```
import tensorflow as tf

# Check if TensorFlow is using GPU
if tf.test.gpu_device_name():
    print('Default GPU Device: {}'.format(tf.test.gpu_device_name()))
else:
    print("No GPU found. Make sure you have enabled GPU acceleration in the runtime settings.")
```

```
Default GPU Device: /device:GPU:0
```

Double-click (or enter) to edit

```
import tensorflow as tf
print(tf.__version__)
```

```
2.13.0
```

```
#importing basic libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

```
dataset= pd.read_csv('Rl.csv')
```

```
dataset
```

```
del dataset['Field']
```

```
dataset = dataset[:40]
```

```
dataset = dataset.drop(dataset.columns[16:42], axis=1)
```

```
dataset
```

	Date	Total Revenue/Income(Cr)	Cost Of Revenue (Cr)	Gross Profit (Cr)	Total Operating Expense(Cr)	Operating Income/Profit(Cr)	EBITDA(Cr)	Reconciled Depreciation (Cr)	EBIT (Cr)	Interest Expense (Cr)	Income/Profit Before Tax(Cr)	Income Tax Expense (Cr)	Net Income From Continuing Operation(Cr)	Net Income(Cr)
0	1.3.23	216376.0	148559.0	64386.0	177936.0	26984.0	39361.0	11456.0	-958.00	5819.0	24083.0	2787.0	21296.0	19299.0
1	1.12.22	220592.0	149165.0	71427.0	185345.0	25060.0	36446.0	10187.0	26259.00	5201.0	23006.0	5266.0	17740.0	15792.0
2	1.9.22	232863.0	162379.0	70484.0	201639.0	21494.0	32807.0	9730.0	14131.00	4554.0	20454.0	4867.0	15587.0	13656.0
3	1.6.22	223113.0	150678.0	72435.0	190253.0	29051.0	31233.0	8946.0	29745.00	0.0	27236.0	7793.0	19443.0	17955.0
4	1.3.22	211887.0	149521.0	62366.0	184010.0	23365.0	25967.0	8001.0	-3830.00	3556.0	22411.0	4390.0	18021.0	16203.0
5	1.12.21	191271.0	132413.0	58858.0	163004.0	24859.0	29039.0	7683.0	27049.00	3812.0	25227.0	4688.0	20539.0	18549.0
6	1.9.21	167611.0	120659.0	46952.0	28162.0	18790.0	21254.0	7230.0	7141.00	3819.0	19234.0	3755.0	15479.0	13680.0
7	1.6.21	139949.0	97188.0	42761.0	123464.0	16485.0	20667.0	6883.0	19134.00	3397.0	17270.0	3464.0	13806.0	12273.0
8	1.3.21	149575.0	108510.0	41065.0	133197.0	16378.0	20426.0	6973.0	-6146.00	4044.0	16382.0	1387.0	14995.0	13227.0
9	1.12.20	117860.0	78914.0	38946.0	102959.0	14901.0	19308.0	6665.0	19308.00	4326.0	14982.0	88.0	14894.0	13101.0
10	1.9.20	116195.0	76410.0	39785.0	98917.0	12319.0	16673.0	6626.0	3739.00	6084.0	10589.0	-13.0	10602.0	9567.0
11	1.6.20	91238.0	50449.0	40789.0	77686.0	15533.0	20243.0	6308.0	20243.00	6735.0	13508.0	260.0	13248.0	13233.0
12	1.3.20	139865.0	92622.0	47243.0	120344.0	11765.0	13195.0	6332.0	-9008.00	3972.0	9223.0	2677.0	6546.0	6348.0
13	1.12.19	156802.0	109334.0	47468.0	136098.0	16664.0	20165.0	5545.0	20366.00	5404.0	14962.0	3121.0	11841.0	11640.0
14	1.9.19	152925.0	103857.0	49068.0	131689.0	16837.0	20415.0	5315.0	20505.00	5450.0	15055.0	3703.0	11352.0	11262.0

```
dataset['Date'] = pd.to_datetime(dataset['Date'])
```

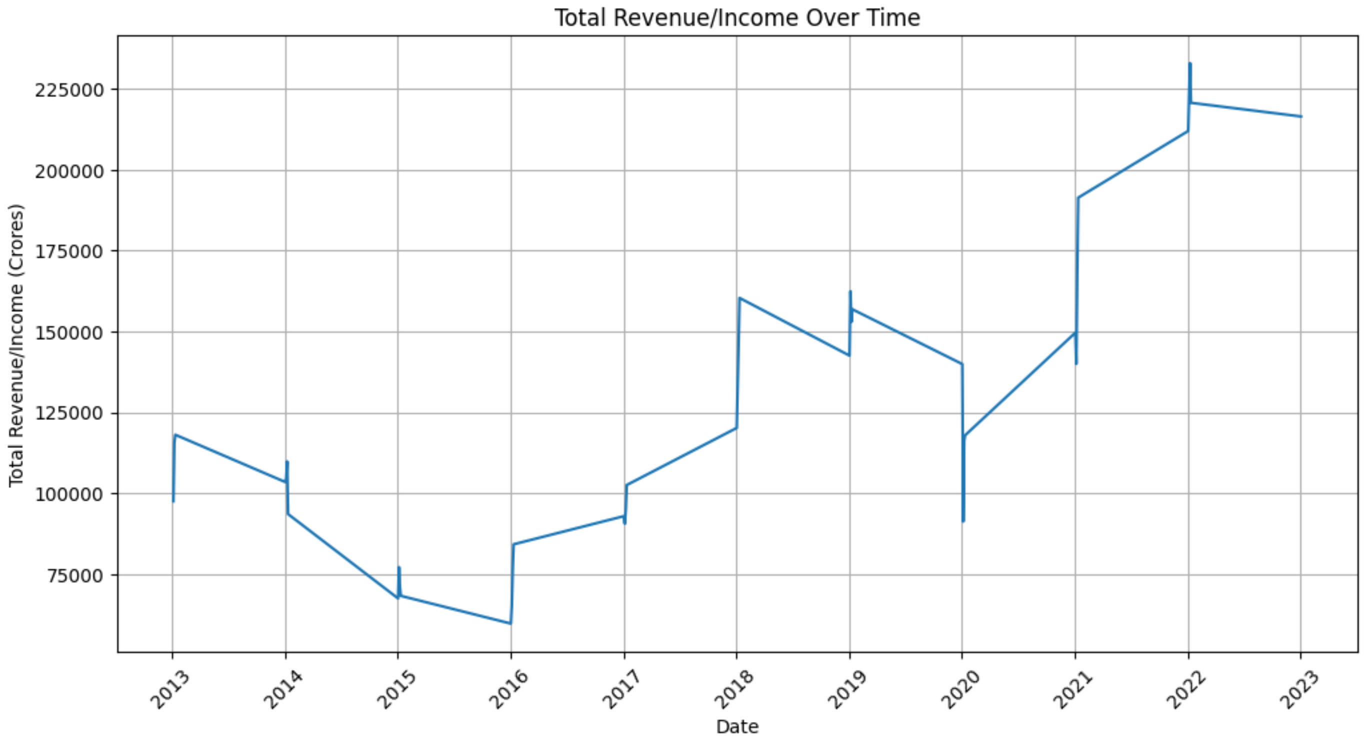
```
dataset.head()
```

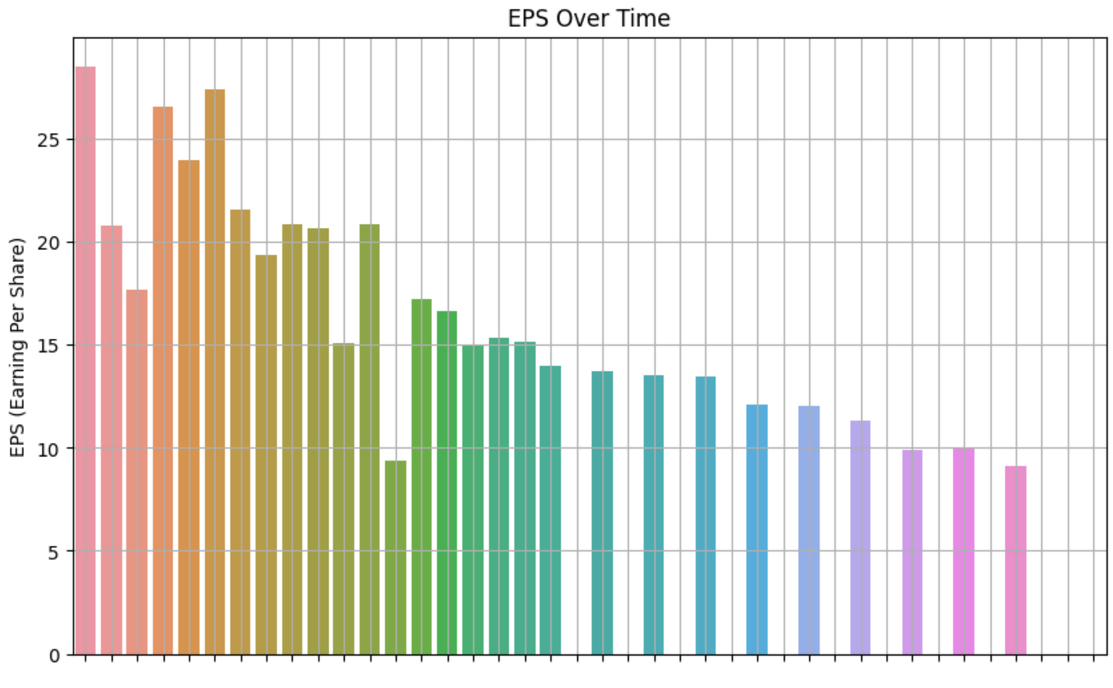
	Date	Total Revenue/Income(Cr)	Cost Of Revenue (Cr)	Gross Profit (Cr)	Total Operating Expense(Cr)	Operating Income/Profit(Cr)	EBITDA(Cr)	Reconciled Depreciation (Cr)	EBIT (Cr)	Interest Expense (Cr)	Income/Profit Before Tax(Cr)	Income Tax Expense (Cr)	Net Income From Continuing Operation(Cr)	Net Income(Cr)
0	2023-01-03	216376.0	148559.0	64386.0	177936.0	26984.0	39361.0	11456.0	-958.0	5819.0	24083.0	2787.0	21296.0	19299.0
1	2022-01-12	220592.0	149165.0	71427.0	185345.0	25060.0	36446.0	10187.0	26259.0	5201.0	23006.0	5266.0	17740.0	15792.0
2	2022-01-09	232863.0	162379.0	70484.0	201639.0	21494.0	32807.0	9730.0	14131.0	4554.0	20454.0	4867.0	15587.0	13656.0
3	2022-01-06	223113.0	150678.0	72435.0	190253.0	29051.0	31233.0	8946.0	29745.0	0.0	27236.0	7793.0	19443.0	17955.0
4	2022-01-03	211887.0	149521.0	62366.0	184010.0	23365.0	25967.0	8001.0	-3830.0	3556.0	22411.0	4390.0	18021.0	16203.0

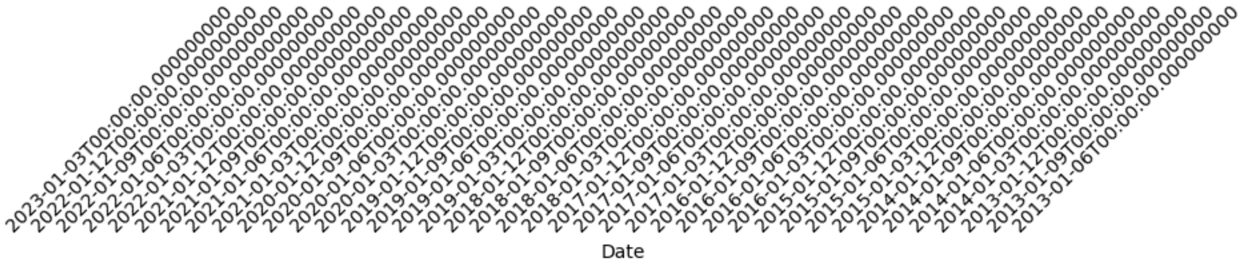
29	1.12.15	68261.0	49451.0	18810.0	10575.0	8235.0	10574.0	0.0	7345.00	921.0	9740.0	2363.0	0.0	7290.0

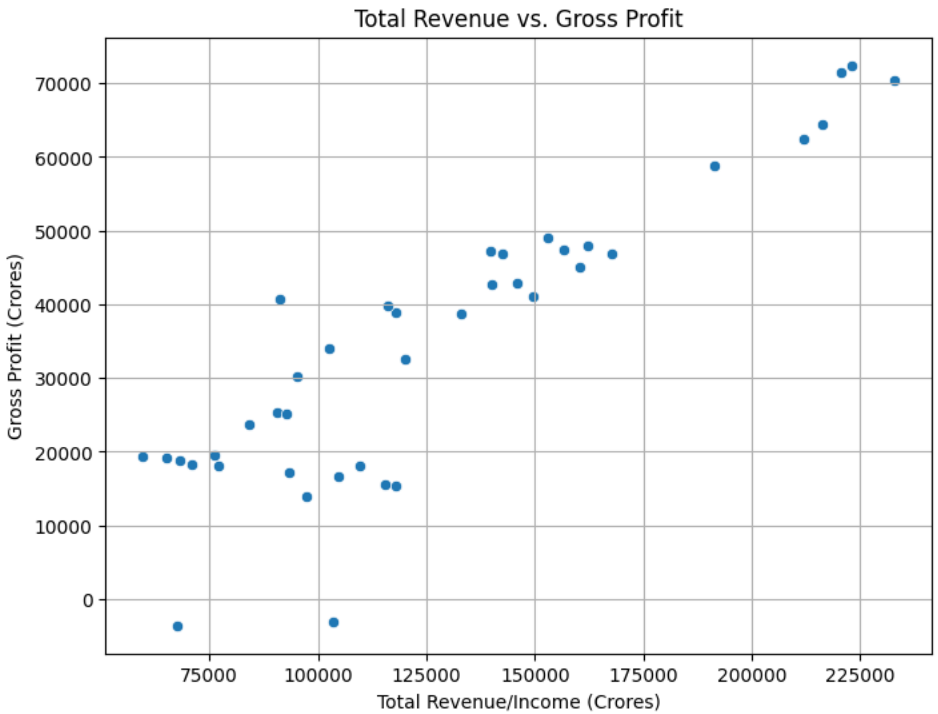
```
import seaborn as sns
```

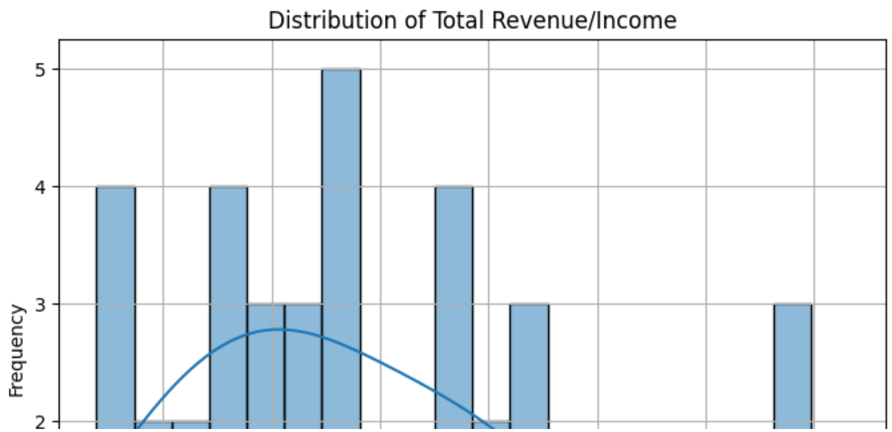
```
# Line Plot - Time Series Data
plt.figure(figsize=(12, 6))
plt.plot(dataset['Date'], dataset['Total Revenue/Income(Cr)'])
plt.xlabel('Date')
plt.ylabel('Total Revenue/Income (Crores)')
plt.title('Total Revenue/Income Over Time')
plt.xticks(rotation=45)
plt.grid(True)
plt.show()

# Bar Plot - Categorical Data
plt.figure(figsize=(10, 6))
sns.barplot(x='Date', y='EPS (Earning Per Share)', data=dataset)
plt.xlabel('Date')
plt.ylabel('EPS (Earning Per Share)')
plt.title('EPS Over Time')
plt.xticks(rotation=45)
plt.grid(True)
plt.show()

# Scatter Plot - Relationship between two numerical features
plt.figure(figsize=(8, 6))
sns.scatterplot(x='Total Revenue/Income(Cr)', y='Gross Profit (Cr)', data=dataset)
plt.xlabel('Total Revenue/Income (Crores)')
plt.ylabel('Gross Profit (Crores)')
plt.title('Total Revenue vs. Gross Profit')
plt.grid(True)
plt.show()

# Histogram - Distribution of a numerical feature
plt.figure(figsize=(8, 6))
sns.histplot(dataset['Total Revenue/Income(Cr)'], bins=20, kde=True)
plt.xlabel('Total Revenue/Income (Crores)')
plt.ylabel('Frequency')
plt.title('Distribution of Total Revenue/Income')
plt.grid(True)
plt.show()

# Box Plot - Distribution and Outliers of a numerical feature
plt.figure(figsize=(8, 6))
sns.boxplot(x=dataset['Total Revenue/Income(Cr)'])
plt.xlabel('Total Revenue/Income (Crores)')
plt.title('Box Plot of Total Revenue/Income')
plt.grid(True)
plt.show()

# Heatmap - Correlation between numerical features
correlation_matrix = dataset.drop(['Date'], axis=1).corr()
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
plt.title('Correlation Heatmap')
plt.show()
```

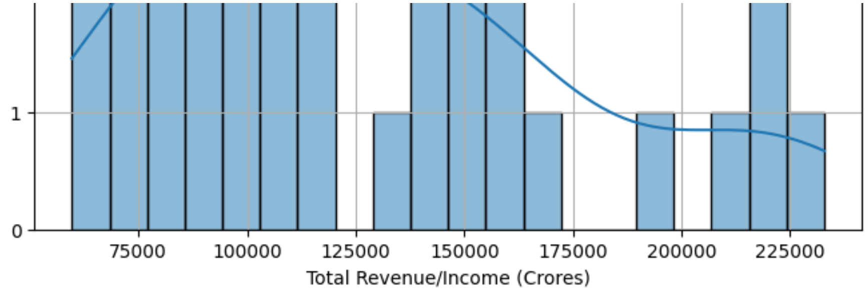

Total Revenue/Income Over Time
Total Revenue/Income (Crores)
225000
200000
175000
150000
125000
100000
75000
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
Date

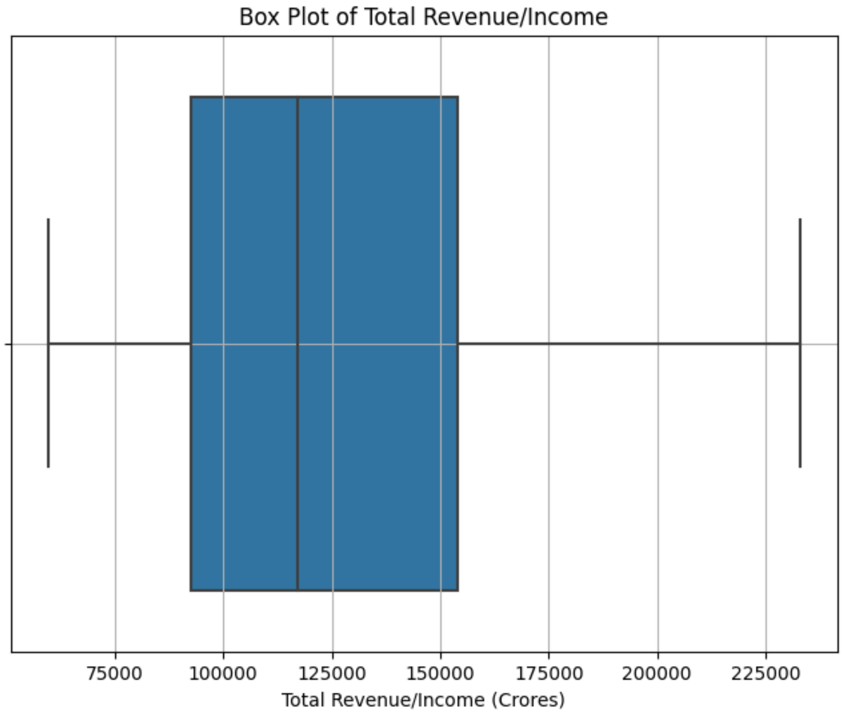

EPS Over Time
EPS (Earning Per Share)
25
20
15
10
5
0

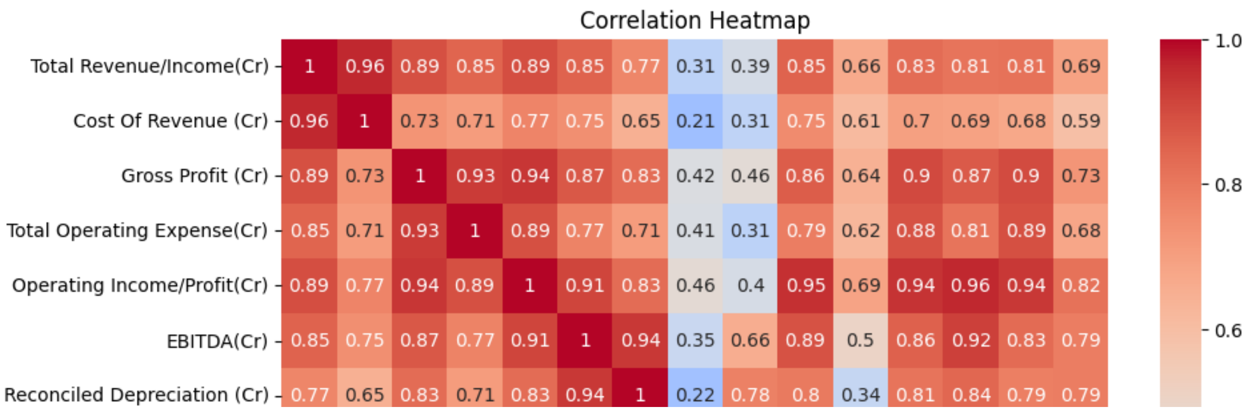

2023-01-03T00:00:00.000000000
2022-01-12T00:00:00.000000000
2022-01-09T00:00:00.000000000
2022-01-06T00:00:00.000000000
2022-01-03T00:00:00.000000000
2021-01-12T00:00:00.000000000
2021-01-09T00:00:00.000000000
2021-01-06T00:00:00.000000000
2021-01-03T00:00:00.000000000
2020-01-12T00:00:00.000000000
2020-01-09T00:00:00.000000000
2020-01-06T00:00:00.000000000
2020-01-03T00:00:00.000000000
2019-01-12T00:00:00.000000000
2019-01-09T00:00:00.000000000
2019-01-06T00:00:00.000000000
2019-01-03T00:00:00.000000000
2018-01-12T00:00:00.000000000
2018-01-09T00:00:00.000000000
2018-01-06T00:00:00.000000000
2018-01-03T00:00:00.000000000
2017-01-12T00:00:00.000000000
2017-01-09T00:00:00.000000000
2017-01-06T00:00:00.000000000
2017-01-03T00:00:00.000000000
2016-01-12T00:00:00.000000000
2016-01-09T00:00:00.000000000
2016-01-06T00:00:00.000000000
2016-01-03T00:00:00.000000000
2015-01-12T00:00:00.000000000
2015-01-09T00:00:00.000000000
2015-01-06T00:00:00.000000000
2015-01-03T00:00:00.000000000
2014-01-12T00:00:00.000000000
2014-01-09T00:00:00.000000000
2014-01-06T00:00:00.000000000
2014-01-03T00:00:00.000000000
2013-01-12T00:00:00.000000000
2013-01-09T00:00:00.000000000
2013-01-06T00:00:00.000000000
Date

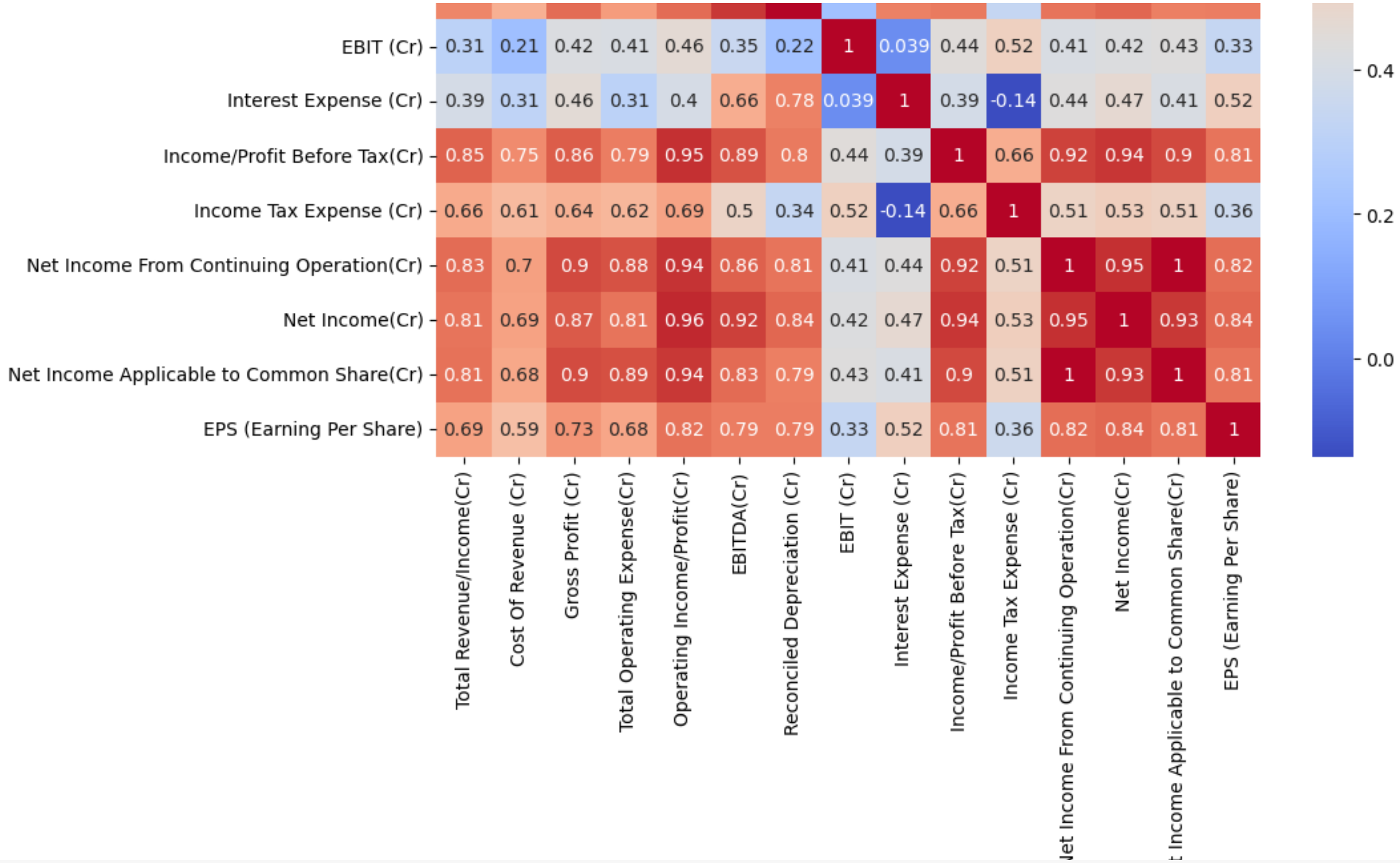

Total Revenue vs. Gross Profit
Gross Profit (Crores)
70000
60000
50000
40000
30000
20000
10000
0
75000
100000
125000
150000
175000
200000
225000
Total Revenue/Income (Crores)


Distribution of Total Revenue/Income
Frequency
5
4
3
2


1
0
75000
100000
125000
150000
175000
200000
225000
Total Revenue/Income (Crores)


Box Plot of Total Revenue/Income
75000
100000
125000
150000
175000
200000
225000
Total Revenue/Income (Crores)


Correlation Heatmap
Total Revenue/Income(Cr) 1 0.96 0.89 0.85 0.89 0.85 0.77 0.31 0.39 0.85 0.66 0.83 0.81 0.81 0.69
Cost Of Revenue (Cr) 0.96 1 0.73 0.71 0.77 0.75 0.65 0.21 0.31 0.75 0.61 0.7 0.69 0.68 0.59
Gross Profit (Cr) 0.89 0.73 1 0.93 0.94 0.87 0.83 0.42 0.46 0.86 0.64 0.9 0.87 0.9 0.73
Total Operating Expense(Cr) 0.85 0.71 0.93 1 0.89 0.77 0.71 0.41 0.31 0.79 0.62 0.88 0.81 0.89 0.68
Operating Income/Profit(Cr) 0.89 0.77 0.94 0.89 1 0.91 0.83 0.46 0.4 0.95 0.69 0.94 0.96 0.94 0.82
EBITDA(Cr) 0.85 0.75 0.87 0.77 0.91 1 0.94 0.35 0.66 0.89 0.5 0.86 0.92 0.83 0.79
Reconciled Depreciation (Cr) 0.77 0.65 0.83 0.71 0.83 0.94 1 0.22 0.78 0.8 0.34 0.81 0.84 0.79 0.79
1.0
0.8
0.6




```
#extract additional time-based features
dataset['Year'] = dataset['Date'].dt.year
dataset['Month'] = dataset['Date'].dt.month
dataset['Day'] = dataset['Date'].dt.day
dataset['DayOfWeek'] = dataset['Date'].dt.dayofweek
```

```
dataset = dataset.drop(['Date', 'Net Income Applicable to Common Share(Cr)','EBIT (Cr)'], axis=1)
```

```
dataset.head()
```

	Total Revenue/Income(Cr)	Cost Of Revenue (Cr)	Gross Profit (Cr)	Total Operating Expense(Cr)	Operating Income/Profit(Cr)	EBITDA(Cr)	Reconciled Depreciation (Cr)	Interest Expense (Cr)	Income/Profit Before Tax(Cr)	Income Tax Expense (Cr)	Net Income From Continuing Operation(Cr)	Net Income(Cr)	EPS (Earning Per Share)

```
# Independent and dependent variable separation
X = dataset.drop(['EPS (Earning Per Share)'], axis=1)
y = dataset[['EPS (Earning Per Share)']]
```

3	223113.0	150678.0	72435.0	190253.0	29051.0	31233.0	8946.0	0.0	27236.0	7793.0	19443.0	17955.0	26.54

```
X.head()
```

	Total Revenue/Income(Cr)	Cost Of Revenue (Cr)	Gross Profit (Cr)	Total Operating Expense(Cr)	Operating Income/Profit(Cr)	EBITDA(Cr)	Reconciled Depreciation (Cr)	Interest Expense (Cr)	Income/Profit Before Tax(Cr)	Income Tax Expense (Cr)	Net Income From Continuing Operation(Cr)	Net Income(Cr)
0	216376.0	148559.0	64386.0	177936.0	26984.0	39361.0	11456.0	5819.0	24083.0	2787.0	21296.0	19299.0
1	220592.0	149165.0	71427.0	185345.0	25060.0	36446.0	10187.0	5201.0	23006.0	5266.0	17740.0	15792.0
2	232863.0	162379.0	70484.0	201639.0	21494.0	32807.0	9730.0	4554.0	20454.0	4867.0	15587.0	13656.0
3	223113.0	150678.0	72435.0	190253.0	29051.0	31233.0	8946.0	0.0	27236.0	7793.0	19443.0	17955.0
4	211887.0	149521.0	62366.0	184010.0	23365.0	25967.0	8001.0	3556.0	22411.0	4390.0	18021.0	16203.0

```
y.head()
```

	EPS (Earning Per Share)
0	28.52
1	20.78
2	17.68
3	26.54
4	23.95

```
#Splitting the data into training and testing set
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(X,y, test_size = 0.3,random_state = 42 )
```

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
# Build the neural network architecture
```

```
model = Sequential()
model.add(Dense(64, activation='relu', input_shape=(X_train.shape[1],)))
model.add(Dense(32, activation='relu'))
model.add(Dense(1))
```

```
# Compile the model
model.compile(optimizer='adam', loss='mean_squared_error')
```

```
history = model.fit(X_train, y_train, epochs=100, batch_size=16, validation_split=0.25)
```

```
Epoch 1/100
2/2 [==============================] - 6s 202ms/step - loss: 241.5291 - val_loss: 189.4535
Epoch 2/100
2/2 [==============================] - 0s 30ms/step - loss: 237.1013 - val_loss: 187.8303
Epoch 3/100
2/2 [==============================] - 0s 33ms/step - loss: 233.0491 - val_loss: 186.2600
Epoch 4/100
2/2 [==============================] - 0s 29ms/step - loss: 228.6842 - val_loss: 184.6399
Epoch 5/100
2/2 [==============================] - 0s 27ms/step - loss: 224.5756 - val_loss: 183.0553
Epoch 6/100
2/2 [==============================] - 0s 27ms/step - loss: 220.9641 - val_loss: 181.3888
Epoch 7/100
2/2 [==============================] - 0s 31ms/step - loss: 216.7635 - val_loss: 179.6370
Epoch 8/100
2/2 [==============================] - 0s 26ms/step - loss: 213.2123 - val_loss: 177.8326
Epoch 9/100
2/2 [==============================] - 0s 40ms/step - loss: 208.5347 - val_loss: 175.9954
Epoch 10/100
2/2 [==============================] - 0s 47ms/step - loss: 204.4821 - val_loss: 174.0408
Epoch 11/100
2/2 [==============================] - 0s 60ms/step - loss: 200.1021 - val_loss: 172.0071
Epoch 12/100
2/2 [==============================] - 0s 50ms/step - loss: 195.2887 - val_loss: 169.8908
Epoch 13/100
2/2 [==============================] - 0s 46ms/step - loss: 191.0541 - val_loss: 167.6914
Epoch 14/100
2/2 [==============================] - 0s 46ms/step - loss: 185.7985 - val_loss: 165.4230
Epoch 15/100
2/2 [==============================] - 0s 50ms/step - loss: 180.7872 - val_loss: 162.9890
Epoch 16/100
2/2 [==============================] - 0s 50ms/step - loss: 176.3896 - val_loss: 160.4767
Epoch 17/100
2/2 [==============================] - 0s 61ms/step - loss: 170.8700 - val_loss: 157.9184
Epoch 18/100
2/2 [==============================] - 0s 62ms/step - loss: 165.8575 - val_loss: 155.2923
Epoch 19/100
2/2 [==============================] - 0s 41ms/step - loss: 160.3849 - val_loss: 152.6503
Epoch 20/100
2/2 [==============================] - 0s 42ms/step - loss: 154.7347 - val_loss: 149.9332
Epoch 21/100
2/2 [==============================] - 0s 42ms/step - loss: 149.8246 - val_loss: 147.1468
Epoch 22/100
2/2 [==============================] - 0s 42ms/step - loss: 144.2040 - val_loss: 144.3775
Epoch 23/100
2/2 [==============================] - 0s 59ms/step - loss: 138.2328 - val_loss: 141.5788
Epoch 24/100
2/2 [==============================] - 0s 50ms/step - loss: 132.2071 - val_loss: 138.6650
```

```
Epoch 25/100
2/2 [==============================] - 0s 59ms/step - loss: 127.0774 - val_loss: 135.5507
Epoch 26/100
2/2 [==============================] - 0s 47ms/step - loss: 120.8653 - val_loss: 132.4118
Epoch 27/100
2/2 [==============================] - 0s 73ms/step - loss: 114.2405 - val_loss: 129.2283
Epoch 28/100
2/2 [==============================] - 0s 58ms/step - loss: 108.3221 - val_loss: 125.8936
Epoch 29/100
2/2 [==============================] - 0s 62ms/step - loss: 101.8658 - val_loss: 122.4131
```

```
# Make predictions on the test data
y_pred = model.predict(X_test)
```

```
1/1 [==============================] - 0s 76ms/step
```

```
print(y_test.shape, y_pred.shape)
```

```
(12, 1) (12, 1)
```

```
#y_test_reshaped = y_test.iloc[:, 0].values.reshape(-1, 1)
```

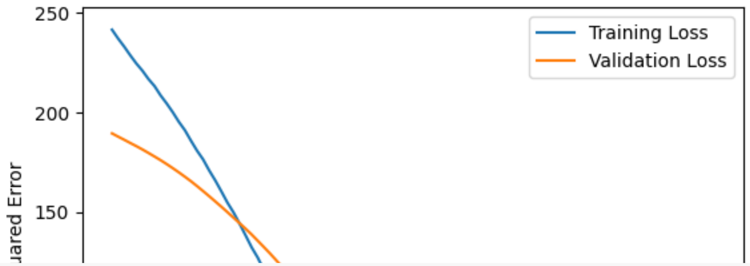
```
from sklearn.metrics import r2_score, mean_squared_error
mse = mean_squared_error(y_test , y_pred)
print(f"Mean Squared Error: {mse:.4f}")
```

```
Mean Squared Error: 28.5282
```

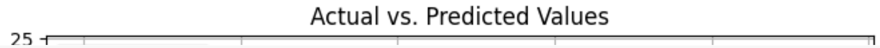
```
# Calculate R-squared
r2 = r2_score(y_test , y_pred)
print(f"R-squared: {r2:.4f}")
```

```
R-squared: 0.6576
```

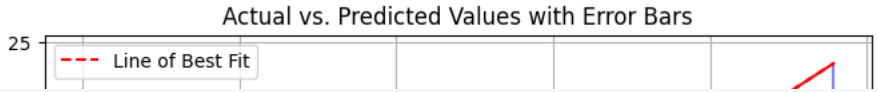
```
# Plot the training and validation loss curves
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Mean Squared Error')
plt.legend()
plt.show()
```




```
# Assuming y_test has shape (8,) and y_pred has shape (8, 1)
# Reshape y_test to (8, 1)
y_test_reshaped = y_test.values.reshape(-1, 1)

# Create the scatter plot
plt.figure(figsize=(8, 6))
plt.scatter(y_test_reshaped, y_pred, color='blue', label='Predicted', alpha=0.7)
plt.scatter(y_test_reshaped, y_test_reshaped, color='red', label='Actual', alpha=0.7)

plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.title('Actual vs. Predicted Values')
plt.legend()
plt.grid(True)
plt.show()

# Plotting the graph
plt.figure(figsize=(10, 6))

# Data points for x-axis (e.g., assuming 100 data points)
data_points = range(len(y_test))

# Plot actual EPS
plt.plot(data_points, y_test, label='Actual EPS', marker='o')

# Plot predicted EPS
plt.plot(data_points, y_pred, label='Predicted EPS', marker='o')

# Add labels and legend
plt.xlabel('Data Points')
plt.ylabel('EPS')
plt.title('Actual vs. Predicted EPS')
plt.legend()
plt.grid(True)

# Show the plot
plt.show()
```




```
# Assuming y_test has shape (8,) and y_pred has shape (8, 1)

y_test_reshaped = y_test.iloc[:, 0].values.reshape(-1, 1)

# Calculate absolute errors
absolute_errors = np.abs(y_test_reshaped - y_pred)

# Flatten the absolute_errors array to make it 1D
absolute_errors = absolute_errors.flatten()

# Create the scatter plot with error bars
plt.figure(figsize=(8, 6))
plt.errorbar(y_test_reshaped, y_pred.flatten(), yerr=absolute_errors, fmt='o', color='blue', alpha=0.5)

# Plot the line of best fit (y = x)
plt.plot(y_test_reshaped, y_test_reshaped, color='red', linestyle='--', label='Line of Best Fit')

plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.title('Actual vs. Predicted Values with Error Bars')
plt.legend()
plt.grid(True)
plt.show()
```




```
# Assuming y_test and y_pred are numpy arrays
y_test = y_test
y_pred = y_pred

# Print the actual and predicted values for the "gross_profit" column
print("Actual EPS:")
print(y_test.values)

print("Predicted EPS:")
print(y_pred)
```

```
Actual EPS:
[[ 0.  ]
 [15.33]
 [14.94]
 [12.09]
 [23.95]
 [ 9.39]
 [ 0.  ]
 [ 0.  ]
 [ 0.  ]
 [21.58]
 [ 0.  ]
 [20.67]]
Predicted EPS:
[[ 6.565065 ]
 [ 9.005335 ]
 [10.417762 ]
 [ 4.199317 ]
 [21.216787 ]
 [ 8.043324 ]
 [ 4.0974   ]
 [ 5.5555553]
 [ 7.6551933]
 [14.762969 ]
 [ 2.5634632]
 [17.851946 ]]
```