**Title:** Task B-5 Report
**Author:** Robin Findlay-Marks, s103603871

### Task information:

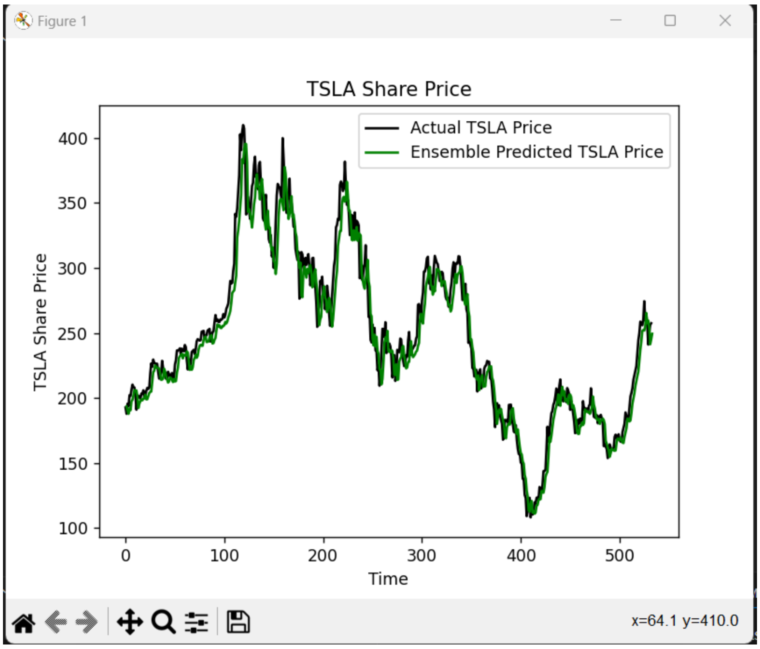
Subtask 1:
For this subtask I had to make the program first of all ran the ARIMA prediction model, and then combined this in an ensemble with the original RNN type models. The first step of this was very easy. There were several tutorials showing me what I had to do so I followed them and made a working ARIMA prediction model in about 15 minutes. A big difference between ARIMA and the RNN models is that they often need to be remade after each prediction. Other than this the whole process of getting test and training data then making the model and predicting (or forecasting) is very straightforward and similar to the RNN models.

```python
def ARIMA_prediction():

    # assign train and test data to variables
    train = trainData['Close'].values
    test1 = testData[1:]
    test = test1['Close'].values
    history = [x for x in train]
    predictions = list()

    # walk-forward validation
    for t in range(len(test)):
        # re-create the ARIMA model after each new observation
        model = ARIMA(history, order=(AUTOREG,DIFERENCE,MOVAVG))
        model_fit = model.fit()
        # make prediction
        output = model_fit.forecast()
        forecast = output[0]
        predictions.append(forecast)
        expected = test[t]
        # keep track of past observations
        history.append(expected)
        print('predicted=%f, expected=%f' % (forecast, expected))
    return predictions


    #LINK https://machinelearningmastery.com/arima-for-time-series-forecasting-with-python/


    #LINK https://medium.com/@cortexmoldova/arima-time-series-forecasting-model-with-python-5b0cfdbb08fa
    # Maybe use this one
```

The next step was significantly more tricky, despite the end solution being very simple. I quickly learnt that an ensemble ai model is a model consisting of two or more prediction models working together somehow. There are several ways of doing this including having an ensemble as a function with inputs of already trained models. I decided to try this method, because I already have trained models. I soon realised that this wouldn't work because of the quirt of ARIMA prediction models where they often need to be remade after each prediction. I tried to find ARIMA prediction code where this did not happen but I couldn't. After several hours of research and thinking I remembered that there is another type of ensemble prediction model that simply averages the outputs of the ensemble sub-model outputs. I quickly used this and made a working ensemble model.

In the end this was all the code needed to turn the LSTM and ARIMA prediction models into an ensemble.

```
ensemble_preds = None
if ENSEMBLE:
    predicted_prices_list = predicted_prices.tolist()

    i = 0
    for value in arima_pred:
        ensemble_preds = np.append(ensemble_preds, (arima_pred[i] + predicted_prices_list[i])/2)
        i += 1
```




Subtask 2:
For this subtask first I modified the program to be able to use SARIMA instead of ARIMA. This was easy as SARIMA is really just ARIMA with extra parameters for a season component.

```
predictions = list()
my_seasonal_order = (SAUTOREG, SDIFERENCE, SMOVAVG, SEASON)

# walk-forward validation
for t in range(len(test)):
    if SARIMA:
        model = ARIMA(history, order=(AUTOREG,DIFERENCE,MOVAVG), seasonal_order=my_seasonal_order)
    else:
        model = ARIMA(history, order=(AUTOREG,DIFERENCE,MOVAVG))
```

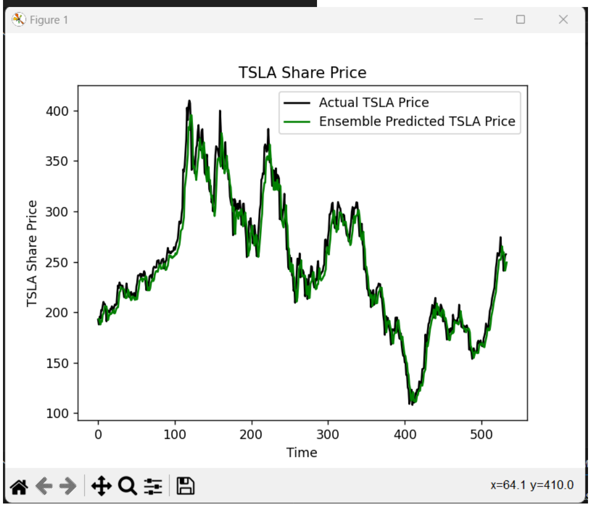
Hyperparameters were added to the parameters.py file
I also renamed to old hyperparameters to RNN_HYPERPARAMETERS
The new hyperparameters are ARIMA_HYPERPARAMETERS for the arima parameters and SARIMA_HYPERPARAMETERS for the seasonal arima parameters.

```
# Default Parameters

#RNN param
LAYER_NUM = 2
LAYER_SIZE = 50
LAYER_NAME = SimpleRNN
DROPOUT = 0.2

#ARIMA param
AUTOREG = 5      #Trend autoregression order
DIFERENCE = 1    #Trend difference order
MOVAVG = 0       #Trend moving average order

#SARIMA param
SAUTOREG = 1     #Seasonal autoregressive order
SDIFERENCE = 1   #Seasonal difference order
SMOVAVG = 0      #Seasonal moving average order
SEASON = 6       #The number of time steps for a single seasonal period

# Hyperparameters

RNN_HYPERPARAM = 5
ARIMA_HYPERPARAM = 1
SARIMA_HYPERPARAM = 1

match RNN_HYPERPARAM:
    case 1: #LSTM
```

# ARIMA

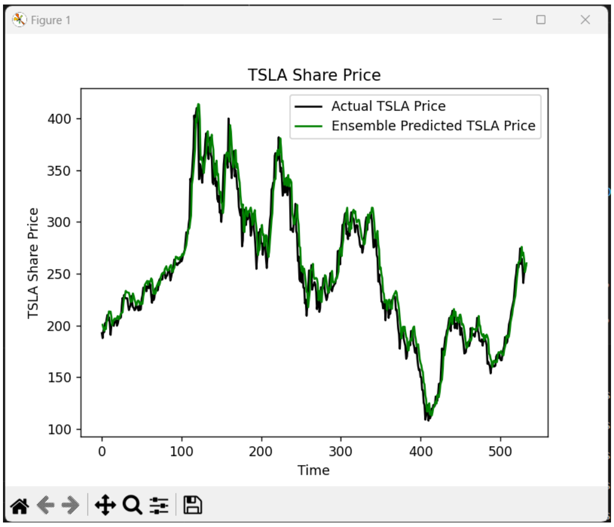
## Hyperparameter 1

```
case 1:
    AUTOREG = 5
    DIFERENCE = 1
    MOVAVG = 0
```



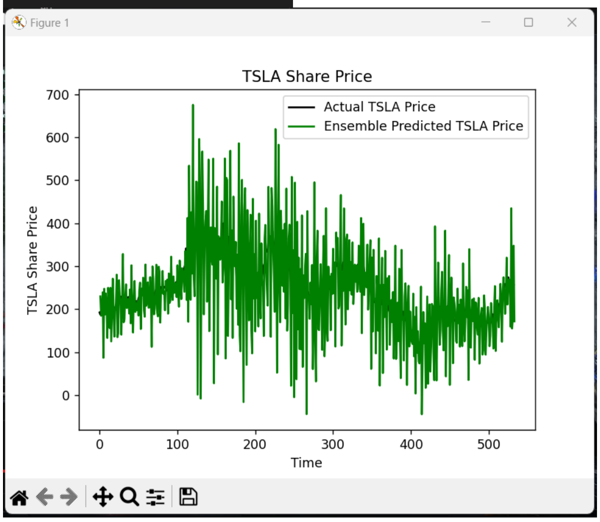

## Hyperparameter 2

```
case 2:
    AUTOREG = 10
    DIFERENCE = 1
    MOVAVG = 0
```



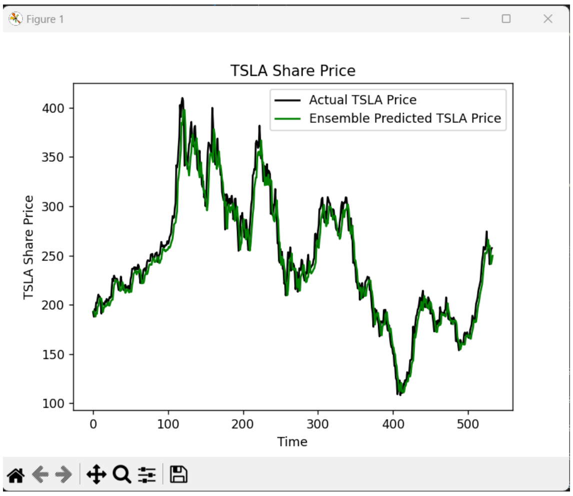

## Hyperparameter 3

```
case 3:
    AUTOREG = 5
    DIFERENCE = 10
    MOVAVG = 0
```



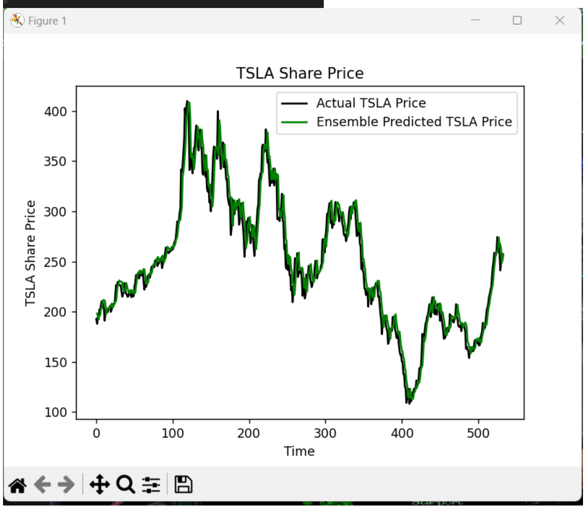

## Hyperparameter 4

```
case 4:
    AUTOREG = 5
    DIFERENCE = 1
    MOVAVG = 1
```




## Hyperparameter 5

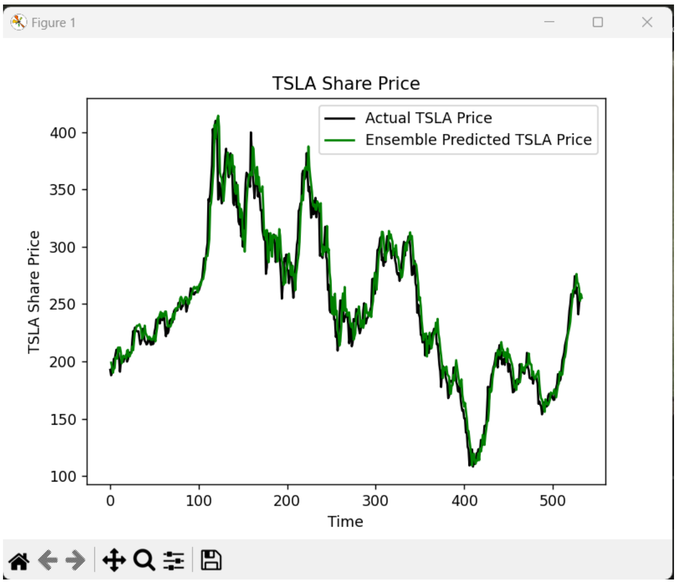
```
case 5:
    AUTOREG = 5
    DIFERENCE = 1
    MOVAVG = 10
```




## SARIMA

## Hyperparameter 1:

```
case 1: #7 day week
    SAUTOREG = 1
    SDIFERENCE = 1
    SMOVAVG = 0
    SEASON = 7
```




Other SARIMA hyperparameters

```
case 2: #30 day month
    SAUTOREG = 1
    SDIFERENCE = 1
    SMOVAVG = 0
    SEASON = 30
case 3: # 365 day year
    SAUTOREG = 1
    SDIFERENCE = 1
    SMOVAVG = 0
    SEASON = 365
```

The second and third SARIMA hyperparameters were not run because they would take over 15 minutes to run

References:

**Brownlee, J 2020, *How to Create an ARIMA Model for Time Series Forecasting in Python*, relataly, viewed 4/10/2023, <https://machinelearningmastery.com/arima-for-time-series-forecasting-with-python/>.**

**Brownlee, J 2021, *Ensemble Machine Learning With Python (7-Day Mini-Course)* 4/10/2023, <https://machinelearningmastery.com/ensemble-machine-learning-with-python-7-day-mini-course/>**

**Brownlee, J 2020, *How to Develop an Ensemble of Deep Learning Models in Keras)* 4/10/2023, <https://machinelearningmastery.com/model-averaging-ensemble-for-deep-learning-neural-networks/>**