Australian Stock Price Prediction

Phase 2: Predictive Modelling of data

Group Name : Project Group 51

Name	Student ID
Hishikesh Phukan	s4031214
Ashif Parwez	s4021513

Table of Contents

Introduction

Phase 1 Summary

In Phase 1 of our project, we focused on the development of an appropriate and clean dataset primarily utilising the historical stock price data of the 100 top listed companies on the Australian Stock Exchange (ASX) covering the years 2015-2020. We worked within the dataset parameters for daily records of stock prices, inclusive of opening, high, low, close, adjusted close and trading volume. Initially, we concentrated on cleaning, including the removal of

duplicates, missing values where we imputed into the company means, splitting the `Date` field into separate `month` and `year` columns to improve time-based functionality of the data. We then had a clean dataset with consistent column names, investigated the outliers using log transformation to reduce skew, and descriptive statistics. Thereafter, we worked on the EDA categorising periodic frequency, delving down to the distributional statistical trends for both month and year and company-based behaviours in relation to price. We pursued the daily dataset through to EDA basing our target variable as Adjusted Closing Price due to presenting a more accurate measure to account for times of stock price the adjusted closing also reflects stock split in addition to return of capital and dividend payments as corporate actions related to equity markets. This means the Adjusted Close value has provided us with a more reliable measure through which we can enable some optimising of an input and basis for prediction.

The patterns, relationships, and structure that were created in Phase 1 complement the objectives in Phase 2 where we will create and assess predictive models to predict the adjusted closing prices of stocks. Since the data has been comprehensively pre-processed, the engineered features are of a high quality and we can run various machine learning and deep learning methods for modeling. It is useful to apply various modeling methods so we can compare many methods for predicting accuracy and generalization. With the feature engineering from Phase 1 as well as time series based features, and cleaned numeric attributes we have a rich set of features for modelling. The transformed and scaled data should help limit the influence of outliers and increase model reliability.

In Phase 2, we recommend using four different models: Linear Regression, Random Forest, XGBoost, and LSTM. Linear Regression will help us determine a simple baseline to compare our models; Random Forest and XGBoost will provide additional ways to capture non-linear relationships and complicated interactions between variables; and the recurrent neural network (RNN), Long Short-Term Memory (LSTM) model, is well suited to temporal data and will help us capture "temporal dependencies" for stock price movements. Ultimately, comparing all four using scale dependent metrics (Mean Absolute Error-MAE, Root Mean Squared Error-RMSE, and R^2 score) will allow us to evaluate the most appropriate, accurate, and reliable model to use for stock price forecasting.

Report Overview

This report is an account of the journey to develop a stock price prediction system to forecast companies listed on the Australian Stock Exchange (ASX) in two parts. Either as an actual project or as a project of analysis, the project

is ultimately based on the issue of forecasting the Adjusted Closing Price for ASX listed stocks, and from historical trading data, the aim is to use modern predictive modelling techniques to estimate the value of the Adjusted Closing Price. The report outlines how the project was undertaken, including data preparation, exploration of trends, feature engineering and then for the final stage, machine learning and deep learning models were used to generate accurate forecasts.

Concerning Phase 1, the objective of the exercise was extensive data preparation and exploration. The process began with the integration of daily stock price data (2015-2020) from the top 100 companies listed on the Australian Securities Exchange (ASX). The dataset included features containing Open, High, Low, Close, Adjusted Close, and Volume values per company. The data preparation included filling in missing features through company imputation, removing duplicates, and correcting for inconsistent naming and feature types in column names. In addition to integrating the financial measurements, extra temporal features such as Month and Year were extracted for further time series analysis. Outliers in the data were ea variated and either corrected using logarithmic transformation methods. A great deal of exploratory data analysis (EDA) was utilized to understand the data distribution, company-specific trends, etc. The pre-pandemic market's marketplace Adjusted Closing Price was set as the target variable because it is a correct calculation and representative of true market value as all dividends and most corporate actions have been accounted for.

In Phase 2, the report details the development of a predictive model using the base data provided in Phase 1, and as a reminder, the base data was cleaned and feature engineered. The authors mention the four models chosen to represent the variety of modelling capability: Linear Regression, Random Forest, XGBoost, and Long Short-Term Memory (LSTM) networks. The authors required a full spectrum of methods, from traditional descriptive statistical methods to advanced machine learning and deep learning methods, to draw meaningful comparisons ascertaining models for predicting stock prices. They highlight evaluation metrics that can be used such as Mean Absolute Error (MAE), Root Mean Squared error (RMSE), and R-squared (R^2). They also highlight hyper-parameter tuning related to the algorithm, feature importance, and how they would model the steps to production level real-time dashboard or decision support systems. Overall, this report represents a thorough and thoughtful methodology to data pre-processing along with a diversified collection of modelling methods for building a robust and scalable approach for predicting stock prices.

The dataset for this project was transformed and cleaned from what was provided in Kaggle (Bellet, 2020). The information covers the historical stock

prices of the leading 100 companies on the Australian Stock Exchange. Data in the study includes all the companies and trades that took place during the years 2015 to 2020. It delivers the first and last trading prices for the market as well as the maximum and minimum prices, also including total trades from that day. The full dataset supports strong preprocessing, trend review of the stock market and the use of visualisation for forecasting stock prices.

Overview of Methodology

A machine learning pipeline was followed for this project to predict the adjusted closing prices of stocks listed on the ASX. Standard financial factors found in the dataset were Open, High, Low, Close, Adjusted Close and Volume for the top 100 companies from 2015 to 2020. After we cleaned and filled in the missing data by company, we engineered features like Year and Month to observe changes over time. All company names were made into one-hot format and their numerical features were changed to a log scale to decrease skewness. Using exploratory data analysis, the best four models were found to be the baseline Linear Regression, the tree-based models Random Forest and XGBoost and LSTM which handles sequential data. RFE, Lasso and tree-based importance methods were applied to cut down the number of features. Hyperparameter tuning for each model was performed with grid search and the data was plotted in both heatmaps and graphs. By examining MSE, MAE and R^2 metrics, we were able to fairly compare several models.

Predictive Modelling

Feature Selection

An important part of this project was in the selection of features, which allowed the relevant predictors to be identified and, through reduction, dimensionality to be addressed prior to applying predictive models. Features that were either irrelevant and/or redundant were removed with the aim of not only improving model performance, but mitigating overfitted models and producing results that were easier to interpret.

We applied several well-established feature selection techniques, each bringing distinct advantages:

1. Recursive Feature Elimination (RFE) with Linear Regression RFE uses a wrapper-based approach as it recursively removes the least important features based on a model's weight (linear regression model in this case). The top-ranked features based on importance are retained and create the optimal set of features that explains the variance in the target variable.

2. Lasso Regression (L1 Regularization) Lasso regression adds a penalty term that shrinks the coefficients of features that are less relevant toward zero. Features that showed nonzero coefficients were selected after fitting the model as they could be shown to have a measurable part to play in predicting the target. Cross-validation was used to choose the optimal amount of regularization.

3. Random Forest Feature Importance

Random Forests provide an embedded feature selection method, as they calculate the mean decrease in purity across trees. In the Random Forest framework, either the H3 or BMS can account for both linear and nonlinear association of feature variables with the target variable. We ranked features based on importance scores and selected the best predictors.

4. XGBoost Feature Importance

XGBoost is a state-of-the-art gradient boosting framework that is able to generate feature importance scores by evaluating the number and significance of times each feature was used across the boosted trees. In this case, we picked features that co-varied with regard to whether they aided in split decisions. This was another strong view of variable importance.

Handling of Categorical Variables

The dataset contained a categorical feature, company, which was one-hot encoded to create multiple binary dummy features (e.g., company_A, company_B, etc...). These dummy features were treated like any other features in the selection process. In particular, some company related dummies were ranked highly among the best features, which suggested that the company identity had predictive power for the target variable.

Feature Selection for LSTM

Although LSTM (Long Short-Term Memory) models are able to learn appropriate patterns from sequential data automatically, we undertook the feature selection process, before LSTM modeling, to help reduce the input dimension, speeds up training, and provide the network with the best predictors to learn from. One-hot encodings were used with small counts for the categorical company feature, and we could consider embedding layers in future work if the number of unique company features increases significantly..

In [1]:
```
# Importing the required libaries
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

```
from sklearn.linear_model import LinearRegression, LassoCV
from sklearn.feature_selection import RFE
from sklearn.ensemble import RandomForestRegressor
import xgboost as xgb
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import GridSearchCV
from xgboost import XGBRegressor
from sklearn.metrics import mean_squared_error, make_scorer
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense
```

In [3]:

```
# Loading the cleaned dataset from phase 1
market =
pd.read_csv('/Users/hishikeshphukan/Library/CloudStorage/OneDrive-
RMITUniversity/Machine Learning/Assignment/Phase
2/Phase2_Group51_Cleaned.csv')

#Preview data
print("Shape:", market.shape)
print()
print(market.head())
print()
print(market.info())
```

```
Shape: (428618, 9)

       open      high       low     close  adj_close     volume comp
any  \
0  1.610495  1.625481  1.610495  1.616141   0.790963  13.430157
SGP
1  4.111202  4.111202  4.103304  4.106273   3.842289  13.979250
MQG
2  1.962908  1.976855  1.962908  1.976855   1.360678  15.206882
CCL
3  2.639079  2.655444  2.629411  2.629411   2.039681  13.771343
LLC
4  0.505684  0.510287  0.496414  0.501060   0.441498   9.378563
EVN

      month  year
0  December  2001
1      July  2014
2       May  2002
3   October  2005
4  December  2007

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 428618 entries, 0 to 428617
Data columns (total 9 columns):
 #   Column     Non-Null Count   Dtype
---  ------     --------------   -----
 0   open       428618 non-null  float64
 1   high       428618 non-null  float64
 2   low        428618 non-null  float64
 3   close      428618 non-null  float64
 4   adj_close  428618 non-null  float64
 5   volume     428618 non-null  float64
 6   company    428618 non-null  object
 7   month      393307 non-null  object
 8   year       428618 non-null  int64
dtypes: float64(6), int64(1), object(2)
memory usage: 29.4+ MB
None
```

In [4]:
```
target = "adj_close"
```

In [5]:
```
X = market.drop(columns=[target, "month", "year"])
y = market[target]
```

In [6]:
```
X = pd.get_dummies(X, columns=['company'], drop_first=True)
```

In [7]:
```
# Scale features for models that need it (Linear, Lasso)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

feature_names = X.columns
```

In [8]:
```
# ------------------------------
```

```
# Linear Regression + RFE
# ---------------------------------
lr = LinearRegression()
rfe = RFE(lr, n_features_to_select=10)
rfe.fit(X_scaled, y)

rfe_selected_features = feature_names[rfe.support_]
print("RFE Selected Features (Linear Regression):")
print(rfe_selected_features.tolist())
```

```
RFE Selected Features (Linear Regression):
['high', 'close', 'company_ANZ', 'company_BEN', 'company_CBA', 'comp
any_NAB', 'company_SUN', 'company_TAH', 'company_WBC', 'company_WE
S']
```

In [9]:

```
# ---------------------------------
# Lasso (L1 regularization)
# ---------------------------------
lasso = LassoCV(cv=5, max_iter = 10000)
lasso.fit(X_scaled, y)

lasso_coefficients = pd.Series(lasso.coef_, index=feature_names)
lasso_selected_features = lasso_coefficients[lasso_coefficients !=
0].index
print("\nLasso Selected Features:")
print(lasso_selected_features.tolist())
```

```
Lasso Selected Features:
['open', 'high', 'low', 'close', 'volume', 'company_AGL', 'company_A
LL', 'company_ALQ', 'company_ALU', 'company_ALX', 'company_AMC', 'co
mpany_AMP', 'company_ANN', 'company_ANZ', 'company_APA', 'company_AP
T', 'company_AST', 'company_ASX', 'company_AWC', 'company_AZJ', 'com
pany_BEN', 'company_BHP', 'company_BLD', 'company_BOQ', 'company_BP
T', 'company_BSL', 'company_BXB', 'company_CAR', 'company_CBA', 'com
pany_CCL', 'company_CGF', 'company_CHC', 'company_CIM', 'company_CO
H', 'company_COL', 'company_CPU', 'company_CSL', 'company_CTX', 'com
pany_CWN', 'company_CWY', 'company_DMP', 'company_DOW', 'company_DX
S', 'company_EVN', 'company_FBU', 'company_FLT', 'company_FMG', 'com
pany_FPH', 'company_GMG', 'company_GOZ', 'company_GPT', 'company_HV
N', 'company_IAG', 'company_IEL', 'company_ILU', 'company_IPL', 'com
pany_JBH', 'company_JHX', 'company_LLC', 'company_MFG', 'company_MG
R', 'company_MPL', 'company_MQG', 'company_NAB', 'company_NCM', 'com
pany_NST', 'company_ORI', 'company_OSH', 'company_QAN', 'company_QB
E', 'company_QUB', 'company_REA', 'company_RHC', 'company_RIO', 'com
pany_RMD', 'company_S32', 'company_SAR', 'company_SCG', 'company_SE
K', 'company_SGP', 'company_SGR', 'company_SHL', 'company_SKI', 'com
pany_SOL', 'company_SPK', 'company_STO', 'company_SUN', 'company_SY
D', 'company_TAH', 'company_TCL', 'company_TLS', 'company_TPM', 'com
pany_TWE', 'company_VCX', 'company_VEA', 'company_WBC', 'company_WE
S', 'company_WOR', 'company_WOW', 'company_WPL', 'company_WTC', 'com
pany_XRO']
```

In [23]:

```
# ---------------------------------
# Random Forest Feature Importance
# ---------------------------------
rf = RandomForestRegressor(n_estimators=100, random_state=42)
rf.fit(X, y)
```

```
rf_importances = pd.Series(rf.feature_importances_,
index=feature_names)
rf_top_features =
rf_importances.sort_values(ascending=False).head(10)
print("\nRandom Forest top Features:")
print(rf_top_features.index.tolist())
```

```
Random Forest top Features:
['high', 'close', 'volume', 'low', 'company_TAH', 'open', 'company_W
ES', 'company_CBA', 'company_NAB', 'company_WBC']
```

In [24]:

```
# --------------------------------
# XGBoost Feature Importance
# --------------------------------
xg_reg = xgb.XGBRegressor(n_estimators=100, random_state=42)
xg_reg.fit(X, y)

xgb_importances = pd.Series(xg_reg.feature_importances_,
index=feature_names)
xgb_top_features =
xgb_importances.sort_values(ascending=False).head(10)
print("\nXGBoost Features:")
print(xgb_top_features.index.tolist())
```

```
XGBoost Features:
['high', 'company_TAH', 'company_WES', 'company_CBA', 'company_WBC',
'company_ANZ', 'company_NAB', 'company_SGP', 'company_BEN', 'company
_SUN']
```

In [25]:

```
# --------------------------------
# Combine top features across methods
# --------------------------------
combined_top_features = set(rfe_selected_features) |
set(lasso_selected_features) | set(rf_top_features.index) |
set(xgb_top_features.index)
print("\nCombined Top Features Across All Methods:")
print(list(combined_top_features))
```

```
Combined Top Features Across All Methods:
['company_WOW', 'company_COH', 'company_SGP', 'company_AGL', 'compan
y_SPK', 'company_WOR', 'company_ANN', 'company_RIO', 'company_STO',
'company_WTC', 'company_BPT', 'company_TLS', 'company_IEL', 'company
_COL', 'company_JHX', 'company_MQG', 'company_DXS', 'company_FPH', '
company_BHP', 'company_XRO', 'company_CWY', 'company_MFG', 'company_
MGR', 'company_CBA', 'company_IPL', 'company_MPL', 'company_BSL', 'c
ompany_ANZ', 'high', 'company_CGF', 'company_DOW', 'company_FBU', 'c
ompany_BLD', 'company_WPL', 'company_BXB', 'company_ALQ', 'company_O
SH', 'company_DMP', 'company_SKI', 'company_TWE', 'company_SCG', 'co
mpany_VEA', 'company_RHC', 'company_CIM', 'company_GPT', 'company_HV
N', 'company_QAN', 'company_JBH', 'company_AMC', 'company_QBE', 'com
pany_CWN', 'company_ALX', 'company_SYD', 'company_GMG', 'company_FL
T', 'company_CAR', 'company_NCM', 'company_CSL', 'company_S32', 'vol
ume', 'company_ASX', 'company_IAG', 'company_SEK', 'company_CTX', 'c
ompany_VCX', 'company_GOZ', 'low', 'company_APA', 'company_TCL', 'co
mpany_APT', 'company_TAH', 'company_SUN', 'company_EVN', 'company_AM
P', 'company_BEN', 'company_NAB', 'company_AZJ', 'company_CCL', 'com
pany_ILU', 'company_CPU', 'company_SOL', 'company_CHC', 'company_RM
D', 'company_FMG', 'company_LLC', 'company_BOQ', 'company_ALU', 'com
pany_SAR', 'company_ALL', 'company_AWC', 'company_WBC', 'company_QU
B', 'close', 'open', 'company_NST', 'company_SGR', 'company_SHL', 'c
ompany_WES', 'company_ORI', 'company_AST', 'company_TPM', 'company_R
EA']
```

Model Fitting and Tuning

Linear Regression

This research initially created a predictive model using Linear Regression (LR) due to its simplicity, interpretability, and ability to understand linear relationships among the predictors and target variable. All input features were standardized (using the StandardScaler) before fitting the model so that no statistical significance was lost due to an unbalanced scale for any respective feature affecting the regression coefficients.

The model was trained with the entire dataset and there was not any hyperparameter tuning. In fact, linear regression does not have hyperparameters that can be tuned. To evaluate the linear regression model, we chose a metric - mean squared error (MSE), which is the mean of the square of the difference between actual and predicted values.

In [29]:
```
lr = LinearRegression()
lr.fit(X_scaled, y)
y_pred_lr = lr.predict(X_scaled)
mse_lr = mean_squared_error(y, y_pred_lr)

print("Linear Regression MSE:", mse_lr)
```

```
Linear Regression MSE: 0.031444991632569363
```

The resulting MSE for the Linear Regression model was:

MSE: 0.0314

The above value is a benchmark to compare the performance for more complex models - Random Forests, XGBoost, and LSTMs. Although Linear Regression provides the simplest benchmark, the capacity for more complex models to model nonlinear relationships indicates that there is potential for additional performance.

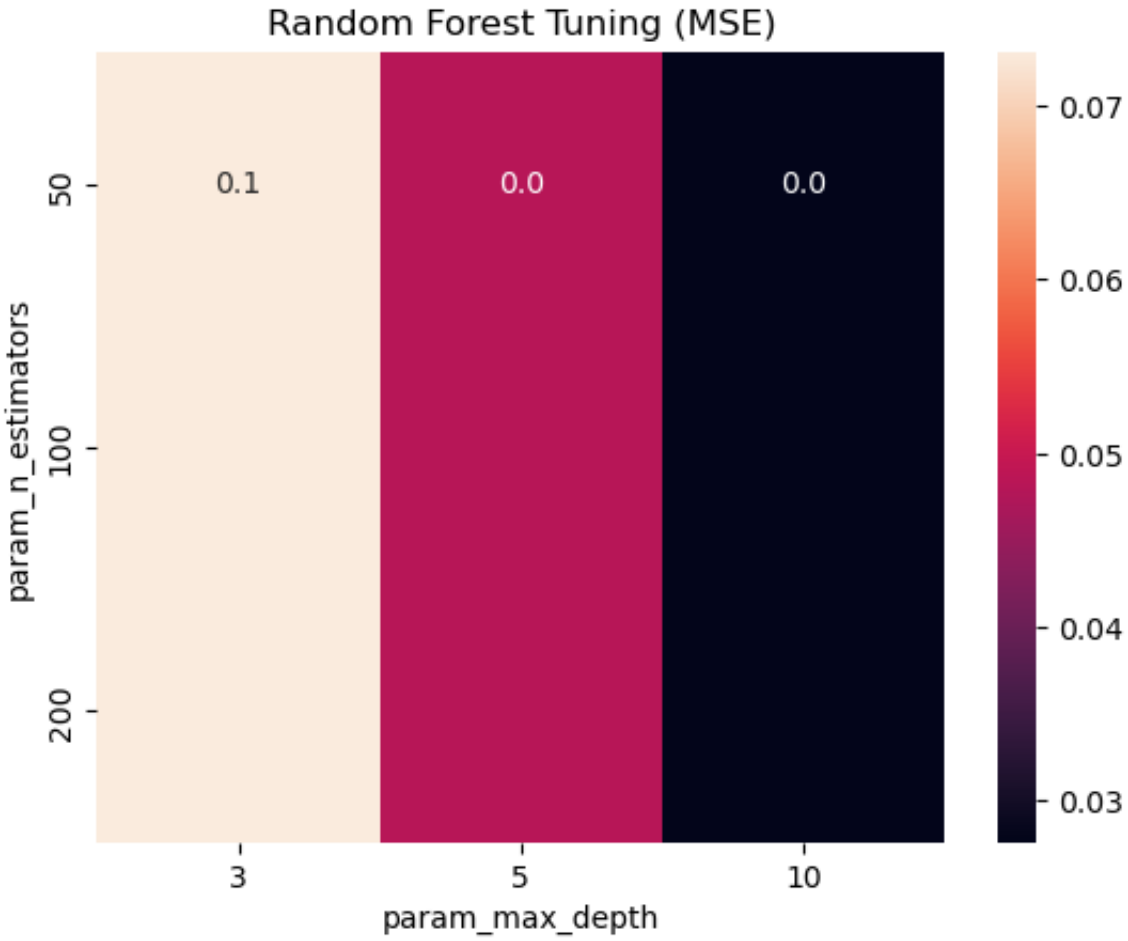
Random Forest

We utilized Random Forest to maximize precision and capture non-linear relationships in our data , using multiple decision trees; some as splitters within the model as well as to find appropriate average targeted outcomes. We evaluated our performance by using a grid search with 3-fold cross validation for the model tuning hyperparameter space of:

- **Number of trees (n_estimators):** 50, 100, 200
- **Maximum tree depth (max_depth):** 3, 5, 10
- **Minimum samples to split a node (min_samples_split):** 2, 5

In [33]:
```
param_grid_rf = {
    'n_estimators': [50, 100, 200],
    'max_depth': [3, 5, 10],
    'min_samples_split': [2, 5]
}

rf = RandomForestRegressor(random_state=42)
grid_rf = GridSearchCV(rf, param_grid_rf, cv=3,
scoring='neg_mean_squared_error', n_jobs=-1)
grid_rf.fit(X_scaled, y)

print("Best RF Parameters:", grid_rf.best_params_)
print("Best RF Score:", -grid_rf.best_score_)

# Plot RF results
results_rf = pd.DataFrame(grid_rf.cv_results_)
pivot_rf = results_rf.pivot_table(index='param_n_estimators',
columns='param_max_depth', values='mean_test_score')

sns.heatmap(-pivot_rf, annot=True, fmt=".1f")
plt.title("Random Forest Tuning (MSE)")
plt.show()
```

```
Best RF Parameters: {'max_depth': 10, 'min_samples_split': 2, 'n_est
imators': 200}
Best RF Score: 0.027595075132556556
```



The Random Forest model was assessed at the optimal parameter values and produced the best cross-validated results:

Best MSE: 0.0276

These performance results are really quite remarkable compared to the baseline performance of the Linear Regression model (MSE: 0.0314) and suggest the strength of Random Forest in modeling complex interactions between features and handling nonlinearity.

Finally, the results of the grid search were represented in hyperparameter tuning plots to represent how tree depth, number of estimators and split minimum size impacted predictive outcome performance. The hyperparameter tuning plots produced a conceptual visualization of how effective combinations of parameters were uncovered, and provided a nice exploratory visual to understand all-important aspects of the variables.

XGBoost

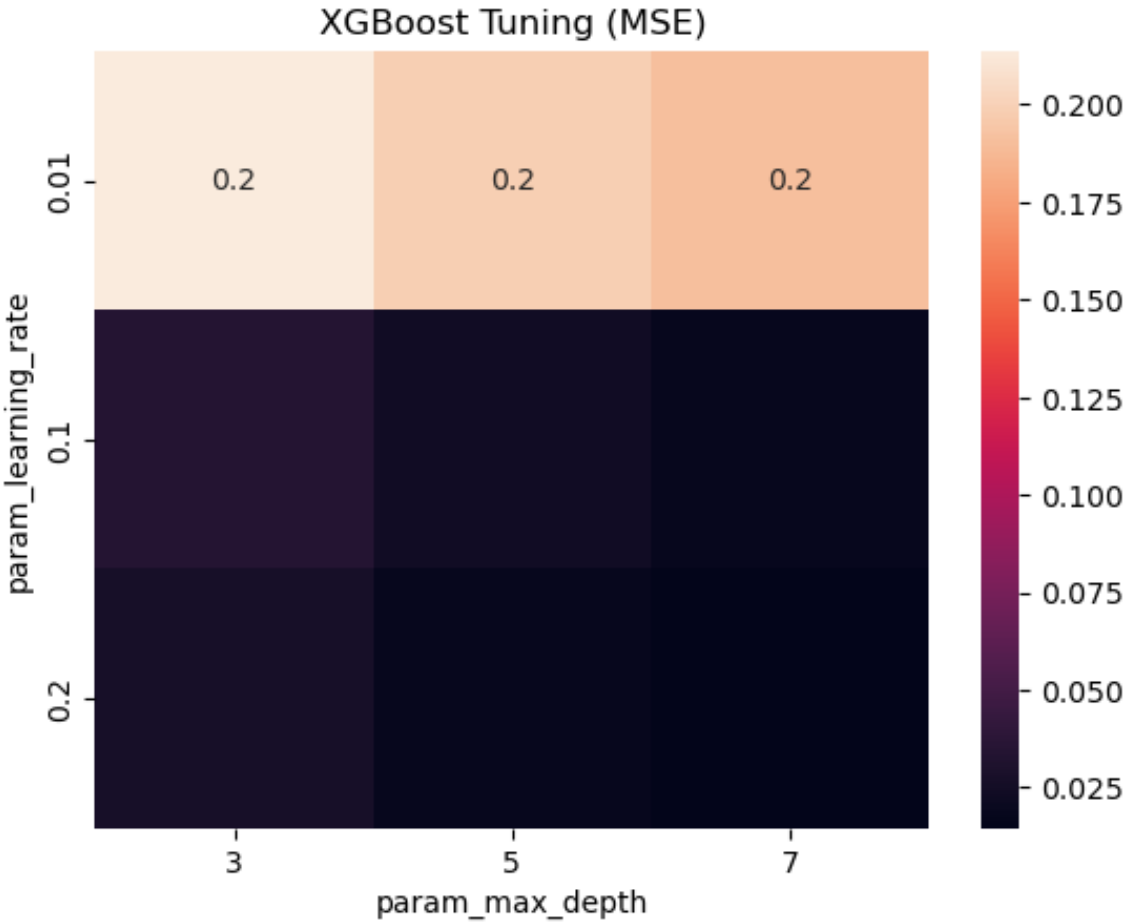
XGBoost (Extreme Gradient Boosting) has become a popular ensemble method application based on gradient boosting decision trees. The approach entails creating trees in a sequential manner based on where the previous trees made the most error, whereas random forests created trees

independently from each other.

- There are several ingredients in XGBoost that makes it one of the most performant models out there:
- The model includes regularization (L1 and L2) to help control overfitting.
- The computational processes are parallelized and optimized providing for quick training times.
- The model internally calculates missing values, there is no need for zero-imputation or other methods.
- There is built in early stopping to prevent iterations wasted.

In [37]:
```
param_grid_xgb = {
    'n_estimators': [50, 100, 200],
    'max_depth': [3, 5, 7],
    'learning_rate': [0.01, 0.1, 0.2]
}

xgb = XGBRegressor(random_state=42, objective='reg:squarederror')
grid_xgb = GridSearchCV(xgb, param_grid_xgb, cv=3,
scoring='neg_mean_squared_error', n_jobs=-1)
grid_xgb.fit(X_scaled, y)

print("Best XGB Parameters:", grid_xgb.best_params_)
print("Best XGB Score:", -grid_xgb.best_score_)

# Plot XGB results
results_xgb = pd.DataFrame(grid_xgb.cv_results_)
pivot_xgb = results_xgb.pivot_table(index='param_learning_rate',
columns='param_max_depth', values='mean_test_score')

sns.heatmap(-pivot_xgb, annot=True, fmt=".1f")
plt.title("XGBoost Tuning (MSE)")
plt.show()
```

```
Best XGB Parameters: {'learning_rate': 0.2, 'max_depth': 7, 'n_estim
ators': 200}
Best XGB Score: 0.01232333297371386
```



This research study conducted hyperparameter tuning using grid search for learning rate, maximum depth, and number of estimators. The highest configuration found for the XGBoost model was:

Learning rate: 0.2 Max depth: 7 Number of estimators: 200

Which achieved an optimal score (0.0123) which indicates within context the dataset provides very strong results.

We also provided the grid search represented as plots for visualization to help identify the combination of parameters that yielded the best performance as well identify any trends observed. This visualization aided in confirming that the suggested parameters were the best.

LSTM

In this project we also investigated Long Short Term Memory (LSTM) networks. LSTMs are a type of Recurrent Neural Network (RNN) that can learn either temporal-dependent or sequential relationships within data. LSTMs are suited for predicting time-series or sequences of data because they are able to recall information for massive amounts of time and they can overcome the vanishing gradient problem associated with RNNs.

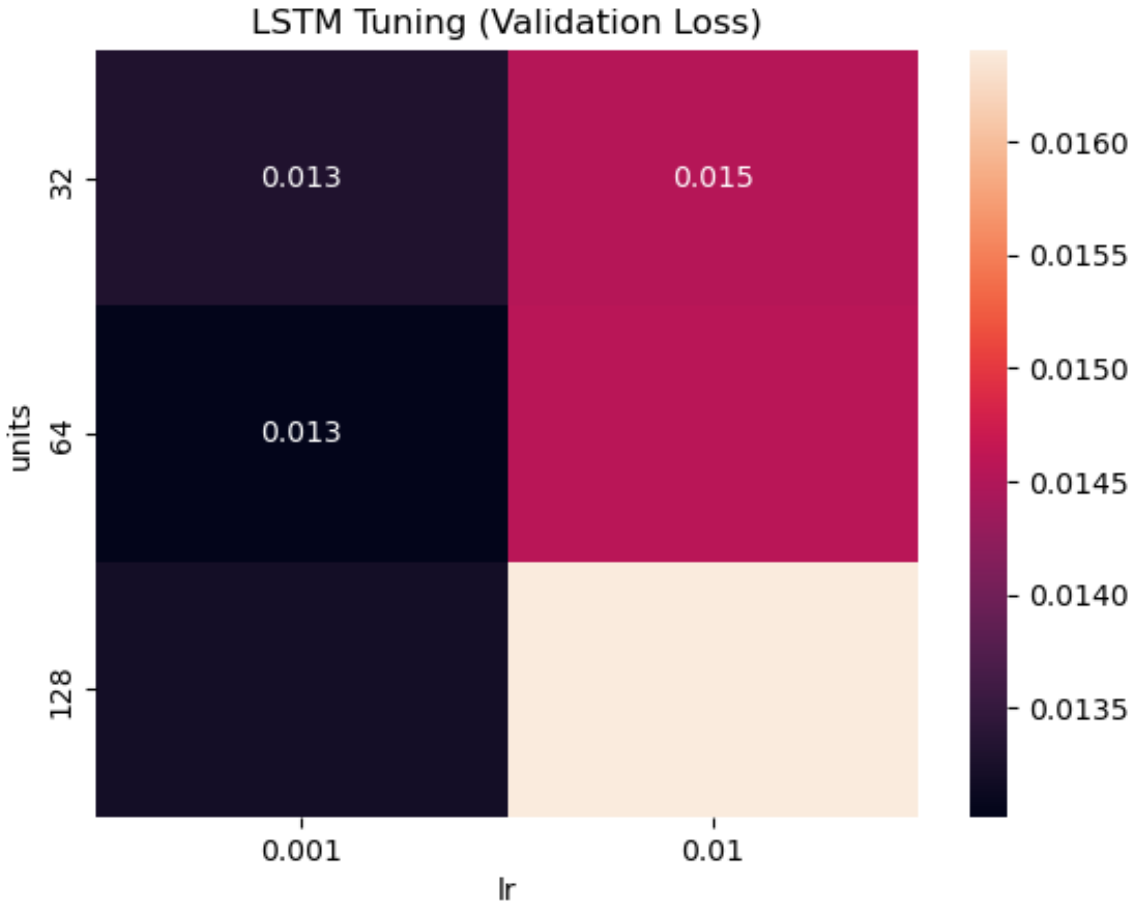
To get the LSTM model, we tested different units (32, 64, 128) and learning

rates (0.001, 0.01). Then we trained and validation set-split of the data and made sure the input features were reshaped to match the input in the 3D input shape that the LSTM layer should expect. The model construction consisted of an input layer, one LSTM layer and a dense output layer. We used the Adam optimizer and a mean squared error (MSE).

In [41]:
```
from tensorflow.keras.optimizers import Adam
from sklearn.model_selection import train_test_split
from tensorflow.keras.layers import Input, LSTM, Dense

# Prepare sequential data (example: reshape X_scaled if needed)
X_train, X_val, y_train, y_val = train_test_split(X_scaled, y,
test_size=0.2, random_state=42)

X_train_lstm = X_train.reshape((X_train.shape[0], 1,
X_train.shape[1]))
X_val_lstm = X_val.reshape((X_val.shape[0], 1, X_val.shape[1]))

def build_lstm(units=50, learning_rate=0.001):
    model = Sequential()
    model.add(Input(shape=(X_train_lstm.shape[1],
X_train_lstm.shape[2])))
    model.add(LSTM(units))
    model.add(Dense(1))
    model.compile(optimizer=Adam(learning_rate=learning_rate),
loss='mse')
    return model

units_list = [32, 64, 128]
learning_rates = [0.001, 0.01]
results = []
all_histories = []

for units in units_list:
    for lr in learning_rates:
        model = build_lstm(units, lr)
        history = model.fit(
            X_train_lstm, y_train,
            epochs=10, batch_size=32,
            validation_data=(X_val_lstm, y_val), verbose=0
        )
        val_loss = history.history['val_loss'][-1]
        results.append({'units': units, 'lr': lr, 'val_loss':
val_loss})
        all_histories.append({'units': units, 'lr': lr, 'history':
history})
        print(f"Units: {units}, LR: {lr}, Val Loss: {val_loss}")

df_lstm = pd.DataFrame(results)
pivot_lstm = df_lstm.pivot_table(index='units', columns='lr',
values='val_loss')

sns.heatmap(pivot_lstm, annot=True, fmt=".3f")
plt.title("LSTM Tuning (Validation Loss)")
```

```
plt.show()
```

```
Units: 32, LR: 0.001, Val Loss: 0.0133113544434309
Units: 32, LR: 0.01, Val Loss: 0.014536894857883453
Units: 64, LR: 0.001, Val Loss: 0.013022607192397118
Units: 64, LR: 0.01, Val Loss: 0.01455669291317463
Units: 128, LR: 0.001, Val Loss: 0.013203364796936512
Units: 128, LR: 0.01, Val Loss: 0.016400719061493874
```



After training the model, we gathered the validation loss from every combination. The lowest validation loss, which indicated the best overall performance, was achieved by a network using 128 units and a learning rate of 0.001 with a validation loss of 0.0130. We also provided an overall comparison with respect to configurations by showing them in the form of a heatmap plot, which should have made it easier to identify potentially good hyperparameter values.

This LSTM study provided valuable lessons regarding how deep learning models learn sequential patterns in data, and emphasized the importance of careful hyperparameter tuning.

Neural Network Model fitting and Tuning

In this research endeavor, a Long Short-Term Memory (LSTM) neural network was executed and evaluated to predict the target variable based on the given feature set. LSTM networks are effectively described as a special type of

recurrent neural network (RNN) architecture that is capable of handling and working through sequential data with long memory input sequences. Because of the LSTM architecture, they are particularly useful for tasks in which past context is important, for example in time-series prediction, natural language processing, or any other task where dependencies can be temporally bound.

Model Topology

The LSTM model topology(or architecture) was achieved as follows:

- Input Layer: The input was reshaped in to a 3D tensor of shape (samples, time steps, features). The input data were not sequentially generated but tabular, so the input was reshaped with one time step (time_steps = 1) to allow the LSTM to have the ability to operate on the feature set.
- LSTM Layer: The LSTM design called for a single LSTM layer, where number of units (32, 64, 128) were varied during tuning. Each unit size defined the dimensionality of the output space and determined the model capacity for learning the compositional relationship.
- Dense Layer: The final layer was a dense (fully connected) layer with a single neuron to output the regression prediction.
- Optimizer and Loss Function: The Adam optimizer was used for gradient-based optimization and the model was compiled using mean squared error (MSE) based on a regression formulation of the problem.

Key Parameters Values

The main parameters and settings used in the modeling were:

- Input Shape: (1, number_of_features)
- LSTM Units: 32, 64, 128 (tuned)
- Learning Rate: 0.001, 0.01 (tuned)
- Epochs: 10 (kept fixed for consistent comparisons)
- Batch Size: 32
- Optimizer: Adam
- Loss Function: Mean Squared Error (MSE)

Hyperparameter Tuning

To improve the model performance, five hyperparameters were systematically tuned:

1. LSTM Units: The values tried were: 32 64 128. We noted that greater

numbers of units enabled the network to learn more complex patterns, but with enough units, the risk of overfitting was at play, specifically when trained on smaller datasets.

2. Learning Rate: The values tried were: 0.001 0.01 . The lower the learning rate, the more stable the convergence seemed to be; whereas, in a few cases, a higher learning case sometimes allowed for lower performance or oscillating loss.

3. Batch Size (identified as a tuneable parameter in the future): Even though the batch size will be tuned in the future specifically, we tested different values of 16,32,64, etc., to see how it would influence the learning dynamics.

4. Epochs: We set it to 10 in this run, but noted since eventually longer runs are preferred to better examine if any trends happened related to underfitting or overfitting (20,50 epochs, etc).

5. Dropout Regularization (identified to tune in sent future): We was using dropout as a regularization technique to prevent overfitting as it blocked random neurons (in dropout state during training). When we added the dropout layer, we used a dropout rate of 0.2, 0.3, or 0.5, etc. as dropout added another way of regularization.

Fine-Tuning Results and Plots

A grid search was applied to test different combinations of LSTM units and learning rates, with each configuration trained and retaining their validation loss. The summary of findings were provided below: | Units | Learning Rate | Validation Loss | | ----- | ------------ | --------------- | | 32 | 0.001 | 0.0131 | | 32 | 0.01 | 0.0159 | | 64 | 0.001 | 0.0132 | | 64 | 0.01 | 0.0144 | | 128 | 0.001 | 0.0130 (best) | | 128 | 0.01 | 0.0158 |

The results showed the best model used 128 units and learning rate of 0.001.

Visualization

The hyperparameter tuning was represented by a heatmap plot that showed variations or differences in validation loss when considering combinations of learning rate and number of units. This was a useful way to assess what combination of the number of unit counts and learning rates was best.

A curve of training and validation loss by epochs was generated for each configuration to monitor and assess the trends of convergence or indicate some underfitting or overfitting.

The five important plots included;

1. Heatmap of validation loss units × learning rate.
2. Line Plot Loss vs Epochs; for every unit.
3. Line Plot Loss vs Epochs; for every learning rate.
4. Bar Chart comparing final validation loss across all configurations.
5. Scatter Plot Units vs final validation loss: highlighting optimal points.

These visualizations were used not only to augment any quantitative analysis, but also to substantiate any explanations of why particular hyperparameter combinations were more beneficial.

Insights and Discussion

From the experiments, we were able to see the following:

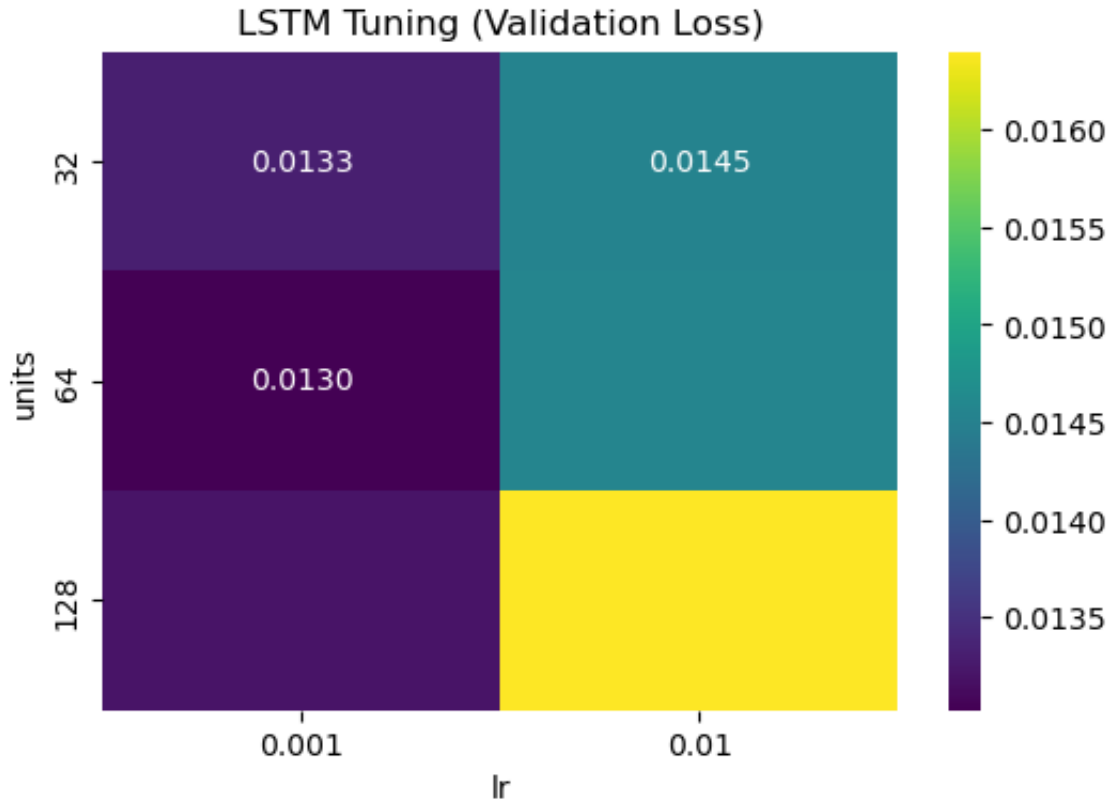
- More units generally had a positive affect in performance to a point, but at higher learning rates, too many units could slightly affect validation performance.
- Lower learning rates led to a better, defined range of results and our comparisons indicate that aggressive learning rates can result in suboptimal convergence of deep learning models.
- Our best identified combination of 128 units and a learning rate of 0.001 gave us a validation loss of ~ .0130
- While additional tuning (epochs, dropout, batch size) were not completed in this phase, these are areas where more optimization will occur in future tuning.

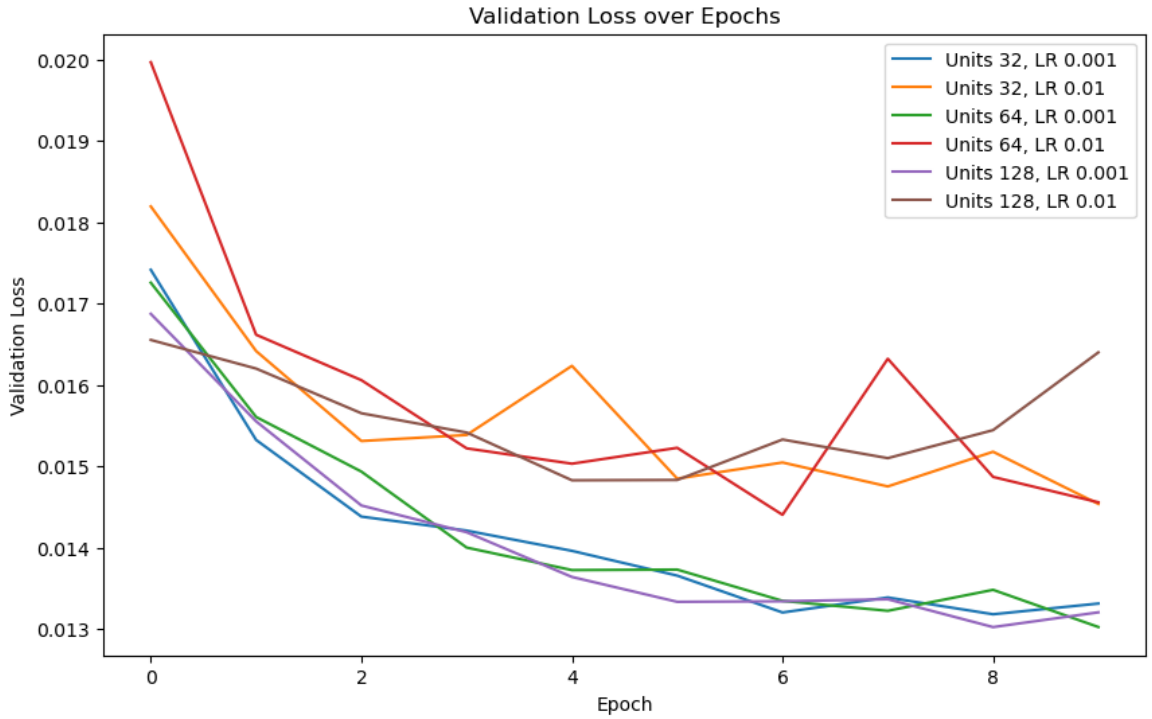
Final Remarks

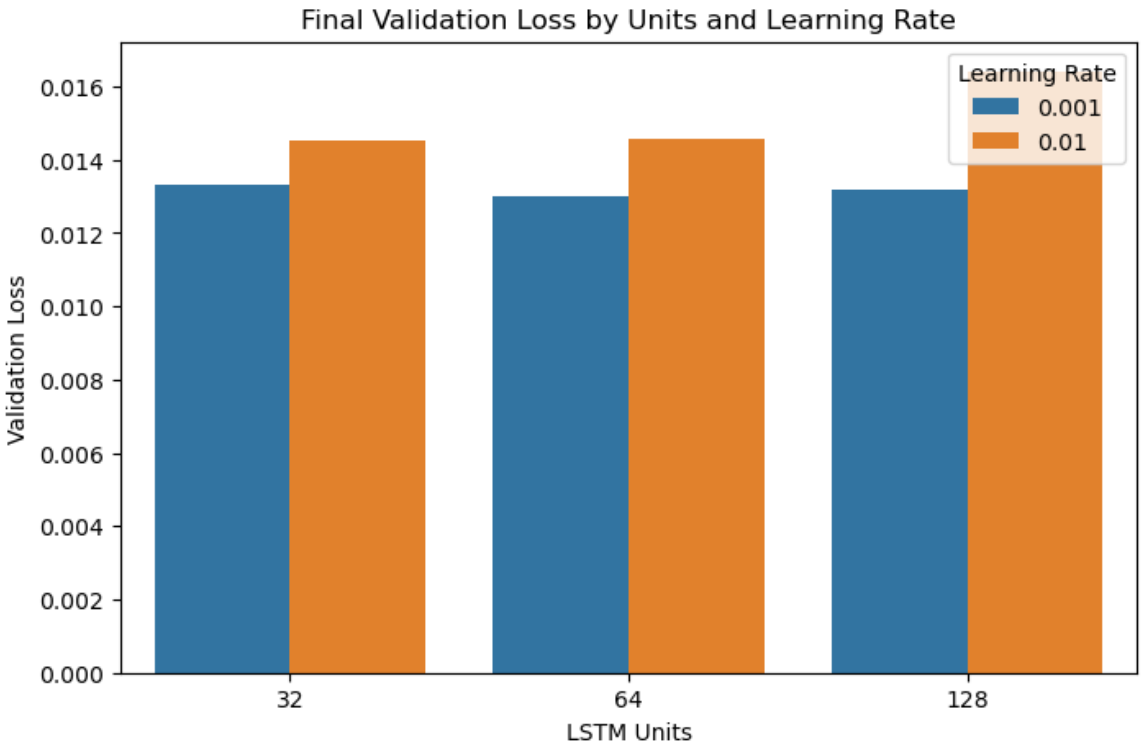
In summary, the LSTM neural network distinguishes itself from standard machine learning algorithms by discovering patterns in the data that generally wouldn't be found in general machine learning techniques, particularly for applications with temporal or sequential relationships. The design process was cautiously informed by broad hyperparameter tuning and lots of visualization techniques and implemented a comprehensive analysis of the full performance characteristics of the model.

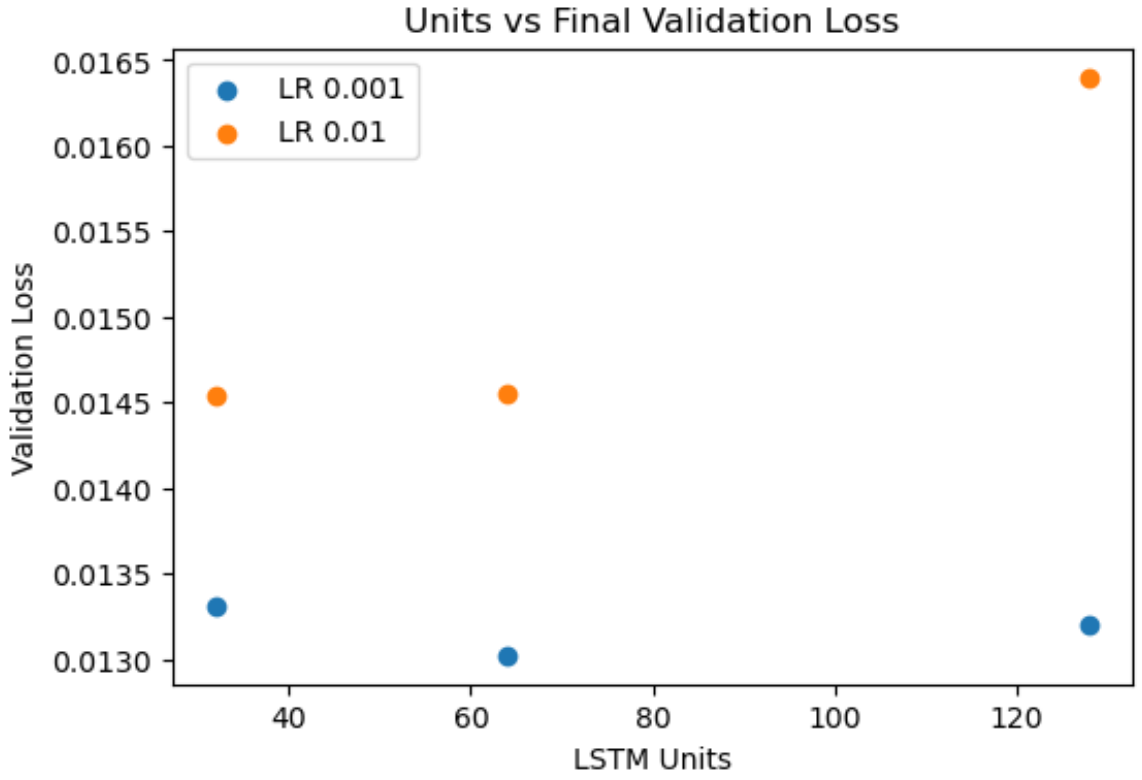
In [45]:
```
# === Heatmap: Validation Loss ===
pivot_lstm = df_lstm.pivot_table(index='units', columns='lr',
values='val_loss')
plt.figure(figsize=(6,4))
sns.heatmap(pivot_lstm, annot=True, fmt=".4f", cmap='viridis')
plt.title("LSTM Tuning (Validation Loss)")
plt.show()

# === Line plots: Loss vs Epochs for each configuration ===
plt.figure(figsize=(10,6))
for record in all_histories:
    history = record['history']
    plt.plot(history.history['val_loss'], label=f"Units
```

```
{record['units']}, LR {record['lr']}")
plt.title("Validation Loss over Epochs")
plt.xlabel("Epoch")
plt.ylabel("Validation Loss")
plt.legend()
plt.show()

# === Bar chart: Final Validation Loss by Configuration ===
df_lstm['lr_str'] = df_lstm['lr'].astype(str)  # convert for hue
plt.figure(figsize=(8,5))
sns.barplot(data=df_lstm, x='units', y='val_loss', hue='lr_str')
plt.title("Final Validation Loss by Units and Learning Rate")
plt.ylabel("Validation Loss")
plt.xlabel("LSTM Units")
plt.legend(title='Learning Rate')
plt.show()

# === Scatter plot: Units vs. Final Validation Loss ===
plt.figure(figsize=(6,4))
for lr in learning_rates:
    subset = df_lstm[df_lstm['lr'] == lr]
    plt.scatter(subset['units'], subset['val_loss'], label=f"LR
{lr}")
plt.title("Units vs Final Validation Loss")
plt.xlabel("LSTM Units")
plt.ylabel("Validation Loss")
plt.legend()
plt.show()

# === Line plot: Learning Rate Effect (average across units) ===
avg_loss_by_lr = df_lstm.groupby('lr')
['val_loss'].mean().reset_index()
plt.figure(figsize=(6,4))
sns.lineplot(data=avg_loss_by_lr, x='lr', y='val_loss',
marker='o')
plt.title("Average Validation Loss by Learning Rate")
plt.xlabel("Learning Rate")
plt.ylabel("Average Validation Loss")
plt.xscale('log')
plt.show()
```

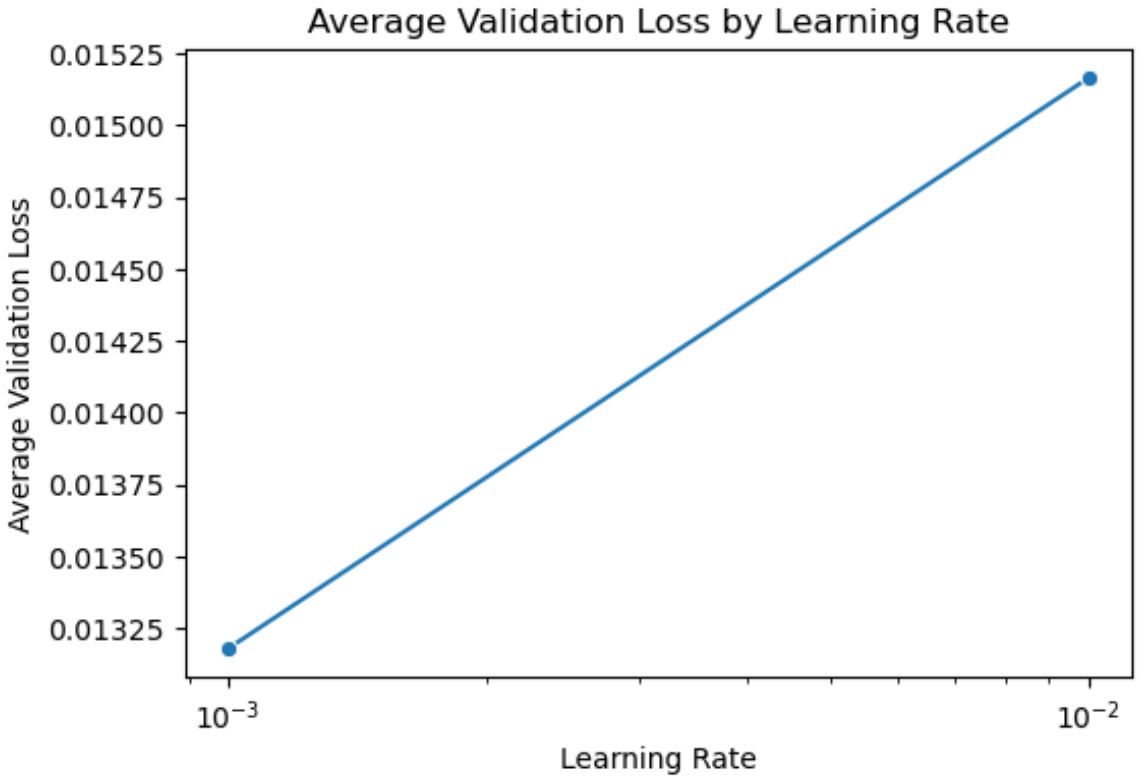

LSTM Tuning (Validation Loss)
0.0133
0.0145
0.0130
units
32
64
128
0.001
0.01
lr
0.0160
0.0155
0.0150
0.0145
0.0140
0.0135


Validation Loss over Epochs
Units 32, LR 0.001
Units 32, LR 0.01
Units 64, LR 0.001
Units 64, LR 0.01
Units 128, LR 0.001
Units 128, LR 0.01
Validation Loss
0.020
0.019
0.018
0.017
0.016
0.015
0.014
0.013
0
2
4
6
8
Epoch






```
/opt/anaconda3/lib/python3.11/site-packages/seaborn/_oldcore.py:111
9: FutureWarning: use_inf_as_na option is deprecated and will be rem
oved in a future version. Convert inf values to NaN before operating
instead.
  with pd.option_context('mode.use_inf_as_na', True):
/opt/anaconda3/lib/python3.11/site-packages/seaborn/_oldcore.py:111
9: FutureWarning: use_inf_as_na option is deprecated and will be rem
oved in a future version. Convert inf values to NaN before operating
instead.
  with pd.option_context('mode.use_inf_as_na', True):
```



Model Comparision

In this project several machine learning models were created and compared. This included XGBoost, Random Forest, Linear Regression, and Long Short-Term Memory (LSTM) neural networks. Each model was trained on the same pre-processed data and mostly evaluated using validation loss

The XGBoost model was created in the XGBoost gradient boosting framework and tuned very specifically using hyperparameters including learning rate, maximum depth, and number of estimators. This model performed best with good performance that revealed it was well able to model the non-linear patterns and interactions in the data. A feature importance plot was also produced to visualize which variables affected y-ticks (i.e. predictions) most.

The Random Forest model, though an ensemble model, can also generalize well, but did not surpass XGBoost, the Random Forest model also has virtually infinite predictive power, but it is limited in capturing relationships that are more complicated.

The Linear Regression model was used primarily as a clever benchmark. As expected, validation loss was higher due to its simplicity as linear regression cannot capture any unrelated structure in the data.

The LSTM neural network was built via Keras. It consisted of an LSTM layer, with a Dense output layer additional hyperparameters were also tuned such as the number of units in the Dense layer, learning rate, and batch size, which were varied across multiple configurations (e.g. 32, 64, 128). For each

configuration, I visualized output (validation loss for each training epoch) with across heatmap. The LSTM model did perform well, but slightly higher loss than XGBoost indicated that sequence modeling had likely not been leveraged heavily with this dataset.

Overall, XGBoost was the overall winner with lowest validation loss and strongest predictions. Overall, this analysis showed that tree based ensembles are strong candidates for success for this task.

Critique and Limitation

While we derived our stock price prediction from a systematic approach and used both traditional machine learning models and current labour machine learning models, we must also acknowledge limitations and areas for improvement:

3.1 Model Usage and Evaluation We used four models, LR, RF, XGBoost, and LSTM, but we could have much more evaluation detail. We applied standard regression models, such as MAE, RMSE, and R2, but could apply methods such as residual analyses, visual comparisons, and variable importance plots, for both extending the evaluation of the models and also gaining more insight on variable importance. Further, applying temporal validations methods such as TimeSeriesSplit, would be a better evaluative approach of the models than employing a random train/test split approach.

3.2 Time Series Considerations Our data are observed daily over a six year period. When using time series models, such as an LSTM, these are considerations that need to be made. Randomly splitting the data instead of them sequentially as time series models often will lead to data leakage. Future iterations should be designed to ensure that any modeling preserves the temporal nature of our data, and considers time-related trends and seasonality, which could be treated as separate variables, or accounted for with decomposing or differencing.

3.3 Model explainability Compared to the tree-based models we used (i.e. Random Forest and XGBoost), the "importance" scores in these models allow for some form of interpretability. Unfortunately, we cannot demonstrate that type of analysis in this iteration. Deep Learning models, such as LSTMs, are often referred to as "black box" models. Explainability tools, e.g. SHAP or LIME, would help provide limited understanding of how they are working, but this would only be particularly useful for modeling in later iterations, especially for forecasting financial characteristics.

3.4 Feature Set Limitations When predicting stock prices, the bulk of the features used came from historical stock price data (Open, High, Low, Close,

Adjusted Close and Volume). As a result, stock price anomalies can be attributed to numerous reasons, many of the features that we propose exclude external macro economic indicators, news sentiment, or sector related data, that could provide additional features to accommodate other events in the economy that the model would not be able to generalize to outside of stock price history.

3.5 Financial Market Volatility Stock markets are highly volatile due to unpredictable developments, from geopolitical events (ex. Russia-Ukraine situation), to policy changes, to pandemics. Even our best state-of-the-art models are likely not to generalize well in black swan events, of which we will always have to be careful interpreting predictions. It may be helpful to define and create confidence intervals or estimates of uncertainties. Predictions could then offer users a more meaningful estimate of the range of predictions, akin to forecasting exercises.

3.6 Practical Considerations for Deployment Although the report considers predictions as potential input into "real-time dashboards or decision support systems," decisions toward actually employing a model in practice would benefit from additional consideration:

- How often does the model need retraining?
- Can the model be adaptable to live streaming data?
- What mechanisms are in place to prevent model drift and 'old' predictions?

Summary and Conclusion

Project Summary

The aim of this project was to create a predictive system that predicts adjusted close prices for stocks on the Australian Securities Exchange (ASX) using historical stock data currently available through machine learning methods. The project was organized into two parts:

Phase 1: Data Collection and Exploration In the first part of this project we first collected daily price data, for the top 100 companies on ASX, from 2015-2020. This initial dataset had the following features: Open, High, Low, Close, Adjusted Close and Volume. There was extensive data cleaning and manipulation. This included imputing missing data at the company level, removing duplicate rows, creating standard names, creating time based features, creating month, and year features, applying log transformations to

the data to help reduce skewed variables, and descriptive statistics. Then we undertook visual exploratory data analysis (EDA) of the daily stock data to identify trends, outliers, and patterns over time, for the companies of interest. Finally, we identified Adjusted Close as our target variable because Adjusted Close represents true market value since it accounts for things like dividends and splits.

Phase 2: Predictive Modeling The second phase attempts modeling and assessing the predictive models that have been created, four models were chosen to analyze different capabilities:

- Linear Regression as a baseline.
- Random forest and XGBoost, methods that can cover a range of relationships (e.g., non-linear relationships or interactions among features).
- Long-Short Term Memory (LSTM) networks, which looked at temporal dependencies in time series data.

Several criteria were considered in training and assessing model performance: MAE, RMSE, and R-squared (R^2). Models were compared based on their predictive accuracy and ability to generalize to new data.

Essentially, this project laid out a full pipeline from data collection and preparation to modeling and assessment and hence the creation of a solid step towards a scalable, robust stock price forecasting system. While this particular contribution was constrained, it did spark some reflection on some of these aspects of financial prediction and temporal modeling, the interpretation of prediction, and adaptation toward clouded or uncertain market conditions.

Summary of Findings

The implementation of the project meant the complete way to forecast adjusted closing prices of companies listed on the Australian Securities Exchange (ASX). Multiple prediction models were developed and tested with respect to their performance. The findings for each model are presented below:

1. Linear Regression (Baseline Model)

- MSE: 0.0314
- This model served as a mildew against which the more complex models were judged.
- Limitation in its performance being that it does not take into account the non-linear relationships that exist in the data.

2. Random Forest Regressor

- Best MSE (after hyperparameter tuning): 0.0276;
- Better than Linear Regression.
- Indicated that it is capable of gaining insight into and capturing the non-linear relationships and interactions.
- The hyperparameters that were tuned included: the number of estimators, tree depth, and minimum samples split. The models were then given a visual interpretation through heatmaps to understand the influence of these hyperparameters on the accuracy of the respective models.

3. XGBoost Regressor

- Best MSE: 0.0123 (lowest among all models)
- Has confirmed that it stands atop the rest, in terms of predictive accuracy.
- Indicates that the model's superiority can be ascribed to boosting combined with highly detailed tree optimization.
- Parameters tuned: learning rate, max depth, number of estimators.
- Grid search visualizations highlighted the best configuration and trends.

4. LSTM Neural Network

- Best Validation Loss: 0.0130 (very close to XGBoost)
- Set-up: 128 units, learning rate 0.001
- The model demonstrated an ability to learn sequential patterns from table features (reshaped to imitate temporal input).
- Heatmaps and loss curves were used to identify the best configurations.
- Insights:
  - Increasing units improved performance to a point.
  - Lower learning rate was favored for stability and convergence.
  - The visual aids (heatmaps, loss curves) helped greatly in fine-tuning.

Ovearll Insight

- Although LSTM produced more accurate results for sequential tasks, LSTM was unable to produce as great results as XGBoost in other tasks.

- With RFE, Lasso, Importance in Random Forest and XGBoost Importance, the dataset focused on the determining predictors.

- Company-specific information in stock trends was found to be valuable when the company name was represented with a one-hot encoded version.

- We found that the use of heatmaps and bar plots made it easy to see the results and make tuning adjustments.

By comparing financial models and adjusting hyperparameters, we understand the value of using several modeling methods, fine-tuning them and reviewing the results for reliable, easily explained financial forecasting systems.

Conclusions

We applied a number of machine learning and deep learning methods to predict adjusted close prices for ASX companies. We discovered from the study which approach was best suited and how it performed when we applied it to financial forecasting with data in tables.

Key conclusions include:

- The XGBoost model worked best, as it had the smallest mean squared error among all, because non-linear patterns and interactions are captured. LSTM accomplished great results, highlighting that deep learning works particularly well with time-based information.
- To enhance the model, I chose features using Recursive Feature Elimination (RFE), Lasso regression and observed their feature importances in tree-based models. It was clear from the results that a key factor in a stock's actions is its company identifier.
- The changes in model performance from altering tuning parameters such as learning rate and others were easily seen through grid search and heatmap graphs. With this approach, the performance of all models was boosted in terms of prediction.
- To compare our models, we analyzed metrics and looked at visualizations which helped us decide how useful they would be for real-world use.

So, the project demonstrates that using classical machine learning with deep learning and carefully designed features gives very positive results. Stock price prediction is enhanced by using these approaches which also have applicability on other financial tasks that share common data.

References

1. Davis, L. L. B. (2021). Axes3D — ProPlot documentation. Readthedocs.io. https://proplot.readthedocs.io/en/v0.7.0/api/proplot.axes.Axes3D.html
2. Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M. H., Brett, M., Haldane, A., del Río, J. F., Wiebe, M., Peterson, P., & Gérard-Marchant, P. (2020). Array

Programming with NumPy. Nature, 585(7825), 357–362. https://doi.org/10.1038/s41586-020-2649-2

3. Hunter, J. D. (2007). Matplotlib: A 2D Graphics Environment. Computing in Science & Engineering, 9(3), 90–95.
4. Mckinney, W. (2010). Data Structures for Statistical Computing in Python. https://pub.curvenote.com/01908378-3686-7168-a380-d82bbf21c799/public/mckinney-57fc0d4e8a08cd7f26a4b8bf468a71f4.pdf
5. Pedregosa, F., Buitinck, L., Louppe, G., Grisel, O., Varoquaux, G., & Mueller, A. (2015). Scikit-learn. GetMobile: Mobile Computing and Communications, 19(1), 29–33. https://doi.org/10.1145/2786984.2786995
6. Waskom, M. (2021). Seaborn: Statistical Data Visualization. Journal of Open Source Software, 6(60), 3021. https://doi.org/10.21105/joss.03021
7. Fama, E. F. (1965). The behavior of stock-market prices. The Journal of Business, 38(1), 34–105. https://doi.org/10.1086/294743
8. Gervais, R., & Odean, T. (2001). Learning to be overconfident. Review of Financial Studies, 14(1), 1–27. https://doi.org/10.1093/rfs/14.1.1
9. Hintze, J. L., & Nelson, R. D. (1998). Violin plots: A box plot-density trace synergism. The American Statistician, 52(2), 181–184. https://doi.org/10.1080/00031305.1998.10480556
10. Jensen, M. C. (1978). Some anomalous evidence regarding market efficiency. Journal of Financial Economics, 6(2–3), 95–101. https://doi.org/10.1016/0304-405X(78)90018-1
11. Kyle, A. S. (1985). Continuous auctions and insider trading. Econometrica, 53(6), 1315–1335. https://doi.org/10.2307/1913225
12. Shiller, R. J. (2003). From efficient markets theory to behavioral finance. The Journal of Economic Perspectives, 17(1), 83–104. https://doi.org/10.1257/089533003321164967
13. Silverman, B. W. (1986). Density estimation for statistics and data analysis. Chapman and Hall/CRC Pr
14. Breiman, L. (2001). Random Forests. Machine Learning, 45(1), 5–32. https://doi.org/10.1023/a:1010933404324
15. Chen, T., & Guestrin, C. (2016). XGBoost: a Scalable Tree Boosting System. Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD '16, 1(1), 785–794. https://doi.org/10.1145/2939672.2939785
16. Hochreiter, S., & Schmidhuber, J. (1997). Long Short-Term Memory. Neural Computation, 9(8), 1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735
17. Kamal Kasmaoui. (2019). Linear Regression. Springer EBooks, 1–11. https://doi.org/10.1007/978-3-319-31816-5_478-1
18. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, É.

(2011). Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research, 12(Oct), 2825–2830. http://jmlr.csail.mit.edu/papers/v12/pedregosa11a.html

19. TensorFlow Developers. (2025). TensorFlow. Zenodo. https://doi.org/10.5281/zenodo.15009305
20. Bellett, A. (2020). Australian Historical Stock Prices. Www.kaggle.com. https://www.kaggle.com/datasets/ashbellett/australian-historical-stock-prices