FDS LAB PROGRAMS

NAME:KIRANHARINARAYANA REG NO:230701152
CLASS:CSE-C

Exp No 1.aAnalyze the trend of data science job postings over the last decade

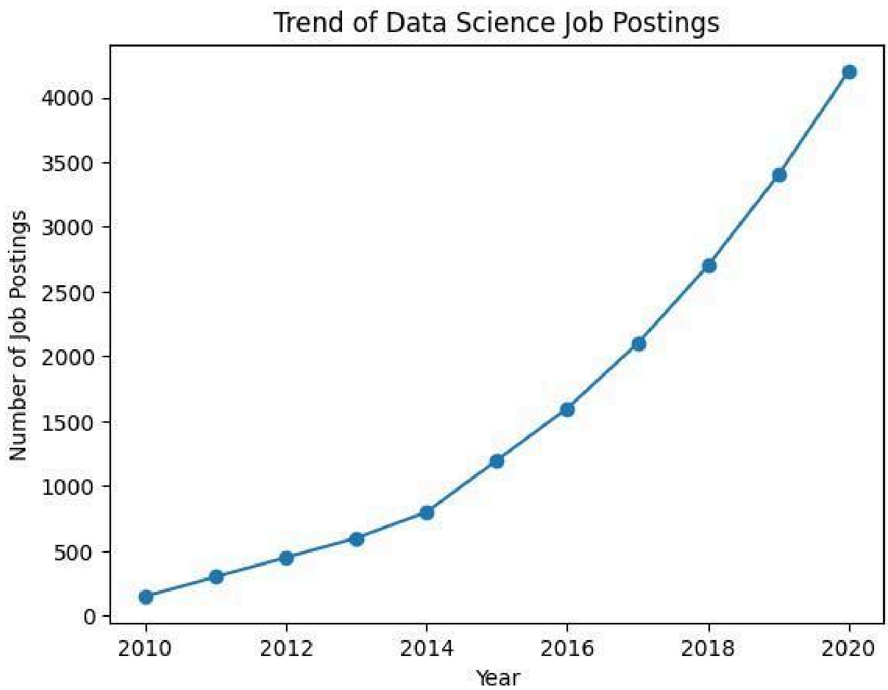
Description: Use web scraping (e.g., BeautifulSoup) or APIs (e.g., LinkedIn API) to gather data on

the number of data science job postings each year. Use pandas for data manipulation and

matplotlib/seaborn for visualization.

Code:

```
'''
1a.No:1.aAnalyze the trend of data science job postings over the last decade

Description: Use web scraping (e.g., BeautifulSoup) or APIs (e.g., LinkedIn API) to gather data on
the number of data science job postings each year. Use pandas for data manipulation and
matplotlib/seaborn for visualization.

Code:
'''
import pandas as pd
import matplotlib.pyplot as plt
data = {'Year': list(range(2010, 2021)),
'Job Postings': [150, 300, 450, 600, 800, 1200, 1600, 2100, 2700, 3400, 4200]}

df = pd.DataFrame(data)
plt.plot(df['Year'], df['Job Postings'], marker='o')
plt.title('Trend of Data Science Job Postings')
plt.xlabel('Year')
plt.ylabel('Number of Job Postings')
plt.show()
```



Analyze and visualize the distribution of various data science roles (Data

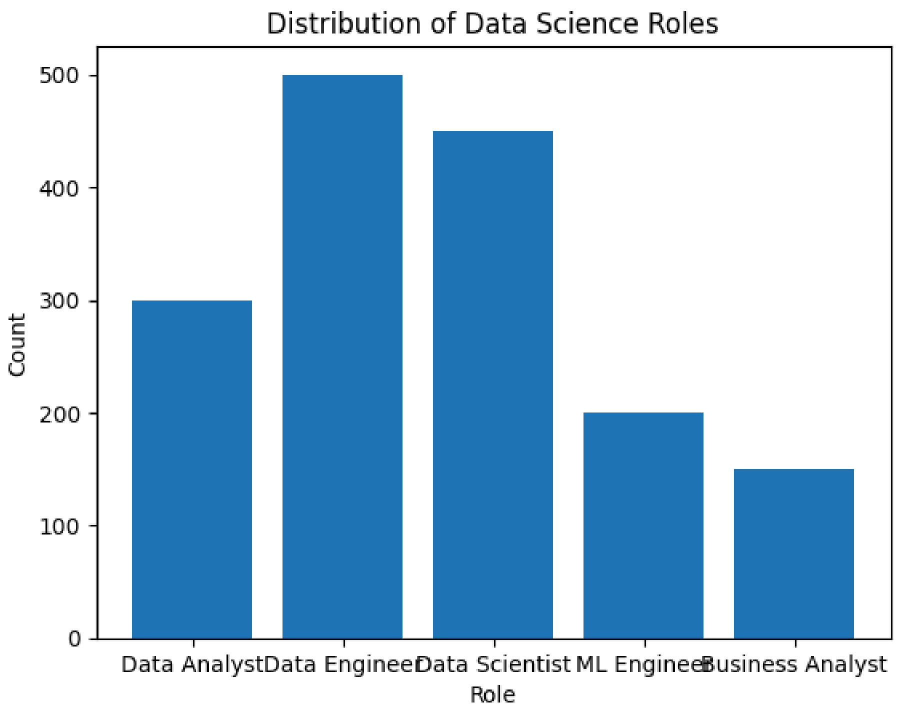
Analyst, Data Engineer, Data Scientist, etc.) from a dataset.

Description: Use a dataset of job postings and categorize them into different roles. Visualize

the distribution using pie charts or bar plots.

Code:

```
roles = ['Data Analyst', 'Data Engineer', 'Data Scientist', 'ML Engineer',
'Business Analyst']
counts = [300, 500, 450, 200, 150]
plt.bar(roles, counts)
plt.title('Distribution of Data Science Roles')
plt.xlabel('Role')
plt.ylabel('Count')
plt.show()
```



Conduct an experiment to differentiate Structured , Un-structured and Semi

structured data based on data sets given.

Description: Create small datasets for each type and explain their characteristics.

Code:

```
# Structured data example
structured_data = pd.DataFrame({
'ID': [1, 2, 3],
'Name': ['Alice', 'Bob', 'Charlie'],
'Age': [25, 30, 35]
})
print('Structured Data:', structured_data)

# Unstructured data example
unstructured_data = 'This is an example of unstructured data. It can be a piece of text, an image,or a video file.'
print('\nUnstructured Data:\n', unstructured_data)

# Semi-structured data example (JSON)
semi_structured_data = {'ID': 1, 'Name': 'Alice', 'Attributes': {'Height': 165, 'Weight': 68}}
print('\nSemi-structured Data:', semi_structured_data)
```

```
Structured Data:    ID     Name  Age
0   1    Alice   25
1   2      Bob   30
2   3  Charlie   35

Unstructured Data:
 This is an example of unstructured data. It can be a piece of text, an image,or a video file.

Semi-structured Data: {'ID': 1, 'Name': 'Alice', 'Attributes': {'Height': 165, 'Weight': 68}}
```

Exp No:1.d Conduct an experiment to encrypt and decrypt given sensitive data.

Description: Use the cryptography library to encrypt and decrypt a piece of data.

```
# Generate key and encrypt data
from cryptography.fernet import Fernet
key = Fernet.generate_key()
f = Fernet(key)
token = f.encrypt(b'Rajalakshmi Engineering College')
token
b'...'
f.decrypt(token)
b'Rajalakshmi Engineering College'
key = Fernet.generate_key()
cipher_suite = Fernet(key)
plain_text = b"Rajalakshmi Engineering College."
cipher_text = cipher_suite.encrypt(plain_text)
# Decrypt data
decrypted_text = cipher_suite.decrypt(cipher_text)
print('Original Data:', plain_text)
print('Encrypted Data:', cipher_text)
print('Decrypted Data:', decrypted_text)
```

```
Original Data: b'Rajalakshmi Engineering College.'
Encrypted Data: b'gAAAAABnPxKg1O1E9GwaFScwOGrka4nMIYRFdC_LB77Pf57aCrCAG6OqAxN4xn0LoKiq1qm24A_X0oYF8Qvb9wknXzChnmYKDsiFXVKGeRHV-zS9OgRzf4RBnxFLM8ANr7J0eb
qTYOZ4'
Decrypted Data: b'Rajalakshmi Engineering College.'
```

1. b. Pandas Buit in function; Numpy Buit in fuction- Array slicing,

Ravel,Reshape,ndim

```
import numpy as np
import pandas as pd
list = [[1,'kaif',100],[2,'caelus',98]]
df =pd.DataFrame(list)
print(df)
file=r'C:\Users\KAIF REHMAN\Downloads\diabetes.csv'
filep= pd.read_csv(file)
dfe = pd.DataFrame(filep)
print(dfe.head())
print(dfe.tail())
print(dfe.info())
print(dfe.Glucose.mean())
print(dfe.Glucose.std())
print(dfe.Glucose.var())
```

Output:

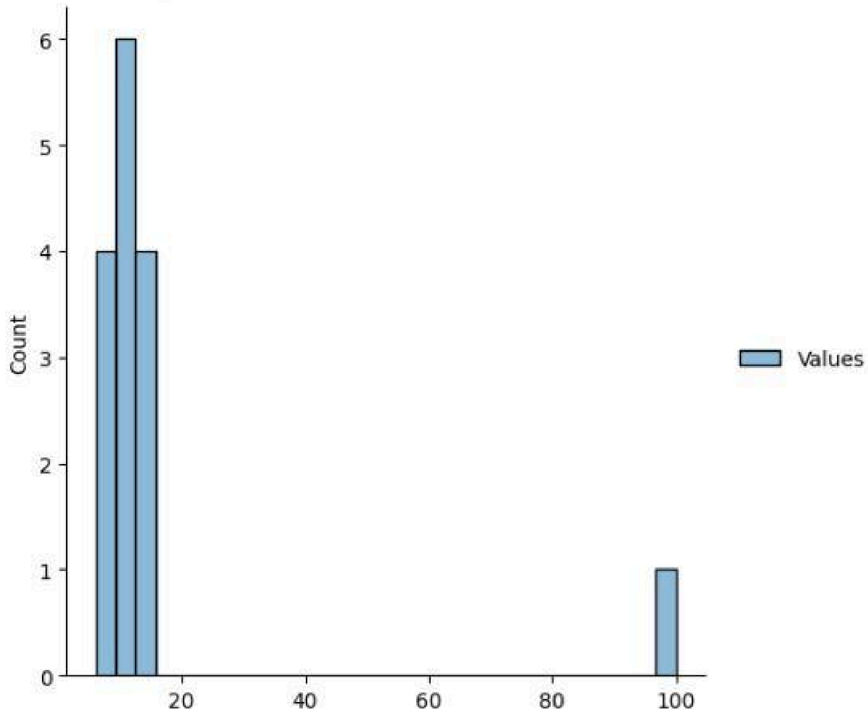
```
   0       1    2
0  1    kaif  100
1  2  caelus   98
   Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
0            6      148             72             35        0  33.6
1            1       85             66             29        0  26.6
2            8      183             64              0        0  23.3
3            1       89             66             23       94  28.1
4            0      137             40             35      168  43.1

   DiabetesPedigreeFunction  Age  Outcome
0                     0.627   50        1
1                     0.351   31        0
2                     0.672   32        1
3                     0.167   21        0
4                     2.288   33        1
     Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
763           10      101             76             48      180  32.9
764            2      122             70             27        0  36.8
765            5      121             72             23      112  26.2
766            1      126             60              0        0  30.1
767            1       93             70             31        0  30.4

     DiabetesPedigreeFunction  Age  Outcome
763                     0.171   63        0
764                     0.340   27        0
765                     0.245   30        0
766                     0.349   47        1
767                     0.315   23        0
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   Pregnancies               768 non-null    int64  
 1   Glucose                   768 non-null    int64  
 2   BloodPressure             768 non-null    int64  
 3   SkinThickness             768 non-null    int64  
 4   Insulin                   768 non-null    int64  
 5   BMI                       768 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64  
 8   Outcome                   768 non-null    int64  
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
None
120.89453125
31.97261819513622
```

2. Outlier detection

Code with output:

```
import pandas as pd
import seaborn as sns

# Sample dataset
arr = {'Values': [10, 12, 15, 14, 10, 11, 100, 13, 12, 9, 10, 14, 8, 7, 6]}
df = pd.DataFrame(arr)

# Outlier detection using IQR
Q1 = df['Values'].quantile(0.25)
Q3 = df['Values'].quantile(0.75)
IQR = Q3 - Q1

lb = Q1 - 1.5 * IQR
ub = Q3 + 1.5 * IQR

print(lb,ub)
sns.displot(arr)
```

```
3.5 19.5
<seaborn.axisgrid.FacetGrid at 0x20448326ab0>
```



3.Missing and inappropriate data

Code:

```
import pandas as pd
import numpy as np

file_path = 'C:\\Users\\KAIF REHMAN\\Downloads\\missing_values\\missing\\pandas_missing_values_dataset.csv'
df = pd.read_csv(file_path)
print(df)
# Replace placeholders with NaN
df.replace(['-', '?'], np.nan, inplace=True)
print(df.duplicated())
# Convert columns to appropriate data types if necessary
df['q1'] = pd.to_numeric(df['q1'], errors='coerce')
df['q4'] = pd.to_numeric(df['q4'], errors='coerce')

# Fill missing values with mean, median, or mode
df['q1'] = df['q1'].fillna(df['q1'].mean())
df['q2'] = df['q2'].fillna(df['q2'].mode()[0])
df['q3'] = df['q3'].fillna(df['q3'].mode()[0])
df['q4'] = df['q4'].fillna(df['q4'].median())

print(df)
print(df.describe())
print(df.isnull().sum())
```

Output:

```
    subject_id    q1               q2     q3  q4
0         1001   7.5            Agree   True   5
1         1002   4.0         Disagree  False   8
2         1003     -                -      -   -
3         1004   7.0   Strongly Agree  False   ?
4         1005     -         Disagree  False   4
5         1006   5.5          Neutral   True   8
6         1007   8.0            Agree      -   7
7         1008  28.0                -  False   9
8         1009     -            Agree  False  12
9         1010     -                -  False   ?
10        1011     -   Strongly Agree  False   1
11        1012     -         Disagree   True   9
12        1013   6.5                -   True   3
13        1014   8.0         Disagree  False   ?
14        1015   8.5          Neutral      2  10
15        1016     -            Agree  False   5
16        1017     9                -   True   7
17        1018   5.5   Strongly Agree  False   8
18        1019   7.0         Disagree  False   5
19        1020   8.0            Agree   True   7
0     False
1     False
2     False
3     False
4     False
5     False
6     False
7     False
8     False
9     False
10    False
11    False
12    False
13    False
14    False
15    False
16    False
17    False
18    False
19    False
dtype: bool
    subject_id         q1               q2     q3    q4
0         1001   7.500000            Agree   True   5.0
1         1002   4.000000         Disagree  False   8.0
2         1003   8.653846            Agree  False   7.0
3         1004   7.000000   Strongly Agree  False   7.0
4         1005   8.653846         Disagree  False   4.0
5         1006   5.500000          Neutral   True   8.0
6         1007   8.000000            Agree  False   7.0
7         1008  28.000000            Agree  False   9.0
8         1009   8.653846            Agree  False  12.0
9         1010   8.653846            Agree  False   7.0
10        1011   8.653846   Strongly Agree  False   1.0
11        1012   8.653846         Disagree   True   9.0
12        1013   6.500000            Agree   True   3.0
13        1014   8.000000         Disagree  False   7.0
14        1015   8.500000          Neutral      2  10.0
15        1016   8.653846            Agree  False   5.0
16        1017   9.000000            Agree   True   7.0
17        1018   5.500000   Strongly Agree  False   8.0
18        1019   7.000000         Disagree  False   5.0
19        1020   8.000000            Agree   True   7.0
       subject_id         q1         q4
count    20.00000  20.000000  20.000000
mean   1010.50000   8.653846   6.800000
std       5.91608   4.750027   2.483631
min    1001.00000   4.000000   1.000000
25%    1005.75000   7.000000   5.000000
50%    1010.50000   8.250000   7.000000
75%    1015.25000   8.653846   8.000000
max    1020.00000  28.000000  12.000000
subject_id    0
q1            0
q2            0
q3            0
q4            0
dtype: int64
```

4. Data

Preprocessing Code :

```python
import pandas as pd
fip='C:\\Users\\KAIF REHMAN\\Downloads\\melb_data\\melb_data.csv'
fp=pd.read_csv(fip)
df=pd.DataFrame(fp)
print(df.head())
df['YearBuilt'] = df['YearBuilt'].fillna(df['YearBuilt'].mode().iloc[0])
df['CouncilArea'] = df['CouncilArea'].fillna(df['CouncilArea'].mode().iloc[0])
df['Bathroom'] = df['Bathroom'].fillna(df['Bathroom'].mode().iloc[0])
print(df.head())
```

Output:

```
   Unnamed: 0     Suburb           Address  Rooms Type      Price Method  \
0           1  Abbotsford      85 Turner St      2    h  1480000.0      S
1           2  Abbotsford   25 Bloomburg St      2    h  1035000.0      S
2           4  Abbotsford      5 Charles St      3    h  1465000.0     SP
3           5  Abbotsford  40 Federation La      3    h   850000.0     PI
4           6  Abbotsford       55a Park St      4    h  1600000.0     VB

  SellerG       Date  Distance  ...  Bathroom  Car  Landsize  BuildingArea  \
0  Biggin  3/12/2016       2.5  ...       1.0  1.0     202.0           NaN
1  Biggin  4/02/2016       2.5  ...       1.0  0.0     156.0          79.0
2  Biggin  4/03/2017       2.5  ...       2.0  0.0     134.0         150.0
3  Biggin  4/03/2017       2.5  ...       2.0  1.0      94.0           NaN
4  Nelson  4/06/2016       2.5  ...       1.0  2.0     120.0         142.0

   YearBuilt  CouncilArea  Lattitude  Longtitude             Regionname  \
0        NaN        Yarra   -37.7996    144.9984  Northern Metropolitan
1     1900.0        Yarra   -37.8079    144.9934  Northern Metropolitan
2     1900.0        Yarra   -37.8093    144.9944  Northern Metropolitan
3        NaN        Yarra   -37.7969    144.9969  Northern Metropolitan
4     2014.0        Yarra   -37.8072    144.9941  Northern Metropolitan

   Propertycount
0         4019.0
1         4019.0
2         4019.0
3         4019.0
4         4019.0

[5 rows x 22 columns]
   Unnamed: 0     Suburb           Address  Rooms Type      Price Method  \
0           1  Abbotsford      85 Turner St      2    h  1480000.0      S
1           2  Abbotsford   25 Bloomburg St      2    h  1035000.0      S
2           4  Abbotsford      5 Charles St      3    h  1465000.0     SP
3           5  Abbotsford  40 Federation La      3    h   850000.0     PI
4           6  Abbotsford       55a Park St      4    h  1600000.0     VB

  SellerG       Date  Distance  ...  Bathroom  Car  Landsize  BuildingArea  \
0  Biggin  3/12/2016       2.5  ...       1.0  1.0     202.0           NaN
1  Biggin  4/02/2016       2.5  ...       1.0  0.0     156.0          79.0
2  Biggin  4/03/2017       2.5  ...       2.0  0.0     134.0         150.0
3  Biggin  4/03/2017       2.5  ...       2.0  1.0      94.0           NaN
4  Nelson  4/06/2016       2.5  ...       1.0  2.0     120.0         142.0

   YearBuilt  CouncilArea  Lattitude  Longtitude             Regionname  \
0     1970.0        Yarra   -37.7996    144.9984  Northern Metropolitan
1     1900.0        Yarra   -37.8079    144.9934  Northern Metropolitan
2     1900.0        Yarra   -37.8093    144.9944  Northern Metropolitan
3     1970.0        Yarra   -37.7969    144.9969  Northern Metropolitan
4     2014.0        Yarra   -37.8072    144.9941  Northern Metropolitan

   Propertycount
0         4019.0
1         4019.0
2         4019.0
3         4019.0
4         4019.0
```

5.EDA-Quantitative and Qualitative plots - Experiments 1

Code:

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
fip='C:\\Users\\KAIF REHMAN\\Downloads\\Social_Network_Ads.csv'
fp=pd.read_csv(fip)
df=pd.DataFrame(fp)
print(df.describe())
print(df.head())
# univariate analysis

df['EstimatedSalary'].hist(bins=20)
plt.title('EstimatedSalary')
plt.show()

df['Age'].hist(bins=20)
plt.title('Age')
plt.show()

# Bivariate Analysis
sns.scatterplot(x='EstimatedSalary', y='Age', data=df)
plt.title('EstimatedSalary vs Age')
plt.show()
numeric_df = df.select_dtypes(include=['float64', 'int64'])

sns.heatmap(numeric_df.corr(), annot=True, cmap='coolwarm')
plt.title('Correlation Matrix')
plt.show()
```

Output :

User ID Age EstimatedSalary Purchased

count 4.000000e+02 400.000000 400.000000
400.000000

mean 1.569154e+07 37.655000 69742.500000
0.357500

std 7.165832e+04 10.482877 34096.960282 0.479864

min 1.556669e+07 18.000000 15000.000000 0.000000

25% 1.562676e+07 29.750000 43000.000000

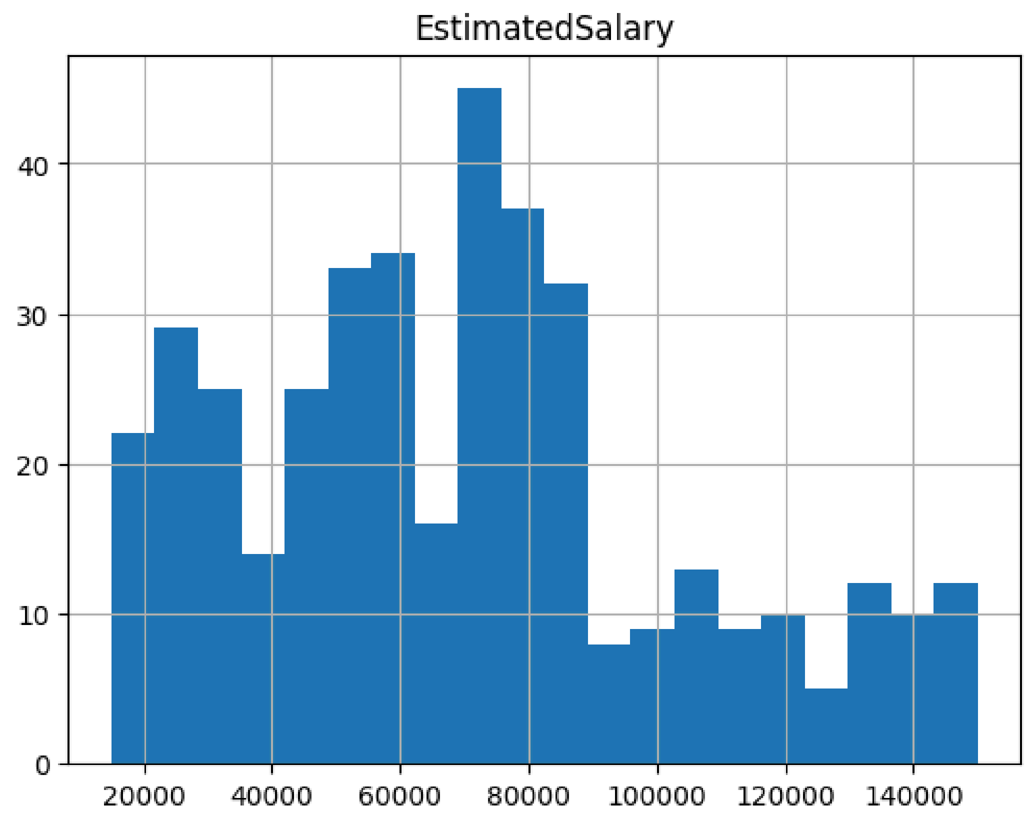
0.000000

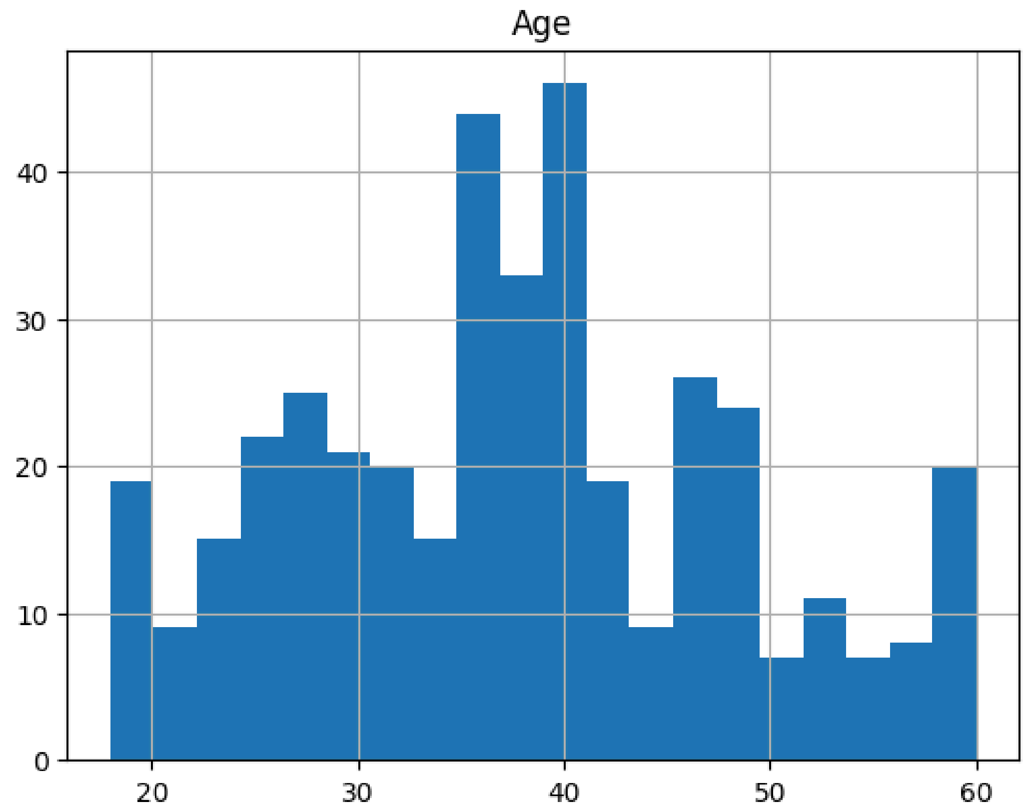
50% 1.569434e+07 37.000000 70000.000000
0.000000

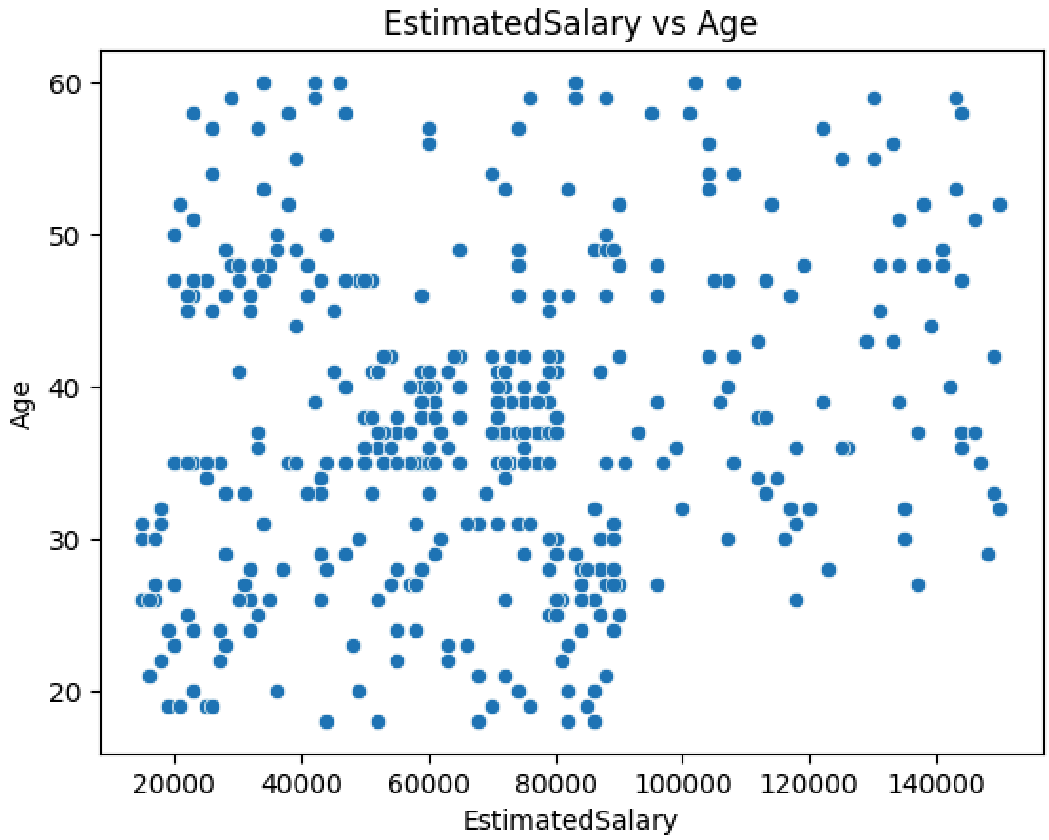
75%   1.575036e+07   46.000000   88000.000000
1.000000

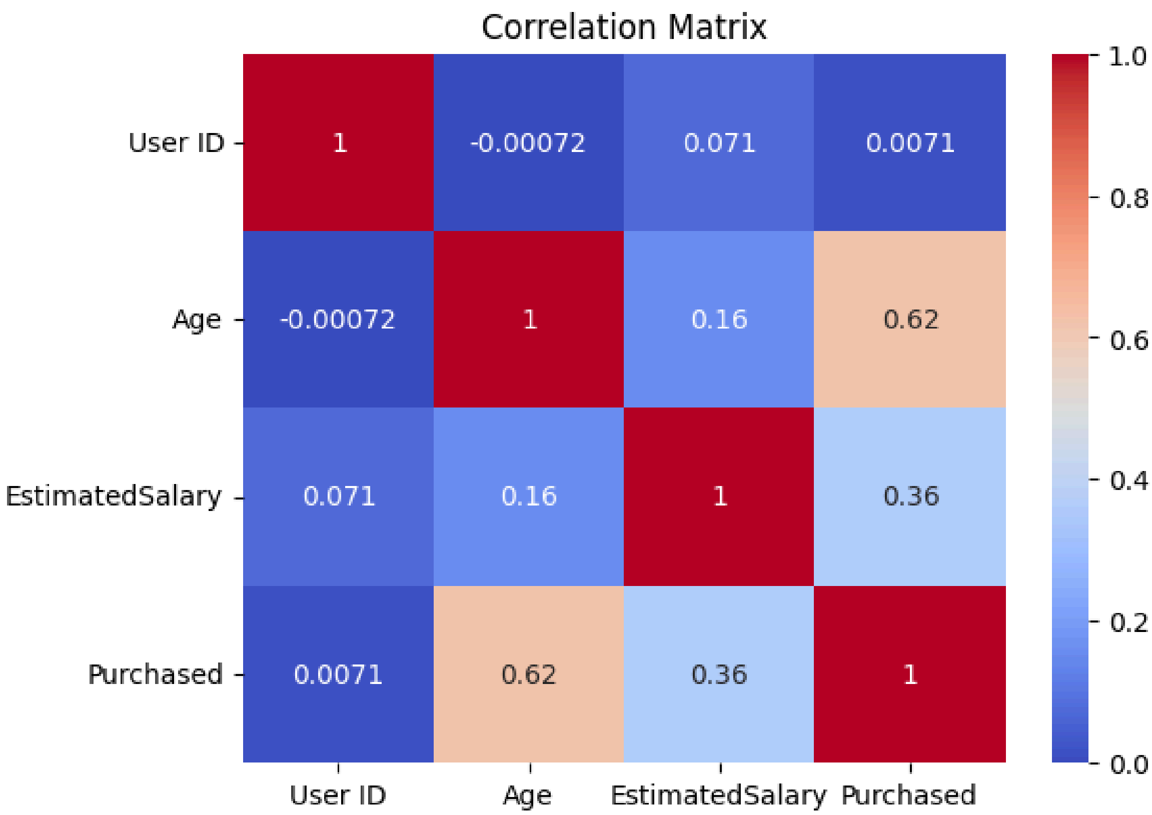
max   1.581524e+07   60.000000   150000.000000
1.000000

	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0




Age
0
10
20
30
40
20
30
40
50
60


EstimatedSalary vs Age
Age
20
30
40
50
60
20000
40000
60000
80000
100000
120000
140000
EstimatedSalary

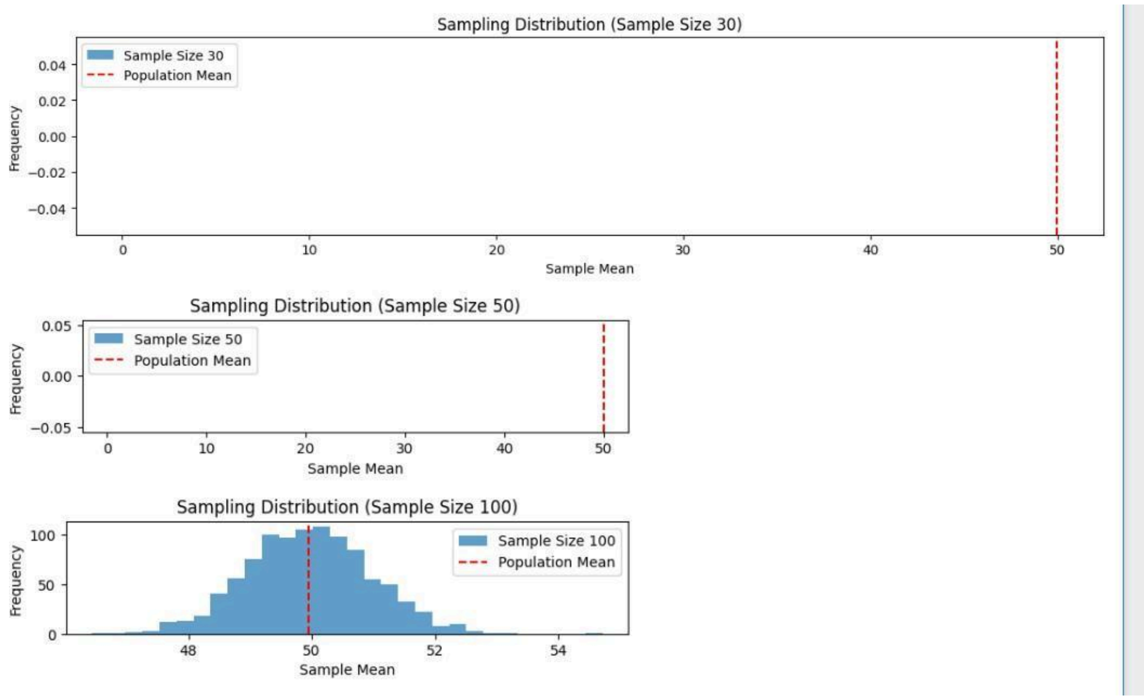


6. Random Sampling and Sampling

Distribution Code:

```python
import numpy as np
import matplotlib.pyplot as plt
# Step 1: Generate a population (e.g., normal distribution)
population_mean = 50
population_std = 10
population_size = 100000
population = np.random.normal(population_mean, population_std, population_size)
# Step 2: Random sampling
sample_sizes = [30, 50, 100] # different sample sizes to consider
num_samples = 1000 # number of samples for each sample size
sample_means = {}
for size in sample_sizes:
    sample_means[size] = []

for _ in range(num_samples):
    sample = np.random.choice(population, size=size, replace=False)
    sample_means[size].append(np.mean(sample))
# Step 3: Plotting sampling distributions
plt.figure(figsize=(12, 8))
for i, size in enumerate(sample_sizes):
    plt.subplot(len(sample_sizes), 1, i+1)
    plt.hist(sample_means[size], bins=30, alpha=0.7, label=f'Sample Size {size}')
    plt.axvline(np.mean(population), color='red', linestyle='dashed', linewidth=1.5,
    label='Population Mean')
    plt.title(f'Sampling Distribution (Sample Size {size})')
    plt.xlabel('Sample Mean')
    plt.ylabel('Frequency')
    plt.legend()
    plt.tight_layout()
    plt.show()
```

Output:



7 Z-Test

Code and Output:

```
import numpy as np
from scipy.stats import norm

# Generate sample data
sample_size = 25
population_mean = 100
population_std = 15  # Known population standard deviation
sample_data = np.random.normal(loc=102, scale=population_std, size=sample_size)

# Calculate sample mean
sample_mean = np.mean(sample_data)

# Calculate the z-statistic
z_statistic = (sample_mean - population_mean) / (population_std / np.sqrt(sample_size))

# Calculate the p-value
p_value = 2 * (1 - norm.cdf(abs(z_statistic)))  # Two-tailed test

# Print results
print(f"Sample Mean: {sample_mean:.2f}")
print(f"Z-Statistic: {z_statistic:.4f}")
print(f"P-Value: {p_value:.4f}")

# Significance level
alpha = 0.05
if p_value < alpha:
    print("Reject the null hypothesis: The average IQ score is significantly different from 100.")
else:
    print("Fail to reject the null hypothesis: There is no significant difference in average IQ score from 100.")
```

```
Sample Mean: 99.85
Z-Statistic: -0.0512
P-Value: 0.9591
Fail to reject the null hypothesis: There is no significant difference in average IQ score from 100.
```

8 T-Test

Code and Output:

```
#34. To test whether the average IQ score of a sample of students differs significantly from a population meanIQ score of 100. Measure the IQ sco

import numpy as np
import scipy.stats as stats
sample_size = 25
sample_data = np.random.normal(loc=102, scale=15,size=sample_size)
population_mean = 100
sample_mean = np.mean(sample_data)
sample_std = np.std(sample_data, ddof=1)

# Number of observations
n = len(sample_data)
t_statistic, p_value = stats.ttest_1samp(sample_data,
population_mean)
# Print results
print(f"Sample Mean: {sample_mean:.2f}")
print(f"T-Statistic: {t_statistic:.4f}")
print(f"P-Value: {p_value:.4f}")
alpha = 0.05
if p_value < alpha:
    print("Reject the null hypothesis: The average IQ score is significantly different from 100.")
else:
    print("Fail to reject the null hypothesis: There is no significant difference in average IQ score from 100.")
```

```
Sample Mean: 105.82
T-Statistic: 2.4858
P-Value: 0.0203
Reject the null hypothesis: The average IQ score is significantly different from 100.
```

9 Anova TEST

Code and Output:

```
import numpy as np
from scipy.stats import f_oneway

# Generate sample data for three groups
group1 = np.random.normal(loc=20, scale=5, size=30)  # Mean = 20, SD = 5
group2 = np.random.normal(loc=22, scale=5, size=30)  # Mean = 22, SD = 5
group3 = np.random.normal(loc=25, scale=5, size=30)  # Mean = 25, SD = 5

# Perform one-way ANOVA
f_statistic, p_value = f_oneway(group1, group2, group3)

# Print results
print(f"F-Statistic: {f_statistic:.4f}")
print(f"P-Value: {p_value:.4f}")

# Significance level
alpha = 0.05
if p_value < alpha:
    print("Reject the null hypothesis: There is a significant difference between the group means.")
else:
    print("Fail to reject the null hypothesis: There is no significant difference between the group means.")
```

```
F-Statistic: 7.9536
P-Value: 0.0007
Reject the null hypothesis: There is a significant difference between the group means.
```

10 Feature

Scaling Code:

```
import pandas as pd
from sklearn.preprocessing import MinMaxScaler, StandardScaler

# Load the dataset
fi = 'C:\\Users\\KAIF REHMAN\\Downloads\\diabetes.csv'
data = pd.read_csv(fi)
print("Print few")
print(data.head())
df=pd.DataFrame(data)
# Min-Max Scaling (scaled between 0 and 1)
min_max_scaler = MinMaxScaler()
scaled_minmax = min_max_scaler.fit_transform(data.iloc[:, :-1])  # Exclude target column

# Standard Scaling (standardize to mean=0 and std=1)
standard_scaler = StandardScaler()
scaled_standard = standard_scaler.fit_transform(data.iloc[:, :-1])  # Exclude target column

# Convert the scaled data back into a DataFrame for better readability
scaled_minmax_df = pd.DataFrame(scaled_minmax, columns=data.columns[:-1])
scaled_standard_df = pd.DataFrame(scaled_standard, columns=data.columns[:-1])

print("\nFirst 5 rows of Min-Max Scaled Data:")
print(scaled_minmax_df.head())

print("\nFirst 5 rows of Standard Scaled Data:")
print(scaled_standard_df.head())
```

Output:

```
Print few
   Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
0            6      148             72             35        0  33.6
1            1       85             66             29        0  26.6
2            8      183             64              0        0  23.3
3            1       89             66             23       94  28.1
4            0      137             40             35      168  43.1

   DiabetesPedigreeFunction  Age  Outcome
0                     0.627   50        1
1                     0.351   31        0
2                     0.672   32        1
3                     0.167   21        0
4                     2.288   33        1

First 5 rows of Min-Max Scaled Data:
   Pregnancies   Glucose  BloodPressure  SkinThickness   Insulin       BMI  \
0     0.352941  0.743719       0.590164       0.353535  0.000000  0.500745
1     0.058824  0.427136       0.540984       0.292929  0.000000  0.396423
2     0.470588  0.919598       0.524590       0.000000  0.000000  0.347243
3     0.058824  0.447236       0.540984       0.232323  0.111111  0.418778
4     0.000000  0.688442       0.327869       0.353535  0.198582  0.642325

   DiabetesPedigreeFunction       Age
0                  0.234415  0.483333
1                  0.116567  0.166667
2                  0.253629  0.183333
3                  0.038002  0.000000
4                  0.943638  0.200000

First 5 rows of Standard Scaled Data:
   Pregnancies   Glucose  BloodPressure  SkinThickness   Insulin       BMI  \
0     0.639947  0.848324       0.149641       0.907270 -0.692891  0.204013
1    -0.844885 -1.123396      -0.160546       0.530902 -0.692891 -0.684422
2     1.233880  1.943724      -0.263941      -1.288212 -0.692891 -1.103255
3    -0.844885 -0.998208      -0.160546       0.154533  0.123302 -0.494043
4    -1.141852  0.504055      -1.504687       0.907270  0.765836  1.409746

   DiabetesPedigreeFunction       Age
0                  0.468492  1.425995
1                 -0.365061 -0.190672
2                  0.604397 -0.105584
3                 -0.920763 -1.041549
4                  5.484909 -0.020496
```

11 Linear Regression

Code:

```python
from sklearn.datasets import load_diabetes
import pandas as pd

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
diabetes = load_diabetes()

df = pd.DataFrame(data=diabetes.data, columns=diabetes.feature_names)
df['target'] = diabetes.target
print(df.head())
X = df.drop('target', axis=1)
y = df['target']

# Split data into training and testing sets (80% training, 20% testing)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create the Linear Regression model
linear_model = LinearRegression()

# Train the model
linear_model.fit(X_train, y_train)

# Predict on test data
y_pred_linear = linear_model.predict(X_test)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred_linear)
r2 = r2_score(y_test, y_pred_linear)

print("\nLinear Regression Results:")
print("Mean Squared Error (MSE):", mse)
print("R-squared (R²):", r2)
```

Output:

```
        age       sex       bmi        bp        s1        s2        s3  \
0  0.038076  0.050680  0.061696  0.021872 -0.044223 -0.034821 -0.043401
1 -0.001882 -0.044642 -0.051474 -0.026328 -0.008449 -0.019163  0.074412
2  0.085299  0.050680  0.044451 -0.005670 -0.045599 -0.034194 -0.032356
3 -0.089063 -0.044642 -0.011595 -0.036656  0.012191  0.024991 -0.036038
4  0.005383 -0.044642 -0.036385  0.021872  0.003935  0.015596  0.008142

         s4        s5        s6  target
0 -0.002592  0.019907 -0.017646   151.0
1 -0.039493 -0.068332 -0.092204    75.0
2 -0.002592  0.002861 -0.025930   141.0
3  0.034309  0.022688 -0.009362   206.0
4 -0.002592 -0.031988 -0.046641   135.0

Linear Regression Results:
Mean Squared Error (MSE): 2900.19362849348
R-squared (R²): 0.4526027629719197
```

12 Logistic

Regression Code:

```python
from sklearn.datasets import load_diabetes
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix

# Load the diabetes dataset
diabetes = load_diabetes()
df = pd.DataFrame(data=diabetes.data, columns=diabetes.feature_names)
df['target'] = diabetes.target
print(df.head())
# Features (X) and target (y)
X = df.drop('target', axis=1)
y = df['target']

# Convert target variable to binary classification
median_target = y.median()
y_binary = (y > median_target).astype(int)

# Split data into training and testing sets (80% training, 20% testing) for classification
X_train_bin, X_test_bin, y_train_bin, y_test_bin = train_test_split(X, y_binary, test_size=0.2, random_state=42)
print("\nBinary Target:")
print(y_binary.head())
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix
# Create Logistic Regression model
logistic_model = LogisticRegression(max_iter=200)
# Train the model
logistic_model.fit(X_train_bin, y_train_bin)
# Predict on test data
y_pred_logistic = logistic_model.predict(X_test_bin)
# Evaluate the model
accuracy = accuracy_score(y_test_bin, y_pred_logistic)
conf_matrix = confusion_matrix(y_test_bin, y_pred_logistic)
print("\nLogistic Regression Results:")
print("Accuracy:", accuracy)
print("Confusion Matrix:\n", conf_matrix)
```

Output:

```
        age       sex       bmi        bp        s1        s2        s3  \
0  0.038076  0.050680  0.061696  0.021872 -0.044223 -0.034821 -0.043401
1 -0.001882 -0.044642 -0.051474 -0.026328 -0.008449 -0.019163  0.074412
2  0.085299  0.050680  0.044451 -0.005670 -0.045599 -0.034194 -0.032356
3 -0.089063 -0.044642 -0.011595 -0.036656  0.012191  0.024991 -0.036038
4  0.005383 -0.044642 -0.036385  0.021872  0.003935  0.015596  0.008142

         s4        s5        s6  target
0 -0.002592  0.019907 -0.017646   151.0
1 -0.039493 -0.068332 -0.092204    75.0
2 -0.002592  0.002861 -0.025930   141.0
3  0.034309  0.022688 -0.009362   206.0
4 -0.002592 -0.031988 -0.046641   135.0

Binary Target:
0    1
1    0
2    1
3    1
4    0
Name: target, dtype: int64

Logistic Regression Results:
Accuracy: 0.7415730337078652
Confusion Matrix:
 [[37 12]
 [11 29]]
```

======================THANK YOU======================