EXERCISE NO.: 09
NEURAL NETWORK BASED TIME SERIES FORECASTING MODEL

AIM:
To implement a neural network based time series forecasting model.

PROCEDURE:

1. **Import the necessary libraries.**
```
import pandas as pd
import numpy as np
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
import matplotlib.pyplot as plt
```

2. **Load the dataset.**
```
df = pd.read_csv('/amazon.csv', encoding='latin1')
```

3. **Preprocess the data.**
```
month_map = {
    'Janeiro': 'January', 'Fevereiro': 'February', 'Março': 'March',
    'Abril': 'April', 'Maio': 'May', 'Junho': 'June',
    'Julho': 'July', 'Agosto': 'August', 'Setembro': 'September',
    'Outubro': 'October', 'Novembro': 'November', 'Dezembro': 'December'
}

df['month'] = df['month'].map(month_map)
df['date'] = pd.to_datetime(df['month'] + ' ' + df['year'].astype(str), format='%B %Y')
df.set_index('date', inplace=True)
```

4. **Aggregate the data.**
```
df_grouped = df.groupby('date')['number'].sum().to_frame()
```

5. **Normalise the data.**
```
scaler = MinMaxScaler()
data_scaled = scaler.fit_transform(df_grouped)
```

6. **Create supervised data for the model.**
```
def create_dataset(series, window_size=3):
    X, y = [], []
    for i in range(len(series) - window_size):
```

```
        X.append(series[i:i+window_size].flatten())
        y.append(series[i+window_size])
    return np.array(X), np.array(y)

X, y = create_dataset(data_scaled, window_size=3)
```

7. **Define train-test split.**
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, shuffle=False)
```

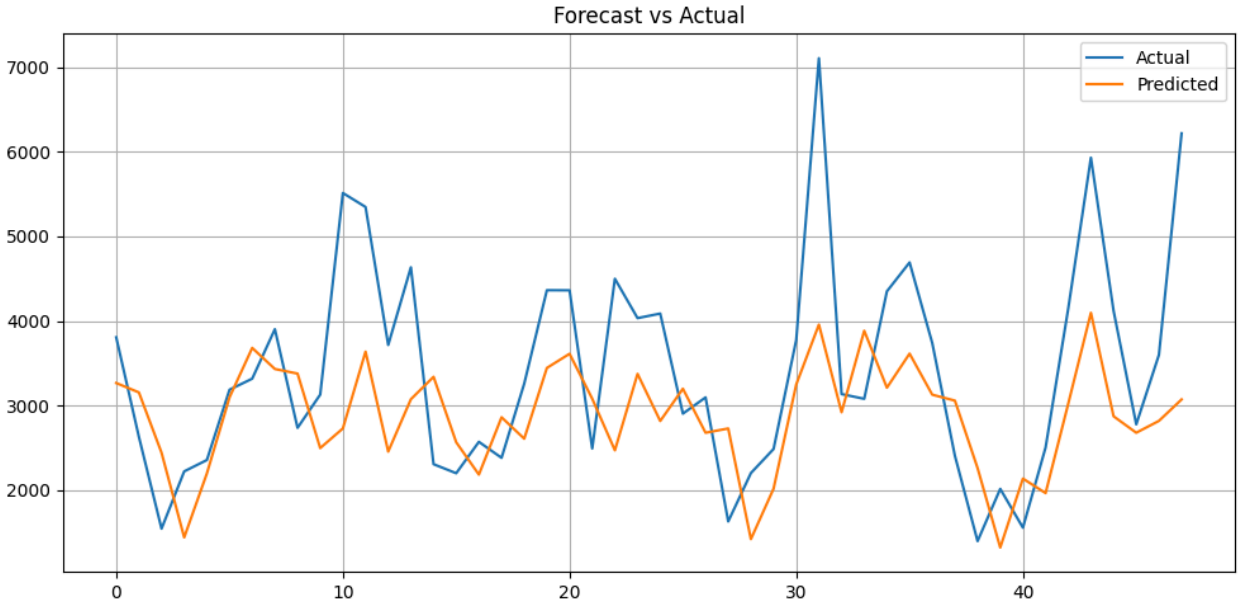
8. **Build the neural network model.**
```
model = Sequential([
    Dense(64, activation='relu', input_shape=(X_train.shape[1],)),
    Dense(32, activation='relu'),
    Dense(1)
])
model.compile(optimizer='adam', loss='mse')
model.fit(X_train, y_train, epochs=50, batch_size=4, verbose=1)
```

9. **Predict the values for the model.**
```
y_pred = model.predict(X_test)
```

10. **Apply the inverse transform.**
```
y_pred_inv = scaler.inverse_transform(y_pred)
y_test_inv = scaler.inverse_transform(y_test)
```

11. **Visualise the forecasting.**
```
plt.figure(figsize=(10, 5))
plt.plot(y_test_inv, label='Actual')
plt.plot(y_pred_inv, label='Predicted')
plt.legend()
plt.title("Forecast vs Actual")
plt.grid(True)
plt.tight_layout()
plt.show()
```

OUTPUT:



RESULT:

Thus the program has been successfully implemented and verified.