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

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

```r
r
```

```
## # A tibble: 3,435 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 3,425 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. Below is last 7 days price in ts object.

```r
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] 19764.41 20131.68 19437.16 19570.39 18869.93 18539.64 19464.32
```

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 = length(df)
n1 = n - 6
n2 = n - 7

df_train = df[c(1:n2)]
df_test = df[c(n1:n)]
```

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 = -2.1268, Lag order = 15, p-value = 0.5246
## 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 their prediction intervals.

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

fcast
```

```
##      Point Forecast    Lo 95    Hi 95
## 3428       19702.17 18241.79 21162.55
## 3429       19702.17 17636.88 21767.46
## 3430       19702.17 17172.71 22231.63
## 3431       19702.17 16781.41 22622.93
## 3432       19702.17 16436.66 22967.68
## 3433       19702.17 16124.98 23279.36
## 3434       19702.17 15838.36 23565.98
```

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
```

```
##        Actual Predicted
## [1,] 19764.41  19702.17
## [2,] 20131.68  19702.17
## [3,] 19437.16  19702.17
## [4,] 19570.39  19702.17
## [5,] 18869.93  19702.17
## [6,] 18539.64  19702.17
## [7,] 19464.32  19702.17
```

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

mape*100
```

```
## [1] 2.34116
```

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

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