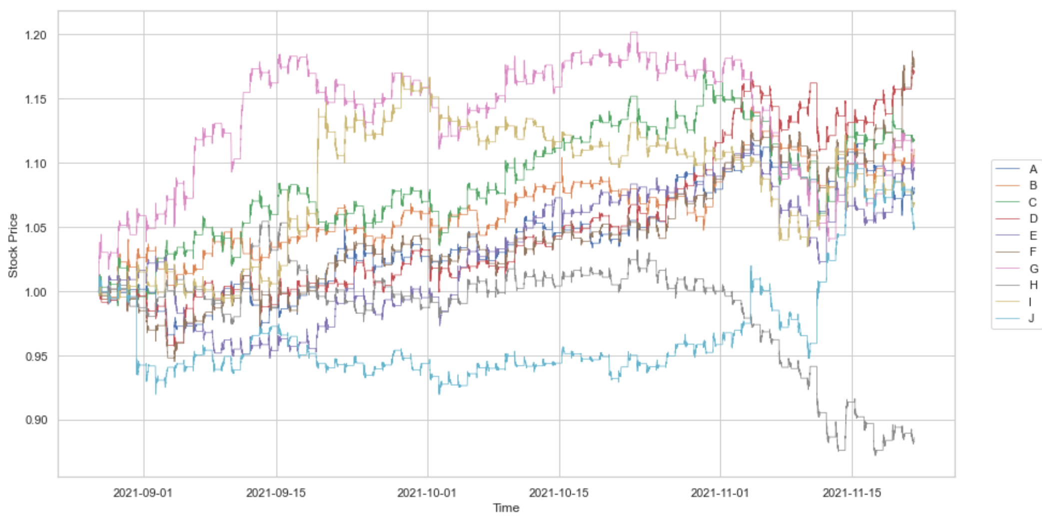




Looking at this, there doesn't seem to be much difference among the columns, so I will select only the `open` prices to use for all future work. Now, let's take a look at the open prices across all tickers.

In [13]:
```
for symbol in symbols:
    stocks[symbol] = stocks[symbol].drop(['close', 'high', 'low', 'average'],
```

In [14]:
```
plt.figure(figsize=(15,8))

for symbol in symbols:

    sns.lineplot(data=stocks[symbol].open.resample('5min').ffill() \
                 /stocks[symbol].open.iloc[0], label=symbol, linewidth=0.75)

plt.xlabel("Time")
plt.ylabel("Stock Price")
plt.legend(bbox_to_anchor=(1.04,0.5), loc="center left", borderaxespad=0);
```



Even with a 5 minute resampling of the data, much of the interday and intraday data is not useful. And since our prediction task is several days into the future, a great starting point would be to use daily data. We can always use higher-resolution data later in more advanced models if we deem it necessary. We now resample the data to a daily resolution.

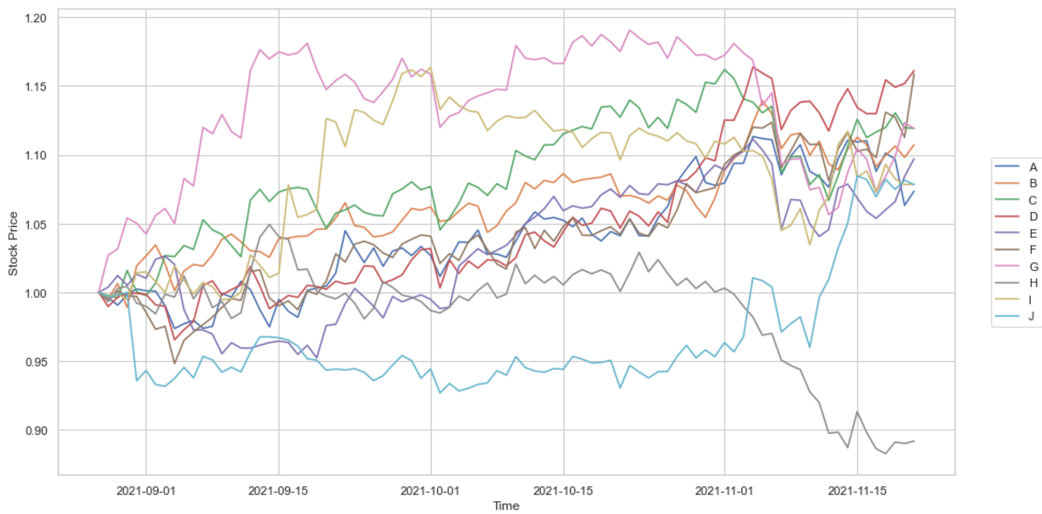
In [15]:
```
for symbol in symbols:
    stocks[symbol] = stocks[symbol].resample('1D').bfill()
```

In [16]:

```
stocks_open_normalized_data = np.zeros((87,10))
for i, symbol in enumerate(symbols):
    stocks_open_normalized_data[:, i] = stocks[symbol].open/stocks[symbol].op

stocks_open_normalized = pd.DataFrame(data=stocks_open_normalized_data,
                                      columns=symbols,
                                      index=stocks["A"].index)
```

In [17]:

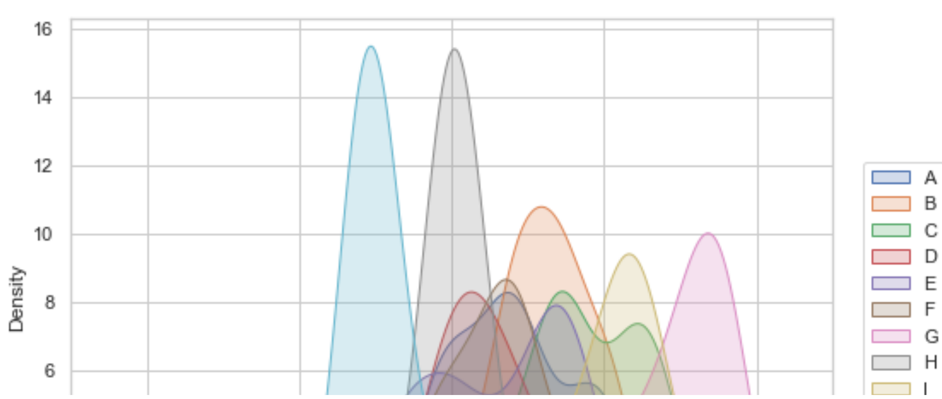
```
plt.figure(figsize=(15,8))

for symbol in symbols:

    sns.lineplot(data=stocks[symbol].open/stocks[symbol].open.iloc[0],
                 label=symbol)

plt.xlabel("Time")
plt.ylabel("Stock Price")
plt.legend(bbox_to_anchor=(1.04,0.5), loc="center left", borderaxespad=0);
```

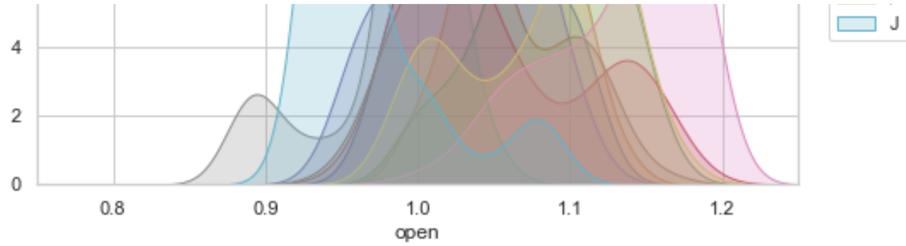


Great! Let's take a deeper look at each of the tickers.

In [18]:

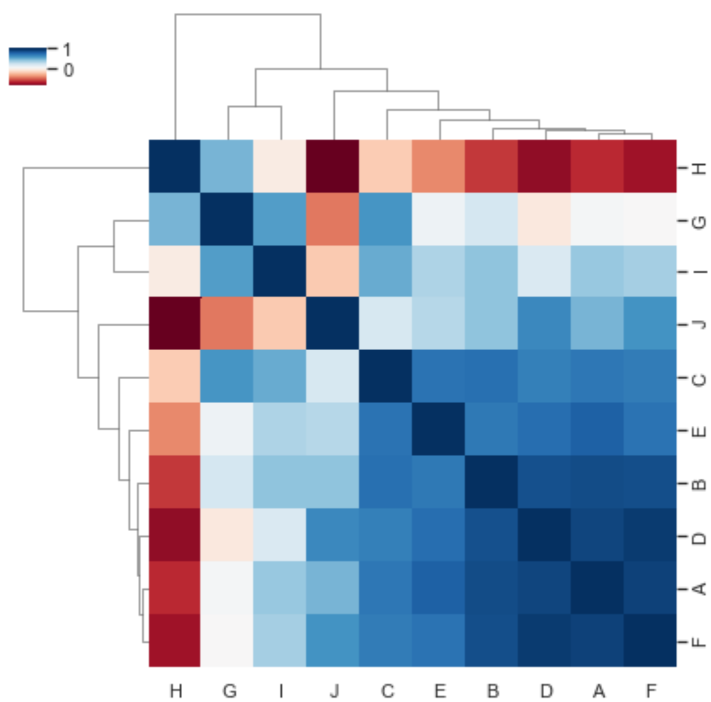
```
plt.figure(figsize=(8,6))

for symbol, df in stocks.items():
    sns.kdeplot(df.open/df.open.iloc[0], shade=True, label=symbol, bw_adjust=

plt.xlim(0.75, 1.25)
plt.legend(bbox_to_anchor=(1.04,0.5), loc="center left", borderaxespad=0);
```





Looking at these KDEs, we see that most stocks are clustered together, but stocks J, H, B, I, and G have slightly different distributions. It might be nice to probe a little further.

In [19]:
```
stocks_together = pd.DataFrame(data=np.array([stocks[symbol]/stocks[symbol].o
                               index=stocks["A"].index,
                               columns=symbols)

sns.clustermap(stocks_together.corr(), cmap="RdBu", clim=(-1, 1), figsize=(6,
plt.gca().set_aspect('equal');
```



Using an average linkage function, we see for a cut at 4 clusters, we have:

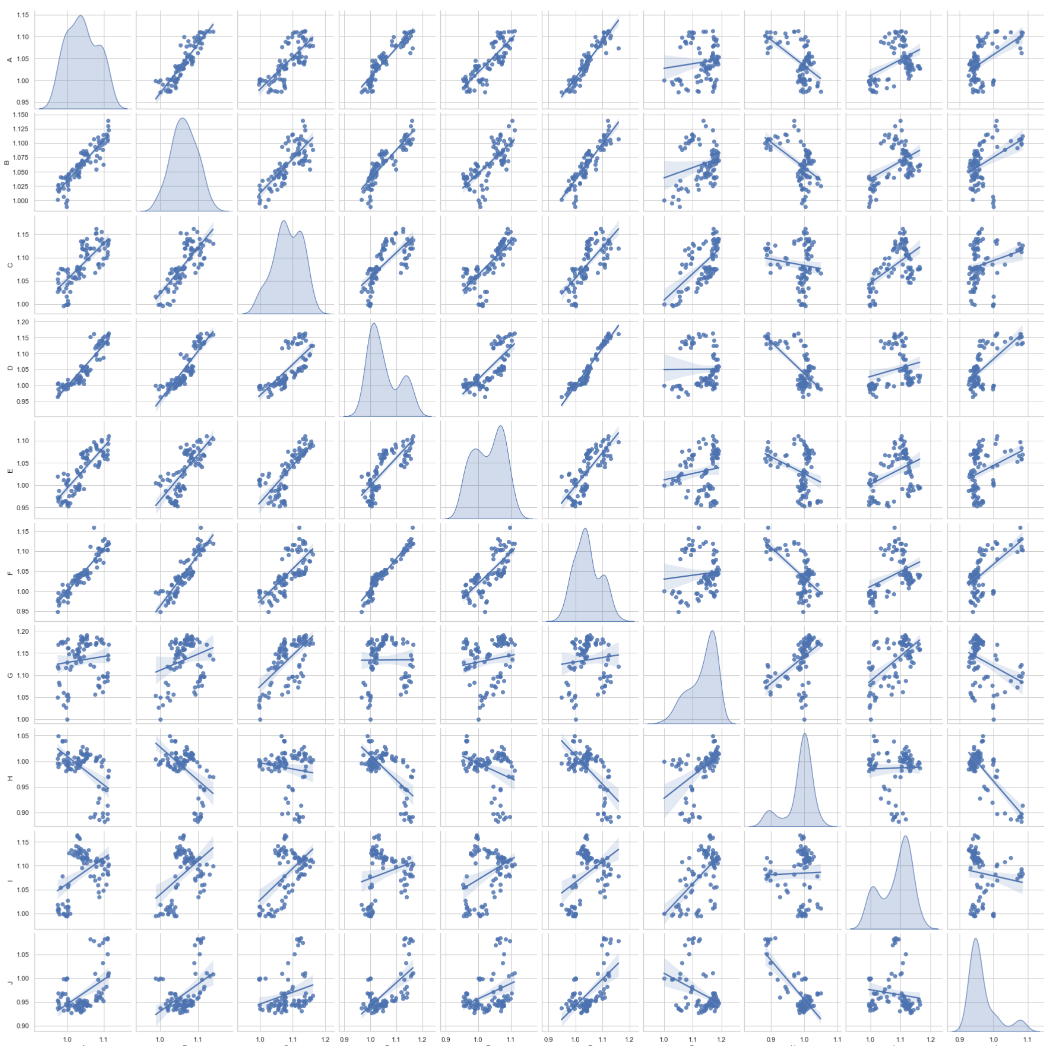
- H, which appears to be negatively correlated with most other stocks and strong negative correlation with stock J
- G & I, which generally have weaker correlation with all other stocks and negative correlation with stock J
- J, which has weaker correlation with most stocks, very strong negative correlation with stock H, and weaker negative correlation with stocks G & I
- C, E, B, D, A, & F, which are all strongly correlated and have strong negative correlation with stock H.

This matches mostly with conclusions we drew from the KDE plot, though stock B does not stand out as much under this metric.

Let's take a deeper look at pairs of variables.

In [20]:

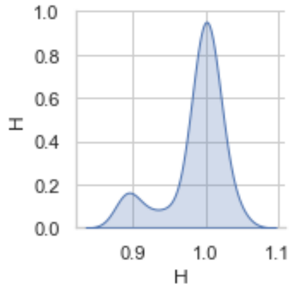
```
sns.pairplot(data=stocks_together, kind='reg', diag_kind='kde');
```



Cluster 1

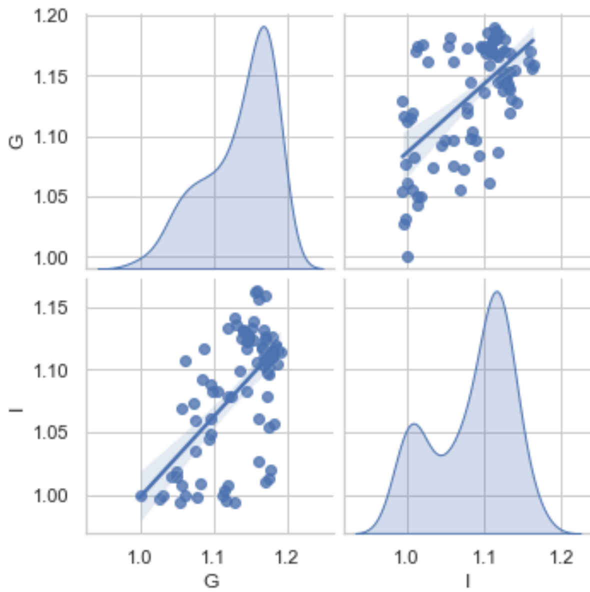
In [21]:

```
sns.pairplot(data=stocks_together, kind='reg', diag_kind='kde', vars=["H"]);
```



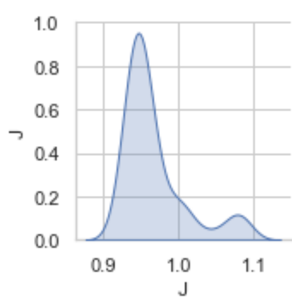
Cluster 2

In [22]:
```
sns.pairplot(data=stocks_together, kind='reg', diag_kind='kde', vars=["G", "I
```



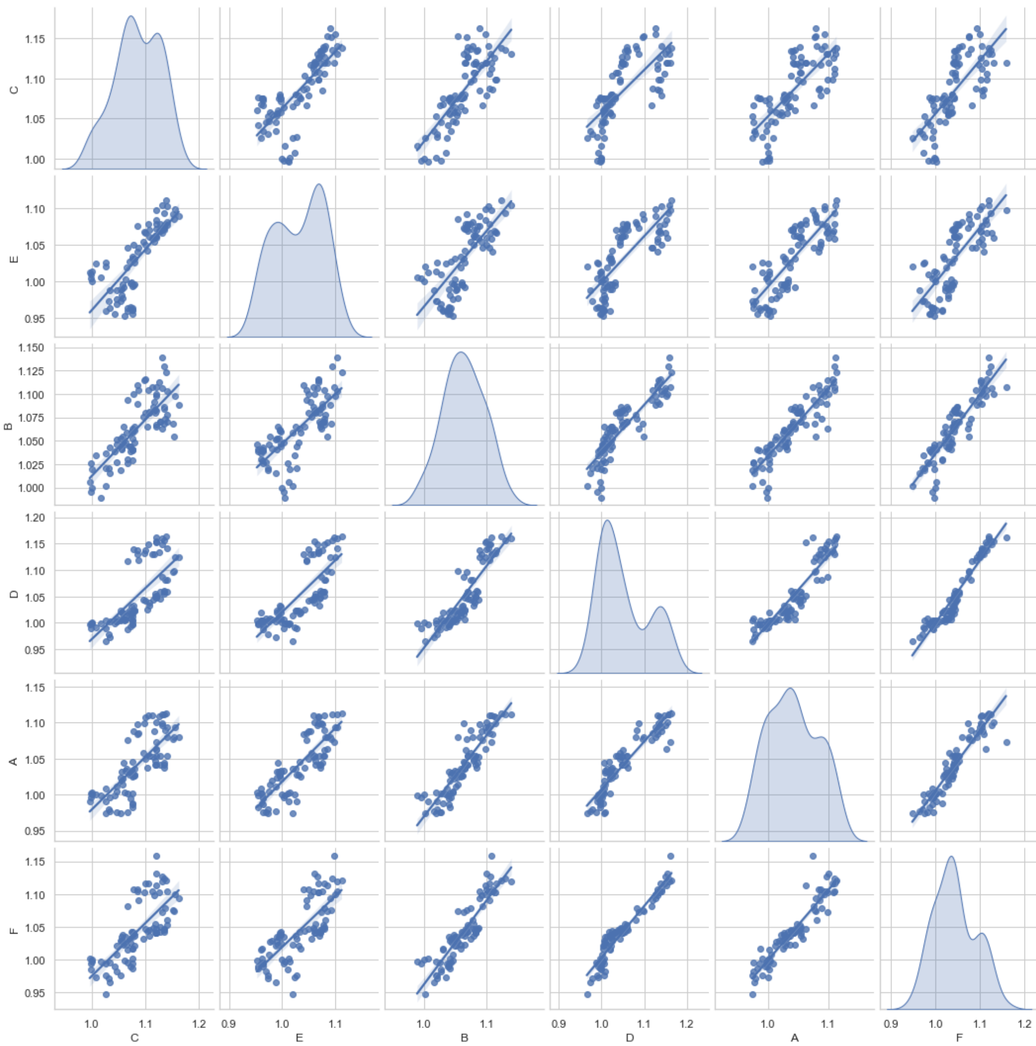
Cluster 3

In [23]:
```
sns.pairplot(data=stocks_together, kind='reg', diag_kind='kde', vars=["J"]);
```



Cluster 4

In [24]:
```
sns.pairplot(data=stocks_together, kind='reg', diag_kind='kde', vars=["C", "E
```

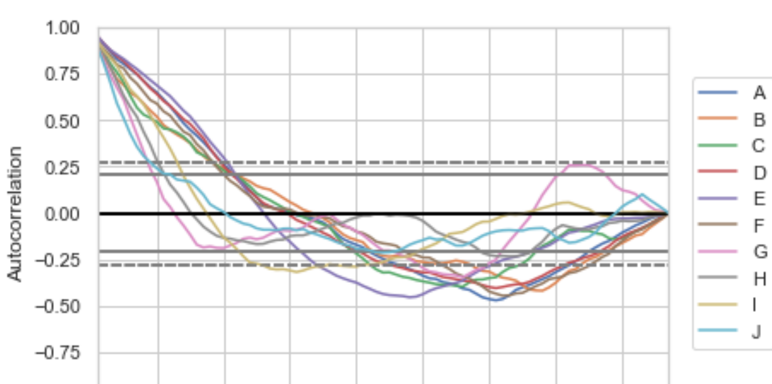


From these plots, we can see just how positively correlated the stocks in cluster 4 are. While there are some interesting nonlinear within-cluster relationships (such as stocks C & D), the relationships are largely linear within the clusters. Between-cluster relationships tend to be more nonlinear (as shown in the large pairplot).

Let's take a look at autocorrelation to see temporal variation.

In [25]:

```
for symbol in symbols:
    pd.plotting.autocorrelation_plot(stocks[symbol].open, label=symbol)

plt.legend(bbox_to_anchor=(1.04,0.5), loc="center left", borderaxespad=0);
```



We observe strong autocorrelation of stocks symbols for 0-20 day lags. Beyond this, the signal is relatively weak, although in our dataset, there is significant anti-autocorrelation around the 50-60 day lag. Overall, this makes a strong case for using traditional stock market indicators (moving averages, etc.) and/or lag variables to learn the autocorrelative model.

Models Using SMA & EMA Indicators

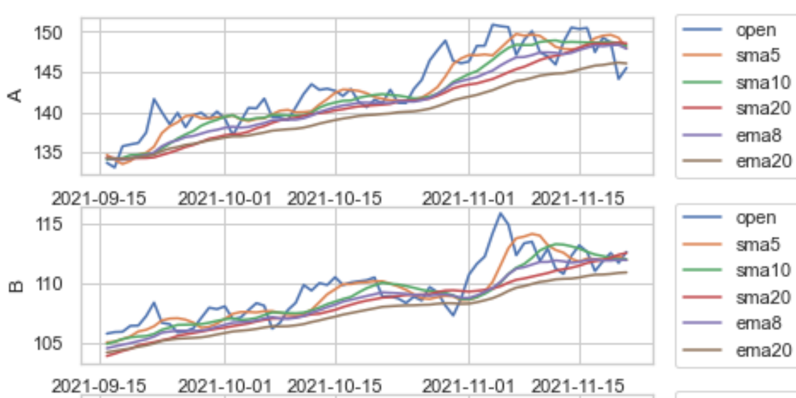
We start with featurizing our dataset by computing the 5-, 10-, and 20-day simple moving averages (SMA) and the 8- and 20-day exponential moving averages (EMA). The SMA is balanced over the given period, so it is better as a long-term indicator, while the EMA is more weighted towards the current day, so it is better as a short-term indicator. Let's create these features now and visualize them.

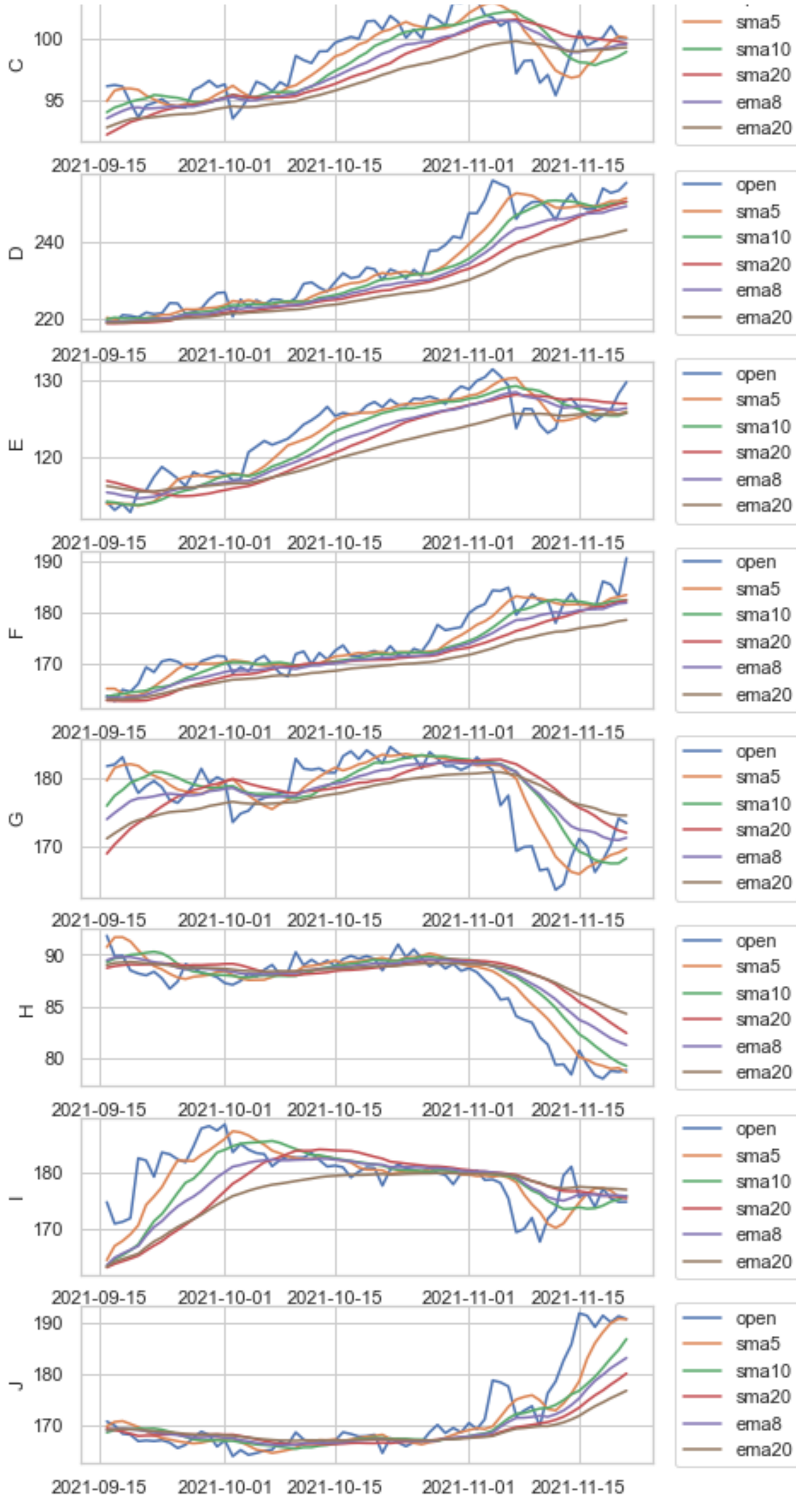
In [26]:

```
for symbol in symbols:

    stocks[symbol]["sma5"]  = stocks[symbol].open.rolling(5).mean().shift(1)
    stocks[symbol]["sma10"] = stocks[symbol].open.rolling(10).mean().shift(1)
    stocks[symbol]["sma20"] = stocks[symbol].open.rolling(20).mean().shift(1)
    stocks[symbol]["ema8"]  = stocks[symbol].open.ewm(8).mean().shift(1)
    stocks[symbol]["ema20"] = stocks[symbol].open.ewm(20).mean().shift(1)
```

In [32]:

```
fig, axs = plt.subplots(10, 1, figsize=(6, 20))
axs = axs.flatten()

for i, symbol in enumerate(symbols):

    for col in stocks[symbol]:

        axs[i].plot(stocks[symbol][col], label=col)
        axs[i].set_ylabel(symbol)

    axs[i].legend(bbox_to_anchor=(1.04,0.5), loc="center left", borderaxespad
```





Since we don't have an SMA or EMA value for 20 days until the 20th day, we need to truncate our dataset so each of the training examples is complete.

In [33]:

```
for symbol in symbols:
    stocks[symbol] = stocks[symbol][20:]
```

We also need to set up a label column for the next open price.

In [34]:

```
for symbol in symbols:
    stocks[symbol]["next_open"] = stocks[symbol].open.shift(-1)
    stocks[symbol]               = stocks[symbol][:-1]
```

We'd like to train models in three ways:

- independently (a model is trained for each stock using only its own data)
- jointly (a model is trained for each stock using all stock data)
- clustered (a model is trained for each stock cluster using only all stock cluster data)

Each of these will require a different train-test split, so we'll do this as part of each model training.

We'd also like to train three kinds of models:

- ridge regression
- elastic net regression
- decision tree regression with gradient boosting (XGBoost)

We believe these models will be able to perform good feature selection and identify the signal within the data.

In [35]:

```
from sklearn.linear_model import RidgeCV, Ridge, ElasticNetCV, ElasticNet
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import mean_squared_error as MSE
```

In [36]:

```
# cross-validate and train ridge regressor
def train_ridge_regressor(X, y, X_train, y_train, X_test, y_test):

    alphas = [1E-3, 1E-2, 1E-1, 1E0, 1E1, 1E2]
    model  = RidgeCV(alphas=alphas)

    model.fit(X, y)

    # optimal cross-validated model
    model = Ridge(alpha=model.alpha_, random_state=0)
    model.fit(X_train, y_train)

    train_mse = MSE(y_train, model.predict(X_train))
    test_mse  = MSE(y_test,  model.predict(X_test))

#     print("Ridge Regressor")
#     print("train loss: %4.3e" % train_mse)
#     print("test loss:  %4.3e" % test_mse)

    return model, train_mse, test_mse
```

In [37]:

```
# cross-validate and train elastic net regressor
def train_elastic_net_regressor(X, y, X_train, y_train, X_test, y_test):

    l1_ratios = [.01, .05, .1, .3, .5, .7, .9, .95, .99, 1]
    model     = ElasticNetCV(l1_ratio=l1_ratios)

    model.fit(X, y)

    # optimal cross-validated model
    model = ElasticNet(alpha=model.alpha_, l1_ratio=model.l1_ratio_, random_s
    model.fit(X_train, y_train)

    train_mse = MSE(y_train, model.predict(X_train))
    test_mse  = MSE(y_test,  model.predict(X_test))

#     print("Elastic Net Regressor")
#     print("train loss: %4.3e" % train_mse)
#     print("test loss:  %4.3e" % test_mse)

    return model, train_mse, test_mse
```

In [38]:

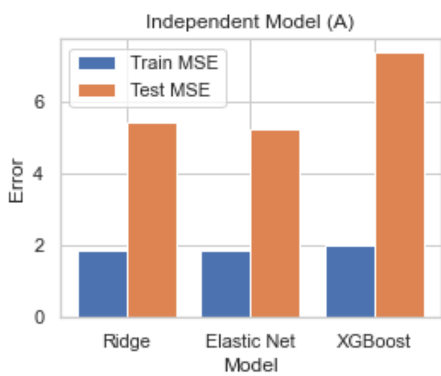
```
# cross-validate and train xgboost regressor
def train_xgboost_regressor(X, y, X_train, y_train, X_test, y_test):

    parameters = {
        'n_estimators': [10, 50, 100],
        'learning_rate': [0.01, 0.05, 0.1, 0.2],
        'max_depth': [2, 5, 8],
        'gamma': [0.01, 0.02, 0.05, 0.1],
        'random_state': [0]
    }

    model_family = xgb.XGBRegressor(objective="reg:squarederror")
    model        = GridSearchCV(model_family, parameters)

    model.fit(X, y)

    # optimal cross-validated model
    model = xgb.XGBRegressor(**model.best_params_, objective="reg:squarederro
    model.fit(X_train, y_train, eval_set=[(X_train, y_train), (X_test, y_test

    train_mse = MSE(y_train, model.predict(X_train))
    test_mse  = MSE(y_test,  model.predict(X_test))

#     print("XGBoost Regressor")
#     print("train loss: %4.3e" % train_mse)
#     print("test loss:  %4.3e" % test_mse)

    return model, train_mse, test_mse
```

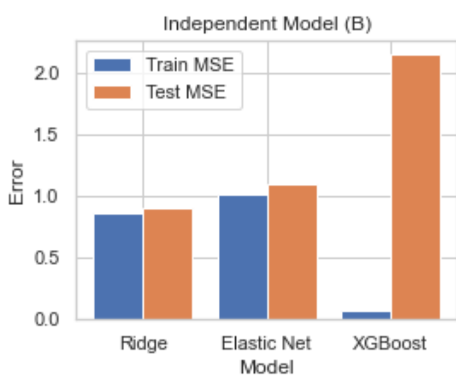
Independent Models

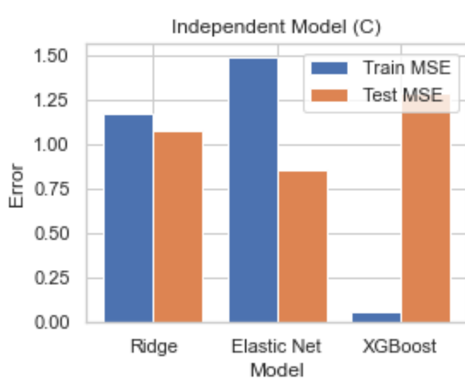
In [39]:

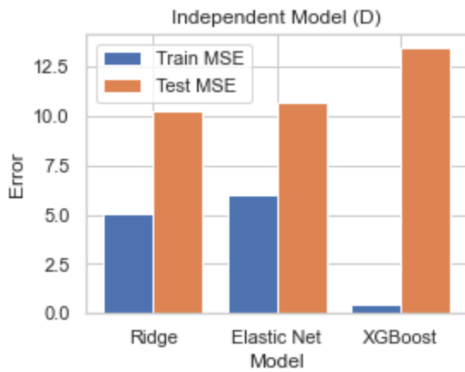
```
import warnings
warnings.filterwarnings('ignore')

for symbol in symbols:

#     print(symbol)

    # train-test split
    X = stocks[symbol].drop(["next_open"], axis=1)
    y = stocks[symbol].next_open

    X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.80
```

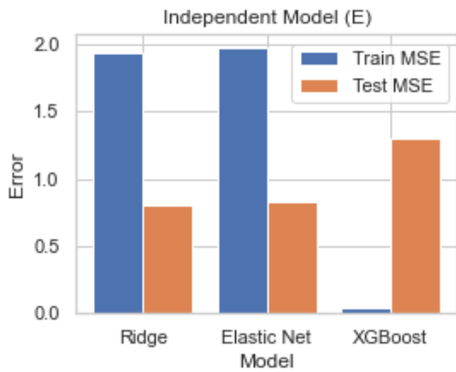
```
# train models
ridge_model, ridge_train, ridge_test = train_ridge_regressor(X,       y,
                                                             X_train, y_t
                                                             X_test,  y_t

elast_model, elast_train, elast_test = train_elastic_net_regressor(X,
                                                                   X_trai
                                                                   X_test

xgb_model,   xgb_train,   xgb_test   = train_xgboost_regressor(X,       y
                                                               X_train, y
                                                               X_test,  y

fig, ax = plt.subplots(figsize=(4, 3))
train = ax.bar(np.arange(0,3),     [ridge_train, elast_train, xgb_train],
               label="Train MSE")
test  = ax.bar(np.arange(0,3)+0.4, [ridge_test,  elast_test,  xgb_test],
               label="Test MSE")

ax.set_xlabel("Model")
ax.set_ylabel("Error")
ax.set_title("Independent Model (%s)" % symbol)
ax.set_xticks(np.arange(0,3) + 0.4/2)
ax.set_xticklabels(["Ridge", "Elastic Net", "XGBoost"])
ax.legend()

plt.show()
```

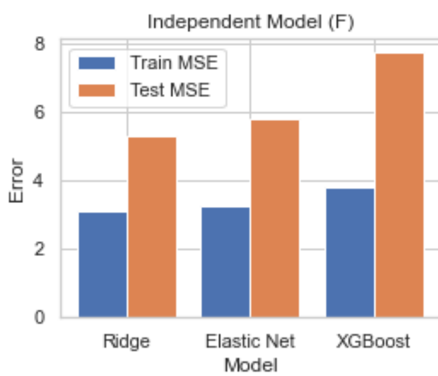


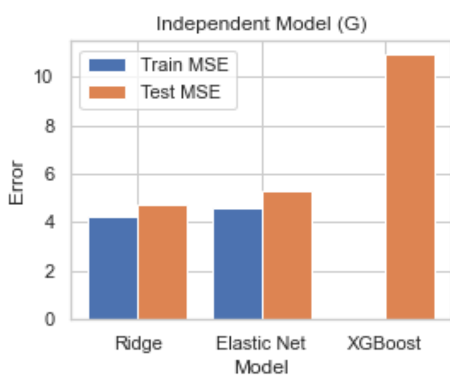


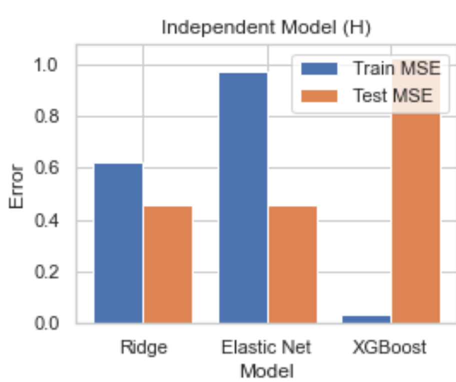


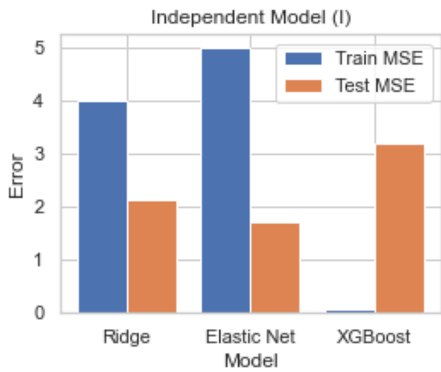

Independent Model (D)
Train MSE
Test MSE
Error
0.0
2.5
5.0
7.5
10.0
12.5
Ridge
Elastic Net
XGBoost
Model

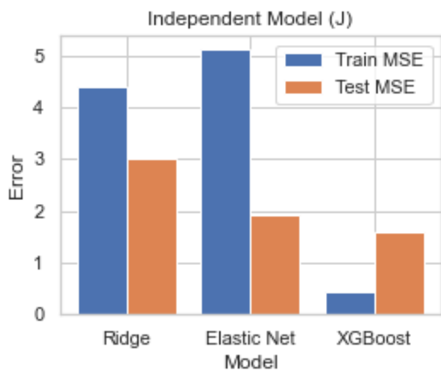

Independent Model (E)
Train MSE
Test MSE
Error
0.0
0.5
1.0
1.5
2.0
Ridge
Elastic Net
XGBoost
Model


Independent Model (F)
Train MSE
Test MSE
Error
0
2
4
6
8
Ridge
Elastic Net
XGBoost
Model


Independent Model (G)
Train MSE
Test MSE
Error
0
2
4
6
8
10
Ridge
Elastic Net
XGBoost
Model


Independent Model (H)
Train MSE
Test MSE
Error
0.0
0.2
0.4
0.6
0.8
1.0
Ridge
Elastic Net
XGBoost
Model





In general, we have the ridge and elastic net models outperforming the XGBoost model for most stocks.

Joint Models

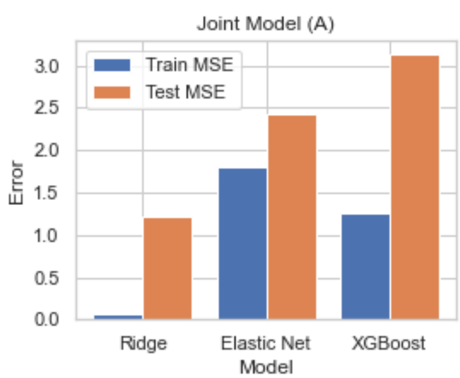
Let's set up the joint dataframe needed to train our model.

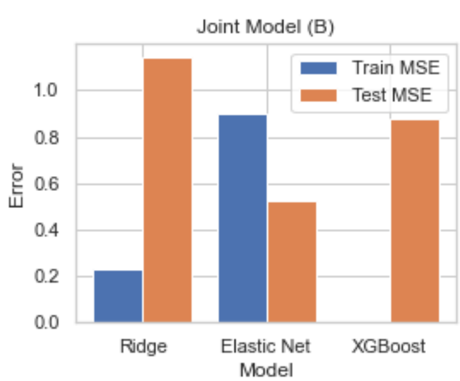
In [40]:

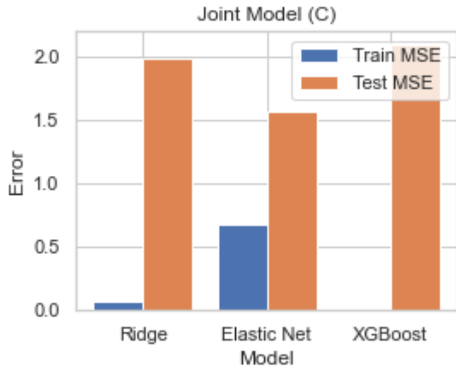
```
stocks = {symbol : raw[raw.symbol == symbol].drop("symbol", axis=1).astype('f

for symbol in symbols:
    stocks[symbol] = stocks[symbol].resample('1D').bfill()
    stocks[symbol] = stocks[symbol].drop(["high", "low", "close", "average"],

    stocks[symbol][symbol + "_sma5"]  = stocks[symbol].open.rolling(5).mean()
    stocks[symbol][symbol + "_sma10"] = stocks[symbol].open.rolling(10).mean(
    stocks[symbol][symbol + "_sma20"] = stocks[symbol].open.rolling(20).mean(
    stocks[symbol][symbol + "_ema8"]  = stocks[symbol].open.ewm(8).mean().shi
    stocks[symbol][symbol + "_ema20"] = stocks[symbol].open.ewm(20).mean().sh

    stocks[symbol] = stocks[symbol][20:]

    stocks[symbol][symbol + "_next_open"] = stocks[symbol].open.shift(-1).ast

    stocks[symbol] = stocks[symbol].rename(columns={"open": symbol + "_open"}

    stocks[symbol] = stocks[symbol][:-1]

# concatenate all stocks into one df
stocks_joint = pd.concat([stocks[symbol] for symbol in symbols], axis=1)
```

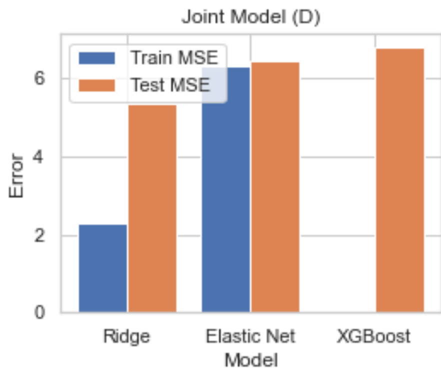
In [41]:

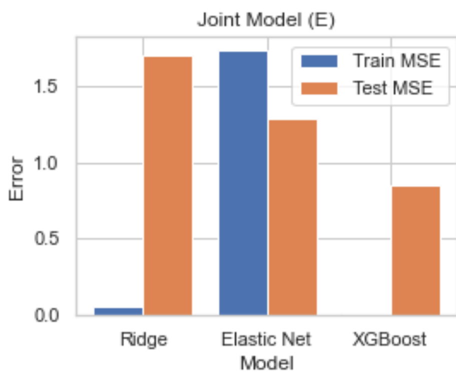
```
for symbol in symbols:

    # train-test split
    X = stocks_joint.drop([symbol + "_next_open" for symbol in symbols], axis
    y = stocks_joint[symbol + "_next_open"]

    X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.80

    # train models
    ridge_model, ridge_train, ridge_test = train_ridge_regressor(X,       y,
                                                                 X_train, y_t
                                                                 X_test,  y_t

    elast_model, elast_train, elast_test = train_elastic_net_regressor(X,
                                                                       X_trai
                                                                       X_test

    xgb_model,   xgb_train,   xgb_test   = train_xgboost_regressor(X,       y
                                                                   X_train, y
                                                                   X_test,  y

    fig, ax = plt.subplots(figsize=(4, 3))
    train = ax.bar(np.arange(0,3),     [ridge_train, elast_train, xgb_train],
                   label="Train MSE")
    test  = ax.bar(np.arange(0,3)+0.4, [ridge_test,  elast_test,  xgb_test],
                   label="Test MSE")

    ax.set_xlabel("Model")
    ax.set_ylabel("Error")
    ax.set_title("Joint Model (%s)" % symbol)
    ax.set_xticks(np.arange(0,3) + 0.4/2)
    ax.set_xticklabels(["Ridge", "Elastic Net", "XGBoost"])
    ax.legend()

    plt.show()
```

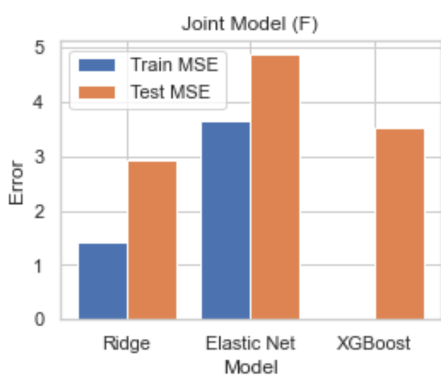


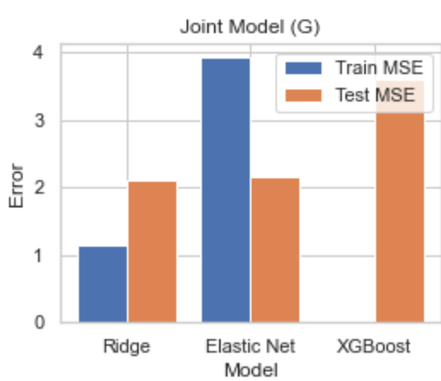


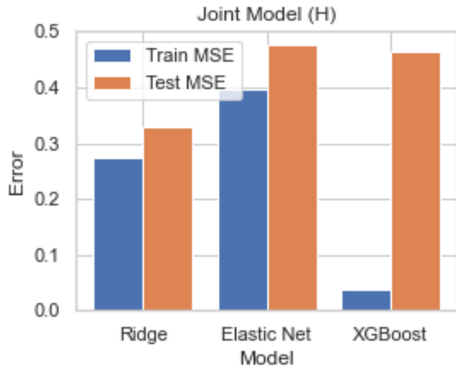

Joint Model (C)
Train MSE
Test MSE
Error
0.0
0.5
1.0
1.5
2.0
Ridge
Elastic Net
XGBoost
Model

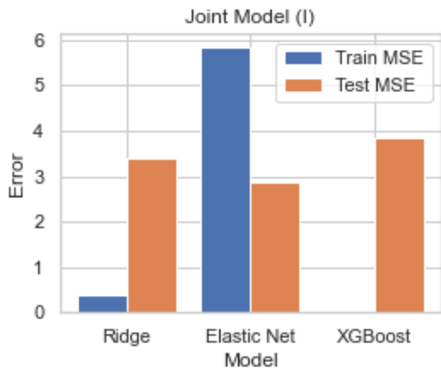

Joint Model (D)
Train MSE
Test MSE
Error
0
2
4
6
Ridge
Elastic Net
XGBoost
Model

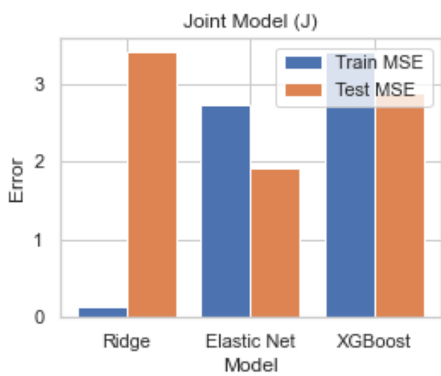

Joint Model (E)
Train MSE
Test MSE
Error
0.0
0.5
1.0
1.5
Ridge
Elastic Net
XGBoost
Model


Joint Model (F)
Train MSE
Test MSE
Error
0
1
2
3
4
5
Ridge
Elastic Net
XGBoost
Model


Joint Model (G)
Train MSE
Test MSE
Error
0
1
2
3
4
Ridge
Elastic Net
XGBoost
Model







Clustered Models

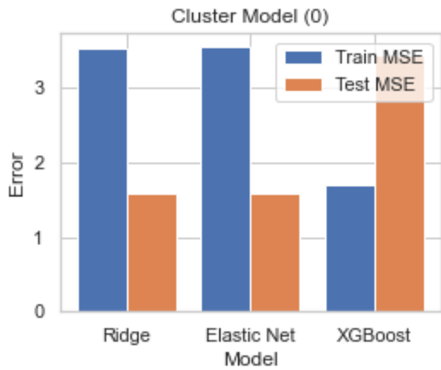
Let's set up our clusters. We'll place A-F and J in the first cluster, G and I in the second cluster, and H in the third cluster. This is close to our initial clustering, but combines some stocks that were more similar despite being singleton clusters.

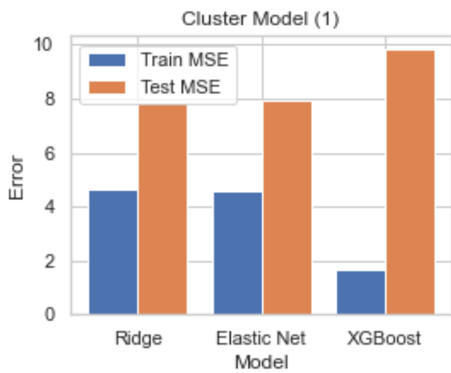
In [42]:

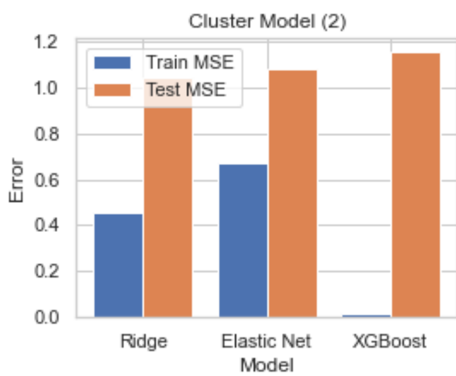
```
clusters = {0 : ["A", "B", "C", "D", "E", "F", "J"], 1 : ["G", "I"], 2 : "H"}

stocks = {symbol : raw[raw.symbol == symbol].drop("symbol", axis=1).astype('f

for symbol in symbols:
    stocks[symbol] = stocks[symbol].resample('1D').bfill()
    stocks[symbol] = stocks[symbol].drop(["high", "low", "close", "average"],

    stocks[symbol]["sma5"]  = stocks[symbol].open.rolling(5).mean().shift(1)
    stocks[symbol]["sma10"] = stocks[symbol].open.rolling(10).mean().shift(1)
    stocks[symbol]["sma20"] = stocks[symbol].open.rolling(20).mean().shift(1)
    stocks[symbol]["ema8"]  = stocks[symbol].open.ewm(8).mean().shift(1)
    stocks[symbol]["ema20"] = stocks[symbol].open.ewm(20).mean().shift(1)

    stocks[symbol] = stocks[symbol][20:]

    stocks[symbol]["next_open"] = stocks[symbol].open.shift(-1).astype('float

    stocks[symbol] = stocks[symbol][:-1]
```

In [43]:

```
for i, cluster in clusters.items():

    # train-test split
    stocks_clust = pd.concat([stocks[symbol] for symbol in cluster])

    X = stocks_clust.drop("next_open", axis=1)
    y = stocks_clust.next_open

    X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.80

    # train models
    ridge_model, ridge_train, ridge_test = train_ridge_regressor(X,       y,
                                                                  X_train, y_t
                                                                  X_test,  y_t

    elast_model, elast_train, elast_test = train_elastic_net_regressor(X,
                                                                        X_trai
                                                                        X_test

    xgb_model,   xgb_train,   xgb_test   = train_xgboost_regressor(X,       y
                                                                    X_train, y
                                                                    X_test,  y

    fig, ax = plt.subplots(figsize=(4, 3))
    train = ax.bar(np.arange(0,3),     [ridge_train, elast_train, xgb_train],
                   label="Train MSE")
    test  = ax.bar(np.arange(0,3)+0.4, [ridge_test,  elast_test,  xgb_test],
                   label="Test MSE")

    ax.set_xlabel("Model")
    ax.set_ylabel("Error")
    ax.set_title("Cluster Model (%d)" % i)
    ax.set_xticks(np.arange(0,3) + 0.4/2)
    ax.set_xticklabels(["Ridge", "Elastic Net", "XGBoost"])
    ax.legend()

    plt.show()
```







These cluster models perform quite well given their simplicity and interpretability, with ridge models tending to lead the way.

Before selecting an ultimate model, I want to also try to use lag variables and see how well we can do on that.

Models Using Lag Variables

Let's train the same three types of models (independent, joint, and clustered) and see our results.

Independent Models

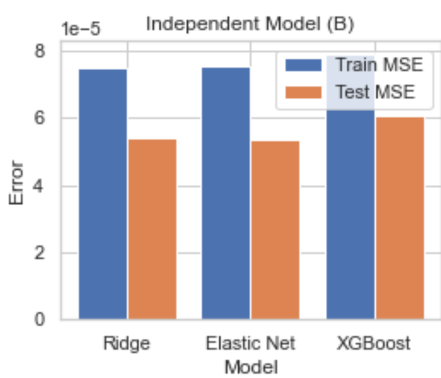
In [44]:

```
stocks = {symbol : raw[raw.symbol == symbol].drop(["symbol", "high", "low", "
lags   = np.arange(1, 15)

for symbol in symbols:

    stocks[symbol] = stocks[symbol].resample('1D').bfill()

    for lag in lags:
        stocks[symbol]["lag" + str(lag)] = (stocks[symbol].open.shift(lag) -
```

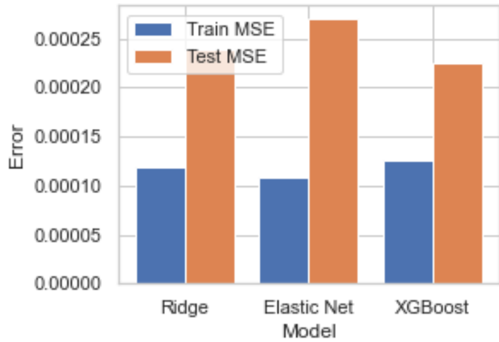
```
    stocks[symbol]["lead1"] = (stocks[symbol].open.shift(-1) - stocks[symbol]

    stocks[symbol] = stocks[symbol][max(lags):-1]
```

In [45]:

```
for symbol in symbols:

    # train-test split
    X = stocks[symbol].drop(["open", "lead1"], axis=1)
    y = stocks[symbol].lead1

    X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.80

    # train models
    ridge_model, ridge_train, ridge_test = train_ridge_regressor(X,       y,
                                                                  X_train, y_t
                                                                  X_test,  y_t

    elast_model, elast_train, elast_test = train_elastic_net_regressor(X,
                                                                        X_trai
                                                                        X_test

    xgb_model,   xgb_train,   xgb_test   = train_xgboost_regressor(X,       y
                                                                    X_train, y
                                                                    X_test,  y

    fig, ax = plt.subplots(figsize=(4, 3))
    train = ax.bar(np.arange(0,3),      [ridge_train, elast_train, xgb_train],
                   label="Train MSE")
    test  = ax.bar(np.arange(0,3)+0.4, [ridge_test,  elast_test,  xgb_test],
                   label="Test MSE")

    ax.set_xlabel("Model")
    ax.set_ylabel("Error")
    ax.set_title("Independent Model (%s)" % symbol)
    ax.set_xticks(np.arange(0,3) + 0.4/2)
    ax.set_xticklabels(["Ridge", "Elastic Net", "XGBoost"])
    ax.legend()

    plt.show()
```

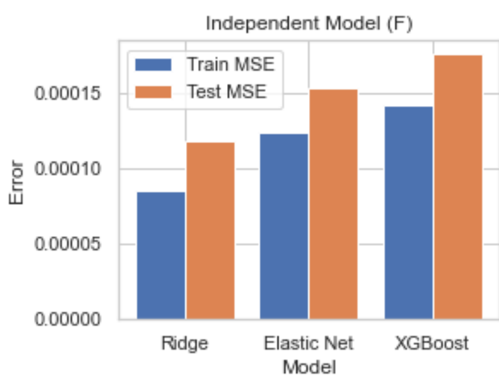


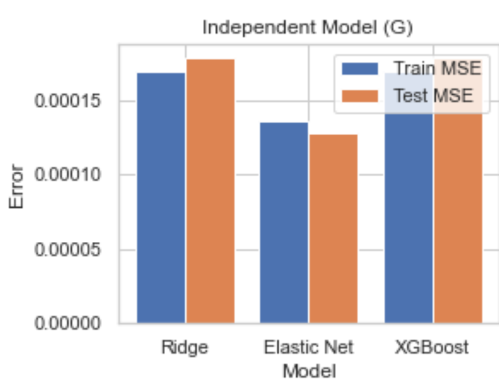
Independent Model (C)

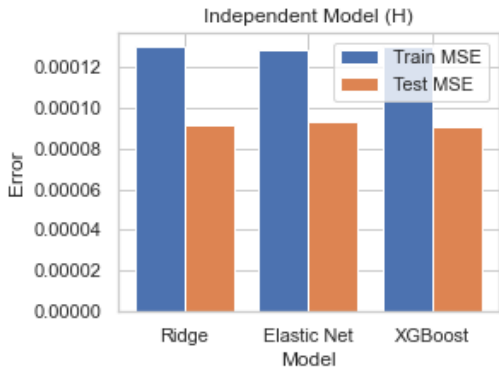

Train MSE
Test MSE
Error
0.00000
0.00005
0.00010
0.00015
0.00020
0.00025
Ridge
Elastic Net
XGBoost
Model

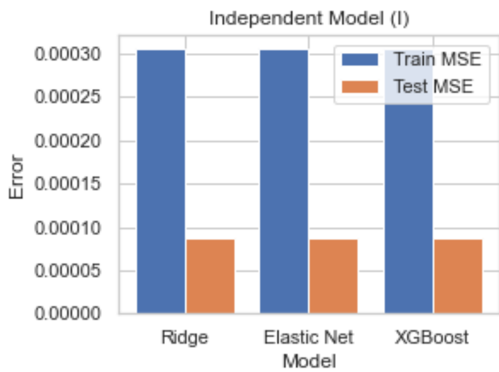

Independent Model (D)
Train MSE
Test MSE
Error
0.00000
0.00002
0.00004
0.00006
0.00008
0.00010
0.00012
Ridge
Elastic Net
XGBoost
Model

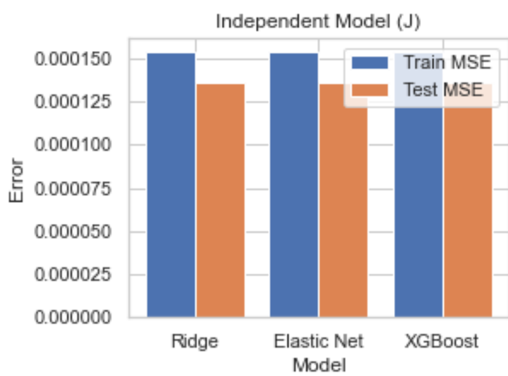

Independent Model (E)
Train MSE
Test MSE
Error
0.00000
0.00002
0.00004
0.00006
0.00008
0.00010
0.00012
Ridge
Elastic Net
XGBoost
Model


Independent Model (F)
Train MSE
Test MSE
Error
0.00000
0.00005
0.00010
0.00015
Ridge
Elastic Net
XGBoost
Model


Independent Model (G)
Train MSE
Test MSE
Error
0.00000
0.00005
0.00010
0.00015
Ridge
Elastic Net
XGBoost
Model







In general, all three model families perform well, with Ridge and ElasticNet edging out the XGBoost model family for some stocks.

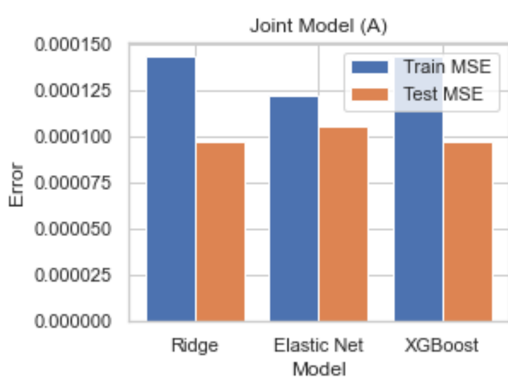
Joint Models

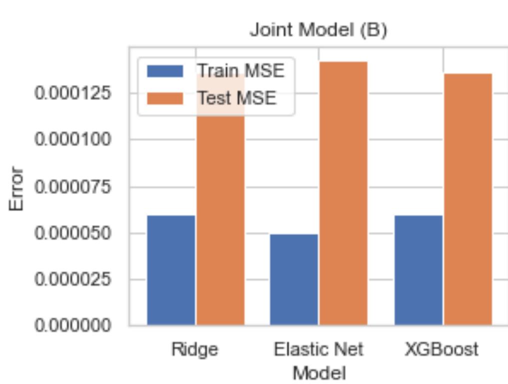
In [46]:

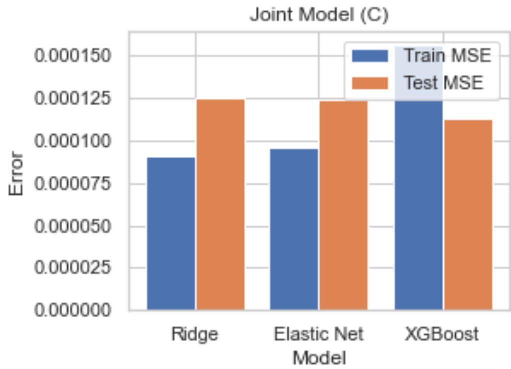
```
stocks = {symbol : raw[raw.symbol == symbol].drop(["symbol", "high", "low", "
lags   = np.arange(1, 15)

for symbol in symbols:

    stocks[symbol] = stocks[symbol].resample('1D').bfill()

    for lag in lags:
        stocks[symbol][symbol + "_lag" + str(lag)] = (stocks[symbol].open.shi

    stocks[symbol][symbol + "_lead1"] = (stocks[symbol].open.shift(-1) - stoc

    stocks[symbol] = stocks[symbol].rename(columns={"open": symbol + "_open"}

    stocks[symbol] = stocks[symbol][max(lags):-1]

# concatenate all stocks into one df
stocks_joint = pd.concat([stocks[symbol] for symbol in symbols], axis=1)
```

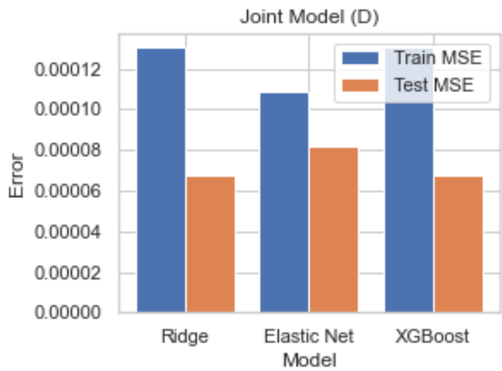
In [47]:

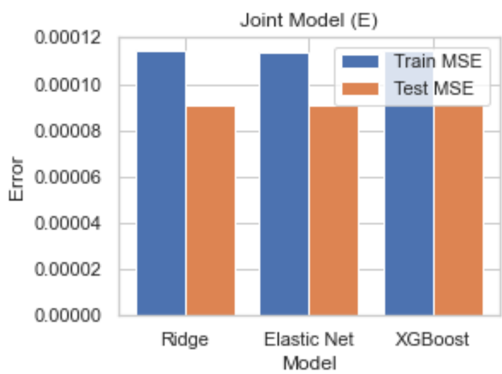
```
for symbol in symbols:

    # train-test split
    X = stocks_joint.drop([*[symbol + "_lead1" for symbol in symbols],
                           *[symbol + "_open"  for symbol in symbols]], axis=
    y = stocks_joint[symbol + "_lead1"]

    X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.80

    # train models
    ridge_model, ridge_train, ridge_test = train_ridge_regressor(X,       y,
                                                                 X_train, y_t
                                                                 X_test,  y_t

    elast_model, elast_train, elast_test = train_elastic_net_regressor(X,
                                                                       X_trai
                                                                       X_test

    xgb_model,   xgb_train,   xgb_test   = train_xgboost_regressor(X,       y
                                                                   X_train, y
                                                                   X_test,  y

    fig, ax = plt.subplots(figsize=(4, 3))
    train = ax.bar(np.arange(0,3),     [ridge_train, elast_train, xgb_train],
                   label="Train MSE")
    test  = ax.bar(np.arange(0,3)+0.4, [ridge_test,  elast_test,  xgb_test],
                   label="Test MSE")

    ax.set_xlabel("Model")
    ax.set_ylabel("Error")
    ax.set_title("Joint Model (%s)" % symbol)
    ax.set_xticks(np.arange(0,3) + 0.4/2)
    ax.set_xticklabels(["Ridge", "Elastic Net", "XGBoost"])
    ax.legend()

    plt.show()
```

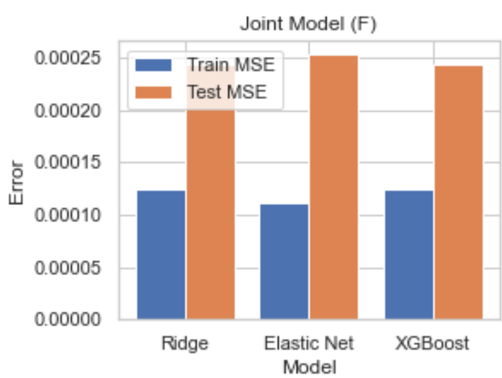


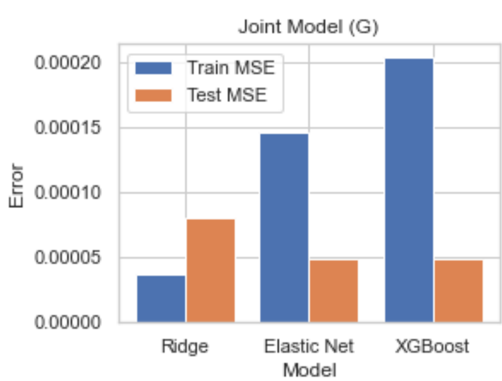


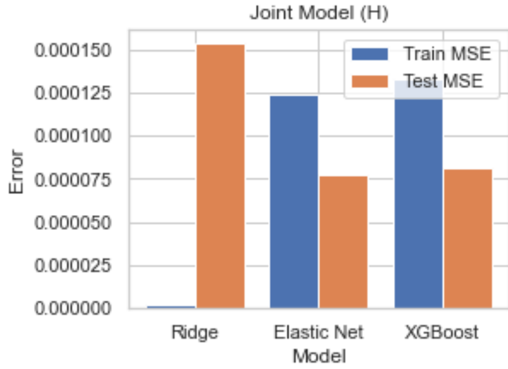

Joint Model (C)
Train MSE
Test MSE
0.000150
0.000125
0.000100
0.000075
0.000050
0.000025
0.000000
Error
Ridge
Elastic Net
XGBoost
Model

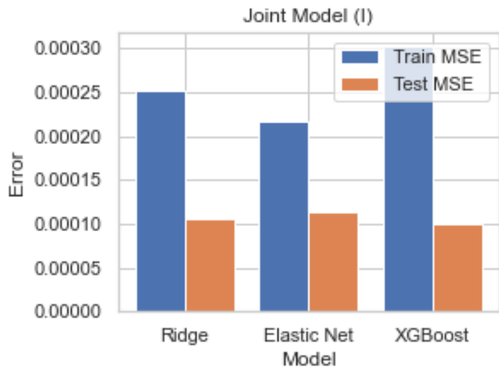

Joint Model (D)
Train MSE
Test MSE
0.00012
0.00010
0.00008
0.00006
0.00004
0.00002
0.00000
Error
Ridge
Elastic Net
XGBoost
Model

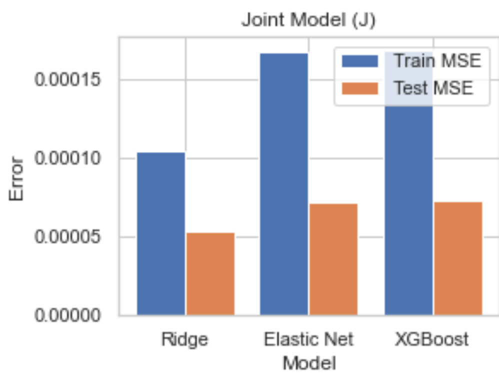

Joint Model (E)
Train MSE
Test MSE
0.00012
0.00010
0.00008
0.00006
0.00004
0.00002
0.00000
Error
Ridge
Elastic Net
XGBoost
Model


Joint Model (F)
Train MSE
Test MSE
0.00025
0.00020
0.00015
0.00010
0.00005
0.00000
Error
Ridge
Elastic Net
XGBoost
Model


Joint Model (G)
Train MSE
Test MSE
0.00020
0.00015
0.00010
0.00005
0.00000
Error
Ridge
Elastic Net
XGBoost
Model







Compared to the independent models, the joint models have slightly lower error for all three model families. In general, ElasticNet more commonly edges out the other two model families, with some small exceptions.

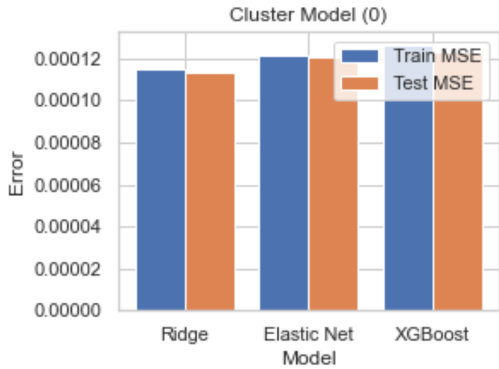
Clustered Models

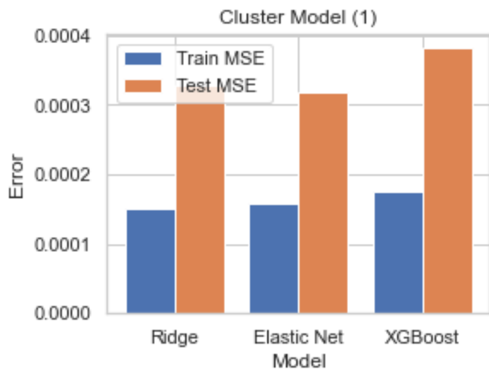
In [48]:

```
clusters = {0 : ["A", "B", "C", "D", "E", "F", "J"], 1 : ["G", "I"], 2 : "H"}

stocks = {symbol : raw[raw.symbol == symbol].drop(["symbol", "high", "low", "
lags   = np.arange(1, 15)

for symbol in symbols:

    stocks[symbol] = stocks[symbol].resample('1D').bfill()

    for lag in lags:
        stocks[symbol]["lag" + str(lag)] = (stocks[symbol].open.shift(lag) - 

    stocks[symbol]["lead1"] = (stocks[symbol].open.shift(-1) - stocks[symbol]

    stocks[symbol] = stocks[symbol][max(lags):-1]
```

In [49]:

```
for i, cluster in clusters.items():

    # train-test split
    stocks_clust = pd.concat([stocks[symbol] for symbol in cluster])

    X = stocks_clust.drop("lead1", axis=1)
    y = stocks_clust.lead1

    X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.80

    # train models
    ridge_model, ridge_train, ridge_test = train_ridge_regressor(X,       y,
                                                                  X_train, y_t
                                                                  X_test,  y_t

    elast_model, elast_train, elast_test = train_elastic_net_regressor(X,
                                                                        X_trai
                                                                        X_test

    xgb_model,   xgb_train,   xgb_test   = train_xgboost_regressor(X,       y
                                                                    X_train, y
                                                                    X_test,  y

    fig, ax = plt.subplots(figsize=(4, 3))
    train = ax.bar(np.arange(0,3),     [ridge_train, elast_train, xgb_train],
                   label="Train MSE")
    test  = ax.bar(np.arange(0,3)+0.4, [ridge_test,  elast_test,  xgb_test],
                   label="Test MSE")

    ax.set_xlabel("Model")
    ax.set_ylabel("Error")
    ax.set_title("Cluster Model (%d)" % i)
    ax.set_xticks(np.arange(0,3) + 0.4/2)
    ax.set_xticklabels(["Ridge", "Elastic Net", "XGBoost"])
    ax.legend()

    plt.show()
```





Model Selection

Ultimately, I want to pick one model from the market indicator models and another model from the lag variable models. I want to pick on from each since their MSE's are on different scales, so a visual comparison of how they perform might help me more clearly identify which model is best.

I'd like to pick a clustered model from each since I think it has a relatively strong benefit of interpretability (grouping together stocks with similar trends and correlations). I'll also pick the ridge model since it seemed to be the best on average for clustered models.

Clustered Ridge Regression Models Using Market

Indicators

In [50]:

```
clusters = {0 : ["A", "B", "C", "D", "E", "F", "J"], 1 : ["G", "I"], 2 : "H"}

stocks = {symbol : raw[raw.symbol == symbol].drop("symbol", axis=1).astype('f

for symbol in symbols:
    stocks[symbol] = stocks[symbol].resample('1D').bfill()
    stocks[symbol] = stocks[symbol].drop(["high", "low", "close", "average"],

    stocks[symbol]["sma5"]  = stocks[symbol].open.rolling(5).mean().shift(1)
    stocks[symbol]["sma10"] = stocks[symbol].open.rolling(10).mean().shift(1)
    stocks[symbol]["sma20"] = stocks[symbol].open.rolling(20).mean().shift(1)
    stocks[symbol]["ema8"]  = stocks[symbol].open.ewm(8).mean().shift(1)
    stocks[symbol]["ema20"] = stocks[symbol].open.ewm(20).mean().shift(1)

    stocks[symbol] = stocks[symbol][20:]

    stocks[symbol]["next_open"] = stocks[symbol].open.shift(-1).astype('float

    stocks[symbol] = stocks[symbol][:-1]
```

In [51]:

```
models = {}

for i, cluster in clusters.items():

    # train-test split
    stocks_clust = pd.concat([stocks[symbol] for symbol in cluster])

    X = stocks_clust.drop("next_open", axis=1)
    y = stocks_clust.next_open

    X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.80

    # train models
    ridge_model, ridge_train, ridge_test = train_ridge_regressor(X,       y,
                                                                 X_train, y_t
                                                                 X_test,  y_t

    models[i] = ridge_model
```

In [52]:

```
import copy

stocks_pred = copy.deepcopy(stocks)
pred_days = 10

for i, cluster in clusters.items():

    for symbol in cluster:

        for day in range(pred_days):

            new_df = pd.DataFrame([[models[i].predict(stocks_pred[symbol].dro
                                    np.NaN, np.NaN, np.NaN, np.NaN, np.NaN, n
                                  columns=stocks_pred[symbol].columns,
                                  index  =stocks_pred[symbol].index[-1:]+pd.D

            stocks_pred[symbol] = stocks_pred[symbol].append(new_df)

            stocks_pred[symbol].iloc[-1, 1:-1] = [stocks_pred[symbol].open.ro
                                                   stocks_pred[symbol].open.ro
                                                   stocks_pred[symbol].open.ro
                                                   stocks_pred[symbol].open.ew
                                                   stocks_pred[symbol].open.ew

stocks_pred_1 = stocks_pred
```

Clustered Ridge Regression Models Using Lag Variables

In [53]:

```
clusters = {0 : ["A", "B", "C", "D", "E", "F", "J"], 1 : ["G", "I"], 2 : "H"}

stocks = {symbol : raw[raw.symbol == symbol].drop(["symbol", "high", "low", "
lags   = np.arange(1, 15)

for symbol in symbols:

    stocks[symbol] = stocks[symbol].resample('1D').bfill()

    for lag in lags:
        stocks[symbol]["lag" + str(lag)] = (stocks[symbol].open.shift(lag) - 

    stocks[symbol]["lead1"] = (stocks[symbol].open.shift(-1) - stocks[symbol]

    stocks[symbol] = stocks[symbol][max(lags):-1]
```

In [54]:

```
models = {}

for i, cluster in clusters.items():

    # train-test split
    stocks_clust = pd.concat([stocks[symbol] for symbol in cluster])

    X = stocks_clust.drop(["open", "lead1"], axis=1)
    y = stocks_clust.lead1

    X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.80

    # train models
    ridge_model, ridge_train, ridge_test = train_ridge_regressor(X,       y,
                                                                 X_train, y_t
                                                                 X_test,  y_t

    models[i] = ridge_model
```

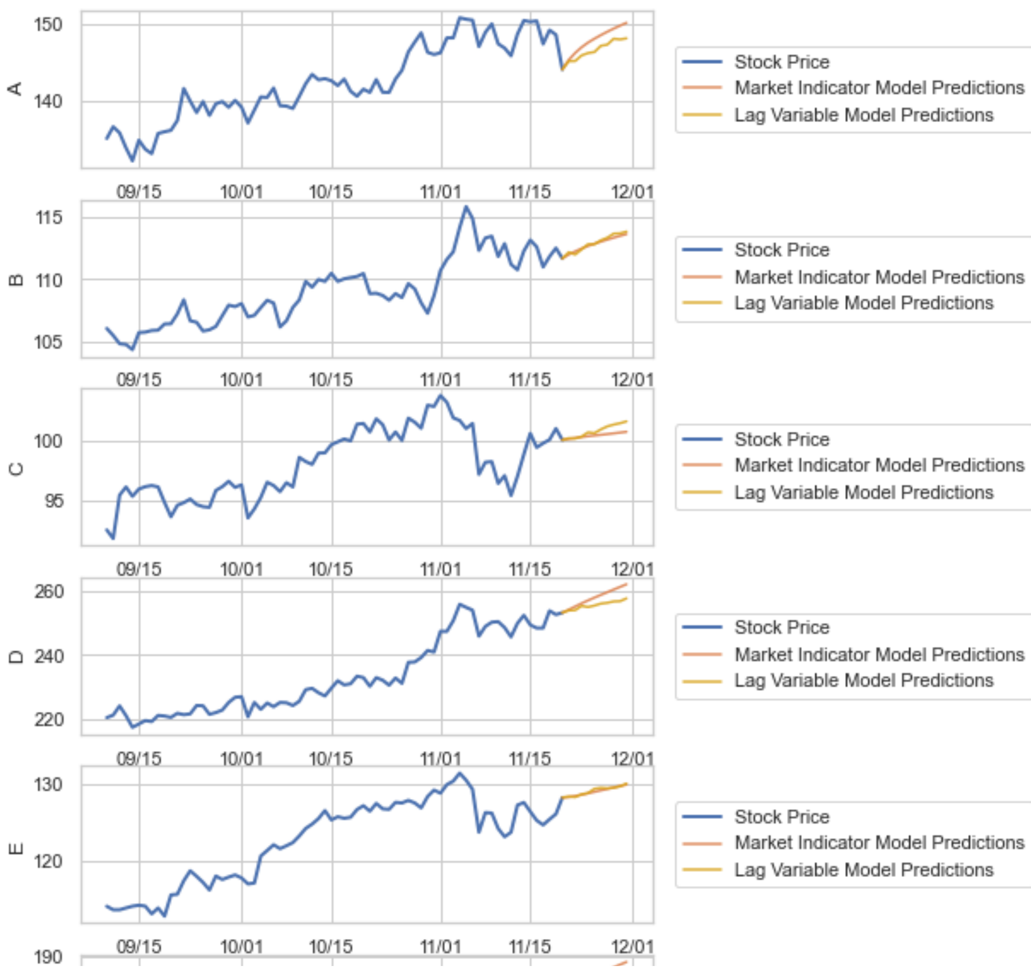
In [55]:

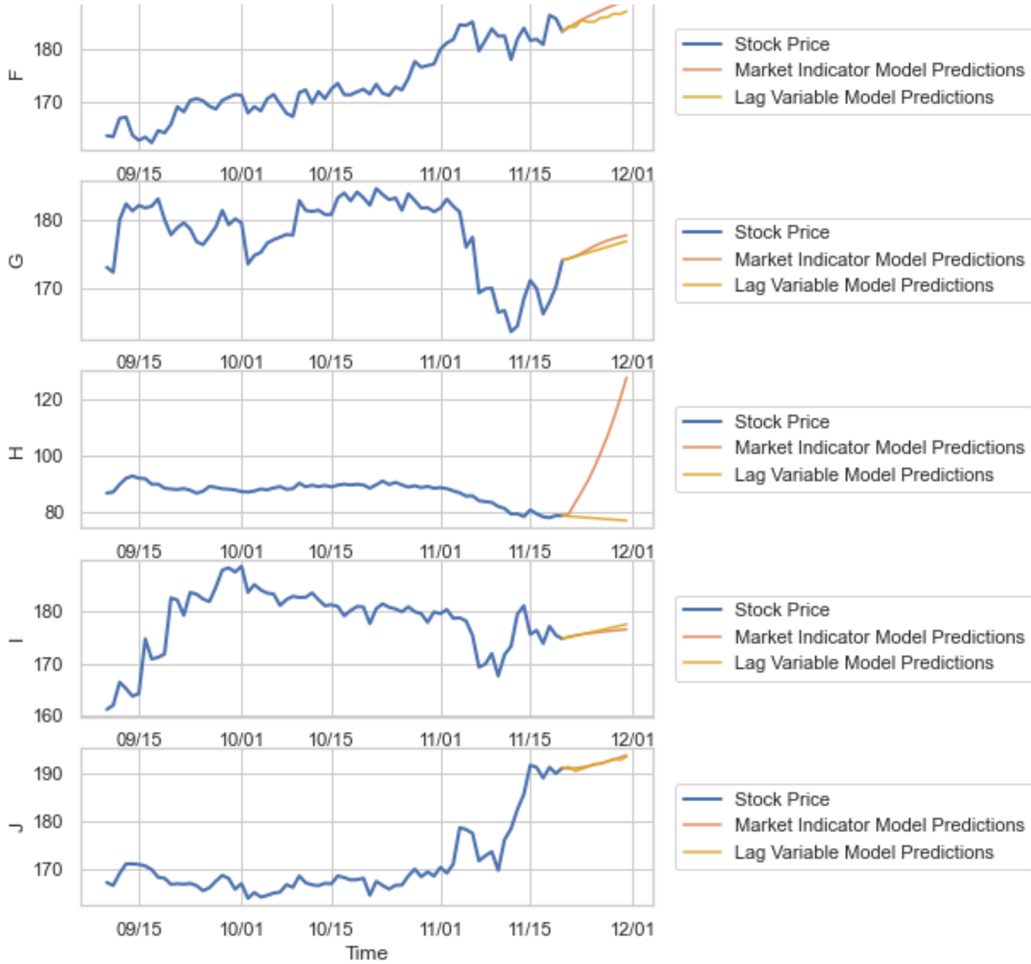
```
import copy

stocks_pred = copy.deepcopy(stocks)
pred_days = 10

for i, cluster in clusters.items():

    for symbol in cluster:

        for day in range(pred_days):

            pred_lead_pct = models[i].predict(stocks_pred[symbol].drop(["open
            prev_open     = stocks_pred[symbol].open.iloc[-1]
            new_open      = prev_open*(1 + pred_lead_pct)
            change        = new_open - prev_open

            new_df = pd.DataFrame([[new_open, *np.repeat(np.NaN, len(lags) + 
                                  columns=stocks_pred[symbol].columns,
                                  index=stocks_pred[symbol].index[-1:]+pd.Dat

            stocks_pred[symbol] = stocks_pred[symbol].append(new_df)
```

```
            for k, lag in enumerate(lags):
                stocks_pred[symbol].iloc[-1, k+1] = (stocks_pred[symbol].open
                                                     stocks_pred[symbol].open



stocks_pred_2 = stocks_pred
```

In [65]:

```
from matplotlib.dates import DateFormatter

fig, axs = plt.subplots(10, 1, figsize=(6, 20))
axs = axs.flatten()

date_form = DateFormatter("%m/%d")

for i, symbol in enumerate(symbols):
    axs[i].plot(stocks[symbol].index, stocks[symbol].open,
                marker='o', markersize=0, lw=2, label="Stock Price")
    axs[i].plot(stocks_pred_1[symbol].index[-pred_days-1:], stocks_pred_1[sym
                marker='o', markersize=0, alpha=0.9, lw=1.5, label="Market In
    axs[i].plot(stocks_pred_2[symbol].index[-pred_days-1:], stocks_pred_2[sym
                marker='o', markersize=0, alpha=0.9, lw=1.5, label="Lag Varia
                color="goldenrod")
    axs[i].set_xlabel("Time")
    axs[i].set_ylabel(symbol)
    axs[i].xaxis.set_major_formatter(date_form)
    axs[i].legend(bbox_to_anchor=(1.04,0.5), loc="center left", borderaxespad
```





Both models look very similar, except for the prediction on stock H using the market indicators model, which quickly diverges, likely because there is little data where the price is lower and we have to make predictions from this point as well.

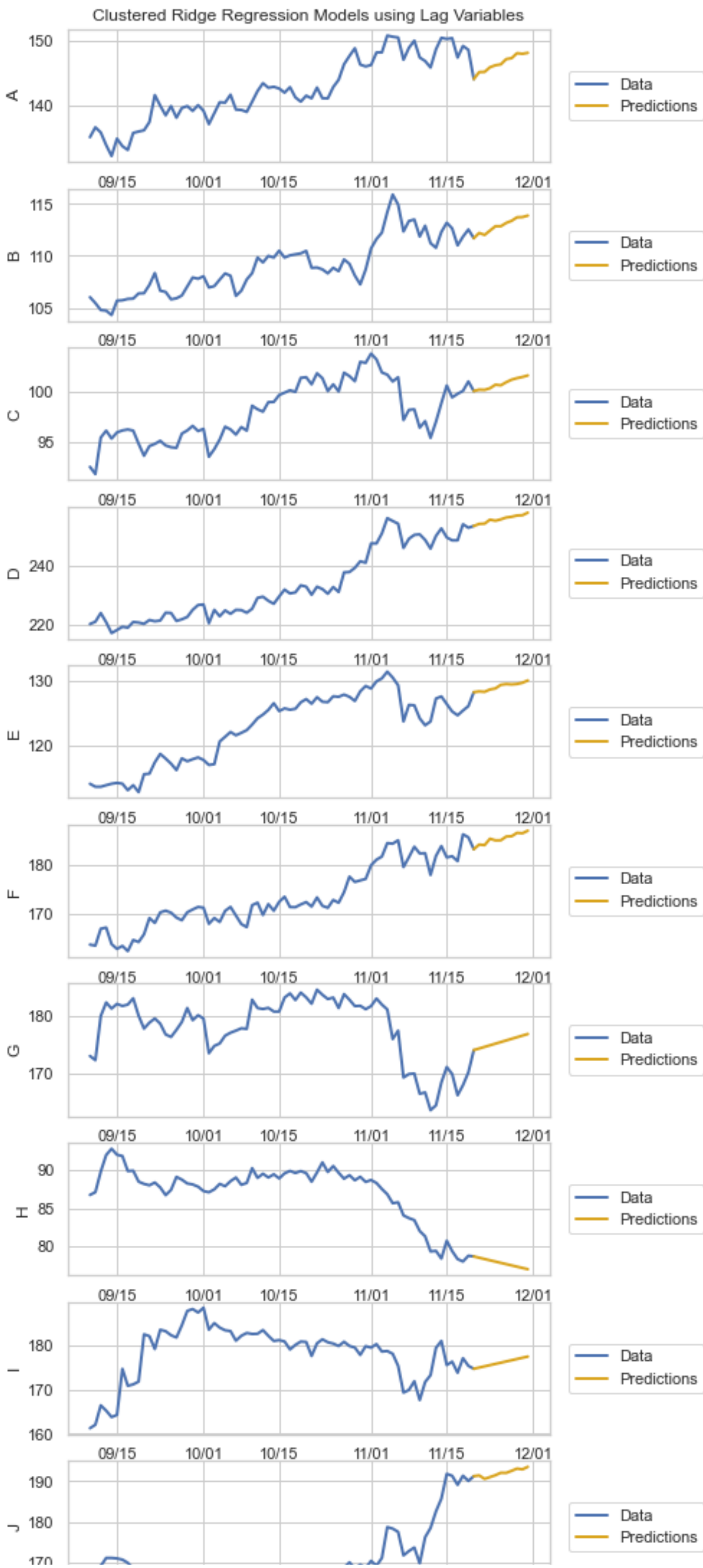
As a result, we'll select the clustered ridge regression models using lag variables. We'll add some additional visualization of the final model.

Model Predictions

In [66]:

```
fig, axs = plt.subplots(10, 1, figsize=(6, 20))
axs = axs.flatten()

date_form = DateFormatter("%m/%d")

for i, symbol in enumerate(symbols):
    if i == 0:
        axs[i].set_title("Clustered Ridge Regression Models using Lag Variabl
    axs[i].plot(stocks[symbol].index, stocks[symbol].open, lw=2, label="Data"
    axs[i].plot(stocks_pred_2[symbol].index[-pred_days-1:], stocks_pred_2[sym
                lw=2, color="goldenrod", label="Predictions")
    axs[i].set_xlabel("Time")
    axs[i].set_ylabel(symbol)
```

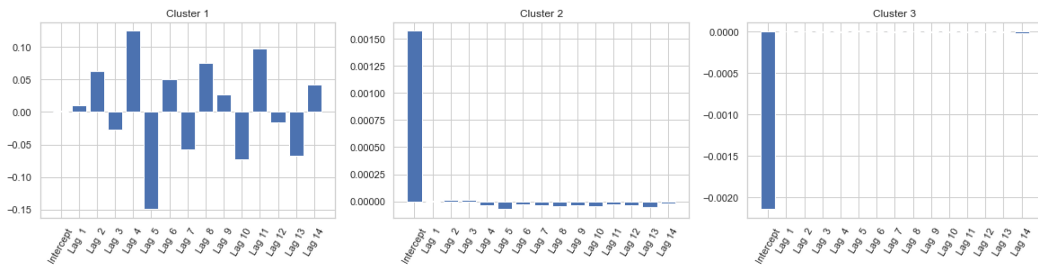
```
    axs[i].set_ylabel(symbol)
    axs[i].xaxis.set_major_formatter(date_form)
    axs[i].legend(bbox_to_anchor=(1.04,0.5), loc="center left", borderaxespad
```



Model Parameters

In [67]:

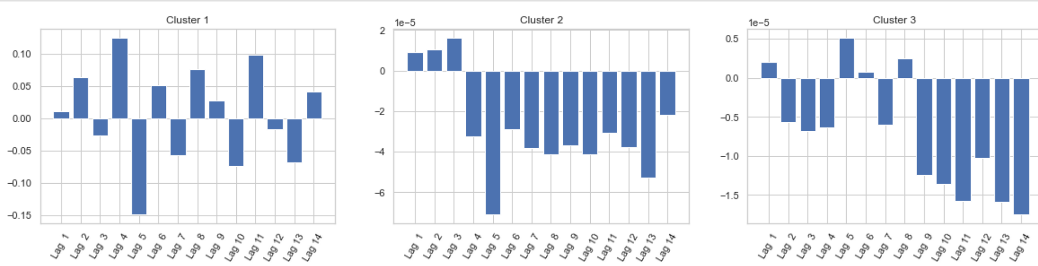
```
fig, axs = plt.subplots(1, 3, figsize=(20, 4))

for i in range(3):
    coefs = [models[i].intercept_, *models[i].coef_]
    axs[i].bar(range(len(coefs)), coefs)
    axs[i].set_xticks(range(len(coefs)))
    axs[i].set_xticklabels(["Intercept", *["Lag %2d" % i for i in np.arange(1
    axs[i].set_title("Cluster %d" %(i+1))
```



Model Parameters (Without Intercept)

In [68]:

```
fig, axs = plt.subplots(1, 3, figsize=(20, 4))

for i in range(3):
    coefs = models[i].coef_
    axs[i].bar(range(len(coefs)), coefs)
    axs[i].set_xticks(range(len(coefs)))
    axs[i].set_xticklabels(["Lag %2d" % i for i in np.arange(1, 15)], rotatio
    axs[i].set_title("Cluster %d" %(i+1))
```



In [69]:

```
import csv

filename = 'preds/preds.csv'

f = open(filename, 'w')

writer = csv.writer(f)
writer.writerow(["id" , "open"])

stocks_pred_out = {symbol : stocks_pred_2[symbol].open.iloc[-pred_days:] for

for symbol in symbols:

    for i, index in enumerate(stocks_pred_out[symbol].index[:-1]):

        # set up the grid of 5-secondly daily prediction times
        date_range = pd.date_range(start=stocks_pred_out[symbol].index[i] +
                                         pd.DateOffset(hours=6,  minutes=0,
                                   end  =stocks_pred_out[symbol].index[i] +
                                         pd.DateOffset(hours=12, minutes=59
                                   freq ='5S')

        # linearly interpolate the daily predictions to 5-secondly prediction
        p0 = stocks_pred_out[symbol].iloc[i]
        pf = stocks_pred_out[symbol].iloc[i+1]
        n  = len(date_range)

        df = pd.DataFrame(data=[p0 + i/(n+1) * (pf - p0) for i in range(n)],
                          index=date_range)

        # print out the linearly interpolated predctions
        for index, row in df.iterrows():
            writer.writerow([symbol + "-" +
                             str(i) + "-" +
                             ("%02d" % index.hour) + ":" + ("%02d" % index.mi
                             "%.3f" % row])
f.close()
```