Predicting Future Stock Returns Using Financial Data

Peng Zhao | Christine Tan | Ashton Cho

December 12, 2024

Berkeley SCHOOL OF INFORMATION

Motivation

- Market capitalization in the US is close to $50T dollars (2024).
- Number of trades in US will approx. $85B (2025).
- Approx 4000 hedge funds just in the US alone
- Research Question: Can we use fundamental data (data published in companies' financial reports) and prior stock prices trends to predict future stock prices?
- Model: (1) Train model on prior stock return; (2) Train model on financial/fundamental company data
- Summary findings:
  - Prior stock returns - slight improvement over baseline model; structural shifts in economic/political landscape may limit the model's validity to predict future returns
  - Extend analyses to financial data for multiple quarters - no evidence that ML models outperform baseline

Data

- Use yfinance library
  - Uses Yahoo Finance API to retrieve market data
    - Has stock price data (including historical going back to prior years)
    - Has financial statement data (Income statement, Balance sheet, Cash Flow statement)
      - Limitation: Goes back to only 2023 Q3
- Focus on S&P 500 companies
- Downloaded all closing stock price
- Downloaded all financial data available (329 possible features)

Data Preparation - Stock Prices (Tesla)

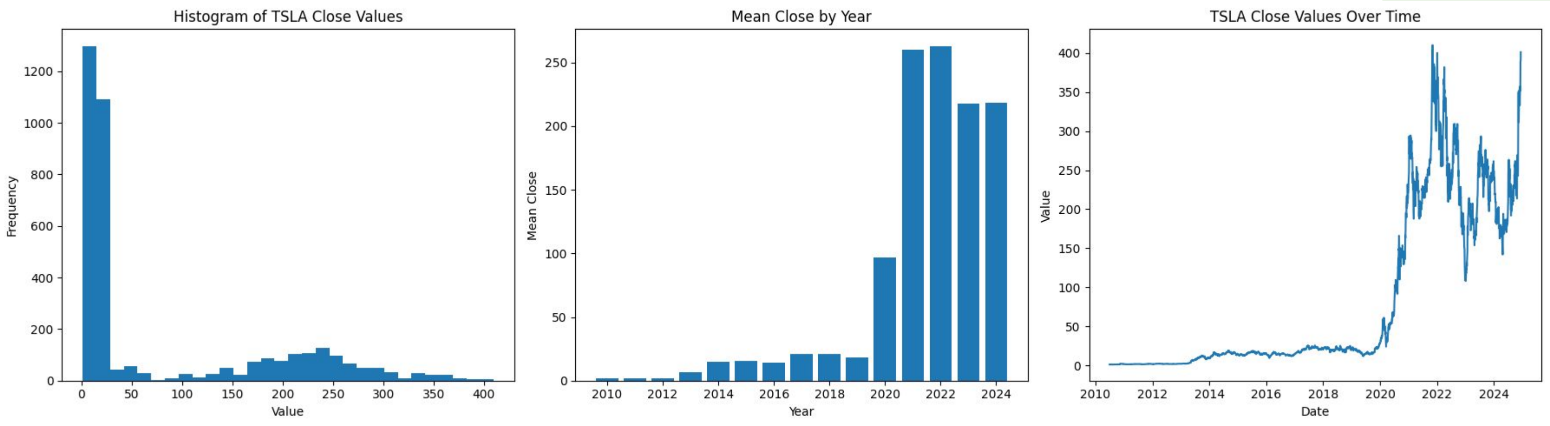
Price	Adj Close	Close	High	Low	Open	Volume
Ticker	TSLA	TSLA	TSLA	TSLA	TSLA	TSLA
Date						
2010-06-29 00:00:00+00:00	1.592667	1.592667	1.666667	1.169333	1.266667	281494500
2010-06-30 00:00:00+00:00	1.588667	1.588667	2.028000	1.553333	1.719333	257806500
2010-07-01 00:00:00+00:00	1.464000	1.464000	1.728000	1.351333	1.666667	123282000
2010-07-02 00:00:00+00:00	1.280000	1.280000	1.540000	1.247333	1.533333	77097000
2010-07-06 00:00:00+00:00	1.074000	1.074000	1.333333	1.055333	1.333333	103003500
...	...	...	...	...	...	...
2024-12-02 00:00:00+00:00	357.089996	357.089996	360.000000	351.149994	352.380005	77986500
2024-12-03 00:00:00+00:00	351.420013	351.420013	355.690002	348.200012	351.799988	58267200
2024-12-04 00:00:00+00:00	357.929993	357.929993	358.100006	348.600006	353.000000	50810900
2024-12-05 00:00:00+00:00	369.489990	369.489990	375.429993	359.500000	359.869995	81403600
2024-12-06 00:00:00+00:00	389.220001	389.220001	389.489990	370.799988	377.420013	80548300

3636 rows × 6 columns

	Date	Close
0	2010-06-29	1.592667
1	2010-06-30	1.588667
2	2010-07-01	1.464000
3	2010-07-02	1.280000
4	2010-07-06	1.074000
...	...	...
3631	2024-12-02	357.089996
3632	2024-12-03	351.420013
3633	2024-12-04	357.929993
3634	2024-12-05	369.489990
3635	2024-12-06	389.220001

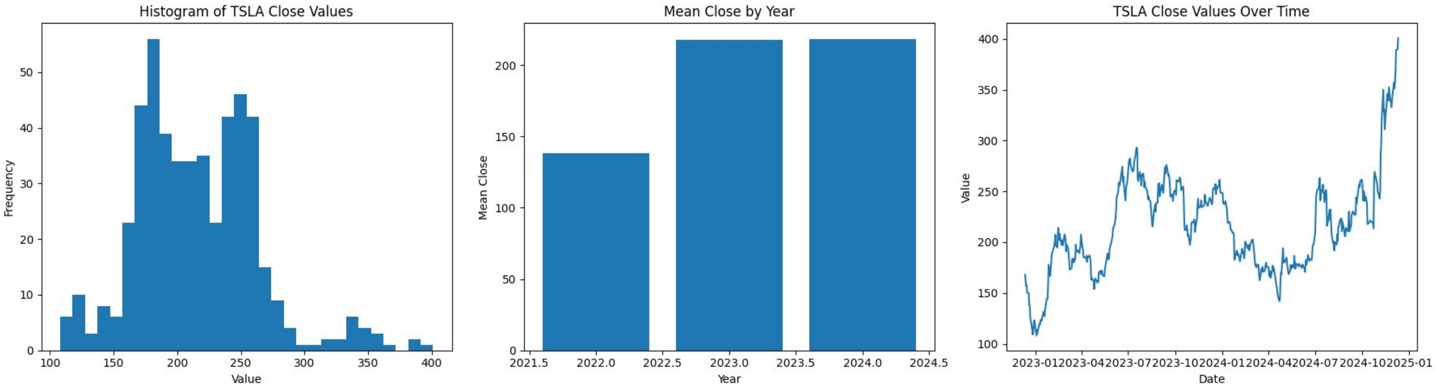
3636 rows × 2 columns

Data Exploration - Stock Prices (Tesla)



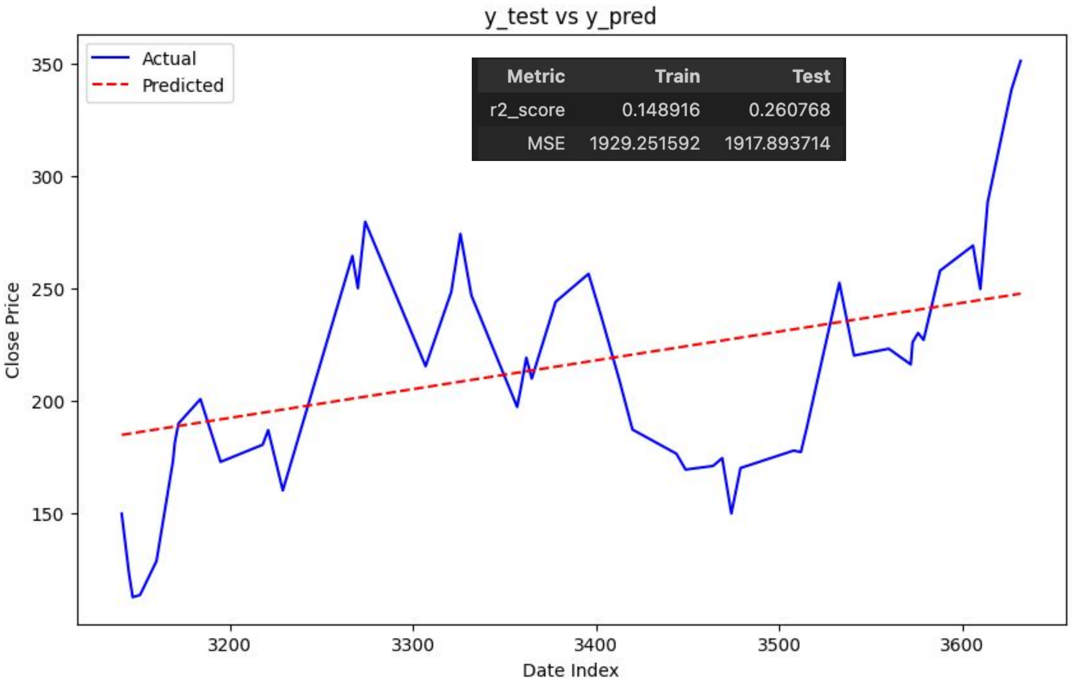
Data Exploration - Stock Prices (Tesla)

Past 2 Years



Modelling - Stock Prices (Tesla)

Using Linear Regression as a Baseline



Metric	Train	Test
r2_score	0.148916	0.260768
MSE	1929.251592	1917.893714

Berkeley SCHOOL OF INFORMATION

Data Preparation - Stock Prices (Tesla)

Restructuring the Data to Fit the LSTM Model

	Close
Date	
2010-06-29	1.592667
2010-06-30	1.588667
2010-07-01	1.464000
2010-07-02	1.280000
2010-07-06	1.074000
...	...
2024-12-02	357.089996
2024-12-03	351.420013
2024-12-04	357.929993
2024-12-05	369.489990
2024-12-06	389.220001

3636 rows × 1 columns

	Target Date	Target-3	Target-2	Target-1	Target
0	2022-12-07	194.860001	182.449997	179.820007	174.039993
1	2022-12-08	182.449997	179.820007	174.039993	173.440002
2	2022-12-09	179.820007	174.039993	173.440002	179.050003
3	2022-12-12	174.039993	173.440002	179.050003	167.820007
4	2022-12-13	173.440002	179.050003	167.820007	160.949997
...	...	...	...	...	...
498	2024-12-02	338.230011	332.890015	345.160004	357.089996
499	2024-12-03	332.890015	345.160004	357.089996	351.420013
500	2024-12-04	345.160004	357.089996	351.420013	357.929993
501	2024-12-05	357.089996	351.420013	357.929993	369.489990
502	2024-12-06	351.420013	357.929993	369.489990	389.220001

503 rows × 5 columns

Data Preparation - Stock Prices (Tesla)

Restructuring the Data to Fit the LSTM Model

	Close
Date	
2010-06-29	1.592667
2010-06-30	1.588667
2010-07-01	1.464000
2010-07-02	1.280000
2010-07-06	1.074000
...	...
2024-12-02	357.089996
2024-12-03	351.420013
2024-12-04	357.929993
2024-12-05	369.489990
2024-12-06	389.220001

3636 rows × 1 columns

Features

	Target Date	Target-3	Target-2	Target-1	Target
0	2022-12-07	194.860001	182.449997	179.820007	174.039993
1	2022-12-08	182.449997	179.820007	174.039993	173.440002
2	2022-12-09	179.820007	174.039993	173.440002	179.050003
3	2022-12-12	174.039993	173.440002	179.050003	167.820007
4	2022-12-13	173.440002	179.050003	167.820007	160.949997
...	...	...	...	...	...
498	2024-12-02	338.230011	332.890015	345.160004	357.089996
499	2024-12-03	332.890015	345.160004	357.089996	351.420013
500	2024-12-04	345.160004	357.089996	351.420013	357.929993
501	2024-12-05	357.089996	351.420013	357.929993	369.489990
502	2024-12-06	351.420013	357.929993	369.489990	389.220001

503 rows × 5 columns

2022/12/07 ~ 2024/12/06

Data Preparation - Stock Prices (Tesla)

Splitting the Data into Training, Validation, and Test Sets

Modelling - Stock Prices (Tesla)

Building the LSTM Model

```
# build model
model = Sequential([layers.Input((3, 1)),
                    layers.LSTM(32),
                    layers.Dense(1)])

# compile model. select loss, optimizer, and learning rate
model.compile(loss='mse',
              optimizer=Adam(learning_rate=0.01),
              metrics=['mean_absolute_error'])

# fit model and set number of epochs
history = model.fit(X_train, y_train, validation_data=(X_val, y_val), epochs=50)
```

Epoch 50/50 - loss: 10693.3867 - mean_absolute_error: 95.4124 - val_loss: 12784.7822 - val_mean_absolute_error: 111.7912

Experiments - Stock Prices (Tesla)

Tuning the Hyperparameters

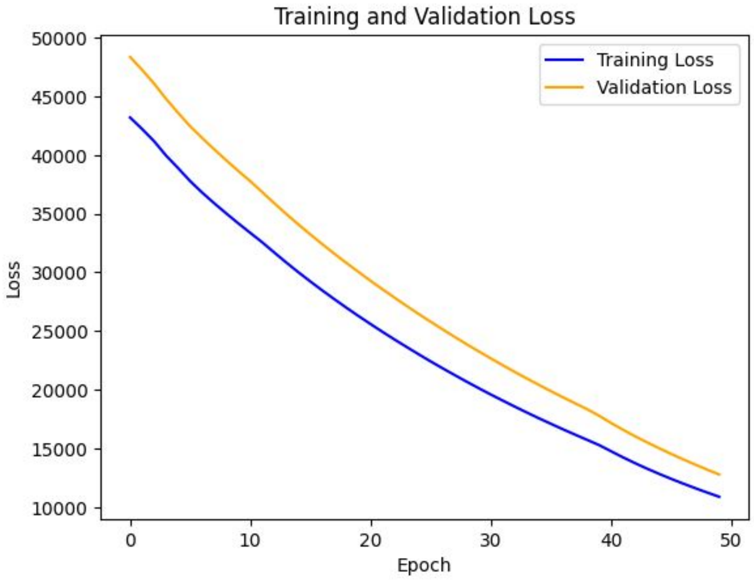
```
# tune model
model = Sequential([layers.Input((3, 1)),
                    layers.LSTM(32),
                    layers.Dense(32, activation='relu'),
                    layers.Dense(32, activation='relu'),
                    layers.Dense(1)])

# compile model. select loss, optimizer, and learning rate
model.compile(loss='mse',
              optimizer=Adam(learning_rate=0.001),
              metrics=['mean_absolute_error'])

# fit model and set number of epochs
history = model.fit(X_train, y_train, validation_data=(X_val, y_val), epochs=50)
```

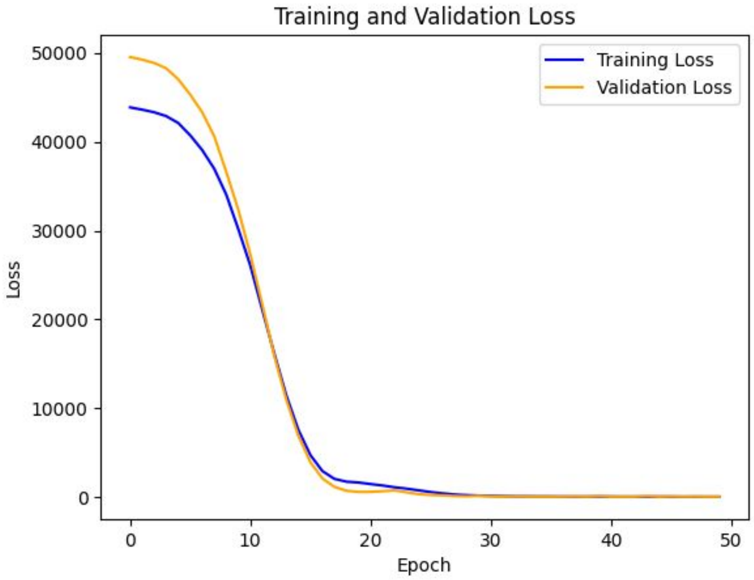
Epoch 50/50 - loss: 59.8151 - mean_absolute_error: 5.6036 - val_loss: 79.2024 - val_mean_absolute_error: 6.8545

Experiments - Stock Prices (Tesla)

Tuning the Hyperparameters

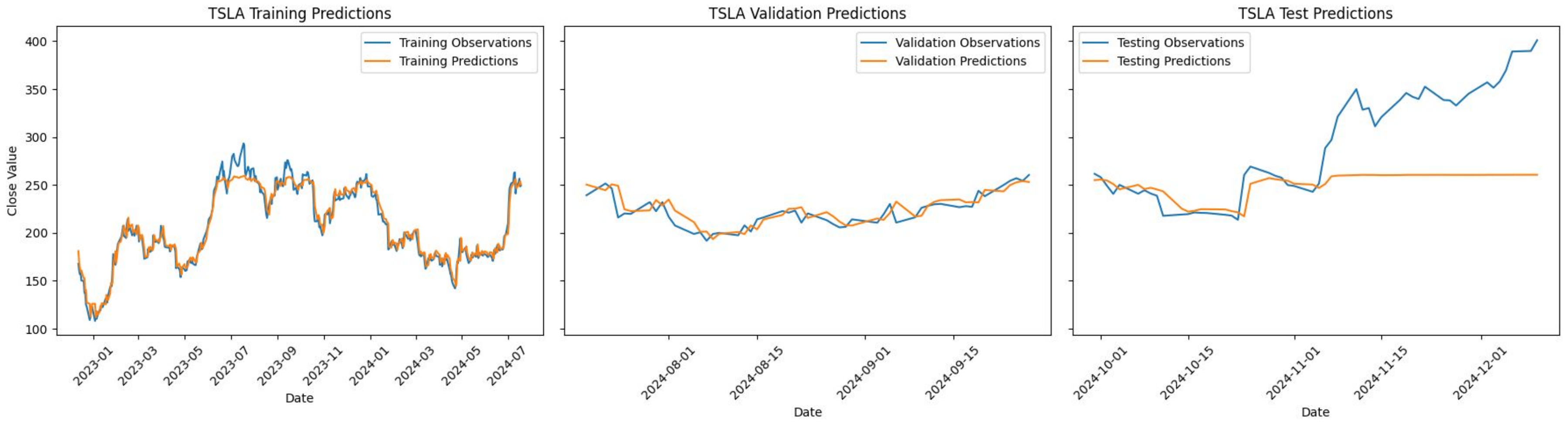
1 → 2





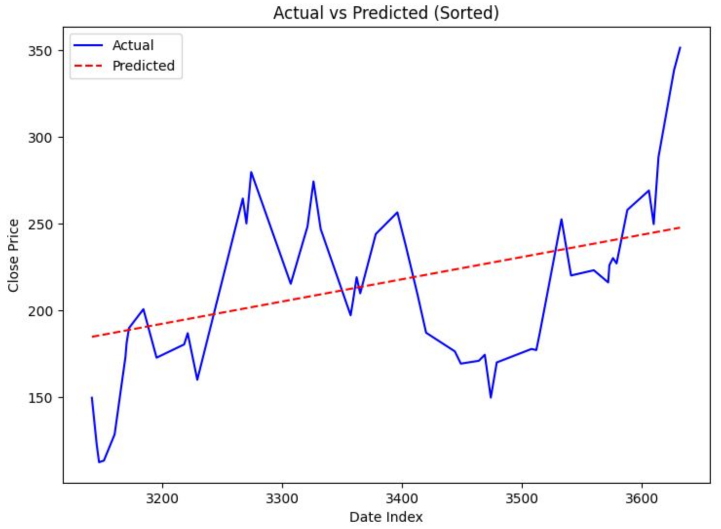
Modelling - Stock Prices (Tesla)

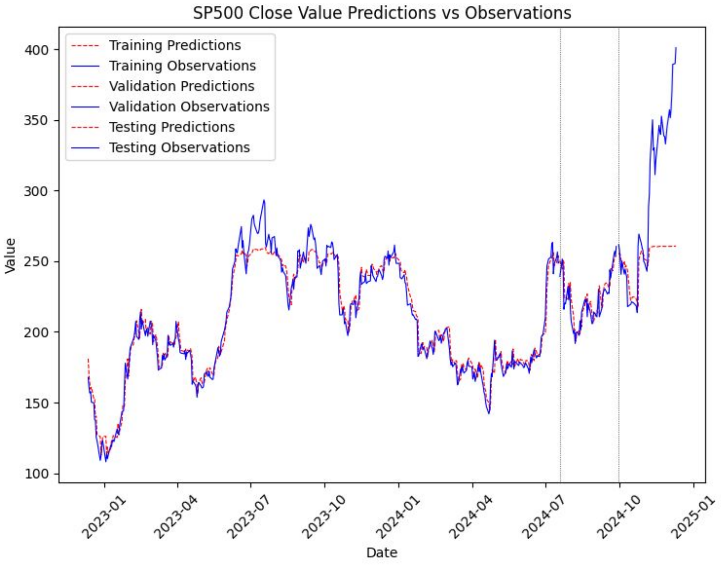
Plotting the Results



Modelling - Stock Prices (Tesla)

Improvements Over the Baseline - Past 2 Years

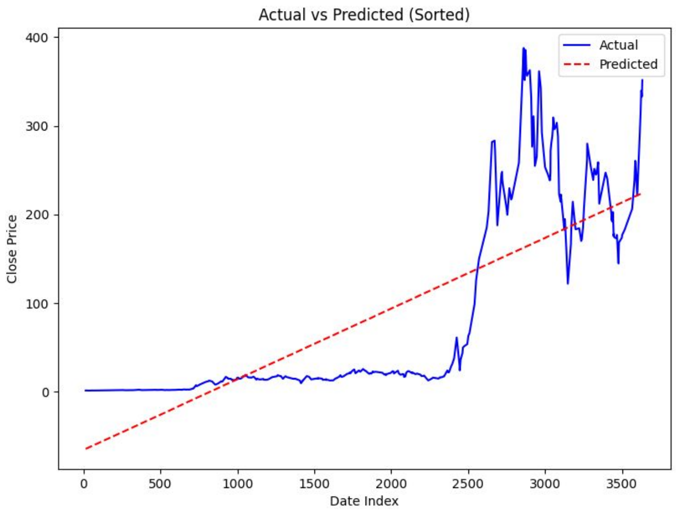


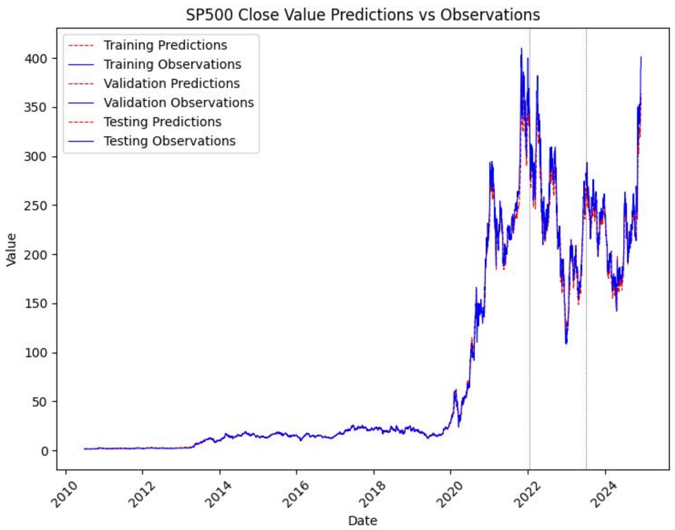


Model	Train	Test
Linear Regression	1929.251592	1917.893714
LSTM	67.511055	3633.377197

Experiments - Stock Prices (Tesla)

Improvements Over the Baseline - All Available Data



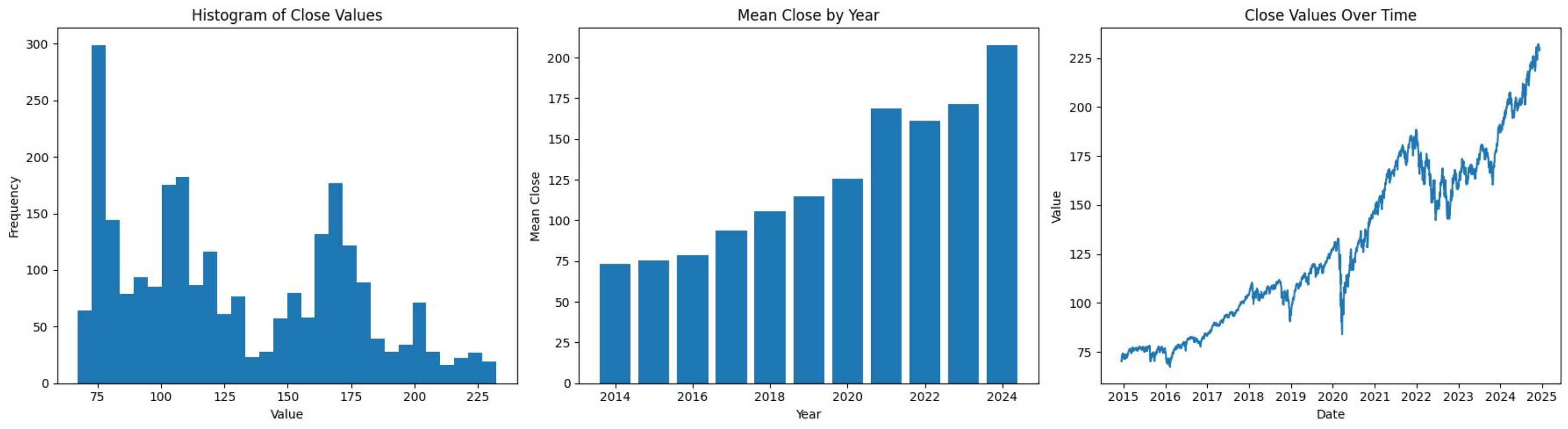


Model	Train	Test
Linear Regression	3984.223793	4559.376192
LSTM	18.692307	157.997055

Berkeley SCHOOL OF INFORMATION

Data Preparation - Stock Prices (SP500)

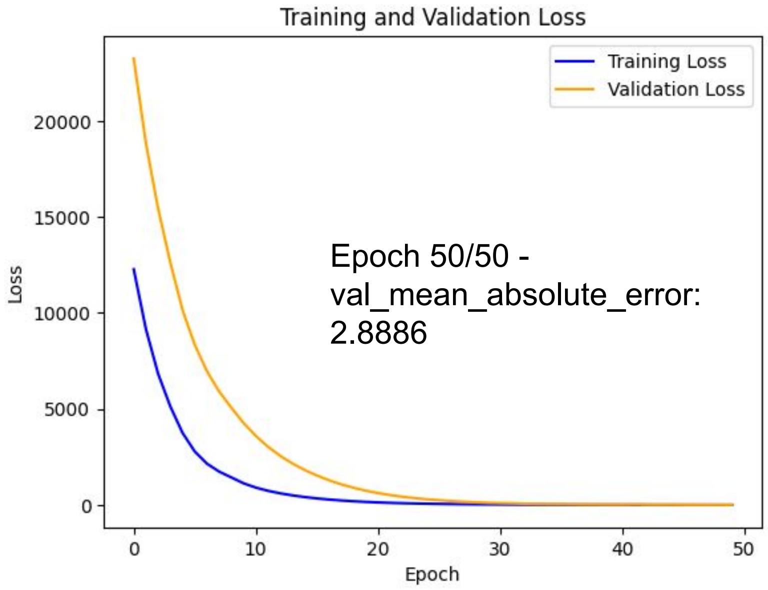
Past 10 Years



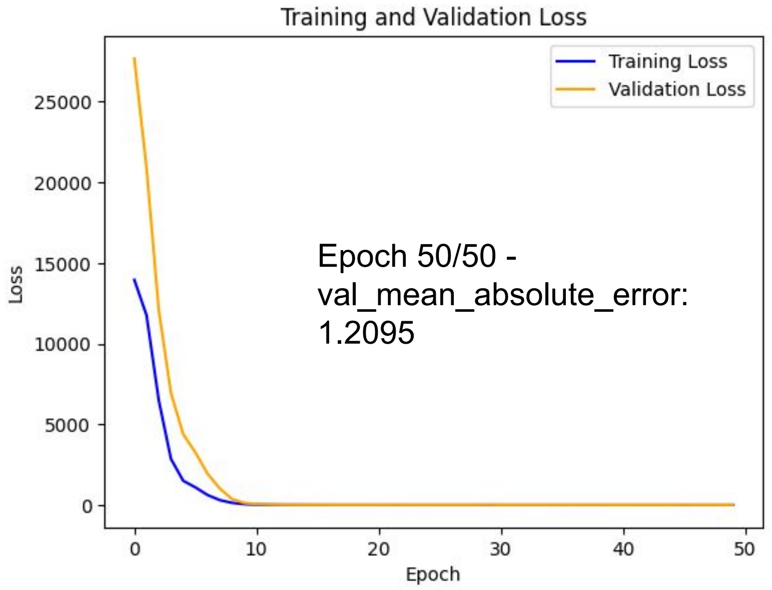
Experiments - Stock Prices (Tesla)

Tuning the Hyperparameters

1 → 2



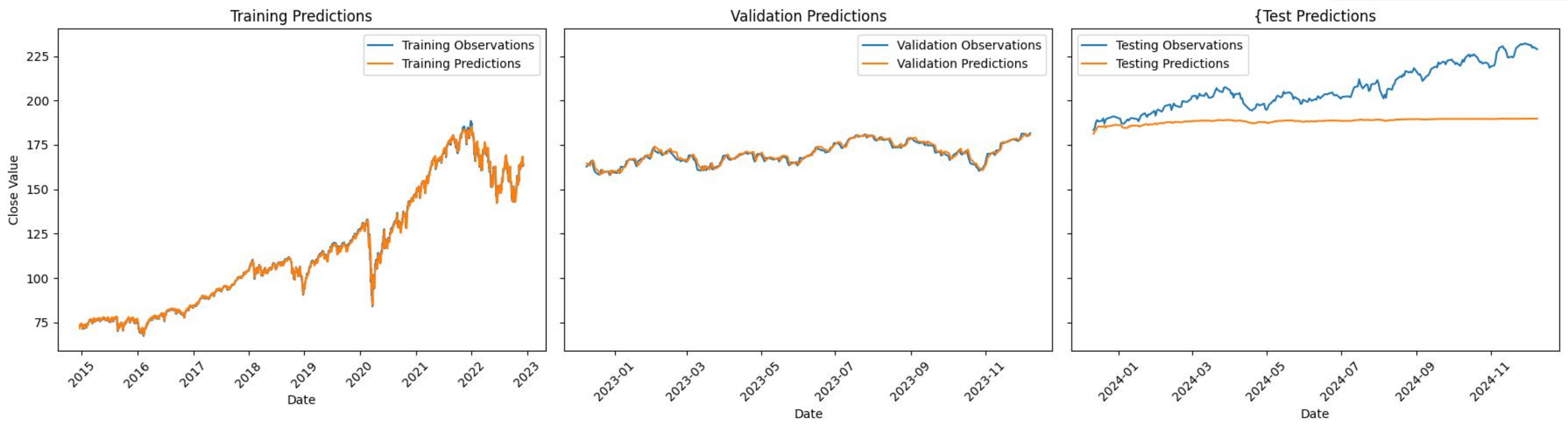
Epoch 50/50 -
val_mean_absolute_error:
2.8886



Epoch 50/50 -
val_mean_absolute_error:
1.2095

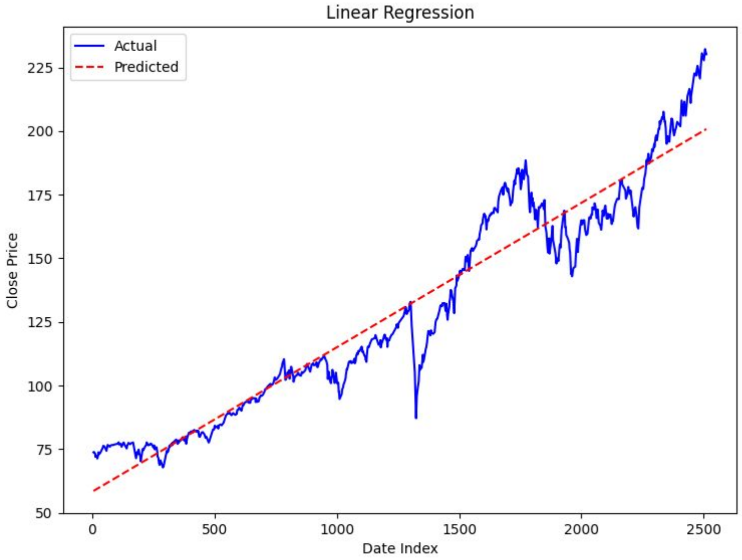
Modelling - Stock Prices (SP500)

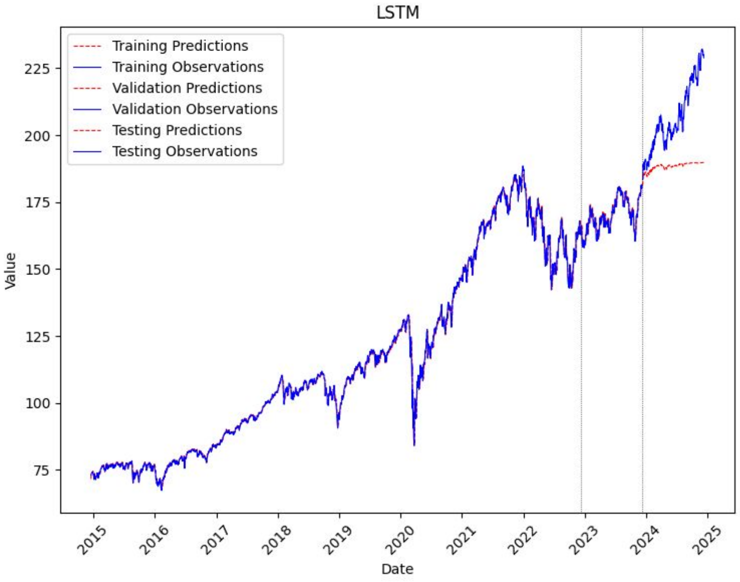
Plotting the Results



Modelling - Stock Prices (SP500)

Improvements Over the Baseline

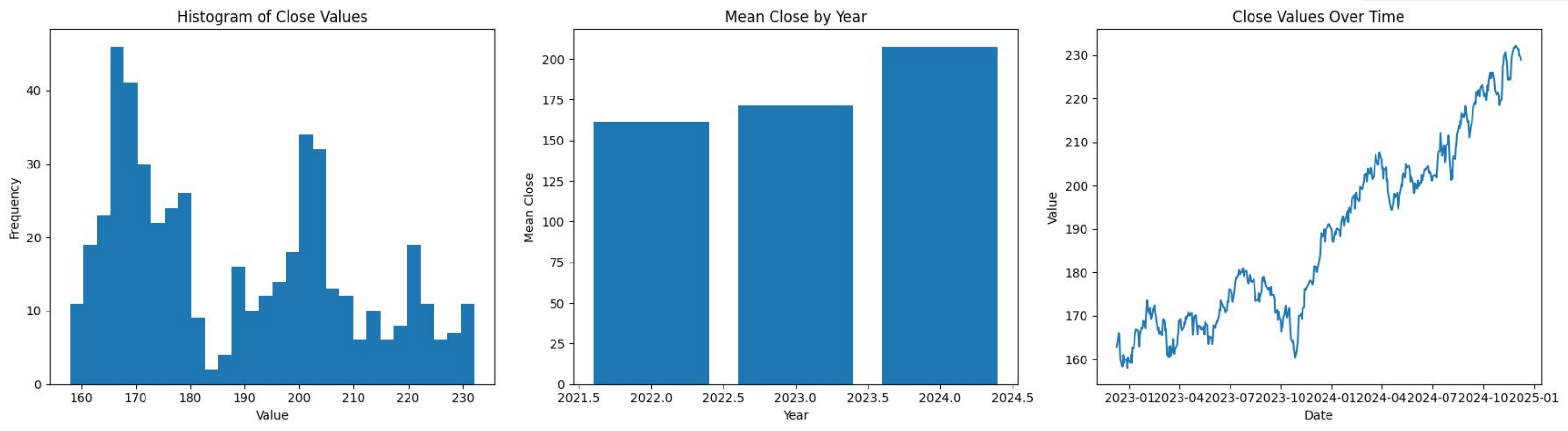




Model	Train	Test
Linear Regression	138.619551	137.897163
LSTM	2.321822	507.896790

Data Preparation - Stock Prices (SP500)

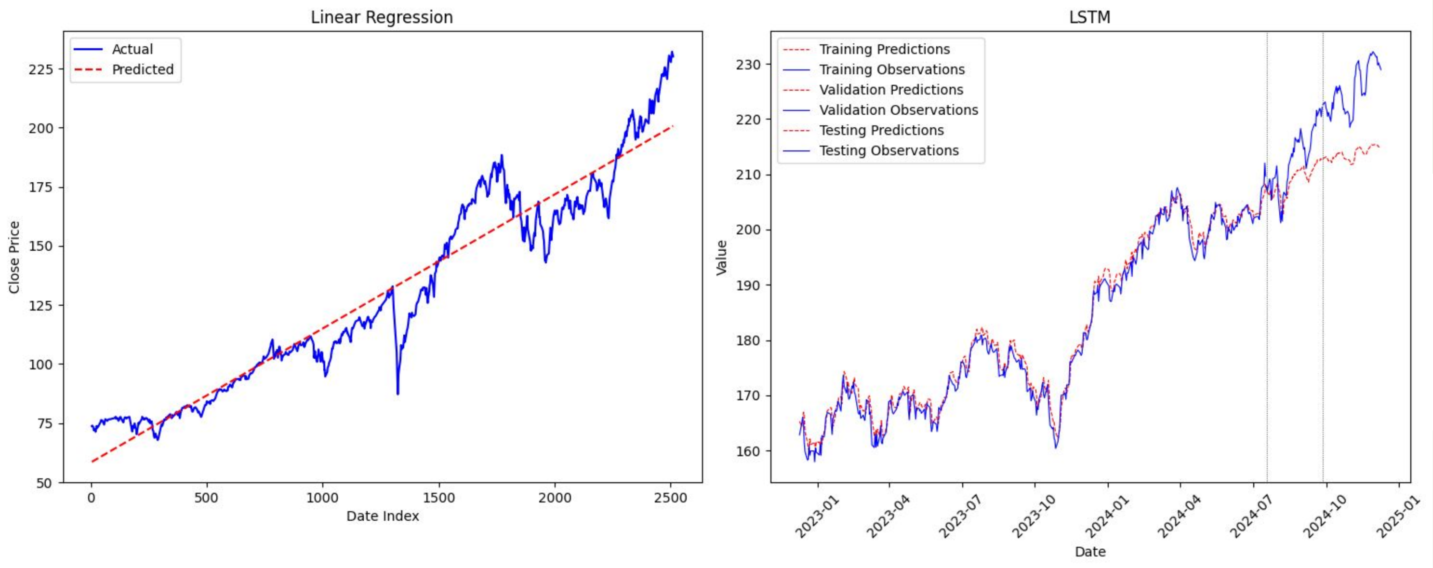
Past 2 Years



Berkeley SCHOOL OF INFORMATION

Modelling - Stock Prices (SP500)

Improvements Over the Baseline

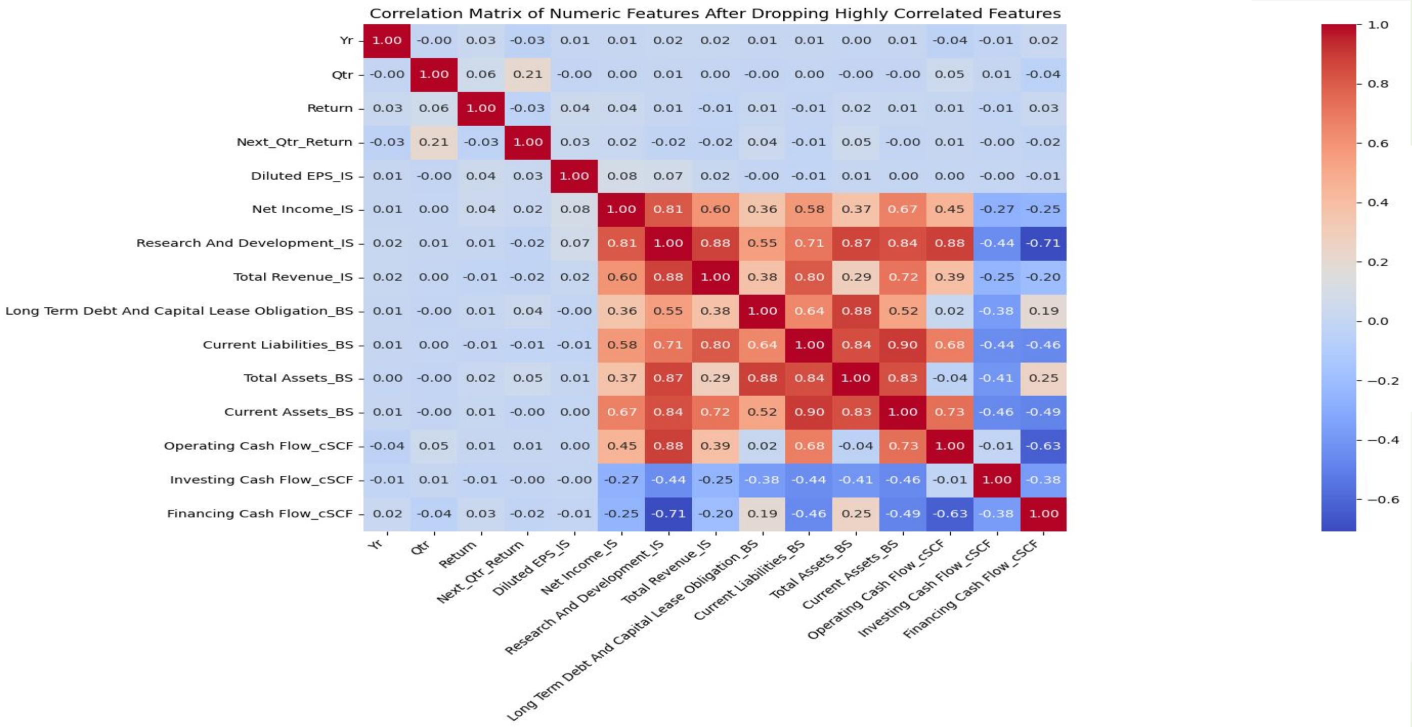


Model	Train	Test
Linear Regression	138.619551	137.897163
LSTM	3.286796	143.644852

Berkeley SCHOOL OF INFORMATION

Expand Analysis - Financial Data: Data Preparation

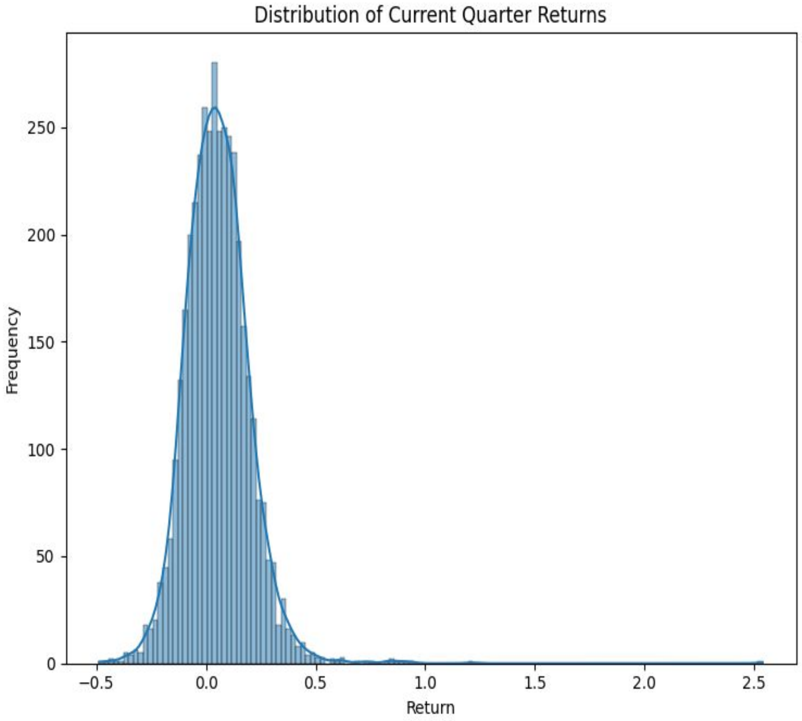
- Analyzed fundamental data and selected 16 financial features

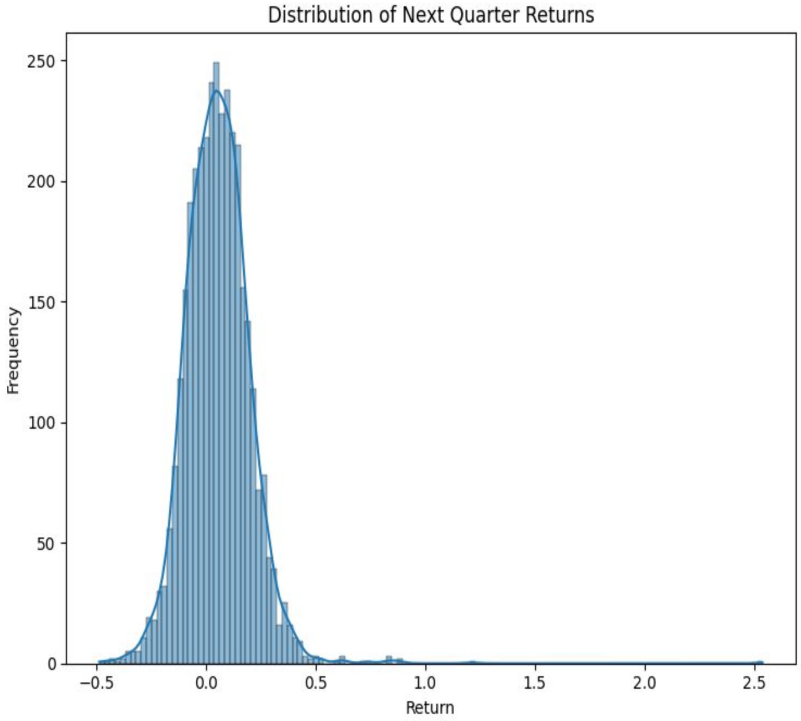


Expand Analysis - Financial Data: Data Preparation

- Aligning dates for stock prices into Year and Quarter (e.g., 2023Q1, 2023Q2) using Datetime
- Calculating target variable - next period return for each company
  - Using features from time t=0 to predict stock returns in time t+1
  - Quarterly return = (Quarter closing stock price - Quarter beginning stock price) / Quarter beginning stock price
  - 2024Q4 return = 11/29/24 closing price - Quarter beginning stock price) / Quarter beginning stock price

Expand Analysis - Financial Data: Data Preparation





Expand Analysis - Financial Data: Data Preparation

- Fundamental data already provided in Year and Quarter format in yfinance
    - Derived average previous 2 quarters for each feature for t=0 (i.e., we averaged t-1 and t-2)
- Train/validation/test split
    - Removed 2024Q3 from training and validation datasets because we are using 2024Q3 fundamental data to predict 2024Q4 returns
    - Split based on 60/20/20 by unique company
    - Test dataset only has 2024Q3
- Data structure:
    - Each row has ticker, year_qtr, next_quarter_return, current quarter fundamentals, average previous 2 quarters fundamental

Expand Analysis - Financial Data: Data Preparation

```
aapl
```

Target

	Date	Ticker	Yr	Qtr	year_qtr	Return	Next_Qtr_Return	EBITDA_IS	Basic EPS_IS	Diluted EPS_IS	...	Research And Development_IS	Gross Profit_IS	Tota Revenue_IS
8	2023-03-31 00:00:00+00:00	AAPL	2023	1	2023Q1	0.320475	0.168913	NaN	NaN	NaN	...	NaN	NaN	Na
9	2023-06-30 00:00:00+00:00	AAPL	2023	2	2023Q2	0.168913	-0.109211	NaN	NaN	NaN	...	NaN	NaN	Na
10	2023-09-30 00:00:00+00:00	AAPL	2023	3	2023Q3	-0.109211	0.109546	3.065300e+10	1.47	1.46	...	7.307000e+09	4.042700e+10	8.949800e+1
11	2023-12-31 00:00:00+00:00	AAPL	2023	4	2023Q4	0.109546	-0.075098	4.322100e+10	2.19	2.18	...	7.696000e+09	5.485500e+10	1.195750e+1
12	2024-03-31 00:00:00+00:00	AAPL	2024	1	2024Q1	-0.075098	0.240403	3.073600e+10	1.53	1.53	...	7.903000e+09	4.227100e+10	9.075300e+1
13	2024-06-30 00:00:00+00:00	AAPL	2024	2	2024Q2	0.240403	0.076215	2.820200e+10	1.40	1.40	...	8.006000e+09	3.967800e+10	8.577700e+1
14	2024-09-30 00:00:00+00:00	AAPL	2024	3	2024Q3	0.076215	0.050312	3.250200e+10	0.97	0.97	...	7.765000e+09	4.387900e+10	9.493000e+1
15	2024-12-31 00:00:00+00:00	AAPL	2024	4	2024Q4	0.050312	NaN	NaN	NaN	NaN	...	NaN	NaN	Na

Further Data Processing and Model Training Strategy

- Stock Identifier (Ticker) Removal
  - Eliminate company-specific bias
  - Focus on universal financial patterns
  - Enable model generalization across different stocks
  - Prevent model from learning stock-specific behaviors

- Date Information Processing
  - Retain quarterly patterns rather than specific dates
  - Avoid temporal overfitting
  - Focus on financial metric trends
  - Reduce time-specific dependencies

Baseline Model: Simple Majority Classifier

```
Training Set Analysis:
(1) The number of positive (class 1) samples in y_train: 1155
(2) The number of negative (class 0) samples in y_train: 652
The majority class in y_train is: Class 1 (Positive)
The baseline accuracy for y_train (majority class classifier) is: 0.6392
The baseline loss for y_train is: 12.4622

Validation Set Analysis:
(1) The number of positive (class 1) samples in y_val: 397
(2) The number of negative (class 0) samples in y_val: 217
The majority class in y_val is: Class 1 (Positive)
The baseline accuracy for y_val (majority class classifier) is: 0.6466
The baseline loss for y_val is: 12.2067
```

Further Data Processing and Model Training Strategy

1, Model Architecture

- three-layer neural network

```
# Define layer configurations
layer_config = [
    # (units, dropout_rate, l2_reg)
    (128, 0.3, None),          # First layer
    (64, 0.2, 0.01),          # Second layer
    (32, 0.1, 0.01)           # Third layer
]
```

2, Key Features:

A. Data cleaning handles outliers

```
def handle_extremes(df):
    """Handle extreme values using percentile clipping"""
    df = df.copy()
    # Select only numeric columns for processing
    numeric_cols = df.select_dtypes(include=['float64']).columns
    for col in numeric_cols:
        # Calculate the 1st and 99th percentiles for each numeric column
        q1 = df[col].quantile(0.01)
        q3 = df[col].quantile(0.99)
```

B. Built-in safeguards prevent overfitting

```
if l2_reg:
    model.add(keras.layers.Dense(units, kernel_regularizer=keras.regularizers.l2(l2_reg)))
else:
    model.add(keras.layers.Dense(units))

model.add(keras.layers.BatchNormalization()) # Normalize activations to stabilize training
```

C. Automatic learning speed adjustment

```
# Define a callback for learning rate reduction
reduce_lr = keras.callbacks.ReduceLROnPlateau(
    monitor='val_loss',  # Monitor the validation loss
    factor=0.2,          # Reduce the learning rate by a factor of 0.2
    patience=5,          # Wait for 5 epochs before reducing learning rate
    min_lr=0.0001        # Set the minimum learning rate to 0.0001
)
```

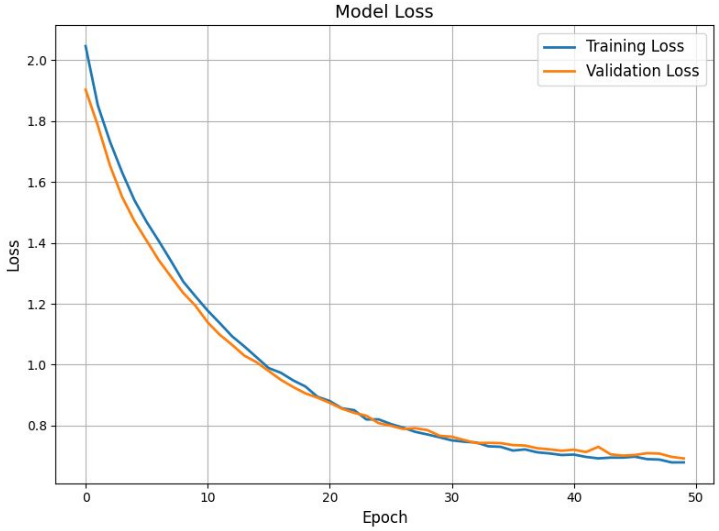
D. Training stops when no more improvement

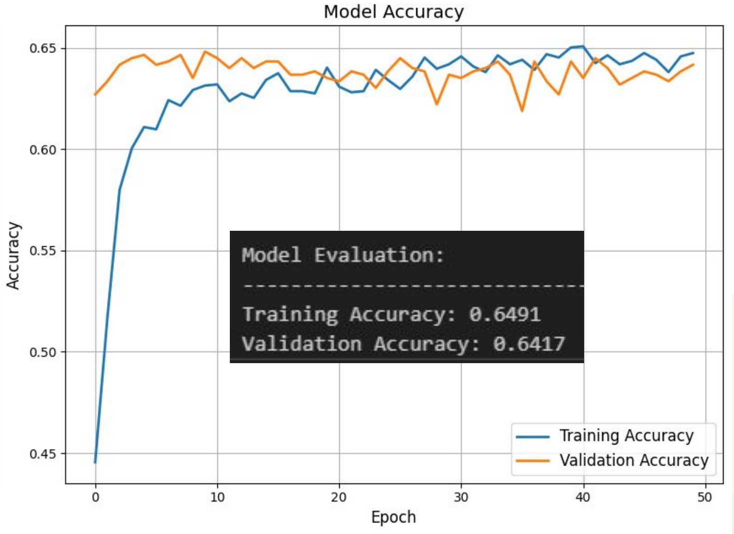
```
# Define a callback for early stopping
early_stopping = keras.callbacks.EarlyStopping(
    monitor='val_loss', # Monitor the validation loss
    patience=10,     # Wait for 10 epochs before stopping
    restore_best_weights=True  # Restore model weights from the epoch with the lowest validation loss
```

Model Training Plot

- **Model Stability Analysis:**

- Loss Curve Characteristics:
  - Initial rapid decrease in first 10 epochs
  - Smooth convergence after epoch 20
  - Final loss stabilizes around 0.7
  - Training and validation loss curves closely aligned
- Accuracy Curve Characteristics:
  - Quick improvement in first 5 epochs
  - Stable performance after epoch 10
  - Consistent accuracy around 64%
  - Minimal gap between training and validation accuracy





Model Training

- Test result

```
Complete Model Evaluation:
--------------------------------------------------
Training Accuracy: 0.6491
Validation Accuracy: 0.6417
Test Accuracy: 0.6206

Test Set Classification Report:
              precision    recall  f1-score   support

           0       0.41      0.10      0.16       178
           1       0.65      0.92      0.76       325

    accuracy                           0.63       503
   macro avg       0.53      0.51      0.46       503
weighted avg       0.57      0.63      0.55       503


Confusion Matrix:
               Predicted: 0   Predicted: 1
True: 0        18             160
True: 1        26             299
```

Class Prediction Capability:

- **Upward Trend (Class 1):** High recall and moderate precision, resulting in strong overall performance. A majority of actual upward trends were correctly identified, and most predicted upward trends were accurate.
- **Downward Trend (Class 0):** Low recall and moderate precision, leading to poor overall performance. Only a small fraction of actual downward trends were detected, and less than half of the predicted downward trends were correct.

Confusion Matrix Analysis:

- **True Negatives (TN):** Correctly identified a small portion of downward trends.
- **False Positives (FP):** Majority of actual downward movements were misclassified as upward trends.
- **False Negatives (FN):** Missed a small portion of upward trends.
- **True Positives (TP):** Correctly identified the majority of actual upward trends.

Experiments

- Hyperparameter Tuning

1. Batch Size (32->64) & Learning Rate Adjustment 0.0005->0.001)
2. Neuron Count Adjustment (128,64,32 -> 64,32,16)
3. Reducing Hidden Layers (3 -> 2)
4. Early Stopping Parameter Adjustment (10 -> 15)

1, Accuracy

	Train	Val	Test	
Base Model:	64.91%	64.17%	62.06%	
Batch&LR Adjust:	64.42%	64.33%	62.64%	👍
Neuron Reduced:	64.08%	64.33%	63.86%	👍
Hidden Layer Red:	63.97%	63.68%	60.95%	
Early Stop Adjust:	64.42%	64.17%	60.89%	

2, Downward Trend(Class2) Prediction

	Precision	Recall	F1	
Base Model:	0.41	0.10	0.16	
Batch&LR Adjust:	0.56	0.11	0.18	👍
Neuron Reduced:	0.23	0.02	0.03	
Hidden Layer Red:	0.40	0.14	0.21	👍
Early Stop Adjust:	0.42	0.15	0.22	👍

3, Upward Trend(Class1) Prediction

	Precision	Recall	F1	
Base Model:	0.65	0.92	0.76	
Batch&LR Adjust:	0.66	0.95	0.78	👍
Neuron Reduced:	0.64	0.97	0.77	
Hidden Layer Red:	0.65	0.88	0.75	
Early Stop Adjust:	0.66	0.89	0.75	

- **Batch Size & Learning Rate** shows best options across all tunings
  - High test accuracy
  - Highest precision for Class 0
  - Most balanced overall performance for Class 1
  - Smallest difference between training and validation metrics
  - Most stable loss and accuracy curves

Experiments

- Hyperparameter Tuning Vs Baseline

- **Overall Accuracy:**
  - All models perform similarly to the baseline on training and validation sets
  - Model improvements mostly focused on predictive capability rather than raw accuracy
  - Test set performance consistently lower than training/validation
- **Model Stability:**
  - Batch&LR adjustment showed most stable loss curves
  - Reduced neuron count led to smoother training process
  - Two-layer model showed good training stability but lower test accuracy
- **Parameter Impact:**
  - Batch size & Learning rate adjustments: Best balance of stability and performance
  - Neuron count reduction: Improved stability but reduced downtrend detection
  - Layer reduction: Simplified model but decreased overall performance
  - Early stopping adjustment: Limited impact on model performance

	Train	Val	Test	vs Baseline(Train)
Baseline:	63.92%	64.66%	-	-
Base Model:	64.91%	64.17%	62.06%	+0.99%
Batch&LR Adjust:	64.42%	64.33%	62.64%	+0.50%
Neuron Reduced:	64.08%	64.33%	63.86%	+0.16%
Hidden Layer Red:	63.97%	63.68%	60.95%	+0.05%
Early Stop Adjust:	64.42%	64.17%	60.89%	+0.50%

- **Model Selection:**
  - Batch Size & Learning Rate adjusted model shows best overall performance
  - Maintains baseline accuracy while adding predictive capabilities
  - Demonstrates most stable training process
- **Trade-offs:**
  - Accuracy vs. Predictive Power: Models barely surpass baseline accuracy
  - Complexity vs. Performance: Simpler models showed lower but more stable performance
  - Up/Down Detection: All models struggle with downward trend detection

Ethical Considerations

- Yahoo Finance API is for personal/educational purposes so cannot be used for commercial purposes (i.e., trading/investing for clients, etc)
  - Should not rely on data to advise clients
  - Data may not be accurate and up-to-date
- Yfinance scrapes data from Yahoo Finance which may potentially breach Yahoo Finance terms of service (note that the official Yahoo Finance API was discontinued in 2017).

Conclusions

- Both basic stock returns and expanded fundamentals models show little improvements over the baseline
- Limitations:
  - Models are too basic
  - We did not use all the possible fundamental features available on Yahoo Finance
  - Data limitations - other types of information may be impounded into stock prices (e.g., market sentiment, political factors)
- Future work:
  - Try different/alternative/advanced model architectures
  - Use all 330 possible fundamental features
  - Use additional features (e.g., ESG scores) available on Yahoo Finance
  - Use another fundamental data source (SEC's EDGAR) that has a longer time-series data
- Summary - not time to retire, YET!

References

https://www.statista.com/outlook/fmo/stocks/united-states

https://www.ibisworld.com/united-states/market-research-reports/hedge-funds-industry/

https://pypi.org/project/yfinance/

https://www.youtube.com/watch?v=OXwZtlcTiuk&t=69s

https://www.youtube.com/watch?v=CbTU92pbDKw

Contributions

- Christine
  - Data sourcing for S&P500 from yfinance
  - Data cleansing, structuring and prepping for stock returns and fundamentals data
  - Feature engineering
- Ashton
  - Data sourcing, cleaning, exploring, structuring, experimenting for:
  - Linear Regression Model
  - LSTM Model
- Peng
  - Data sourcing, cleaning, exploring, structuring, feature engineering, experimenting for:
  - Neural Network model Training