Exploring Prediction Techniques for November 2022 Inflation Betting on Kalshi

Armaan Bhasin

CMSC320 Fall 2022

I met with Max a few times to discuss this project. He suggested the dataset and helped me focus in on historical markets data. Given how messy the data ended up, he said he would grade this one.

Motivation

Kalshi is the first federally regulated exchange where you can trade on the outcome of events. Think sports betting but much broader. An untapped market founded in 2019! You can bet on economics like inflation rates, politics like Biden's approval rating, and even the weather like global average temperature deviation.

Given the novel nature of Kalshi, there is a lack of analysis on trends in Kalshi event outcomes and profits. Kalshi provides an API for data access and basic trend graphs, but comprehensive analysis is left to users. Successfully detecting events signals can lead to money making opportunities. If you're able to detect that an event will or will not occur, you can get in, bet, and profit more than others.

The purpose of this project is to explore the Kalshi API and dataset and then explore volilitiy, moving average, and bid ask spread as potential markers of event outcome.

Background/Understanding

Now you might be wondering, what does event outcomes betting look like? How does it work?

Event outcomes are binary-- yes or no. Either a clearly defined criteria was met or it wasn't. For example, I will be looking at "What will CPI inflation be in November?". In particular, will inflation rate be >0.2%? Just a few days ago, the Bureau of Labor Statistics annouced that the inflation rate was 0.1%. Thus the market "resolved" to no.

Prices range from $0 to \$1 for an event. These prices lend themselves well to being interpretted as the probability of an event. More so, it represents what people think is the likelihood the event will be yes or no.

You can buy yes or no. If your bet wins, you win $1. If you were incorrect, you win \$0. A price closer to 1 means less profit margin but also more perceived probability.

Trades occur between a maker and a taker. A maker desires a certain price for --the "ask". A taker wants the contract and what they're willing to pay is the "bid".

COVID-19 related inflation surpasses 40 year record!

If you're interested, read more at: https://www.forbes.com/sites/mikepatton/2022/04/13/covid-19-related-inflation-surpasses-40-year-record-how-long-will-it-persist/?sh=1059e6401a82

Inflation betting is the most popular monthly outcome contract on Kalshi and one of the outcome contracts with the highest total trades volume. I chose to explore monthly inflation because it provides enough data points for analysis but does not provide an overwhelming amount of data like annual S&P500 closing value. Given it's relevance to today's economy and established nature on Kalshi, monthly inflation will be the market I study.

Here is the specific outcome market I analyzed: https://kalshi.com/events/CPI-22NOV/markets/CPI-22NOV-T0.2/

Data Collection

Kalshi provides a REST API to access their data. I did not have prior experience with the Kalshi API or APIs in general, so it took me 5 hours getting used to the API and learning how to navigate to the data of interest. Hopefully, the sections below can help you do so with ease!

Please visit: https://trading-api.readme.io/reference/getting-started

Highly reccomended: To learn about API requests, parameters, and return types, go to the drop down on the left under "Kalshi Trade API". To explore parameters, look at "path params" of a request. To learn about the return types, click the expanding arrows for a given "Response" of a request.

If you'd like templates for getting started, Kalshi provides "REST API Starter Code". I reccomend taking a look at "KalshiClientsBaseV2.py" file if you are interested in how the API makes HTTP requests.

If you'd prefer to not use the API and get data directly from Kalshi's website, visit the request (on the website above) you would like to use, input your parameters, and click "Try It!"

I do not cover events, series, or trading data in this tutorial. If you are interesting in using this data for your financial analysis, feel free to read more at the API documentation above. The trading data could be particularly insight for future directions: https://trading-api.readme.io/reference/gettrades

In [1]:
```
from KalshiClientsBaseV2 import ExchangeClient
import time
import json
import uuid
```

Kalshi has both Production and Demo services. Given the introductory scope of this tutorial, I reccomend creating a Demo account. For security purposes, I have replaced my actual email and password in the code below.

In [2]:
```
# To start off, you need to have created an account at https://kalshi.com (Production)
# or an account on the Demo https://demo.kalshi.co/

demo_email = "abhasin33@gmail.com" # change these to be your personal credentials
demo_password = "tevsi7-xadsam-Zovcyv" # (for extra security, recommend using a config f

demo_api_base = "https://demo-api.kalshi.co/trade-api/v2"

## to test in demo
exchange_client = ExchangeClient(exchange_api_base = demo_api_base, email = demo_email,
```

```
# check the exchange status to confirm we are properly connected...
print(exchange_client.get_exchange_status())
```

```
{'exchange_active': True, 'trading_active': False}
```

Great! We've successfully connecetd to the API. Now, let's look at look at some basic API functionality. While I will be doing this example with a particular market in mind, you can explore markets of interest using get_markets().

In [3]:
```
# You can discover markets through the get_markets endpoint...

# and use query parameters to filter your search!
market_params = {'limit':100,
                 'cursor':None, # passing in the cursor from the previous get_markets
                 'event_ticker': None,
                 'series_ticker':None,
                 'max_close_ts':None, # pass in unix_ts
                 'min_close_ts':None, # pass in unix_ts
                 'status':None,
                 'tickers':None}

markets_response = exchange_client.get_markets(**market_params)
cursor = markets_response['cursor']

print('keys:', markets_response.keys())
print()
print('number of objects:', len(markets_response['markets'])) # 100 objects!
print()
print('first market in payload:', markets_response['markets'][0])
print()
print('cursor:', cursor)
```

```
keys: dict_keys(['markets', 'cursor'])

number of objects: 100

first market in payload: {'ticker': 'USDJPY-22DEC1610-138.499', 'event_ticker': 'USDJPY-
22DEC1610', 'subtitle': '138.5 or higher', 'open_time': '2022-12-15T23:25:16Z', 'close_t
ime': '2022-12-16T15:00:00Z', 'expiration_time': '2022-12-23T15:00:00Z', 'status': 'acti
ve', 'yes_bid': 0, 'yes_ask': 0, 'no_bid': 100, 'no_ask': 100, 'last_price': 0, 'previou
s_yes_bid': 0, 'previous_yes_ask': 0, 'previous_price': 0, 'volume': 0, 'volume_24h': 0,
'liquidity': 0, 'open_interest': 0, 'result': '', 'can_close_early': True, 'expiration_v
alue': '', 'category': 'Financials', 'risk_limit_cents': 0}

cursor: CgwIitHnnAYQyNa0lwESGFVTREpQWS0yMkRFQzE0MTgtMTM0LjI1MA
```

At this point, I explored the different marketdatasets availible! If you're interested in analyzing a different dataset from Kalshi you could discover it here.

In [4]:
```
actives = exchange_client.get_markets()
actives
```

Out[4]:
```
{'markets': [{'ticker': 'USDJPY-22DEC1610-138.499',
   'event_ticker': 'USDJPY-22DEC1610',
   'subtitle': '138.5 or higher',
   'open_time': '2022-12-15T23:25:16Z',
   'close_time': '2022-12-16T15:00:00Z',
   'expiration_time': '2022-12-23T15:00:00Z',
   'status': 'active',
   'yes_bid': 0,
   'yes_ask': 0,
   'no_bid': 100,
   'no_ask': 100,
   'last_price': 0,
```

```
 'previous_yes_bid': 0,
 'previous_yes_ask': 0,
 'previous_price': 0,
 'volume': 0,
 'volume_24h': 0,
 'liquidity': 0,
 'open_interest': 0,
 'result': '',
 'can_close_early': True,
 'expiration_value': '',
 'category': 'Financials',
 'risk_limit_cents': 0},
{'ticker': 'USDJPY-22DEC1610-138.375',
 'event_ticker': 'USDJPY-22DEC1610',
 'subtitle': '138.25 to 138.499',
 'open_time': '2022-12-15T23:25:16Z',
 'close_time': '2022-12-16T15:00:00Z',
 'expiration_time': '2022-12-23T15:00:00Z',
 'status': 'active',
 'yes_bid': 0,
 'yes_ask': 0,
 'no_bid': 100,
 'no_ask': 100,
 'last_price': 0,
 'previous_yes_bid': 0,
 'previous_yes_ask': 0,
 'previous_price': 0,
 'volume': 0,
 'volume_24h': 0,
 'liquidity': 0,
 'open_interest': 0,
 'result': '',
 'can_close_early': True,
 'expiration_value': '',
 'category': 'Financials',
 'risk_limit_cents': 0},
{'ticker': 'USDJPY-22DEC1610-138.125',
 'event_ticker': 'USDJPY-22DEC1610',
 'subtitle': '138 to 138.249',
 'open_time': '2022-12-15T23:25:16Z',
 'close_time': '2022-12-16T15:00:00Z',
 'expiration_time': '2022-12-23T15:00:00Z',
 'status': 'active',
 'yes_bid': 0,
 'yes_ask': 0,
 'no_bid': 100,
 'no_ask': 100,
 'last_price': 0,
 'previous_yes_bid': 0,
 'previous_yes_ask': 0,
 'previous_price': 0,
 'volume': 0,
 'volume_24h': 0,
 'liquidity': 0,
 'open_interest': 0,
 'result': '',
 'can_close_early': True,
 'expiration_value': '',
 'category': 'Financials',
 'risk_limit_cents': 0},
{'ticker': 'USDJPY-22DEC1610-137.875',
 'event_ticker': 'USDJPY-22DEC1610',
 'subtitle': '137.75 to 137.999',
 'open_time': '2022-12-15T23:25:16Z',
 'close_time': '2022-12-16T15:00:00Z',
 'expiration_time': '2022-12-23T15:00:00Z',
```

```
 'status': 'active',
 'yes_bid': 0,
 'yes_ask': 0,
 'no_bid': 100,
 'no_ask': 100,
 'last_price': 0,
 'previous_yes_bid': 0,
 'previous_yes_ask': 0,
 'previous_price': 0,
 'volume': 0,
 'volume_24h': 0,
 'liquidity': 0,
 'open_interest': 0,
 'result': '',
 'can_close_early': True,
 'expiration_value': '',
 'category': 'Financials',
 'risk_limit_cents': 0},
{'ticker': 'USDJPY-22DEC1610-137.625',
 'event_ticker': 'USDJPY-22DEC1610',
 'subtitle': '137.5 to 137.749',
 'open_time': '2022-12-15T23:25:16Z',
 'close_time': '2022-12-16T15:00:00Z',
 'expiration_time': '2022-12-23T15:00:00Z',
 'status': 'active',
 'yes_bid': 0,
 'yes_ask': 0,
 'no_bid': 100,
 'no_ask': 100,
 'last_price': 0,
 'previous_yes_bid': 0,
 'previous_yes_ask': 0,
 'previous_price': 0,
 'volume': 0,
 'volume_24h': 0,
 'liquidity': 0,
 'open_interest': 0,
 'result': '',
 'can_close_early': True,
 'expiration_value': '',
 'category': 'Financials',
 'risk_limit_cents': 0},
{'ticker': 'USDJPY-22DEC1610-137.375',
 'event_ticker': 'USDJPY-22DEC1610',
 'subtitle': '137.25 to 137.499',
 'open_time': '2022-12-15T23:25:16Z',
 'close_time': '2022-12-16T15:00:00Z',
 'expiration_time': '2022-12-23T15:00:00Z',
 'status': 'active',
 'yes_bid': 0,
 'yes_ask': 0,
 'no_bid': 100,
 'no_ask': 100,
 'last_price': 0,
 'previous_yes_bid': 0,
 'previous_yes_ask': 0,
 'previous_price': 0,
 'volume': 0,
 'volume_24h': 0,
 'liquidity': 0,
 'open_interest': 0,
 'result': '',
 'can_close_early': True,
 'expiration_value': '',
 'category': 'Financials',
 'risk_limit_cents': 0},
```

```
{'ticker': 'USDJPY-22DEC1610-137.250',
 'event_ticker': 'USDJPY-22DEC1610',
 'subtitle': '137.249 or lower',
 'open_time': '2022-12-15T23:25:16Z',
 'close_time': '2022-12-16T15:00:00Z',
 'expiration_time': '2022-12-23T15:00:00Z',
 'status': 'active',
 'yes_bid': 0,
 'yes_ask': 0,
 'no_bid': 100,
 'no_ask': 100,
 'last_price': 0,
 'previous_yes_bid': 0,
 'previous_yes_ask': 0,
 'previous_price': 0,
 'volume': 0,
 'volume_24h': 0,
 'liquidity': 0,
 'open_interest': 0,
 'result': '',
 'can_close_early': True,
 'expiration_value': '',
 'category': 'Financials',
 'risk_limit_cents': 0},
{'ticker': 'EURUSD-22DEC1610-1.06799',
 'event_ticker': 'EURUSD-22DEC1610',
 'subtitle': '1.068 or higher',
 'open_time': '2022-12-15T23:25:16Z',
 'close_time': '2022-12-16T15:00:00Z',
 'expiration_time': '2022-12-23T15:00:00Z',
 'status': 'active',
 'yes_bid': 0,
 'yes_ask': 0,
 'no_bid': 100,
 'no_ask': 100,
 'last_price': 0,
 'previous_yes_bid': 0,
 'previous_yes_ask': 0,
 'previous_price': 0,
 'volume': 0,
 'volume_24h': 0,
 'liquidity': 0,
 'open_interest': 0,
 'result': '',
 'can_close_early': True,
 'expiration_value': '',
 'category': 'Financials',
 'risk_limit_cents': 0},
{'ticker': 'EURUSD-22DEC1610-1.067',
 'event_ticker': 'EURUSD-22DEC1610',
 'subtitle': '1.066 to 1.06799',
 'open_time': '2022-12-15T23:25:16Z',
 'close_time': '2022-12-16T15:00:00Z',
 'expiration_time': '2022-12-23T15:00:00Z',
 'status': 'active',
 'yes_bid': 0,
 'yes_ask': 0,
 'no_bid': 100,
 'no_ask': 100,
 'last_price': 0,
 'previous_yes_bid': 0,
 'previous_yes_ask': 0,
 'previous_price': 0,
 'volume': 0,
 'volume_24h': 0,
 'liquidity': 0,
```

```
 'open_interest': 0,
 'result': '',
 'can_close_early': True,
 'expiration_value': '',
 'category': 'Financials',
 'risk_limit_cents': 0},
{'ticker': 'EURUSD-22DEC1610-1.065',
 'event_ticker': 'EURUSD-22DEC1610',
 'subtitle': '1.064 to 1.06599',
 'open_time': '2022-12-15T23:25:16Z',
 'close_time': '2022-12-16T15:00:00Z',
 'expiration_time': '2022-12-23T15:00:00Z',
 'status': 'active',
 'yes_bid': 0,
 'yes_ask': 0,
 'no_bid': 100,
 'no_ask': 100,
 'last_price': 0,
 'previous_yes_bid': 0,
 'previous_yes_ask': 0,
 'previous_price': 0,
 'volume': 0,
 'volume_24h': 0,
 'liquidity': 0,
 'open_interest': 0,
 'result': '',
 'can_close_early': True,
 'expiration_value': '',
 'category': 'Financials',
 'risk_limit_cents': 0},
{'ticker': 'EURUSD-22DEC1610-1.063',
 'event_ticker': 'EURUSD-22DEC1610',
 'subtitle': '1.062 to 1.06399',
 'open_time': '2022-12-15T23:25:16Z',
 'close_time': '2022-12-16T15:00:00Z',
 'expiration_time': '2022-12-23T15:00:00Z',
 'status': 'active',
 'yes_bid': 0,
 'yes_ask': 0,
 'no_bid': 100,
 'no_ask': 100,
 'last_price': 0,
 'previous_yes_bid': 0,
 'previous_yes_ask': 0,
 'previous_price': 0,
 'volume': 0,
 'volume_24h': 0,
 'liquidity': 0,
 'open_interest': 0,
 'result': '',
 'can_close_early': True,
 'expiration_value': '',
 'category': 'Financials',
 'risk_limit_cents': 0},
{'ticker': 'EURUSD-22DEC1610-1.061',
 'event_ticker': 'EURUSD-22DEC1610',
 'subtitle': '1.06 to 1.06199',
 'open_time': '2022-12-15T23:25:16Z',
 'close_time': '2022-12-16T15:00:00Z',
 'expiration_time': '2022-12-23T15:00:00Z',
 'status': 'active',
 'yes_bid': 0,
 'yes_ask': 0,
 'no_bid': 100,
 'no_ask': 100,
 'last_price': 0,
```

```
 'previous_yes_bid': 0,
 'previous_yes_ask': 0,
 'previous_price': 0,
 'volume': 0,
 'volume_24h': 0,
 'liquidity': 0,
 'open_interest': 0,
 'result': '',
 'can_close_early': True,
 'expiration_value': '',
 'category': 'Financials',
 'risk_limit_cents': 0},
{'ticker': 'EURUSD-22DEC1610-1.059',
 'event_ticker': 'EURUSD-22DEC1610',
 'subtitle': '1.058 to 1.05999',
 'open_time': '2022-12-15T23:25:16Z',
 'close_time': '2022-12-16T15:00:00Z',
 'expiration_time': '2022-12-23T15:00:00Z',
 'status': 'active',
 'yes_bid': 0,
 'yes_ask': 0,
 'no_bid': 100,
 'no_ask': 100,
 'last_price': 0,
 'previous_yes_bid': 0,
 'previous_yes_ask': 0,
 'previous_price': 0,
 'volume': 0,
 'volume_24h': 0,
 'liquidity': 0,
 'open_interest': 0,
 'result': '',
 'can_close_early': True,
 'expiration_value': '',
 'category': 'Financials',
 'risk_limit_cents': 0},
{'ticker': 'EURUSD-22DEC1610-1.05800',
 'event_ticker': 'EURUSD-22DEC1610',
 'subtitle': '1.05799 or lower',
 'open_time': '2022-12-15T23:25:16Z',
 'close_time': '2022-12-16T15:00:00Z',
 'expiration_time': '2022-12-23T15:00:00Z',
 'status': 'active',
 'yes_bid': 0,
 'yes_ask': 0,
 'no_bid': 100,
 'no_ask': 100,
 'last_price': 0,
 'previous_yes_bid': 0,
 'previous_yes_ask': 0,
 'previous_price': 0,
 'volume': 0,
 'volume_24h': 0,
 'liquidity': 0,
 'open_interest': 0,
 'result': '',
 'can_close_early': True,
 'expiration_value': '',
 'category': 'Financials',
 'risk_limit_cents': 0},
{'ticker': 'USDJPY-22DEC1518-136.999',
 'event_ticker': 'USDJPY-22DEC1518',
 'subtitle': '137 or higher',
 'open_time': '2022-12-15T15:25:17Z',
 'close_time': '2022-12-15T23:00:00Z',
 'expiration_time': '2022-12-22T23:00:00Z',
```

```
 'status': 'closed',
 'yes_bid': 0,
 'yes_ask': 0,
 'no_bid': 100,
 'no_ask': 100,
 'last_price': 0,
 'previous_yes_bid': 0,
 'previous_yes_ask': 0,
 'previous_price': 0,
 'volume': 0,
 'volume_24h': 0,
 'liquidity': 0,
 'open_interest': 0,
 'result': '',
 'can_close_early': True,
 'expiration_value': '',
 'category': 'Financials',
 'risk_limit_cents': 0},
{'ticker': 'USDJPY-22DEC1518-136.875',
 'event_ticker': 'USDJPY-22DEC1518',
 'subtitle': '136.75 to 136.999',
 'open_time': '2022-12-15T15:25:17Z',
 'close_time': '2022-12-15T23:00:00Z',
 'expiration_time': '2022-12-22T23:00:00Z',
 'status': 'closed',
 'yes_bid': 0,
 'yes_ask': 0,
 'no_bid': 100,
 'no_ask': 100,
 'last_price': 0,
 'previous_yes_bid': 0,
 'previous_yes_ask': 0,
 'previous_price': 0,
 'volume': 0,
 'volume_24h': 0,
 'liquidity': 0,
 'open_interest': 0,
 'result': '',
 'can_close_early': True,
 'expiration_value': '',
 'category': 'Financials',
 'risk_limit_cents': 0},
{'ticker': 'USDJPY-22DEC1518-136.625',
 'event_ticker': 'USDJPY-22DEC1518',
 'subtitle': '136.5 to 136.749',
 'open_time': '2022-12-15T15:25:17Z',
 'close_time': '2022-12-15T23:00:00Z',
 'expiration_time': '2022-12-22T23:00:00Z',
 'status': 'closed',
 'yes_bid': 0,
 'yes_ask': 0,
 'no_bid': 100,
 'no_ask': 100,
 'last_price': 0,
 'previous_yes_bid': 0,
 'previous_yes_ask': 0,
 'previous_price': 0,
 'volume': 0,
 'volume_24h': 0,
 'liquidity': 0,
 'open_interest': 0,
 'result': '',
 'can_close_early': True,
 'expiration_value': '',
 'category': 'Financials',
 'risk_limit_cents': 0},
```

```
{'ticker': 'USDJPY-22DEC1518-136.375',
 'event_ticker': 'USDJPY-22DEC1518',
 'subtitle': '136.25 to 136.499',
 'open_time': '2022-12-15T15:25:17Z',
 'close_time': '2022-12-15T23:00:00Z',
 'expiration_time': '2022-12-22T23:00:00Z',
 'status': 'closed',
 'yes_bid': 0,
 'yes_ask': 0,
 'no_bid': 100,
 'no_ask': 100,
 'last_price': 0,
 'previous_yes_bid': 0,
 'previous_yes_ask': 0,
 'previous_price': 0,
 'volume': 0,
 'volume_24h': 0,
 'liquidity': 0,
 'open_interest': 0,
 'result': '',
 'can_close_early': True,
 'expiration_value': '',
 'category': 'Financials',
 'risk_limit_cents': 0},
{'ticker': 'USDJPY-22DEC1518-136.125',
 'event_ticker': 'USDJPY-22DEC1518',
 'subtitle': '136 to 136.249',
 'open_time': '2022-12-15T15:25:17Z',
 'close_time': '2022-12-15T23:00:00Z',
 'expiration_time': '2022-12-22T23:00:00Z',
 'status': 'closed',
 'yes_bid': 0,
 'yes_ask': 0,
 'no_bid': 100,
 'no_ask': 100,
 'last_price': 0,
 'previous_yes_bid': 0,
 'previous_yes_ask': 0,
 'previous_price': 0,
 'volume': 0,
 'volume_24h': 0,
 'liquidity': 0,
 'open_interest': 0,
 'result': '',
 'can_close_early': True,
 'expiration_value': '',
 'category': 'Financials',
 'risk_limit_cents': 0},
{'ticker': 'USDJPY-22DEC1518-135.875',
 'event_ticker': 'USDJPY-22DEC1518',
 'subtitle': '135.75 to 135.999',
 'open_time': '2022-12-15T15:25:17Z',
 'close_time': '2022-12-15T23:00:00Z',
 'expiration_time': '2022-12-22T23:00:00Z',
 'status': 'closed',
 'yes_bid': 0,
 'yes_ask': 0,
 'no_bid': 100,
 'no_ask': 100,
 'last_price': 0,
 'previous_yes_bid': 0,
 'previous_yes_ask': 0,
 'previous_price': 0,
 'volume': 0,
 'volume_24h': 0,
 'liquidity': 0,
```

```
   'open_interest': 0,
   'result': '',
   'can_close_early': True,
   'expiration_value': '',
   'category': 'Financials',
   'risk_limit_cents': 0}],
 'cursor': 'CgwIivTsnAYQsPvw2wMSGFVTREpQWS0yMkRFQzE1MTgtMTM1Ljg3NQ'}
```

An important part of the using the Kalshi API request methods is cursors.

The Cursor represents a pointer to the next page of records in the pagination. By default, Kalshi API get results from the first page of results. Providing a cursor allows us to access the next page of results. The cursor is returned as a string in the previous request.

Please do take care to use cursors because otherwise you may get duplicated data of the first page of results when you want multiple pages of data! I made this mistake at first, so be mindful.

In [5]:
```
market_params = {'limit':100,
                 'cursor':cursor, # passing in the cursor from the previous get_marke
                 'event_ticker': None,
                 'series_ticker': None,
                 'max_close_ts': None, # pass in unix_ts
                 'min_close_ts': None, # pass in unix_ts
                 'status': None,
                 'tickers':None}

markets_response2 = exchange_client.get_markets(**market_params)
cursor = markets_response['cursor']

print('keys:', markets_response.keys())
print()
print('number of objects:', len(markets_response['markets'])) # 100 objects!
print()
print('first market in market_response payload:', markets_response['markets'][0]) # new
print()
print('new cursor!', cursor)
```

```
keys: dict_keys(['markets', 'cursor'])

number of objects: 100

first market in market_response payload: {'ticker': 'USDJPY-22DEC1610-138.499', 'event_t
icker': 'USDJPY-22DEC1610', 'subtitle': '138.5 or higher', 'open_time': '2022-12-15T23:2
5:16Z', 'close_time': '2022-12-16T15:00:00Z', 'expiration_time': '2022-12-23T15:00:00Z',
'status': 'active', 'yes_bid': 0, 'yes_ask': 0, 'no_bid': 100, 'no_ask': 100, 'last_pric
e': 0, 'previous_yes_bid': 0, 'previous_yes_ask': 0, 'previous_price': 0, 'volume': 0,
'volume_24h': 0, 'liquidity': 0, 'open_interest': 0, 'result': '', 'can_close_early': Tr
ue, 'expiration_value': '', 'category': 'Financials', 'risk_limit_cents': 0}

new cursor! CgwIitHnnAYQyNa0lwESGFVTREpQWS0yMkRFQzE0MTgtMTM0LjI1MA
```

Confusing timestamp format alert!

Please note that timestamps ('max_ts', 'min_ts', 'ts') are in Unix Epoch format. This was a new time format for me, so let me provide some background. Unix Epoch time represents the number of seconds that have elapsed since 00:00:00 UTC on 1 January 1970. However, we can easily change this to a more interpretable datatime later.

Below, I attempted to look for recent events. 1000000 seconds is roughly 11 days. Remember how I mentioned the cursor is important. Well since the first page of results does not contain the 11 days of most recent data, no results were returned.

In [6]:
```
ticker = 'CPI-22NOV-T0.2'

market_history_params = {'ticker': ticker,
                         'limit': 100,
                         'cursor': None,
                         'max_ts': None, # pass in unix_ts
                         'min_ts': round(time.time()-1000000) # passing a recent unix
                             }
market_history_response = exchange_client.get_market_history(**market_history_params)

print('keys:', market_history_response.keys())
print()
print('most recent market history object:', market_history_response['history'])
print()
```

```
keys: dict_keys(['ticker', 'history', 'cursor'])

most recent market history object: []
```

Again, the dataset I'm interested in is What will CPI inflation be in November? (ticker = 'CPI-22NOV-T0.2')

In [7]:
```
mkt = exchange_client.get_market_history('CPI-22NOV-T0.2')
mkt
```

Out[7]:

```
{'ticker': 'CPI-22NOV-T0.2',
 'history': [{'yes_price': 0,
   'yes_bid': 1,
   'yes_ask': 100,
   'no_bid': 0,
   'no_ask': 99,
   'volume': 0,
   'open_interest': 0,
   'ts': 1653520255},
  {'yes_price': 0,
   'yes_bid': 2,
   'yes_ask': 100,
   'no_bid': 0,
   'no_ask': 98,
   'volume': 0,
   'open_interest': 0,
   'ts': 1653520255},
  {'yes_price': 0,
   'yes_bid': 3,
   'yes_ask': 100,
   'no_bid': 0,
   'no_ask': 97,
   'volume': 0,
   'open_interest': 0,
   'ts': 1653520255},
  {'yes_price': 0,
   'yes_bid': 6,
   'yes_ask': 100,
   'no_bid': 0,
   'no_ask': 94,
   'volume': 0,
   'open_interest': 0,
   'ts': 1653520255},
  {'yes_price': 0,
   'yes_bid': 12,
   'yes_ask': 100,
   'no_bid': 0,
   'no_ask': 88,
```

```
 'volume': 0,
 'open_interest': 0,
 'ts': 1653520255},
{'yes_price': 0,
 'yes_bid': 22,
 'yes_ask': 100,
 'no_bid': 0,
 'no_ask': 78,
 'volume': 0,
 'open_interest': 0,
 'ts': 1653520255},
{'yes_price': 0,
 'yes_bid': 33,
 'yes_ask': 100,
 'no_bid': 0,
 'no_ask': 67,
 'volume': 0,
 'open_interest': 0,
 'ts': 1653520255},
{'yes_price': 0,
 'yes_bid': 38,
 'yes_ask': 100,
 'no_bid': 0,
 'no_ask': 62,
 'volume': 0,
 'open_interest': 0,
 'ts': 1653520255},
{'yes_price': 0,
 'yes_bid': 39,
 'yes_ask': 100,
 'no_bid': 0,
 'no_ask': 61,
 'volume': 0,
 'open_interest': 0,
 'ts': 1653520255},
{'yes_price': 0,
 'yes_bid': 40,
 'yes_ask': 100,
 'no_bid': 0,
 'no_ask': 60,
 'volume': 0,
 'open_interest': 0,
 'ts': 1653520255},
{'yes_price': 0,
 'yes_bid': 41,
 'yes_ask': 100,
 'no_bid': 0,
 'no_ask': 59,
 'volume': 0,
 'open_interest': 0,
 'ts': 1653520256},
{'yes_price': 0,
 'yes_bid': 42,
 'yes_ask': 100,
 'no_bid': 0,
 'no_ask': 58,
 'volume': 0,
 'open_interest': 0,
 'ts': 1653520256},
{'yes_price': 0,
 'yes_bid': 43,
 'yes_ask': 100,
 'no_bid': 0,
 'no_ask': 57,
 'volume': 0,
 'open_interest': 0,
```

```
  'ts': 1653520256},
 {'yes_price': 0,
  'yes_bid': 50,
  'yes_ask': 100,
  'no_bid': 0,
  'no_ask': 50,
  'volume': 0,
  'open_interest': 0,
  'ts': 1653520256},
 {'yes_price': 0,
  'yes_bid': 51,
  'yes_ask': 100,
  'no_bid': 0,
  'no_ask': 49,
  'volume': 0,
  'open_interest': 0,
  'ts': 1653520256},
 {'yes_price': 0,
  'yes_bid': 52,
  'yes_ask': 100,
  'no_bid': 0,
  'no_ask': 48,
  'volume': 0,
  'open_interest': 0,
  'ts': 1653520256},
 {'yes_price': 0,
  'yes_bid': 52,
  'yes_ask': 99,
  'no_bid': 1,
  'no_ask': 48,
  'volume': 0,
  'open_interest': 0,
  'ts': 1653520256},
 {'yes_price': 0,
  'yes_bid': 52,
  'yes_ask': 98,
  'no_bid': 2,
  'no_ask': 48,
  'volume': 0,
  'open_interest': 0,
  'ts': 1653520256},
 {'yes_price': 0,
  'yes_bid': 52,
  'yes_ask': 97,
  'no_bid': 3,
  'no_ask': 48,
  'volume': 0,
  'open_interest': 0,
  'ts': 1653520256},
 {'yes_price': 0,
  'yes_bid': 52,
  'yes_ask': 79,
  'no_bid': 21,
  'no_ask': 48,
  'volume': 0,
  'open_interest': 0,
  'ts': 1653520256}],
'cursor': 'CgwIgO-6lAYQuJ6wggM'}
```


In [8]:
```
import pandas as pd
import matplotlib.pyplot as plt
```

Let's take a preliminary look at the data! It looks like the first page of data starts in May 2022. I converted the returned dictionary to a dataframe for easier analysis. I also changed the Unix time to

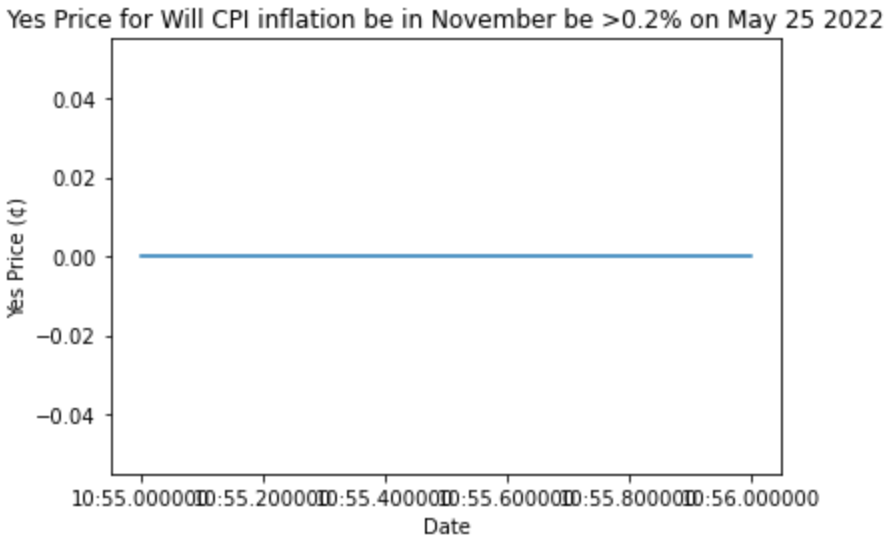
standard datetime.

In [9]:
```
initial_df = pd.DataFrame.from_dict(mkt['history'])

# Unix time to datatime
initial_df['ts'] = pd.to_datetime(initial_df["ts"], unit="s")

initial_df
```

Out[9]:

	yes_price	yes_bid	yes_ask	no_bid	no_ask	volume	open_interest	ts
0	0	1	100	0	99	0	0	2022-05-25 23:10:55
1	0	2	100	0	98	0	0	2022-05-25 23:10:55
2	0	3	100	0	97	0	0	2022-05-25 23:10:55
3	0	6	100	0	94	0	0	2022-05-25 23:10:55
4	0	12	100	0	88	0	0	2022-05-25 23:10:55
5	0	22	100	0	78	0	0	2022-05-25 23:10:55
6	0	33	100	0	67	0	0	2022-05-25 23:10:55
7	0	38	100	0	62	0	0	2022-05-25 23:10:55
8	0	39	100	0	61	0	0	2022-05-25 23:10:55
9	0	40	100	0	60	0	0	2022-05-25 23:10:55
10	0	41	100	0	59	0	0	2022-05-25 23:10:56
11	0	42	100	0	58	0	0	2022-05-25 23:10:56
12	0	43	100	0	57	0	0	2022-05-25 23:10:56
13	0	50	100	0	50	0	0	2022-05-25 23:10:56
14	0	51	100	0	49	0	0	2022-05-25 23:10:56
15	0	52	100	0	48	0	0	2022-05-25 23:10:56
16	0	52	99	1	48	0	0	2022-05-25 23:10:56
17	0	52	98	2	48	0	0	2022-05-25 23:10:56
18	0	52	97	3	48	0	0	2022-05-25 23:10:56
19	0	52	79	21	48	0	0	2022-05-25 23:10:56

In [10]:
```
plt.plot(initial_df['ts'], initial_df['yes_price'])
plt.xlabel("Date")
plt.ylabel("Yes Price (¢)")
plt.title("Yes Price for Will CPI inflation be in November be >0.2% on May 25 2022")
plt.show()
```



The plot above illustrates that 20 data points is not enough to observe the movement of the 'yes_price'. Let's get some more data!

Data Management/Representation

Below I generate a dataframe with 5000 market history entries for inflation in Nov 2022. Notice how I have to use the prevCursor to access the next page of results. The results start in May 2022 and go through October 2022.

I recognize this only represents a small portion of the data. I decided to limit my data to this time period and number of points because the API requests were taking too long and there were too many concentrated data points, especially in October. If you have the computing power and storage to make requests and locally store the data, I reccomend increasing the 50 in the for loop below.

In [18]:
```
market = exchange_client.get_market_history(ticker='CPI-22NOV-T0.2', limit=100)
df = pd.DataFrame(market['history'])
prevCursor = market['cursor']

for i in range(1,50):
    market = exchange_client.get_market_history(ticker='CPI-22NOV-T0.2', cursor=prevCurs
    prevCursor = market['cursor']
    df_tmp = pd.DataFrame(market['history'])
    df = pd.concat([df, df_tmp])

df
```

Out[18]:

	yes_price	yes_bid	yes_ask	no_bid	no_ask	volume	open_interest	ts
0	0	1	100	0	99	0	0	1653520255
1	0	2	100	0	98	0	0	1653520255
2	0	3	100	0	97	0	0	1653520255
3	0	6	100	0	94	0	0	1653520255
4	0	12	100	0	88	0	0	1653520255
...	...	...	...	...	...	...	...	...
95	91	66	73	27	34	885	460	1666095247
96	91	66	72	28	34	885	460	1666095247
97	91	66	70	30	34	885	460	1666095247

98	91	66	69	31	34	885	460	1666095247
99	91	66	70	30	34	885	460	1666095250

5000 rows × 8 columns

In [20]:
```
# Removing duplicate rows
df.drop_duplicates(inplace=True)
```

In [21]:
```
# from Unix Epoch to datetime format
df['ts'] = pd.to_datetime(df['ts'], unit="s")
df.reset_index(inplace=True)
df
```

Out[21]:

	index	yes_price	yes_bid	yes_ask	no_bid	no_ask	volume	open_interest	ts
0	0	0	1	100	0	99	0	0	2022-05-25 23:10:55
1	1	0	2	100	0	98	0	0	2022-05-25 23:10:55
2	2	0	3	100	0	97	0	0	2022-05-25 23:10:55
3	3	0	6	100	0	94	0	0	2022-05-25 23:10:55
4	4	0	12	100	0	88	0	0	2022-05-25 23:10:55
...	...	...	...	...	...	...	...	...	...
4340	95	91	66	73	27	34	885	460	2022-10-18 12:14:07
4341	96	91	66	72	28	34	885	460	2022-10-18 12:14:07
4342	97	91	66	70	30	34	885	460	2022-10-18 12:14:07
4343	98	91	66	69	31	34	885	460	2022-10-18 12:14:07
4344	99	91	66	70	30	34	885	460	2022-10-18 12:14:10

4345 rows × 9 columns

Historical CPI data will serve as the primary signal/independent variable

CPI, All Urban Consumers Less food and energy, seasonally adjusted, inflation data (from BLS, Bureau of Labor Statistics)

Data can be found at: https://data.bls.gov/timeseries/CUSR0000SA0L1E&output_view=pct_1mth

There is no data for Dec, as it has not yet ben released. November data was actually only released on December 13.

In [22]:
```
cpi = pd.read_excel("cpidata.xlsx")
cpi.drop(['HALF1', 'HALF2'], axis=1, inplace=True)
cpi
```

Out[22]:

	Year	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
0	2012	0.2	0.1	0.2	0.2	0.1	0.2	0.2	0.1	0.2	0.2	0.1	0.2
1	2013	0.2	0.1	0.1	0.0	0.1	0.2	0.2	0.2	0.2	0.1	0.2	0.2
2	2014	0.1	0.1	0.2	0.2	0.2	0.1	0.2	0.1	0.2	0.2	0.1	0.1
3	2015	0.1	0.2	0.2	0.2	0.1	0.2	0.2	0.1	0.2	0.2	0.2	0.1
4	2016	0.2	0.2	0.2	0.3	0.2	0.2	0.1	0.2	0.2	0.1	0.1	0.2
5	2017	0.2	0.2	0.0	0.1	0.1	0.1	0.1	0.2	0.1	0.3	0.1	0.2
6	2018	0.3	0.2	0.2	0.2	0.2	0.1	0.2	0.1	0.2	0.2	0.2	0.2
7	2019	0.2	0.1	0.2	0.2	0.1	0.2	0.2	0.2	0.2	0.2	0.2	0.1
8	2020	0.3	0.2	-0.1	-0.4	-0.1	0.2	0.6	0.4	0.2	0.1	0.2	0.1
9	2021	0.0	0.2	0.3	0.9	0.7	0.8	0.3	0.2	0.3	0.6	0.5	0.6
10	2022	0.6	0.5	0.3	0.6	0.6	0.7	0.3	0.6	0.6	0.3	0.1	NaN

Data Exploration

First of all, let's get an idea of our data big picture!

In [23]:
```
df.describe()
```

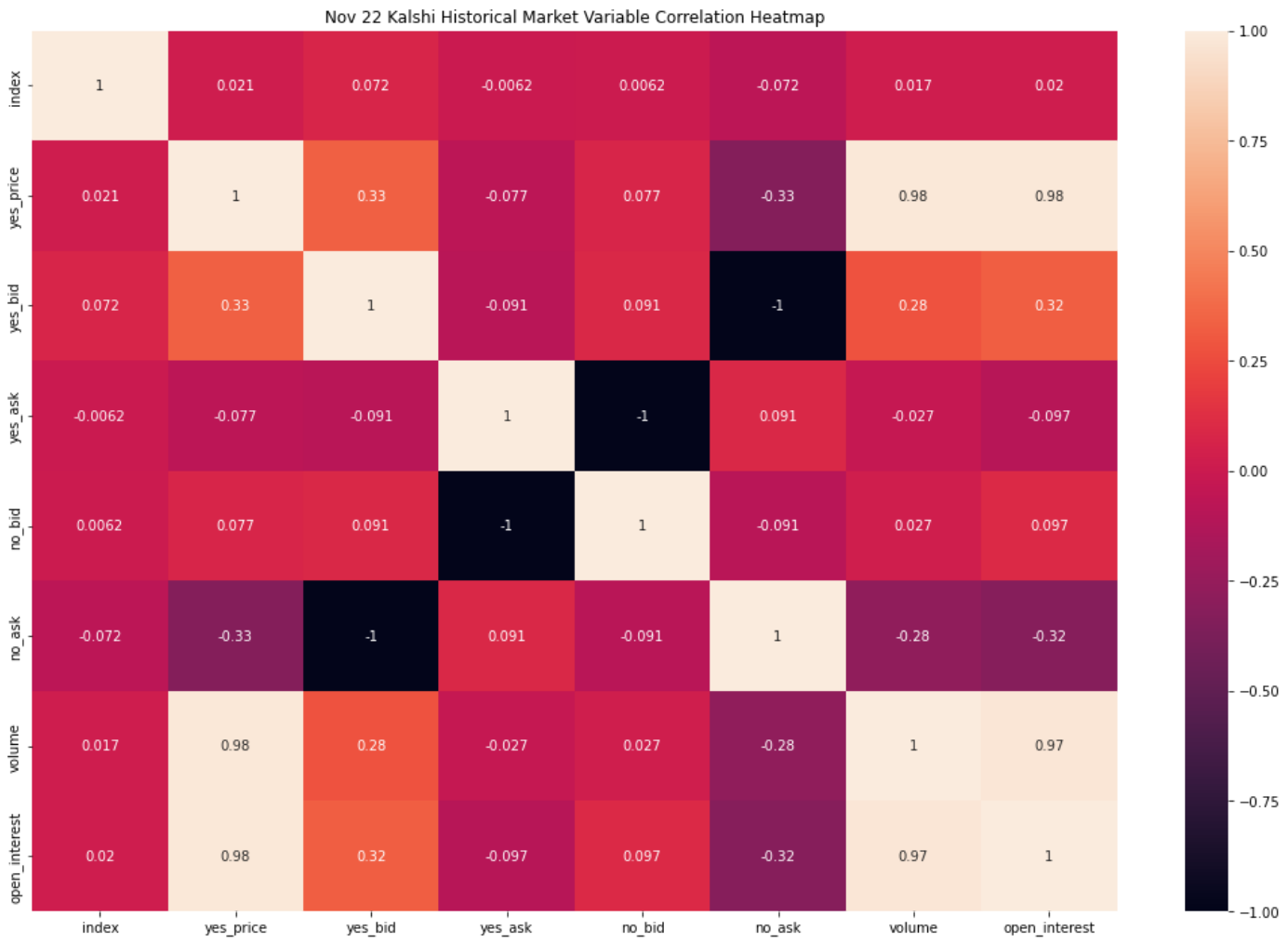
Out[23]:

	index	yes_price	yes_bid	yes_ask	no_bid	no_ask	volume	c
count	4345.000000	4345.000000	4345.000000	4345.000000	4345.000000	4345.000000	4345.000000	
mean	49.848562	86.516916	51.179056	84.700806	15.299194	48.820944	836.427158	
std	28.725812	18.734460	6.602093	10.632431	10.632431	6.602093	192.733505	
min	0.000000	0.000000	0.000000	56.000000	0.000000	27.000000	0.000000	
25%	25.000000	91.000000	50.000000	77.000000	3.000000	50.000000	885.000000	
50%	51.000000	91.000000	50.000000	82.000000	18.000000	50.000000	885.000000	
75%	75.000000	91.000000	50.000000	97.000000	23.000000	50.000000	885.000000	
max	99.000000	91.000000	73.000000	100.000000	44.000000	100.000000	885.000000	

It looks like yes_price doesn't tell us much since it mostly stays at 0.91.

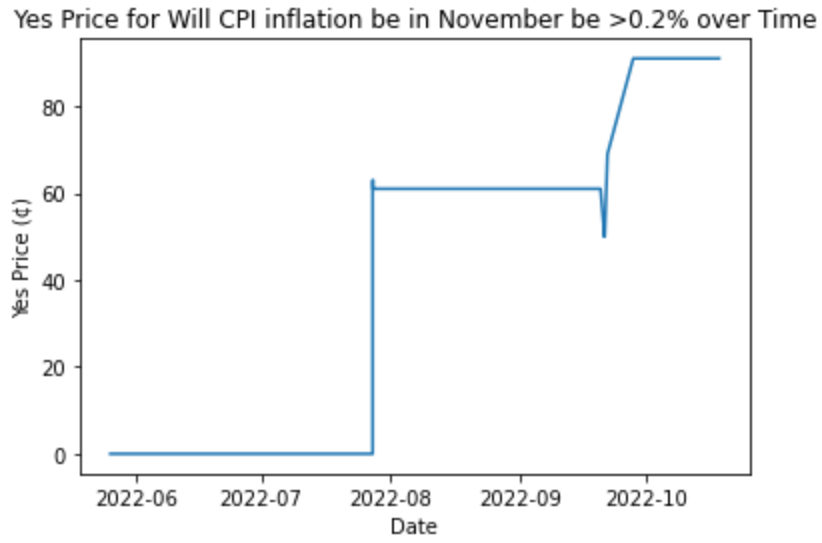
Let's look at correlations between variables!

In [24]:
```
import seaborn as sns

corr = df.corr()

plt.figure(figsize=(18,12))

ax = plt.axes()
sns.heatmap(corr, annot=True)

ax.set_title('Nov 22 Kalshi Historical Market Variable Correlation Heatmap')
plt.show()
```



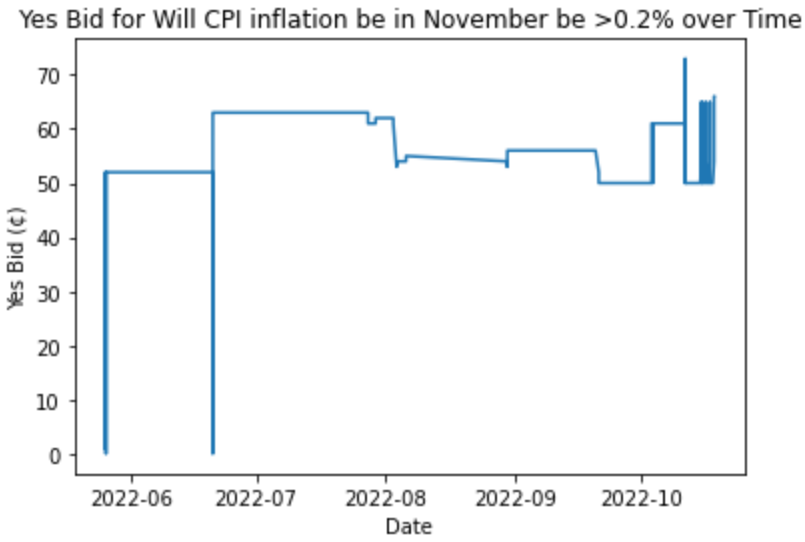
We can observe the properties of the Kalshi contracts structure. Yes and No contracts are negatively correlated as expected. Most other variables are correlated though.

In [25]:
```
plt.plot(df["ts"],df['yes_price'])
plt.xlabel("Date")
plt.ylabel("Yes Price (¢)")
plt.title("Yes Price for Will CPI inflation be in November be >0.2% over Time")
plt.show()
```

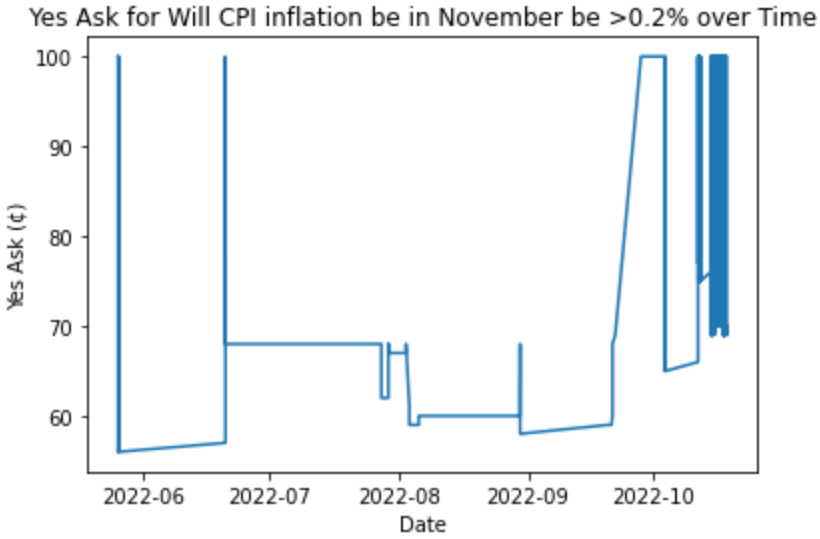


In [26]:
```
plt.plot(df["ts"],df['yes_bid'])
plt.xlabel("Date")
plt.ylabel("Yes Bid (¢)")
```

```
plt.title("Yes Bid for Will CPI inflation be in November be >0.2% over Time")
plt.show()
```

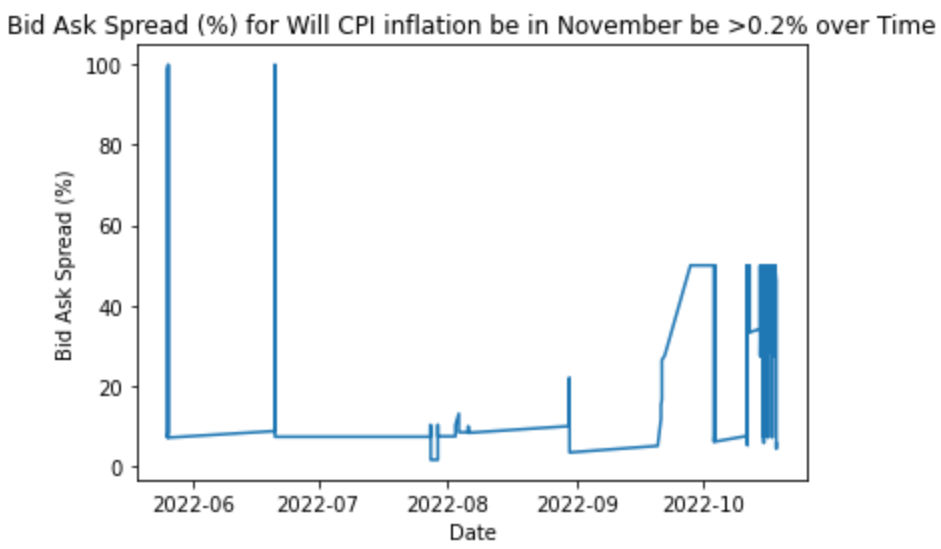


In [27]:
```
plt.plot(df["ts"],df['yes_ask'])
plt.xlabel("Date")
plt.ylabel("Yes Ask (¢)")
plt.title("Yes Ask for Will CPI inflation be in November be >0.2% over Time")
plt.show()
```



There is a lot of volatility as the date approaches November. This can be observed towards the end of October in every month. This may negatively impact results if the selected model is selective to noise.

In [28]:
```
df['bid_ask_spread'] = df['yes_ask'] - df['yes_bid']
df['BAS_pct'] = 100 * df['bid_ask_spread'] / df['yes_ask']
plt.plot(df["ts"],df['BAS_pct'])
plt.xlabel("Date")
plt.ylabel("Bid Ask Spread (%)")
plt.title("Bid Ask Spread (%) for Will CPI inflation be in November be >0.2% over Time")
plt.show()
```



Bid Ask Spread is 100% * (yes_ask - yes_bid) / yes_ask

The bid ask spread represents the amount that the ask price exceeds the bid price. It is the transaction cost, excluding commissions.

For example, a median bid ask spread means that, the lowest a seller "asks" for is 33 cents away from the higher a buyer "bids" for. Thus, either the median seller has to accept losing 33 cents, or the median buyer has to be willing to pay an extra 33 cents.

Bid-ask spread is an important financial metric because it represents the liquidity of an asset. The liquidity of an asset means how quickly can be turn it into cash. The most liquid asset is of course currency. It is very quick to change from USD to EUR for example.

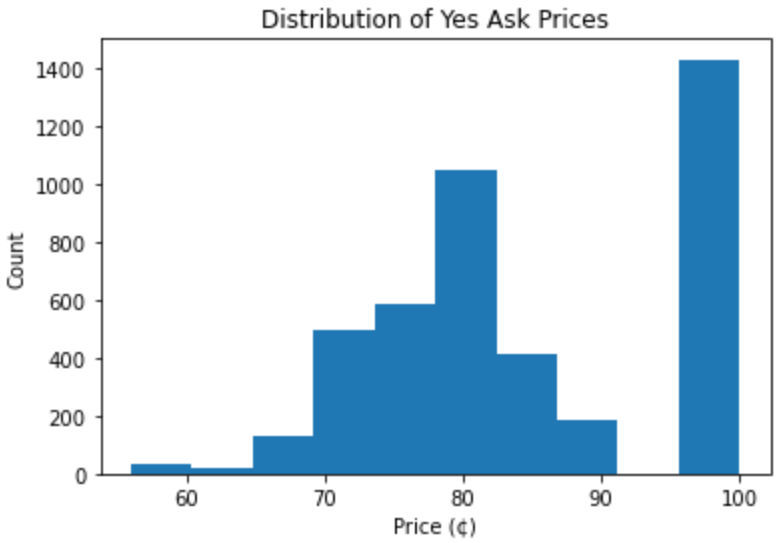
High bid-ask spread corresponds to low liquidity; Low bid-ask spread with high liquidity. This makes sense because lower transaction costs make trades easier.

We see in May, 6 months before November, there is a a high-bid ask spread. The outcomes contract is less liquid. In the summer, the liquid is relatively low. In October, the liquidity becomes quite volitile with the outcomes contract itself.

To further explore the what we're working with, Let's look at the distributions of each variable.

In [29]:
```
fig, ax = plt.subplots(1, 1)

ax.hist(df['yes_ask'])
ax.set_xlabel('Price (¢)')
ax.set_ylabel('Count')
ax.set_title('Distribution of Yes Ask Prices')
```

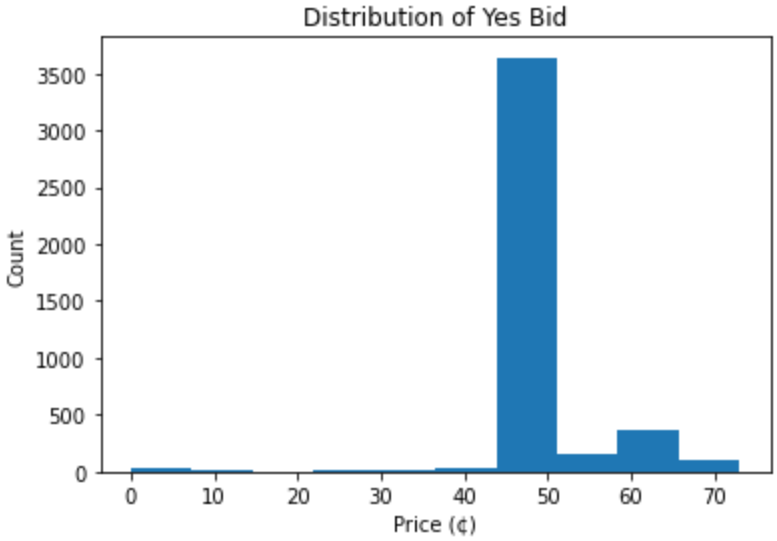
Out[29]:
```
Text(0.5, 1.0, 'Distribution of Yes Ask Prices')
```



Yes_ask is the price for the lowest YES sell offer on the market by the timestamp in the request (ts).

In [30]:
```
fig, ax = plt.subplots(1, 1)

ax.hist(df['yes_bid'])
ax.set_xlabel('Price (¢)')
ax.set_ylabel('Count')
ax.set_title('Distribution of Yes Bid')
```

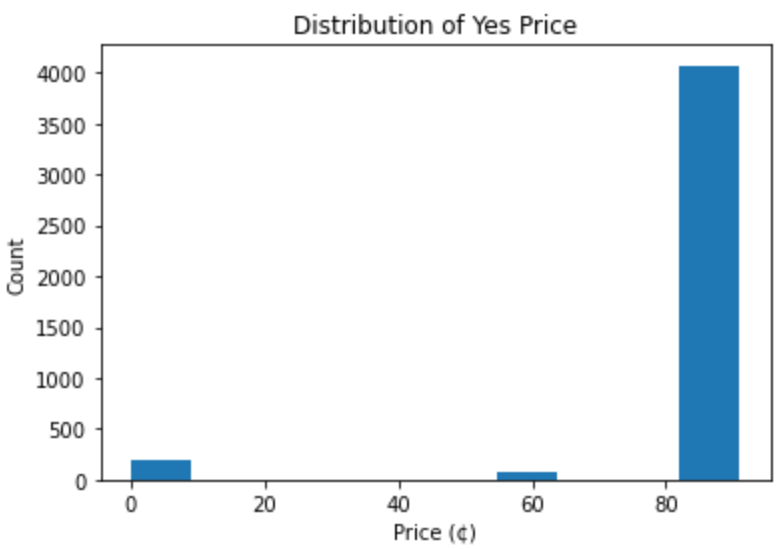
Out[30]:
```
Text(0.5, 1.0, 'Distribution of Yes Bid')
```



Yes_bid is price for the highest YES buy offer on the market by the timestamp in the request (ts).

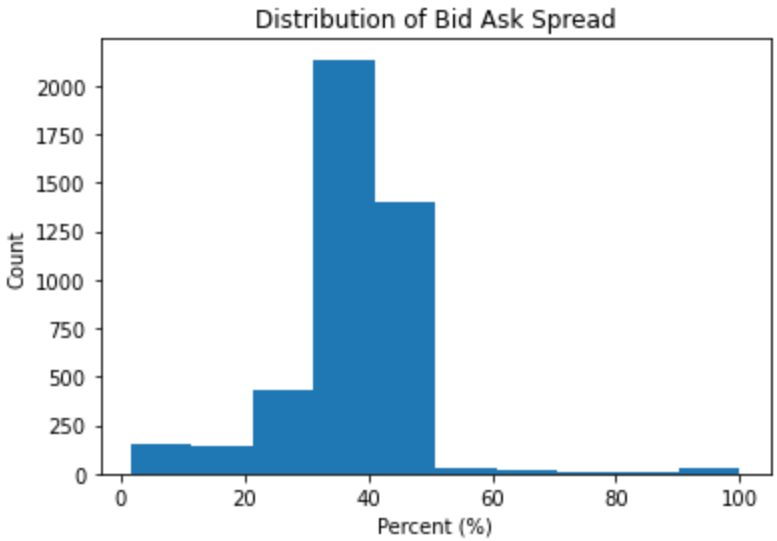
In [31]:
```
fig, ax = plt.subplots(1, 1)

ax.hist(df['yes_price'])
ax.set_xlabel('Price (¢)')
ax.set_ylabel('Count')
ax.set_title('Distribution of Yes Price')
```

Out[31]:
```
Text(0.5, 1.0, 'Distribution of Yes Price')
```



In [32]:
```
fig, ax = plt.subplots(1, 1)

ax.hist(df['BAS_pct'])
ax.set_xlabel('Percent (%)')
ax.set_ylabel('Count')
ax.set_title('Distribution of Bid Ask Spread')
```

Out[32]:
```
Text(0.5, 1.0, 'Distribution of Bid Ask Spread')
```



In [33]:
```
df['bid_ask_spread'].median()
```

Out[33]:
```
31.0
```

Yes_price is the price for the lowest YES sell offer on the market by the timestamp in the request (ts). This makes sense because the greatest concentration of data points are in late October, and the volitility centers are 0.3.

Let's prepare the dataframe for hypthesis testing.

This means we want to combine the CPI data with the Kalshi inflation betting data. Since we only have 1 inflation measurement (CPI percent change) for each month, I will add the same value to each entry for the month. For example, each row in May gets the value 0.6.

In [35]:
```
# first let's isolate the year of interest
cpi22_dict = cpi.iloc[10].reset_index(drop=True).to_dict()
```

```
# The indices in cpi22 correspond with the month of interst. 1 -> Jan, 2 -> Feb, and so
df['month'] = df['ts'].dt.month
df['cpi_pct'] = df['month'].map(cpi22_dict)
df['cpi_pct']
```

Out[35]:
```
0       0.6
1       0.6
2       0.6
3       0.6
4       0.6
       ...
4340    0.3
4341    0.3
4342    0.3
4343    0.3
4344    0.3
Name: cpi_pct, Length: 4345, dtype: float64
```

Now that we have the CPI percent change associated with each row, we are ready to test our hypothesis!

In [36]:
```
df.head()
```

Out[36]:

	index	yes_price	yes_bid	yes_ask	no_bid	no_ask	volume	open_interest	ts	bid_ask_spread
0	0	0	1	100	0	99	0	0	2022-05-25 23:10:55	99
1	1	0	2	100	0	98	0	0	2022-05-25 23:10:55	98
2	2	0	3	100	0	97	0	0	2022-05-25 23:10:55	97
3	3	0	6	100	0	94	0	0	2022-05-25 23:10:55	94
4	4	0	12	100	0	88	0	0	2022-05-25 23:10:55	88

Hypothesis Testing

Null hypothesis: The calculated inflation rate of the current month does not impact the yes_price.

I will test this using linear regression of the CPI relative percentage on yes_ask prices. I am interested in yes_ask because these are the prices we would buy for. We want to buy before the profit margins are too low.

In [37]:
```
import statsmodels.api as sm

x = df['cpi_pct'].tolist()
y = df['yes_ask'].tolist()

x = sm.add_constant(x)

model = sm.OLS(y,x)
```

```
results = model.fit()

print(results.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.002
Model:                            OLS   Adj. R-squared:                  0.002
Method:                 Least Squares   F-statistic:                     10.19
Date:                Fri, 16 Dec 2022   Prob (F-statistic):            0.00142
Time:                        04:04:39   Log-Likelihood:                -16431.
No. Observations:                4345   AIC:                         3.287e+04
Df Residuals:                    4343   BIC:                         3.288e+04
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         82.5888      0.681    121.263      0.000      81.254      83.924
x1             6.6382      2.080      3.192      0.001       2.561      10.716
==============================================================================
Omnibus:                     1587.209   Durbin-Watson:                   0.095
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              218.043
Skew:                           0.086   Prob(JB):                     4.49e-48
Kurtosis:                       1.916   Cond. No.                         14.2
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specifi
ed.
```

Since the p value (P > |t|) is 0.001 < 0.05. We reject the null hypothesis. This means that the relative inflation rate in a given month does impact the ask_price, BUT there is very little relationship. The R^2 value is basically 0.

Let's explore if we can apply Autoregressive Integrated Moving Averages (ARIMA) to our problem

ARIMA is used to measure events that take place over time. It's often used in statistics, economics, and finance.

For a deep dive into ARIMA, visit https://www.machinelearningplus.com/time-series/arima-model-time-series-forecasting-python/

For more information on the procedure used below, visit https://www.nbshare.io/notebook/136553745/Time-Series-Analysis-Using-ARIMA-From-StatsModels/

Testing For Stationarity of Data

First, we need to check if the data is stationary. This means that there is no trend with time.

In [38]:
```
### Testing For Stationarity
from statsmodels.tsa.stattools import adfuller
```

In [39]:
```
test_result=adfuller(df['yes_ask'])
```

In [40]:
```
#Ho: It is non stationary
#H1: It is stationary
```

```
def adfuller_test(sales):
    result=adfuller(sales)
    labels = ['ADF Test Statistic','p-value','#Lags Used','Number of Observations Used']
    for value,label in zip(result,labels):
        print(label+' : '+str(value) )
    if result[1] <= 0.05:
        print("P value is less than 0.05 that means we can reject the null hypothesis(Ho
    else:
        print("Weak evidence against null hypothesis that means time series has a unit r
```

In [41]:
```
adfuller_test(df['yes_ask'])
```

```
ADF Test Statistic : -6.423476686602462
p-value : 1.767902378243942e-08
#Lags Used : 31
Number of Observations Used : 4313
P value is less than 0.05 that means we can reject the null hypothesis(Ho). Therefore we
can conclude that data has no unit root and is stationary
```

If the data was not stationary, it is necessary to difference it. For example, differencing may mean adjusting for seasonal differences.

Auto Regressive Model

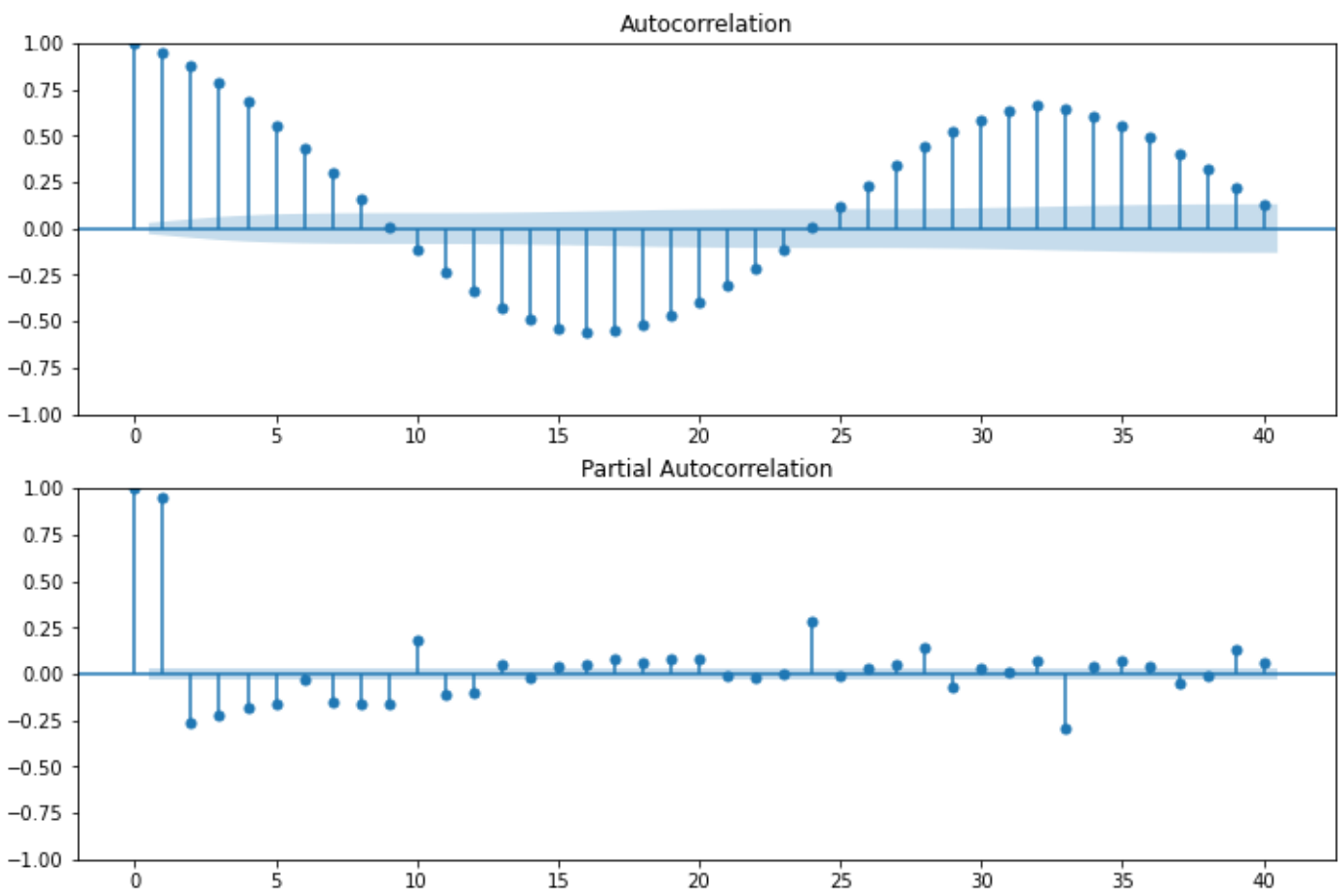
In [53]:
```
from statsmodels.tsa.arima.model import ARIMA
import statsmodels.api as sm
```

In [56]:
```
df.reset_index(inplace=True, drop=True)
```

Let us plot lags on the horizontal and the correlations on vertical axis using plot_acf and plot_pacf function.

In [57]:
```
fig = plt.figure(figsize=(12,8))
ax1 = fig.add_subplot(211)
fig = sm.graphics.tsa.plot_acf(df['yes_ask'].iloc[13:],lags=40,ax=ax1)
ax2 = fig.add_subplot(212)
fig = sm.graphics.tsa.plot_pacf(df['yes_ask'].iloc[13:],lags=40,ax=ax2)
```

```
/Users/armaanbhasin/opt/anaconda3/lib/python3.9/site-packages/statsmodels/graphics/tsapl
ots.py:348: FutureWarning: The default method 'yw' can produce PACF values outside of th
e [-1,1] interval. After 0.13, the default will change tounadjusted Yule-Walker ('ywm').
You can use this method now by setting method='ywm'.
  warnings.warn(
```



The points not in the blue shading are considering statistically significant.

In [73]:
```
model=ARIMA(df['yes_ask'],order=(1,1,1))
model_fit=model.fit()
```

In [74]:
```
model_fit.summary()
```

Out[74]:

SARIMAX Results

Dep. Variable:	yes_ask	**No. Observations:**	4345
Model:	ARIMA(1, 1, 1)	**Log Likelihood**	-11120.470
Date:	Fri, 16 Dec 2022	**AIC**	22246.939
Time:	04:22:00	**BIC**	22266.069
Sample:	0	**HQIC**	22253.692
	- 4345		
Covariance Type:	opg		

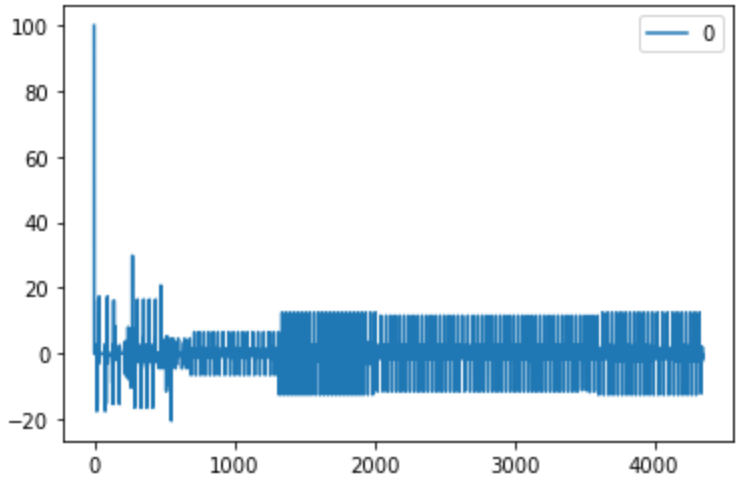
	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1	0.7695	0.042	18.326	0.000	0.687	0.852
ma.L1	-0.5689	0.049	-11.574	0.000	-0.665	-0.473
sigma2	9.7964	0.090	108.609	0.000	9.620	9.973

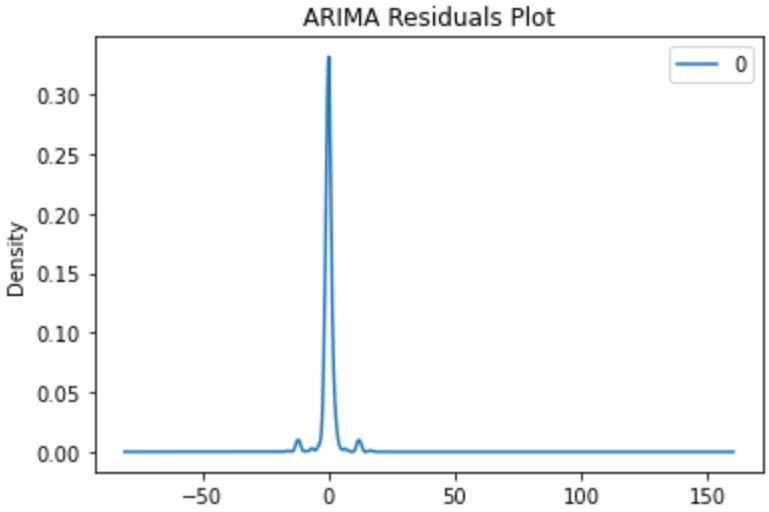
Ljung-Box (L1) (Q):	4.72	**Jarque-Bera (JB):**	32435.82
Prob(Q):	0.03	**Prob(JB):**	0.00
Heteroskedasticity (H):	1.18	**Skew:**	0.11
Prob(H) (two-sided):	0.00	**Kurtosis:**	16.38

Warnings:

[1] Covariance matrix calculated using the outer product of gradients (complex-step).

In [81]:
```
from matplotlib import pyplot
residuals = pd.DataFrame(model_fit.resid)
residuals.plot()
pyplot.show()
# density plot of residuals
residuals.plot(kind='kde', title = 'ARIMA Residuals Plot')
pyplot.show()
# summary stats of residuals
print(residuals.describe())
```

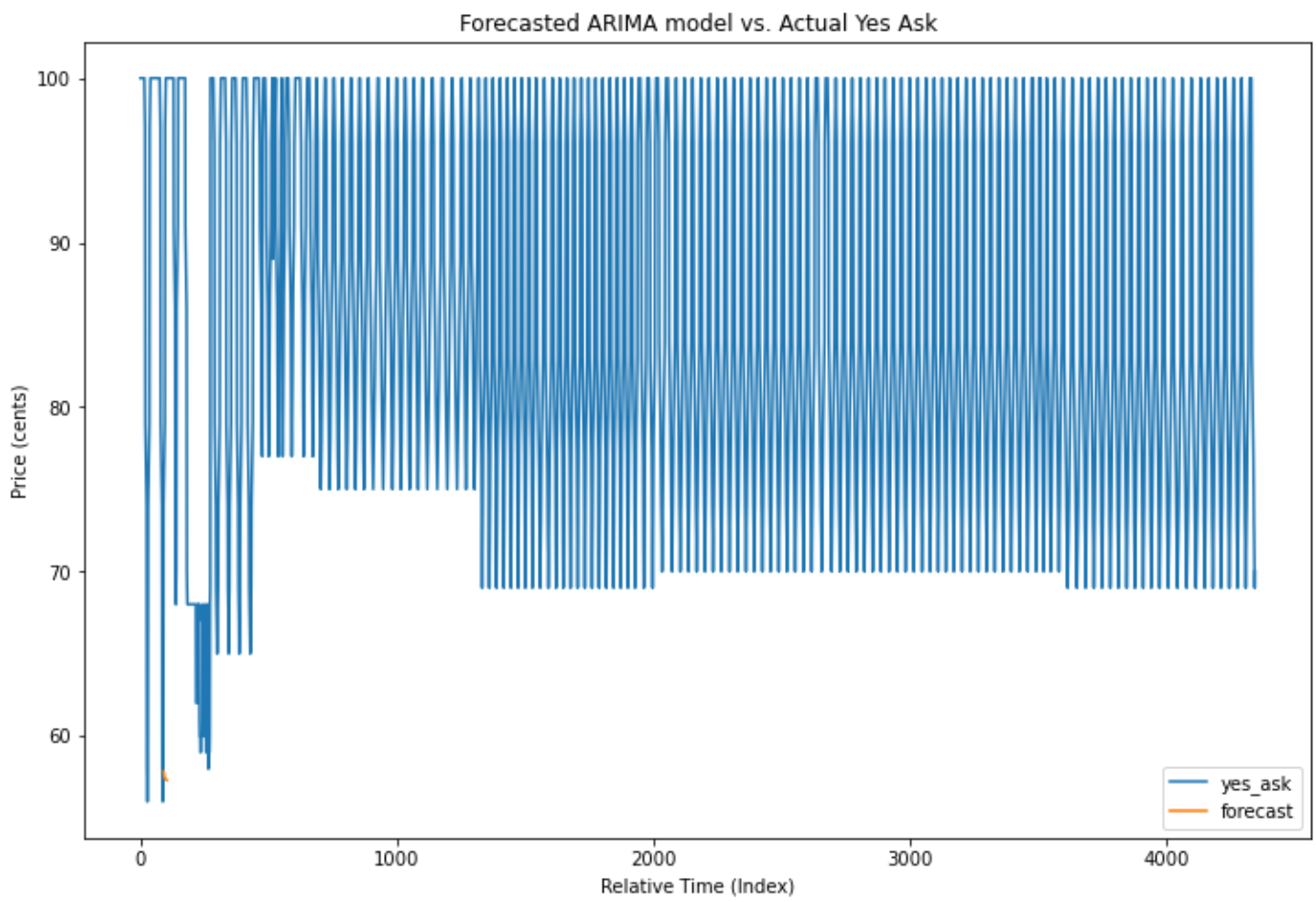





```
                0
count  4345.000000
mean      0.019241
std       3.478223
min     -20.551824
25%      -0.725700
50%      -0.010640
75%       0.509692
max     100.000000
```

In [80]:
```
df['forecast']=model_fit.predict(start=90,end=103,dynamic=True)
df[['yes_ask','forecast']].plot(figsize=(12,8), title="Forecasted ARIMA model vs. Actual
                                xlabel='Relative Time (Index)', ylabel='Price (cents)')
```

Out[80]:
```
<AxesSubplot:title={'center':'Forecasted ARIMA model vs. Actual Yes Ask'}, xlabel='Relat
ive Time (Index)', ylabel='Price (cents)'>
```



The ARIMA analysis was unsuccessful. I had difficulty implementing it because the model kept saying that my datetime did not have an inferred frequency. When I tried to coerce to a frequency of seconds, it did not work either. I'm not sure if seconds data is too frequent or this error had to be with the fact that there is much more data in October than there is in the prior months.

Please note, ultimately the graph above is plotted with the x axis using dataframe index, not datetime. I tried trouble shooting this for a few hours. First, I changed the index to datetime. Then I handled duplicates and any other concerns ARIMA had. Ultimately, datetime through indexing didn't work despite removing duplicates. Alternatively, I tried passing my datetime column 'ts' to ARIMA using the the dates parameter. This did not work because the frequency 'freq' could not be inferred or assigned.

Overall, I did get a good feel for when ARIMA should used and what considerations need to be made. In particular, seasonality needs to addressed when the data is non-stationary. Most example of ARIMA I saw online used daily data and not sporadic secondly data, so this may be to blame.

Conclusion

I learned a lot from this project. First, I learned how to use APIs. This was my first experience using APIs to gather a substantial dataset on my own. APIs are frequently used and created in industry, so I'm thankful for that exposure. It's good I got exposed to a nuanced API that included cursors and lots of documentation. For example, I had to learn not only the API but also the HTTP requests and associated functions necessary.

In addition, I learned a lot about a new financal market. Sports betting was legalized in Maryland just last month. I think this project was a great chance to learn about events contracts before this area becomes wildly popular.

At first, I felt excited to be working with raw, largely untouched data. At the same time, this came with much headache. Data is not always clean or readily accessible for analysis. Even after cleaning and manipulation, sometimes we're not lucky, and the data is not so great. This is where I had to make the most of what I hadx.

Despite poor data quality, I was able to thoroughly explore trends. I looked into the price movement over time, distributions of the various prices, and liquidity through bid-ask spreads. My analysis involved linear regression with a signal of historical CPI prices. Lastly, I thoroughly enjoyed delving into ARIMA. ARIMA is an industry standard time series data analysis tool, and I'm glad I'm now aware of and able to leverage it.