2024 Crypto Final Report

Date: Sep 22nd, 2024

1. Business Background:

DeFiner is a DeFi (Decentralized Finance) platform that aims to address the limitations and problems with current DeFi lending protocols. The main issue with existing DeFi lending platforms is that they often serve as gatekeepers, deciding which tokens are eligible to be part of their lending pools. This selective approach leaves many tokens and digital assets unavailable for lending.In DeFiner 2.0's use cases, the goal is providing a permissionless, configurable, and private lending protocol that accommodates a wide range of tokens and assets.

DeFiner and "Fast Valuer" are more than platforms and products; they are catalysts for change, heralding a new era in decentralized finance.

In this transformative journey, "Fast Valuer" takes center stage as a groundbreaking solution that has the potential to reshape the cryptocurrency landscape and propel DeFiner to new heights in the world of decentralized finance.

"Fast Valuer" is not merely a tool; it is a game-changer. This automated product swiftly determines pricing based on cryptocurrency price fluctuations, offering real-time anomaly tracking using price and volume data. This innovative capability streamlines the integration of new currencies into the DeFiner platform, enhancing accessibility and offering a competitive edge in the ever-evolving cryptocurrency market.

2. Product advantages (need)

1.Real-Time Pricing Insights: The cryptocurrency market is known for its rapid fluctuations. "Fast Valuer" is designed to provide real-time pricing insights, ensuring that our DeFiner platform stays ahead of the curve. This capability is essential for making timely and informed decisions in a market that never sleeps.

2.Anomaly Tracking: Beyond pricing, "Fast Valuer" excels in anomaly tracking. By monitoring both price and volume data, it empowers our platform to identify hidden opportunities and market irregularities. This critical feature positions DeFiner to seize these opportunities and harness their potential.

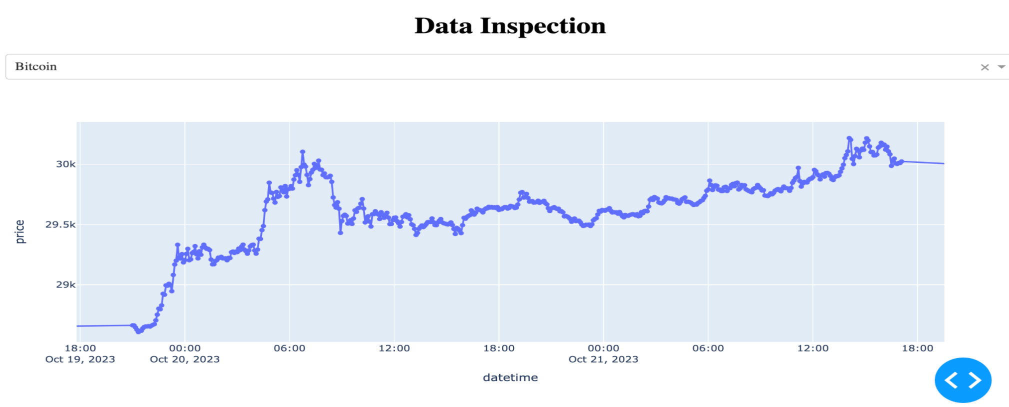
3.Simplified Integration: We understand the importance of seamless currency integration for DeFiner. "Fast Valuer" simplifies the onboarding of new currencies, enhancing diversification and offering a streamlined approach to scaling our platform's offerings.

4.Global Reach, Local Benefits: Our product is versatile, catering to the diverse needs of both seasoned traders and newcomers to the cryptocurrency scene. It brings global capabilities while maintaining a focus on localized benefits, aligning perfectly with DeFiner's mission and user-centric approach.

5.Data-Driven Decisions: In an ever-evolving market, data-driven decisions are paramount. "Fast Valuer" equips our platform with real-time, data-backed insights, enabling DeFiner users to stay agile and responsive in their investment strategies.

3. Data introduction

1. Data Source:
    a. We used the prices of top 10 crypto. Data is obtained from coinmarketcap.com
    b. Why this data:
        i. Price and volume play important roles in deciding whether a crypto is promising.
        ii. This data is always available, we can fetch the data whenever we want
        iii. This data is continuous, so we can aggregate the data to different levels, where different business logics and opportunities could happen.
        iv. The continuous data made it feasible to do time series analysis
    c. How we used the data:
        i. Visualization



        ii. Statistics analysis
        iii. ML to find anomalies
2. Demo
    a. Prototype walkthrough
    b. Data collection & scheduling
    c. Data visualization
    d. Data analysis
    e. Anomaly detection

4. Anomalies detection

4.1 Statistical Analysis

We calculated different features to illustrate the possibilities and directions of anomaly detection. Ideally, we would use different parameters to generate the features based on different crypto, but in this section, we use BTC to illustrate the concepts.

```
In [10]: df_btc_prc['itd_std'] = df_btc_prc['price'].expanding(min_periods=2).std()
         df_btc_prc['std_5p'] = df_btc_prc['price'].rolling(5).std()
         df_btc_prc['std_3p'] = df_btc_prc['price'].rolling(3).std()
         df_btc_prc['avg_3p'] = df_btc_prc['price'].shift(1).rolling(3).mean()
         df_btc_prc['prc_move'] = df_btc_prc['price'] - df_btc_prc['avg_3p']
```

```
In [11]: df_btc_prc.head(10)
```

Out[11]:

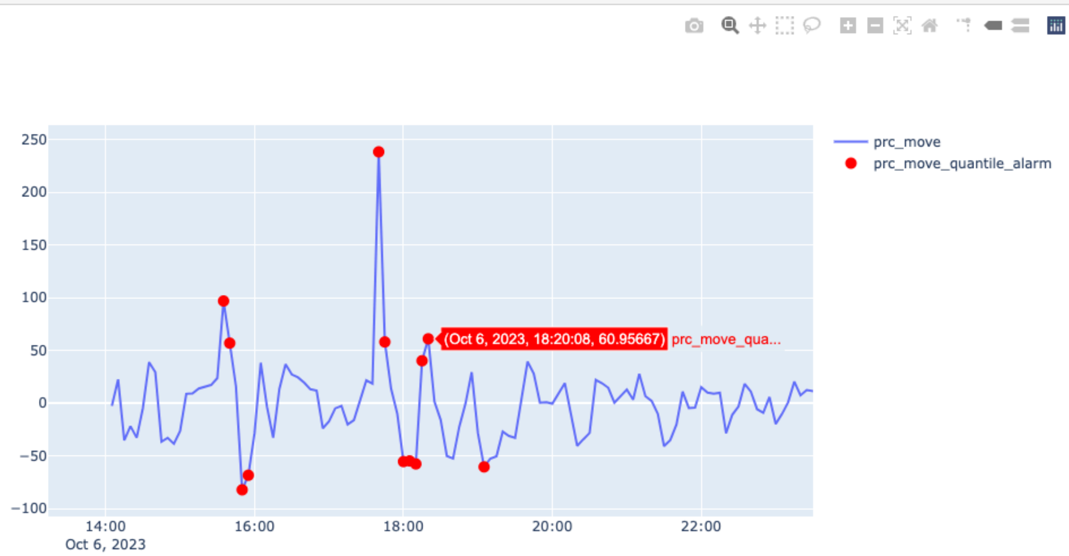
time	price	itd_std	std_5p	std_3p	avg_3p	prc_move
2023-10-06 13:50:24.682403	27988.90	NaN	NaN	NaN	NaN	NaN
2023-10-06 13:55:08.840394	27957.07	22.507209	NaN	NaN	NaN	NaN
2023-10-06 14:00:07.234874	27971.84	15.928724	NaN	15.928724	NaN	NaN
2023-10-06 14:05:07.708910	27970.10	13.065839	NaN	8.072189	27972.603333	-2.503333
2023-10-06 14:10:08.645585	27988.79	13.585623	13.585623	10.325100	27966.336667	22.453333
2023-10-06 14:15:06.776204	27941.59	18.371148	17.648076	23.769645	27976.910000	-35.320000
2023-10-06 14:20:06.366697	27945.02	19.192959	19.838429	26.316718	27966.826667	-21.806667
2023-10-06 14:25:07.250407	27925.61	22.837655	25.049692	10.359162	27958.466667	-32.856667
2023-10-06 14:30:06.799465	27932.67	23.372327	24.708931	9.824410	27937.406667	-4.736667
2023-10-06 14:35:06.393472	27973.26	22.560928	18.226852	25.716124	27934.433333	38.826667

In the first experiment, we used a very naive statistical way to find the anomalies. We used fixed 5% and 95% threshold on price to find the anomalies of prices, and here's the results:

```
In [13]: # fixed threshold:
col = 'price'
min_t = df_btc_prc[col].quantile(0.05)
max_t = df_btc_prc[col].quantile(0.95)

df_btc_prc[col+'_quantile_alarm'] = (df_btc_prc[col].clip(lower = min_t,upper=max_t) != df_btc_prc[col])
plot_anomaly(df_btc_prc[col],
             anomaly_pred = df_btc_prc[df_btc_prc[col+'_quantile_alarm']==True][col+'_quantile_alarm'])
```

2. Then We used feature price_move to do the second experiment, we still used 5% and 95% threshold to filter out the anomalies.

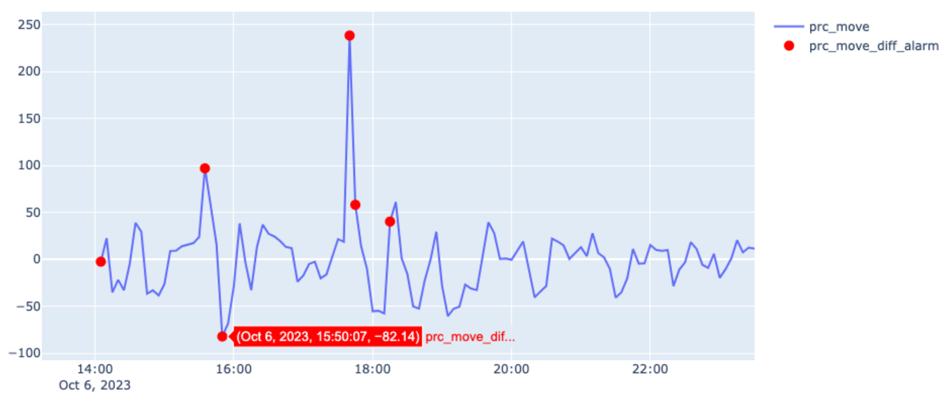
```
In [14]: # fixed threshold:
col = 'prc_move'
min_t = df_btc_prc[col].quantile(0.05)
max_t = df_btc_prc[col].quantile(0.95)

df_btc_prc[col+'_quantile_alarm'] = (df_btc_prc[col].clip(lower = min_t,upper=max_t) != df_btc_prc[col])
plot_anomaly(df_btc_prc[col],
             anomaly_pred = df_btc_prc[df_btc_prc[col+'_quantile_alarm']==True][col+'_quantile_alarm'])
```



As you can see this method to some extent takes seasonality into consideration, and it also made a reasonable detection report.

3.After that, we used the interquartile range of price_move of the previous N time window to limit the normal data; if the upcoming data is out of that range, it's an anomaly. Here's an example:

```
In [15]: # Windowed Quantile threshold
window = 1
col = 'prc_move'
df_btc_prc[col+'_diff'] = df_btc_prc[col].diff(periods= window)
Q1 = df_btc_prc[col+'_diff'].quantile(0.25)
Q3 = df_btc_prc[col+'_diff'].quantile(0.75)
IQR = Q3- Q1
c = 2
min_t = Q1 - c*IQR
max_t = Q3 + c*IQR
df_btc_prc[col+'_diff_alarm'] = (df_btc_prc[col+'_diff'].clip(lower = min_t,upper=max_t)!= df_btc_prc[col+'_diff'])
plot_anomaly(df_btc_prc[col],
             anomaly_pred = df_btc_prc[df_btc_prc[col+'_diff_alarm']==True][col+'_diff_alarm'])
```



We found it hard to decide the lookback window. The model gives a conservative detection report, which can be used in systems that require high accuracy of anomaly detection.

4.2 Isolation Forest

BTC data overview:

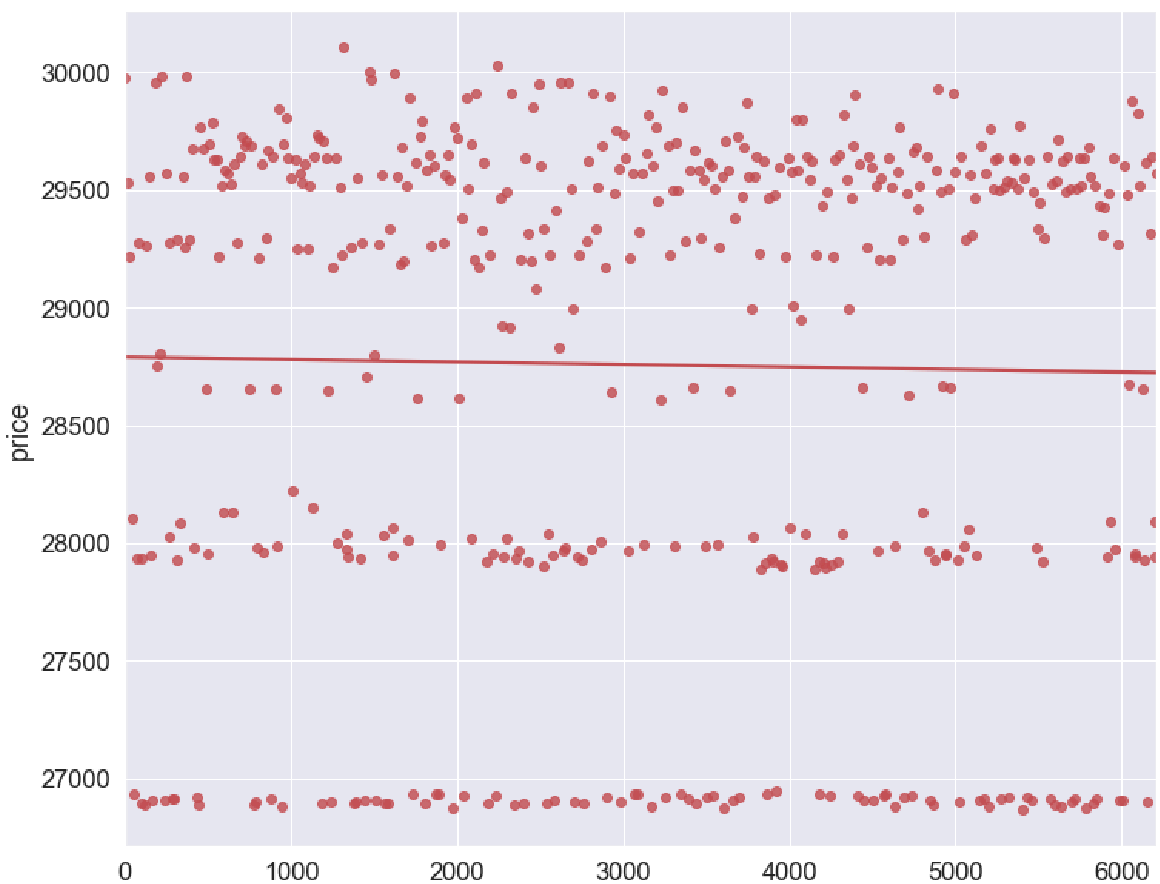
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6223 entries, 0 to 6222
Data columns (total 11 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   name                6223 non-null   object 
 1   symbol              6223 non-null   object 
 2   price               6223 non-null   float64
 3   change_1h           6223 non-null   float64
 4   change_24h          6223 non-null   float64
 5   change_7d           6223 non-null   float64
 6   market_cap          6223 non-null   int64  
 7   volume_24h_cash     6223 non-null   int64  
 8   volume_24h_token    6223 non-null   int64  
 9   circulating_supply  6223 non-null   int64  
 10  time                6223 non-null   object 
dtypes: float64(4), int64(4), object(3)
memory usage: 534.9+ KB
```

	name	symbol	price	change_1h	change_24h	change_7d	market_cap	volume_24h_cash	volume_24h_token	circulating_supply	time
0	Bitcoin	BTC	29976.29000	0.0058	0.0541	0.1197	583216899100	20638601990	690726	19518900	2023-10-20 06:40:07.217668
1	Ethereum	ETH	1622.45000	0.0067	0.0469	0.0501	194743284962	6405434454	3955717	120264972	2023-10-20 06:40:07.217668
2	Tether USDt	USDT	1.00000	-0.0000	0.0001	0.0010	83968887148	39212347282	39192065877	83925456770	2023-10-20 06:40:07.217668
3	BNB	BNB	215.54000	0.0040	0.0242	0.0499	32699085838	338402718	1569998	151705326	2023-10-20 06:40:07.217668
4	XRP	XRP	0.51690	-0.0024	0.0770	0.0777	27623661521	1752940800	3391257789	53441027384	2023-10-20 06:40:07.217668
5	USDC	USDC	0.99990	-0.0001	-0.0003	-0.0001	25510301771	3336379745	3336759586	25513206071	2023-10-20 06:40:07.217668
6	Solana	SOL	27.02000	0.0064	0.1279	0.2634	11256503955	996821817	36892010	416599090	2023-10-20 06:40:07.217668
7	Cardano	ADA	0.25190	0.0026	0.0416	0.0259	8872246995	135099806	536310646	35220483670	2023-10-20 06:40:07.217668
8	Dogecoin	DOGE	0.06017	0.0005	0.0336	0.0395	8513863462	144351535	2398907456	141487726384	2023-10-20 06:40:07.217668
9	TRON	TRX	0.09174	-0.0015	0.0329	0.0773	8155042268	217014907	2365620983	88895916554	2023-10-20 06:40:07.217668

	name	symbol	price	change_1h	change_24h	change_7d	market_cap	volume_24h_cash	volume_24h_token	circulating_supply	time
0	Bitcoin	BTC	29976.29	0.0058	0.0541	0.1197	583216899100	20638601990	690726	19518900	2023-10-20 06:40:07.217668
16	Bitcoin	BTC	29527.81	-0.0010	0.0265	0.1059	576192789785	22675784647	768165	19519100	2023-10-20 12:10:06.755215
29	Bitcoin	BTC	29217.08	0.0101	0.0338	0.0912	570292085975	15676368048	536534	19518618	2023-10-19 23:45:06.747351
42	Bitcoin	BTC	28103.55	0.0049	0.0246	0.0445	548186022516	13184967323	469157	19505937	2023-10-06 18:15:07.279526
55	Bitcoin	BTC	26932.33	0.0040	-0.0056	0.0122	525075704573	11121935207	413029	19499425	2023-09-29 15:36:11.545694
...	...	...	...	...	...	...	...	...	...	...	...
6170	Bitcoin	BTC	29316.47	0.0022	0.0363	0.0923	572157506393	17381415501	592954	19518725	2023-10-20 03:15:07.067325
6186	Bitcoin	BTC	29643.16	0.0004	0.0346	0.0964	578526314955	21444031053	723515	19519312	2023-10-20 17:45:06.416040
6199	Bitcoin	BTC	27941.59	0.0035	0.0177	0.0437	545288837435	12467441014	445979	19505775	2023-10-06 14:15:06.776204
6204	Bitcoin	BTC	28093.76	0.0035	0.0242	0.0455	547990472258	13233441615	471051	19506012	2023-10-06 18:30:07.156321
6210	Bitcoin	BTC	29570.18	-0.0010	0.0284	0.1047	577063418439	22274336132	753430	19519175	2023-10-20 12:35:07.496750

470 rows × 11 columns

Price visualization:



Isolation Forest model: A tree-based **unsupervised learning** algorithm for anomaly detection that works on the principle of isolating anomalies. It's particularly efficient for high-dimensional and time series datasets.

How does it work: The algorithm works by randomly selecting an attribute and randomly selecting a split value between the maximum and minimum values for that attribute. This partitioning is done many times until the algorithm has isolated each point in the dataset.

Steps:

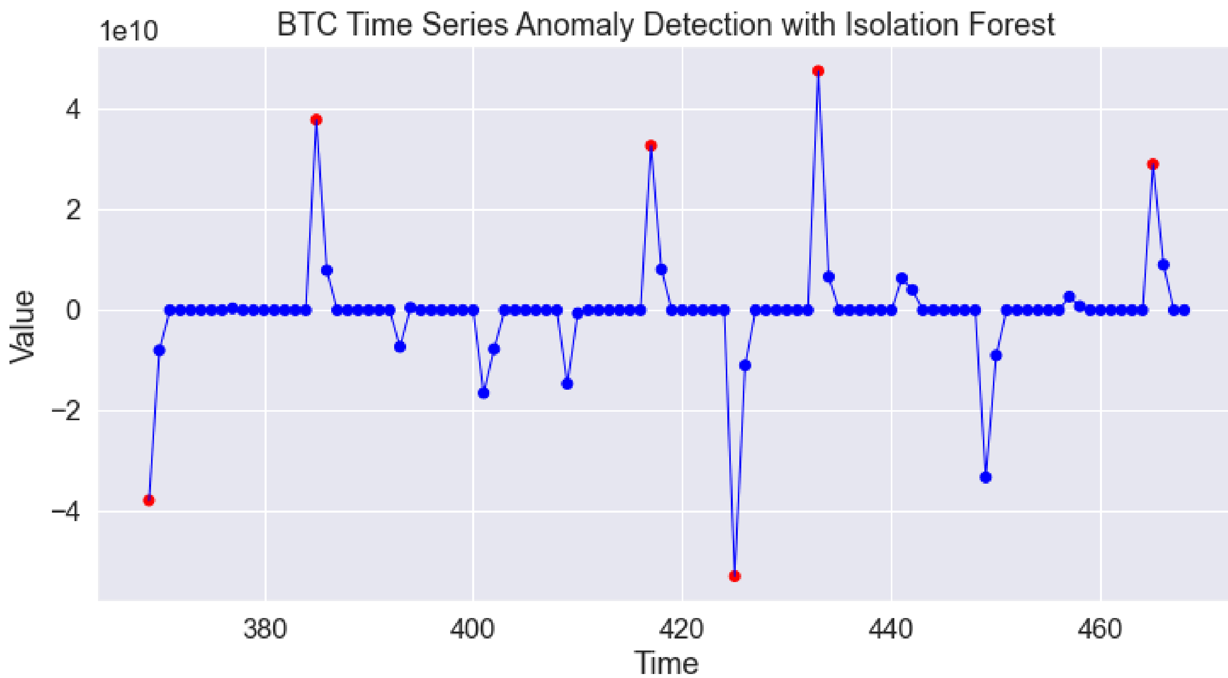
1. A random subsample of the data is selected from the given dataset and assigned to a binary tree.
2. Branching of the tree starts by selecting a **random feature** (from the set of all N features). And then branching is done on a random threshold (any value in the range of minimum and maximum values of the selected feature).
3. If the value of a data point is less than the selected threshold, it goes to the left branch else to the right. And thus a node is split into left and right branches.
4. This process continues recursively till each data point is completely isolated or till max depth(if defined) is reached.
5. The above steps are repeated to construct **random binary trees**.

Build the model:

```
# Create and fit the Isolation Forest model
isolation_forest = IsolationForest(contamination = 0.05, random_state = 0)
isolation_forest.fit(data_diff)
```

```
▾                    IsolationForest
IsolationForest(contamination=0.05, random_state=0)
```

Predict anomaly scores for each data point:



Limitations:
1. The final anomaly score depends on the contamination parameter.
2. Inherent Bias when detecting anomalies that are more "isolatable" based on the data's structure.
3. Dimensionality reduction might be needed to achieve higher performance.

4.3 LSTM Autoencoder

An LSTM has a similar control flow as a recurrent neural network. It processes data passing on information as it propagates forward. The differences are the operations within the LSTM's **cells**.

Autoencoders are a type of **self-supervised learning model** that can learn a compressed representation of input data. The goal is to **minimize reconstruction error based on a loss function**, such as the mean squared error.

An LSTM Autoencoder is an implementation of an autoencoder for **sequence data** using an **Encoder-Decoder LSTM architecture**.

Steps

(1) Download 5-minute Bitcoin Data from Yahoo! Finance

```
In [2]:  1 df_BTC = yf.download('BTC-USD', "2023-08-31", "2023-10-17", interval='5m')
[*********************100%%**********************]  1 of 1 completed
```

(2) Data splitting

Take the first 20% window as the test set, the next 80% as the training set.

(3) Normalizing data with MinMaxScaler

After the data has been split into training, validation and test sets, scaling is done by training set.

```
scaler = MinMaxScaler().fit(train[['Close']])
```

(4) Define lookback period

A “lookback period” defines **how many previous timesteps are used** in order to predict the subsequent timestep. For example, the lookback period is set to **5** means we are using the time steps at **t-4, t-3, t-2, t-1, and t** to **predict the value at time t+1**.

```
# Lookback window is 3 time steps
```

```python
#### Sequences generation and dataset creation, lookback window is 3
def shift_samples(data,column_name,lookback=3):
    """This function takes a *data* dataframe and returns two numpy arrays:
    - X corresponds to the same values but packed into n frames of *lookback* values each
    - Y corresponds to the sample shifted *lookback* steps to the future
    """
    data_x = []
    data_y = []
    for i in range(len(data) - int(lookback)):
        x_floats = np.array(data.iloc[i:i+lookback])
        y_floats = np.array(data.iloc[i+lookback])
        data_x.append(x_floats)
        data_y.append(y_floats)
    return np.array(data_x), np.array(data_y)
```

```python
import gc
X_train, y_train = shift_samples(train[['Close']],train.columns[3])
X_test, y_test = shift_samples(test[['Close']], test.columns[3])
gc.collect()


print("Final datasets' shapes:")
print('X_train: '+str(X_train.shape)+', y_train: '+str(y_train.shape))
print('X_test: '+str(X_test.shape)+', y_train: '+str(y_test.shape))
```

```
Final datasets' shapes:
X_train: (10817, 3, 1), y_train: (10817, 1)
X_test: (2701, 3, 1), y_train: (2701, 1)
```

(5) LSTM Modelling

The adam optimizer is used, and loss function is MAE.

tsteps = 3

nfeatures = 1

```python
##### Anomaly detectors' training

detector = Sequential()
detector.add(layers.LSTM(128, input_shape=(tsteps, nfeatures),dropout=0.2))
detector.add(layers.Dropout(rate=0.5))
detector.add(layers.RepeatVector(tsteps))
detector.add(layers.LSTM(128, return_sequences=True,dropout=0.2))
detector.add(layers.Dropout(rate=0.5))
detector.add(layers.TimeDistributed(layers.Dense(nfeatures)))

detector.compile(loss='mae', optimizer='adam')
detector.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_4 (LSTM)               (None, 128)               66560

 dropout_2 (Dropout)         (None, 128)               0

 repeat_vector_1 (RepeatVec  (None, 3, 128)            0
 tor)

 lstm_5 (LSTM)               (None, 3, 128)            131584

 dropout_3 (Dropout)         (None, 3, 128)            0

 time_distributed_1 (TimeDi  (None, 3, 1)              129
 stributed)

=================================================================
Total params: 198273 (774.50 KB)
Trainable params: 198273 (774.50 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
# Fit the model on the training set
# ModelCheckpoint saves the best model obtained during training

checkpoint = ModelCheckpoint("detector3.hdf5", monitor='val_loss',
verbose=1,save_best_only=True, mode='auto', period=1)

history3 = etector.fit(X_train,y_train,epochs=50,batch_size=128,verbose=1,
validation_split=0.1, callbacks=[checkpoint],shuffle=False)
```

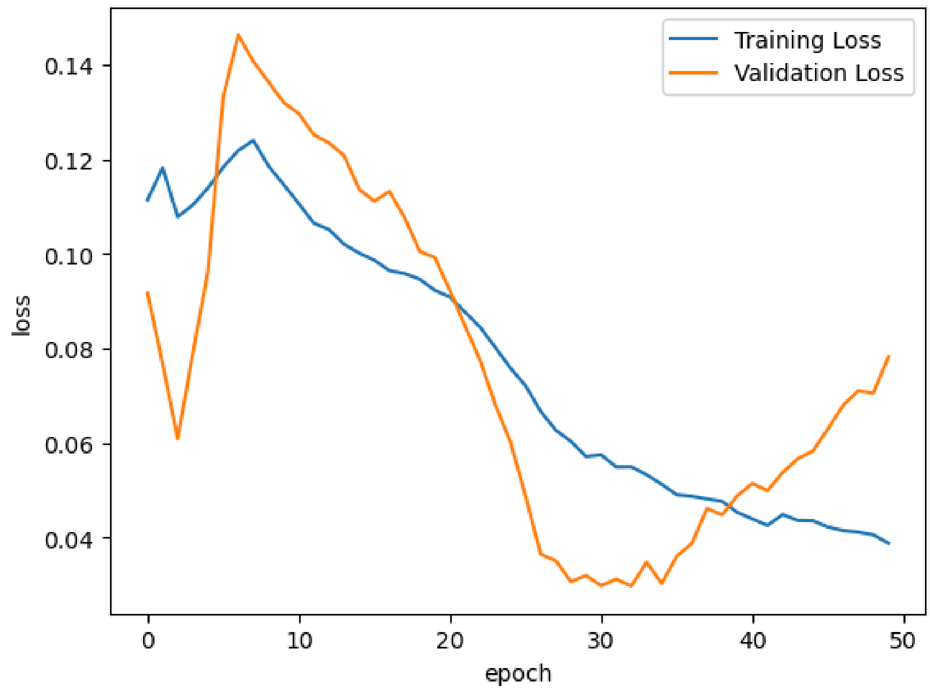
```
75/77 [============================>.] - ETA: 0s - loss: 0.0419
Epoch 46: val_loss did not improve from 0.02969
77/77 [==============================] - 1s 17ms/step - loss: 0.0422 - val_loss: 0.0630
Epoch 47/50
76/77 [============================>.] - ETA: 0s - loss: 0.0414
Epoch 47: val_loss did not improve from 0.02969
77/77 [==============================] - 1s 15ms/step - loss: 0.0414 - val_loss: 0.0679
Epoch 48/50
76/77 [============================>.] - ETA: 0s - loss: 0.0411
Epoch 48: val_loss did not improve from 0.02969
77/77 [==============================] - 1s 17ms/step - loss: 0.0411 - val_loss: 0.0710
Epoch 49/50
73/77 [===========================>..] - ETA: 0s - loss: 0.0396
Epoch 49: val_loss did not improve from 0.02969
77/77 [==============================] - 1s 18ms/step - loss: 0.0405 - val_loss: 0.0705
Epoch 50/50
76/77 [============================>.] - ETA: 0s - loss: 0.0388
Epoch 50: val_loss did not improve from 0.02969
77/77 [==============================] - 1s 14ms/step - loss: 0.0388 - val_loss: 0.0782
```

```
# Load the best model obtained during training
detector = load_model("detector3.hdf5")
detector.evaluate(X_test, y_test)
```

```
85/85 [==============================] - 1s 4ms/step - loss: 0.0282

Out[16]: 0.028156884014606476
```

```
# Visualization of the training loss and validation loss
```

The validation loss has an initial increase, and starts to decrease after 8 epochs. But it gets bigger after 30 epochs.



(6) Generate the prediction and determining MAE threshold for Autoencoder detector

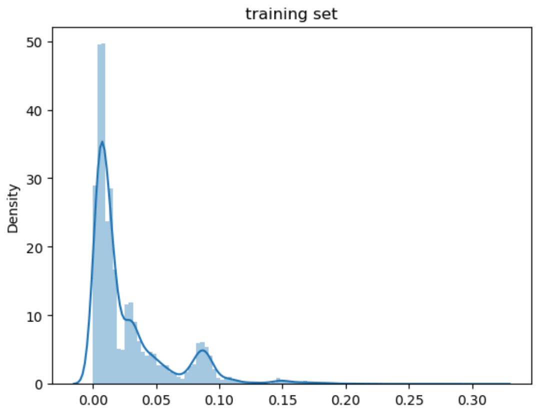
Anomaly will be detected when the error is larger than the selected threshold value. We visually determine the threshold as **0.05** by using the MAE loss for the training and test sets.

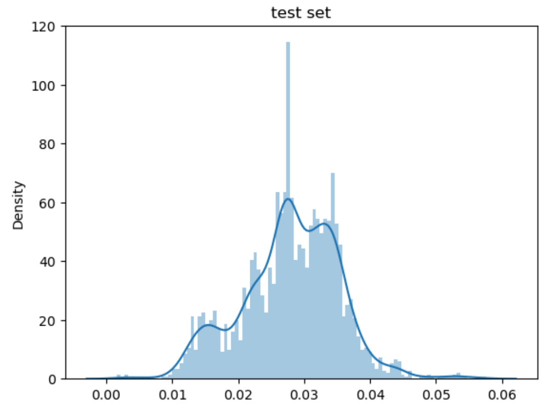
```
### Determining threshold for Autoencoder detector

X_train_pred = detector.predict(X_train)
loss_mae = np.mean(np.abs(X_train_pred - X_train), axis=1) # MAE formula
sns.distplot(loss_mae, bins=100, kde=True).set_title('training set')
plt.show()
```

```
339/339 [==============================] - 1s 4ms/step
```

```
X_test_pred = detector.predict(X_test)
loss_mae = np.mean(np.abs(X_test_pred - X_test), axis=1)
sns.distplot(loss_mae, bins=100, kde=True).set_title('test set')
plt.show()
```

```
85/85 [==============================] - 0s 4ms/step
```





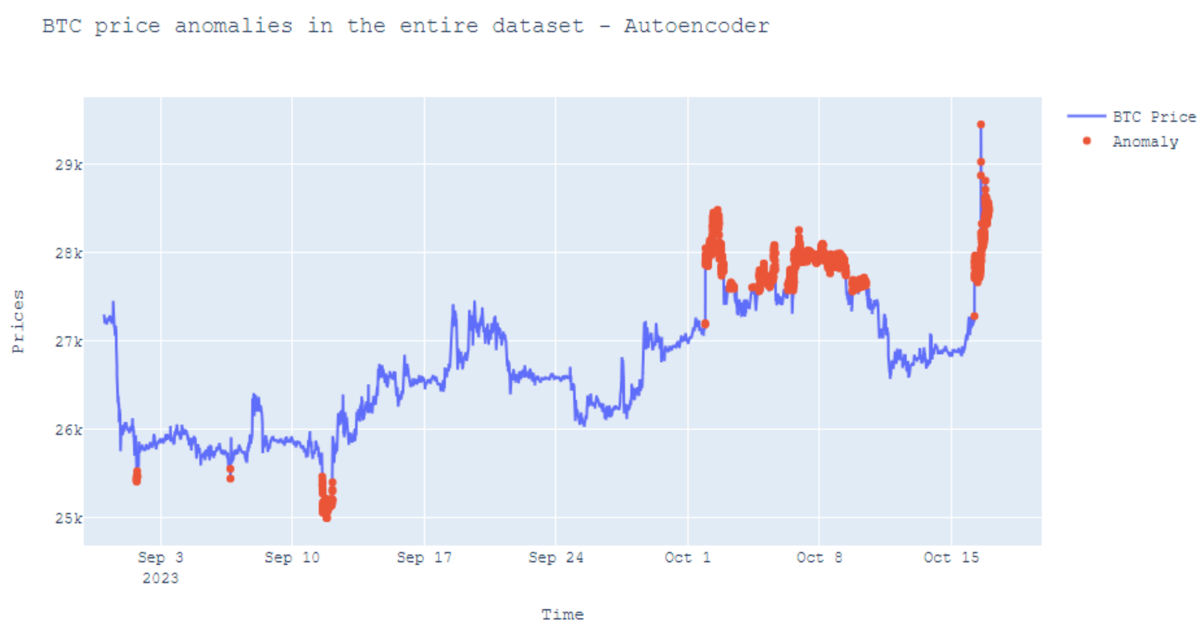
Plot of anomalies in the test set:

```
test_df[['Close','loss','threshold','anomaly']]
```

Datetime	Close	loss	threshold	anomaly
2023-08-31 00:15:00+00:00	0.509160	0.020061	0.05	False
2023-08-31 00:20:00+00:00	0.506958	0.020033	0.05	False
2023-08-31 00:25:00+00:00	0.507877	0.019322	0.05	False
2023-08-31 00:30:00+00:00	0.503364	0.019475	0.05	False
2023-08-31 00:35:00+00:00	0.498249	0.018652	0.05	False
...	...	...	...	...
2023-09-09 08:55:00+00:00	0.196808	0.027035	0.05	False
2023-09-09 09:00:00+00:00	0.196517	0.027010	0.05	False
2023-09-09 09:05:00+00:00	0.195588	0.027423	0.05	False
2023-09-09 09:10:00+00:00	0.196226	0.027600	0.05	False
2023-09-09 09:15:00+00:00	0.196446	0.027709	0.05	False

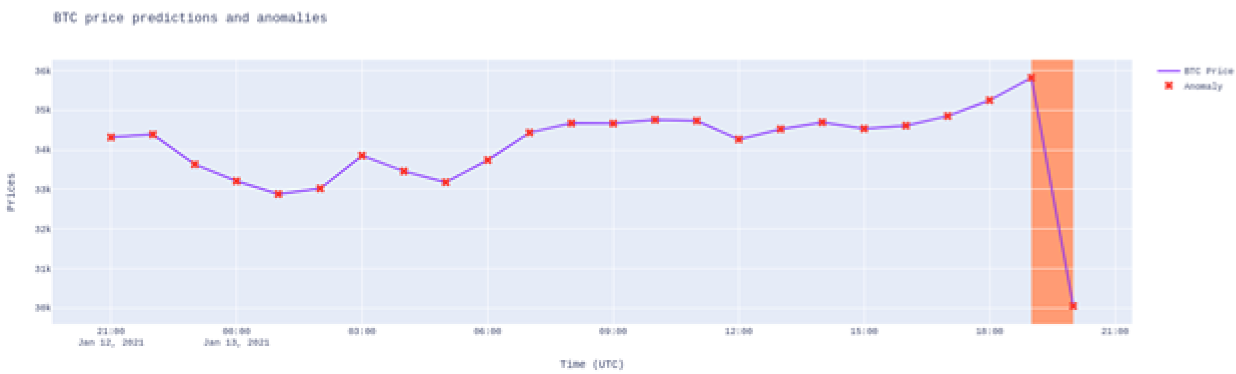
2701 rows × 4 columns

(7) Generate the prediction for the entire 13,521 Close Prices with 1,880 anomalies:



(8) Forecast anomalies on streaming Bitcoin prices

Connect to the Scraper and generate the real time alert.



5. Evaluation & Iteration

4.1 Model Evaluation

For model evaluation, we can experiment with different anomaly detection models and compare their performances to select the most suitable one. Model's performance can be evaluated on historical datasets, utilizing techniques like cross-validation and time-series splitting.

Our Statistics-based , LSTM autoencoder model, and isolation forest model, most common evaluation metrics are **MSE** (mean squared error), **RMSE**(root mean squared error). They measure the average (square root) of the squared errors between the model's predictions and the actual values. **MAE** and **MAPE** measure the absolute value (percentage) difference between predictions and the actual values; they are less sensitive to outliers but may not account for larger errors.

Precision and **recall** are also useful measures to evaluate classification. Precision measures the proportion of data points labeled as anomalies by the model that are actually true anomalies, while recall measures how many true anomalies are successfully detected. **ROC curve** plots true positive rate vs. false positive rate at different classification thresholds, which could be a useful graphic tool to visualize if our algorithms detects true anomalies with skillfulness and with low cost of false positive predictions. Precision-Recall curve can also serve as an important tool for us to illustrate the trade-off between precision and recall, especially when our price data have few anomalies.

Beyond, Isolation Forest models typically assign an **anomaly score** to each data point, indicating the degree to which a data point is considered an anomaly. Thresholds can be set based on these scores to determine which data points should be labeled as anomalies.

4.2 Gather User Feedback

We can continuously perform evaluation, gather user feedback, and monitor the product performance to optimize the product and improve user experience. Our product sends email alert notifications from the terminal once anomalies are detected. User training and support are also important services to ensure they understand and can communicate issues about product usage.

6. Potential improvements

f. How to better attract old users.
g. Choose from multiple new cryptos.
h. Develop more features to detect anomalies.
i. Email notification using Mutt.

Project Proposal:

Scraper

- Input: https://coinmarketcap.com/ top 10 crypto all info
- Output: generates one .csv file with timestamp
- Logs: saves in ./logs with timestamp
- Results:
  - data:

	A	B	C	D	E	F	G	H	I	J	K
1	name	symbol	price	change_1h	change_24h	change_7d	market_cap	volume_24h_	volume_24h_	circulating_s	time
2	Bitcoin	BTC	26928.81	0.0035	-0.0063	0.0126	5.2516E+11	1.1084E+10	411535	19499425	40:08.5
3	Ethereum	ETH	1665.14	0.0016	0.0053	0.0431	2.0024E+11	5784761819	3473548	120236031	40:08.5
4	Tether USDt	USDT	1	0	0	-0.0003	8.3266E+10	3.2047E+10	3.2046E+10	8.3277E+10	40:08.5
5	BNB	BNB	215.67	0.0019	0.0025	0.0218	3.318E+10	341300402	1582540	153846752	40:08.5
6	XRP	XRP	0.532	-0.0029	0.0478	0.0308	2.8364E+10	1561138975	2934281650	5.3312E+10	40:08.5
7	USDC	USDC	1	0.0002	-0.0001	-0.0002	2.5229E+10	2630658784	2630033065	2.5223E+10	40:08.5
8	Cardano	ADA	0.2496	-0.0018	0.0032	0.0214	8757111081	158739767	635972501	3.5128E+10	40:08.5
9	Dogecoin	DOGE	0.06196	0.0024	0.0091	0.0067	8749861950	139052212	2244080923	1.4121E+11	40:08.5
10	Solana	SOL	20.28	0.0054	0.0323	0.0423	8372859967	239661658	11817080	412843476	40:08.5
11	TRON	TRX	0.08967	0.0033	0.0388	0.075	7987082265	228315823	2546282640	8.9076E+10	40:08.5
12	Toncoin	TON	2.21	0.0021	-0.011	-0.0397	7597867448	29895984	13498165	3431892088	40:08.5
13	Dai	DAI	0.9999	0.0003	-0.0007	0	5346681228	94449773	94453120	5347888596	40:08.5
14	Polkadot	DOT	4.08	0.0001	-0.0008	0.0151	5008615887	79366375	19467038	1229009055	40:08.5

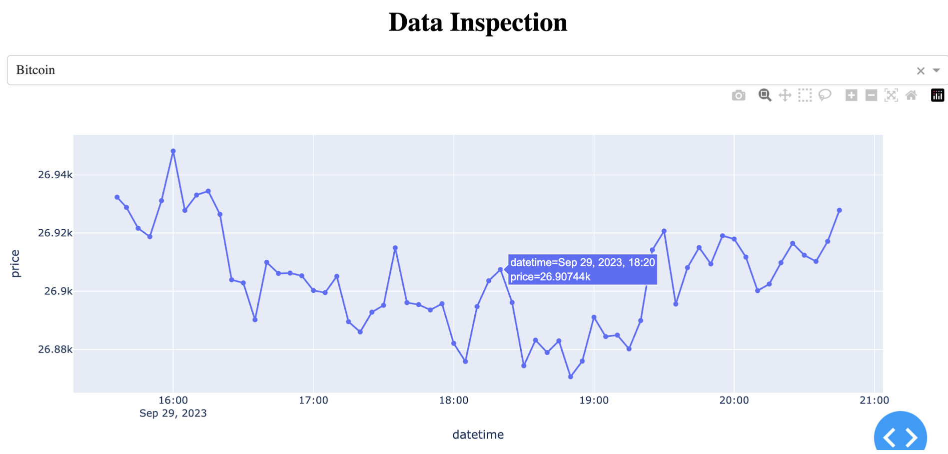
  - file structure:

```
G3_crypto_proj
  > .ipynb_checkpoints
  > data
  > logs
  > src
```

Monitor:

- Target:
  - Job start/end/duration time
  - Price dip/surge alert

- Output: Interactive plots



- Tool: Plotly/Dash

Analyzer:

- Anomalies detection:
    - Crypto price change VS historical price change (statistics/time series)
- Tool: jupyter notebook
- Output: Anomalies Alerts/Messages
    - Oral Demo (low fidelity prototype)

Scheduler:

- Target: every 5 minutes run scraper & analyzer
- Output:
    - Shell script*
    - Airflow Python script
- Tool: Shell script, Crontab or Airflow (if time allows)

3. Evaluation & Iteration
    a. Existing users using new crypto
    b. User feedback