Project 2: Crypto Golden Cross ML Algo Trading

Samirah Djachechi, Benjamin Boule, Eugene Njinkeu,
Ian Melhorn, Charles Panagopoulos, Alex Toenshoff, Will Pape

Overview

Trained support vector machine (SVM) model to make trading predictions, backtested the results, and evaluated its performance compared to that of a logistic regression model for BTC and ETH.

- **Technical indicators:**
  - FinTA 50 SMA x 200 SMA = Golden Cross
- **ML comparative analysis:**
  - Support Vector Machine model
  - Logistic Regression model
- **AWS chatbot:**
  - Front-end user interface for customers to chat in and purchase one of our trading models

Imports & Libraries

```
# Import the finta Python library and the TA module
from finta import TA
```

```
# Create a signals_df DataFrame
signals_df = trading_df.copy()

# Set the short window and long windows
short_window = 50
long_window = 200

# Add the SMA technical indicators for the short and long windows
signals_df["sma_fast"] = TA.SMA(signals_df, short_window)
signals_df["sma_slow"] = TA.SMA(signals_df, long_window)
```

- We used the Santiment (Sanpy) to retrieve the cryptocurrency data.
- We used the FinTA (Financial Technical Analysis) library to populate the golden cross indicator.
- We used Pandas to clean and format our datasets and Numpy and hvplot for financial analysis and visualizations.
- We imported StandardScaler, SVM, and LogisticRegression from SKlearn to standardize the data and populate the machine learning classifier models.

Bitcoin Data Analysis

BTC Golden Cross Model:

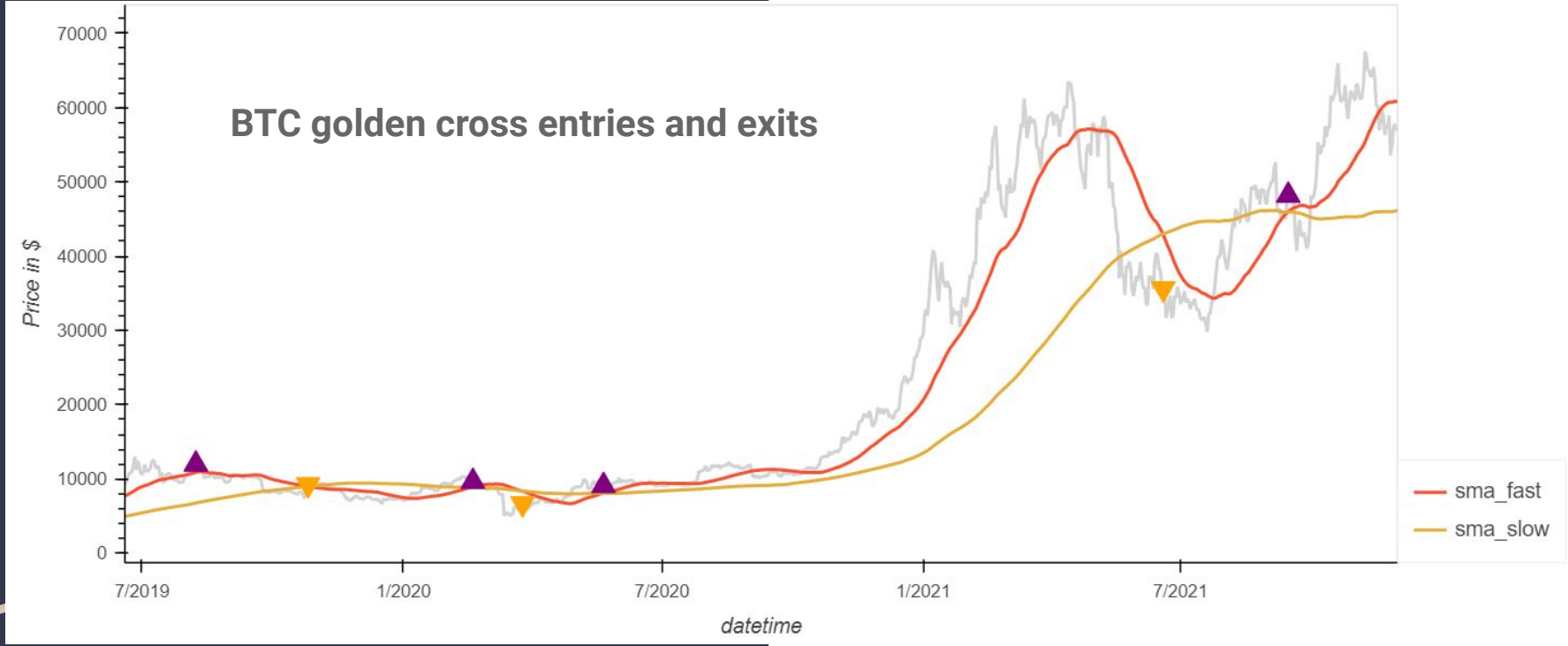
- Santiment BTC data time frame: 12/01/2018 - 12/01/2021 (3 years)
- FinTA 50 SMA x 200 SMA = Golden Cross
- 4 entries, 3 exits
- Split data into training and test datasets
- Standardized data using StandardScaler
- **Training** data **Support Vector Machine** model accuracy score = 60%
- **Training** data **Logistic Regression** model accuracy score = 54
- **Testing** data **Support Vector Machine** model accuracy score = 53%
    - Trading algorithm returns **outperforms** actual returns
- **Testing** data **Logistic Regression** model accuracy score = 48%

Ethereum Data Analysis

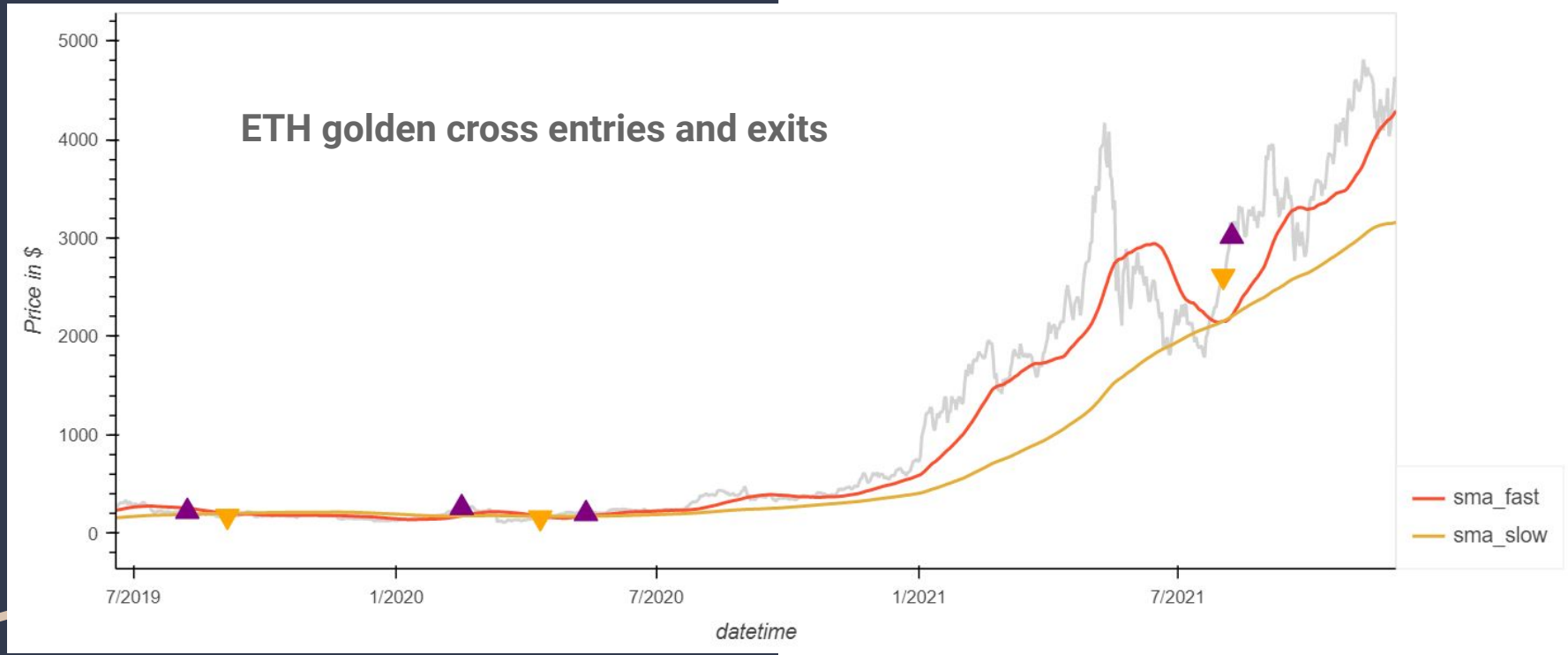
ETH Golden Cross Model:

- Santiment ETH data time frame: 12/01/2018 - 12/01/2021 (3 years)
- FinTA 50 SMA x 200 SMA = Golden Cross
- 4 entries, 3 exits
- Split data into training and test datasets
- Standardized data using StandardScaler
- **Training** data **Support Vector Machine** model accuracy score = 58%
- **Training** data **Logistic Regression** model accuracy score = 55%
- **Testing** data **Support Vector Machine** model accuracy score = 54%
    - Trading algorithm returns **underperforms** actual returns
- **Testing** data **Logistic Regression** model accuracy score = 45%

Data Exploration & Results Examples



Data Exploration & Results Examples



Data Exploration & Results Examples

Creating the Training Datasets

[16]:
```
# Imports
from pandas.tseries.offsets import DateOffset
```

[17]:
```
# Select the start of the training period
training_begin = X.index.min()

# Display the training begin date
print(training_begin)
```

```
2019-06-19 00:00:00+00:00
```

[18]:
```
# Select the ending period for the training data with an offset of 3 months
training_end = X.index.min() + DateOffset(months=3)

# Display the training end date
print(training_end)
```

```
2019-09-19 00:00:00+00:00
```

[19]:
```
# Generate the X_train and y_train DataFrames
X_train = X.loc[training_begin:training_end]
y_train = y.loc[training_begin:training_end]

# Display sample data
X_train.head()
```

[19]:

datetime	sma_fast	sma_slow
2019-06-19 00:00:00+00:00	7713.055810	4969.832575
2019-06-20 00:00:00+00:00	7795.545075	4996.768986
2019-06-21 00:00:00+00:00	7888.325162	5028.021115
2019-06-22 00:00:00+00:00	7986.993194	5061.745101
2019-06-23 00:00:00+00:00	8087.477265	5097.251982

BTC training datasets with an offset of 3 months

Data Exploration & Results Examples

Creating the Training Datasets

[14]:
```
# Imports
from pandas.tseries.offsets import DateOffset
```

[15]:
```
# Select the start of the training period
training_begin = X.index.min()

# Display the training begin date
print(training_begin)
```

```
2019-06-19 00:00:00+00:00
```

[16]:
```
# Select the ending period for the training data with an offset of 3 months
training_end = X.index.min() + DateOffset(months=3)

# Display the training end date
print(training_end)
```

```
2019-09-19 00:00:00+00:00
```

[17]:
```
# Generate the X_train and y_train DataFrames
X_train = X.loc[training_begin:training_end]
y_train = y.loc[training_begin:training_end]

# Display sample data
X_train.head()
```

[17]:

datetime	sma_fast	sma_slow
2019-06-19 00:00:00+00:00	234.051936	158.616174
2019-06-20 00:00:00+00:00	236.269469	159.392705
2019-06-21 00:00:00+00:00	238.926172	160.322631
2019-06-22 00:00:00+00:00	241.754730	161.318462
2019-06-23 00:00:00+00:00	244.630754	162.345223

ETH training datasets with an offset of 3 months

Data Exploration & Results Examples

Creating the Testing Datasets

[20]:
```
# Generate the X_test and y_test DataFrames
X_test = X.loc[training_end:]
y_test = y.loc[training_end:]

# Display sample data
X_test.head()
```

[20]:

datetime	sma_fast	sma_slow
2019-09-19 00:00:00+00:00	10524.851472	8165.232394
2019-09-20 00:00:00+00:00	10520.490909	8197.356318
2019-09-21 00:00:00+00:00	10510.499681	8227.961461
2019-09-22 00:00:00+00:00	10495.424404	8258.779991
2019-09-23 00:00:00+00:00	10470.607194	8287.869189

Standardizing the Data

[21]:
```
# Imports
from sklearn.preprocessing import StandardScaler
```

[22]:
```
# Create a StandardScaler instance
scaler = StandardScaler()

# Apply the scaler model to fit the X-train data
X_scaler = scaler.fit(X_train)

# Transform the X_train and X_test DataFrames using the X_scaler
X_train_scaled = X_scaler.transform(X_train)
X_test_scaled = X_scaler.transform(X_test)
```

BTC testing datasets and standardizing the data with StandardScaler

Data Exploration & Results Examples

Creating the Testing Datasets

[18]:
```
# Generate the X_test and y_test DataFrames
X_test = X.loc[training_end:]
y_test = y.loc[training_end:]

# Display sample data
X_test.head()
```

[18]:

datetime	sma_fast	sma_slow
2019-09-19 00:00:00+00:00	194.433018	206.540982
2019-09-20 00:00:00+00:00	194.437851	206.994934
2019-09-21 00:00:00+00:00	194.390264	207.383026
2019-09-22 00:00:00+00:00	194.174721	207.747646
2019-09-23 00:00:00+00:00	193.759753	208.067080

Standardizing the Data

[19]:
```
# Imports
from sklearn.preprocessing import StandardScaler
```

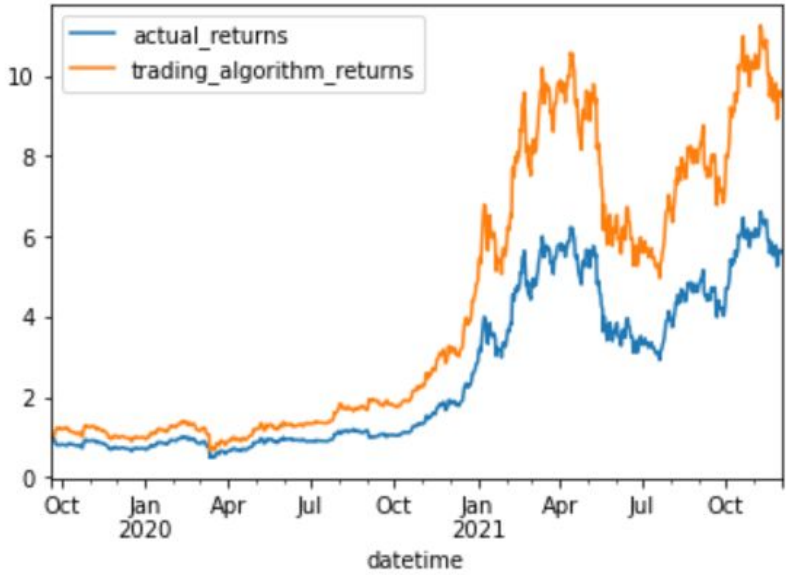
[20]:
```
# Create a StandardScaler instance
scaler = StandardScaler()

# Apply the scaler model to fit the X-train data
X_scaler = scaler.fit(X_train)

# Transform the X_train and X_test DataFrames using the X_scaler
X_train_scaled = X_scaler.transform(X_train)
X_test_scaled = X_scaler.transform(X_test)
```

ETH testing datasets and standardizing the data with StandardScaler

Data Exploration & Results Examples

```
[29]: # Calculate and plot the cumulative returns for the `actual_returns` and the `trading_algorithm_returns`
      (1 + predictions_df[["actual_returns", "trading_algorithm_returns"]]).cumprod().plot()
```

```
[29]: <AxesSubplot:xlabel='datetime'>
```

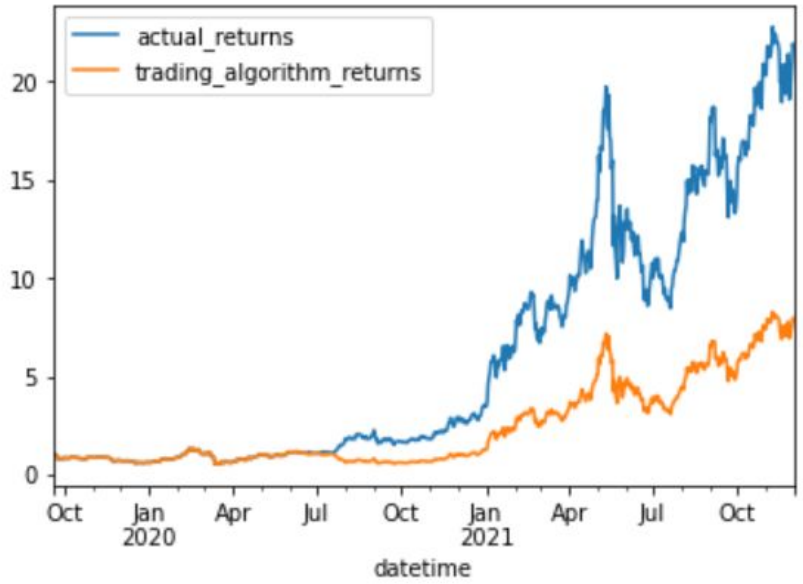


BTC SVM testing data model outperforms actual BTC returns

Data Exploration & Results Examples

```
[27]: # Calculate and plot the cumulative returns for the `actual_returns` and the `trading_algorithm_returns`
      (1 + predictions_df[["actual_returns", "trading_algorithm_returns"]]).cumprod().plot()
```

```
[27]: <AxesSubplot:xlabel='datetime'>
```



ETH SVM testing data model underperforms actual ETH returns

Data Exploration & Results Examples

[36]:
```
# Print the classification report for the Logistic Regression model using the test data
print("Logistic Regression Classification Report")
print(lr_testing_report)
```

```
Logistic Regression Classification Report
              precision    recall  f1-score   support

        -1.0       0.48      0.91      0.62       381
         1.0       0.55      0.10      0.17       424

    accuracy                           0.48       805
   macro avg       0.51      0.50      0.40       805
weighted avg       0.51      0.48      0.38       805
```

[37]:
```
# Print the classification report for the SVM model using the test data
print("SVM Classification Report")
print(svm_testing_report)
```

```
SVM Classification Report
              precision    recall  f1-score   support

        -1.0       0.53      0.14      0.22       381
         1.0       0.53      0.89      0.67       424

    accuracy                           0.53       805
   macro avg       0.53      0.51      0.44       805
weighted avg       0.53      0.53      0.45       805
```

The BTC SVM testing data model performs a bit better than the logistic regression model since it has a higher accuracy score (53% to 48%)

Data Exploration & Results Examples

[35]:
```
# Print the classification report for the Logistic Regression model using the test data
print("Logistic Regression Classification Report")
print(lr_testing_report)
```

```
Logistic Regression Classification Report
              precision    recall  f1-score   support

        -1.0       0.43      0.58      0.49       367
         1.0       0.49      0.34      0.40       438

    accuracy                           0.45       805
   macro avg       0.46      0.46      0.45       805
weighted avg       0.46      0.45      0.44       805
```

[36]:
```
# Print the classification report for the SVM model using the test data
print("SVM Classification Report")
print(svm_testing_report)
```

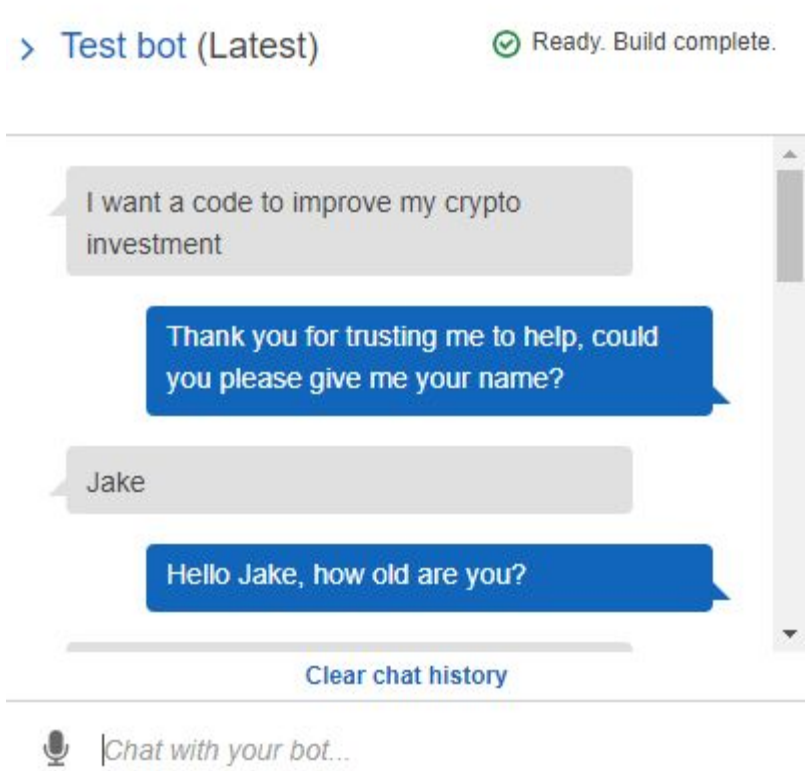
```
SVM Classification Report
              precision    recall  f1-score   support

        -1.0       0.47      0.05      0.10       367
         1.0       0.54      0.95      0.69       438

    accuracy                           0.54       805
   macro avg       0.50      0.50      0.39       805
weighted avg       0.51      0.54      0.42       805
```

The ETH SVM testing data model performs a bit better than the logistic regression model since it has a higher accuracy score (54% to 45%)

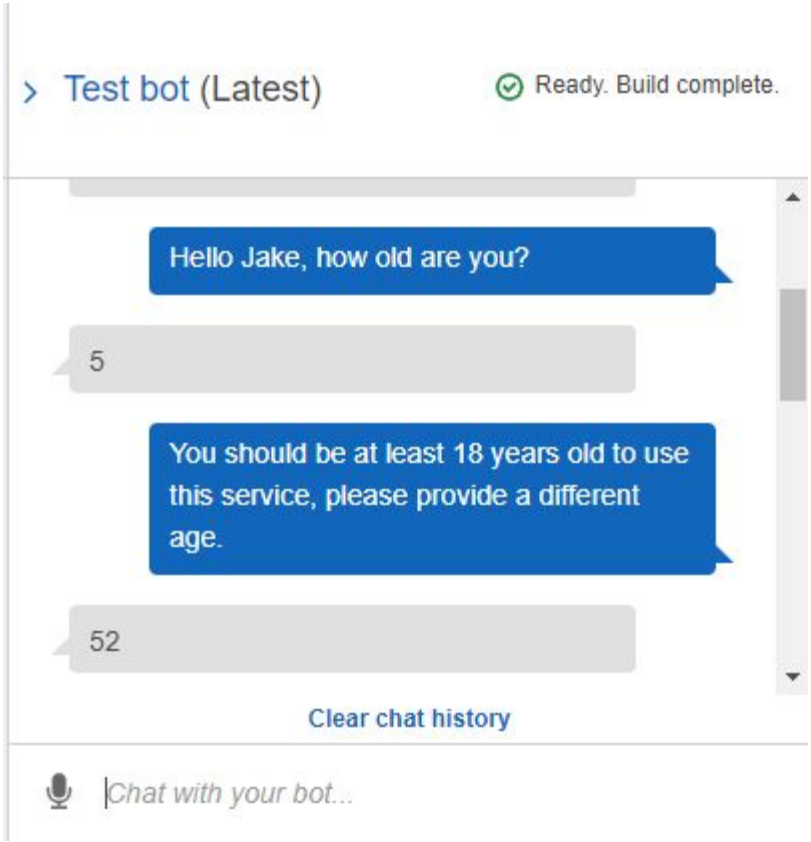
AWS Trading Bot

- Based on the technical indicators, the group decided to incorporate a trading bot.
- The investor initially answers basic questions.
- Based on the answers provided the bot makes an investment recommendation.

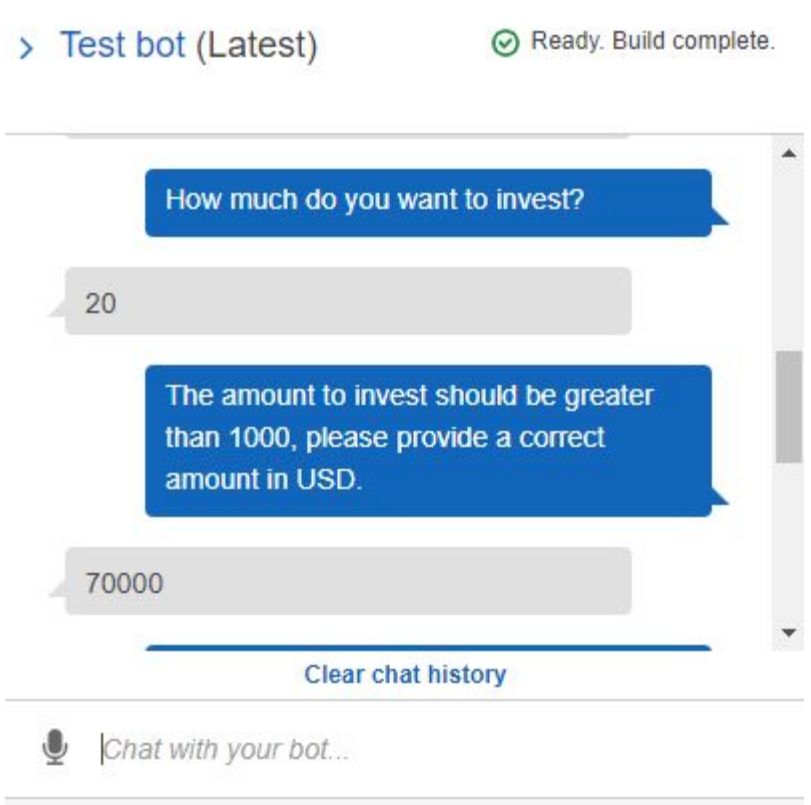
User Engagement



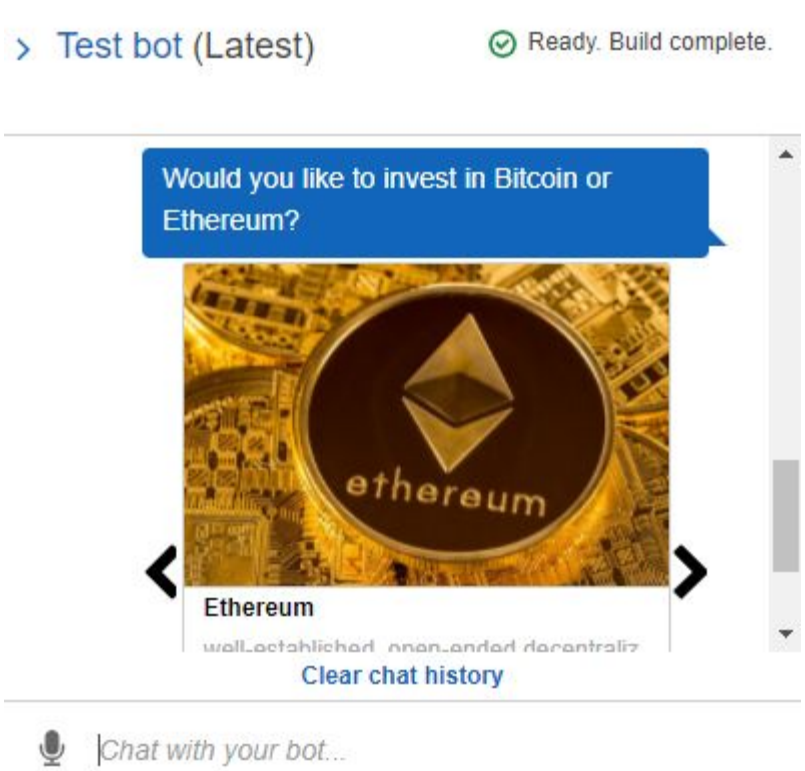
User Parameters



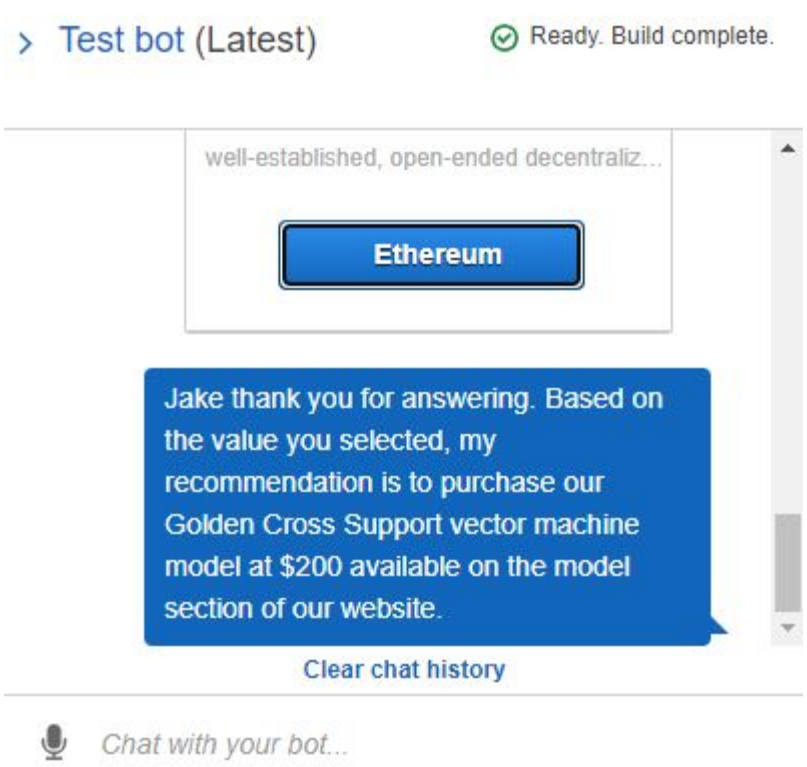
Initial Investment



Trading Bot Recommendation



Trading Bot


Test bot (Latest)
Ready. Build complete.
well-established, open-ended decentraliz...
Ethereum
Jake thank you for answering. Based on the value you selected, my recommendation is to purchase our Golden Cross Support vector machine model at $200 available on the model section of our website.
Clear chat history
Chat with your bot...

Postmortem

- The primary issue we faced was troubleshooting the trading bot.
- The training models created aren’t accurate and we struggled to adjust them for effective performance.

Questions?