CODE EXPLANATION

```
import streamlit as st
```

This line imports the Streamlit library and renames it as st.

```
def predict_price(ticker, days):
```

This defines a function predict_price that takes two arguments, ticker (the stock symbol) and days (the number of days to predict the stock price).

```
# Importing required libraries
    import numpy as np
    import pandas as pd
    import yfinance as yf
    from sklearn.preprocessing import MinMaxScaler
    from tensorflow.keras.models import Sequential
    from tensorflow.keras.layers import Dense, LSTM
```

This section imports all the necessary libraries for the project. numpy, pandas, and yfinance are used for data processing and retrieval. MinMaxScaler is used to scale the data, and Sequential, Dense, and LSTM are used to create the machine learning model.

```
    # Defining the function to get real-time data
    def get_realtime_data(ticker, interval='1m', range='1d'):
        data = yf.download(tickers=ticker, interval=interval, period=range)
        return data
```

This defines a function get_realtime_data that takes three arguments: ticker (the stock symbol), interval (the time interval for the data), and range (the range for the data). This function uses the yfinance library to download the stock data and returns it.

```
    # Preprocessing the data
    def preprocess_data(data):
        scaler = MinMaxScaler(feature_range=(0, 1))
        scaled_data = scaler.fit_transform(data['Close'].values.reshape(-1, 1))
        return scaled_data, scaler
```

This defines a function preprocess_data that takes the stock data as an argument. This function scales the data using MinMaxScaler and returns the scaled data and the scaler object.

```
    # Creating the dataset for LSTM
    def create_dataset(dataset, look_back=1, days=1):
        X, Y = [], []
        for i in range(len(dataset)-look_back-days):
            a = dataset[i:(i+look_back), 0]
            X.append(a)
            Y.append(dataset[i + look_back:i + look_back + days, 0])
        return np.array(X), np.array(Y)
```

This defines a function create_dataset that takes the scaled data, look_back (the number of previous days to consider), and days (the number of days to predict) as arguments. This function creates the input and output sequences for the LSTM model.

```
    # Creating the LSTM model
    def create_model():
        model = Sequential()
        model.add(LSTM(50, return_sequences=True, input_shape=(look_back, 1)))
        model.add(LSTM(50, return_sequences=True))
        model.add(LSTM(50))
        model.add(Dense(1))
        model.compile(optimizer='adam', loss='mean_squared_error')
        return model
```

This defines a function create_model that creates the LSTM model using the Sequential API in Keras. It consists of three LSTM layers with 50 units each, followed by a Dense layer with one output. The model is compiled with the adam optimizer and the mean squared error loss.

```
def predict_prices(model, data, scaler):
    last_data = data.tail(look_back)
    last_data_scaled = scaler.transform(last_data['Close'].values.reshape(-1, 1))
    X_test = np.array([last_data_scaled])
    X_test = np.reshape(X_test, (X_test.shape[0], X_test.shape[1], 1))
    prediction = model.predict(X_test)
    prediction = scaler.inverse_transform(prediction)
    return prediction[0][0]
```

This function takes in a trained model, the data to predict on, and a scaler object to transform the data. It first gets the last look_back number of rows of the data, scales the 'Close' column using the scaler object, and reshapes it to a 3D array to match the model's input shape. It then makes a prediction using the model, scales the prediction back to the original range using the scaler, and returns the first value of the prediction.

```
def main(ticker):
    data = get_realtime_data(ticker)
    scaled_data, scaler = preprocess_data(data)
    X, Y = create_dataset(scaled_data, look_back)
    X = np.reshape(X, (X.shape[0], X.shape[1], 1))
    model = create_model()
    model.fit(X, Y, batch_size=64, epochs=100,verbose=1)
    prediction = predict_prices(model, data, scaler)
    return prediction
```

This function takes in a ticker symbol and uses it to fetch real-time data using the get_realtime_data() function. It then preprocesses the data by scaling it and splitting it into training data (X) and target data (Y) using the create_dataset() function. It reshapes X to a 3D array to match the model's input shape, creates a new model, fits the model to the data, and finally calls the predict_prices() function to make a prediction on the latest data. It returns the predicted value.

```
look_back = 60
predicted_price = main(ticker)
return predicted_price
```

This sets the look_back value to 60, calls the main() function with a ticker argument to get the predicted price, and returns the predicted price.

```
def main():
    st.title("Real-Time Stock Price Prediction")

    # Create input fields for entering the stock symbol and the number of days to predict
    symbol = st.text_input("Enter stock symbol")
    days = st.slider("Enter number of days to predict", min_value=1, max_value=30, value=1)
```

```
    # Create a button to trigger the prediction
    if st.button("Predict"):
        predicted_price = predict_price(symbol, days)
        st.success(f"Predicted price for {symbol} in {days} days: {predicted_price:.2f}")
```

This function defines the main user interface for the web app. It creates a title and input fields for the symbol and days variables, which are used as inputs for the prediction function. It then creates a button that triggers the prediction and displays the predicted price in a success message.

```
if __name__ == "__main__":

    st.set_page_config(
        page_title="STOCK PRICE PREDICTION",
        page_icon=":chart_with_upwards_trend:",
        layout="wide",
        initial_sidebar_state="collapsed",
        )
    main()
```

This code block defines the main entry point of the program, where the __name__ variable is used to determine whether the script is being run as the main program or if it is being imported as a module.

if __name__ == "__main__": checks whether the current script is being run as the main program.

st.set_page_config sets the configuration options for the web app page. It sets the page title to "STOCK PRICE PREDICTION", the page icon to ":chart_with_upwards_trend:", the layout to "wide", and the initial sidebar state to "collapsed".

main() calls the main() function, which is defined elsewhere in the code and is responsible for creating the user interface and making the stock price predictions.

\

FUNCTION EXPLANATION

fit_transform is a method in scikit-learn's preprocessing module that combines the fit and transform methods for a given transformer. It first fits the transformer to the data by computing the mean and variance (or any other statistics) of the training data, and then transforms the data by scaling it based on the computed statistics. The fit_transform method returns the transformed data, which can be used for further processing or modeling.

look_back is a variable used in the code to specify the number of previous days' data to consider for predicting the stock price. It is used in the create_dataset() function to create the input and output data for the LSTM model. It is also used in the predict_prices() function to extract the last look_back days' data and use it for making the prediction

look_back=60

---the model is using the previous 60 days' worth of closing stock prices to predict the next day's closing price.

len(dataset)-look_back-days computes the number of samples that can be used to create the input-output pairs for the LSTM model, where each input contains look_back number of historical prices and the corresponding output contains days number of predicted prices.

optimizer='adam' is a parameter of the create_model() function which sets the optimizer algorithm used during training. Here, 'adam' refers to the Adaptive Moment Estimation algorithm, which is a popular optimization algorithm for training deep learning models. The 'adam' optimizer is known for being computationally efficient, having low memory requirements, and providing good performance on a wide range of deep learning tasks.

The first line adds an LSTM layer with 50 units, return_sequences=True means that this LSTM layer returns the full sequence of outputs (one output for each input time step). The input_shape parameter is set to (look_back, 1) which means that the input to this layer is a sequence of look_back time steps, each containing a single feature.

The second line adds another LSTM layer with 50 units, also with return_sequences=True. This layer also returns a sequence of outputs, but it takes as input the sequence of outputs produced by the previous LSTM layer.

The third line adds a third LSTM layer with 50 units, but this time return_sequences=False which means that this layer only returns the last output in the sequence.

Finally, a Dense layer with a single output neuron is added to produce the predicted stock price value.

Sequential() is a class in the Keras API that allows for the creation of a neural network model layer-by-layer in a sequential manner. Each layer is added to the model with the .add() method. The Sequential() function returns a new instance of a Keras model.

model.fit(X, Y, batch_size=64, epochs=100,verbose=1) is a method call to train the neural network model created using the Keras API.

X is the input dataset.

Y is the output/target dataset.

batch_size is the number of samples that will be propagated through the network at a time.

epochs is the number of times the training process will iterate over the entire training set.

verbose is a parameter that controls the level of logging output during the training process. A value of 1 will show a progress bar and status updates.