**Machine Learning Lab**

**Exercise :3**

**Fashion Trends Online (FTO) is an e-commerce company that sells women apparel. It is observed that**

**10% of their customers return the items purchased by them for many reasons (such as size, color, and**

**material mismatch). On a specific day, 20 customers purchased items from FTO. Calculate:**

**Probability that exactly 5 customers will return the items. To visualize how the PMF varies with increasing number of successful trials, we will create a list of all possible number of successes (0 to 20) and corresponding PMF values and draw a bar plot.**

**HINT:**

**IMPORT SCIPY**

**from scipy import stats**

**The function stats.binom.pmf() calculates PMF for binomial distribution and takes three parameters:**

**(a) Expected number of successful trials (5)**

**(b) Total number of trials (20)**

**(c) The probability of success (0.1)**

**2. Probability that a maximum of 5 customers will return the items.**

**Hint:The class stats.binom.cdf() computes the CDF for binomial distribution. In this case the cumulative distribution function returns the probability that a maximum of 5 customers will return items.**

**3.Probability that more than 5 customers will return the items purchased by them.**

**Hint: 1 - stats.binom.cdf(5, 20, 0.1)**

**4. Average number of customers who are likely to return the items and the variance and the**

**standard deviation of the number of returns.**

**Hint: Average of a binomial distribution is given by n \* p**

**(b) Variance of the binomial distribution is given by n \* p \* (1 − p)**

**mean, var = stats.binom.stats(20, 0.1)**

from scipy import stats

import matplotlib.pyplot as plt

n=20

r=list(range(n+1))

p=0.1

pmf\_values=stats.binom.pmf(r,n,p)

print('The probability that exactly 5 customers will return the product is %.3f' % stats.binom.pmf(5,n,p))

print('The probability that maximum of 5 customers will return the product is %0.3f' % stats.binom.cdf(5,n,p))

print('The probability that more than 5 customers will return the product is %0.3f'% (1-(stats.binom.cdf(5,n,p))))

mean,var=stats.binom.stats(n,p)

print('Average number of customers who are likely to return the items ',mean)

print('The variance is',var)

print('Standard deviation %0.3f' % var\*\*0.5)

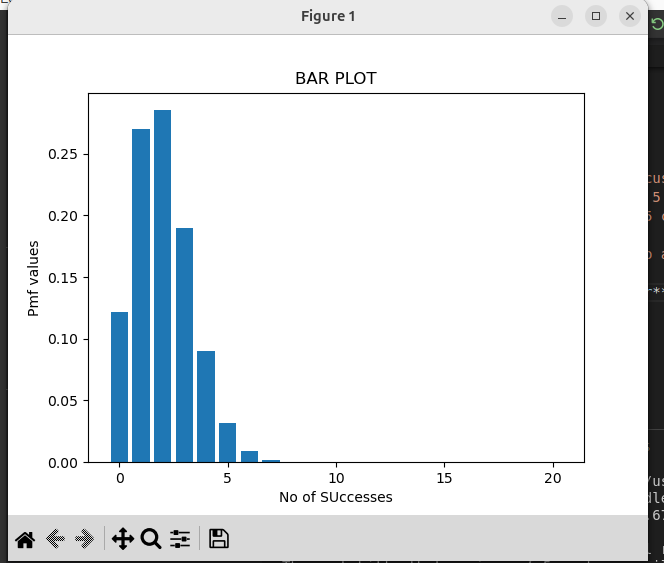
plt.bar(r,pmf\_values)

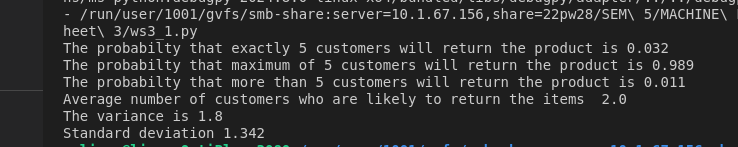
plt.xlabel("No of SUccesses")

plt.ylabel("Pmf values")

plt.title('BAR PLOT')

plt.show()





**Poisson Distribution**

5**. The number of calls arriving at a call center follows a Poisson distribution at 10 calls per hour.**

**1. Calculate the probability that the number of calls will be maximum 5.**

**Hint:**

**stats.poisson.cdf(5, 10)**

**2. Calculate the probability that the number of calls over a 3-hour period will exceed 30.**

**Hint:1 - stats.poisson.cdf(30, 30)**

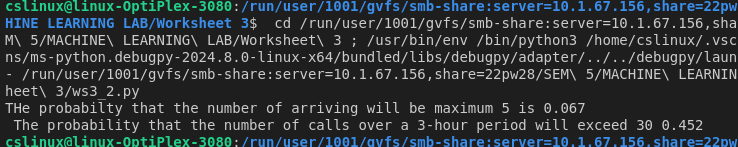
from scipy import stats

import matplotlib.pyplot as plt

print('THe probabilty that the number of arriving will be maximum 5 is %0.3f' % stats.poisson.cdf(5,10))

print(' The probability that the number of calls over a 3-hour period will exceed 30 %0.3f' % (1-stats.poisson.cdf(30,30)))

OUTPUT:



**Exponential Distribution**

The time-to-failure of an avionic system follows an exponential distribution with a mean time between

failures (MTBF) of 1000 hours. Calculate

1. The probability that the system will fail before 1000 hours.

2. The probability that it will not fail up to 2000 hours.

3. The time by which 10% of the system will fail (i.e., calculate P10 life)

Hint: stats.expon.cdf()

import numpy as np

from scipy.stats import expon

**# Given MTBF**

**MTBF = 1000**

**lambda\_ = 1 / MTBF**

**# 1. Probability that the system will fail before 1000 hours**

**prob\_fail\_1000 = expon.cdf(1000, scale=1000)**

**print(f"Probability of failure before 1000 hours: {prob\_fail\_1000:.4f}")**

**# 2. Probability that the system will not fail up to 2000 hours**

**prob\_not\_fail\_2000 = 1 - expon.cdf(2000, scale=1000)**

**print(f"Probability of not failing up to 2000 hours: {prob\_not\_fail\_2000:.4f}")**

**# 3. Time by which 10% of the system will fail (P10 life)**

**P10\_life = expon.ppf(0.10,scale=1000)**

**print(f"Time by which 10% of the system will fail (P10 life): {P10\_life:.2f} hours")**

**OUTPUT:**

**Probability of failure before 1000 hours: 0.6321**

**Probability of not failing up to 2000 hours: 0.1353**

**Time by which 10% of the system will fail (P10 life): 105.36 hours**

**Normal Distribution**

Dataset:

https://drive.google.com/file/d/1Z8Z0VOkYrGsgtQSA4QQQOO\_GZqUE4Qrq/view?usp=drive\_link

The dataset contains daily Open and Close price along with daily High and Low prices, Total Trade

Quantity, and Turnover (Lacs). Our discussion will involve only close price. The daily returns of a stock

are calculated as the change in close prices with respect to the close price of yesterday.

1. plot BEML stock close price trend.
2. Using the given formula the gain can be calculated as a percentage change in close price, from the previous day’s close price.
3. 
4. Plot the gain chart and check whether its normally distributed
5. What if a short-term (intraday) investor is interested in understanding the following characteristics about these stocks:
6. What is the expected daily rate of return of these stocks?
7. Which stocks have higher risk or volatility as far as daily returns are concerned?
8. What is the expected range of return for 95% confidence interval?

**# -\*- coding: utf-8 -\*-**

**"""**

**Created on Wed Jul 24 09:53:37 2024**

**@author: 22pw28**

**"""**

**import pandas as pd**

**import numpy as np**

**import matplotlib.pyplot as plt**

**import seaborn as sns**

**from scipy.stats import norm**

**# Assuming you have a DataFrame 'df' with a 'Close' column for BEML stock prices**

**df = pd.read\_csv("Z:\SEM 5\MACHINE LEARNING LAB\Worksheet 3\BEML.csv") # Load your dataset**

**# Plotting the close price trend**

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

**plt.plot(df['Close'], label='BEML Close Price')**

**plt.title('BEML Stock Close Price Trend')**

**plt.xlabel('Date')**

**plt.ylabel('Close Price')**

**plt.legend()**

**plt.show()**

**# Calculating daily returns**

**df['Daily Return'] = df['Close'].pct\_change()**

**# Plotting the gain chart**

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

**plt.plot(df['Daily Return'], label='Daily Returns')**

**plt.title('BEML Stock Daily Returns')**

**plt.xlabel('Date')**

**plt.ylabel('Daily Return')**

**plt.legend()**

**plt.show()**

**# Checking if the daily returns are normally distributed**

**sns.histplot(df['Daily Return'].dropna(), kde=True, stat="density", linewidth=0)**

**plt.title('Distribution of Daily Returns')**

**plt.xlabel('Daily Return')**

**plt.ylabel('Density')**

**plt.show()**

**# Expected daily rate of return**

**expected\_daily\_return = df['Daily Return'].mean()**

**print(f"Expected Daily Rate of Return: {expected\_daily\_return:.4f}")**

**# Volatility (standard deviation of daily returns)**

**volatility = df['Daily Return'].std()**

**print(f"Volatility (Standard Deviation of Daily Returns): {volatility:.4f}")**

**# 95% confidence interval for daily returns**

**confidence\_interval = norm.interval(0.95, loc=expected\_daily\_return, scale=volatility)**

**print(f"95% Confidence Interval for Daily Returns: {confidence\_interval}")**

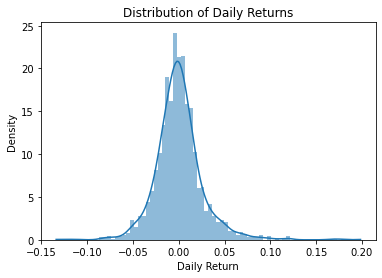
**OUTPUT:**

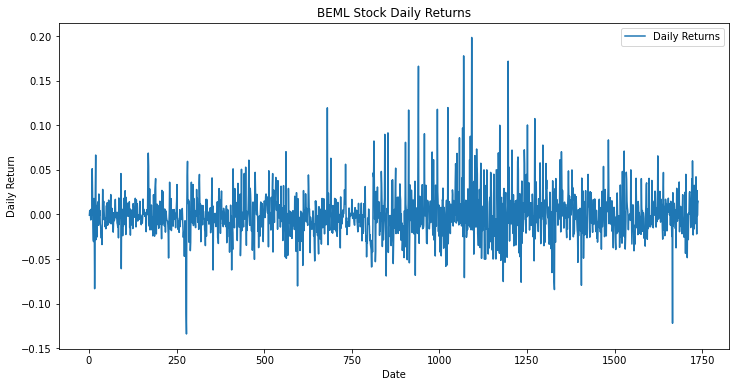
**Expected Daily Rate of Return: 0.0003**

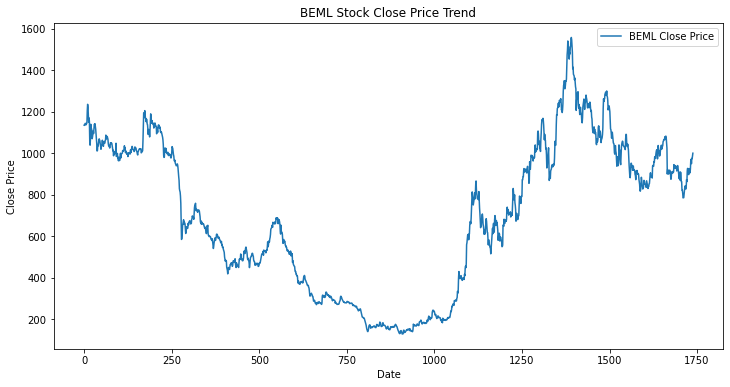
**Volatility (Standard Deviation of Daily Returns): 0.0264**

**95% Confidence Interval for Daily Returns: (-0.051532729680601415, 0.05207422583871588)**

**fIGURES:**

****

****

****

**Polynomial regression**

y = 13x2 + 2x + 7

Genearte a random number for x (x = np.array(7 \* np.random.rand(100, 1) - 3) ).

(I)**Try to fit the data with a linear model. Plot the data points and the linear line.** Calculate the performance of the model in terms of mean square error, root mean square error and r2 score.

**(ii)Try Polynomial Regression with degree 2**

poly\_features**=**PolynomialFeatures(degree **=**2, include\_bias**=**False)

x\_poly**=**poly\_features.fit\_transform(x1)

Plot the quadratic equation obtained.

Calculate the performance of the model obtained by Polynomial Regression.

Compare the performance of polynomial regression and linear regression model for the given quadratic equation.

**Hint :**

# Importing the libraries

import numpy as np

import matplotlib.pyplot as plt

from sklearn.linear\_model import LinearRegression

from sklearn.preprocessing import PolynomialFeatures

from sklearn.metrics import mean\_squared\_error, r2\_score

# Importing the libraries

import numpy as np

import matplotlib.pyplot as plt

from sklearn.linear\_model import LinearRegression

from sklearn.preprocessing import PolynomialFeatures

from sklearn.metrics import mean\_squared\_error, r2\_score

# Generating random data

np.random.seed(0)

x = np.array(7 \* np.random.rand(100, 1) - 3)

y = 13 \* x\*\*2 + 2 \* x + 7 + np.random.randn(100, 1) \* 10

# Linear Regression

linear\_regressor = LinearRegression()

linear\_regressor.fit(x, y)

y\_pred\_linear = linear\_regressor.predict(x)

# Plotting Linear Regression

plt.scatter(x, y, color='blue', label='Data points')

plt.plot(x, y\_pred\_linear, color='red', label='Linear fit')

plt.title('Linear Regression')

plt.xlabel('x')

plt.ylabel('y')

plt.legend()

plt.show()

# Performance of Linear Regression

mse\_linear = mean\_squared\_error(y, y\_pred\_linear)

rmse\_linear = np.sqrt(mse\_linear)

r2\_linear = r2\_score(y, y\_pred\_linear)

print(f"Linear Regression - MSE: {mse\_linear}, RMSE: {rmse\_linear}, R2 Score: {r2\_linear}")

# Polynomial Regression (degree 2)

poly\_features = PolynomialFeatures(degree=2, include\_bias=False)

x\_poly = poly\_features.fit\_transform(x)

poly\_regressor = LinearRegression()

poly\_regressor.fit(x\_poly, y)

y\_pred\_poly = poly\_regressor.predict(x\_poly)

# Plotting Polynomial Regression

plt.scatter(x, y, color='blue', label='Data points')

plt.scatter(x, y\_pred\_poly, color='green', label='Polynomial fit')

plt.title('Polynomial Regression (degree 2)')

plt.xlabel('x')

plt.ylabel('y')

plt.legend()

plt.show()

# Performance of Polynomial Regression

mse\_poly = mean\_squared\_error(y, y\_pred\_poly)

rmse\_poly = np.sqrt(mse\_poly)

r2\_poly = r2\_score(y, y\_pred\_poly)

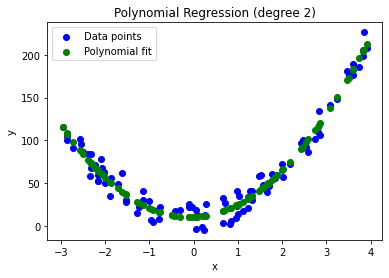
print(f"Polynomial Regression - MSE: {mse\_poly}, RMSE: {rmse\_poly}, R2 Score: {r2\_poly}")

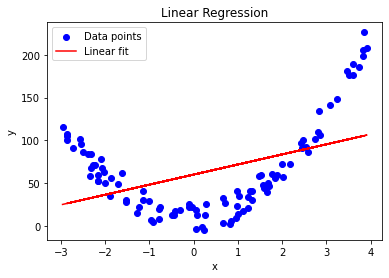
# Comparison

print("\nComparison:")

print(f"Linear Regression - MSE: {mse\_linear}, RMSE: {rmse\_linear}, R2 Score: {r2\_linear}")

print(f"Polynomial Regression - MSE: {mse\_poly}, RMSE: {rmse\_poly}, R2 Score: {r2\_poly}")





**OUTPUT:**

**Linear Regression - MSE: 2310.8719440883233, RMSE: 48.071529454431996, R2 Score: 0.19601179017197823**

**Polynomial Regression - MSE: 97.35576723414219, RMSE: 9.866902616026074, R2 Score: 0.9661284177969058**

**Comparison:**

**Linear Regression - MSE: 2310.8719440883233, RMSE: 48.071529454431996, R2 Score: 0.19601179017197823**

**Polynomial Regression - MSE: 97.35576723414219, RMSE: 9.866902616026074, R2 Score: 0.9661284177969058**