JPX Tokyo Stock Exchange Prediction

1 | Competition Overview

Success in any financial market requires one to identify solid investments. When a stock or derivative is undervalued, it makes sense to buy. If it's overvalued, perhaps it's time to sell. While these finance decisions were historically made manually by professionals, technology has ushered in new opportunities for retail investors. Data scientists, specifically, may be interested to explore quantitative trading, where decisions are executed programmatically based on predictions from trained models. In this competition, financial data for the Japanese market will be provided, allowing retail investors to analyze the market to the fullest extent.

Japan Exchange Group, Inc. (JPX) is a holding company operating one of the largest stock exchanges in the world, Tokyo Stock Exchange (TSE), and derivatives exchanges Osaka Exchange (OSE) and Tokyo Commodity Exchange (TOCOM). JPX is hosting this competition (https://www.kaggle.com/competitions/jpx-tokyo-stock-exchange-prediction) and is supported by AI technology company AlpacaJapan Co., Ltd.

This competition will involve building portfolios from the stocks eligible for predictions. Specifically, each participant ranks the stocks from highest to lowest expected returns and is evaluated on the difference in returns between the top and bottom 200 stocks. You'll have access to financial data from the Japanese market, such as stock information and historical stock prices to train and test your model. After the training phase is complete, the competition will compare your models against real future returns.

The training data begins in January 2017 with about 1,860 stocks with new stocks added through December 2020 for a total of 2,000 stocks. Below is a brief overview and descriptive statistics of the variables in the training data.

In [2]:
```
import warnings, gc
import numpy as np
import pandas as pd
import matplotlib.colors
import seaborn as sns
import plotly.express as px
import plotly.graph_objects as go
from plotly.subplots import make_subplots
from plotly.offline import init_notebook_mode
from datetime import datetime, timedelta
from sklearn.model_selection import TimeSeriesSplit
from sklearn.metrics import mean_squared_error,mean_absolute_error
from lightgbm import LGBMRegressor
from decimal import ROUND_HALF_UP, Decimal
warnings.filterwarnings("ignore")
import plotly.figure_factory as ff

init_notebook_mode(connected=True)
temp = dict(layout=go.Layout(font=dict(family="Franklin Gothic", size=12), width=
colors=px.colors.qualitative.Plotly

train=pd.read_csv("F:\\IIMN\\FINAL PROJECT\\jpx-tokyo-stock-exchange-prediction\\
                  parse_dates=['Date'])
stock_list=pd.read_csv("F:\\IIMN\\FINAL PROJECT\\jpx-tokyo-stock-exchange-predict

print("The training data begins on {} and ends on {}.\n".format(train.Date.min(),
display(train.describe().style.format('{:,.2f}'))
```

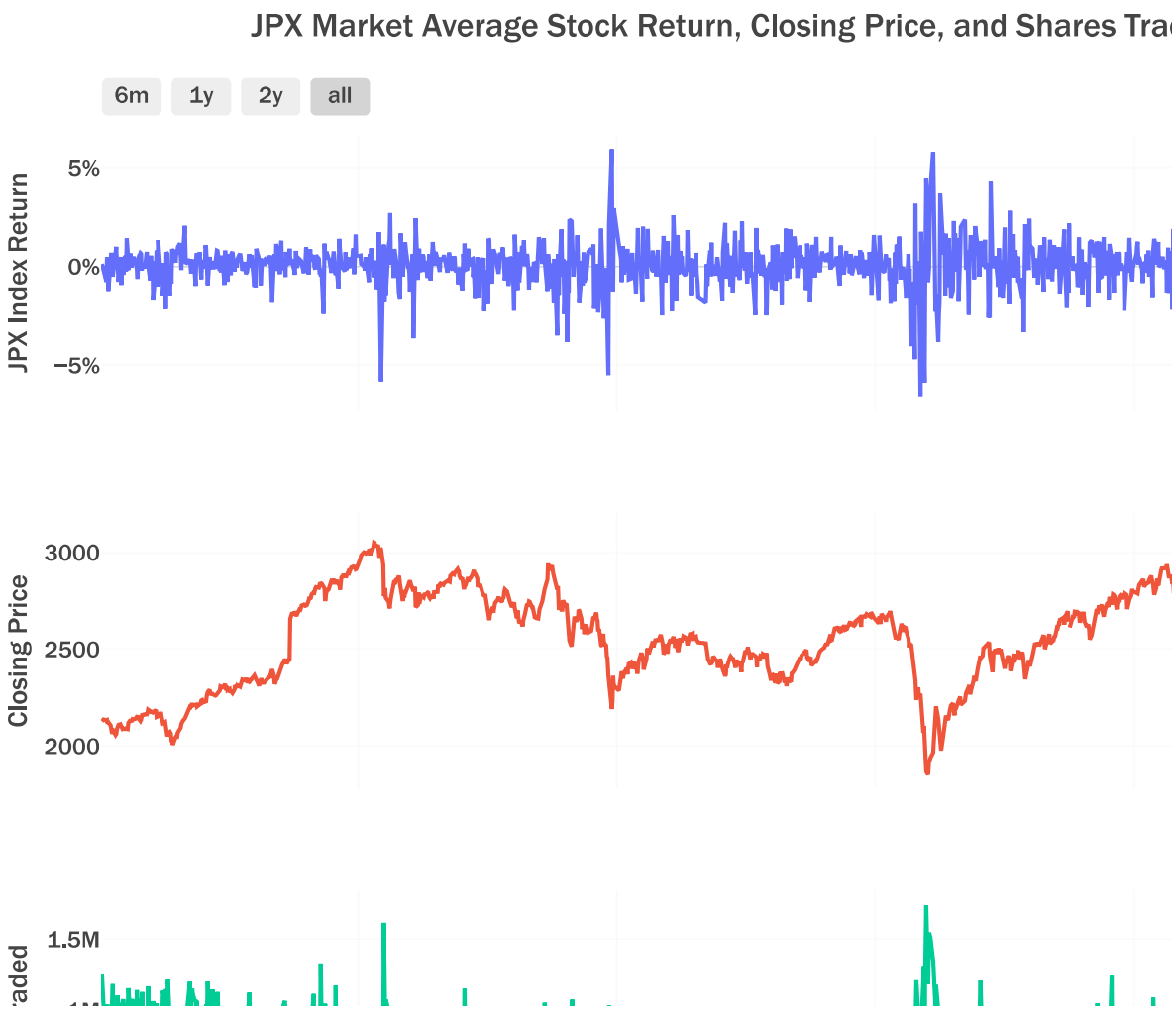
```
The training data begins on 2017-01-04 00:00:00 and ends on 2021-12-03 00:00:0
0.
```

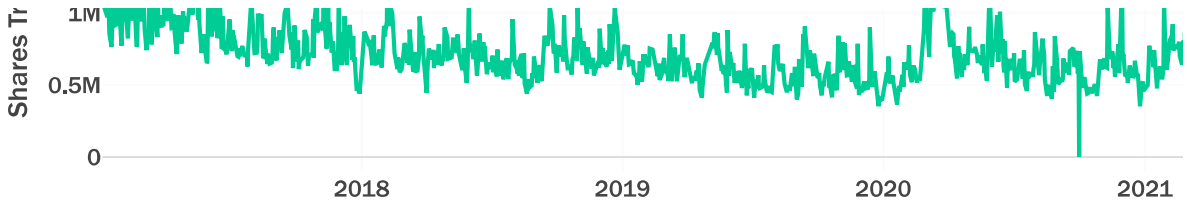
	SecuritiesCode	Open	High	Low	Close	Volume	Adjustr
count	2,332,531.00	2,324,923.00	2,324,923.00	2,324,923.00	2,324,923.00	2,332,531.00	2
mean	5,894.84	2,594.51	2,626.54	2,561.23	2,594.02	691,936.56	
std	2,404.16	3,577.19	3,619.36	3,533.49	3,576.54	3,911,255.94	
min	1,301.00	14.00	15.00	13.00	14.00	0.00	
25%	3,891.00	1,022.00	1,035.00	1,009.00	1,022.00	30,300.00	
50%	6,238.00	1,812.00	1,834.00	1,790.00	1,811.00	107,100.00	
75%	7,965.00	3,030.00	3,070.00	2,995.00	3,030.00	402,100.00	
max	9,997.00	109,950.00	110,500.00	107,200.00	109,550.00	643,654,000.00	

In [ ]:
```
```

2 | Exploratory Data Analysis

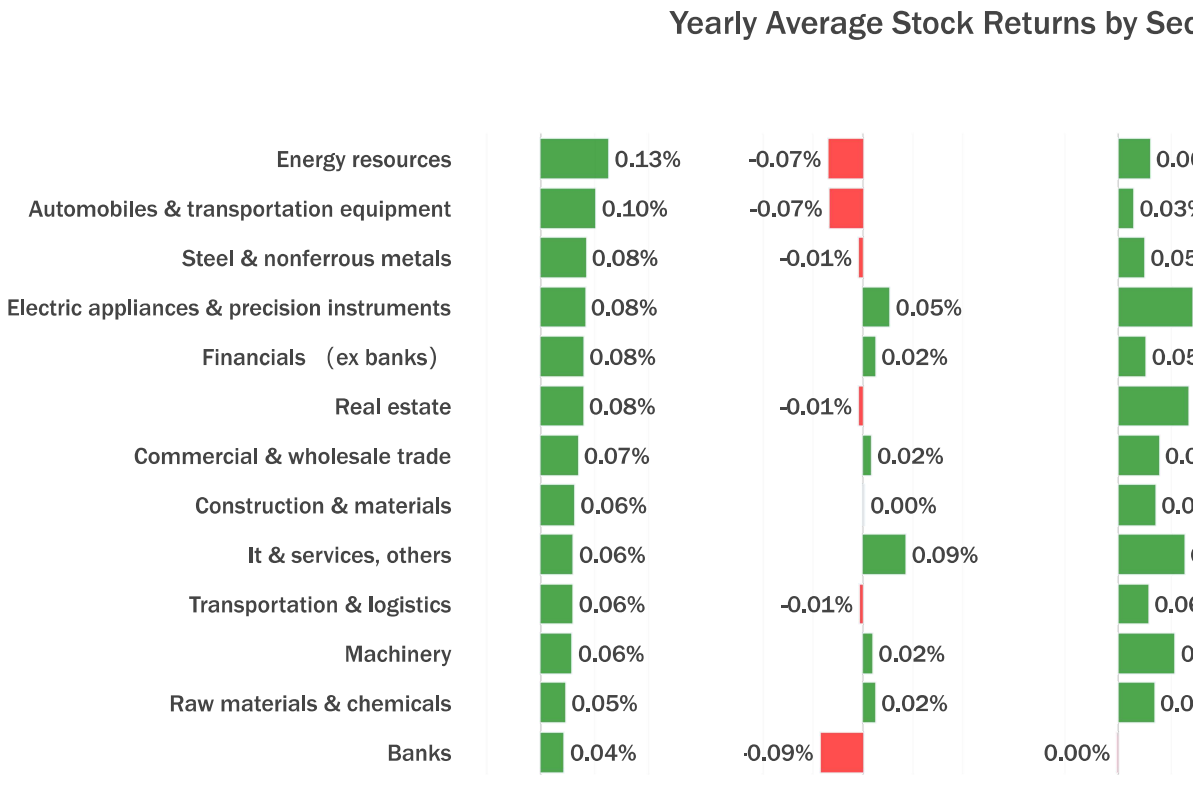
In [3]:
```
train_date=train.Date.unique()
returns=train.groupby('Date')['Target'].mean().mul(100).rename('Average Return')
close_avg=train.groupby('Date')['Close'].mean().rename('Closing Price')
vol_avg=train.groupby('Date')['Volume'].mean().rename('Volume')

fig = make_subplots(rows=3, cols=1,
                    shared_xaxes=True)
for i, j in enumerate([returns, close_avg, vol_avg]):
    fig.add_trace(go.Scatter(x=train_date, y=j, mode='lines',
                             name=j.name, marker_color=colors[i]), row=i+1, col=1
fig.update_xaxes(rangeslider_visible=False,
                 rangeselector=dict(
                     buttons=list([
                         dict(count=6, label="6m", step="month", stepmode="backwa
                         dict(count=1, label="1y", step="year", stepmode="backwar
                         dict(count=2, label="2y", step="year", stepmode="backwar
                         dict(step="all")])),
                 row=1,col=1)
fig.update_layout(template=temp,title='JPX Market Average Stock Return, Closing P
                  hovermode='x unified', height=700,
                  yaxis1=dict(title='JPX Index Return', ticksuffix='%'),
                  yaxis2_title='Closing Price', yaxis3_title='Shares Traded',
                  showlegend=False)
fig.show()
```

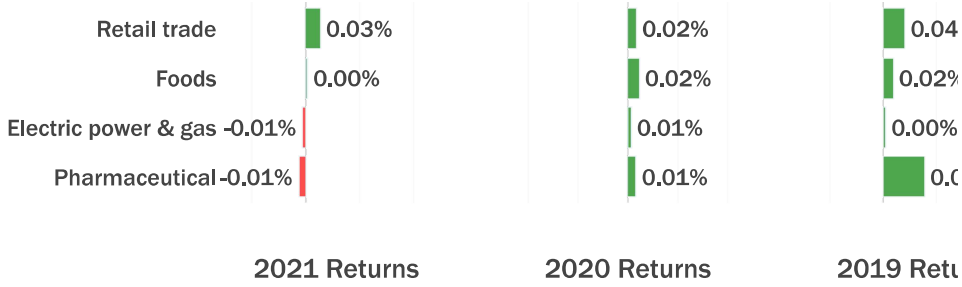




The graphs above show the market's average stock return, closing price, and shares traded since January 2017. While there has been much fluctuation over the past four years, the number of shares traded has slightly decreased from the volume in early 2017.

In [4]:
```
stock_list['SectorName']=[i.rstrip().lower().capitalize() for i in stock_list['17
stock_list['Name']=[i.rstrip().lower().capitalize() for i in stock_list['Name']]
train_df = train.merge(stock_list[['SecuritiesCode','Name','SectorName']], on='Se
train_df['Year'] = train_df['Date'].dt.year
years = {year: pd.DataFrame() for year in train_df.Year.unique()[::-1]}
for key in years.keys():
    df=train_df[train_df.Year == key]
    years[key] = df.groupby('SectorName')['Target'].mean().mul(100).rename("Avg_r
df=pd.concat((years[i].to_frame() for i in years.keys()), axis=1)
df=df.sort_values(by="Avg_return_2021")

fig = make_subplots(rows=1, cols=5, shared_yaxes=True)
for i, col in enumerate(df.columns):
    x = df[col]
    mask = x<=0
    fig.add_trace(go.Bar(x=x[mask], y=df.index[mask],orientation='h',
                         text=x[mask], texttemplate='%{text:.2f}%',textposition='
                         hovertemplate='Average Return in %{y} Stocks = %{x:.4f}%
                         marker=dict(color='red', opacity=0.7),name=col[-4:]),
                  row=1, col=i+1)
    fig.add_trace(go.Bar(x=x[~mask], y=df.index[~mask],orientation='h',
                         text=x[~mask], texttemplate='%{text:.2f}%', textposition
                         hovertemplate='Average Return in %{y} Stocks = %{x:.4f}%
                         marker=dict(color='green', opacity=0.7),name=col[-4:]),
                  row=1, col=i+1)
    fig.update_xaxes(range=(x.min()-.15,x.max()+.15), title='{} Returns'.format(c
                     showticklabels=False, row=1, col=i+1)
fig.update_layout(template=temp,title='Yearly Average Stock Returns by Sector',
                  hovermode='closest',margin=dict(l=250,r=50),
                  height=600, width=1000, showlegend=False)
fig.show()
```





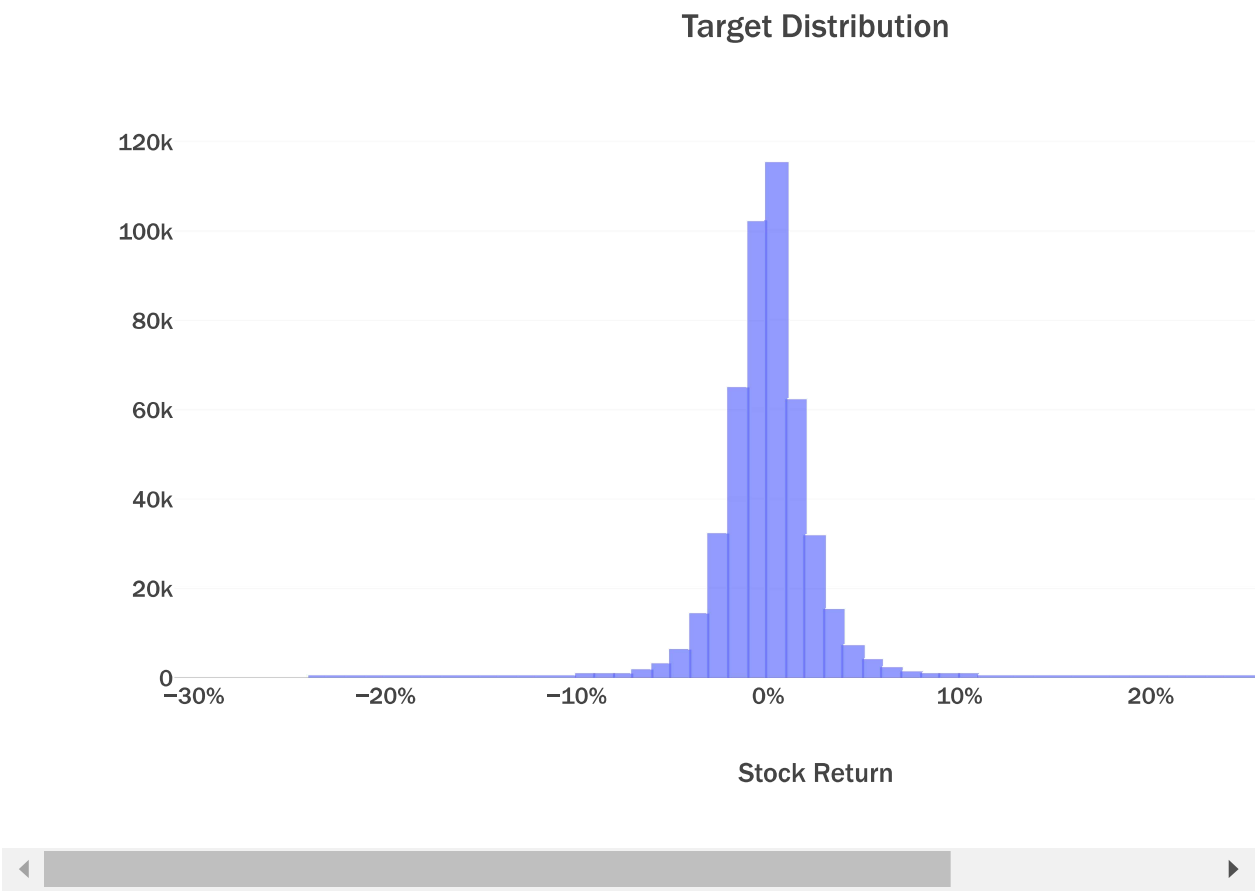
In 2021, nearly all industries saw a positive return on average, with the highest in Energy Resources at about 0.13% overall, while in 2018, all sectors saw a negative return except for Electric Power & Gas.

Since some of the stocks were added in December 2020, I will use the data filtered after this date so that the data will consist of 231 days of stock prices for all 2,000 stocks.

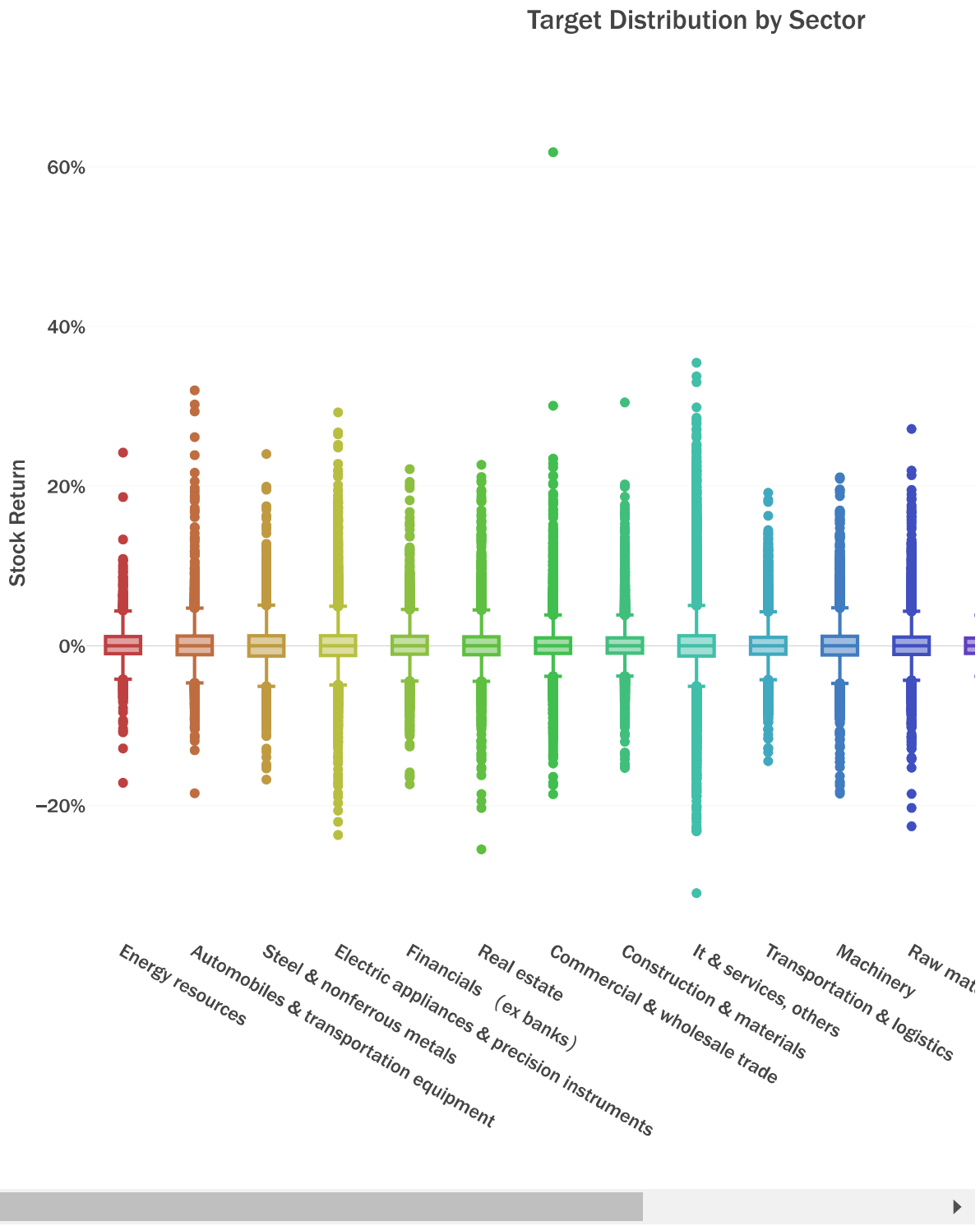
In [5]:
```
train_df=train_df[train_df.Date>'2020-12-23']
print("New Train Shape {}.\nMissing values in Target = {}".format(train_df.shape,
```

```
New Train Shape (462000, 15).
Missing values in Target = 0
```

In [6]:
```
fig = go.Figure()
x_hist=train_df['Target']
fig.add_trace(go.Histogram(x=x_hist*100,
                           marker=dict(color=colors[0], opacity=0.7,
                                       line=dict(width=1, color=colors[0])),
                           xbins=dict(start=-40,end=40,size=1)))
fig.update_layout(template=temp,title='Target Distribution',
                  xaxis=dict(title='Stock Return',ticksuffix='%'), height=450)
fig.show()
```



In [7]:
```
pal = ['hsl('+str(h)+',50%'+',50%)' for h in np.linspace(0, 360, 18)]
fig = go.Figure()
for i, sector in enumerate(df.index[::-1]):
    y_data=train_df[train_df['SectorName']==sector]['Target']
    fig.add_trace(go.Box(y=y_data*100, name=sector,
                         marker_color=pal[i], showlegend=False))
fig.update_layout(template=temp, title='Target Distribution by Sector',
                  yaxis=dict(title='Stock Return',ticksuffix='%'),
                  margin=dict(b=150), height=750, width=900)
fig.show()
```

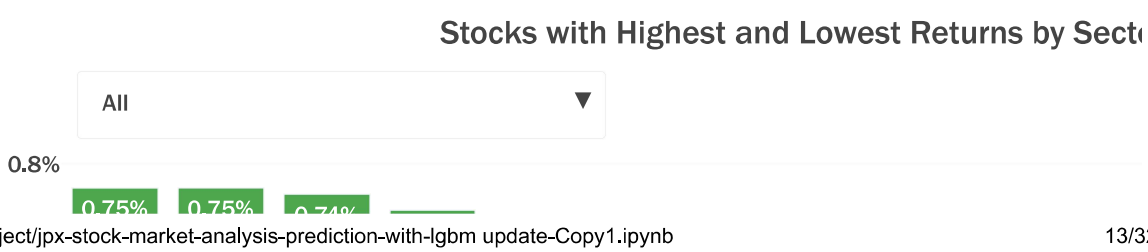


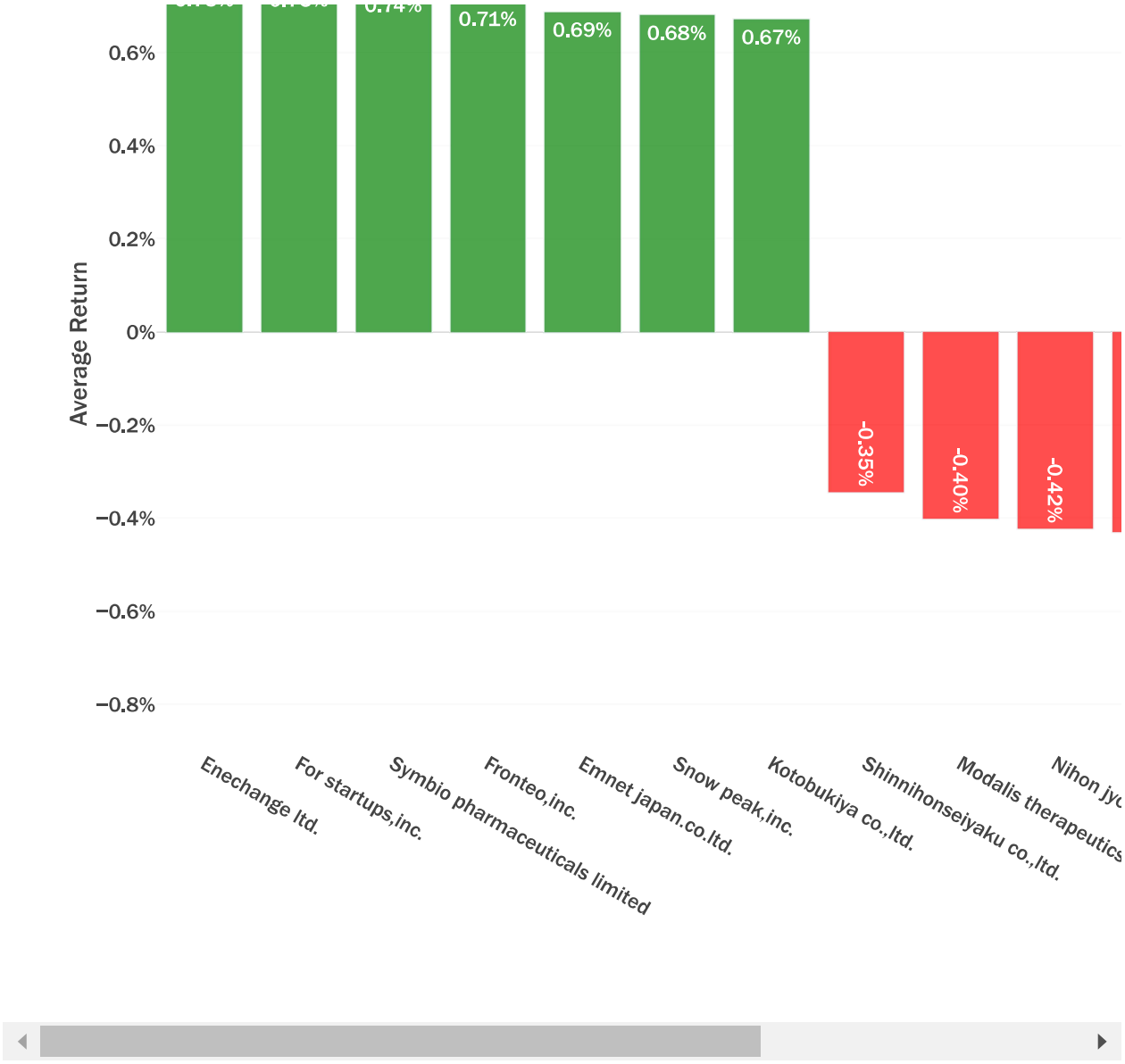
While most sectors have returns between 10% and -10%, there are quite a few outliers across all industries, with some returns as high as 62% in Commercial & Wholesale Trade and others as low as -31% in IT & Services sector. The graph below shows the stock price movements within each sector.

In [8]:
```
train_date=train_df.Date.unique()
sectors=train_df.SectorName.unique().tolist()
sectors.insert(0, 'All')
open_avg=train_df.groupby('Date')['Open'].mean()
high_avg=train_df.groupby('Date')['High'].mean()
low_avg=train_df.groupby('Date')['Low'].mean()
close_avg=train_df.groupby('Date')['Close'].mean()
buttons=[]

fig = go.Figure()
for i in range(18):
    if i != 0:
        open_avg=train_df[train_df.SectorName==sectors[i]].groupby('Date')['Open'
        high_avg=train_df[train_df.SectorName==sectors[i]].groupby('Date')['High'
        low_avg=train_df[train_df.SectorName==sectors[i]].groupby('Date')['Low'].
        close_avg=train_df[train_df.SectorName==sectors[i]].groupby('Date')['Clos

    fig.add_trace(go.Candlestick(x=train_date, open=open_avg, high=high_avg,
                                 low=low_avg, close=close_avg, name=sectors[i],
                                 visible=(True if i==0 else False)))

    visibility=[False]*len(sectors)
    visibility[i]=True
    button = dict(label = sectors[i],
                  method = "update",
                  args=[{"visible": visibility}])
    buttons.append(button)

fig.update_xaxes(rangeslider_visible=True,
                 rangeselector=dict(
                     buttons=list([
                         dict(count=3, label="3m", step="month", stepmode="backwa
                         dict(count=6, label="6m", step="month", stepmode="backwa
                         dict(step="all")]), xanchor='left',yanchor='bottom', y=1
fig.update_layout(template=temp,title='Stock Price Movements by Sector',
                  hovermode='x unified', showlegend=False, width=1000,
                  updatemenus=[dict(active=0, type="dropdown",
                                    buttons=buttons, xanchor='left',
                                    yanchor='bottom', y=1.01, x=.01)],
                  yaxis=dict(title='Stock Price'))
fig.show()
```

In the candlestick charts above, the boxes represent the daily spread between the open and close prices and the lines represent the spread between the low and high prices. The color of the boxes indicates whether the close price was greater or lower than the open price, with green indicating a higher closing price on that day and red indicating a lower closing price. In late August, the market saw a consecutive 14-day period where the close price was greater than the open price.

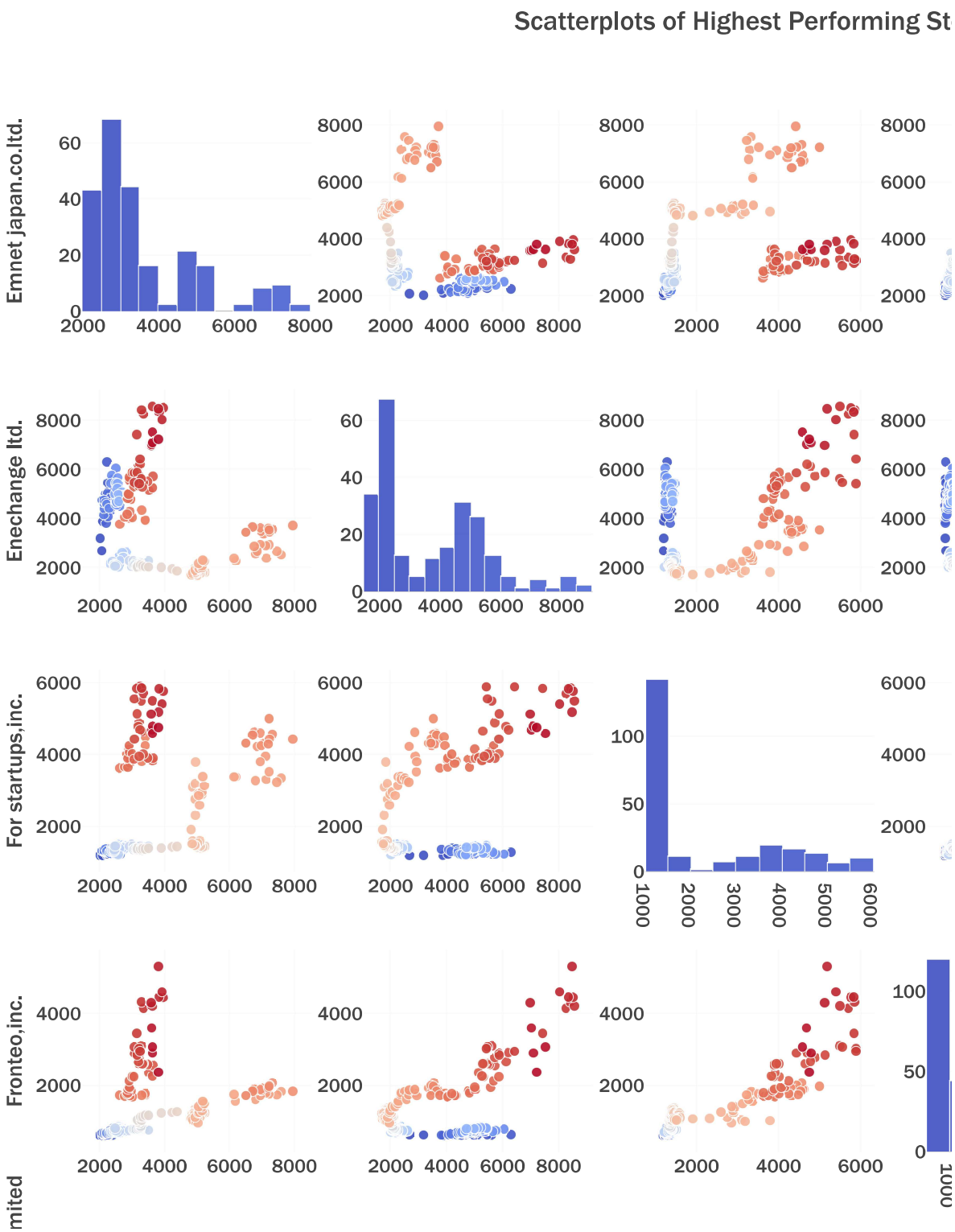
In [9]:
```
stock=train_df.groupby('Name')['Target'].mean().mul(100)
stock_low=stock.nsmallest(7)[::-1].rename("Return")
stock_high=stock.nlargest(7).rename("Return")
stock=pd.concat([stock_high, stock_low], axis=0).reset_index()
stock['Sector']='All'
for i in train_df.SectorName.unique():
    sector=train_df[train_df.SectorName==i].groupby('Name')['Target'].mean().mul(
    stock_low=sector.nsmallest(7)[::-1].rename("Return")
    stock_high=sector.nlargest(7).rename("Return")
    sector_stock=pd.concat([stock_high, stock_low], axis=0).reset_index()
    sector_stock['Sector']=i
    stock=stock.append(sector_stock,ignore_index=True)

fig=go.Figure()
buttons = []
for i, sector in enumerate(stock.Sector.unique()):

    x=stock[stock.Sector==sector]['Name']
    y=stock[stock.Sector==sector]['Return']
    mask=y>0
    fig.add_trace(go.Bar(x=x[mask], y=y[mask], text=y[mask],
                         texttemplate='%{text:.2f}%',
                         textposition='auto',
                         name=sector, visible=(False if i != 0 else True),
                         hovertemplate='%{x} average return: %{y:.3f}%',
                         marker=dict(color='green', opacity=0.7)))
    fig.add_trace(go.Bar(x=x[~mask], y=y[~mask], text=y[~mask],
                         texttemplate='%{text:.2f}%',
                         textposition='auto',
                         name=sector, visible=(False if i != 0 else True),
                         hovertemplate='%{x} average return: %{y:.3f}%',
                         marker=dict(color='red', opacity=0.7)))

    visibility=[False]*2*len(stock.Sector.unique())
    visibility[i*2],visibility[i*2+1]=True,True
    button = dict(label = sector,
                  method = "update",
                  args=[{"visible": visibility}])
    buttons.append(button)

fig.update_layout(title='Stocks with Highest and Lowest Returns by Sector',
                  template=temp, yaxis=dict(title='Average Return', ticksuffix='%
                  updatemenus=[dict(active=0, type="dropdown",
                                    buttons=buttons, xanchor='left',
                                    yanchor='bottom', y=1.01, x=.01)],
                  margin=dict(b=150),showlegend=False,height=700, width=900)
fig.show()
```

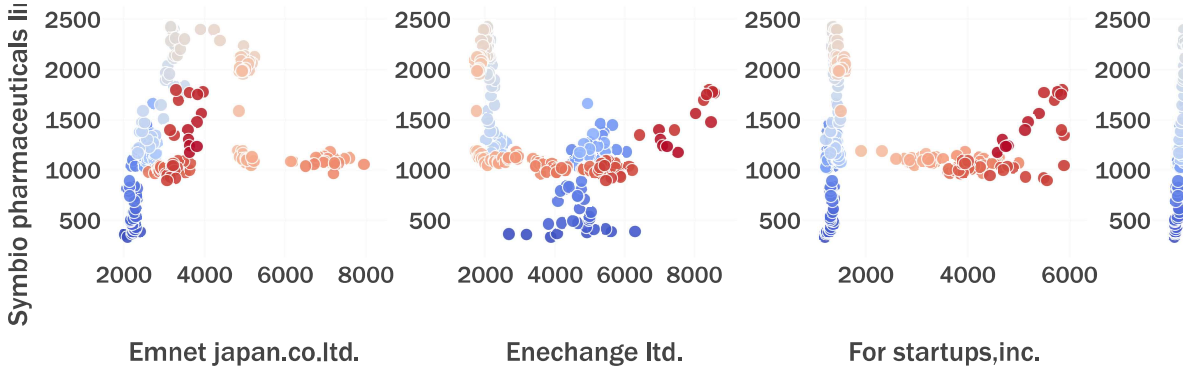




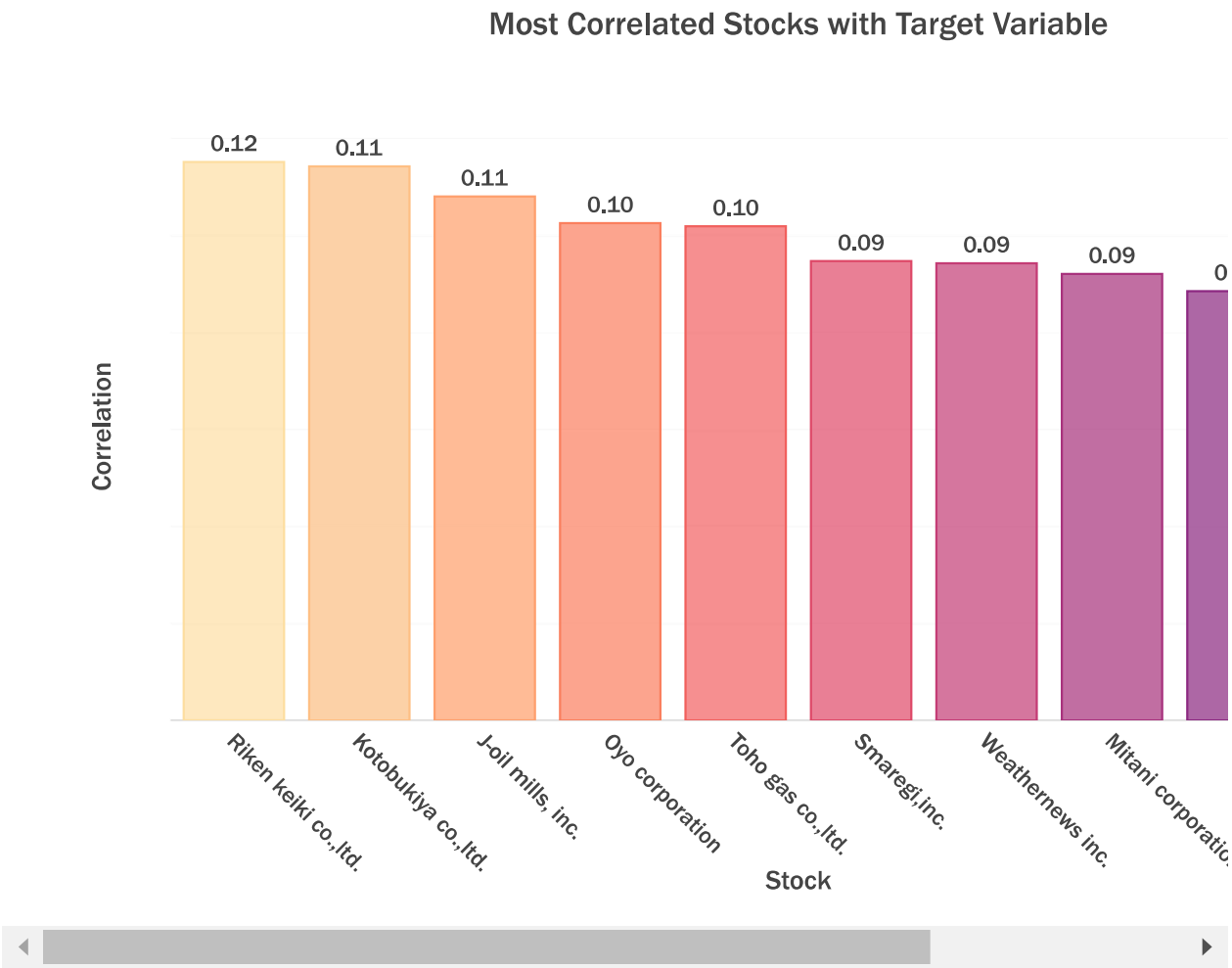
Among stocks with the highest return on average since December 2020, 6 of the 7 were in the IT & Services sector and one was in the Pharmaceutical sector. The IT & Services and Pharmaceutical sectors also make up 6 of the stocks with the lowest returns on average. The graph below shows the relationships between the top 5 stocks with the highest average returns, Enechange Ltd., For Startups, Inc., Symbio Pharmaceuticals, Fronteo, Inc., and Emnet Japan Co. Ltd.

In [10]:
```
stocks=train_df[train_df.SecuritiesCode.isin([4169,7089,4582,2158,7036])]
df_pivot=stocks.pivot_table(index='Date', columns='Name', values='Close').reset_i
pal=['rgb'+str(i) for i in sns.color_palette("coolwarm", len(df_pivot))]

fig = ff.create_scatterplotmatrix(df_pivot.iloc[:,1:], diag='histogram', name='')
fig.update_traces(marker=dict(color=pal, opacity=0.9, line_color='white', line_wi
fig.update_layout(template=temp, title='Scatterplots of Highest Performing Stocks
                  height=1000, width=1000, showlegend=False)
fig.show()
```

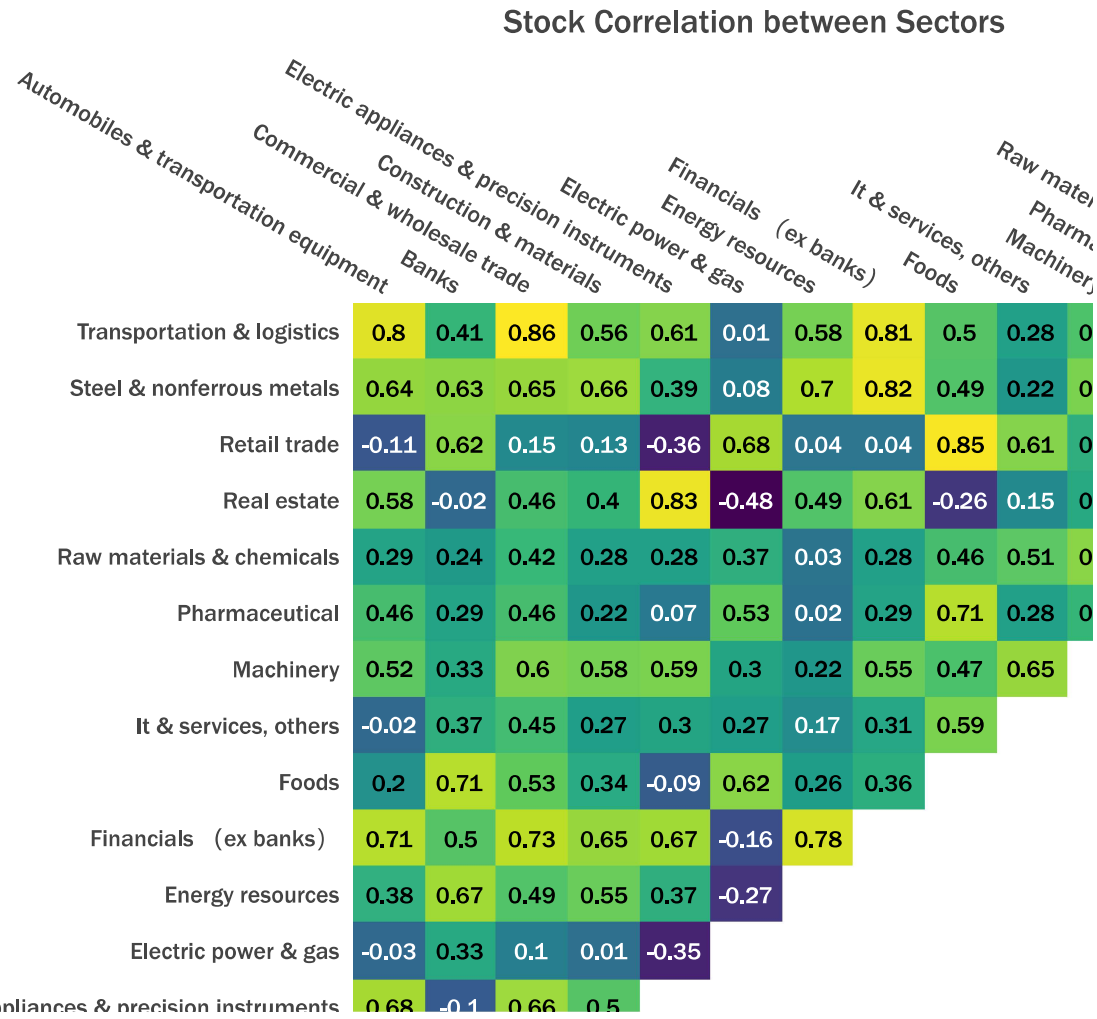


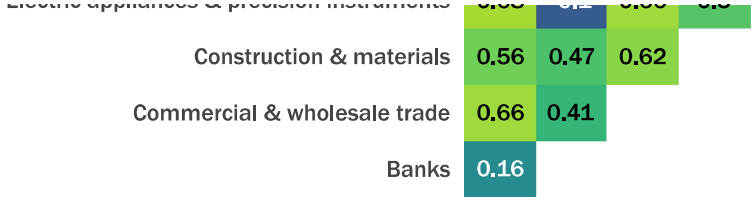

Symbio pharmaceuticals lii
2500
2000
1500
1000
500
2000
4000
6000
8000
Emnet japan.co.ltd.
Enechange ltd.
For startups,inc.

In [11]:
```
corr=train_df.groupby('SecuritiesCode')[['Target','Close']].corr().unstack().iloc
stocks=corr.nlargest(10).rename("Return").reset_index()
stocks=stocks.merge(train_df[['Name','SecuritiesCode']], on='SecuritiesCode').dro
pal=sns.color_palette("magma_r", 14).as_hex()
rgb=['rgba'+str(matplotlib.colors.to_rgba(i,0.7)) for i in pal]

fig = go.Figure()
fig.add_trace(go.Bar(x=stocks.Name, y=stocks.Return, text=stocks.Return,
                     texttemplate='%{text:.2f}', name='', width=0.8,
                     textposition='outside',marker=dict(color=rgb, line=dict(colo
                     hovertemplate='Correlation of %{x} with target = %{y:.3f}'))
fig.update_layout(template=temp, title='Most Correlated Stocks with Target Variab
                  yaxis=dict(title='Correlation',showticklabels=False),
                  xaxis=dict(title='Stock',tickangle=45), margin=dict(b=100),
                  width=800,height=500)
fig.show()
```



In [12]:
```
df_pivot=train_df.pivot_table(index='Date', columns='SectorName', values='Close')
corr=df_pivot.corr().round(2)
mask=np.triu(np.ones_like(corr, dtype=bool))
c_mask = np.where(~mask, corr, 100)
c=[]
for i in c_mask.tolist()[1:]:
    c.append([x for x in i if x != 100])

cor=c[::-1]
x=corr.index.tolist()[:-1]
y=corr.columns.tolist()[1:][::-1]
fig=ff.create_annotated_heatmap(z=cor, x=x, y=y,
                                hovertemplate='Correlation between %{x} and %{y}
                                colorscale='viridis', name='')
fig.update_layout(template=temp, title='Stock Correlation between Sectors',
                  margin=dict(l=250,t=270),height=800,width=900,
                  yaxis=dict(showgrid=False, autorange='reversed'),
                  xaxis=dict(showgrid=False))
fig.show()
```





3 | Feature Engineering

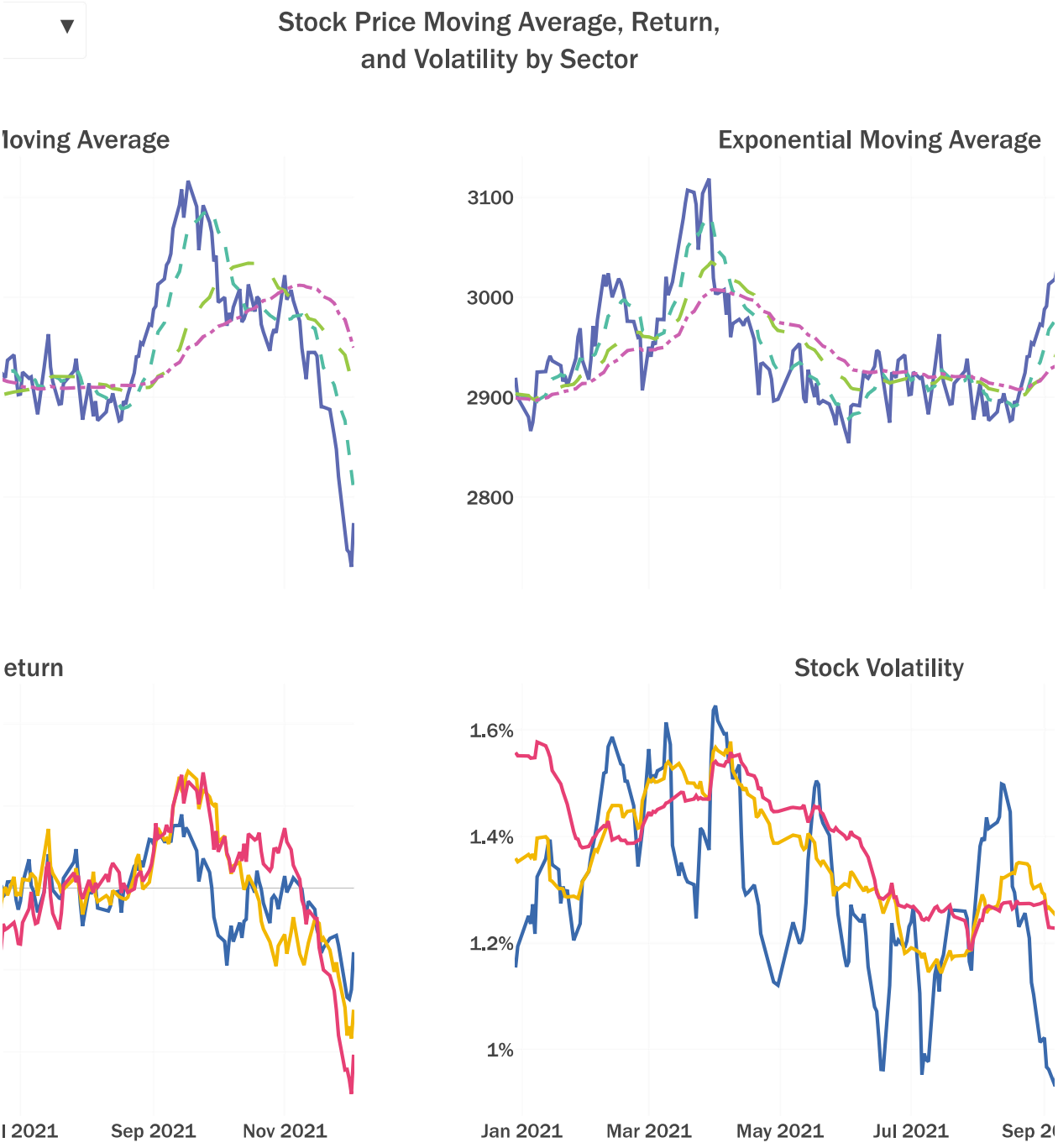
Before creating additional features, I will first generate the adjusted close price using the function provided by the competition hosts in the Train Demo (https://www.kaggle.com/code/smeitoma/train-demo#Generating-AdjustedClose-price) Notebook, which will adjust the close price to account for any stock splits or reverse splits. Using the stock's adjusted close price, I will create a set of new features, including the stock price moving average, exponential moving average, return, and volatility, each over a period of 5, 10, 20, 30, and 50 days. These features are shown in the graphs below across each sector.

In [13]:
```
def adjust_price(price):
    """
    Args:
        price (pd.DataFrame)  : pd.DataFrame include stock_price
    Returns:
        price DataFrame (pd.DataFrame): stock_price with generated AdjustedClose
    """
    # transform Date column into datetime
    price.loc[: ,"Date"] = pd.to_datetime(price.loc[: ,"Date"], format="%Y-%m-%d"

    def generate_adjusted_close(df):
        """
        Args:
            df (pd.DataFrame)  : stock_price for a single SecuritiesCode
        Returns:
            df (pd.DataFrame): stock_price with AdjustedClose for a single Securi
        """
        # sort data to generate CumulativeAdjustmentFactor
        df = df.sort_values("Date", ascending=False)
        # generate CumulativeAdjustmentFactor
        df.loc[:, "CumulativeAdjustmentFactor"] = df["AdjustmentFactor"].cumprod(
        # generate AdjustedClose
        df.loc[:, "AdjustedClose"] = (
            df["CumulativeAdjustmentFactor"] * df["Close"]
        ).map(lambda x: float(
            Decimal(str(x)).quantize(Decimal('0.1'), rounding=ROUND_HALF_UP)
        ))
        # reverse order
        df = df.sort_values("Date")
        # to fill AdjustedClose, replace 0 into np.nan
        df.loc[df["AdjustedClose"] == 0, "AdjustedClose"] = np.nan
        # forward fill AdjustedClose
        df.loc[:, "AdjustedClose"] = df.loc[:, "AdjustedClose"].ffill()
        return df

    # generate AdjustedClose
    price = price.sort_values(["SecuritiesCode", "Date"])
    price = price.groupby("SecuritiesCode").apply(generate_adjusted_close).reset_
    return price

train=train.drop('ExpectedDividend',axis=1).fillna(0)
prices=adjust_price(train)
```

In [14]:
```
def create_features(df):
    df=df.copy()
    col='AdjustedClose'
    periods=[5,10,20,30,50]
    for period in periods:
        df.loc[:,"Return_{}Day".format(period)] = df.groupby("SecuritiesCode")[co
        df.loc[:,"MovingAvg_{}Day".format(period)] = df.groupby("SecuritiesCode")
        df.loc[:,"ExpMovingAvg_{}Day".format(period)] = df.groupby("SecuritiesCod
        [col].ewm(span=period,adjust=False).mean().values
        df.loc[:,"Volatility_{}Day".format(period)] = np.log(df[col]).groupby(df[
    return df

price_features=create_features(df=prices)
price_features.drop(['RowId','SupervisionFlag','AdjustmentFactor','CumulativeAdju
```

In [15]:
```
price_names=price_features.merge(stock_list[['SecuritiesCode','Name','SectorName'
price_names=price_names[price_names.index>='2020-12-29']
price_names.fillna(0, inplace=True)

features=['MovingAvg','ExpMovingAvg','Return', 'Volatility']
names=['Average', 'Exp. Moving Average', 'Period', 'Volatility']
buttons=[]

fig = make_subplots(rows=2, cols=2,
                    shared_xaxes=True,
                    vertical_spacing=0.1,
                    subplot_titles=('Adjusted Close Moving Average',
                                    'Exponential Moving Average',
                                    'Stock Return', 'Stock Volatility'))

for i, sector in enumerate(price_names.SectorName.unique()):

    sector_df=price_names[price_names.SectorName==sector]
    periods=[0,10,30,50]
    colors=px.colors.qualitative.Vivid
    dash=['solid','dash', 'longdash', 'dashdot', 'longdashdot']
    row,col=1,1

    for j, (feature, name) in enumerate(zip(features, names)):
        if j>=2:
            row,periods=2,[10,30,50]
            colors=px.colors.qualitative.Bold[1:]
        if j%2==0:
            col=1
        else:
            col=2

        for k, period in enumerate(periods):
            if (k==0)&(j<2):
                plot_data=sector_df.groupby(sector_df.index)['AdjustedClose'].mea
            elif j>=2:
                plot_data=sector_df.groupby(sector_df.index)
                ['{}_{}Day'.format(feature,period)].mean().mul(100).rename('{}-da
            else:
                plot_data=sector_df.groupby(sector_df.index)
                ['{}_{}Day'.format(feature,period)].mean().rename('{}-day {}'.for
            fig.add_trace(go.Scatter(x=plot_data.index, y=plot_data, mode='lines'
                                     name=plot_data.name, marker_color=colors[k+1
                                     line=dict(width=2,dash=(dash[k] if j<2 else
                                     showlegend=(True if (j==0) or (j==2) else Fa
                                     visible=(False if i != 0 else True)), row=rc

    visibility=[False]*14*len(price_names.SectorName.unique())
    for l in range(i*14, i*14+14):
        visibility[l]=True
    button = dict(label = sector,
                  method = "update",
                  args=[{"visible": visibility}])
    buttons.append(button)

fig.update_layout(title='Stock Price Moving Average, Return,<br>and Volatility by
```

```
                  template=temp, yaxis3_ticksuffix='%', yaxis4_ticksuffix='%',
                  legend_title_text='Period', legend_tracegroupgap=250,
                  updatemenus=[dict(active=0, type="dropdown",
                                    buttons=buttons, xanchor='left',
                                    yanchor='bottom', y=1.105, x=.01)],
                  hovermode='x unified', height=800,width=1200, margin=dict(t=150
fig.show()
```



In the graphs of the stock price Moving Averages (MA) and Exponential Moving Averages (EMA), when the shorter period, the 10-Day average, crosses above the longer period, the 50-day average, the closing price tends to decrease, which is typically indicative of a sell signal. Conversely, when the 10-Day MA/EMA crosses the 50-Day MA/EMA from below, the closing price increases, which typically indicates a buy signal. In addition, the Exponential Moving Averages tend to respond to stock price changes more quickly as it puts greater weight on more recent observations. In the graphs of the Stock Return, there is greater fluctuation between longer periods, while Stock Volatility tends to be more stable over time, with more fluctuation in shorter periods.

4 | Stock Price Prediction

Submissions for this competition are evaluated based on the Sharpe Ratio (https://en.wikipedia.org/wiki/Sharpe_ratio) of the daily spread of returns. For each forecast day, all active stocks will be ranked in order of their predicted return. The returns for a single day treat the 200 highest (e.g. 0 to 199) ranked stocks as purchased and the lowest (e.g. 1800 to 1999) ranked 200 stocks as shorted. The stocks are then weighted based on their ranks and the total returns for the portfolio are calculated assuming the stocks were purchased the next day and sold the day after that.

Since risk control is also an important element of investment, the competing score is the mean/standard deviation of the time series of daily spread returns, rather than the simple mean or sum of daily spread returns. This makes it necessary to build a model that can respond to changes in the distribution of data and produce stable and high performance, rather than a model that only wins big on certain days. You can find a python implementation of this metric here (https://www.kaggle.com/code/smeitoma/jpx-competition-metric-definition).

In [16]:
```
def calc_spread_return_sharpe(df: pd.DataFrame, portfolio_size: int = 200, toprar
    """
    Args:
        df (pd.DataFrame): predicted results
        portfolio_size (int): # of equities to buy/sell
        toprank_weight_ratio (float): the relative weight of the most highly rank
    Returns:
        (float): sharpe ratio
    """
    def _calc_spread_return_per_day(df, portfolio_size, toprank_weight_ratio):
        """
        Args:
            df (pd.DataFrame): predicted results
            portfolio_size (int): # of equities to buy/sell
            toprank_weight_ratio (float): the relative weight of the most highly
        Returns:
            (float): spread return
        """
        assert df['Rank'].min() == 0
        assert df['Rank'].max() == len(df['Rank']) - 1
        weights = np.linspace(start=toprank_weight_ratio, stop=1, num=portfolio_s
        purchase = (df.sort_values(by='Rank')['Target'][:portfolio_size] * weight
        short = (df.sort_values(by='Rank', ascending=False)['Target'][:portfolio_
        return purchase - short

    buf = df.groupby('Date').apply(_calc_spread_return_per_day, portfolio_size, t
    sharpe_ratio = buf.mean() / buf.std()
    return sharpe_ratio
```

In [17]:
```
ts_fold = TimeSeriesSplit(n_splits=10, gap=10000)
prices=price_features.dropna().sort_values(['Date','SecuritiesCode',])
y=prices['Target'].to_numpy()
X=prices.drop(['Target'],axis=1)

feat_importance=pd.DataFrame()
sharpe_ratio=[]

for fold, (train_idx, val_idx) in enumerate(ts_fold.split(X, y)):

    print("\n========================== Fold {} ==========================".forma
    X_train, y_train = X.iloc[train_idx,:], y[train_idx]
    X_valid, y_val = X.iloc[val_idx,:], y[val_idx]

    print("Train Date range: {} to {}".format(X_train.Date.min(),X_train.Date.max
    print("Valid Date range: {} to {}".format(X_valid.Date.min(),X_valid.Date.max

    # Split into Train and Validation

    #Train Part
    X_train.drop(['Date','SecuritiesCode'], axis=1, inplace=True)\

    #Validation part

    X_val=X_valid[X_valid.columns[~X_valid.columns.isin(['Date','SecuritiesCode']
    val_dates=X_valid.Date.unique()[1:-1]

    # Check Shape of Train and Validation
    print("\nTrain Shape: {} {}, Valid Shape: {} {}".format(X_train.shape, y_trai

    params = {'n_estimators': 500,
              'num_leaves' : 100,
              'learning_rate': 0.1,
              'colsample_bytree': 0.9,
              'subsample': 0.8,
              'reg_alpha': 0.4,
              'metric': 'mae',
              'random_state': 21}

    #Select Gradient Boosting Method Algorithm

    gbm = LGBMRegressor(**params).fit(X_train, y_train,
                                      eval_set=[(X_train, y_train), (X_val, y_val
                                      verbose=300,
                                      eval_metric=['mae','mse'])

    # Make Prediction using Gradient Boosting Method Machine Learning Model
    y_pred = gbm.predict(X_val)

    # Check the Error Measurement Mean Square Error and Root Mean Square Error

    rmse = np.sqrt(mean_squared_error(y_val, y_pred))
    mae = mean_absolute_error(y_val, y_pred)

    # Select Most Important Features
    feat_importance["Importance_Fold"+str(fold)]=gbm.feature_importances_
```

```
    feat_importance.set_index(X_train.columns, inplace=True)

    #Give A Rank to Stock Script
    rank=[]
    X_val_df=X_valid[X_valid.Date.isin(val_dates)]
    for i in X_val_df.Date.unique():
        temp_df = X_val_df[X_val_df.Date == i].drop(['Date','SecuritiesCode'],axi
        temp_df["pred"] = gbm.predict(temp_df)
        temp_df["Rank"] = (temp_df["pred"].rank(method="first", ascending=False)-
        rank.append(temp_df["Rank"].values)

    stock_rank=pd.Series([x for y in rank for x in y], name="Rank")
    df=pd.concat([X_val_df.reset_index(drop=True),stock_rank,
                  prices[prices.Date.isin(val_dates)]['Target'].reset_index(drop=
    sharpe=calc_spread_return_sharpe(df)
    sharpe_ratio.append(sharpe)
    print("Valid Sharpe: {}, RMSE: {}, MAE: {}".format(sharpe,rmse,mae))

    del X_train, y_train,  X_val, y_val
    gc.collect()

print("\nAverage cross-validation Sharpe Ratio:{:.4f}, standard deviation = {:.2f
      .format(np.mean(sharpe_ratio),np.std(sharpe_ratio)))
```

```
========================== Fold 1 ==========================
Train Date range: 2017-03-16 00:00:00 to 2017-08-16 00:00:00
Valid Date range: 2017-08-23 00:00:00 to 2018-02-01 00:00:00

Train Shape: (192937, 25) (192937,), Valid Shape: (202933, 25) (202933,)
[300]   training's l2: 0.000284614      training's l1: 0.011257 valid_1's l2:
0.000397078     valid_1's l1: 0.012787
Valid Sharpe: 0.3009079954329179, RMSE: 0.02000817838989654, MAE: 0.01286053423
6739253

========================== Fold 2 ==========================
Train Date range: 2017-03-16 00:00:00 to 2018-01-25 00:00:00
Valid Date range: 2018-02-01 00:00:00 to 2018-07-09 00:00:00

Train Shape: (395870, 25) (395870,), Valid Shape: (202933, 25) (202933,)
[300]   training's l2: 0.000313241      training's l1: 0.011671 valid_1's l2:
0.000528003     valid_1's l1: 0.0153293
Valid Sharpe: 0.1231874743642734, RMSE: 0.023040598992935184, MAE: 0.0153818028
75060946

========================== Fold 3 ==========================
Train Date range: 2017-03-16 00:00:00 to 2018-07-02 00:00:00
Valid Date range: 2018-07-09 00:00:00 to 2018-12-11 00:00:00

Train Shape: (598803, 25) (598803,), Valid Shape: (202933, 25) (202933,)
[300]   training's l2: 0.000374375      training's l1: 0.0127406        valid_
1's l2: 0.000585164     valid_1's l1: 0.0163726
Valid Sharpe: 0.11665275234065432, RMSE: 0.02424497851104492, MAE: 0.0164191731
8877288

========================== Fold 4 ==========================
Train Date range: 2017-03-16 00:00:00 to 2018-12-04 00:00:00
```

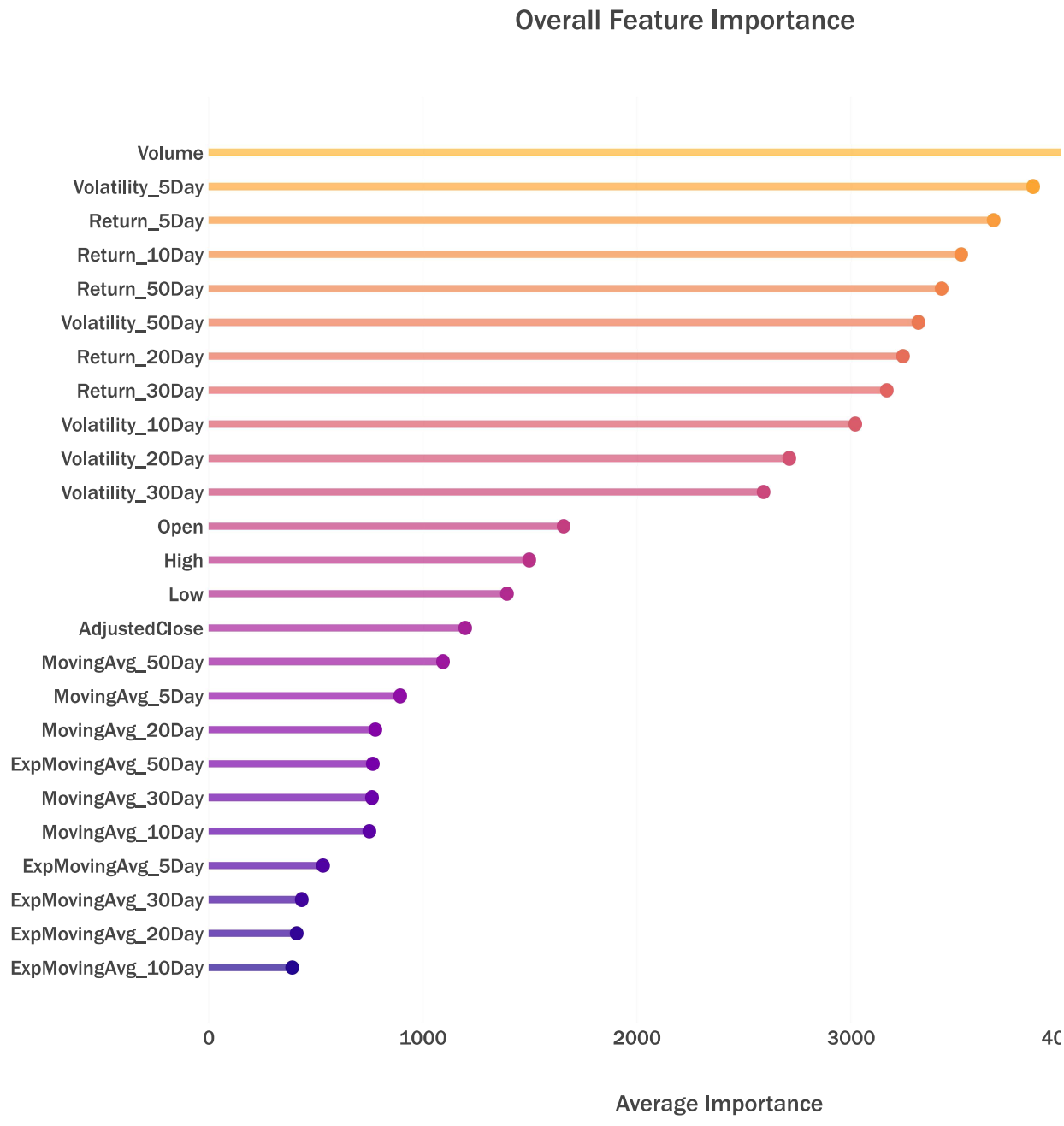
```
Valid Date range: 2018-12-11 00:00:00 to 2019-05-28 00:00:00

Train Shape: (801736, 25) (801736,), Valid Shape: (202933, 25) (202933,)
[300]   training's l2: 0.00041735       training's l1: 0.0135303        valid_
1's l2: 0.000639188     valid_1's l1: 0.0169536
Valid Sharpe: 0.09956267267315957, RMSE: 0.025328539748735123, MAE: 0.016993029
83794215

========================== Fold 5 ==========================
Train Date range: 2017-03-16 00:00:00 to 2019-05-21 00:00:00
Valid Date range: 2019-05-28 00:00:00 to 2019-10-29 00:00:00

Train Shape: (1004669, 25) (1004669,), Valid Shape: (202933, 25) (202933,)
[300]   training's l2: 0.000456202      training's l1: 0.0141866        valid_
1's l2: 0.00041566      valid_1's l1: 0.0140313
Valid Sharpe: 0.10185013885768678, RMSE: 0.020422467541968928, MAE: 0.014059015
581982444

========================== Fold 6 ==========================
Train Date range: 2017-03-16 00:00:00 to 2019-10-21 00:00:00
Valid Date range: 2019-10-29 00:00:00 to 2020-04-03 00:00:00

Train Shape: (1207602, 25) (1207602,), Valid Shape: (202933, 25) (202933,)
[300]   training's l2: 0.000447661      training's l1: 0.0141477        valid_
1's l2: 0.000952086     valid_1's l1: 0.0202354
Valid Sharpe: 0.04159893487714153, RMSE: 0.030919071062728776, MAE: 0.020272609
942744726

========================== Fold 7 ==========================
Train Date range: 2017-03-16 00:00:00 to 2020-03-27 00:00:00
Valid Date range: 2020-04-03 00:00:00 to 2020-09-04 00:00:00

Train Shape: (1410535, 25) (1410535,), Valid Shape: (202933, 25) (202933,)
[300]   training's l2: 0.000490864      training's l1: 0.0147274        valid_
1's l2: 0.000750961     valid_1's l1: 0.0190358
Valid Sharpe: -0.010676879843599713, RMSE: 0.027465699441421413, MAE: 0.0190805
45411257786

========================== Fold 8 ==========================
Train Date range: 2017-03-16 00:00:00 to 2020-08-28 00:00:00
Valid Date range: 2020-09-04 00:00:00 to 2021-02-04 00:00:00

Train Shape: (1613468, 25) (1613468,), Valid Shape: (202933, 25) (202933,)
[300]   training's l2: 0.000523939      training's l1: 0.015305 valid_1's l2:
0.000558608     valid_1's l1: 0.0161299
Valid Sharpe: 0.04957088148001701, RMSE: 0.02366925682150921, MAE: 0.0161539824
22306083

========================== Fold 9 ==========================
Train Date range: 2017-03-16 00:00:00 to 2021-01-28 00:00:00
Valid Date range: 2021-02-04 00:00:00 to 2021-07-06 00:00:00

Train Shape: (1816401, 25) (1816401,), Valid Shape: (202933, 25) (202933,)
[300]   training's l2: 0.000526224      training's l1: 0.0153749        valid_
1's l2: 0.000462635     valid_1's l1: 0.014938
Valid Sharpe: 0.3948583357035285, RMSE: 0.021525593871776102, MAE: 0.0149496192
68698432
```

```
========================== Fold 10 ==========================
Train Date range: 2017-03-16 00:00:00 to 2021-06-29 00:00:00
Valid Date range: 2021-07-06 00:00:00 to 2021-12-03 00:00:00

Train Shape: (2019334, 25) (2019334,), Valid Shape: (202933, 25) (202933,)
[300]   training's l2: 0.000520722      training's l1: 0.0153515        valid_
1's l2: 0.000494113     valid_1's l1: 0.0151446
Valid Sharpe: -0.0332920881464442, RMSE: 0.02225001300018992, MAE: 0.0151583815
46143187

Average cross-validation Sharpe Ratio: 0.1184, standard deviation = 0.13.
```

In [18]:
```
gc.collect()
```

Out[18]:
```
0
```

In [19]:
```
feat_importance['avg'] = feat_importance.mean(axis=1)
feat_importance = feat_importance.sort_values(by='avg',ascending=True)
pal=sns.color_palette("plasma_r", 29).as_hex()[2:]

fig=go.Figure()
for i in range(len(feat_importance.index)):
    fig.add_shape(dict(type="line", y0=i, y1=i, x0=0, x1=feat_importance['avg'][i
                       line_color=pal[::-1][i],opacity=0.7,line_width=4))
fig.add_trace(go.Scatter(x=feat_importance['avg'], y=feat_importance.index, mode=
                         marker_color=pal[::-1], marker_size=8,
                         hovertemplate='%{y} Importance = %{x:.0f}<extra></extra>
fig.update_layout(template=temp,title='Overall Feature Importance',
                  xaxis=dict(title='Average Importance',zeroline=False),
                  yaxis_showgrid=False, margin=dict(l=120,t=80),
                  height=700, width=800)
fig.show()
```



In [ ]:

In [20]:
```
cols_fin=feat_importance.avg.nlargest(3).index.tolist()
cols_fin.extend(('Open','High','Low'))
X_train=prices[cols_fin]
y_train=prices['Target']
gbm = LGBMRegressor(**params).fit(X_train, y_train)
```

In [21]:
```
import joblib
```

In [22]:
```
export_path = "C:\\Users\\PC\\Final Project"
filename="regressor_model_new.sav"
joblib.dump(gbm, export_path+ filename)
```

Out[22]:
```
['C:\\Users\\PC\\Final Projectregressor_model_new.sav']
```

In [ ]:

Thank you for reading!

.

Notebook References

- https://www.kaggle.com/code/smeitoma/train-demo (https://www.kaggle.com/code/smeitoma/train-demo)
- https://www.kaggle.com/code/smeitoma/submission-demo (https://www.kaggle.com/code/smeitoma/submission-demo)
- https://www.kaggle.com/code/smeitoma/jpx-competition-metric-definition (https://www.kaggle.com/code/smeitoma/jpx-competition-metric-definition)
- https://www.kaggle.com/code/chumajin/easy-to-understand-the-competition (https://www.kaggle.com/code/chumajin/easy-to-understand-the-competition)
- https://www.kaggle.com/code/riteshsinha/useful-features-in-predicting-stock-prices (https://www.kaggle.com/code/riteshsinha/useful-features-in-predicting-stock-prices)

In [ ]: