Smart Natural Language Querying and Forecasting

Udacity Capstone Project Report

Gbadamosi Farouk

5th of November 2021

UDACITY

Project Overview

Natural Language Processing (NLP) is one of the most powerful fields in Machine Learning/Artificial Intelligence that helps to further bridge the communication gap between humans and the machine. NLP leverages advanced mathematical concepts that can easily map natural languages like English, Spanish, French, etc. to machine-understandable language. This innovation currently is at the heart of some of the most advanced products in the market which include, Apple's Siri, Google-translate, Interactive Voice Receiver, and so on.

The ability to completely remove the barriers between human and computer interactions and creating a universally accepted language is a fairly complex problem and still an area of continuous research. This area is popularly referred to as Natural Language Interfaces (NLI).

Problem & Solution Statements

It is no longer news that data is the greatest asset of any organization. Organizations are realizing the need to have a resilient data strategy that provides a 360-degree view of their business. Today, C-Level executives, decision-makers are constantly demanding insights and forecasts from their data increasing the pressures on their data teams (data engineers, data scientists, and analysts). Analysts are spending more time than ever developing business reports leaving little room for innovative and critical thinking.

One of the biggest challenges to automating business reporting and empowering C-Level executives has been querying the datasets according to the user's discretion. To interact with most databases, special skillsets such SQL is necessary to dice and slice the data that most end-users lack. In addition, most dashboards or charts today are developed using pre-defined queries based on user's demand. The challenge is that user's demands are constantly shifting as there is always a new question that needs to answered. This makes the process of constantly updating dashboards impracticable to accommodate all requests from users.

The **Smart Natural Language Query and Forecasting Solution** is being proposed to democratize access to insights using natural language. Users will now be empowered to ask questions about their data directly using free form text and will be given real-time answers in the form of charts and visuals. With this solution, data analysts can focus on automating ETL pipelines and other innovative data modeling tasks rather than responding to every new question from the end-user. Also, this solution has a plethora of use cases but most notably, this solution can be used to replace the Frequently Asked Questions (FAQ) sections of most apps today.

Datasets and Inputs

This is a supervised learning task and each dataset comes with target labels. Two types of datasets will be employed in this project. One is the training dataset and the other is a custom dataset. Both datasets are described below:

1. **Training Dataset**: The **SparC** dataset that we would be used to train our model. The SParC dataset is a multi-domain or multi-context dataset that contains over 200 databases from over 100 different domains. The dataset was developed by Yale & Salesforce from about 12K unique coherent questions from user interactions in those various domains. The dataset is one of the most used and widely cited for the text to SQL challenge and currently maintains a leaderboard

of the most “state-of-the-art” models built on its dataset. The dataset can be downloaded from the official page of the challenge

2. **Custom Dataset**: This is the dataset that will be used to test and tune our model. This database to the SParC dataset to test our model’s text-to-SQL conversion performance on the new schema. In addition, an interactive web application dashboard would be developed where users can query to generate insights. This dataset can be accessed at: Cryptocurrency time series – which is a minute by minute volume and value data of popular cryptocurrencies like Bitcoin, Ethereum and Litecoin.

Methodology

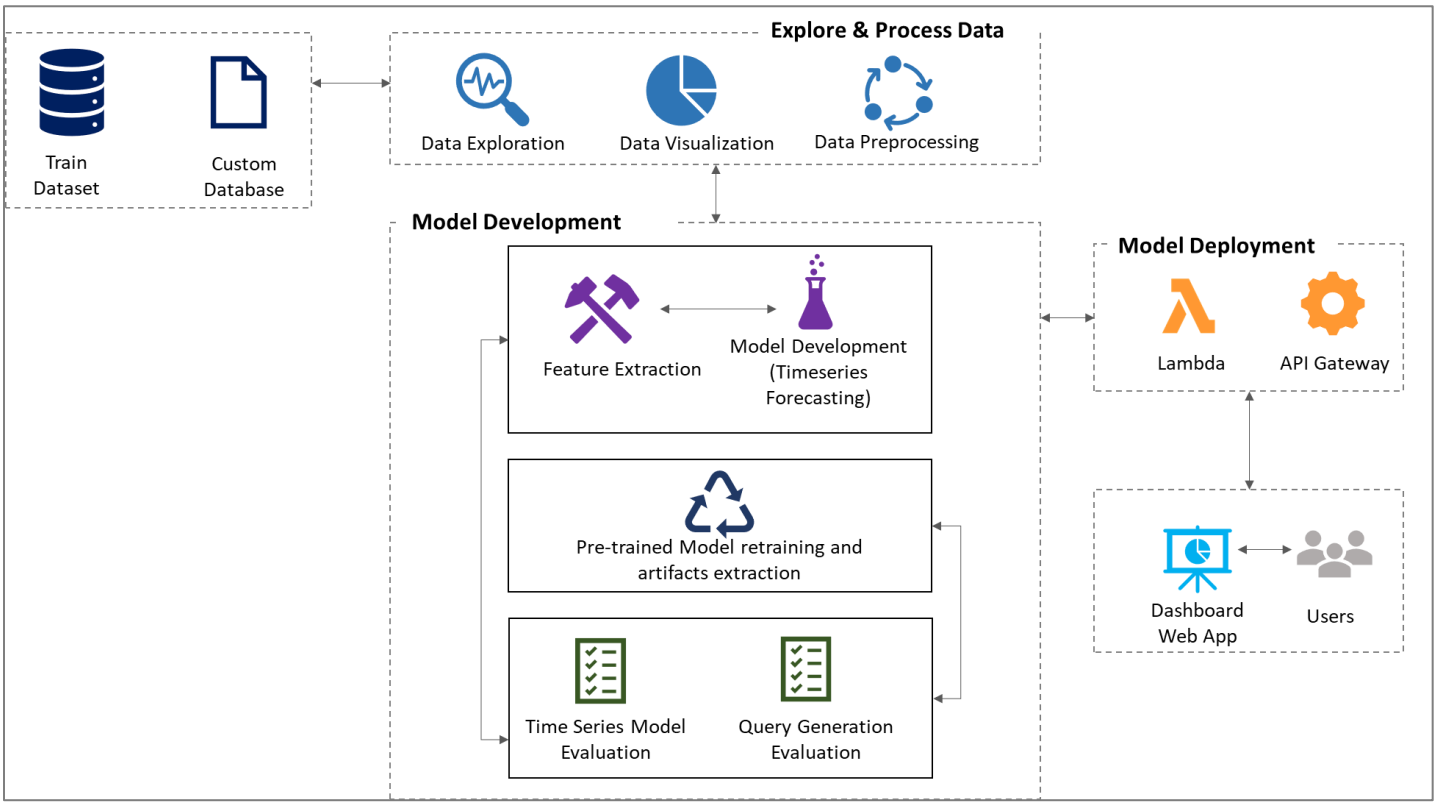

Figure 1: High Level Solution Architecture

Some important things to highlight in the high-level architecture:

1. **The Model**: Two models will be employed in this project – the forecast model and the text-to-SQL model. The text-to-SQL model would be pre-trained. The pre-trained model would be a deep learning framework from any of the benchmark models or a combination of benchmark models. The choice of the pre-trained model would be determined by **model simplicity, compatibility with AWS Sagemaker and computational requirement** while the choice of the forecast model would be determined by the **performance on validation set**.

2. **The Evaluation Metric**: The time-series prediction is a regression task; therefore, the evaluation metric would be **mean-absolute-error** while for the pre-trained model, a combination of **predicted and answer query comparison and accuracy of query result.**

3. **The Lambda function**: The lambda function would be first used to determine the user's intent – whether a forecast or query – before the data is preprocessed and passed to the right model for results.

Analysis

Data Exploration & Visualization

The custom dataset contained **658K** records of minute by minute Bitcoin price information (Open and Closing Prices). The dataset had zero missing data and the data time span was between **1/1/2020 0:00 to 4/20/2021 0:02.**

In preparation for the baseline modelling, **Autoregressive Integrated Moving Average (ARIMA)** was chosen due to its popularity, efficacy and ease of use. ARIMA is a statistical model that depends heavily on the most recent data points to model and forecast the time series.

The time series closing price data was tested for stationarity using Augmented Dickey-Fuller Test (ADF) which is the first step required before using an ARIMA model. A stationary series is one that reverts to the mean after a few cycles while a non-stationary series does not revert to the mean but continues on a drifting path. Stationary time series are easier to model for obvious reasons.

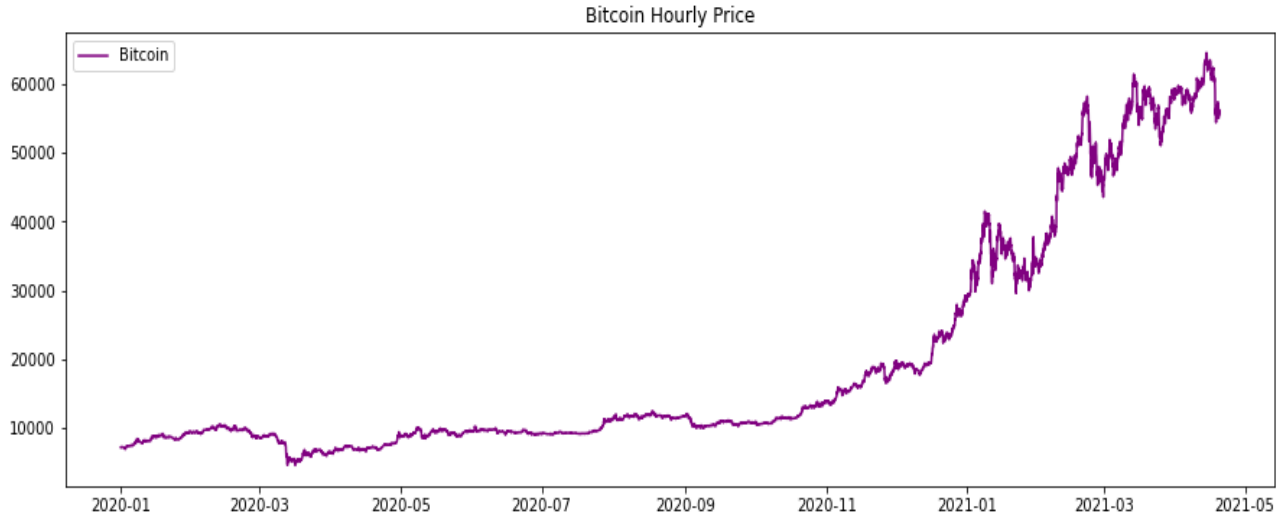

Figure 2: Bitcoin Hourly Price Trend from January 2020 to April 2021

The results of the ADF test reveal that BTC as expected, is a non-stationary time series (non-mean reverting) hence the need to transform the series in order to achieve stationarity.

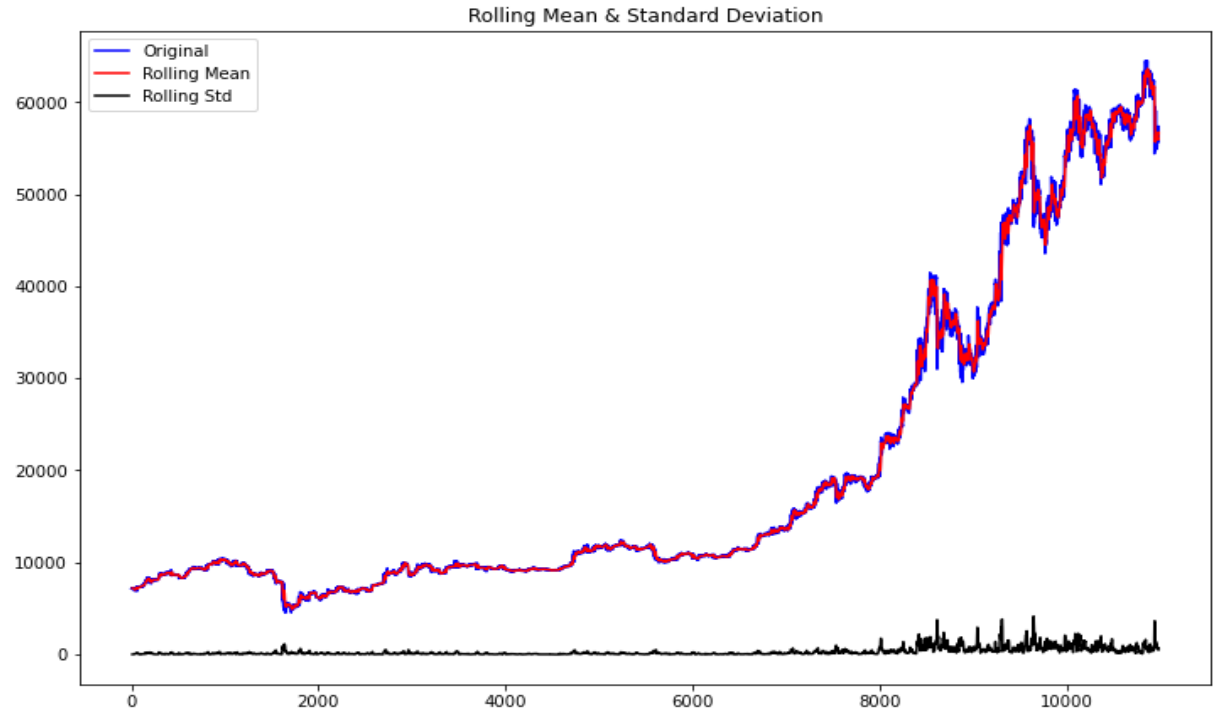

Figure 3: BTC test for stationarity visualization

Results of Dickey-Fuller Test:
p-value = 0.9969. The series is likely non-stationary.

Test Statistic	1.358134
p-value	0.996915
#Lags Used	20.000000
Number of Observations Used	10966.000000
Critical Value (1%)	-3.430946
Critical Value (5%)	-2.861804
Critical Value (10%)	-2.566910

Data Preprocessing for Baseline Modeling

In this project, we want to understand the hourly trend of BTC and be able to predict Bitcoin prices in the next few hours therefore, the data was aggregated on an hourly basis and prices were averaged within each hour. The resulting dataset had **10,987** records

For the baseline modeling, the time series was transformed using a technique called time differencing. Time differencing subtracts the previous data point from the next data point. The newly transformed time series was subjected to the same ADF test and the series was confirmed stationary (p value = 0.00). At this point, the time series was ready for modelling and prediction.

Baseline Modeling

Similar to other Machine Learning tasks, the data would be trained and validated on a train and validation set respectively. The only difference with the ARIMA model training is that the validation data would be passed to the model recursively for predicting the next data point. This prediction would be compared to the validation data point to evaluate the training performance. The order of the ARIMA model is (1,1,1) after displaying the Autocorrelation and Partial autocorrelation charts.

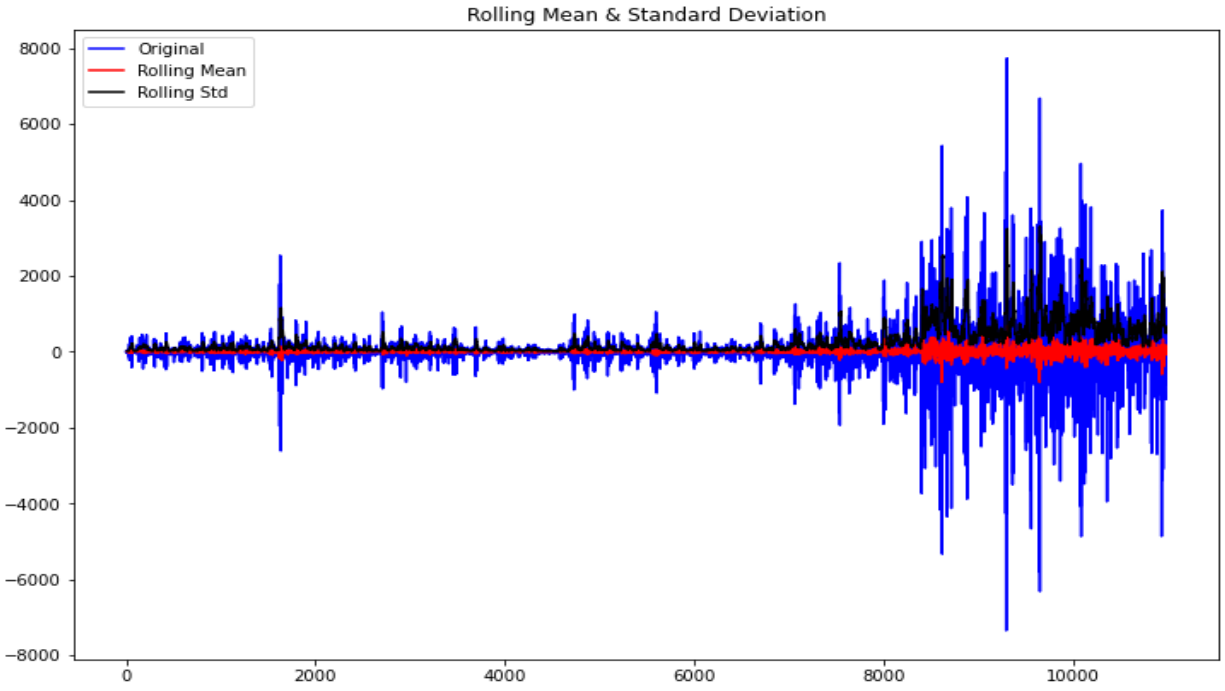

Figure 4: Time differenced BTC test for stationarity visualization

Results of Dickey-Fuller Test:
p-value = 0.0000. The series is likely stationary.

Test Statistic	-28.740895
p-value	0.000000
#Lags Used	20.000000
Number of Observations Used	10966.000000
Critical Value (1%)	-3.430946
Critical Value (5%)	-2.861804
Critical Value (10%)	-2.566910

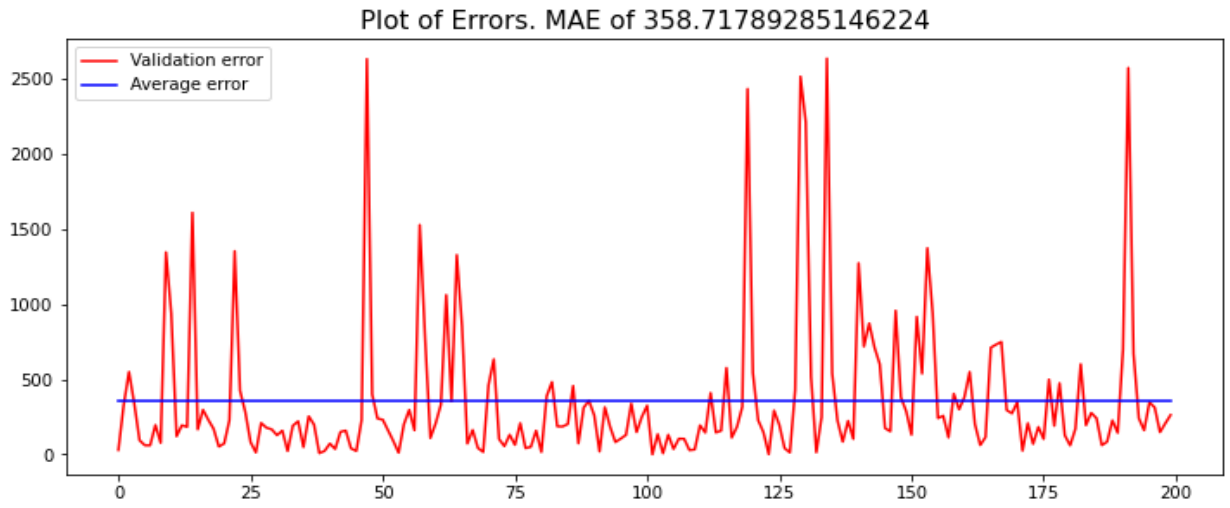

Figure 5: Validation Error

Over 95% of the data was used for training while about 2% was used for validation and the remaining dataset was used for testing. The training time was the biggest consideration in the data split due to the recursive nature of the training. The model produced a **validation mean-absolute-error (MAE) of 358** and **test mean-absolute-error of 3442.** This formed the baseline for the eventual time series model.

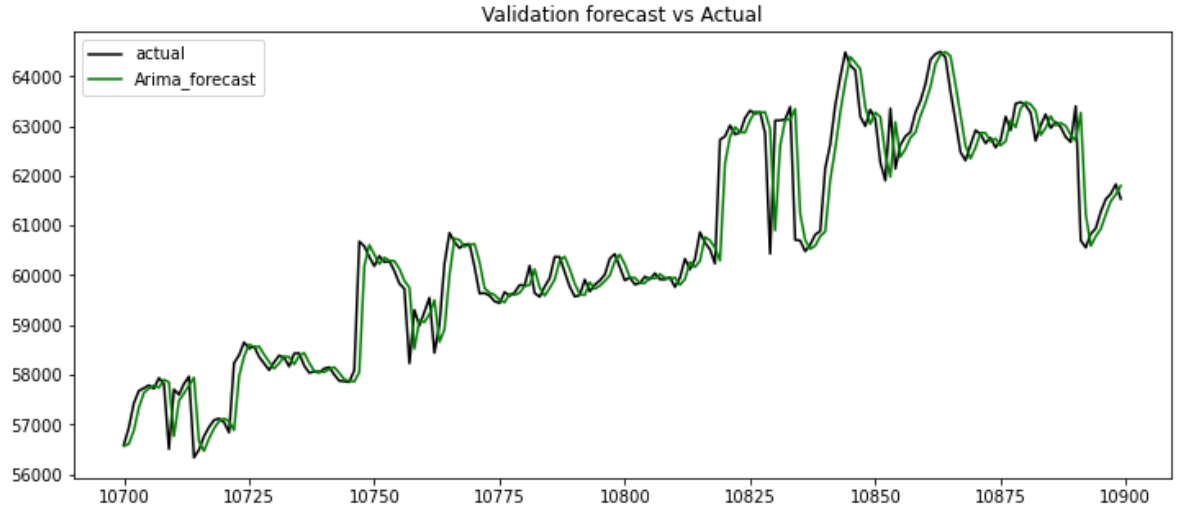

Figure 6: Validation Forecast vs Actual

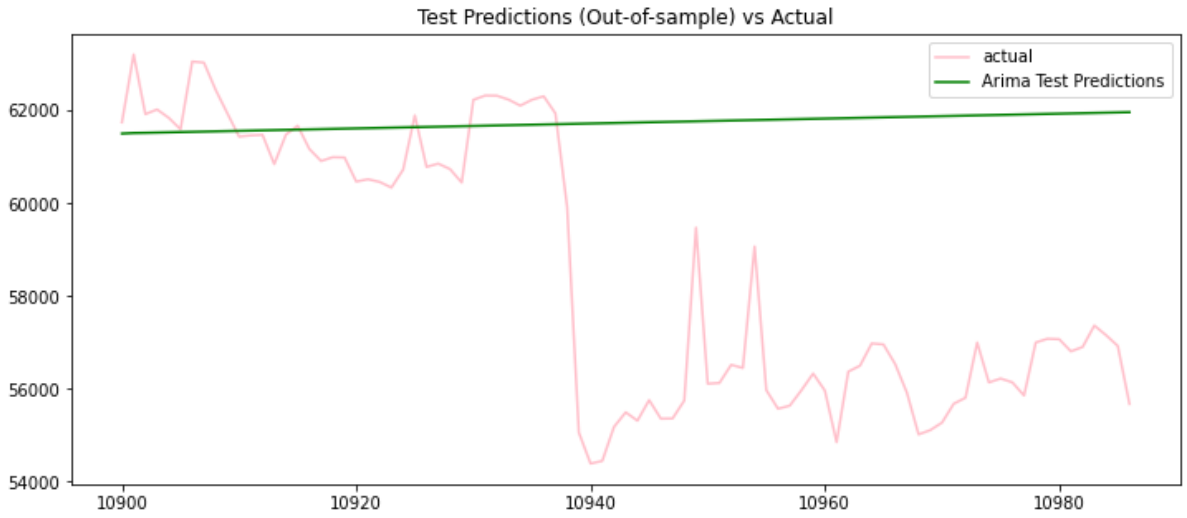

Figure 7: Out-of-sample (test) prediction vs Actual

The Key takeaway from the baseline model: From the figure above, it is apparent that the ARIMA model cannot account for big shocks in the BTC price as there are so many other exogeneous factors that impact price movements beyond the price itself. Because of this, **we can establish a prediction length of not more than one week (24hours x 7 days) for our eventual served model.**

Time Series Forecasting Using DeepAR

Data Preprocessing

The aggregated time series needed to be prepared for DeepAR modeling and forecasting. Firstly, the index of the hourly data was set to a one-hour frequency to ensure every single hour within the period of the data was considered. This step increased the data points to 11,401 records.

According to the DeepAR documentation, DeepAR expects to see input training data in a JSON format, with the following fields:

- **start**: A string that defines the starting date of the time series, with the format 'YYYY-MM-DD HH:MM:SS'.
- **target**: An array of numerical values that represent the time series.
- **cat** (optional): A numerical array of categorical features that can be used to encode the groups that the record belongs to. This is useful for finding models per class of item, such as in retail sales, where you might have {'shoes', 'jackets', 'pants'} encoded as categories {0, 1, 2}.

The input data should be formatted with one time series per line in a JSON file. Each line looks a bit like a dictionary, for example:

{"start":'2007-01-01 00:00:00', "target": [2.54, 6.3, ...], "cat": [1]}

{"start": "2012-01-30 00:00:00", "target": [1.0, -5.0, ...], "cat": [0]}

A json helper function was used to convert each time series to the expected format.

Modeling

As induced from the baseline model, the context and the prediction length were set to 168 (one week) which means one week of hourly data would be used as the context during training. The dataset was uploaded to s3 bucket to enable the DeepAR access the data during training.

The model was trained using 50 epochs, in 2 layers of 50 neurons each using a learning rate of 0.01. The training was concluded in 17 minutes and the next step involved deploying the endpoint for inference.

Inference

The model was deployed using adequate compute resources. It is important to note that a predictor class was packaged in the deployed model to ease the serialization and deserialization of the input data for inference. The predictor class gave us the opportunity to predict on the time series without using any transformation.

The DeepAR produced a **MAE of 8362** which appear to be worse than the baseline model but the advantage of the DeepAR is the ability to set the context of the predictions which is very useful to ensure predictions are as realistic as possible.

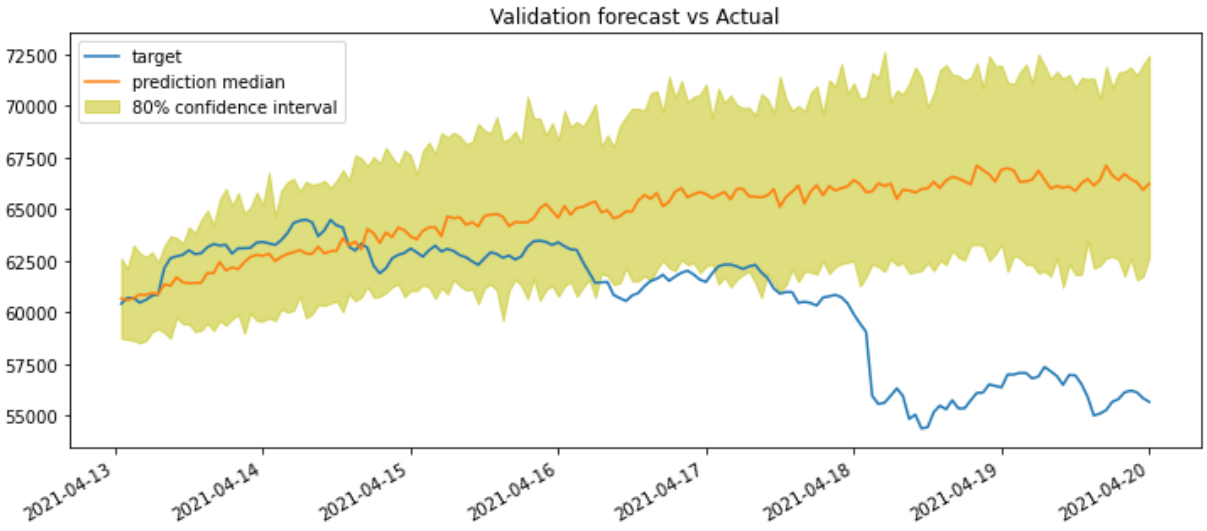

Figure 8: Validation prediction vs Actual (One-week span)

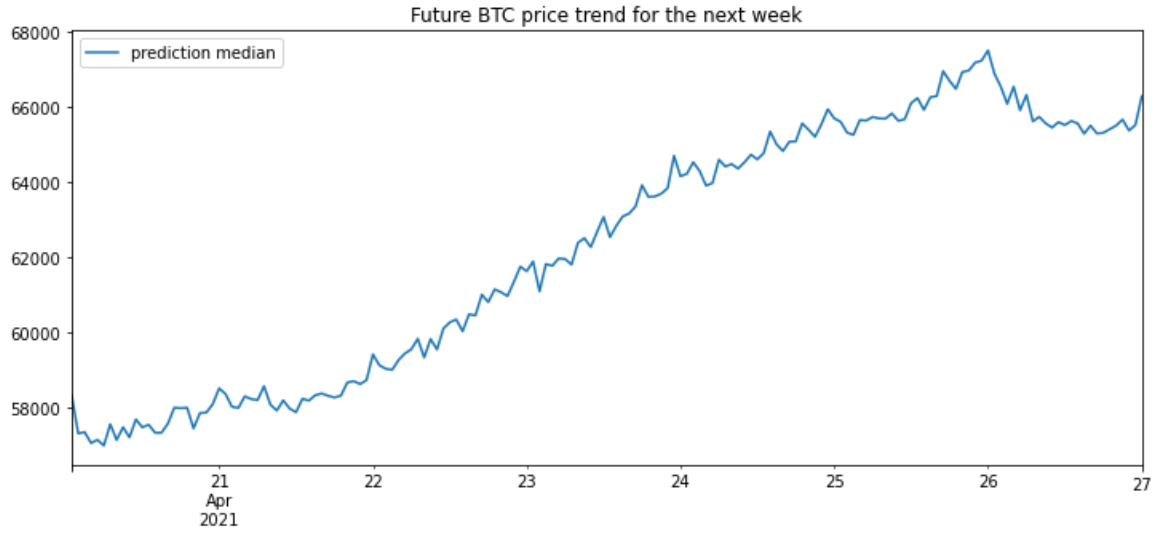

Figure 9: Future BTC price prediction

Text-to-SQL Modeling - Using Pre-trained model

The project wished to use a pretrained model on the custom BTC dataset leveraging Editsql however and as earlier mentioned, the eventual choice of the pre-trained model would be determined by the ease of use, compatibility with AWS sagemaker and the computational requirement. Editsql proved complex to replicate and train on a custom dataset. The solution required troubleshooting and customization in a number of areas but the lack of adequate resources to learn from or community support was an hinderance.

Because of these, a **pretrained huggingface model** developed on text and corresponding sql output as obtained in the Wikisql database was employed for the text to sql translation task in the project. Huggingface hosts thousands of pretrained models and is currently supported by AWS Sagemaker making it ideal. All that was required was to download the model and the tokenizer from huggingface and deploy for predictions. The model is a transformer model called Google T5, fine-tuned to handle text to sql translation. See here for a detailed description of the model.

Model Benchmark

Today, the most advanced text-to-sql model on the Wikisql data has an exact-match accuracy of 84% (https://paperswithcode.com/sota/code-generation-on-wikisql) however, the performance result on the custom BTC dataset is unavailable since the model was not retrained or fine-tuned. In addition, the training data would need time to be developed.

Serving the Models (DeepAR and HuggingFace)

A lambda function was used to handle the data processing and sending of user's requests coming from the Web application through the API Gateway to the correct endpoint for model prediction. Four key functions/steps were necessary in the lambda function:

1. A function that can detect user's intent
2. A function that preprocess the user's data depending on the intent
3. Logic to route the data to the correct endpoint
4. Logic to handle responses from the endpoints

The Web Application

The web application was built using python dash. The major component of the web app was the callback which seamlessly enabled the updating of the displayed graph based on the response from the lambda function through the API gateway.

Challenges & Solutions/Areas of Improvements

In addition to the challenges experienced while using EditSql on the custom dataset, below are the other challenges encountered and possible solutions.

S/N	Challenge	Task	Description	Possible Solution(s)
1	Limited Custom Dataset	Time series	One year and 4 months data was used in the training of the DeepAR model	Use BTC data from 1at least 2010 to improve model prediction.
2	Performance improvement	Time series	The model produced MAE of over 8,000 on the validation set	1. Consider deeper network architectures 2. Consider various hourly data aggregation methods e.g. max, last price etc.
3	Sub-optimal text-to-sql translation	Text-to-sql	While the transformer was able to generate close sql statements in some cases, it could not generate in most cases. For this to be adopted by general public, it needs to generate exact sql queries most of the time.	Consider fine-tuning the t5 model on the custom dataset. The custom data would include questions in natural language and corresponding queries in SQL.
4	Distorted response data	Text-to-sql	Lambda did not return the SQL response text via the API Gateway but returned an	

			unusable form data. This was tested and confirmed in the lambda function but behaved differently via the API GW. This was difficult to trouble shoot and was only solved by transforming the SQL text response to a list of strings and concatenating the strings back to text. This work-around increased model response time.	

Justification/Conclusion

The objectives of this project to enable users understand their data through natural language was achieved partly. Partially because, users are now able to ask for future price predictions of BTC using natural language however, users are not getting the desired responses when basic questions about the BTC are asked which is also evident in the screenshots gotten from the web application.