# Data Visualization Techniques Lab

**Pulla Manoj Kumar**
Assistant Professor, CSD, ACEEC Hyderabad

## Time-series analysis – Stock Market

## STEP 1: Import Required Libraries

```python
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
import seaborn as sns
from pandas.plotting import lag_plot

from statsmodels.tsa.arima.model import ARIMA
from statsmodels.tsa.stattools import adfuller

from sklearn.metrics import mean_squared_error

import warnings
warnings.filterwarnings('ignore')
```

## STEP 2: Upload and Load Dataset

```python
df = pd.read_csv("Tesla_stock_price.csv")
df.head(10)
```

## STEP 3: Visualizing Closing Price

```python
plt.figure(figsize=(10,5))
df['Close'].plot()
plt.title("Tesla Closing Price")
plt.xlabel("Date")
plt.ylabel("Price")
plt.show()
```

## STEP 4: Cumulative Returns Visualization

```
returns = df['Close'].pct_change()

cumulative_returns = returns.cumsum()

plt.figure(figsize=(10,5))
cumulative_returns.plot()
plt.title("Tesla Cumulative Returns")
plt.xlabel("Date")
plt.ylabel("Cumulative Return")
plt.show()
```

## STEP 5: Autocorrelation Plot

```
plt.figure(figsize=(10,5))
lag_plot(df['Open'], lag=5)
plt.title("Tesla Autocorrelation Plot")
plt.show()
```

## STEP 6: Stationarity Check using ADF Test

```
def adf_test(series):
    result = adfuller(series.dropna())

    print("ADF Statistic:", result[0])
    print("p-value:", result[1])

    if result[1] <= 0.05:
        print("Data is Stationary")
    else:
        print("Data is Non-Stationary")

adf_test(df['Open'])
```

## STEP 7: Train and Test Split

```
train_size = int(len(df) * 0.8)

train_data = df.iloc[:train_size]
test_data = df.iloc[train_size:]
```

## STEP 8: Training and Testing Visualization

```
plt.figure(figsize=(12,6))

plt.plot(train_data['Open'], color='blue', label='Training Data')
plt.plot(test_data['Open'], color='green', label='Testing Data')

plt.title("Training vs Testing Data")
plt.legend()
plt.show()
```

## STEP 9: Define SMAPE Function

```
def smape_kun(y_true, y_pred):
    return np.mean(
        (np.abs(y_pred - y_true) * 200) /
        (np.abs(y_pred) + np.abs(y_true))
    )
```

## STEP 10: ARIMA Model Prediction (Walk-Forward Validation)

```
train_ar = train_data['Open'].values
test_ar = test_data['Open'].values

history = list(train_ar)
predictions = []

for t in range(len(test_ar)):

    model = ARIMA(history, order=(5,1,0))
    model_fit = model.fit()

    output = model_fit.forecast()
    yhat = output[0]

    predictions.append(yhat)
    history.append(test_ar[t])
```

## STEP 11: Error Calculation

```
mse_error = mean_squared_error(test_ar, predictions)
print("Testing Mean Squared Error:", mse_error)


smape_error = smape_kun(test_ar, predictions)
print("SMAPE Error:", smape_error)
```

## STEP 12: Final Prediction Visualization

```
plt.figure(figsize=(12,6))

plt.plot(test_data.index, test_data['Open'],
        color='red', label='Actual Price')

plt.plot(test_data.index, predictions,
        color='green', linestyle='dashed', marker='o',
        label='Predicted Price')

plt.title("Tesla Price Prediction using ARIMA")
plt.legend()
plt.show()
```