**COMPUTATIONAL STATISTICS LABORATORY**

**[As per Choice Based Credit System (CBCS) scheme]**

**(Effective from the academic year 2024 -2025) SEMESTER – III**

|  |  |  |  |
| --- | --- | --- | --- |
| **Subject Code** | **BCBL504** | **CIE Marks** | **50** |
| **Number of Lecture** | **40H** | **SIE Marks** | **50** |
| **Total Number of Lab** | **20H** | **Exam Hours** | **-** |
|  | **CREDITS – 0** | **1** |  |

**Course Outcome:**

|  |  |
| --- | --- |
| **CO 1** | **Design the experiment for the given problem using statistical methods.** |
| **CO 2** | **Develop the solution for the given real world problem using statistical techniques.** |
| **CO 3** | **Analyze the results and produce substantial written documentation.** |
| **CO 4** | **Analyze I/O management and file system, concepts of protection and security.** |

**PART A**

**1)Program on data wrangling: Combining and merging datasets, Reshaping and Pivoting**

**2) Program on Data Transformation: String Manipulation, Regular Expressions**

**3) Program on Time series: GroupBy Mechanics to display in data vector, multivariate time series and forecasting formats**

**4) Program to measure central tendency and measures of dispersion: Mean, Median, Mode, Standard Deviation, Variance, Mean deviation and Quartile deviation for a frequency distribution/data.**

**5) Program to perform cross validation for a given dataset to measure Root Mean Squared Error (RMSE), Mean Absolute Error (MAE) and R2 Error using Validation Set, Leave One Out Cross-Validation(LOOCV) and K-fold Cross-Validation approaches**

**6) Program to display Normal, Binomial Poisson, Bernoulli disrtibutions for a given frequency distribution and analyze the results.**

**7) Program to implement one sample, two sample and paired sample t-tests for a sample data and analyse the results.**

**8) Program to implement One-way and Two-way ANOVA tests and analyze the results**

**9) Program to implement correlation, rank correlation and regression and plot x-y plot and heat maps of correlation matrices**

**10) Program to implement PCA for Wisconsin dataset, visualize and analyze the results.**

**11) Program to implement the working of linear discriminant analysis using iris dataset and visualize the results.**

**12) Program to Implement multiple linear regression using iris dataset, visualize and analyze the results.**

**1)Program on data wrangling: Combining and merging datasets, Reshaping and Pivoting**

**1 import pandas as pd**

**2**

**3 # Combining and Merging Datasets**

**4 print("Combining and Merging Datasets")**

**5 print("-------------------------------")**

**6**

**7 # Create two sample datasets**

**8 dataset1 = pd.DataFrame({**

**9 'Name': ['John', 'Mary', 'David'],**

**10 'Age': [25, 31, 42]**

**11 })**

**12**

**13 dataset2 = pd.DataFrame({**

**14 'Name': ['Emily', 'Michael', 'Sarah'],**

**15 'Age': [28, 35, 38]**

**16 })**

**17**

**18 # Concatenate the datasets**

**19 combined\_dataset = pd.concat([dataset1, dataset2])**

**20**

**21 print("Concatenated Dataset:")**

**22 print(combined\_dataset)**

**23**

**24 # Create two sample datasets for merging**

**25 dataset1 = pd.DataFrame({**

**26 'ID': [1, 2, 3],**

**27 'Name': ['John', 'Mary', 'David']**

**28 })**

**29**

**30 dataset2 = pd.DataFrame({**

**31 'ID': [1, 2, 3],**

**32 'Age': [25, 31, 42]**

**33 })**

**34**

**35 # Merge the datasets**

**36 merged\_dataset = pd.merge(dataset1, dataset2, on='ID')**

**37**

**38 print("\nMerged Dataset:")**

**39 print(merged\_dataset)**

**40**

**41 # Reshaping and Pivoting**

**42 print("\nReshaping and Pivoting")**

**43 print("-------------------------------")**

**44**

**45 # Create a sample dataset for reshaping**

**46 dataset = pd.DataFrame({**

**47 'ID': [1, 1, 2, 2],**

**48 'Year': [2018, 2019, 2018, 2019],**

**49 'Sales': [100, 120, 80, 90]**

**50 })**

**51**

**52 # Reshape the dataset**

**53 reshaped\_dataset = pd.pivot\_table(dataset, values='Sales', index='ID', columns='Year')**

**54**

**55 print("Reshaped Dataset:")**

**56 print(reshaped\_dataset)**

**57**

**58 # Create a sample dataset for pivoting**

**59 dataset = pd.DataFrame({**

**60 'ID': [1, 2],**

**61 '2018': [100, 80],**

**62 '2019': [120, 90]**

**63 })**

**64**

**65 # Pivot the dataset**

**66 pivoted\_dataset = pd.melt(dataset, id\_vars='ID', value\_vars=['2018', '2019'],**

**67 var\_name='Year', value\_name='Sales')**

**68**

**69 print("\nPivoted Dataset:")**

**70 print(pivoted\_dataset)**

**2)Program on Data Transformation: String Manipulation, Regular Expressions**

**1 import pandas as pd**

**2 import re**

**3**

**4 # String Manipulation**

**5 print("String Manipulation")**

**6 print("-------------------")**

**7**

**8 # Create a sample dataset**

**9 dataset = pd.DataFrame({**

**10 'Name': ['John Smith', 'Mary Johnson', 'David Lee'],**

**11 'Address': ['123 Main St', '456 Elm St', '789 Oak St']**

**12 })**

**13**

**14 print("Original Dataset:")**

**15 print(dataset)**

**16**

**17 # Lowercase strings**

**18 dataset['Name'] = dataset['Name'].str.lower()**

**19 dataset['Address'] = dataset['Address'].str.lower()**

**20**

**21 print("\nLowercase Strings:")**

**22 print(dataset)**

**23**

**24 # Uppercase strings**

**25 dataset['Name'] = dataset['Name'].str.upper()**

**26 dataset['Address'] = dataset['Address'].str.upper()**

**27**

**28 print("\nUppercase Strings:")**

**29 print(dataset)**

**30**

**31 # Strip whitespace**

**32 dataset['Name'] = dataset['Name'].str.strip()**

**33 dataset['Address'] = dataset['Address'].str.strip()**

**34**

**35 print("\nStrip Whitespace:")**

**36 print(dataset)**

**37**

**38 # Replace strings**

**39 dataset['Name'] = dataset['Name'].str.replace('John', 'Jonathan')**

**40 dataset['Address'] = dataset['Address'].str.replace('St', 'Street')**

**41**

**42 print("\nReplace Strings:")**

**43 print(dataset)**

**44**

**45 # Regular Expressions**

**46 print("\nRegular Expressions")**

**47 print("-------------------")**

**48**

**49 # Create a sample dataset**

**50 dataset = pd.DataFrame({**

**51 'Email': ['john@example.com', 'mary@example.org', 'david@example.net'],**

**52 'Phone': ['123-456-7890', '098-765-4321', '555-123-4567']**

**53 })**

**54**

**55 print("Original Dataset:")**

**56 print(dataset)**

**57**

**58 # Extract domain from email using regular expression**

**59 dataset['Domain'] = dataset['Email'].str.extract(r'@(.\*)', expand=False)**

**60**

**61 print("\nExtract Domain:")**

**62 print(dataset)**

**63**

**64 # Validate phone number using regular expression**

**65 dataset['Valid Phone'] = dataset['Phone'].str.contains(r'^\d{3}-\d{3}-\d{4}$', regex=True)**

**66**

**67 print("\nValidate Phone Number:")**

**68 print(dataset)**

**69**

**70 # Extract numbers from phone number using regular expression**

**71 dataset[['Area Code', 'Prefix', 'Line Number']] = dataset['Phone'].str.extract(r'(\d{3})-(\d{3})-(\d{4})')**

**72**

**73 print("\nExtract Phone Numbers:")**

**74 print(dataset)**

**3) Program on Time series: GroupBy Mechanics to display in data vector, multivariate time series and forecasting formats**

**1 import pandas as pd**

**2 import numpy as np**

**3 import matplotlib.pyplot as plt**

**4 from statsmodels.tsa.seasonal import seasonal\_decompose**

**5 from statsmodels.tsa.arima\_model import ARIMA**

**6**

**7 # Load sample time series data**

**8 data = pd.read\_csv('data.csv', index\_col='Date', parse\_dates=['Date'])**

**9**

**10 # Convert to datetime index**

**11 data.index = pd.to\_datetime(data.index)**

**12**

**13 # GroupBy Mechanics**

**14 print("GroupBy Mechanics")**

**15 print("------------------")**

**16**

**17 # Group by year and month**

**18 grouped\_data = data.groupby([data.index.year, data.index.month]).mean()**

**19**

**20 print("Grouped Data:")**

**21 print(grouped\_data)**

**22**

**23 # Vector Format**

**24 print("\nVector Format")**

**25 print("--------------")**

**26**

**27 # Resample data to quarterly frequency**

**28 quarterly\_data = data.resample('Q').mean()**

**29**

**30 print("Quarterly Data:")**

**31 print(quarterly\_data)**

**32**

**33 # Multivariate Time Series**

**34 print("\nMultivariate Time Series")**

**35 print("------------------------")**

**36**

**37 # Create a multivariate time series dataset**

**38 multivariate\_data = pd.DataFrame({**

**39 'Var1': np.random.normal(0, 1, len(data)),**

**40 'Var2': np.random.normal(0, 1, len(data)),**

**41 'Var3': np.random.normal(0, 1, len(data))**

**42 }, index=data.index)**

**43**

**44 print("Multivariate Data:")**

**45 print(multivariate\_data)**

**46**

**47 # Forecasting**

**48 print("\nForecasting")**

**49 print("------------")**

**50**

**51 # Decompose time series into trend, seasonality, and residuals**

**52 decomposition = seasonal\_decompose(data, model='additive')**

**53**

**54 trend = decomposition.trend**

**55 seasonal = decomposition.seasonal**

**56 residual = decomposition.resid**

**57**

**58 print("Decomposition:")**

**59 print("Trend:", trend)**

**60 print("Seasonal:", seasonal)**

**61 print("Residual:", residual)**

**62**

**63 # Fit ARIMA model**

**64 model = ARIMA(data, order=(1, 1, 1))**

**65 model\_fit = model.fit()**

**66**

**67 print("ARIMA Model:")**

**68 print("Coefficients:", model\_fit.params)**

**69 print("Standard Errors:", model\_fit.bse)**

**70**

**71 # Forecast future values**

**72 forecast = model\_fit.forecast(steps=30)**

**73**

**74 print("Forecast:")**

**75 print(forecast)**

**76**

**77 # Plot the data**

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

**79 plt.plot(data)**

**80 plt.plot(quarterly\_data)**

**81 plt.plot(multivariate\_data)**

**82 plt.plot(trend)**

**83 plt.plot(seasonal)**

**84 plt.plot(residual)**

**85 plt.plot(forecast)**

**86 plt.legend(['Original', 'Quarterly', 'Multivariate', 'Trend', 'Seasonal', 'Residual', 'Forecast'])**

**87 plt.show()**

**4)Program to measure central tendency and measures of dispersion: Mean, Median, Mode, Standard Deviation, Variance, Mean deviation and Quartile deviation for a frequency distribution/data.**

**1 import pandas as pd**

**2 import numpy as np**

**3**

**4 # Load sample frequency distribution data**

**5 data = pd.read\_csv('data.csv')**

**6**

**7 # Calculate Central Tendency Measures**

**8 print("Central Tendency Measures")**

**9 print("----------------------------")**

**10**

**11 # Mean**

**12 mean = np.mean(data['Values'])**

**13 print("Mean:", mean)**

**14**

**15 # Median**

**16 median = np.median(data['Values'])**

**17 print("Median:", median)**

**18**

**19 # Mode**

**20 mode = data['Values'].mode().iloc[0]**

**21 print("Mode:", mode)**

**22**

**23 # Calculate Measures of Dispersion**

**24 print("\nMeasures of Dispersion")**

**25 print("----------------------------")**

**26**

**27 # Standard Deviation**

**28 std\_dev = np.std(data['Values'])**

**29 print("Standard Deviation:", std\_dev)**

**30**

**31 # Variance**

**32 variance = np.var(data['Values'])**

**33 print("Variance:", variance)**

**34**

**35 # Mean Deviation**

**36 mean\_dev = np.mean(np.abs(data['Values'] - mean))**

**37 print("Mean Deviation:", mean\_dev)**

**38**

**39 # Quartile Deviation**

**40 q1 = np.percentile(data['Values'], 25)**

**41 q3 = np.percentile(data['Values'], 75)**

**42 quartile\_dev = (q3 - q1) / 2**

**43 print("Quartile Deviation:", quartile\_dev)**

**44**

**45 # Print summary statistics**

**46 print("\nSummary Statistics:")**

**47 print(data['Values'].describe())**

**5)Program to perform cross validation for a given dataset to measure Root Mean Squared Error (RMSE), Mean Absolute Error (MAE) and R2 Error using Validation Set, Leave One Out Cross-Validation(LOOCV) and K-fold Cross-Validation approaches**

**1 import pandas as pd**

**2 from sklearn.model\_selection import train\_test\_split, cross\_val\_score, LeaveOneOut, KFold**

**3 from sklearn.linear\_model import LinearRegression**

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

**5**

**6 # Load sample dataset**

**7 data = pd.read\_csv('data.csv')**

**8**

**9 # Split data into features (X) and target (y)**

**10 X = data.drop('target', axis=1)**

**11 y = data['target']**

**12**

**13 # Split data into training and validation sets**

**14 X\_train, X\_val, y\_train, y\_val = train\_test\_split(X, y, test\_size=0.2, random\_state=42)**

**15**

**16 # Define linear regression model**

**17 model = LinearRegression()**

**18**

**19 # Validation Set Approach**

**20 print("Validation Set Approach")**

**21 print("----------------------------")**

**22**

**23 # Train model on training data**

**24 model.fit(X\_train, y\_train)**

**25**

**26 # Make predictions on validation data**

**27 y\_pred = model.predict(X\_val)**

**28**

**29 # Calculate RMSE, MAE, and R2 Error**

**30 rmse = np.sqrt(mean\_squared\_error(y\_val, y\_pred))**

**31 mae = mean\_absolute\_error(y\_val, y\_pred)**

**32 r2 = r2\_score(y\_val, y\_pred)**

**33**

**34 print("RMSE:", rmse)**

**35 print("MAE:", mae)**

**36 print("R2 Error:", r2)**

**37**

**38 # Leave One Out Cross-Validation (LOOCV) Approach**

**39 print("\nLeave One Out Cross-Validation (LOOCV) Approach")**

**40 print("------------------------------------------------")**

**41**

**42 # Define LOOCV object**

**43 loo = LeaveOneOut()**

**44**

**45 # Initialize lists to store scores**

**46 rmse\_loo = []**

**47 mae\_loo = []**

**48 r2\_loo = []**

**49**

**50 # Perform LOOCV**

**51 for train\_index, val\_index in loo.split(X):**

**52 X\_train\_loo, X\_val\_loo = X.iloc[train\_index], X.iloc[val\_index]**

**53 y\_train\_loo, y\_val\_loo = y.iloc[train\_index], y.iloc[val\_index]**

**54**

**55 # Train model on training data**

**56 model.fit(X\_train\_loo, y\_train\_loo)**

**57**

**58 # Make predictions on validation data**

**59 y\_pred\_loo = model.predict(X\_val\_loo)**

**60**

**61 # Calculate RMSE, MAE, and R2 Error**

**62 rmse\_loo.append(np.sqrt(mean\_squared\_error(y\_val\_loo, y\_pred\_loo)))**

**63 mae\_loo.append(mean\_absolute\_error(y\_val\_loo, y\_pred\_loo))**

**64 r2\_loo.append(r2\_score(y\_val\_loo, y\_pred\_loo))**

**65**

**66 print("RMSE (LOOCV):", np.mean(rmse\_loo))**

**67 print("MAE (LOOCV):", np.mean(mae\_loo))**

**68 print("R2 Error (LOOCV):", np.mean(r2\_loo))**

**69**

**70 # K-fold Cross-Validation Approach**

**71 print("\nK-fold Cross-Validation Approach")**

**72 print("-------------------------------")**

**73**

**74 # Define K-fold object**

**75 kf = KFold(n\_splits=5, shuffle=True, random\_state=42)**

**76**

**77 # Initialize lists to store scores**

**78 rmse\_kf = []**

**79 mae\_kf = []**

**80 r2\_kf = []**

**81**

**82 # Perform K-fold cross-validation**

**83 for train\_index, val\_index in kf.split(X):**

**84 X\_train\_kf, X\_val\_kf = X.iloc[train\_index], X.iloc[val\_index]**

**85 y\_train\_kf, y\_val\_kf = y.iloc[train\_index], y.iloc[val\_index]**

**86**

**87 # Train model on training data**

**88 model.fit(X\_train\_kf, y\_train\_kf)**

**89**

**90 # Make predictions on validation data**

**91 y\_pred\_kf = model.predict(X\_val\_kf)**

**92**

**93 # Calculate RMSE, MAE, and R2 Error**

**94 rmse\_kf.append(np.sqrt(mean\_squared\_error(y\_val\_kf, y\_pred\_kf)))**

**95 mae\_kf.append(mean\_absolute\_error(y\_val\_kf, y\_pred\_kf))**

**96 r2\_kf.append(r2\_score(y\_val\_kf, y\_pred\_kf))**

**97**

**98 print("RMSE (K-fold):", np.mean(rmse\_kf))**

**99 print("MAE (K-fold):", np.mean(mae\_kf))**

**100 print("R2 Error (K-fold):", np.mean(r2\_kf))**

**6)Program to display Normal, Binomial Poisson, Bernoulli disrtibutions for a given frequency distribution and analyze the results.**

**1 import pandas as pd**

**2 import numpy as np**

**3 import matplotlib.pyplot as plt**

**4 from scipy.stats import norm, binom, poisson, bernoulli**

**5**

**6 # Load sample frequency distribution data**

**7 data = pd.read\_csv('data.csv')**

**8**

**9 # Calculate mean and standard deviation of the data**

**10 mean = np.mean(data['Values'])**

**11 std\_dev = np.std(data['Values'])**

**12**

**13 # Display Normal Distribution**

**14 print("Normal Distribution")**

**15 print("------------------")**

**16**

**17 # Create a normal distribution with the same mean and standard deviation as the data**

**18 normal\_dist = norm(loc=mean, scale=std\_dev)**

**19**

**20 # Generate random samples from the normal distribution**

**21 normal\_samples = normal\_dist.rvs(size=len(data))**

**22**

**23 # Plot the normal distribution**

**24 plt.hist(normal\_samples, bins=30, density=True)**

**25 plt.title("Normal Distribution")**

**26 plt.xlabel("Value")**

**27 plt.ylabel("Probability")**

**28 plt.show()**

**29**

**30 # Display Binomial Distribution**

**31 print("\nBinomial Distribution")**

**32 print("---------------------")**

**33**

**34 # Define the number of trials and probability of success**

**35 n = 10**

**36 p = 0.5**

**37**

**38 # Create a binomial distribution**

**39 binom\_dist = binom(n=n, p=p)**

**40**

**41 # Generate random samples from the binomial distribution**

**42 binom\_samples = binom\_dist.rvs(size=len(data))**

**43**

**44 # Plot the binomial distribution**

**45 plt.hist(binom\_samples, bins=30, density=True)**

**46 plt.title("Binomial Distribution")**

**47 plt.xlabel("Value")**

**48 plt.ylabel("Probability")**

**49 plt.show()**

**50**

**51 # Display Poisson Distribution**

**52 print("\nPoisson Distribution")**

**53 print("---------------------")**

**54**

**55 # Define the rate parameter**

**56 lam = 5**

**57**

**58 # Create a Poisson distribution**

**59 poisson\_dist = poisson(lam=lam)**

**60**

**61 # Generate random samples from the Poisson distribution**

**62 poisson\_samples = poisson\_dist.rvs(size=len(data))**

**63**

**64 # Plot the Poisson distribution**

**65 plt.hist(poisson\_samples, bins=30, density=True)**

**66 plt.title("Poisson Distribution")**

**67 plt.xlabel("Value")**

**68 plt.ylabel("Probability")**

**69 plt.show()**

**70**

**71 # Display Bernoulli Distribution**

**72 print("\nBernoulli Distribution")**

**73 print("---------------------")**

**74**

**75 # Define the probability of success**

**76 p = 0.5**

**77**

**78 # Create a Bernoulli distribution**

**79 bernoulli\_dist = bernoulli(p=p)**

**80**

**81 # Generate random samples from the Bernoulli distribution**

**82 bernoulli\_samples = bernoulli\_dist.rvs(size=len(data))**

**83**

**84 # Plot the Bernoulli distribution**

**85 plt.hist(bernoulli\_samples, bins=30, density=True)**

**86 plt.title("Bernoulli Distribution")**

**87 plt.xlabel("Value")**

**88 plt.ylabel("Probability")**

**89 plt.show()**

**90**

**91 # Analyze the results**

**92 print("\nAnalysis of Results")**

**93 print("---------------------")**

**94**

**95 # Calculate the mean and standard deviation of each distribution**

**96 normal\_mean = np.mean(normal\_samples)**

**97 normal\_std\_dev = np.std(normal\_samples)**

**98**

**99 binom\_mean = np.mean(binom\_samples)**

**100 binom\_std\_dev = np.std(binom\_samples)**

**101**

**102 poisson\_mean = np.mean(poisson\_samples)**

**103 poisson\_std\_dev = np.std(poisson\_samples)**

**104**

**105 bernoulli\_mean = np.mean(bernoulli\_samples)**

**106 bernoulli\_std\_dev = np.std(bernoulli\_samples)**

**107**

**108 print("Normal Distribution: Mean =", normal\_mean, ", Standard Deviation =", normal\_std\_dev)**

**109 print("Binomial Distribution: Mean =", binom\_mean, ", Standard Deviation =", binom\_std\_dev)**

**110 print("Poisson Distribution: Mean =", poisson\_mean, ", Standard Deviation =", poisson\_std\_dev)**

**111 print("Bernoulli Distribution: Mean =", bernoulli\_mean, ", Standard Deviation =", bernoulli\_std\_dev)**

**7)Program to implement one sample, two sample and paired sample t-tests for a sample data and analyse the results.**

**1 import pandas as pd**

**2 import numpy as np**

**3 from scipy import stats**

**4**

**5 # Load sample data**

**6 data = pd.read\_csv('data.csv')**

**7**

**8 # One Sample T-Test**

**9 print("One Sample T-Test")**

**10 print("------------------")**

**11**

**12 # Define the null hypothesis mean**

**13 null\_mean = 0**

**14**

**15 # Perform one sample t-test**

**16 t\_stat, p\_val = stats.ttest\_1samp(data['Values'], null\_mean)**

**17**

**18 print("T-Statistic:", t\_stat)**

**19 print("P-Value:", p\_val)**

**20**

**21 # Interpret the results**

**22 if p\_val < 0.05:**

**23 print("Reject the null hypothesis. The sample mean is significantly different from the null mean.")**

**24 else:**

**25 print("Fail to reject the null hypothesis. The sample mean is not significantly different from the null mean.")**

**26**

**27 # Two Sample T-Test**

**28 print("\nTwo Sample T-Test")**

**29 print("------------------")**

**30**

**31 # Load second sample data**

**32 data2 = pd.read\_csv('data2.csv')**

**33**

**34 # Perform two sample t-test**

**35 t\_stat, p\_val = stats.ttest\_ind(data['Values'], data2['Values'])**

**36**

**37 print("T-Statistic:", t\_stat)**

**38 print("P-Value:", p\_val)**

**39**

**40 # Interpret the results**

**41 if p\_val < 0.05:**

**42 print("Reject the null hypothesis. The two sample means are significantly different.")**

**43 else:**

**44 print("Fail to reject the null hypothesis. The two sample means are not significantly different.")**

**45**

**46 # Paired Sample T-Test**

**47 print("\nPaired Sample T-Test")**

**48 print("---------------------")**

**49**

**50 # Load paired sample data**

**51 data3 = pd.read\_csv('data3.csv')**

**52**

**53 # Perform paired sample t-test**

**54 t\_stat, p\_val = stats.ttest\_rel(data['Values'], data3['Values'])**

**55**

**56 print("T-Statistic:", t\_stat)**

**57 print("P-Value:", p\_val)**

**58**

**59 # Interpret the results**

**60 if p\_val < 0.05:**

**61 print("Reject the null hypothesis. The paired sample means are significantly different.")**

**62 else:**

**63 print("Fail to reject the null hypothesis. The paired sample means are not significantly different.")**

**64**

**65 # Analyze the results**

**66 print("\nAnalysis of Results")**

**67 print("---------------------")**

**68**

**69 # Calculate the mean and standard deviation of each sample**

**70 mean1 = np.mean(data['Values'])**

**71 std\_dev1 = np.std(data['Values'])**

**72**

**73 mean2 = np.mean(data2['Values'])**

**74 std\_dev2 = np.std(data2['Values'])**

**75**

**76 mean3 = np.mean(data3['Values'])**

**77 std\_dev3 = np.std(data3['Values'])**

**78**

**79 print("Sample 1: Mean =", mean1, ", Standard Deviation =", std\_dev1)**

**80 print("Sample 2: Mean =", mean2, ", Standard Deviation =", std\_dev2)**

**81 print("Sample 3: Mean =", mean3, ", Standard Deviation =", std\_dev3)**

**8)Program to implement One-way and Two-way ANOVA tests and analyze the results**

**1 import pandas as pd**

**2 import numpy as np**

**3 from scipy import stats**

**4**

**5 # Load sample data**

**6 data = pd.read\_csv('data.csv')**

**7**

**8 # One-way ANOVA**

**9 print("One-way ANOVA")**

**10 print("-------------")**

**11**

**12 # Define the groups**

**13 groups = data['Group']**

**14**

**15 # Perform one-way ANOVA**

**16 f\_stat, p\_val = stats.f\_oneway(data['Values'], groups)**

**17**

**18 print("F-Statistic:", f\_stat)**

**19 print("P-Value:", p\_val)**

**20**

**21 # Interpret the results**

**22 if p\_val < 0.05:**

**23 print("Reject the null hypothesis. The means of the groups are significantly different.")**

**24 else:**

**25 print("Fail to reject the null hypothesis. The means of the groups are not significantly different.")**

**26**

**27 # Two-way ANOVA**

**28 print("\nTwo-way ANOVA")**

**29 print("-------------")**

**30**

**31 # Load second sample data**

**32 data2 = pd.read\_csv('data2.csv')**

**33**

**34 # Define the groups**

**35 groups1 = data['Group1']**

**36 groups2 = data['Group2']**

**37**

**38 # Perform two-way ANOVA**

**39 f\_stat1, p\_val1 = stats.f\_oneway(data['Values'], groups1)**

**40 f\_stat2, p\_val2 = stats.f\_oneway(data['Values'], groups2)**

**41**

**42 print("F-Statistic (Group 1):", f\_stat1)**

**43 print("P-Value (Group 1):", p\_val1)**

**44 print("F-Statistic (Group 2):", f\_stat2)**

**45 print("P-Value (Group 2):", p\_val2)**

**46**

**47 # Interpret the results**

**48 if p\_val1 < 0.05 and p\_val2 < 0.05:**

**49 print("Reject the null hypothesis. The means of the groups are significantly different.")**

**50 else:**

**51 print("Fail to reject the null hypothesis. The means of the groups are not significantly different.")**

**52**

**53 # Analyze the results**

**54 print("\nAnalysis of Results")**

**55 print("---------------------")**

**56**

**57 # Calculate the mean and standard deviation of each group**

**58 mean1 = np.mean(data['Values'])**

**59 std\_dev1 = np.std(data['Values'])**

**60**

**61 mean2 = np.mean(data2['Values'])**

**62 std\_dev2 = np.std(data2['Values'])**

**63**

**64 print("Group 1: Mean =", mean1, ", Standard Deviation =", std\_dev1)**

**65 print("Group 2: Mean =", mean2, ", Standard Deviation =", std\_dev2)**

**66**

**67 # Post-hoc analysis**

**68 print("\nPost-hoc Analysis")**

**69 print("------------------")**

**70**

**71 # Perform Tukey's HSD test**

**72 from statsmodels.stats.multicomp import pairwise\_tukeyhsd**

**73 tukey = pairwise\_tukeyhsd(data['Values'], groups, alpha=0.05)**

**74**

**75 print("Tukey's HSD Test:")**

**76 print(tukey)**

**77**

**78 # Perform Scheffé test**

**79 from statsmodels.stats.multicomp import pairwise\_scheffe**

**80 scheffe = pairwise\_scheffe(data['Values'], groups, alpha=0.05)**

**81**

**82 print("Scheffé Test:")**

**83 print(scheffe)**

**9)Program to implement correlation, rank correlation and regression and plot x-y plot and heat maps of correlation matrices.**

**1 import pandas as pd**

**2 import numpy as np**

**3 import matplotlib.pyplot as plt**

**4 import seaborn as sns**

**5 from scipy.stats import pearsonr, spearmanr**

**6 from sklearn.linear\_model import LinearRegression**

**7**

**8 # Load sample data**

**9 data = pd.read\_csv('data.csv')**

**10**

**11 # Correlation**

**12 print("Correlation")**

**13 print("-----------")**

**14**

**15 # Calculate correlation matrix**

**16 corr\_matrix = data.corr()**

**17**

**18 # Print correlation matrix**

**19 print(corr\_matrix)**

**20**

**21 # Plot heat map of correlation matrix**

**22 plt.figure(figsize=(10, 8))**

**23 sns.heatmap(corr\_matrix, annot=True, cmap='coolwarm', square=True)**

**24 plt.title("Correlation Matrix")**

**25 plt.show()**

**26**

**27 # Rank Correlation**

**28 print("\nRank Correlation")**

**29 print("----------------")**

**30**

**31 # Calculate rank correlation matrix**

**32 rank\_corr\_matrix = data.rank().corr()**

**33**

**34 # Print rank correlation matrix**

**35 print(rank\_corr\_matrix)**

**36**

**37 # Plot heat map of rank correlation matrix**

**38 plt.figure(figsize=(10, 8))**

**39 sns.heatmap(rank\_corr\_matrix, annot=True, cmap='coolwarm', square=True)**

**40 plt.title("Rank Correlation Matrix")**

**41 plt.show()**

**42**

**43 # Regression**

**44 print("\nRegression")**

**45 print("-----------")**

**46**

**47 # Define independent and dependent variables**

**48 X = data['X']**

**49 y = data['y']**

**50**

**51 # Perform linear regression**

**52 model = LinearRegression()**

**53 model.fit(X.values.reshape(-1, 1), y)**

**54**

**55 # Print coefficients**

**56 print("Coefficients:", model.coef\_)**

**57**

**58 # Plot x-y plot**

**59 plt.figure(figsize=(10, 8))**

**60 plt.scatter(X, y)**

**61 plt.plot(X, model.predict(X.values.reshape(-1, 1)), color='red')**

**62 plt.title("Linear Regression")**

**63 plt.xlabel("X")**

**64 plt.ylabel("y")**

**65 plt.show()**

**66**

**67 # Plot residual plot**

**68 plt.figure(figsize=(10, 8))**

**69 plt.scatter(y, model.predict(X.values.reshape(-1, 1)) - y)**

**70 plt.title("Residual Plot")**

**71 plt.xlabel("y")**

**72 plt.ylabel("Residuals")**

**73 plt.show()**

**74**

**75 # Pearson Correlation**

**76 print("\nPearson Correlation")**

**77 print("------------------")**

**78**

**79 # Calculate Pearson correlation coefficient**

**80 pearson\_corr, \_ = pearsonr(X, y)**

**81**

**82 # Print Pearson correlation coefficient**

**83 print("Pearson Correlation Coefficient:", pearson\_corr)**

**84**

**85 # Spearman Rank Correlation**

**86 print("\nSpearman Rank Correlation")**

**87 print("------------------------")**

**88**

**89 # Calculate Spearman rank correlation coefficient**

**90 spearman\_corr, \_ = spearmanr(X, y)**

**91**

**92 # Print Spearman rank correlation coefficient**

**93 print("Spearman Rank Correlation Coefficient:", spearman\_corr)**

**10)Program to implement PCA for Wisconsin dataset, visualize and analyze the results.**

**1 import pandas as pd**

**2 import numpy as np**

**3 import matplotlib.pyplot as plt**

**4 from sklearn.decomposition import PCA**

**5 from sklearn.preprocessing import StandardScaler**

**6**

**7 # Load Wisconsin dataset**

**8 data = pd.read\_csv('wisconsin.csv')**

**9**

**10 # Drop the target variable**

**11 data = data.drop('target', axis=1)**

**12**

**13 # Scale the data using StandardScaler**

**14 scaler = StandardScaler()**

**15 data\_scaled = scaler.fit\_transform(data)**

**16**

**17 # Perform PCA**

**18 pca = PCA(n\_components=2)**

**19 data\_pca = pca.fit\_transform(data\_scaled)**

**20**

**21 # Print the explained variance ratio**

**22 print("Explained Variance Ratio:", pca.explained\_variance\_ratio\_)**

**23**

**24 # Plot the data in the new coordinate system**

**25 plt.figure(figsize=(10, 8))**

**26 plt.scatter(data\_pca[:, 0], data\_pca[:, 1])**

**27 plt.title("PCA of Wisconsin Dataset")**

**28 plt.xlabel("Principal Component 1")**

**29 plt.ylabel("Principal Component 2")**

**30 plt.show()**

**31**

**32 # Plot the cumulative explained variance**

**33 plt.figure(figsize=(10, 8))**

**34 plt.plot(np.cumsum(pca.explained\_variance\_ratio\_))**

**35 plt.title("Cumulative Explained Variance")**

**36 plt.xlabel("Number of Components")**

**37 plt.ylabel("Cumulative Explained Variance")**

**38 plt.show()**

**39**

**40 # Analyze the results**

**41 print("\nAnalysis of Results")**

**42 print("---------------------")**

**43**

**44 # Print the shape of the original data**

**45 print("Shape of Original Data:", data.shape)**

**46**

**47 # Print the shape of the data after PCA**

**48 print("Shape of Data after PCA:", data\_pca.shape)**

**49**

**50 # Print the number of components retained**

**51 print("Number of Components Retained:", pca.n\_components\_)**

**52**

**53 # Print the explained variance ratio**

**54 print("Explained Variance Ratio:", pca.explained\_variance\_ratio\_)**

**55**

**56 # Print the singular values**

**57 print("Singular Values:", pca.singular\_values\_)**

**58**

**59 # Print the components**

**60 print("Components:", pca.components\_)**

**11)Program to implement the working of linear discriminant analysis using iris dataset and visualize the results.**

**1 import pandas as pd**

**2 import numpy as np**

**3 import matplotlib.pyplot as plt**

**4 from sklearn.discriminant\_analysis import LinearDiscriminantAnalysis**

**5 from sklearn.decomposition import PCA**

**6 from sklearn.preprocessing import StandardScaler**

**7**

**8 # Load iris dataset**

**9 from sklearn.datasets import load\_iris**

**10 iris = load\_iris()**

**11 data = pd.DataFrame(data=iris.data, columns=iris.feature\_names)**

**12 data['target'] = iris.target**

**13**

**14 # Scale the data using StandardScaler**

**15 scaler = StandardScaler()**

**16 data\_scaled = scaler.fit\_transform(data.drop('target', axis=1))**

**17**

**18 # Perform PCA to reduce dimensionality**

**19 pca = PCA(n\_components=2)**

**20 data\_pca = pca.fit\_transform(data\_scaled)**

**21**

**22 # Perform LDA**

**23 lda = LinearDiscriminantAnalysis(n\_components=2)**

**24 data\_lda = lda.fit\_transform(data\_scaled, data['target'])**

**25**

**26 # Plot the data in the new coordinate system**

**27 plt.figure(figsize=(10, 8))**

**28 plt.scatter(data\_lda[:, 0], data\_lda[:, 1], c=data['target'])**

**29 plt.title("LDA of Iris Dataset")**

**30 plt.xlabel("Linear Discriminant 1")**

**31 plt.ylabel("Linear Discriminant 2")**

**32 plt.show()**

**33**

**34 # Plot the data in the original coordinate system**

**35 plt.figure(figsize=(10, 8))**

**36 plt.scatter(data\_pca[:, 0], data\_pca[:, 1], c=data['target'])**

**37 plt.title("PCA of Iris Dataset")**

**38 plt.xlabel("Principal Component 1")**

**39 plt.ylabel("Principal Component 2")**

**40 plt.show()**

**41**

**42 # Analyze the results**

**43 print("\nAnalysis of Results")**

**44 print("---------------------")**

**45**

**46 # Print the shape of the original data**

**47 print("Shape of Original Data:", data.shape)**

**48**

**49 # Print the shape of the data after LDA**

**50 print("Shape of Data after LDA:", data\_lda.shape)**

**51**

**52 # Print the number of components retained**

**53 print("Number of Components Retained:", lda.n\_components)**

**54**

**55 # Print the explained variance ratio**

**56 print("Explained Variance Ratio:", lda.explained\_variance\_ratio\_)**

**57**

**58 # Print the singular values**

**59 print("Singular Values:", lda.singular\_values\_)**

**60**

**61 # Print the coefficients**

**62 print("Coefficients:", lda.coef\_)**

**12)Program to Implement multiple linear regression using iris dataset, visualize and analyze the results.**

**1 import pandas as pd**

**2 import numpy as np**

**3 import matplotlib.pyplot as plt**

**4 from sklearn.model\_selection import train\_test\_split**

**5 from sklearn.linear\_model import LinearRegression**

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

**7 from sklearn.preprocessing import StandardScaler**

**8**

**9 # Load iris dataset**

**10 from sklearn.datasets import load\_iris**

**11 iris = load\_iris()**

**12 data = pd.DataFrame(data=iris.data, columns=iris.feature\_names)**

**13 data['target'] = iris.target**

**14**

**15 # Define features (X) and target (y)**

**16 X = data.drop('target', axis=1)**

**17 y = data['target']**

**18**

**19 # Split data into training and testing sets**

**20 X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)**

**21**

**22 # Scale the data using StandardScaler**

**23 scaler = StandardScaler()**

**24 X\_train\_scaled = scaler.fit\_transform(X\_train)**

**25 X\_test\_scaled = scaler.transform(X\_test)**

**26**

**27 # Perform multiple linear regression**

**28 model = LinearRegression()**

**29 model.fit(X\_train\_scaled, y\_train)**

**30**

**31 # Make predictions on testing data**

**32 y\_pred = model.predict(X\_test\_scaled)**

**33**

**34 # Calculate mean squared error and R-squared value**

**35 mse = mean\_squared\_error(y\_test, y\_pred)**

**36 r2 = r2\_score(y\_test, y\_pred)**

**37**

**38 print("Mean Squared Error:", mse)**

**39 print("R-squared Value:", r2)**

**40**

**41 # Visualize the results**

**42 plt.figure(figsize=(10, 8))**

**43 plt.scatter(y\_test, y\_pred)**

**44 plt.plot([y\_test.min(), y\_test.max()], [y\_test.min(), y\_test.max()], 'r--')**

**45 plt.title("Multiple Linear Regression")**

**46 plt.xlabel("Actual Values")**

**47 plt.ylabel("Predicted Values")**

**48 plt.show()**

**49**

**50 # Analyze the results**

**51 print("\nAnalysis of Results")**

**52 print("---------------------")**

**53**

**54 # Print the coefficients**

**55 print("Coefficients:", model.coef\_)**

**56**

**57 # Print the intercept**

**58 print("Intercept:", model.intercept\_)**

**59**

**60 # Print the feature importances (Note: Not available in LinearRegression)**

**61 print("Feature Importances: Not available for LinearRegression model")**

**62**

**63 # Print the mean squared error and R-squared value**

**64 print("Mean Squared Error:", mse)**

**65 print("R-squared Value:", r2)**

**66**

**67 # Print the training and testing data shapes**

**68 print("Training Data Shape:", X\_train\_scaled.shape)**

**69 print("Testing Data Shape:", X\_test\_scaled.shape)**

|  |
| --- |
| **Srinivas Institute of Technology**  **Department of Computer Science and Engineering** |
| **Program Outcomes ( POs)** |
| 1. **Engineering Knowledge:Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.** 2. **Problem Analysis:Identify, formulate, review research literature, and analyse complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences and engineering sciences.** 3. **Design/Development of Solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.** 4. **Conduct Investigations of Complex Problems:Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions for complex problems.** 5. **Modern Tool Usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modelling to complex engineering activities with an understanding of the limitations.** 6. **The Engineer and Society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.** 7. **Environment and Sustainability:Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.** 8. **Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.** 9. **Individual and Team Work:Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.** 10. **Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.** 11. **Project Management and Finance: Demonstrate knowledge and understanding of the engineering**   **and management principles and apply these to one’s own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.**   1. **Life-long Learning: Recognize the need for, and have the preparation and ability to engage in independent and lifelong learning in the broadest context of technological change.** |
| **Program Specific Outcomes (PSOs)** |
| **PSO 1: Programming and software development skills: Ability to apply the concepts and practical knowledge in analysis, design and development of computing systems and applications to multi-disciplinary problems.**  **PSO2: Domain specific skills: To provide a concrete foundation and enrich their abilities to qualify for Employment, Higher studies and Research in Artificial Intelligence and Data science with ethical values.** |