customer-churn

April 28, 2024

0.1 Importing all the Libraries

[4]:
```
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
import plotly.express as px
import plotly.graph_objects as go
```

0.2 Loading the data

[5]:
```
data = pd.read_csv("customer_churn.csv")
```

[6]:
```
data
```

[6]:
```
      customerID  gender  SeniorCitizen Partner Dependents  tenure  \
0     7590-VHVEG  Female              0     Yes         No       1
1     5575-GNVDE    Male              0      No         No      34
2     3668-QPYBK    Male              0      No         No       2
3     7795-CFOCW    Male              0      No         No      45
4     9237-HQITU  Female              0      No         No       2
…          …     …              …      …        …     …
7038  6840-RESVB    Male              0     Yes        Yes      24
7039  2234-XADUH  Female              0     Yes        Yes      72
7040  4801-JZAZL  Female              0     Yes        Yes      11
7041  8361-LTMKD    Male              1     Yes         No       4
7042  3186-AJIEK    Male              0      No         No      66

     PhoneService     MultipleLines InternetService OnlineSecurity  …  \
0              No  No phone service             DSL             No  …
1             Yes                No             DSL            Yes  …
2             Yes                No             DSL            Yes  …
3              No  No phone service             DSL            Yes  …
4             Yes                No     Fiber optic             No  …
…           …                …              …            …  …
7038          Yes               Yes             DSL            Yes  …
7039          Yes               Yes     Fiber optic             No  …
7040           No  No phone service             DSL            Yes  …
```

```
7041          Yes               Yes     Fiber optic             No  …
7042          Yes                No     Fiber optic            Yes  …

     DeviceProtection TechSupport StreamingTV StreamingMovies        Contract  \
0                  No          No          No              No  Month-to-month
1                 Yes          No          No              No        One year
2                  No          No          No              No  Month-to-month
3                 Yes         Yes          No              No        One year
4                  No          No          No              No  Month-to-month
…                 …           …           …               …              …
7038              Yes         Yes         Yes             Yes        One year
7039              Yes          No         Yes             Yes        One year
7040               No          No          No              No  Month-to-month
7041               No          No          No              No  Month-to-month
7042              Yes         Yes         Yes             Yes        Two year

     PaperlessBilling              PaymentMethod MonthlyCharges  TotalCharges  \
0                 Yes           Electronic check          29.85         29.85
1                  No               Mailed check          56.95        1889.5
2                 Yes               Mailed check          53.85        108.15
3                  No  Bank transfer (automatic)          42.30       1840.75
4                 Yes           Electronic check          70.70        151.65
…                 …                          …             …           …
7038              Yes               Mailed check          84.80        1990.5
7039              Yes    Credit card (automatic)         103.20        7362.9
7040              Yes           Electronic check          29.60        346.45
7041              Yes               Mailed check          74.40         306.6
7042              Yes  Bank transfer (automatic)         105.65        6844.5

     Churn
0       No
1       No
2      Yes
3       No
4      Yes
…      …
7038    No
7039    No
7040    No
7041   Yes
7042    No

[7043 rows x 21 columns]
```

1 Data Manipulation

1.1 Extract the 5th column & store it in 'customer_5'

[10]:
```
c_5= data.iloc[:,4]
# iloc is used to select rows and columns by integer position
# The part "[:,4]" is used to select all rows (: for all rows) and the 4th column (index 4)
# So, c_5 will be a pandas Series containing all the values from the 4th column of the DataFrame
```

[9]:
```
c_5.head(10)
```

[9]:
```
0     No
1     No
2     No
3     No
4     No
5     No
6    Yes
7     No
8     No
9    Yes
Name: Dependents, dtype: object
```

1.2 Extract the 15th column & store it in 'customer_15'

[11]:
```
c_15= data.iloc[:,14]
```

[12]:
```
c_15.head(17)
```

[12]:
```
0                     No
1                     No
2                     No
3                     No
4                     No
5                    Yes
6                     No
7                     No
8                    Yes
9                     No
10                    No
11    No internet service
12                   Yes
13                   Yes
14                   Yes
15                   Yes
```

```
16    No internet service
Name: StreamingMovies, dtype: object
```

1.3 Extract all the male senior citizens whose Payment Method is Electronic check & store the result in 'senior_male_electronic'

[13]:
```
data.columns
```

[13]:
```
Index(['customerID', 'gender', 'SeniorCitizen', 'Partner', 'Dependents',
       'tenure', 'PhoneService', 'MultipleLines', 'InternetService',
       'OnlineSecurity', 'OnlineBackup', 'DeviceProtection', 'TechSupport',
       'StreamingTV', 'StreamingMovies', 'Contract', 'PaperlessBilling',
       'PaymentMethod', 'MonthlyCharges', 'TotalCharges', 'Churn'],
      dtype='object')
```

[14]:
```
data[(data['gender']=='Male') & (data['SeniorCitizen']== 1) &␣
 ↪(data['PaymentMethod']=='Electronic check')]
```

[14]:
```
      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]
```

[17]:
```
senior_male_electronic = data[(data['gender']=='Male') &␣
 ↪(data['SeniorCitizen']== 1) & (data['PaymentMethod']=='Electronic check')]
# The '&' operator is used to combine multiple conditions (logical AND)
# The result is a new DataFrame 'senior_male_electronic' containing only
# the rows where all three conditions are met:
# - Gender is Male
# - SeniorCitizen is True (1)
# - PaymentMethod is 'Electronic check'
```

[18]:
```
senior_male_electronic.head()
```

[18]:
```
    customerID gender  SeniorCitizen Partner Dependents  tenure PhoneService  \
20  8779-QRDMV   Male              1      No         No       1           No
55  1658-BYGOY   Male              1      No         No      18          Yes
57  5067-XJQFU   Male              1     Yes        Yes      66          Yes
78  0191-ZHSKZ   Male              1      No         No      30          Yes
91  2424-WVHPL   Male              1      No         No       1          Yes

       MultipleLines InternetService OnlineSecurity  … DeviceProtection  \
20  No phone service             DSL             No  …              Yes
55               Yes     Fiber optic             No  …               No
57               Yes     Fiber optic             No  …              Yes
```

```
78                No             DSL            Yes  …               No
91                No     Fiber optic             No  …               No

   TechSupport StreamingTV StreamingMovies        Contract PaperlessBilling  \
20          No          No              Yes  Month-to-month              Yes
55          No         Yes              Yes  Month-to-month              Yes
57         Yes         Yes              Yes        One year              Yes
78          No         Yes              Yes  Month-to-month              Yes
91         Yes          No               No  Month-to-month               No

       PaymentMethod MonthlyCharges  TotalCharges Churn
20  Electronic check          39.65         39.65   Yes
55  Electronic check          95.45       1752.55   Yes
57  Electronic check         108.45       7076.35    No
78  Electronic check          74.75        2111.3    No
91  Electronic check          74.70          74.7    No

[5 rows x 21 columns]
```

1.4 Extract all those customers whose tenure is greater than 70 months or their Monthly charges is more than 100$ & store the result in 'customer_total_tenure'

[23]:
```
customer_total_tenure= data[(data['tenure']> 70) | (data['MonthlyCharges']>100␣
↪)]
```

[24]:
```
customer_total_tenure.head()
```

[24]:
```
     customerID  gender  SeniorCitizen Partner Dependents  tenure PhoneService  \
8    7892-POOKP  Female              0     Yes         No      28          Yes
12   8091-TTVAX    Male              0     Yes         No      58          Yes
13   0280-XJGEX    Male              0      No         No      49          Yes
14   5129-JLPIS    Male              0      No         No      25          Yes
15   3655-SNQYZ  Female              0     Yes        Yes      69          Yes

   MultipleLines InternetService OnlineSecurity  … DeviceProtection  \
8            Yes     Fiber optic             No  …              Yes
12           Yes     Fiber optic             No  …              Yes
13           Yes     Fiber optic             No  …              Yes
14            No     Fiber optic            Yes  …              Yes
15           Yes     Fiber optic            Yes  …              Yes

   TechSupport StreamingTV StreamingMovies        Contract PaperlessBilling  \
8          Yes         Yes              Yes  Month-to-month              Yes
12          No         Yes              Yes        One year               No
13          No         Yes              Yes  Month-to-month              Yes
14         Yes         Yes              Yes  Month-to-month              Yes
```

```
15         Yes         Yes              Yes        Two year                No

                PaymentMethod MonthlyCharges  TotalCharges Churn
8             Electronic check         104.80       3046.05   Yes
12    Credit card (automatic)         100.35        5681.1    No
13  Bank transfer (automatic)         103.70        5036.3   Yes
14            Electronic check         105.50       2686.05    No
15    Credit card (automatic)         113.25       7895.15    No

[5 rows x 21 columns]
```

1.5 Extract all the customers whose Contract is of two years, payment method is Mailed check & the value of Churn is 'Yes' & store the result in 'two_mail_yes'

[25]:
```
data.columns
```

[25]:
```
Index(['customerID', 'gender', 'SeniorCitizen', 'Partner', 'Dependents',
       'tenure', 'PhoneService', 'MultipleLines', 'InternetService',
       'OnlineSecurity', 'OnlineBackup', 'DeviceProtection', 'TechSupport',
       'StreamingTV', 'StreamingMovies', 'Contract', 'PaperlessBilling',
       'PaymentMethod', 'MonthlyCharges', 'TotalCharges', 'Churn'],
      dtype='object')
```

[27]:
```
two_mail_yes= data[(data['Contract']=='Two year') & (data['PaymentMethod']==␣
 ↪'Mailed check') & (data['Churn']=='Yes')]
```

[28]:
```
two_mail_yes
```

[28]:
```
      customerID  gender  SeniorCitizen Partner Dependents  tenure  \
268   6323-AYBRX    Male              0      No         No      59
5947  7951-QKZPL  Female              0     Yes        Yes      33
6680  9412-ARGBX  Female              0      No        Yes      48

     PhoneService MultipleLines InternetService       OnlineSecurity  …  \
268           Yes            No              No  No internet service  …
5947          Yes           Yes              No  No internet service  …
6680          Yes            No     Fiber optic                  No  …

         DeviceProtection          TechSupport          StreamingTV  \
268   No internet service  No internet service  No internet service
5947  No internet service  No internet service  No internet service
6680                  Yes                  Yes                  Yes

          StreamingMovies  Contract PaperlessBilling PaymentMethod  \
268   No internet service  Two year               No  Mailed check
5947  No internet service  Two year              Yes  Mailed check
```

```
6680                  No  Two year              Yes  Mailed check

     MonthlyCharges  TotalCharges Churn
268           19.35        1099.6   Yes
5947          24.50         740.3   Yes
6680          95.50       4627.85   Yes

[3 rows x 21 columns]
```

1.6 Extract 333 random records from the customer_churn dataframe & store the result in 'customer_333'

[31]:
```
customer_333 = data.sample(n=333)
# The sample() method is used to select a random sample of rows from the␣
 ↪DataFrame
# n=333 means that we want to select 333 rows randomly from the DataFrame
# The resulting DataFrame 'customer_333' will have 333 rows
```

[32]:
```
customer_333.head()
```

[32]:
```
      customerID  gender  SeniorCitizen Partner Dependents  tenure  \
3087  6979-ZNSFF  Female              0      No         No       8
496   4134-BSXLX    Male              0     Yes         No      28
4061  1629-DQQVB  Female              0      No         No      14
1858  7517-SAWMO  Female              0     Yes         No      19
5801  8695-ARGXZ    Male              1     Yes         No      34

     PhoneService MultipleLines InternetService OnlineSecurity  …  \
3087          Yes           Yes     Fiber optic             No  …
496           Yes            No             DSL            Yes  …
4061          Yes            No             DSL            Yes  …
1858          Yes            No     Fiber optic            Yes  …
5801          Yes           Yes     Fiber optic             No  …

     DeviceProtection TechSupport StreamingTV StreamingMovies        Contract  \
3087               No          No         Yes              No  Month-to-month
496                No         Yes          No              No  Month-to-month
4061               No          No          No              No  Month-to-month
1858               No          No          No              No  Month-to-month
5801               No          No          No              No  Month-to-month

     PaperlessBilling              PaymentMethod MonthlyCharges  TotalCharges  \
3087              Yes           Electronic check          87.05         762.1
496                No               Mailed check          60.90       1785.65
4061               No  Bank transfer (automatic)          50.10         709.5
1858              Yes           Electronic check          73.20        1441.1
5801              Yes           Electronic check          75.55        2425.4
```

```
     Churn
3087   Yes
496     No
4061    No
1858   Yes
5801    No

[5 rows x 21 columns]
```

1.7 Get the count of different levels from the 'Churn' column

[35]:
```
data['Churn'].value_counts()
# The 'value_counts()' method is used to count the number of occurrences of␣
 ↪each unique value in a pandas Series
```

[35]:
```
Churn
No     5174
Yes    1869
Name: count, dtype: int64
```

[34]:
```
data['Contract'].value_counts()
```

[34]:
```
Contract
Month-to-month    3875
Two year          1695
One year          1473
Name: count, dtype: int64
```

2 Data Visualization:

2.1 Build a bar-plot for the 'InternetService' columnnge'

[54]:
```
# Get the unique values from the 'InternetService' column as a list
internet_service_categories = data['InternetService'].value_counts().keys().
 ↪tolist()
# 'value_counts()' counts the occurrences of each unique value in the␣
 ↪'InternetService' Series
# The result is a new Series with the unique values as the index and their␣
 ↪counts as the values
# 'keys()' returns a view of the unique values (the index) of the Series
# 'tolist()' converts this view into a list
```

[55]:
```
internet_service_categories
```

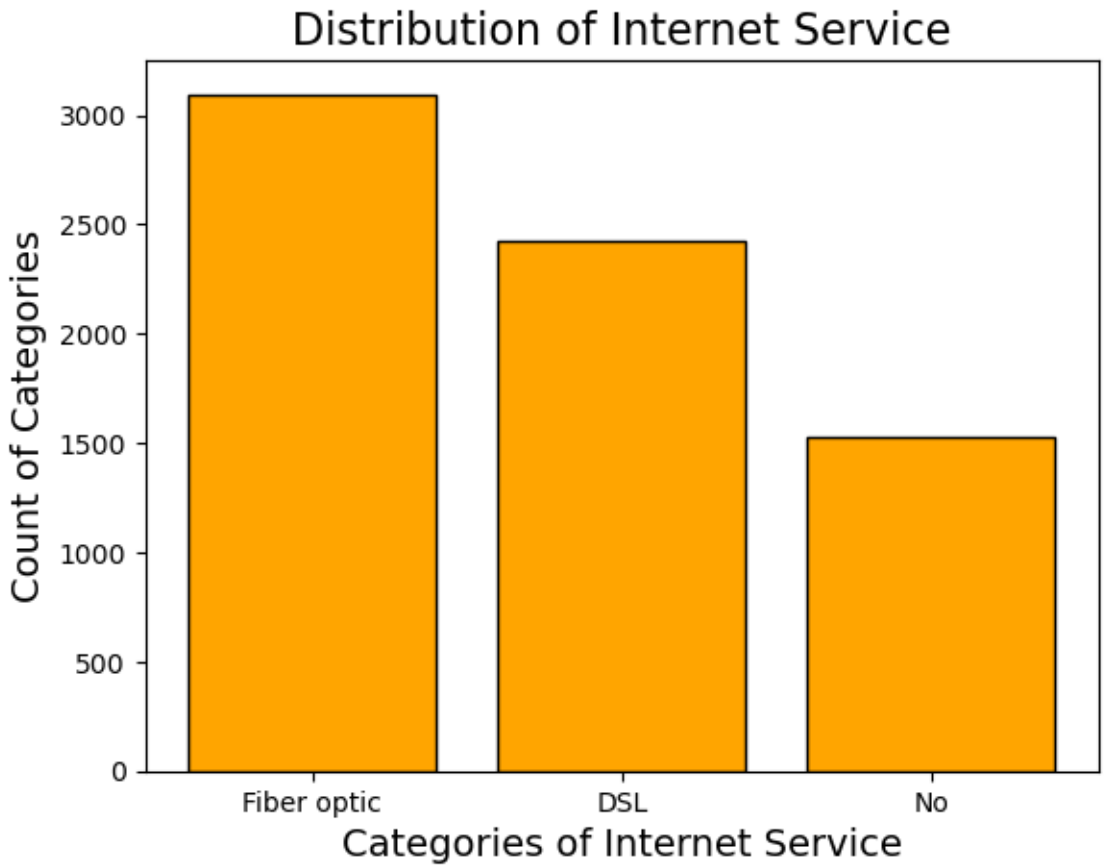
[55]:
```
['Fiber optic', 'DSL', 'No']
```

[56]:
```
# Get the counts of each unique value from the 'InternetService' column as a␣
↪list
internet_service_counts = data['InternetService'].value_counts().tolist()
```

[57]:
```
internet_service_counts
```

[57]:
```
[3096, 2421, 1526]
```

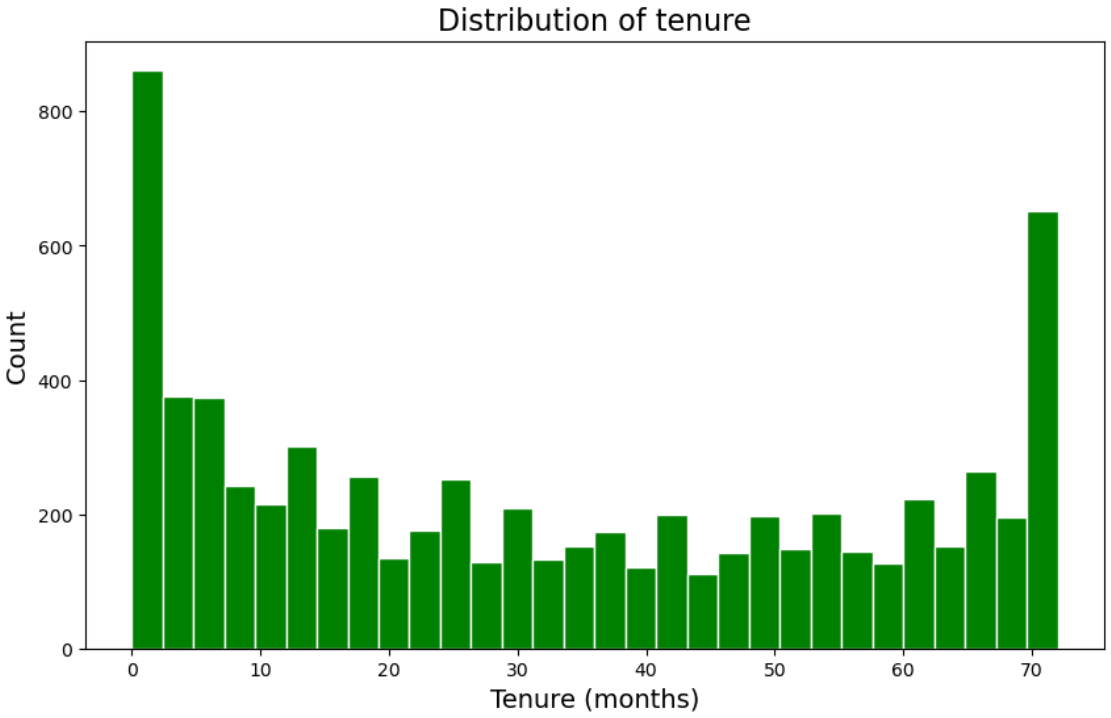
[65]:
```
#plt.bar(data['InternetService'].value_counts().keys().
↪tolist(),data['InternetService'].value_counts().tolist(), color='orange' )
#or
plt.bar(internet_service_categories, internet_service_counts, color='orange',␣
↪edgecolor='black')
# Customize the plot
plt.xlabel('Categories of Internet Service', fontsize=14)  # Set x-axis label
plt.ylabel('Count of Categories', fontsize=14)  # Set y-axis label
plt.title('Distribution of Internet Service', fontsize=16)  # Set plot title

plt.show() #displays the plot with the customizations we made
```



2.2 b. Build a histogram for the 'tenure' column:

i. Set the number of bins to be 30
ii. Set the color of the bins to be 'green'
iii. Assign the title 'Distribution of tenure'

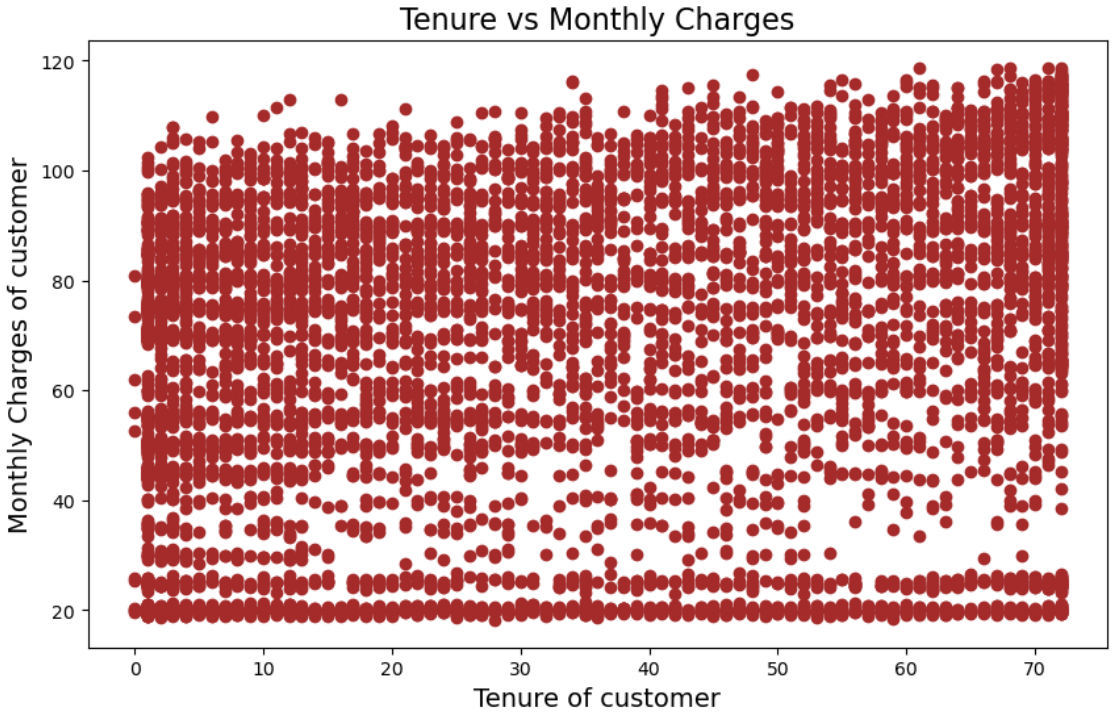
[66]:
```
# Create a histogram for the 'tenure' column
plt.figure(figsize=(10, 6))  # Set the figure size
# Set the number of bins to 30
plt.hist(data['tenure'], bins=30, color='green', edgecolor='white')
# Assign the title 'Distribution of tenure'
plt.title('Distribution of tenure', fontsize=16)
# Label the x-axis and y-axis
plt.xlabel('Tenure (months)', fontsize=14)
plt.ylabel('Count', fontsize=14)
# Display the histogram
plt.show()
```



2.3 c. Build a scatter-plot between 'MonthlyCharges' & 'tenure'. Map 'MonthlyCharges' to the y-axis & 'tenure' to the 'x-axis':

i. Assign the points a color of 'brown'
ii. Set the x-axis label to 'Tenure of customer'
iii. Set the y-axis label to 'Monthly Charges of customer'
iv. Set the title to 'Tenure vs Monthly Charges'

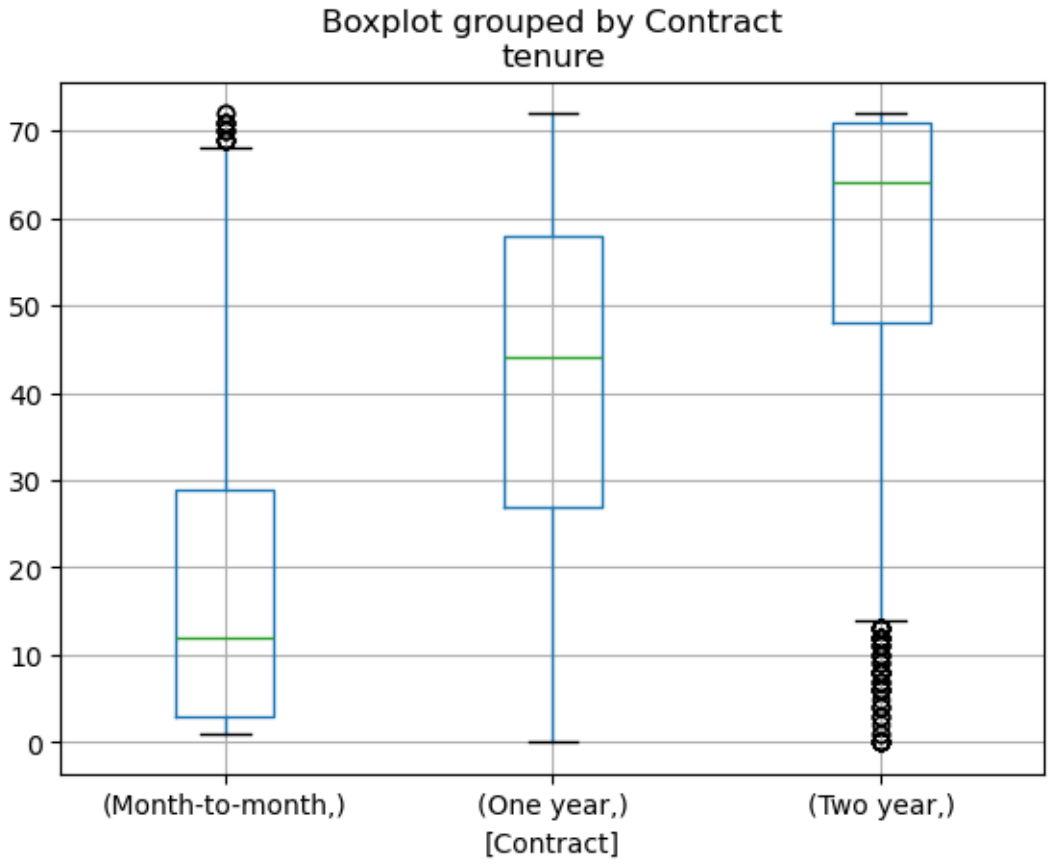
[69]:
```
# Create a scatter plot
plt.figure(figsize=(10, 6))  # Set the figure size
# This line creates a new figure with a specified size (10 inches wide and 6␣
 ↪inches tall).

# Plot the scatter plot with 'tenure' on the x-axis and 'MonthlyCharges' on the␣
 ↪y-axis
plt.scatter(data['tenure'], data['MonthlyCharges'], color='brown')
#data['tenure'] selects the 'tenure' column from the data DataFrame and maps it␣
 ↪to the x-axis.
#data['MonthlyCharges'] selects the 'MonthlyCharges' column from the data␣
 ↪DataFrame and maps it to the y-axis.
#color='brown' sets the color of the scatter points to brown.
#plt.scatter() creates a scatter plot using the specified parameters

# Set the x-axis label
plt.xlabel('Tenure of customer', fontsize=14)
# Set the y-axis label
plt.ylabel('Monthly Charges of customer', fontsize=14)
# Set the title
plt.title('Tenure vs Monthly Charges', fontsize=16)
# Display the scatter plot
plt.show()
```



2.4 Build a box-plot between 'tenure' & 'Contract'. Map 'tenure' on the y-axis & 'Contract' on the x-axis.

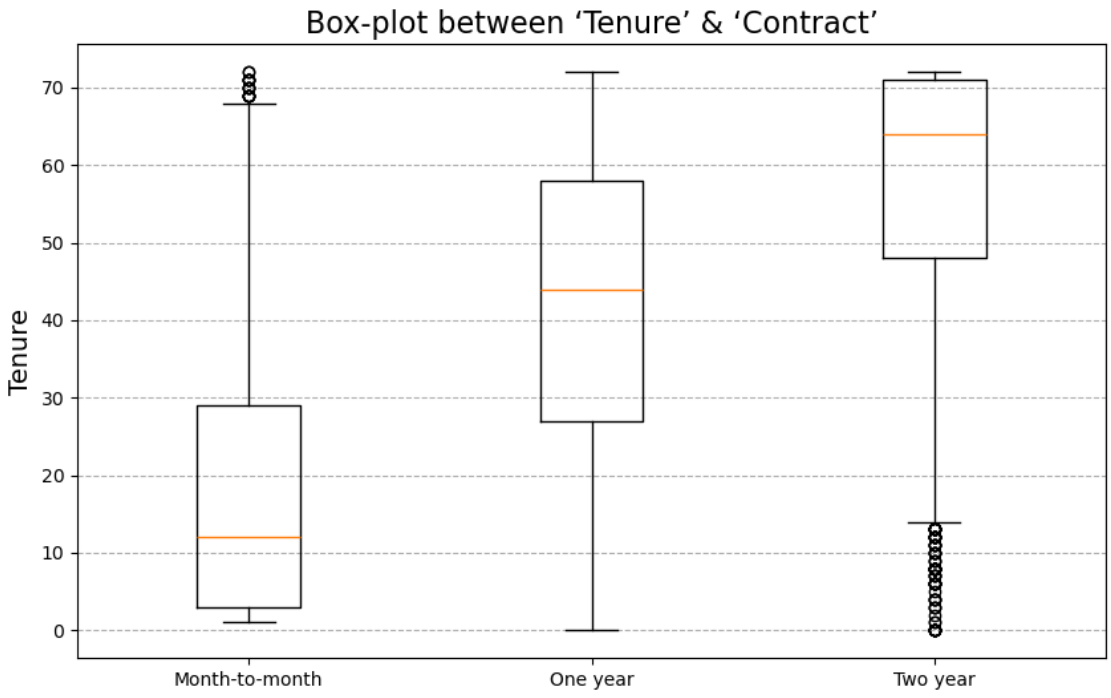
[77]:
```
data.boxplot(column = ['tenure'], by = ['Contract'])

# column=['tenure'],  # Specify the column(s) to plot
#  by=['Contract']  # Specify the column to group the data by
```

[77]:
```
<Axes: title={'center': 'tenure'}, xlabel='[Contract]'>
```



[85]:
```
# Create a box plot
plt.figure(figsize=(10, 6))  # Set the figure size

# Plot the box plot with 'tenure' on the y-axis and 'Contract' on the x-axis
#The lists containing these 'tenure' values are passed to plt.boxplot() to␣
 ↪create a box plot for each contract type.
plt.boxplot([data[data['Contract'] == 'Month-to-month']['tenure'], # selects␣
 ↪the 'tenure' values for customers with a 'Month-to-month' contract.
             data[data['Contract'] == 'One year']['tenure'], #  selects the␣
 ↪'tenure' values for customers with a 'One year' contract.
```

```
            data[data['Contract'] == 'Two year']['tenure']], # selects the␣
 ↪'tenure' values for customers with a 'Two year' contract.
           labels=['Month-to-month', 'One year', 'Two year']) #sets the labels␣
 ↪for the x-axis, corresponding to the contract types.

# Set the y-axis label
plt.ylabel('Tenure', fontsize=14)
# set the title
plt.title('Box-plot between ‘Tenure’ & ‘Contract’', fontsize=16)

# Rotate the x-axis labels for better visibility
#plt.xticks(rotation=45)

# Add grid lines
      #plt.grid(True, linestyle='--')  # Dashed grid lines on both direction
# True as the first argument enables the grid lines.
# linestyle='--' sets the line style of the grid lines to dashed.
plt.grid(True, axis='y', linestyle='--')  # Dashed grid lines in the y-direction

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



3 Linear Regression:

3.1 Build a simple linear model where dependent variable is 'MonthlyCharges' and independent variable is 'tenure'

i. Divide the dataset into train and test sets in 70:30 ratio.
ii. Build the model on train set and predict the values on test set
iii. After predicting the values, find the root mean square error
iv. Find out the error in prediction & store the result in 'error'
v. Find the root mean square error

[7]:
```
from sklearn import linear_model
# sklearn is a popular machine learning library in Python.
#linear_model is a module within sklearn that contains various linear models,␣
 ↪including linear regression, logistic regression, and others.
from sklearn.linear_model import LinearRegression
# This line imports the LinearRegression class from the linear_model module␣
 ↪within sklearn.
# The LinearRegression class is used to create and train a linear regression␣
 ↪model, which is a fundamental machine learning algorithm for predicting a␣
 ↪continuous target variable based on one or more input features.
from sklearn.model_selection import train_test_split
# model_selection is another module within sklearn that provides utilities for␣
 ↪splitting data into training and testing sets, as well as other functions␣
 ↪related to model evaluation and selection.
# train_test_split is a function within the model_selection module that splits␣
 ↪the input data into training and testing sets.
# By importing train_test_split directly, you can use this function without␣
 ↪having to reference the full module path (model_selection.train_test_split).
```

[7]:
```
data.columns
```

[7]:
```
Index(['customerID', 'gender', 'SeniorCitizen', 'Partner', 'Dependents',
       'tenure', 'PhoneService', 'MultipleLines', 'InternetService',
       'OnlineSecurity', 'OnlineBackup', 'DeviceProtection', 'TechSupport',
       'StreamingTV', 'StreamingMovies', 'Contract', 'PaperlessBilling',
       'PaymentMethod', 'MonthlyCharges', 'TotalCharges', 'Churn'],
      dtype='object')
```

[9]:
```
y = data[['MonthlyCharges']]
# data[['MonthlyCharges']] selects the 'MonthlyCharges' column from the data␣
 ↪DataFrame and creates a new DataFrame containing only that column.
x = data[['tenure']]
# data[['tenure']] selects the 'tenure' column from the data DataFrame and␣
 ↪creates a new DataFrame containing only that column.
```

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

[14]:
```
# train_test_split is a function from the sklearn.model_selection module that␣
 ↪splits the input data into training and testing sets.
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.3,␣
 ↪random_state=0)
# x and y are the input features (independent variables) and target variable␣
 ↪(dependent variable), respectively. These are typically pandas DataFrames or␣
 ↪NumPy arrays.
# test_size=0.3 specifies that 30% of the data should be used for the testing␣
 ↪set, and the remaining 70% for the training set.
# random_state=42 is a seed value for the random number generator used to␣
 ↪shuffle the data before splitting. Setting a fixed value ensures␣
 ↪reproducibility of the split across different runs.
# X_train: Features (independent variables) for the training set
# X_test: Features (independent variables) for the testing set
# y_train: Target variable (dependent variable) for the training set
# y_test: Target variable (dependent variable) for the testing set
```

[15]:
```
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

[15]:
```
((4930, 1), (2113, 1), (4930, 1), (2113, 1))
```

3.3 Build the model on train set and predict the values on test set

[19]:
```
# Build the model on the train set
model = LinearRegression()
model.fit(X_train, y_train)

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

# model.fit(X_train, y_train) trains the linear regression model on the␣
 ↪training data.
# X_train is the input features (independent variables) for the training set.
# y_train is the target variable (dependent variable) for the training set.
```

3.4 After predicting the values, find the root mean square error

[24]:
```
from sklearn.metrics import mean_squared_error
# mean_squared_error function from the sklearn.metrics module, which is used to␣
 ↪calculate the mean squared error between the predicted and actual values.

rmse = np.sqrt(mean_squared_error(y_test, y_pred))
#mean_squared_error(y_test, y_pred) calculates the mean squared error between␣
 ↪the actual values (y_test) and the predicted values (y_pred).
```

```
#y_test is the target variable (dependent variable) for the testing set.␣
 ↪#y_pred is the predicted values obtained from the model.predict(X_test) step.
#np.sqrt() calculates the square root of the mean squared error, giving you the␣
 ↪root mean square error (RMSE).#The RMSE value is stored in the variable rmse.

print(f"Root Mean Square Error: {rmse:.2f}")
# f"Root Mean Square Error: {rmse:.2f}" is an f-string (formatted string␣
 ↪literal) in Python, which allows you to embed expressions within the string␣
 ↪using curly braces {}.
# {rmse:.2f} formats the rmse value as a floating-point number with two decimal␣
 ↪places.
```

```
Root Mean Square Error: 29.39
```

3.5 Find out the error in prediction & store the result in 'error'

[28]:
```
# Find the error in prediction & store the result in 'error'
error = y_test - y_pred

# y_test is the target variable (dependent variable) for the testing set,␣
 ↪containing the actual values.
# y_pred is the predicted values obtained from the model.predict(X_test) step.
# y_test - y_pred calculates the difference between the actual values (y_test)␣
 ↪and the predicted values (y_pred) for each instance in the testing set.
```

[29]:
```
error.head()
```

[29]:

	MonthlyCharges
2200	-2.750896
4627	43.619033
3225	12.759602
2828	-35.219402
3768	6.236115

4 Logistic Regression:

4.1 Build a simple logistic regression model where dependent variable is 'Churn' & independent variable is 'MonthlyCharges'

i. Divide the dataset in 65:35 ratio
ii. Build the model on train set and predict the values on test set
iii. Build the confusion matrix and get the accuracy score

[42]:
```
X = data[['MonthlyCharges']]  # Independent variable
y = data['Churn']  # Dependent variable
```

4.2 Divide the dataset in 65:35 ratio

[43]:
```
#x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.35,␣
 ↪random_state=0)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.35,␣
 ↪random_state=42)
```

[44]:
```
x_train.shape, x_test.shape, y_train.shape, y_test.shape
```

[44]:
```
((4577, 1), (2466, 1), (4577,), (2466,))
```

4.3 Build the model on train set and predict the values on test set

[47]:
```
from sklearn.linear_model import LogisticRegression

# Build the model on the train set
model = LogisticRegression()
model.fit(X_train, y_train)

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

4.4 Build the confusion matrix and get the accuracy score

[49]:
```
from sklearn.metrics import confusion_matrix, accuracy_score
```

[55]:
```
cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(cm)

accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy Score: {accuracy:.4f}")
```

```
Confusion Matrix:
[[1797    0]
 [ 669    0]]
Accuracy Score: 0.7287
```

5 Build a multiple logistic regression model where dependent variable is 'Churn' & independent variables are 'tenure' & 'MonthlyCharges'

i. Divide the dataset in 80:20 ratio
ii. Build the model on train set and predict the values on test set
iii. Build the confusion matrix and get the accuracy score

[8]:
```
X = data[['tenure', 'MonthlyCharges']]  # Independent variables
y = data['Churn']  # Dependent variable
```

5.1 Divide the dataset in 80:20 ratio

[9]:
```
# Divide the dataset into train and test sets in an 80:20 ratio
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=0)

# We divide the dataset into train and test sets using train_test_split from␣
 ↪sklearn.model_selection.
#The test_size=0.2 parameter specifies that 20% of the data should be used for␣
 ↪testing, and the remaining 80% for training.
```

[11]:
```
X_train.shape,X_test.shape, y_train.shape, y_test.shape
```

[11]:
```
((5634, 2), (1409, 2), (5634,), (1409,))
```

5.2 Build the model on train set and predict the values on test set

[15]:
```
from sklearn.linear_model import LogisticRegression

# Build the model on the train set
model = LogisticRegression()
model.fit(X_train, y_train)

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

# We create an instance of the LogisticRegression model from sklearn.
 ↪linear_model.
# We fit the model to the training data using model.fit(X_train, y_train).
# We predict the 'Churn' values for the test set using model.predict(X_test)␣
 ↪and store the predictions in y_pred.
```

5.3 Build the confusion matrix and get the accuracy score

[19]:
```
from sklearn.metrics import confusion_matrix, accuracy_score

# Build the confusion matrix and get the accuracy score
cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(cm)
# We build the confusion matrix using confusion_matrix(y_test, y_pred) and␣
 ↪print it.
```

```
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy Score: {accuracy:.2f}")
#We calculate the accuracy score using accuracy_score(y_test, y_pred) and print␣
 ↪it.
```

```
Confusion Matrix:
[[1650  147]
 [ 359  310]]
Accuracy Score: 0.79
```

6 Decision Tree:

6.1 Build a decision tree model where dependent variable is 'Churn' & independent variable is 'tenure'

i. Divide the dataset in 80:20 ratio
ii. Build the model on train set and predict the values on test set iii.Build the confusion matrix and calculate the accuracy

[20]:
```
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
```

[21]:
```
# Assuming 'data' is your pandas DataFrame
X = data[['tenure']]  # Independent variable
y = data[['Churn']]  # Dependent variable
```

6.2 Divide the dataset in 80:20 ratio

[22]:
```
# Divide the dataset into train and test sets in an 80:20 ratio
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=0)

# We divide the dataset into train and test sets using train_test_split from␣
 ↪sklearn.model_selection.
# The test_size=0.2 parameter specifies that 20% of the data should be used for␣
 ↪testing, and the remaining 80% for training.
```

[23]:
```
X_train.shape,X_test.shape, y_train.shape, y_test.shape
```

[23]:
```
((5634, 1), (1409, 1), (5634, 1), (1409, 1))
```

6.3 Build the model on train set and predict the values on test set

[33]:
```
# Build the model on the train set
model = DecisionTreeClassifier()  #We create an instance of the␣
 ↪DecisionTreeClassifier model from sklearn.tree.
model.fit(X_train, y_train)       # We fit the model to the training data using␣
 ↪model.fit(X_train, y_train).

# Predict the values on the test set
y_pred = model.predict(X_test)
# We predict the 'Churn' values for the test set using model.predict(X_test)␣
 ↪and store the predictions in y_pred.
```

6.4 Build the confusion matrix and calculate the accuracy

[30]:
```
# The confusion matrix is a table that summarizes the performance of a␣
 ↪classification model by
# showing the counts of true positives, true negatives, false positives, and␣
 ↪false negatives.
# It helps you understand how well the model is classifying the instances.
# Build the confusion matrix and calculate the accuracy
cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(cm)
# We build the confusion matrix using confusion_matrix(y_test, y_pred) and␣
 ↪print it.

# The accuracy score is a metric that measures the proportion of correctly␣
 ↪classified instances over the total number of instances.
# It gives you an overall idea of how well the model is performing, but it may␣
 ↪not be the best metric for imbalanced datasets.
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy Score: {accuracy:.2f}")
# We calculate the accuracy score using accuracy_score(y_test, y_pred) and␣
 ↪print it.
```

```
Confusion Matrix:
[[965  76]
 [281  87]]
Accuracy Score: 0.75
```

7 Random Forest:

7.1 Build a Random Forest model where dependent variable is 'Churn' & independent variables are 'tenure' and 'MonthlyCharges'

i. Divide the dataset in 70:30 ratio

ii. Build the model on train set and predict the values on test set
iii. Build the confusion matrix and calculate the accuracy

[34]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix, accuracy_score
```

[35]:
```
# Assuming 'data' is your pandas DataFrame
X = data[['tenure', 'MonthlyCharges']]  # Independent variables
y = data['Churn']  # Dependent variable
```

7.2 Divide the dataset in 70:30 ratio

[36]:
```
# Divide the dataset into train and test sets in a 70:30 ratio
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
 ↪random_state=0)
# We divide the dataset into train and test sets using train_test_split from␣
 ↪sklearn.model_selection.
# The test_size=0.3 parameter specifies that 30% of the data should be used for␣
 ↪testing, and the remaining 70% for training.
```

[37]:
```
X_train.shape,X_test.shape, y_train.shape, y_test.shape
```

[37]:
```
((4930, 2), (2113, 2), (4930,), (2113,))
```

7.3 Build the model on train set and predict the values on test set

[40]:
```
# Build the model on the train set
model = RandomForestClassifier()          # We create an instance of the␣
 ↪RandomForestClassifier model from sklearn.ensemble.
model.fit(X_train, y_train)               # We fit the model to the training data␣
 ↪using model.fit(X_train, y_train).

# Predict the values on the test set
y_pred = model.predict(X_test)
# We predict the 'Churn' values for the test set using model.predict(X_test)␣
 ↪and store the predictions in y_pred.
```

7.4 Build the confusion matrix and calculate the accuracy

[39]:
```
# Build the confusion matrix and calculate the accuracy
cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(cm)
```

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

```
Confusion Matrix:
[[1345  215]
 [ 321  232]]
Accuracy Score: 0.75
```

8 END