Within this project, I would like to explore the CoinGecko API and the package "geckor" in R. Then I would collect the current and historical cryptocurrency market data for a coin using the public 'CoinGecko' API and do a Arima forecast.

Data preparation

Let's use dogecoin as an example here. First, I want to obtain historical data for dogecoin. The function "coin_history" can retrieve coin-specific market data for the last n days. If open-high-low-close price data is needed, use function "coin_history_ohlc" instead.

```
r <- coin_history(coin_id = "dogecoin", vs_currency = "usd", days = "max")
```

```
## Warning: Missing values found in column(s)
## * market_cap
```

```
r
```

```
## # A tibble: 3,204 x 6
##    timestamp           coin_id  vs_currency    price total_volume market_cap
##    <dttm>              <chr>    <chr>          <dbl>        <dbl>      <dbl>
##  1 2013-12-15 00:00:00 dogecoin usd         0.000559            0    3488670
##  2 2013-12-17 00:00:00 dogecoin usd         0.000218            0    1619159
##  3 2013-12-18 00:00:00 dogecoin usd         0.000268            0    2191987
##  4 2013-12-19 00:00:00 dogecoin usd         0.000475            0    4299422
##  5 2013-12-20 00:00:00 dogecoin usd         0.000989            0    9866232
##  6 2013-12-21 00:00:00 dogecoin usd         0.000438            0    4686300
##  7 2013-12-22 00:00:00 dogecoin usd         0.000405            0    4639566
##  8 2013-12-23 00:00:00 dogecoin usd         0.000330            0    4022744
##  9 2013-12-24 00:00:00 dogecoin usd         0.000712            0    9213651
## 10 2013-12-25 00:00:00 dogecoin usd         0.000582            0    7955874
## # ... with 3,194 more rows
```

Since we only need date and price from obtained data, I save them in a dataframe and convert it to timeserie formatt for later use. So now df has all daily market price for dogecoin. Below is last 7 days price in ts object. Since each coin has different start date in coin market, we need to record the first date in df.

```
var <- c("timestamp", "price")
df <- r[var]
df <- df[c(1:nrow(df) - 1),]
date <- df$timestamp
start_date <- date[1]
dayOfYear <- as.numeric(format(as.Date(start_date),"%j"))
year <- as.numeric(format(as.Date(start_date),"%Y"))
df <- ts(df$price, start = c(year, dayOfYear), frequency = 365)
tail(df,7)
```

```
## Time Series:
## Start = c(2022, 260)
## End = c(2022, 266)
## Frequency = 365
## [1] 0.06229599 0.05760080 0.05871669 0.05851463 0.05749412 0.05979688 0.06337732
```

Now, let's define the training and test period. I will use the last 7 days as test set and all other historical data as training set.

```
train_end <-  length(df) - 7
test_start <-  length(df) - 6

df_train <-  ts(df[c(1:train_end)])
df_test <-  ts(df[c(test_start:length(df))])
```

We now need to check the stationary of our data. We can do that with the Augmented Dickey-Fuller Test.

H0: The time series is non-stationary. H1: The time series is stationary. Since the p-value is not less than .05, we fail to reject the null hypothesis. This means the time series is non-stationary. Our data is depend on the time at which the series is observed.

```
adf.test(df)
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  df
## Dickey-Fuller = -2.7914, Lag order = 14, p-value = 0.2432
## alternative hypothesis: stationary
```

Time series modeling

ARIMA models which aim to describe the autocorrelations in the data.

We will run the auto.arima function on our training data, which will help us to forecast next 7 days market price and their prediction intervals.

```
fit_arima <- auto.arima(df_train)
fcast_arima <- forecast(fit_arima, h = 7, level = 95)

fcast_arima
```

```
##      Point Forecast      Lo 95      Hi 95
## 3197     0.06084483 0.04389762 0.07779204
## 3198     0.06193122 0.03977521 0.08408722
## 3199     0.06277872 0.03536605 0.09019138
## 3200     0.06299779 0.03053659 0.09545898
## 3201     0.06253890 0.02568147 0.09939633
## 3202     0.06168589 0.02133391 0.10203786
## 3203     0.06088812 0.01787553 0.10390072
```

Now, let's get the Metrics MAPE which is used to judge the performance of the model.It gives the average deviation between the forecast value and actual values.

```
mape_arima <- mape(df_test,as.numeric(fcast_arima$mean))*100
mape_arima
```

```
## [1] 5.75545
```

Exponential smoothing models (ETS) are based on a description of the trend and seasonality in the data.

```
fit_ets <- ets(df_train)
fcast_ets <- forecast(fit_ets, h = 7, level = 95)

mape_ets <- mape(df_test,as.numeric(fcast_ets$mean))*100
mape_ets
```

```
## [1] 3.208157
```

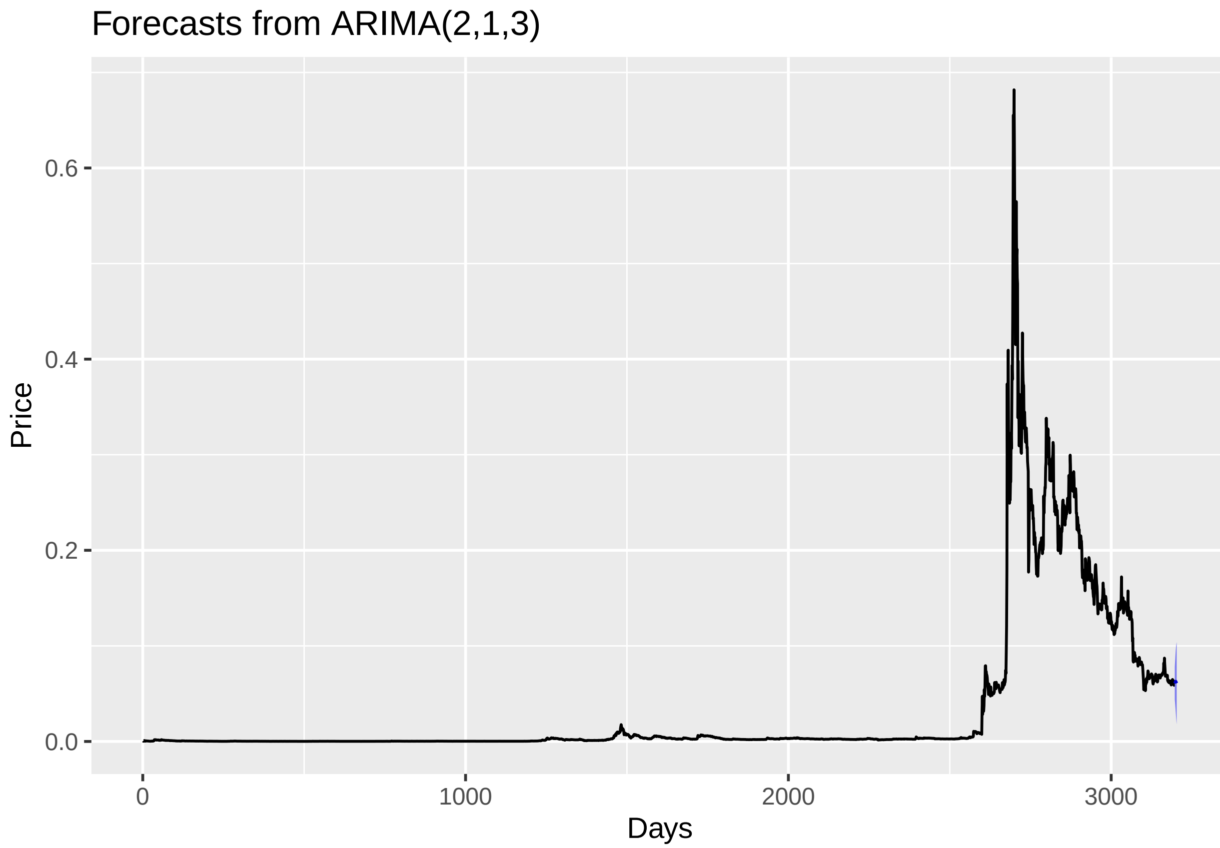
let's create a table of predicted price, actual price and their MAPE at last row. This will help us to do comparison.

```
result <- cbind(df_test,as.numeric(fcast_arima$mean),as.numeric(fcast_ets$mean))
result <- rbind(result,c(0,mape_arima,mape_ets))
colnames(result) <- c("Actual","ARIMA","ETS")
round(result,4)
```

```
##      Actual  ARIMA    ETS
## [1,] 0.0623 0.0608 0.0602
## [2,] 0.0576 0.0619 0.0600
## [3,] 0.0587 0.0628 0.0598
## [4,] 0.0585 0.0630 0.0595
## [5,] 0.0575 0.0625 0.0593
## [6,] 0.0598 0.0617 0.0591
## [7,] 0.0634 0.0609 0.0589
## [8,] 0.0000 5.7554 3.2082
```

Lastly, plot the historical price for bitcion, certainly the price has been jumping insane last few years.

```
autoplot(fcast_arima) + (labs(y = "Price", x = "Days"))
```


Forecasts from ARIMA(2,1,3)
Price
0.0
0.2
0.4
0.6
0
1000
2000
3000
Days