Project : Customer Churn

Dataset 1. Data Manipulation:

● Extract the 5th column and store it in ‘customer_5

Code:-

```
import pandas as pd

# Load the dataset (example: 'data.csv')
data = pd.read_csv('/content/customer_churn (1).csv')

# Extract the 5th column and store it in 'customer_5'
customer_5 = data.iloc[:, 4]  # Indexing starts from 0, so the 5th column is index 4

# Print or verify the extracted column
print(customer_5)
```

Ans:-

```
0        No
1        No
2        No
3        No
4        No
       ...
7038    Yes
7039    Yes
7040    Yes
7041     No
7042     No
Name: Dependents, Length: 7043, dtype: object
```

● Extract the 15th column and store it in ‘customer_15’ Code:-

```
import pandas as pd

# Load the dataset (example: 'data.csv')
data = pd.read_csv('/content/customer_churn (1).csv')

# Extract the 15th column and store it in 'customer_15'
customer_15 = data.iloc[:, 14]  # Indexing starts from 0, so the 15th column is index 14

# Print or verify the extracted column
print(customer_15)
```

Ans :-

```
0        No
1        No
2        No
3        No
4        No
       ...
7038    Yes
7039    Yes
7040     No
7041     No
7042    Yes
Name: StreamingMovies, Length: 7043, dtype: object
```

● Extract all the male senior cizens whose payment method is electronic check and store the result in ‘senior_male_electronic’

Code:-

```
import pandas as pd

# Load the dataset (example: 'data.csv')
data = pd.read_csv('/content/customer_churn (1).csv')

# Filter male senior citizens with payment method "Electronic check"
senior_male_electronic = data[(data['gender'] == 'Male') &
                              (data['SeniorCitizen'] == 1) &
                              (data['PaymentMethod'] == 'Electronic check')]

# Print or verify the filtered result
print(senior_male_electronic)
```

Ans :-

```
      customerID gender  SeniorCitizen Partner Dependents  tenure  \
20    8779-QRDMV   Male              1      No         No       1
55    1658-BYGOY   Male              1      No         No      18
57    5067-XJQFU   Male              1     Yes        Yes      66
78    0191-ZHSKZ   Male              1      No         No      30
91    2424-WVHPL   Male              1      No         No       1
...          ...    ...            ...     ...        ...     ...
6837  6229-LSCKB   Male              1      No         No       6
6894  1400-MMYXY   Male              1     Yes         No       3
6914  7142-HVGBG   Male              1     Yes         No      43
6967  8739-WWKDU   Male              1      No         No      25
7032  6894-LFHLY   Male              1      No         No       1

     PhoneService     MultipleLines InternetService OnlineSecurity  ...  \
20             No  No phone service             DSL             No  ...
55            Yes               Yes     Fiber optic             No  ...
57            Yes               Yes     Fiber optic             No  ...
78            Yes                No             DSL            Yes  ...
91            Yes                No     Fiber optic             No  ...
...           ...               ...             ...            ...  ...
6837          Yes                No     Fiber optic             No  ...
6894          Yes               Yes     Fiber optic             No  ...
6914          Yes               Yes     Fiber optic             No  ...
6967          Yes               Yes     Fiber optic             No  ...
7032          Yes               Yes     Fiber optic             No  ...

     DeviceProtection TechSupport StreamingTV StreamingMovies        Contract  \
20                Yes          No          No              Yes  Month-to-month
55                 No          No         Yes              Yes  Month-to-month
57                Yes         Yes         Yes              Yes        One year
78                 No          No         Yes              Yes  Month-to-month
91                 No         Yes          No               No  Month-to-month
...               ...         ...         ...              ...             ...
6837               No          No         Yes               No  Month-to-month
6894              Yes          No         Yes              Yes  Month-to-month
6914              Yes          No         Yes              Yes  Month-to-month
6967               No          No         Yes               No  Month-to-month
7032               No          No          No               No  Month-to-month

     PaperlessBilling     PaymentMethod MonthlyCharges  TotalCharges Churn
20                Yes  Electronic check          39.65         39.65   Yes
55                Yes  Electronic check          95.45       1752.55   Yes
57                Yes  Electronic check         108.45       7076.35    No
78                Yes  Electronic check          74.75       2111.3     No
91                 No  Electronic check          74.70         74.7     No
...               ...               ...            ...          ...    ...
6837              Yes  Electronic check          79.70         497.6     No
6894              Yes  Electronic check         105.90        334.65   Yes
6914              Yes  Electronic check         103.00        4414.3   Yes
6967              Yes  Electronic check          89.50       2196.15   Yes
7032              Yes  Electronic check          75.75         75.75   Yes

[298 rows x 21 columns]
```

● Extract all those customers whose tenure is greater than 70 months or their monthly charges is more than $100 and store the result in ‘customer_total_tenure’

Code:-

```
import pandas as pd

# Sample data (replace this with your actual data loading step)
data = {
    'CustomerID': [1, 2, 3, 4, 5],
    'Tenure': [50, 80, 60, 75, 40],
    'MonthlyCharges': [90, 110, 70, 120, 100]
}
df = pd.DataFrame(data)

# Extract customers meeting the conditions
customer_total_tenure = df[(df['Tenure'] > 70) | (df['MonthlyCharges'] > 100)]

# Display the result
print(customer_total_tenure)
```

Ans :-

```
   CustomerID  Tenure  MonthlyCharges
1           2      80             110
3           4      75             120
```

● Extract all the customers whose contract is of two years, payment method is mailed check and the value of churn is ‘Yes’ and store the result in ‘two_mail_yes’

Code :-

```
import pandas as pd

# Sample data (replace this with your actual data loading step)
data = {
    'CustomerID': [1, 2, 3, 4, 5],
    'Contract': ['Two years', 'Month-to-month', 'Two years', 'Two years', 'One year'],
    'PaymentMethod': ['Mailed check', 'Electronic check', 'Mailed check', 'Mailed check', 'Bank transfer'],
    'Churn': ['No', 'Yes', 'Yes', 'No', 'Yes']
}
df = pd.DataFrame(data)

# Extract customers meeting the conditions
two_mail_yes = df[
    (df['Contract'] == 'Two years') &
    (df['PaymentMethod'] == 'Mailed check') &
    (df['Churn'] == 'Yes')
]

# Display the result
print(two_mail_yes)
```

```
   CustomerID   Contract PaymentMethod Churn
2           3  Two years  Mailed check   Yes
```

Ans :- CustomerID Contract Payment CustomerID Contract PaymentMethod Churn

2 3 Two Year Mailed Check Yes

● Extract 333 random records from the customer_churndataframe and store the result in ‘customer_333’

Code:-

```
import pandas as pd

# Assuming customer_churndataframe is already loaded
# For demonstration, replace this with your actual DataFrame
customer_churndataframe = pd.DataFrame({
    'CustomerID': range(1, 1001),  # Example data
    'Feature': ['SampleData'] * 1000
})

# Extract 333 random records
customer_333 = customer_churndataframe.sample(n=333, random_state=42)  # random_state for reproducibility

# Display the result
print(customer_333)
```

Ans :-

```
     CustomerID     Feature
521         522  SampleData
737         738  SampleData
740         741  SampleData
660         661  SampleData
411         412  SampleData
..          ...         ...
711         712  SampleData
133         134  SampleData
703         704  SampleData
311         312  SampleData
722         723  SampleData

[333 rows x 2 columns]
```

● Get the count of different levels from the ‘Churn’ column Code:-

```
import pandas as pd

# Sample data (replace this with your actual data)
data = {
    'CustomerID': [1, 2, 3, 4, 5],
    'Churn': ['Yes', 'No', 'No', 'Yes', 'No']
}
df = pd.DataFrame(data)

# Get the count of different levels in the 'Churn' column
churn_counts = df['Churn'].value_counts()

# Display the result
print(churn_counts)
```

Ans :-

```
Churn
No     3
Yes    2
Name: count, dtype: int64
```

2. Data Visualization:

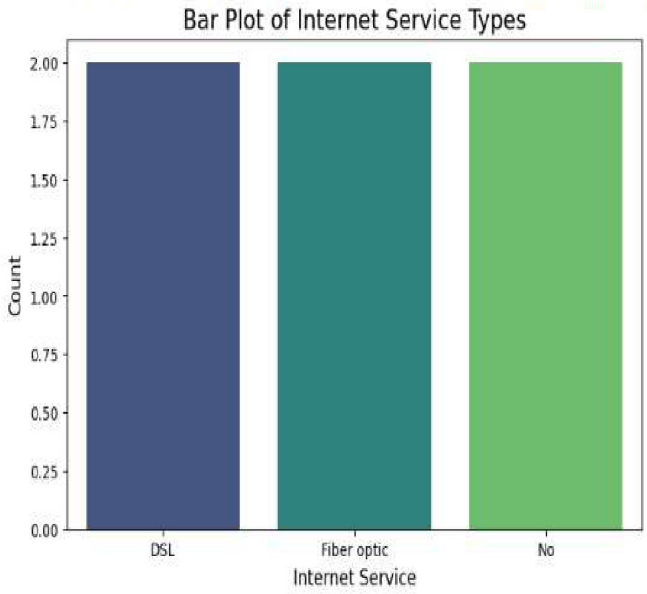
● Build a bar-plot for the ’InternetService’ column:

Code :-

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Sample data (replace this with your actual data)
data = {
    'CustomerID': [1, 2, 3, 4, 5, 6],
    'InternetService': ['DSL', 'Fiber optic', 'No', 'DSL', 'Fiber optic', 'No']
}
df = pd.DataFrame(data)

# Count the occurrences of each category in the 'InternetService' column
internet_service_counts = df['InternetService'].value_counts()

# Create a bar plot
plt.figure(figsize=(8, 5))
sns.barplot(x=internet_service_counts.index, y=internet_service_counts.values, palette='viridis')

# Customize the plot
plt.title('Bar Plot of Internet Service Types', fontsize=16)
plt.xlabel('Internet Service', fontsize=12)
plt.ylabel('Count', fontsize=12)
plt.xticks(fontsize=10)
plt.yticks(fontsize=10)
plt.show()
```

Ans :-

```
<ipython-input-7-3128da88bacb>:17: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

  sns.barplot(x=internet_service_counts.index, y=internet_service_counts.values, palette='viridis')
```



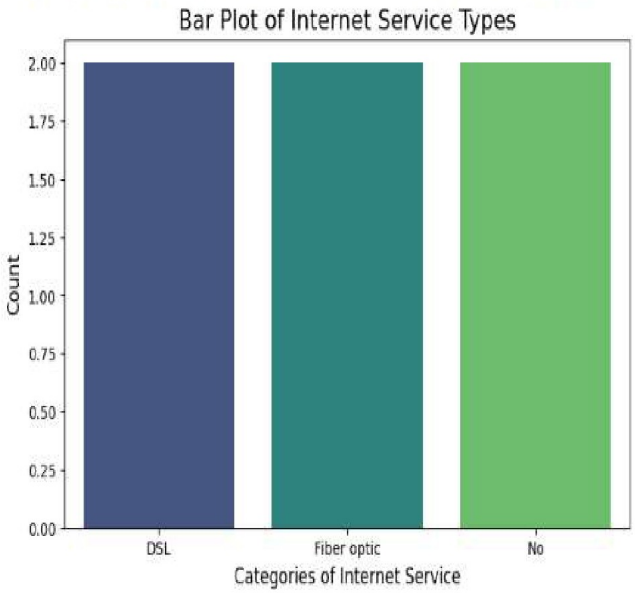
a. Set x-axis label to ‘Categories of Internet Service’

Code:-

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Sample data (replace this with your actual data)
data = {
    'CustomerID': [1, 2, 3, 4, 5, 6],
    'InternetService': ['DSL', 'Fiber optic', 'No', 'DSL', 'Fiber optic', 'No']
}
df = pd.DataFrame(data)

# Count the occurrences of each category in the 'InternetService' column
internet_service_counts = df['InternetService'].value_counts()

# Create a bar plot
plt.figure(figsize=(8, 5))
sns.barplot(x=internet_service_counts.index, y=internet_service_counts.values, palette='viridis')

# Customize the plot
plt.title('Bar Plot of Internet Service Types', fontsize=16)
plt.xlabel('Categories of Internet Service', fontsize=12)  # Set the x-axis label
plt.ylabel('Count', fontsize=12)
plt.xticks(fontsize=10)
plt.yticks(fontsize=10)
plt.show()
```

Ans ;-

```
<ipython-input-8-2fd4b0fc7948>:17: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

  sns.barplot(x=internet_service_counts.index, y=internet_service_counts.values, palette='viridis')
```



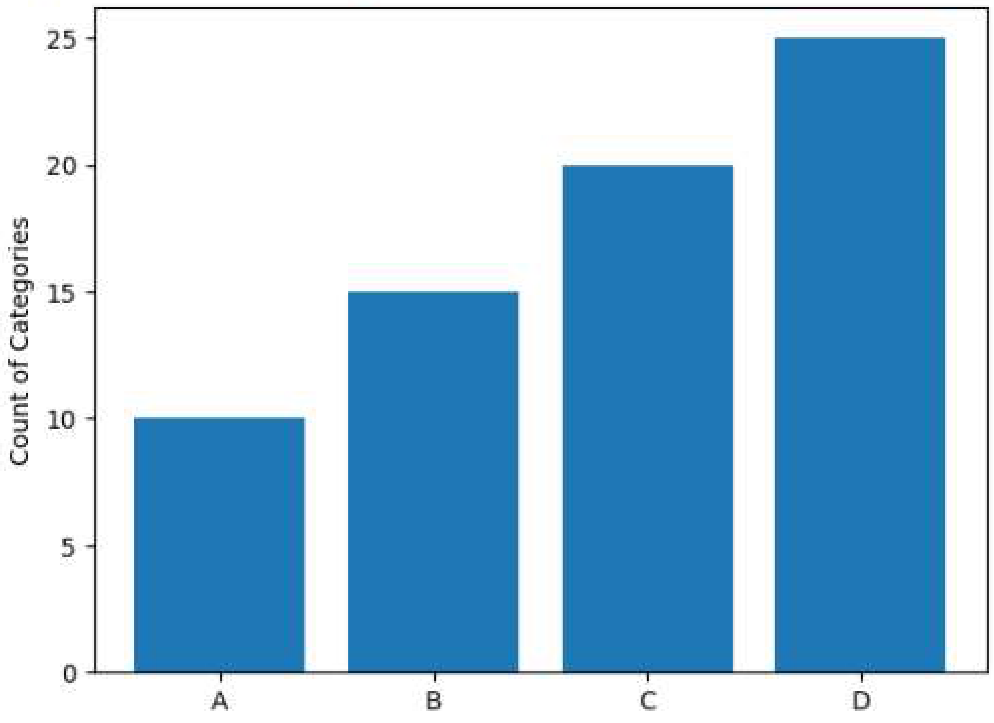
b. Set y-axis label to ‘Count of Categories’

Code:-

```
import matplotlib.pyplot as plt

# Example data
categories = ['A', 'B', 'C', 'D']
counts = [10, 15, 20, 25]

# Creating the bar chart
plt.bar(categories, counts)

# Setting the y-axis label
plt.ylabel('Count of Categories')

#
```

Ans:-

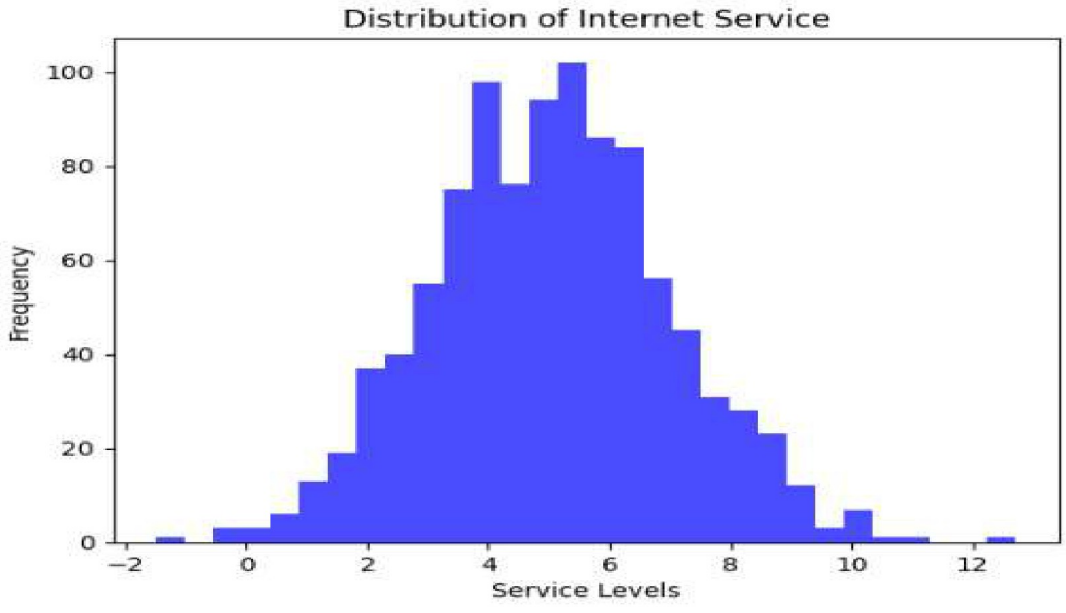
```
Text(0, 0.5, 'Count of Categories')
```



c. Set the title of plot to be ‘Distribution of Internet Service’ Code:-

```
import matplotlib.pyplot as plt
import numpy as np

# Generate some example data
np.random.seed(42)
data = np.random.normal(loc=5, scale=2, size=1000)  # example normal distribution

# Create a histogram
plt.hist(data, bins=30, color='blue', alpha=0.7)

# Set the title
plt.title('Distribution of Internet Service')

# Labeling the axes (optional)
plt.xlabel('Service Levels')
plt.ylabel('Frequency')

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

Ans:-

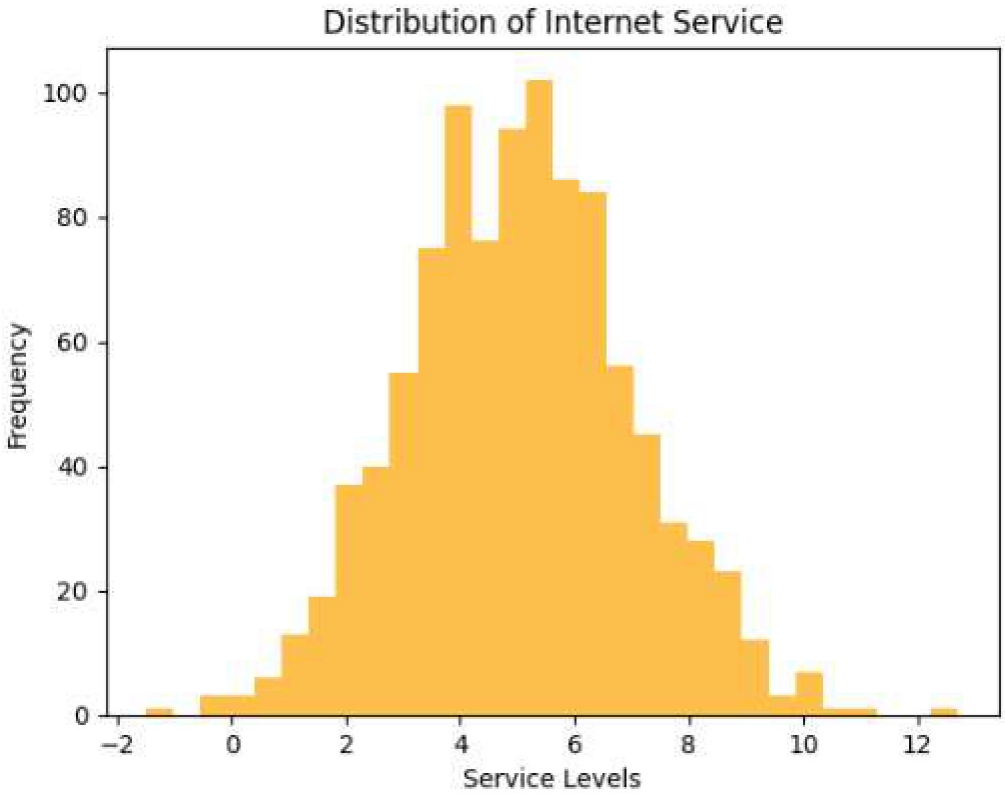


d. Set the color of the bars to be ‘orange’

Code:-

```
import matplotlib.pyplot as plt
import numpy as np

# Generate some example data
np.random.seed(42)
data = np.random.normal(loc=5, scale=2, size=1000)  # example normal distribution

# Create a histogram with orange bars
plt.hist(data, bins=30, color='orange', alpha=0.7)

# Set the title
plt.title('Distribution of Internet Service')

# Labeling the axes (optional)
plt.xlabel('Service Levels')
plt.ylabel('Frequency')

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

Ans :-


Distribution of Internet Service
Frequency
Service Levels
0
20
40
60
80
100
−2
0
2
4
6
8
10
12

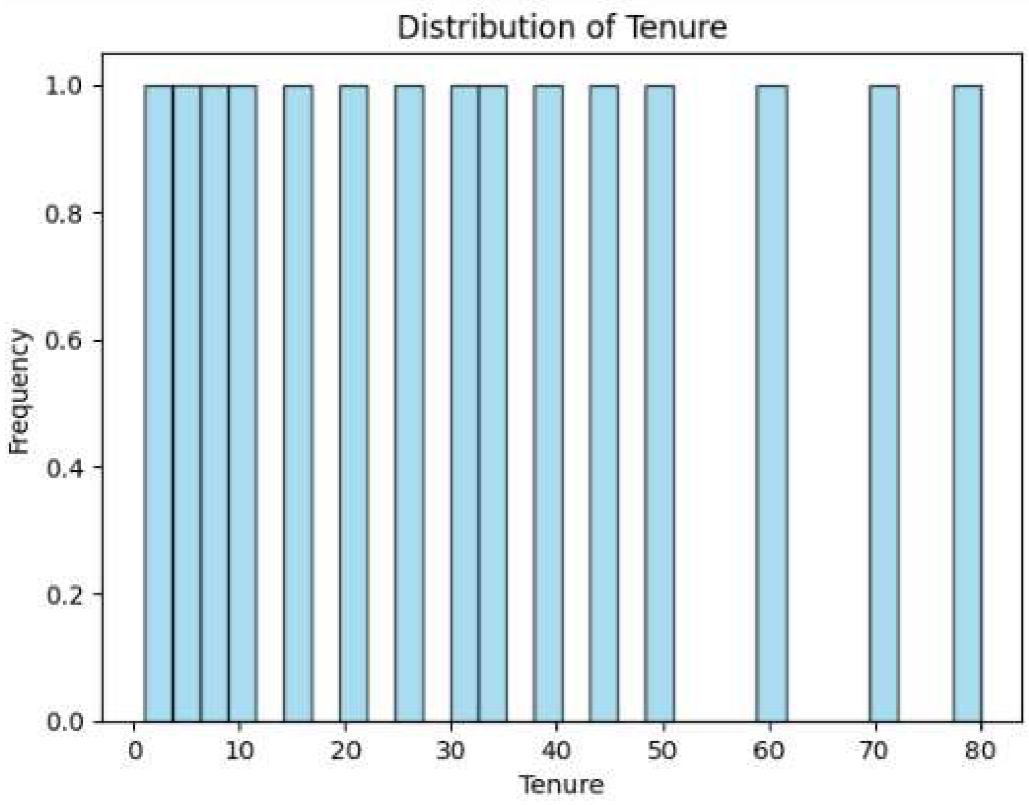
● Build a histogram for the ‘tenure’ column:

a. Set the number of bins to be 30

Code:-

```
import pandas as pd
import matplotlib.pyplot as plt

# Example DataFrame (replace with your actual dataset)
data = {'tenure': [1, 5, 8, 10, 15, 20, 25, 30, 35, 40, 45, 50, 60, 70, 80]}
df = pd.DataFrame(data)

# Create a histogram for the 'tenure' column with 30 bins
plt.hist(df['tenure'], bins=30, color='skyblue', edgecolor='black', alpha=0.7)

# Set title and labels
plt.title('Distribution of Tenure')
plt.xlabel('Tenure')
plt.ylabel('Frequency')

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

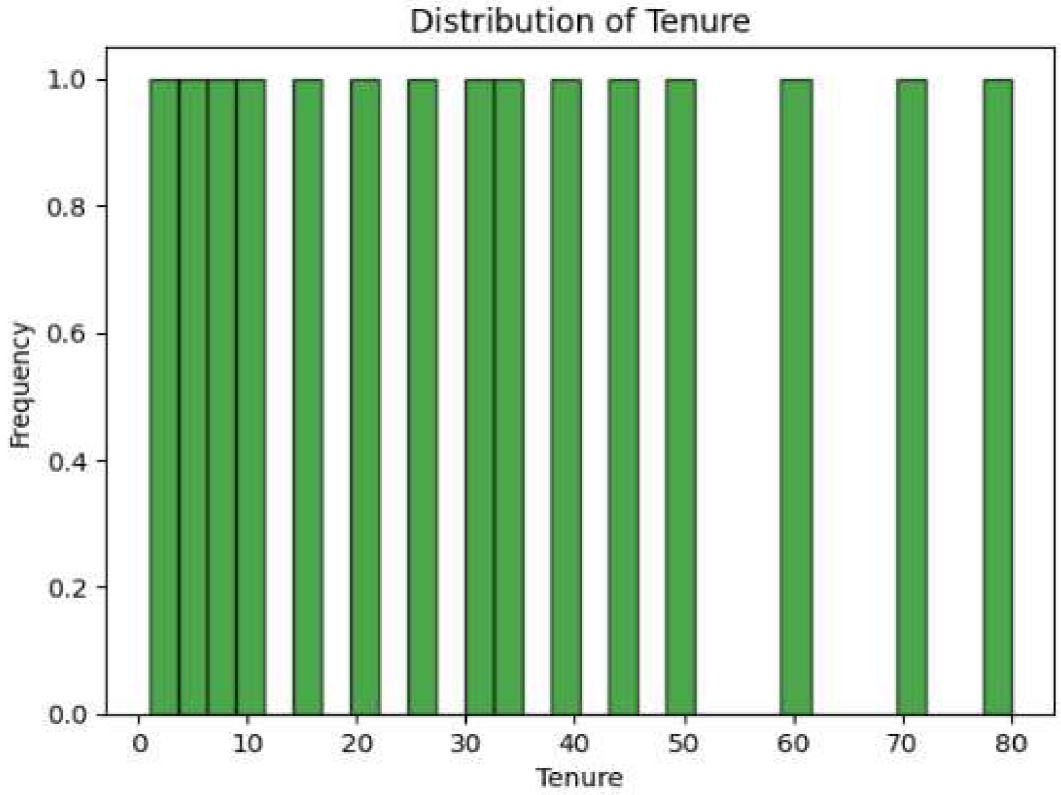
Ans:-


Distribution of Tenure
Frequency
Tenure
0.0
0.2
0.4
0.6
0.8
1.0
0
10
20
30
40
50
60
70
80

b. Set the color of the bins to be
‘green’
Code:--

```
import pandas as pd
import matplotlib.pyplot as plt

# Example DataFrame (replace with your actual dataset)
data = {'tenure': [1, 5, 8, 10, 15, 20, 25, 30, 35, 40, 45, 50, 60, 70, 80]}
df = pd.DataFrame(data)

# Create a histogram for the 'tenure' column with 30 bins and green color
plt.hist(df['tenure'], bins=30, color='green', edgecolor='black', alpha=0.7)

# Set title and labels
plt.title('Distribution of Tenure')
plt.xlabel('Tenure')
plt.ylabel('Frequency')

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

Ans:-

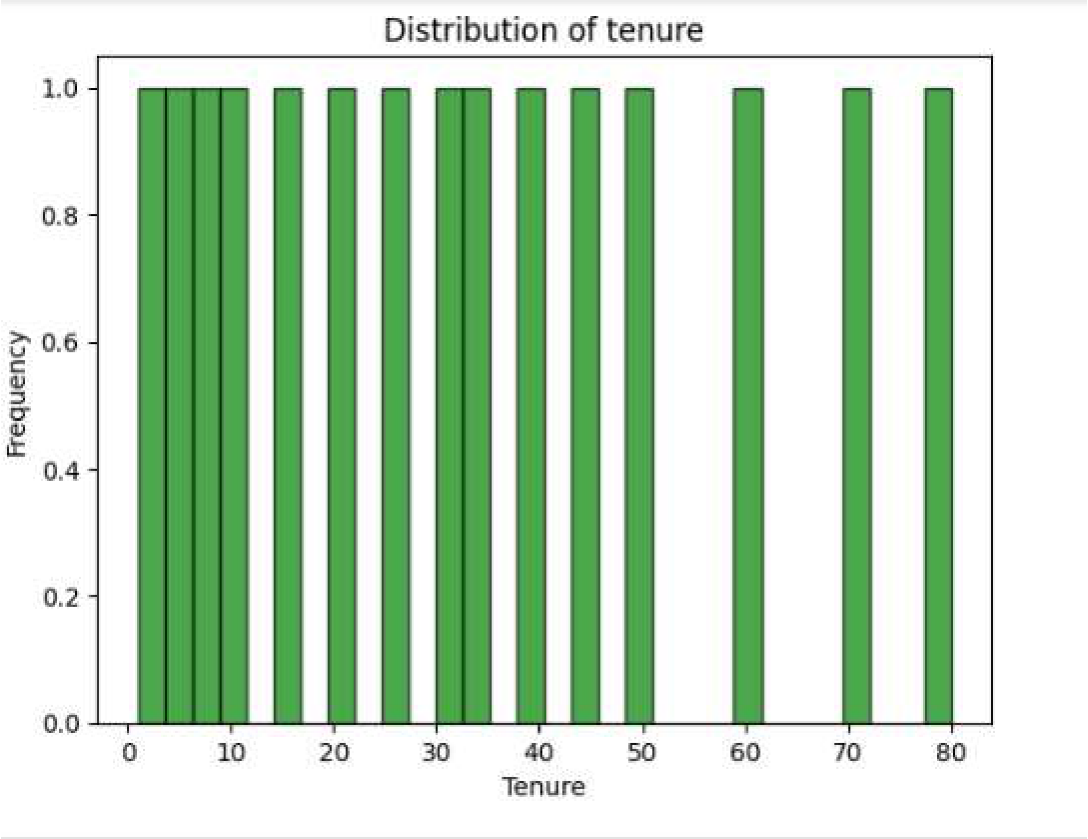

Distribution of Tenure
Frequency
1.0
0.8
0.6
0.4
0.2
0.0
0
10
20
30
40
50
60
70
80
Tenure

c. Assign the title ‘Distribution of tenure’

Code:-

```
import pandas as pd
import matplotlib.pyplot as plt

# Example DataFrame (replace with your actual dataset)
data = {'tenure': [1, 5, 8, 10, 15, 20, 25, 30, 35, 40, 45, 50, 60, 70, 80]}
df = pd.DataFrame(data)

# Create a histogram for the 'tenure' column with 30 bins and green color
plt.hist(df['tenure'], bins=30, color='green', edgecolor='black', alpha=0.7)

# Set the title
plt.title('Distribution of tenure')

# Set labels for the axes
plt.xlabel('Tenure')
plt.ylabel('Frequency')

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

Ans:-


Distribution of tenure
Frequency
Tenure
0.0
0.2
0.4
0.6
0.8
1.0
0
10
20
30
40
50
60
70
80

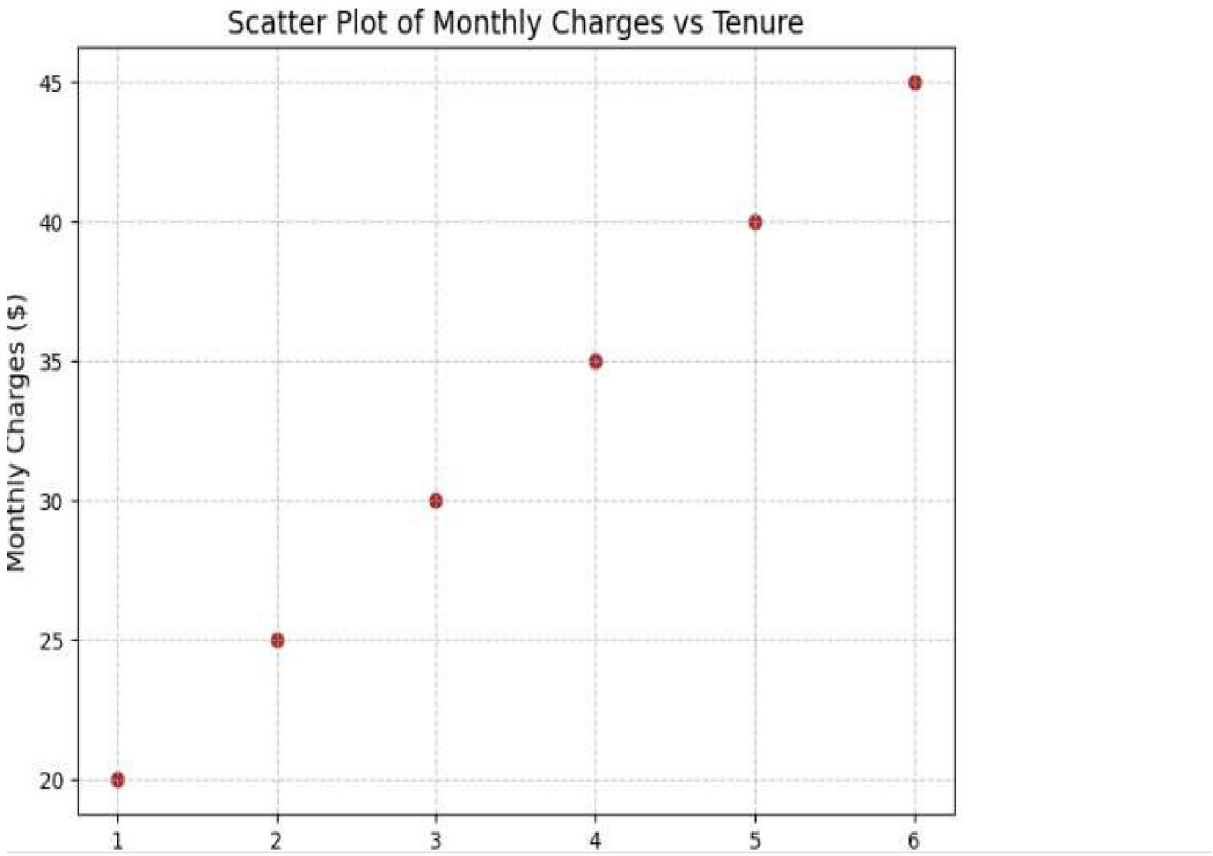
● Build a scaer-plot between ‘MonthlyCharges’ and ‘tenure’. Map plot between ‘MonthlyCharges’ and ‘tenure’. Map ‘MonthlyCharges’ to the y‘MonthlyCharges’ to the y-axis and
‘tenure’ to the ‘x-axis’:

a. Assign the points a color of ‘brown’

Code:-

```
import matplotlib.pyplot as plt

# Sample data (replace with your dataset)
# Replace these lists with your actual data for MonthlyCharges and tenure
tenure = [1, 2, 3, 4, 5, 6]  # Example data for tenure
monthly_charges = [20, 25, 30, 35, 40, 45]  # Example data for MonthlyCharges

# Create the scatter plot
plt.figure(figsize=(8, 6))
plt.scatter(tenure, monthly_charges, color='brown')

# Label the axes
plt.xlabel('Tenure (months)', fontsize=12)
plt.ylabel('Monthly Charges ($)', fontsize=12)

# Title of the plot
plt.title('Scatter Plot of Monthly Charges vs Tenure', fontsize=14)

# Show the plot
plt.grid(True, linestyle='--', alpha=0.7)
plt.show()
```

Ans:-

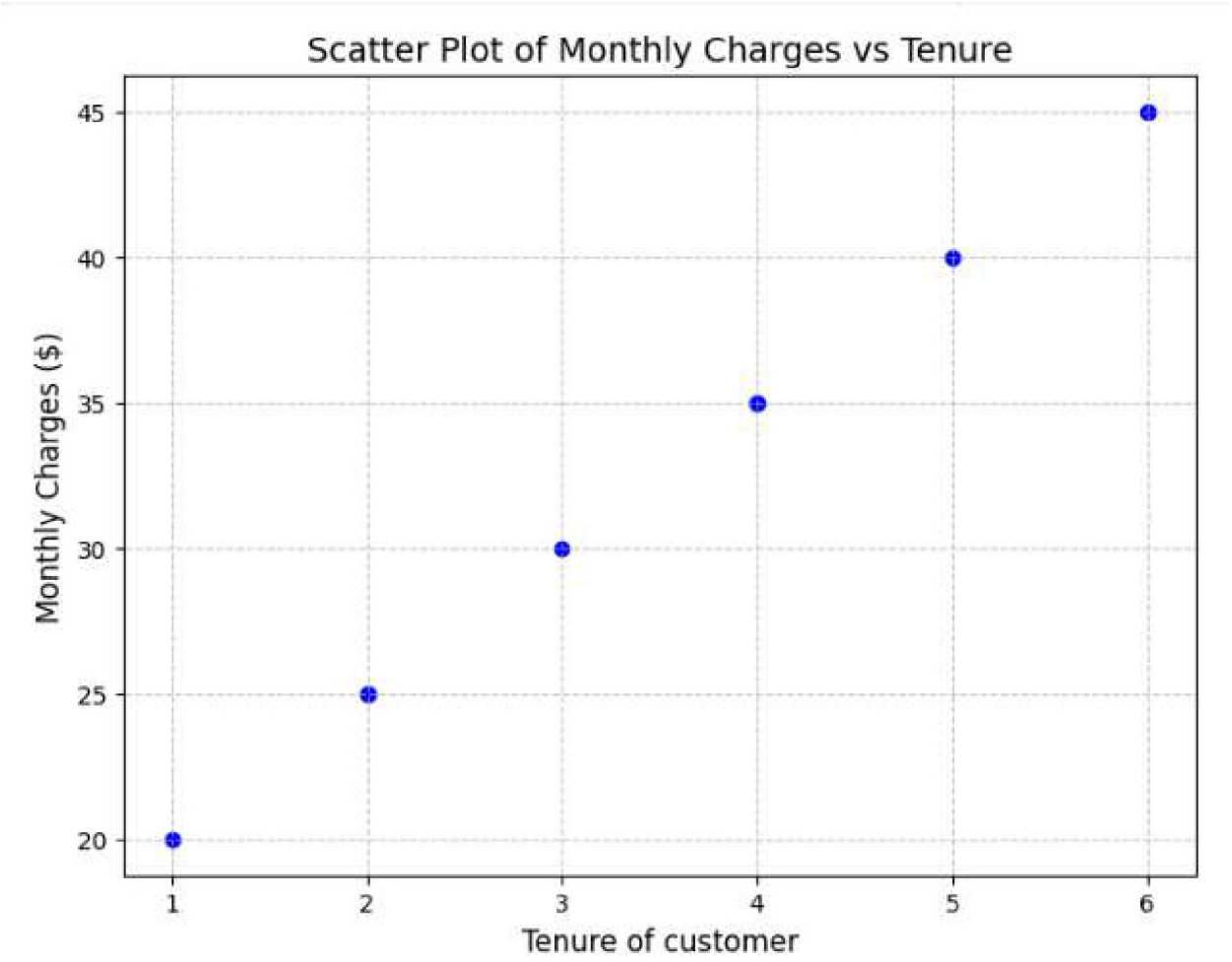

Scatter Plot of Monthly Charges vs Tenure
Monthly Charges ($)
45
40
35
30
25
20
1
2
3
4
5
6

b. Set the x-axis label to ‘Tenure of customer’

Code:-

```
import matplotlib.pyplot as plt

# Sample data (replace with your dataset)
# Replace these lists with your actual data for MonthlyCharges and tenure
tenure = [1, 2, 3, 4, 5, 6]  # Example data for tenure
monthly_charges = [20, 25, 30, 35, 40, 45]  # Example data for MonthlyCharges

# Create the scatter plot
plt.figure(figsize=(8, 6))
plt.scatter(tenure, monthly_charges, color='blue')  # Updated color to 'blue'

# Label the axes
plt.xlabel('Tenure of customer', fontsize=12)
plt.ylabel('Monthly Charges ($)', fontsize=12)

# Title of the plot
plt.title('Scatter Plot of Monthly Charges vs Tenure', fontsize=14)

# Show the plot
plt.grid(True, linestyle='--', alpha=0.7)
plt.show()
```

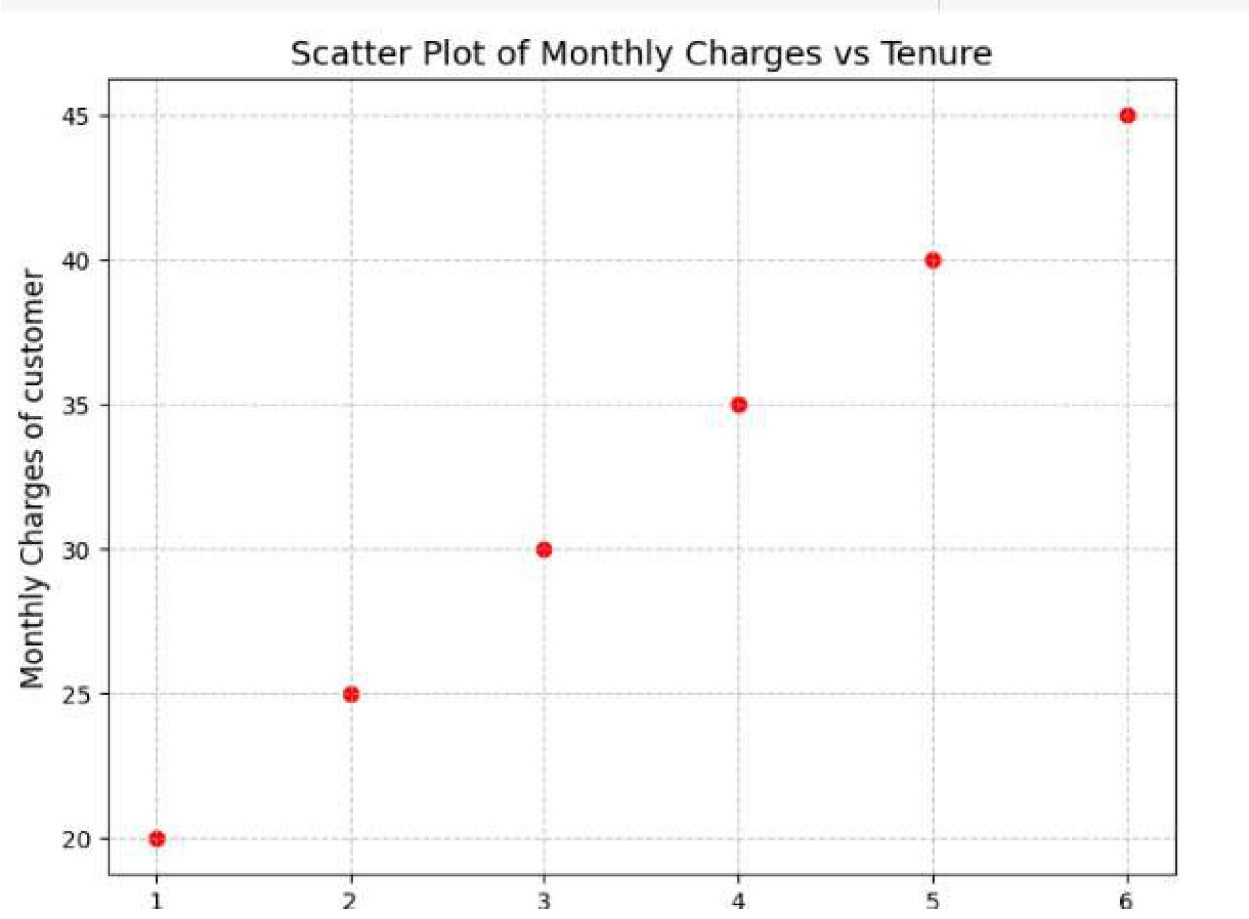
Ans:-


Scatter Plot of Monthly Charges vs Tenure
Monthly Charges ($)
45
40
35
30
25
20
1
2
3
4
5
6
Tenure of customer

c. Set the y-axis label to ‘Monthly Charges of customer’Code:-

```
import matplotlib.pyplot as plt

# Sample data (replace with your dataset)
# Replace these lists with your actual data for MonthlyCharges and tenure
tenure = [1, 2, 3, 4, 5, 6]  # Example data for tenure
monthly_charges = [20, 25, 30, 35, 40, 45]  # Example data for MonthlyCharges

# Create the scatter plot
plt.figure(figsize=(8, 6))
plt.scatter(tenure, monthly_charges, color='red')  # You can change the color if needed

# Label the axes
plt.xlabel('Tenure of customer', fontsize=12)
plt.ylabel('Monthly Charges of customer', fontsize=12)  # Updated y-axis label

# Title of the plot
plt.title('Scatter Plot of Monthly Charges vs Tenure', fontsize=14)

# Show the plot
plt.grid(True, linestyle='--', alpha=0.7)
plt.show()
```

Ans:-

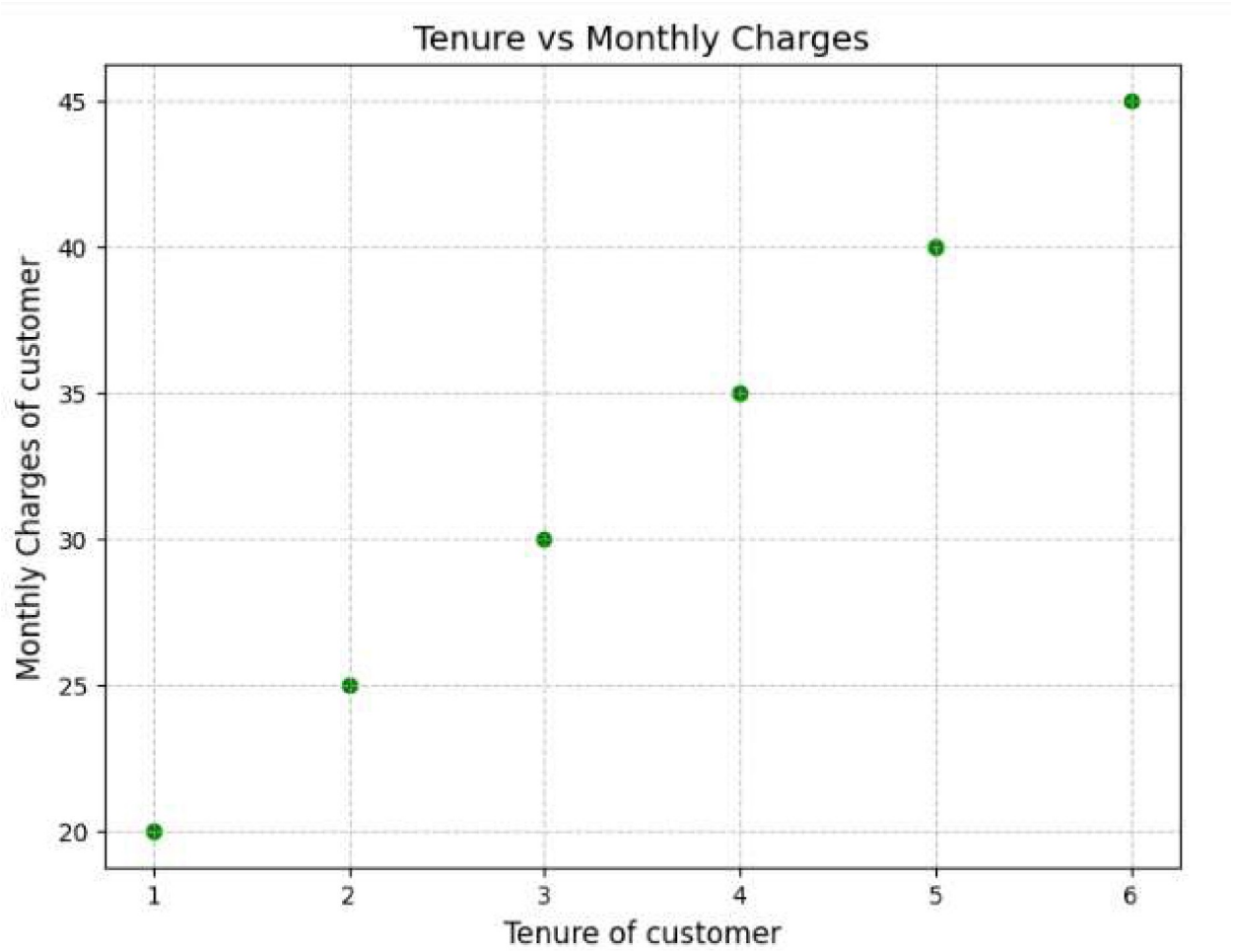

Scatter Plot of Monthly Charges vs Tenure
Monthly Charges of customer
45
40
35
30
25
20
1
2
3
4
5
6

d. Set the title to ‘Tenure vs Monthly Charges’

Code:-

```
import matplotlib.pyplot as plt

# Sample data (replace with your dataset)
# Replace these lists with your actual data for MonthlyCharges and tenure
tenure = [1, 2, 3, 4, 5, 6]  # Example data for tenure
monthly_charges = [20, 25, 30, 35, 40, 45]  # Example data for MonthlyCharges

# Create the scatter plot
plt.figure(figsize=(8, 6))
plt.scatter(tenure, monthly_charges, color='green')  # Updated color to 'green'

# Label the axes
plt.xlabel('Tenure of customer', fontsize=12)
plt.ylabel('Monthly Charges of customer', fontsize=12)

# Title of the plot
plt.title('Tenure vs Monthly Charges', fontsize=14)

# Show the plot
plt.grid(True, linestyle='--', alpha=0.7)
plt.show()
```

Ans:-


Tenure vs Monthly Charges
Monthly Charges of customer
Tenure of customer
45
40
35
30
25
20
1
2
3
4
5
6

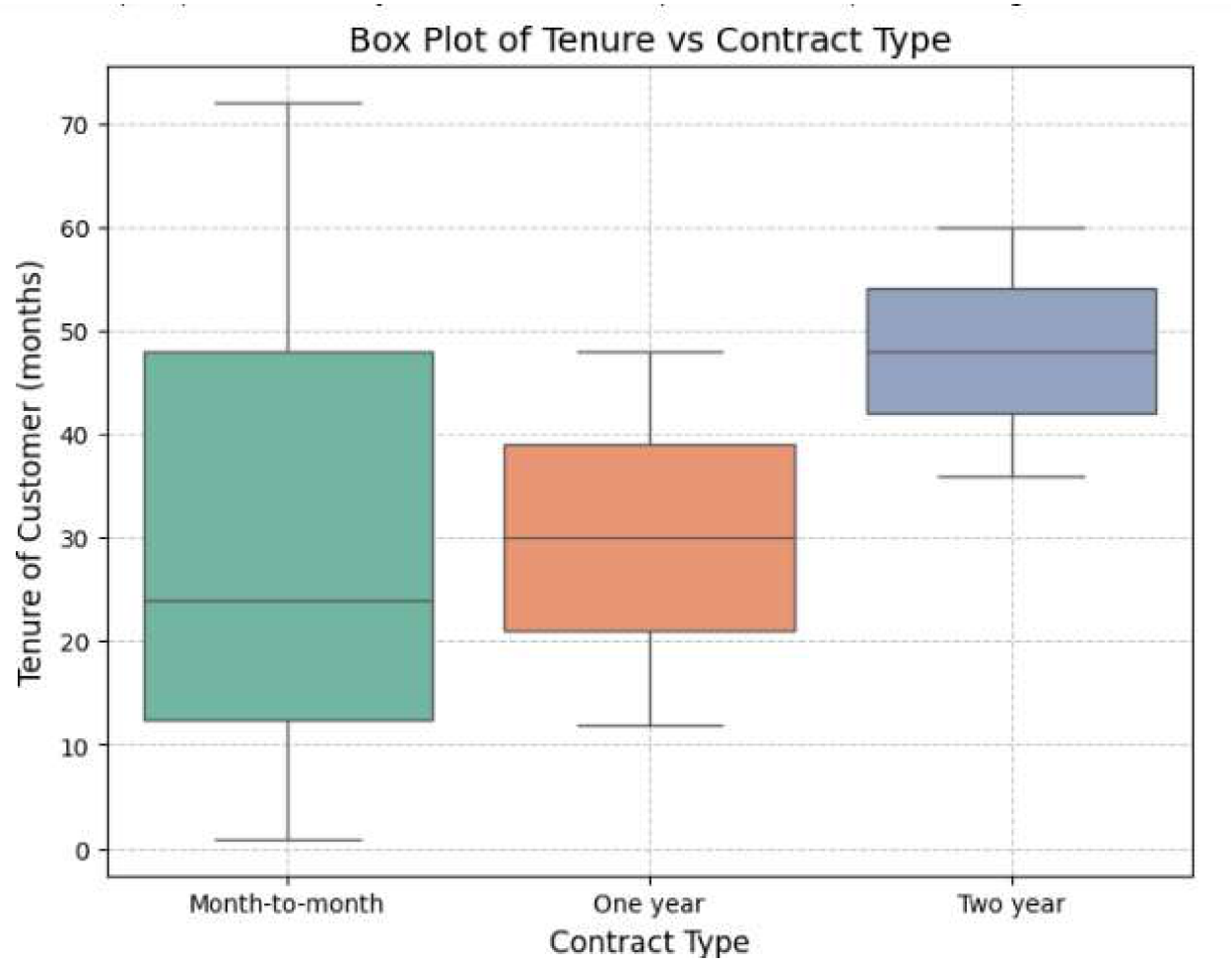
e. Build a box-plot between ‘tenure’ & ‘Contract’. Map ‘tenure’ on the y plot between ‘tenure’ & ‘Contract’. Map ‘tenure’ on the y-axis &

Code:-

```
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd

# Sample data (replace with your dataset)
data = {
    'tenure': [1, 12, 24, 36, 48, 60, 72],  # Example data for tenure
    'Contract': ['Month-to-month', 'One year', 'Month-to-month', 'Two year', 'One year', 'Two year', 'Month-to-month']  # Example data for Contract
}

# Convert to a DataFrame
df = pd.DataFrame(data)

# Create the box plot
plt.figure(figsize=(8, 6))
sns.boxplot(x='Contract', y='tenure', data=df, palette='Set2')  # 'Set2' gives a nice color palette

# Label the axes and title
plt.xlabel('Contract Type', fontsize=12)
plt.ylabel('Tenure of Customer (months)', fontsize=12)
plt.title('Box Plot of Tenure vs Contract Type', fontsize=14)

# Show the plot
plt.grid(True, linestyle='--', alpha=0.7)
plt.show()
```

Ans:-

```
<ipython-input-10-51c446351dc4>:16: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

  sns.boxplot(x='Contract', y='tenure', data=df, palette='Set2')  # 'Set2' gives a nice color palette
```



3. Linear Regression:

● Build a simple linear model where dependent variable is ‘MonthlyCharges’ and independent variable is ‘tenure’:

a. Divide the dataset into train and test sets in 70:30 ratio

Code:-

```
# Import necessary libraries
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

# Load the dataset (replace 'your_dataset.csv' with your actual dataset file)
data = pd.read_csv('/content/customer_churn (1).csv')

# Select the dependent and independent variables
X = data[['tenure']]  # Independent variable
y = data['MonthlyCharges']  # Dependent variable

# Split the dataset into training and testing sets (70:30 ratio)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Initialize and train the linear regression model
model = LinearRegression()
model.fit(X_train, y_train)

# Predict on the test set
y_pred = model.predict(X_test)

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

# Display results
print(f"Mean Squared Error (MSE): {mse}")
print(f"R-squared (R2): {r2}")
```

```
# Display model coefficients
print(f"Intercept: {model.intercept_}")
print(f"Coefficient for 'tenure': {model.coef_[0]}")
```

Ans:-

```
Mean Squared Error (MSE): 845.6091871095867
R-squared (R2): 0.05856035027031625
Intercept: 54.79837462739951
Coefficient for 'tenure': 0.3082154776200297
```

b. Build the model on train set and predict the values on test set

Code:-

```python
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

# Load dataset
data = pd.read_csv('/content/customer_churn (1).csv')
X, y = data[['tenure']], data['MonthlyCharges']

# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Train and predict
model = LinearRegression().fit(X_train, y_train)
y_pred = model.predict(X_test)

# Evaluate
print(f"MSE: {mean_squared_error(y_test, y_pred)}")
print(f"R²: {r2_score(y_test, y_pred)}")
print(f"Intercept: {model.intercept_}, Coefficient: {model.coef_[0]}")
```

Ans:-

```
MSE: 845.6091871095867
R²: 0.05856035027031625
Intercept: 54.79837462739951, Coefficient: 0.3082154776200297
```

c. After predicting the values, find the root mean square error

Code:-

```python
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

X, y = data[['tenure']], data['MonthlyCharges']

# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Train and predict
model = LinearRegression().fit(X_train, y_train)
y_pred = model.predict(X_test)

# Calculate RMSE manually
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
print(f"Root Mean Square Error (RMSE): {rmse}")
```

Asn:-

```
Root Mean Square Error (RMSE): 29.07936015646814
```

d. Find out the error in prediction & store the result in ‘error’ Code:-

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

X, y = data[['tenure']], data['MonthlyCharges']

# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Train and predict
model = LinearRegression().fit(X_train, y_train)
y_pred = model.predict(X_test)

# Calculate error and store in 'error'
error = y_test - y_pred

# Display errors
print("Prediction Errors:")
print(error)
```

Ans:-

```
Prediction Errors:
185    -30.306590
2715   -42.185209
3825   -51.475579
1807    21.243410
132    -24.898812
          ...
5522    15.043410
6377   -10.456590
5500   -12.791303
2392    29.737316
6705   -10.479115
Name: MonthlyCharges, Length: 2113, dtype: float64
```

e. Find the root mean square error

Code:-

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

X, y = data[['tenure']], data['MonthlyCharges']

# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Train and predict
model = LinearRegression().fit(X_train, y_train)
y_pred = model.predict(X_test)

# Calculate RMSE
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
print(f"Root Mean Square Error (RMSE): {rmse}")
```

Ans:-

```
Root Mean Square Error (RMSE): 29.07936015646814
```

4. Logistic Regression:

● Build a simple logisc regression model where dependent variable is ‘Churn’ and independent variable is ‘MonthlyCharges’:

a. Divide the dataset in 65:35 ratio

Code:-

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report


# Select the dependent and independent variables
X = data[['MonthlyCharges']]  # Independent variable
y = data['Churn']  # Dependent variable (ensure this is binary: 0 or 1)

# Split dataset into training and test sets (65:35 ratio)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.35, random_state=42)

# Initialize and train the logistic regression model
model = LogisticRegression()
model.fit(X_train, y_train)

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

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy}")
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
print("Classification Report:")
print(classification_report(y_test, y_pred))
```

Ans:-

```
Accuracy: 0.7287104622871047
Confusion Matrix:
[[1797    0]
 [ 669    0]]
Classification Report:
              precision    recall  f1-score   support

          No       0.73      1.00      0.84      1797
         Yes       0.00      0.00      0.00       669

    accuracy                           0.73      2466
   macro avg       0.36      0.50      0.42      2466
weighted avg       0.53      0.73      0.61      2466

/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted sa
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted sa
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted sa
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

b. Build the model on train set and predict the values on test set code:-

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score



# Select the dependent and independent variables
X = data[['MonthlyCharges']]  # Independent variable
y = data['Churn']  # Dependent variable (binary: 0 or 1)

# Split dataset into training and test sets (65:35 ratio)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.35, random_state=42)

# Initialize and train the logistic regression model on the training set
model = LogisticRegression()
model.fit(X_train, y_train)

# Predict values on the test set
y_pred = model.predict(X_test)

# Display predictions
print("Predicted values on the test set:")
print(y_pred)

# Evaluate the model (optional)
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy}")
```

Ans:-

```
Predicted values on the test set:
['No' 'No' 'No' ... 'No' 'No' 'No']
Accuracy: 0.7287104622871047
```

c. Build the confusion matrix and get the accuracy score Code:-

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix, accuracy_score


# Select the dependent and independent variables
X = data[['MonthlyCharges']]  # Independent variable
y = data['Churn']  # Dependent variable (binary: 0 or 1)

# Split dataset into training and test sets (65:35 ratio)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.35, random_state=42)

# Initialize and train the logistic regression model
model = LogisticRegression()
model.fit(X_train, y_train)

# Predict values on the test set
y_pred = model.predict(X_test)

# Build the confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(conf_matrix)

# Calculate and display the accuracy score
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy Score: {accuracy}")
```

Ans:-

```
Confusion Matrix:
[[1797    0]
 [ 669    0]]
Accuracy Score: 0.7287104622871047
```

d. Build a multiple logistic regression model where dependent variable is ‘Churn’ and independent variables are ‘tenure’ and ‘MonthlyCharges’

Code:-

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix, accuracy_score


# Select the dependent and independent variables
X = data[['tenure', 'MonthlyCharges']]  # Independent variables
y = data['Churn']  # Dependent variable (binary: 0 or 1)

# Split dataset into training and test sets (65:35 ratio)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.35, random_state=42)

# Initialize and train the logistic regression model
model = LogisticRegression()
model.fit(X_train, y_train)

# Predict values on the test set
y_pred = model.predict(X_test)

# Build the confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(conf_matrix)

# Calculate and display the accuracy score
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy Score: {accuracy}")
```

Ans:-

```
Confusion Matrix:
[[1650  147]
 [ 359  310]]
Accuracy Score: 0.7948094079480941
```

e. Divide the dataset in
80:20 ratio
Code:-

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix, accuracy_score


# Define features and target
X = data[['tenure', 'MonthlyCharges']]
y = data['Churn']

# Split data (80:20 ratio)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train logistic regression model
model = LogisticRegression()
model.fit(X_train, y_train)

# Predict and evaluate
y_pred = model.predict(X_test)
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
print(f"Accuracy Score: {accuracy_score(y_test, y_pred)}")
```

```
Confusion Matrix:
[[944  92]
 [193 180]]
Accuracy Score: 0.7977288857345636
```

Ans:-

Confusion Matrix:- [[944 92]

[193 180]]

Accuracy Score:

0.7977288857345636

5. Decision Tree:

● Build a decision tree model where dependent variable is ‘Churn’ and independent variable is ‘tenure’.

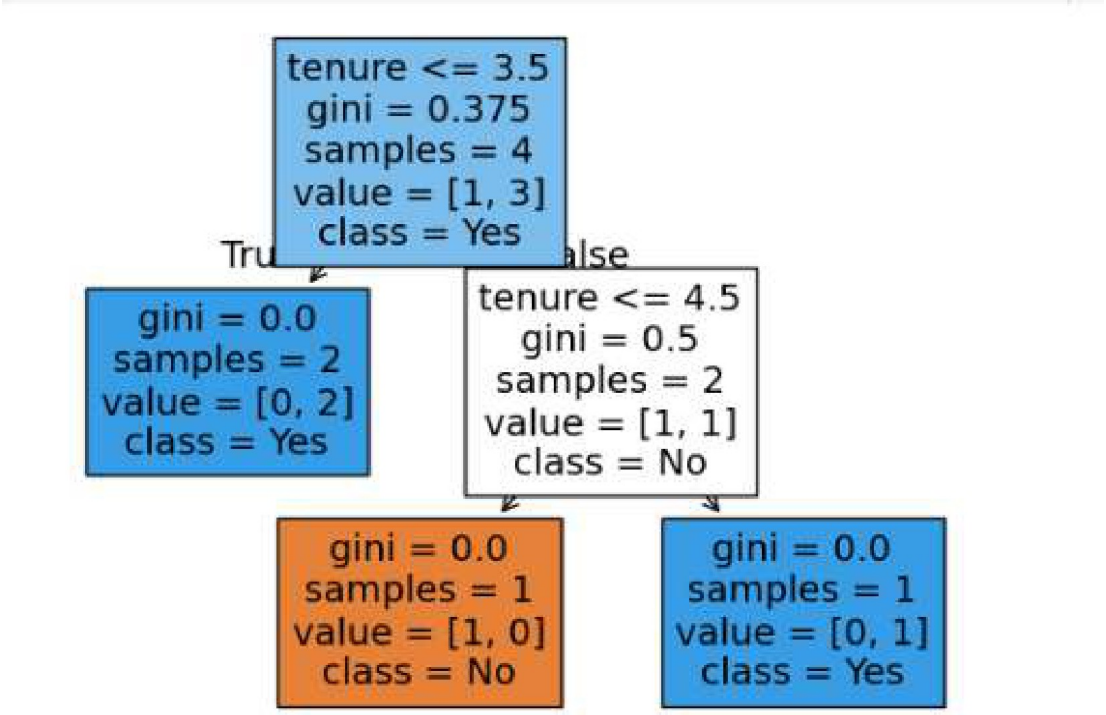
a. Divide the dataset in

80:20 ratio

Code:-

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier, plot_tree
import matplotlib.pyplot as plt

# Load your dataset
df = pd.DataFrame({'tenure': [1, 2, 3, 4, 5], 'Churn': ['Yes', 'No', 'Yes', 'No', 'Yes']})

# Prepare data
X = df[['tenure']]
y = df['Churn'].map({'Yes': 1, 'No': 0})  # Convert 'Churn' to numeric

# Split data (80:20)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train decision tree
clf = DecisionTreeClassifier(random_state=42)
clf.fit(X_train, y_train)

# Visualize tree
plot_tree(clf, feature_names=['tenure'], class_names=['No', 'Yes'], filled=True)
plt.show()

# Print accuracy
print(f"Accuracy: {clf.score(X_test, y_test):.2f}")
```

Ans:-




```
Accuracy: 0.00
```

b. Build the model on train set and predict the values on test set Code:-

```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report

# Example dataset
data = {
    'tenure': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
    'Churn': ['Yes', 'No', 'Yes', 'No', 'Yes', 'Yes', 'No', 'No', 'Yes', 'No']
}
df = pd.DataFrame(data)

# Map 'Churn' to numeric values
X = df[['tenure']]  # Independent variable
y = df['Churn'].map({'Yes': 1, 'No': 0})  # Dependent variable

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

# Train the Decision Tree model
clf = DecisionTreeClassifier(max_depth=3, random_state=42)
clf.fit(X_train, y_train)

# Predict values on the test set
y_pred = clf.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy on Test Set: {accuracy:.2f}")
print("Classification Report:")
print(classification_report(y_test, y_pred, target_names=['No', 'Yes']))
```

Ans:-

```
Accuracy on Test Set: 0.50
Classification Report:
              precision    recall  f1-score   support

          No       0.50      1.00      0.67         1
         Yes       0.00      0.00      0.00         1

    accuracy                           0.50         2
   macro avg       0.25      0.50      0.33         2
weighted avg       0.25      0.50      0.33         2

/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted sa
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted sa
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted sa
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

c. Build the confusion matrix and calculate the accuracy Code:-

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report

# Example dataset
data = {
    'tenure': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
    'Churn': ['Yes', 'No', 'Yes', 'No', 'Yes', 'Yes', 'No', 'No', 'Yes', 'No']
}
df = pd.DataFrame(data)

# Map 'Churn' to numeric values
X = df[['tenure']]  # Independent variable
y = df['Churn'].map({'Yes': 1, 'No': 0})  # Dependent variable

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

# Train the Decision Tree model
clf = DecisionTreeClassifier(max_depth=3, random_state=42)
clf.fit(X_train, y_train)

# Predict values on the test set
y_pred = clf.predict(X_test)

# Build the confusion matrix
cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(cm)

# Calculate accuracy
```

```
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")

# Optionally, print the classification report
print("Classification Report:")
print(classification_report(y_test, y_pred, target_names=['No', 'Yes']))
```

Ans:-

```
Confusion Matrix:
[[1 0]
 [1 0]]
Accuracy: 0.50
Classification Report:
              precision    recall  f1-score   support

          No       0.50      1.00      0.67         1
         Yes       0.00      0.00      0.00         1

    accuracy                           0.50         2
   macro avg       0.25      0.50      0.33         2
weighted avg       0.25      0.50      0.33         2

/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted sa
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted sa
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted sa
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

6. Random Forest:

● Build a Random Forest model where dependent variable is ‘Churn’ and independent variables are ‘tenure’ and ‘MonthlyCharges’:

a. Divide the dataset in 70:30 ratio

Code:-

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

# Sample Data
df = pd.DataFrame({
    'tenure': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
    'MonthlyCharges': [20, 30, 40, 50, 60, 70, 80, 90, 100, 110],
    'Churn': ['Yes', 'No', 'Yes', 'No', 'Yes', 'Yes', 'No', 'No', 'Yes', 'No']
})

# Prepare Data
X = df[['tenure', 'MonthlyCharges']]
y = df['Churn'].map({'Yes': 1, 'No': 0})
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, stratify=y, random_state=42)

# Train and Predict
clf = RandomForestClassifier(n_estimators=100, random_state=42)
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)

# Results
print(f"Accuracy: {accuracy_score(y_test, y_pred):.2f}")
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred, target_names=['No', 'Yes']))
```

Ans:-

```
Accuracy: 0.33
Confusion Matrix:
 [[1 1]
 [1 0]]
Classification Report:
              precision    recall  f1-score   support

          No       0.50      0.50      0.50         2
         Yes       0.00      0.00      0.00         1

    accuracy                           0.33         3
   macro avg       0.25      0.25      0.25         3
weighted avg       0.33      0.33      0.33         3
```

b. Build the model on train set and predict the values on test set Code:-

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

# Sample Data
df = pd.DataFrame({
    'tenure': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
    'MonthlyCharges': [20, 30, 40, 50, 60, 70, 80, 90, 100, 110],
    'Churn': ['Yes', 'No', 'Yes', 'No', 'Yes', 'Yes', 'No', 'No', 'Yes', 'No']
})

# Prepare Data
X = df[['tenure', 'MonthlyCharges']]
y = df['Churn'].map({'Yes': 1, 'No': 0})
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, stratify=y, random_state=42)

# Train Model
clf = RandomForestClassifier(n_estimators=100, random_state=42)
clf.fit(X_train, y_train)

# Predict on Test Set
y_pred = clf.predict(X_test)

# Results
print(f"Predictions on Test Set: {y_pred}")
print(f"Accuracy: {accuracy_score(y_test, y_pred):.2f}")
```

Ans:-

```
Predictions on Test Set: [1 0 0]
Accuracy: 0.33
```

c. Build the confusion matrix and calculate
the accuracy Code:-

```
from sklearn.metrics import confusion_matrix, accuracy_score

# Example true labels and predicted labels
y_true = [1, 0, 1, 1, 0, 1, 0, 0, 1, 0]  # Example true labels
y_pred = [1, 0, 1, 0, 0, 1, 0, 1, 1, 0]  # Example predicted labels

# Generate the confusion matrix
conf_matrix = confusion_matrix(y_true, y_pred)
print("Confusion Matrix:")
print(conf_matrix)

# Calculate accuracy
accuracy = accuracy_score(y_true, y_pred)
print("\nAccuracy:", accuracy)
```

```
Confusion Matrix:
[[4 1]
 [1 4]]

Accuracy: 0.8
```

Ans:-

Confusion Matrix:-

[[4 1]

[1

4]

Accuaracy : 0.8