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.

Let's use bitcoin as an example here. First, I want to obtain historical data for bitcoin. 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 = "bitcoin", vs_currency = "usd", days = "max")
```

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

```
r
```

```
## # A tibble: 3,432 x 6
##    timestamp           coin_id vs_currency price total_volume market_cap
##    <dttm>              <chr>   <chr>       <dbl>        <dbl>      <dbl>
##  1 2013-04-28 00:00:00 bitcoin usd         135.             0 1500517590
##  2 2013-04-29 00:00:00 bitcoin usd         142.             0 1575032004
##  3 2013-04-30 00:00:00 bitcoin usd         135.             0 1501657493
##  4 2013-05-01 00:00:00 bitcoin usd         117              0 1298951550
##  5 2013-05-02 00:00:00 bitcoin usd         103.             0 1148667722
##  6 2013-05-03 00:00:00 bitcoin usd          91.0            0 1011066494
##  7 2013-05-04 00:00:00 bitcoin usd         111.             0 1236351844
##  8 2013-05-05 00:00:00 bitcoin usd         117.             0 1298377788
##  9 2013-05-06 00:00:00 bitcoin usd         118.             0 1315992304
## 10 2013-05-07 00:00:00 bitcoin usd         106.             0 1183766500
## # ... with 3,422 more rows
```

Since we only need timestamp 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 bitcoin.

```
var <- c("timestamp", "price")
df <- r[var]
df <- df[c(1:nrow(df) - 1),]
df <- ts(df["price"], start = c(2013,4,28), frequency = 365)
```

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

```
n = nrow(df)
n1 = nrow(df) - 6
n2 = nrow(df) - 7

df_train = ts(df[c(4:n2),])
df_test = ts(df[c(n1:n),])
df_full = ts(df[c(4:n),])
```

We now need to check the stationary of our data. We need a unstationary data because a stationary time series is one whose properties do not depend on the time at which the series is observed. 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_full)
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  df_full
## Dickey-Fuller = -2.1295, Lag order = 15, p-value = 0.5235
## alternative hypothesis: stationary
```

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

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

fcast
```

```
##      Point Forecast    Lo 95    Hi 95
## 3422       22339.67 20879.91 23799.43
## 3423       22339.67 20275.26 24404.08
## 3424       22339.67 19811.30 24868.04
## 3425       22339.67 19420.16 25259.19
## 3426       22339.67 19075.55 25603.79
## 3427       22339.67 18764.01 25915.33
## 3428       22339.67 18477.52 26201.83
```

let's create a table of predicted price and actual price.

```
result <- cbind(df_test,as.numeric(fcast$mean))
colnames(result) <- c("Actual","Predicted")
result
```

```
## Time Series:
## Start = 1
## End = 7
## Frequency = 1
##     Actual Predicted
## 1 20184.97  22339.67
## 2 20255.92  22339.67
## 3 19702.17  22339.67
## 4 19764.41  22339.67
## 5 20131.68  22339.67
## 6 19437.16  22339.67
## 7 19570.39  22339.67
```

Now, let's get the Metrics MAPE which is used to judge the performance of the model. we have a mape of 12.49%, which means the average deviation between the forecasted value and actual values was 12.49%.

```r
mape <- mape(df_test,as.numeric(fcast$mean))

mape*100
```

```
## [1] 12.48992
```

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

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