Methodology and Experiments

The system is divided into three parts,

1. Financial Risk Tolerance Level Prediction
2. Equity Forecasting model
3. Advanced mutual funds information search.

Financial Risk Tolerance Level Prediction

This module is the first step in the whole portfolio recommender pipeline. In this module we determine the Financial Risk Tolerance level of an individual by different forms of regression models to achieve maximum accuracy.

The dataset we have consists of following columns after data preprocessing:

'Gender', 'Car_Owner', 'Property_Owner', 'Children', 'Annual_income', 'Income_source', 'Education','Marital_status', 'Housing_type', 'Birthday_count', 'Employed_days',

'Occupation_type', 'Family_Members', 'Age', 'Employed']

I had to perform some more steps for preprocessing the data for getting it ready for the regression model, these include:

1. Occupation_type column consisted of some empty rows, so I randomly added some occupations from the other non-empty rows.
2. I created a new column Cost_of_living, which actually determines the cost of living an individual can afford for it's family members based on it's income.

   Cost_of_living = Total Annual Income / Total no. of Family member
3. In the next step I categorized the Occupation_type column into 3 different categories namely, Professional/Managers, Skilled Labour, Unskilled Labour and added them into a new column named Occupation_category. The clear bifurcation of these categories is mentioned below:
   - **Professional/Management**: Managers, High skill tech staff, Accountants, Medicine Staff, HR Staff, IT Staff
   - **Skilled Labour**: Sales staff, Core staff, Security staff, Cooking staff, Private service staff, Secretaries, Realty agents
   - **Unskilled Labour**: Laborers, Drivers, Cleaning staff, Low-skill labours, Waiters/barmen staff

4. Next, I categorized Education column into three categories, Higher Education, Secondary Education and Lower Secondary Education and added them into a new column Education_category. The detailed division is shown below:
    - **Higher Education**: Higher education, Academic degree
    - **Secondary Education**: Secondary / secondary special, Incomplete higher
    - **Lower Secondary Education**: Lower secondary
5. After getting all these columns I calculated the Financial Risk Tolerance Score of all these individuals based on the following table below. In the table below the columns include:
    - Column name represents the actual column name
    - Options refers to what all unique values that column has.
    - Weight represents how much weightage that column has on the score.
    - Score represents the score for that option.
    - Total column is multiplication of weight and score.

Column name	Options		Weight	Score	Total
Cost_of_living (Annual_income / Family_members)	<50000		3	10	30
	50000 - 100000			25	75
	100000 - 150000			40	120
	150000 - 200000			60	180
	200000-250000			80	240
	>250000			100	300
Car_Owner	Y		1	100	100
	N			50	50
Property_Owner	Y		3	100	300
	N			25	75
Employed	Y		2	100	200
	N			25	50
Marital_status	Married		1	50	50
	Single / not married			100	100
	Civil Marriage			50	50
	Sperated			25	25
	Widow			10	10
Age	<25		1	100	100
	25-40			80	80
	40-50			60	60
	50-60			40	40
	>60			20	20
Housing_type	House / apartment		2	100	200
	With parents			90	180
	Mumcipal apartment			25	50
	Rented apartment			40	80
	Office apartment			60	120
	Co-op apartment			90	180
Occupation_category & Education_category	Professional/Management	Higher Education	2	100	200
		Secondary Education		75	150
		Lower Secondary		50	100
	Skilled Labour	Higher Education		90	180
		Secondary Education		60	120
		Lower Secondary		30	60
	Unskilled Labour	Higher Education		50	100
		Secondary Education		30	60
		Lower Secondary		15	30
		Total Weight:	**15**		

The financial risk tolerance score is calculated by adding all the total “score * weight” for each column based on the options and/or range he/she falls into. The table above is also available in this google sheet.

6. After getting the financial risk tolerance, I divided the individuals into 6 different risk categories, Very high risk, High risk, Medium to high risk, Medium risk, Low to medium risk, low risk based on their financial risk tolerance score and added to a new column named Risk_category. The detailed classification is shown below:
    - Very High Risk: Score >= 1250
    - High Risk: 1250 < Score >= 1100
    - Medium to High Risk: 1100 < Score >= 950
    - Medium Risk: 950 < Score >= 850
    - Low to Medium Risk: 850 < Score >= 700
    - Low Risk: Score < 700

After the preprocessing now the data is ready for the regression model for predicting the Financial Risk Tolerance Score and Risk Category in which that individual falls in.

For adding into the model, I converted the required column data into categorical using a special mapper function.

For regression I have experimented three different models and monitored the errors and accuracy of prediction for each. The three models used are namely, Decision Tree, Random Forest and Gradient Boosting Machine.

1. **Decision Tree**

    For performing decision tree I have used sklearn.tree’s DecisionTreeRegressor and evaluated the model on three basis, Root Mean Square Error (RMSE), R2 Score and Accuracy.
    - Columns used for prediction: 'Gender', 'Car_Owner', 'Property_Owner', 'Marital_status', 'Housing_type', 'Age', 'Employed', 'Cost_of_living', 'Occupation_category', 'Education_category'.
    - Target Column: Financial Risk Tolerance Score.
    - Train test split: 80:20 with random_state = 25.
    - DecisionTreeRegressor: random_state = 25.
2. **Random Forest**

    For performing random forest regression I have used sklearn.ensemble’s RandomForestRegressor and evaluated the model on three basis, Root Mean Square Error (RMSE), R2 Score and Accuracy.

- Columns used for prediction: 'Gender', 'Car_Owner', 'Property_Owner', 'Marital_status', 'Housing_type', 'Age', 'Employed', 'Cost_of_living', 'Occupation_category', 'Education_category'.
- Target Column: Financial Risk Tolerance Score.
- Train test split: 80:20 with random_state = 25.
- DecisionTreeRegressor: n_estimators = 250, random_state = 25.

3. **Gradient Boosting Machine (GBM)**

   For performing GBM regression I have used sklearn.ensemble's GradientBoostingRegressor and evaluated the model on three basis, Root Mean Square Error (RMSE), R2 Score and Accuracy.

   - Columns used for prediction: 'Gender', 'Car_Owner', 'Property_Owner', 'Marital_status', 'Housing_type', 'Age', 'Employed', 'Cost_of_living', 'Occupation_category', 'Education_category'.
   - Target Column: Financial Risk Tolerance Score.
   - Train test split: 80:20 with random_state = 25.
   - DecisionTreeRegressor: n_estimators = 100, learning_rate = 0.1, random_state = 25.

For evaluating the models above, I used 3 different parameters namely, Root Mean Square Error (RMSE), R2 Score and Accuracy as follows:

- RMSE: square root of mean_sqaured_error function from sklearn.metrics

  RMSE = sqrt(∑ [predicted - actual]^2/n)

- R2: r2_score function from sklearn.metrics.

  R2 = 1 – [∑(actual – predicted) / ∑(actual – mean of actual)]

- Accuracy: For calculating accuracy I got the risk category for the predicted and actual financial risk score and then using accuracy function from sklearn.metrics calculated the accuracy on the predicted and actual risk category.

  Accuracy = (True Positive + True Negative) / (True Positive + True Negative + False Positive + False Negative)

The table below shows the comparison between Decision Tree, Random Forest and Gradient Boosting Machine Regressors based on the evaluation model mentioned above:

Regressor	RMSE	R2	Accuracy
Decision Tree	39.526	0.941	0.83
Random Forest	27.535	0.971	0.846
Gradient Boosting Machine	13.131	0.993	0.934

We can conclude from this that Gradient Boosting Machine provides the best scores across evaluation parameters then the other two regressor. The table above is also available in this google sheet.

Equity Forecasting model

Once we have identified the financial risk category of an individual, the next step is to suggest them well performing equities based on the risk category they fall into. This module helps in forecasting the future trend and prices of equities based on it's previous 10 years movement.

For equity forecasting, I experimented around 8 different algorithms and combination of algorithms on a random equity data and monitored it's results on the last 21 days of data and plotted the same. The algorithms experimented are:

1. LSTM
2. SVM
3. ARIMA
4. SARIMA
5. SARIMA + LGMB
6. SARIMA + XGBoost
7. SARIMA + Random Forest
8. SARIMA + GARCH

LSTM – Long Short-Term Memory

For performing LSTM, I have used keras Sequential, LSTM and Dense package and evaluated the model on 5 parameters Root Mean Square Error (RMSE), Mean Forecasting Error (MFE), Cumulative Forecasting Error (CFE), Tracking Signal and Coverage Probability. The results of the evaluating parameters are mentioned in the table at the end of this section.

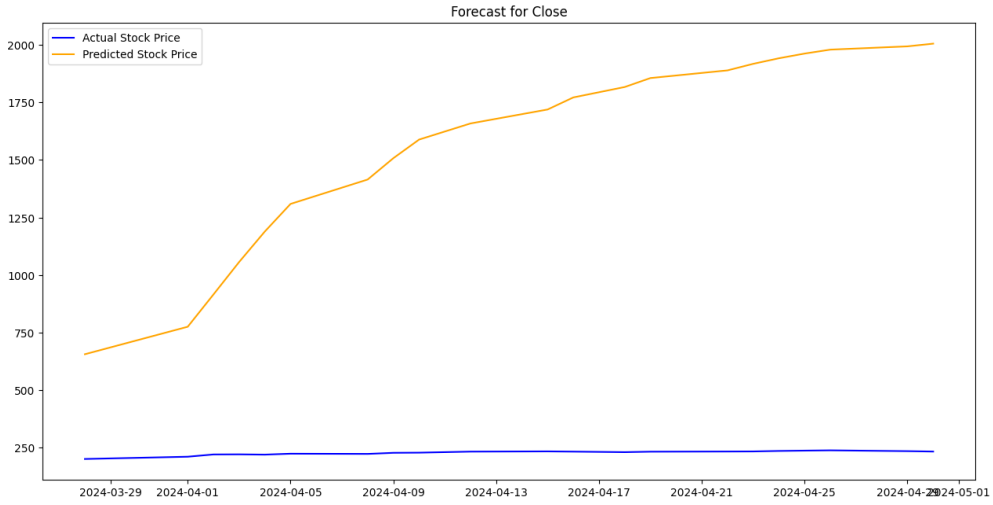
Preprocessing:

- Setting 'Date' column as index column
- Splitting the last 21 days for plotting from the main dataframe.
- Getting necessary features for prediction: ['Open', 'High', 'Low', 'Close', 'Volume', 'No. of Trades']
- Normalising the data using MinMaxScaler from sklearn.preprocessing with feature_range = (0,1)
- Then I created sequences with time_step = 2000, as the data is a time series and split the Close column as a target variable.

Model Parameters:

- Train_test_split = 80:20, shuffle = False
- LSTM 1st Layer: units = 50, return_sequences = True
- LSTM 2nd Layer: units = 50, return_sequences = False
- Dense 1st Layer: units = 25
- Dense 2nd Layer: units = 1 (Output)
- Model Compiler: optimizer = adam, loss = mean_sqaured_error
- Model training: batch_size = 1, epoch = 10
- After training the model, I created a predict_future function, which actually takes in the time_step which is 2000 and based on the time step predicts the next date close price using the lstm model and adds the prediction to the data to use it to predict for the next day.

Results:



From the graph we can conclude the predictions are way far from the actual values, but it gives a trend analysis due to the volatility of the predicted outputs. Still LSTM is not a good model for our data predictions.

SVM – Support Vector Machine

For performing SVM, I have used SVR package from sklearn.svm and evaluated the model on 5 parameters Root Mean Square Error (RMSE), Mean Forecasting Error (MFE), Cumulative Forecasting Error (CFE), Tracking Signal and Coverage Probability. The results of the evaluating parameters are mentioned in the table at the end of this section.

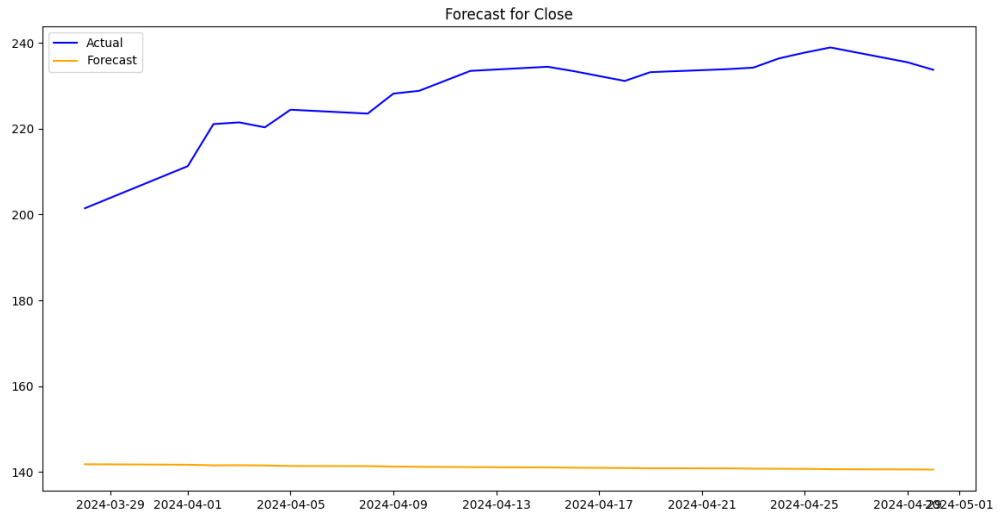
Preprocessing:

- Setting ‘Date’ column as index column
- Getting necessary features for prediction: ['Open', 'High', 'Low', 'Close', 'WAP', 'Volume', 'No. of Trades', 'Total Turnover (Rs.)', 'Deliverable Quantity', '% Deli. Qty to Traded Qty', 'Spread High-Low', 'Spread Close-Open']
- Normalising the data using StandardScaler from sklearn.preprocessing.
- Then I created sequences with time_step = 2000, as the data is a time series and also split the Close column as a target variable.

Model Parameters:

- Train_test_split = I physically split the previous month’s data as testing and rest as training.
- Flattened the input for model training.
- SVR model: kernel = Radial Basis Function (‘rbf’)
- After model training, I predicted the output on the flattened input file, and then evaluated the results and plotted them using matplotlib.pyplot.

Results:



From the graph we can conclude the predictions are better then LSTM, but also the predictions are flat and can be more optimized for better results and volatility.

ARIMA – AutoRegressive Integrated Moving Average

For performing ARIMA, I have used ARIMA package from statsmodels.tsa.arima.model and evaluated the model on 5 parameters Root Mean Square Error (RMSE), Mean Forecasting

Error (MFE), Cumulative Forecasting Error (CFE), Tracking Signal and Coverage Probability. The results of the evaluating parameters are mentioned in the table at the end of this section.

Preprocessing:

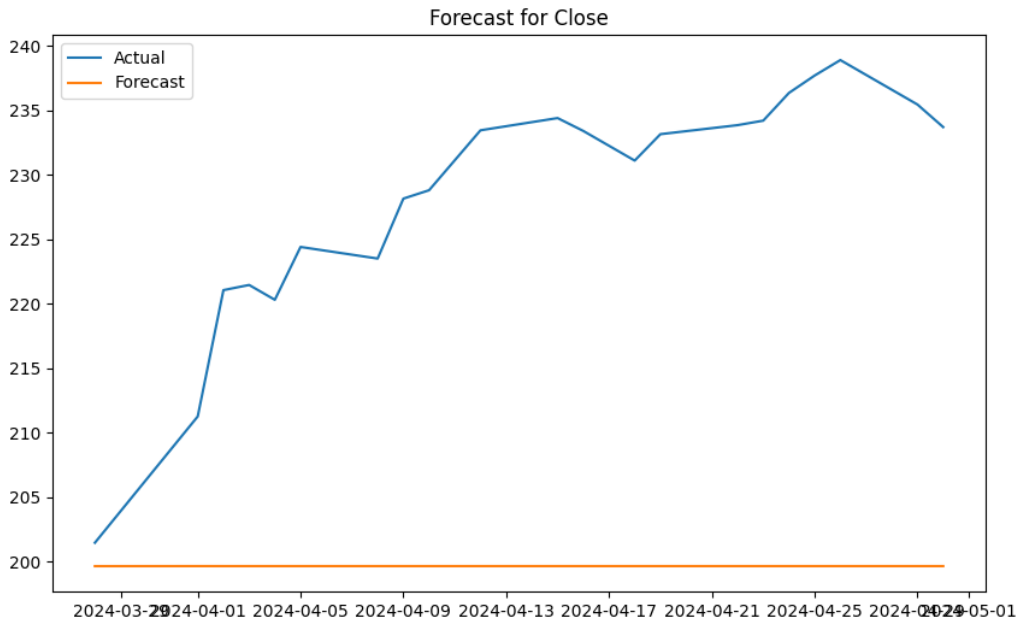
- Setting ‘Date’ column as index column.
- Physically splitting the data into train and test set where test set consists of the last months trading days (21 to be precise) and rest dataframe for training the model.
- Getting features for prediction: 'Close' (For models like ARIMA you can predict all the features, but for evaluating I have selected only Close price of the equity).
- After selecting the features, the next step is to check stationarity of the features, using Augmented Dickey-Fuller (ADF) test. The ADF tests the null hypothesis that a unit root is present in a time series sample. Rejection of the null hypothesis suggests stationarity.
- Then I performed differencing to remove stationarity from the data for plotting of ACF and PACF graphs.

Model Parameters:

- First, I plotted the AutoCorrelation Function (ACF) and Partial AutoCorrelation Function (PACF), to determine the p, d and q parameters for the model where,
  - p: Autoregressive Order
  - d: Differencing Order
  - q: Moving Average
- Using the p = 1, q = 1 and d = 1 values for ARIMA model and trained on the train dataset.
- For forecasting the future close prices, I added the forecast_period as 21 (assuming trading days in a month), and forecasted using model.forecast function with forecast_period as a parameter.
- Evaluated the forecasted values and plotted to see it on the graph.

Results:

From the graph below we can say the results are like SVM but are closer to the actual values. Overall, this model is better than SVM but in terms of volatility it’s linear.



SARIMA – Seasonal AutoRegressive Integrated Moving Average

SARIMA extends ARIMA by adding a seasonal component into the picture. For performing SARIMA, I have used SARIMA package from statsmodels.tsa.statespace.sarimax and evaluated the model on 5 parameters Root Mean Square Error (RMSE), Mean Forecasting Error (MFE), Cumulative Forecasting Error (CFE), Tracking Signal and Coverage Probability. The results of the evaluating parameters are mentioned in the table at the end of this section.

Preprocessing:

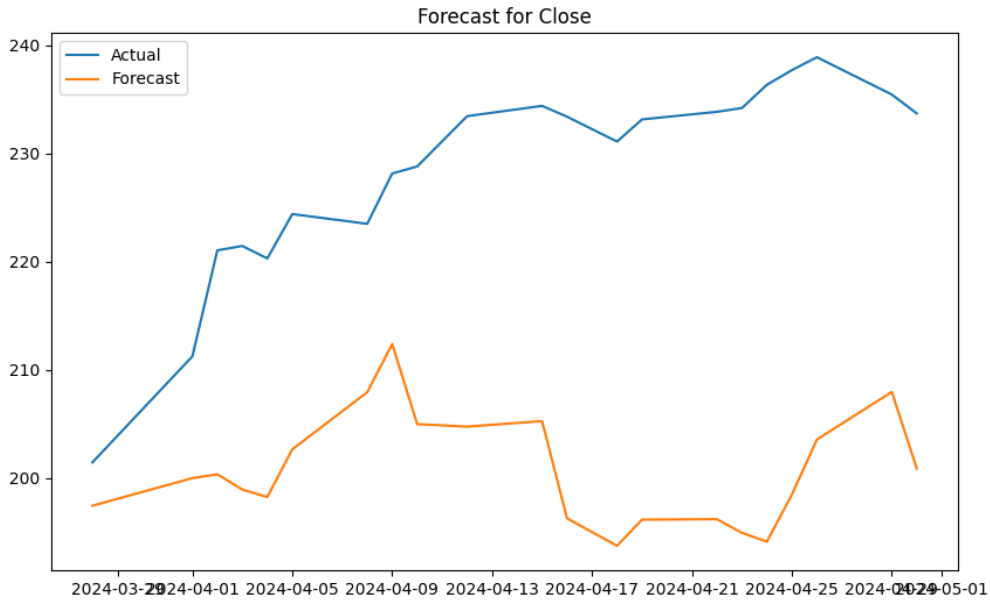
- Setting ‘Date’ column as index column.
- Physically splitting the data into train and test set where test set consists of the last months trading days (21 to be precise) and rest dataframe for training the model.
- Getting features for prediction: 'Close' (For models like SARIMA you can predict all the features, but for evaluating I have selected only Close price of the equity).
- After selecting the features, the next step is to check stationarity of the features, using Augmented Dickey-Fuller (ADF) test. The ADF tests the null hypothesis that a unit root is present in a time series sample. Rejection of the null hypothesis suggests stationarity.
- Then I performed differencing to remove stationarity from the data fro plotting ACF and PACF graphs.

Model Parameters:

- First, I plotted the AutoCorrelation Function (ACF) and Partial AutoCorrelation Function (PACF), to determine the p, d, q, P, Q, D and s parameters for the model.
  - p: Autoregressive Order
  - d: Differencing Order
  - q: Moving Average
  - P: Seasonal Autoregressive Order
  - D: Seasonal Differencing Order
  - Q: Seasonal Moving Average
  - s: Seasonal period
- Using the p = 1, q = 1, d = 1, P = 1, D = 1, Q = 1 and s = 12 (for monthly seasonality) for the SARIMA model and trained on the train dataset.
- For forecasting the future close prices, I added the forecast_period as 21 (assuming trading days in a month), and forecasted using model.forecast function with forecast_period as a parameter.
- Converted the forecast values into forecast series for practical use and interpretation.
- Evaluated the forecast series values and plotted to see it on the graph.

Results:

From the graph below, we can say the results are better than ARIMA and provide a good volatility but from the two lines you can see it's closer to the actual values but is not following the original trend. Overall, this model is better than ARIMA in terms of volatility.



SARIMA + LGBM – Seasonal AutoRegressive Integrated Moving Average + Light Gradient Boosting Machine.

In this combination of models, I combined SARIMA and LGBM to get better results in terms of trend setting and volatility and trying to get more closer to the expected value. In this combination first we train the SARIMA model on the test dataset and then the residuals we get from that are used as input data for LGBM and the results from both the models are combined to get a single forecast for the Close Prices. Later I evaluated the model on 5 parameters Root Mean Square Error (RMSE), Mean Forecasting Error (MFE), Cumulative Forecasting Error (CFE), Tracking Signal and Coverage Probability. The results of the evaluating parameters are mentioned in the table at the end of this section.

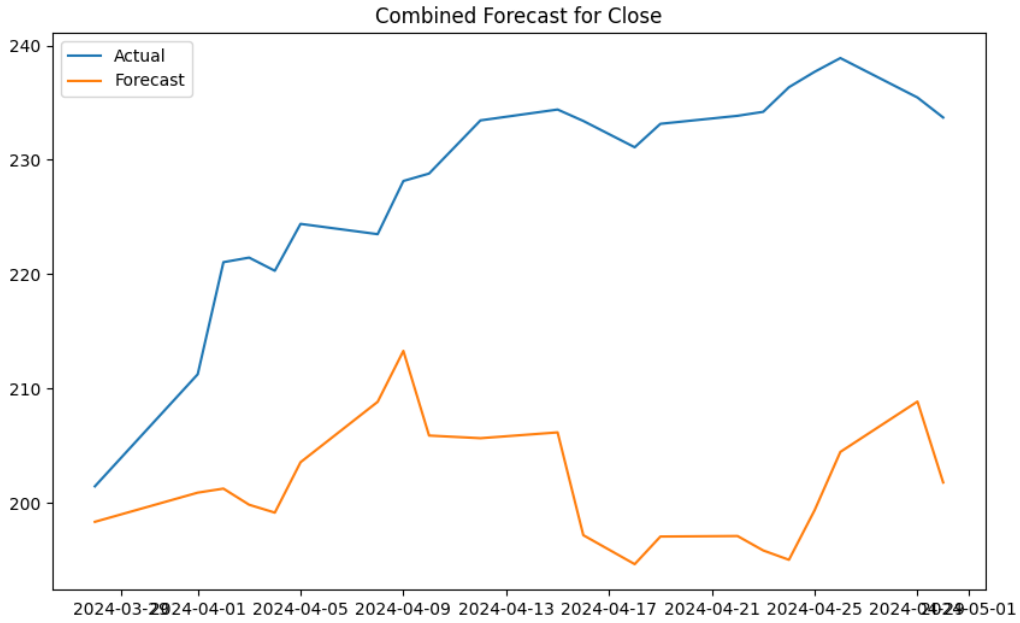
Preprocessing:

- Setting ‘Date’ column as index column.
- Physically splitting the data into train and test set where test set consists of the last months trading days (21 to be precise) and rest dataframe for training the model.
- Getting features for prediction: 'Close' (For models like SARIMA you can predict all the features, but for evaluating I have selected only Close price of the equity).

Model Parameters:

- Using the p = 1, q = 1, d = 1, P = 1, D = 1, Q = 1 and s = 12 (for monthly seasonality) for the SARIMA model and trained on the train dataset.
- After training the SARIMA model I got the residuals from it using resid function and stored them into different array.
- Then the array containing the residuals, was reshaped in order to fit the LGBM, and also separated the Close prices columns.
- Later I split them into train and test set, 80:20 with random_state = 42.
- LGBM was then trained using the trained data with n_estimators = 100 and random_state = 42.
- For forecasting the future close prices, I added the forecast_period as 21 (assuming trading days in a month), and forecasted using model.forecast function with forecast_period as a parameter and using the predict function predicted on the LGBM to get the outputs.
- Converted the forecast values into forecast series for practical use and interpretation.
- Evaluated the combined forecast values by adding forecast values from both models and plotted to see it on the graph.

Results:



From the graph above we can say the results are similar to SARIMA and also provide a good volatility but from the two lines you can see it's closer to the actual values but is not following the original trend. Overall, this model is better than ARIMA but gives us similar results in terms of volatility and trend setting.

SARIMA + XGBoost – Seasonal AutoRegressive Integrated Moving Average + eXtreme Gradient Boosting.

In this combination of models, I combined SARIMA and XGBoost to get better results in terms of regression from the residuals of SARIMA and try to get more closer to the expected value. Later I evaluated the model on 5 parameters Root Mean Square Error (RMSE), Mean Forecasting Error (MFE), Cumulative Forecasting Error (CFE), Tracking Signal and Coverage Probability. The results of the evaluating parameters are mentioned in the table at the end of this section.

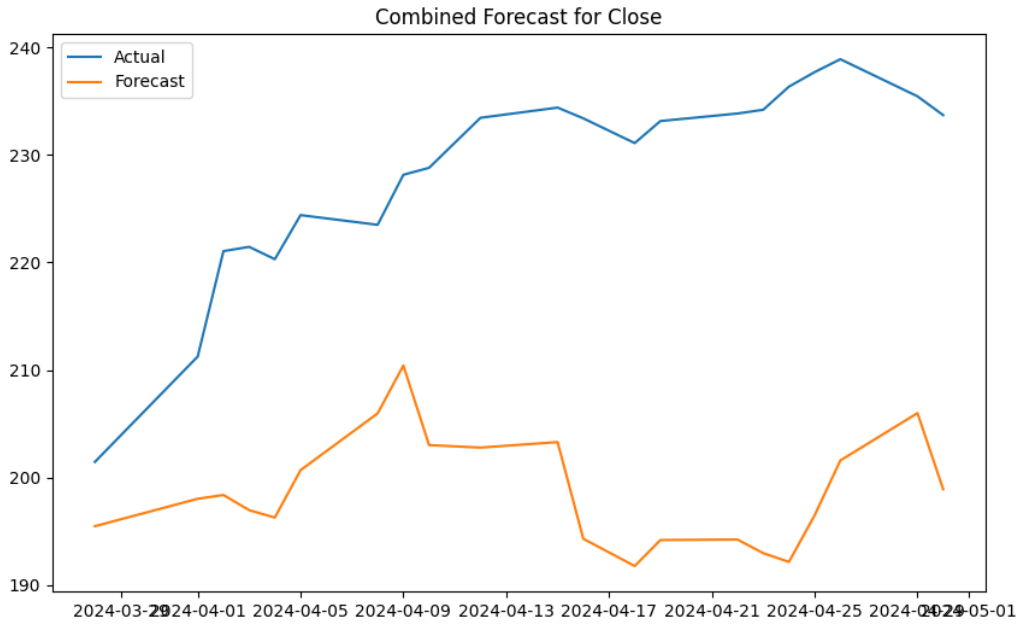
Preprocessing:

- Setting 'Date' column as index column.
- Physically splitting the data into train and test set where test set consists of the last months trading days (21 to be precise) and rest dataframe for training the model.
- Getting features for prediction: 'Close' (For models like SARIMA you can predict all the features, but for evaluating I have selected only Close price of the equity).

Model Parameters:

- Using the p = 1, q = 1, d = 1, P = 1, D = 1, Q = 1 and s = 12 (for monthly seasonality) for the SARIMA model and trained on the train dataset.
- After training the SARIMA model I got the residuals from it using resid function and stored them into different array.
- Then the array containing the residuals, was reshaped to fit the XGBoost, and also separated the Close prices columns.
- Later I split them into train and test set, 80:20 with random_state = 42.
- XGBoost was then trained using the trained data with n_estimators = 100 and random_state = 42.
- For forecasting the future close prices, I added the forecast_period as 21 (assuming trading days in a month), and forecasted using model.forecast function with forecast_period as a parameter and using the predict function predicted on the XGBoost to get the outputs.
- Converted the forecast values into forecast series for practical use and interpretation.
- Evaluated the combined forecast values by adding forecast values from both models and plotted to see it on the graph.

Results:



From the graph above we can say that there is not much difference between the outputs of SARIMA + LGBM and SARIMA + XGBoost, but if we see the evaluation parameters the LGBM combination beats XGBoost combination by minute margins.

SARIMA + Random Forest – Seasonal AutoRegressive Integrated Moving Average + Random Forest Regressor.

In this combination of models, I combined SARIMA and Random Forest to get better results in terms complexity of data and regression from the residuals of SARIMA and try to get more closer to the expected value. Later I evaluated the model on 5 parameters Root Mean Square Error (RMSE), Mean Forecasting Error (MFE), Cumulative Forecasting Error (CFE), Tracking Signal and Coverage Probability. The results of the evaluating parameters are mentioned in the table at the end of this section.

Preprocessing:

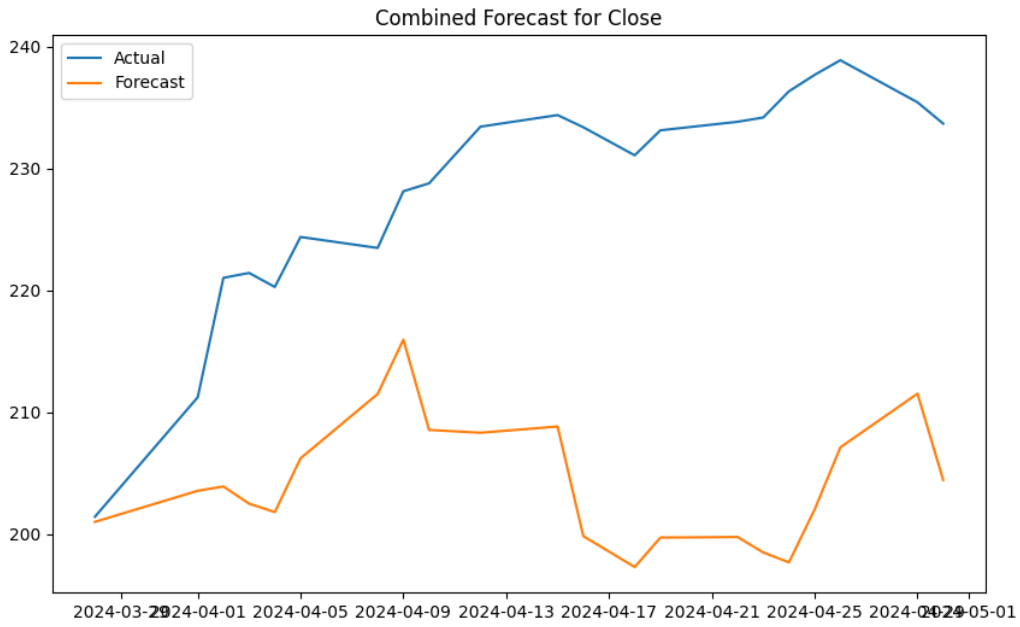
- Setting ‘Date’ column as index column.
- Physically splitting the data into train and test set where test set consists of the last months trading days (21 to be precise) and rest dataframe for training the model.
- Getting features for prediction: 'Close' (For models like SARIMA you can predict all the features, but for evaluating I have selected only Close price of the equity).

Model Parameters:

- Using the p = 1, q = 1, d = 1, P = 1, D = 1, Q = 1 and s = 12 (for monthly seasonality) for the SARIMA model and trained on the train dataset.
- After training the SARIMA model I got the residuals from it using resid function and stored them into different array.
- Then the array containing the residuals, was reshaped to fit the Random Forest Regressor, and also separated the Close prices columns.
- Later I split them into train and test set, 80:20 with random_state = 42.
- Random Forest Regressor was then trained using the trained data with n_estimators = 100 and random_state = 42.
- For forecasting the future close prices, I added the forecast_period as 21 (assuming trading days in a month), and forecasted using model.forecast function with forecast_period as a parameter and using the predict function predicted on the Random Forest Regressor to get the outputs.
- Converted the forecast values into forecast series for practical use and interpretation.
- Evaluated the combined forecast values by adding forecast values from both models and plotted to see it on the graph.

Results:

From the graph below we can say that the close prices are closer to the actual prices though the issue here remains the trend which the original close price line follows is still not as satisfying as it should be, but evaluation parameter wise it performs much better then XGBoost and LGBM combinations with SARIMA.



SARIMA + GARCH – Seasonal AutoRegressive Integrated Moving Average + Generalized AutoRegressive Conditional Heteroskedasticity

In this combination of models, I combined SARIMA and GARCH, where SARIMA is used to predict the forecast for the time series and GARCH will be used to understand the volatility of the time series in order to get closer to the actual values with maintaining a trend of the equity. For implementing GARCH I used arch_model package from acrh. Later I evaluated the model on 5 parameters Root Mean Square Error (RMSE), Mean Forecasting Error (MFE), Cumulative Forecasting Error (CFE), Tracking Signal and Coverage Probability. The results of the evaluating parameters are mentioned in the table at the end of this section.

Preprocessing:

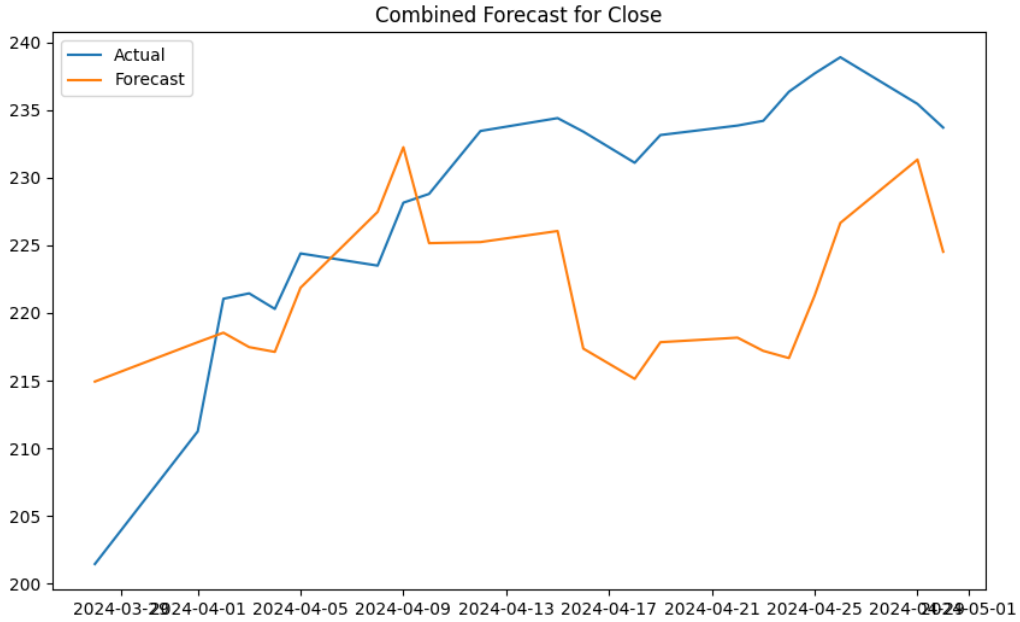
- Setting ‘Date’ column as index column.
- Physically splitting the data into train and test set where test set consists of the last months trading days (21 to be precise) and rest dataframe for training the model.

- Getting features for prediction: 'Close' (For models like SARIMA you can predict all the features, but for evaluating I have selected only Close price of the equity).

Model Parameters:

- Using the p = 1, q = 1, d = 1, P = 1, D = 1, Q = 1 and s = 12 (for monthly seasonality) for the SARIMA model and trained on the train dataset.
- After training the SARIMA model I got the residuals from it using resid function and stored them into different array.
- Then the array containing the residuals was trained using arch_model with vol = 'GARCH', p = 1 and q = 1.
- For forecasting the future close prices, I added the forecast_period as 21 (assuming trading days in a month), and forecasted using model.forecast function with forecast_period as a parameter and using the same model.forecast function forecasted on the GARCH to get the volatility of the close prices.
- Converted the forecast values into forecast series for practical use and interpretation.
- Evaluated the combined forecast values by adding forecast values from both models and plotted to see it on the graph.

Results:



From the graph above we can see that the predicted close prices are now very close to the actual values and is able to match the trend of the actual close prices. Overall this combination was able to deliver a lot better score in evaluation parameters.

Model Evaluation:

For evaluating all the model above, I chose 5 different evaluation parameter which gives us an overall idea of the model performance over multiple factors. The evaluation parameters used were:

1. Root Mean Squared Error (RMSE): It gives a good idea about model's accuracy in predicting.

   RMSE = sqrt(∑ [predicted - actual]^2/n)

2. Mean Forecasting Error (MFE): It measures the average error in the forecasts.

   MFE = 1/n ∑ [actual – predicted]

3. Cumulative Forecast Error (CFE): Measures the total sum of the errors encountered during forecasts.

   CFE = ∑ [actual – predicted]

4. Tracking Signal: Tracking signal helps to detect bias in the forecast.

   Tracking Signal = CFE / MAD (Mean Absolute Deviation)

   MAD = 1/n abs(∑ [actual – predicted])

5. Coverage Probability: It determines whether the forecast is within the High and Low for that day.

   Coverage Probability = 1/n ∑(predicted >= Low & predicted <= High)

The table below shows the results of the evaluation parameters for all the 8 models experimented for forecasting equity prices.

Model	RMSE	MFE	CFE	Tracking Signal	Coverage Probability
LSTM	1711.06	-1676.14	-179346.71	-107	0
SVM	87.76	87.25	1823.34	21	0
ARIMA	30.15	28.74	603.634	21	0.04
SARIMA	29.49	27.65	580.78	21	0
SARIMA + Light Gradient Boosting Machine	28.65	26.75	561.79	21	0
SARIMA + XGBoost	31.35	29.62	622.21	21	0
SARIMA + Random Forest	26.17	24.06	505.43	21	0.04
SARIMA + GARCH	11.2	6.94	145.84	15.15	0.14

From the table below I can conclude that combination of SARIMA and GARCH provides the best results and match the overall trend of equity. The table above is also present in this google sheet.