```
import numpy as np
import pandas as pd
```

1.Descriptive statistics on the data

```
df=pd.read_csv("/content/Churn_Modelling.csv")
df.head()
```

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Exited
0	1	15634602	Hargrave	619	France	Female	42	2	0.00	1	1	1	101348.88	1
1	2	15647311	Hill	608	Spain	Female	41	1	83807.86	1	0	1	112542.58	0
2	3	15619304	Onio	502	France	Female	42	8	159660.80	3	1	0	113931.57	1
3	4	15701354	Boni	699	France	Female	39	1	0.00	2	0	0	93826.63	0
4	5	15737888	Mitchell	850	Spain	Female	43	2	125510.82	1	1	1	79084.10	0

```
print(f"Dataset Dimension: **{df.shape[0]}** rows,  **{df.shape[1]}** columns")
```

```
Dataset Dimension: **10000** rows,  **14** columns
```

```
df.info()
print("<br>**SeniorCitizen** is already in integer form<br><br>**TotalCharges** should be converted to float")
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 14 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   RowNumber        10000 non-null  int64
 1   CustomerId       10000 non-null  int64
 2   Surname          10000 non-null  object
 3   CreditScore      10000 non-null  int64
 4   Geography        10000 non-null  object
 5   Gender           10000 non-null  object
 6   Age              10000 non-null  int64
 7   Tenure           10000 non-null  int64
 8   Balance          10000 non-null  float64
 9   NumOfProducts    10000 non-null  int64
 10  HasCrCard        10000 non-null  int64
 11  IsActiveMember   10000 non-null  int64
 12  EstimatedSalary  10000 non-null  float64
 13  Exited           10000 non-null  int64
dtypes: float64(2), int64(9), object(3)
memory usage: 1.1+ MB
<br>**SeniorCitizen** is already in integer form<br><br>**TotalCharges** should be converted to float
```

```
print('Known observations: {}\nUnique observations: {}'.format(len(df.index),len(df.drop_duplicates().index)))
```

print("**No duplicates Found!**")

Known observations: 10000

Unique observations: 10000

No duplicates Found!

df.describe(include=['object']).T

	count	unique	top	freq
Surname	10000	2932	Smith	32
Geography	10000	3	France	5014
Gender	10000	2	Male	5457

2.Missing values

df.isna().sum()

```
RowNumber          0
CustomerId         0
Surname            0
CreditScore        0
Geography          0
Gender             0
Age                0
Tenure             0
Balance            0
NumOfProducts      0
HasCrCard          0
IsActiveMember     0
EstimatedSalary    0
Exited             0
dtype: int64
```

3.Univariate analysis

monthly charges

```
stat, p = stats.normaltest(df_churn['MonthlyCharges'])
print('Statistics=%.5f, p=%.3f' % (stat, p))
# interpret
alpha = 0.05
if p > alpha:
    print('Sample looks Gaussian (fail to reject H0)')
else:
    print('Sample does not look Gaussian (reject H0)')
```

Total charges

```
result = stats.anderson(df_churn['TotalCharges'])
print('Statistic: %.3f' % result.statistic)
p = 0
for i in range(len(result.critical_values)):
    sl, cv = result.significance_level[i], result.critical_values[i]
    if result.statistic < result.critical_values[i]:
        print(f'Significance level {sl:.2f} % : critical value {cv:.3f}, data looks normal (fail to reject H0)')
    else:
        print(f'Significance level {sl:.2f} % : critical value {cv:.3f}, data does not look normal (reject H0)')
```

4.Bivariate analysis

```
def cal_spearmanr(c1, c2):
alpha = 0.05
    correlation, p_value = stats.spearmanr(df_churn[c1], df_churn[c2])
    print(f'{c1}, {c2} correlation : {correlation}, p : {p_value}')
    if p_value > alpha:
        print('Probably do not have monotonic relationship (fail to reject H0)')
    else:
        print('Probably have monotonic relationship (reject H0)')
def kendall_rank_correlation(feature1, feature2):
coef, p_value = stats.kendalltau(df_churn[feature1], df_churn[feature2])
    print(f"Correlation between {feature1} and {feature2} ")
    print('Kendall correlation coefficient = %.5f, p = %.5f' % (coef, p_value))
    # interpret the significance
    alpha = 0.05
    if p_value > alpha:
        print('Samples are uncorrelated (fail to reject H0) p=%.3f' % p_value)
    else:
        print('Samples are correlated (reject H0) p=%.3f' % p_value)
    print('----\n')
ordinal_features = ['tenure-binned','MonthlyCharges-binned', 'TotalCharges-binned']
for ord in ordinal_features:
    printmd(f"Correlation with **{ord}**")
    kendall_rank_correlation('tenure',ord)
    kendall_rank_correlation('MonthlyCharges',ord)
    kendall_rank_correlation('TotalCharges',ord)
def mannwhitneyu_correlation(feature1):
    stat, p_value = stats.mannwhitneyu(df_churn[feature1], (df_churn['Churn'] == 'Yes').astype(int))
    print(f"Correlation between {feature1} and Churn")
    print('Statistics = %.5f, p = %.5f' % (stat, p_value))
    # interpret the significance
    alpha = 0.05
    if p_value > alpha:
        print('Same distribution (fail to reject H0)')
    else:
        print('Different distribution (reject H0)')
    print('----\n')
def correlation_ratio(categories, measurements):
    fcat, _ = pd.factorize(categories)
    cat_num = np.max(fcat)+1
    y_avg_array = np.zeros(cat_num)
    n_array = np.zeros(cat_num)
    for i in range(0,cat_num):
        cat_measures = measurements[np.argwhere(fcat == i).flatten()]
        n_array[i] = len(cat_measures)
        y_avg_array[i] = np.average(cat_measures)
```

```
    y_total_avg = np.sum(np.multiply(y_avg_array,n_array))/np.sum(n_array)
    numerator = np.sum(np.multiply(n_array,np.power(np.subtract(y_avg_array,y_total_avg),2)))
    denominator = np.sum(np.power(np.subtract(measurements,y_total_avg),2))
    if numerator == 0:
        eta = 0.0
    else:
        eta = np.sqrt(numerator/denominator)
    return eta
def cramers_v(x, y):
    """ calculate Cramers V statistic for categorial-categorial association.
      uses correction from Bergsma and Wicher,
      Journal of the Korean Statistical Society 42 (2013): 323-328  """
    confusion_matrix = pd.crosstab(x,y)
    chi2 = stats.chi2_contingency(confusion_matrix)[0]
    n = confusion_matrix.sum().sum()
    phi2 = chi2/n
    r,k = confusion_matrix.shape
    phi2corr = max(0, phi2-((k-1)*(r-1))/(n-1))
    rcorr = r-((r-1)**2)/(n-1)
    kcorr = k-((k-1)**2)/(n-1)
    return np.sqrt(phi2corr/min((kcorr-1),(rcorr-1)))
printmd("**Correlation Between Polytomous Features with Target : Churn**")
cramer_v_val_dict = {}
for col in polytomous_cols:
    cramer_v_val_dict[col] = cramers_v(df_churn[col], df_churn['Churn'])
cramer_v_val_dict_sorted = sorted(cramer_v_val_dict.items(), key=lambda x:x[1], reverse=True)
for k,v in cramer_v_val_dict_sorted:
    print(k.ljust(left_padding), v)
printmd("<br>**Contract, OnlineSecurity, TechSupport, InternetService are moderately correlated with
Churn**<br>")
printmd("**Cramers V Heatmap on Polytomous Features and Target: Churn**")
cramers_v_val = pd.DataFrame(index=['Churn'], columns=polytomous_cols)
for j in range(0,len(polytomous_cols)):
    u = cramers_v(df_churn['Churn'], df_churn[polytomous_cols[j]])
    cramers_v_val.loc[:,polytomous_cols[j]] = ucramers_v_val.fillna(value=np.nan,inplace=True)
plt.figure(figsize=(20,1))
sns.heatmap(cramers_v_val,annot=True,fmt='.3f', cmap="YlGnBu")
plt.show()
theilu = pd.DataFrame(index=['Churn'], columns=cat_cols)
for j in range(0,len(cat_cols)):
    u = theil_u(df_churn['Churn'].tolist(),df_churn[cat_cols[j]].tolist())
    theilu.loc[:,cat_cols[j]] = u
theilu.fillna(value=np.nan,inplace=True)
plt.figure(figsize=(20,1))
sns.heatmap(theilu,annot=True,fmt='.2f')
plt.show()
```

printmd("**Contract, OnlineSecurity, TechSupport, tenure-binned are moderately correlated with Churn**")

5.Multivariate analysis

```
# compare samples
stat, p = stats.kruskal(df_churn['TotalCharges'], df_churn['tenure'], df_churn['MonthlyCharges'])
print('Statistics=%.3f, p=%.3f' % (stat, p))# interpret
alpha = 0.05
if p > alpha:
    print('Same distributions (fail to reject H0)')
else:
    print('Different distributions (reject H0)')
# compare samples
stat, p = stats.kruskal(df_churn['DeviceProtection'], df_churn['StreamingMovies'], df_churn['PhoneService'])
print('Statistics=%.3f, p=%.3f' % (stat, p))
# interpret
alpha = 0.05
if p > alpha:
    print('Same distributions (fail to reject H0)')
else:
    print('Different distributions (reject H0)')
# compare samples
stat, p = stats.kruskal(df_churn['Contract'], df_churn['PaymentMethod'], df_churn['PhoneService'],
df_churn['InternetService'])
print('Statistics=%.3f, p=%.3f' % (stat, p))
# interpret
alpha = 0.05
if p > alpha:if p > alpha:
    print('Same distributions (fail to reject H0)')
else:
    print('Different distributions (reject H0)')
```

6.outliers and replace the outliers

```
do_col <- function(c){
b <- boxplot(c, plot = FALSE)
s1 <- c
s1[which(c %in% b$out)] <- mean(c[which(! c %in% b$out)],na.rm=TRUE)
return(s1)}
# (testvec <- c(rep(1,9),100))
# do_col(testvec)
library(tidyverse)
columns_to_do <- names(select_if(iris,is.numeric))
purrr::map_dfc(columns_to_do,
        ~do_col(iris[[.]])) %>% set_names(columns_to_do)
```

7.Check for Categorical columns and perform encoding.

```
# Define the headers since the data does not have any
headers = ["symboling", "normalized_losses", "make", "fuel_type", "aspiration",
        "num_doors", "body_style", "drive_wheels", "engine_location",
        "wheel_base", "length", "width", "height", "curb_weight",
```

```
          "engine_type", "num_cylinders", "engine_size", "fuel_system",
          "bore", "stroke", "compression_ratio", "horsepower", "peak_rpm",
          "city_mpg", "highway_mpg", "price"]
# Read in the CSV file and convert "?" to NaN
df = pd.read_csv("https://archive.ics.uci.edu/ml/machine-learning-databases/autos/imports-85.data",
                 header=None, names=headers, na_values="?" )
df.head()
```

symboling normalized_losses make fuel_type aspiration num_doors body_style
drive_wheels engine_location wheel_base ... engine_size fuel_system
bore stroke compression_ratio horsepower peak_rpm city_mpg highway_mpg
price

0 3 NaN alfa-romero gas std two convertible rwd front
88.6 ... 130 mpfi 3.47 2.68 9.0 111.0 5000.0 21 27
13495.0

1 3 NaN alfa-romero gas std two convertible rwd front
88.6 ... 130 mpfi 3.47 2.68 9.0 111.0 5000.0 21 27
16500.0

2 1 NaN alfa-romero gas std two hatchback rwd front
94.5 ... 152 mpfi 2.68 3.47 9.0 154.0 5000.0 19 26
16500.0

3 2 164.0 audi gas std four sedan fwd front 99.8 ...
109 mpfi 3.19 3.40 10.0 102.0 5500.0 24 30 13950.0

4 2 164.0 audi gas std four sedan 4wd front 99.4 ...
136 mpfi 3.19 3.40 8.0 115.0 5500.0 18 22 17450.0

5 rows × 26 columns

```
cleanup_nums = {"num_doors":     {"four": 4, "two": 2},
         "num_cylinders": {"four": 4, "six": 6, "five": 5, "eight": 8,
                   "two": 2, "twelve": 12, "three":3 }}
df = df.replace(cleanup_nums)
df.head()
```

symboling normalized_losses make fuel_type aspiration num_doors body_style
drive_wheels engine_location wheel_base ... engine_size fuel_system
bore stroke compression_ratio horsepower peak_rpm city_mpg highway_mpg
price

0 3 NaN alfa-romero gas std 2.0 convertible rwd front
88.6 ... 130 mpfi 3.47 2.68 9.0 111.0 5000.0 21 27
13495.0

1 3 NaN alfa-romero gas std 2.0 convertible rwd front
88.6 ... 130 mpfi 3.47 2.68 9.0 111.0 5000.0 21 27
16500.0

2 1 NaN alfa-romero gas std 2.0 hatchback rwd front
94.5 ... 152 mpfi 2.68 3.47 9.0 154.0 5000.0 19 26
16500.0

3 2 164.0 audi gas std 4.0 sedan fwd front 99.8 ...
109 mpfi 3.19 3.40 10.0 102.0 5500.0 24 30 13950.0

4 2 164.0 audi gas std 4.0 sedan 4wd front 99.4 ...
136 mpfi 3.19 3.40 8.0 115.0 5500.0 18 22 17450.0

5 rows × 26 columns

df.dtypessymboling int64
normalized_losses float64
make object
fuel_type object
aspiration object
num_doors float64
body_style object
drive_wheels object
engine_location object
wheel_base float64
length float64
width float64
height float64
curb_weight int64
engine_type object
num_cylinders int64
engine_size int64
fuel_system object
bore float64
stroke float64
compression_ratio float64
horsepower float64
peak_rpm float64
city_mpg int64
highway_mpg int64
price float64
dtype: object

8.Split the data into training and testing

#Importing Libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
#Importing data
dataset = pd.read_csv('Decision Tree Data.csv')
x = dataset.iloc[:,1:2].values
y =dataset.iloc[:,2].values
#Split Training Set and Testing Set
from sklearn.cross_validation import train_test_split
xtrain, xtest, ytrain, ytest =train_test_split(x,y,test_size=0.2

9.Split the data into dependent and independent variables.

X = df.iloc[:, :-1].values
print(X) [3 nan 'alfa-romero' ... 5000.0 21 27]
 [1 nan 'alfa-romero' ... 5000.0 19 26]
 ...
 [-1 95.0 'volvo' ... 5500.0 18 23]

```
 [-1 95.0 'volvo' ... 4800.0 26 27]
 [-1 95.0 'volvo' ... 5400.0 19 25]]
Y = df.iloc[:, -1].values
print(Y)
[13495. 16500. 16500. 13950. 17450. 15250. 17710. 18920. 23875.    nan
 16430. 16925. 20970. 21105. 24565. 30760. 41315. 36880.  5151.  6295.
  6575.  5572.  6377.  7957.  6229.  6692.  7609.  8558.  8921. 12964.
  6479.  6855.  5399.  6529.  7129.  7295.  7295.  7895.  9095.  8845.
 10295. 12945. 10345.  6785.    nan    nan 11048. 32250. 35550. 36000.
  5195.  6095.  6795.  6695.  7395. 10945. 11845. 13645. 15645.  8845.
  8495. 10595. 10245. 10795. 11245. 18280. 18344. 25552. 28248. 28176.
 31600. 34184. 35056. 40960. 45400. 16503.  5389.  6189.  6669.  7689.
  9959.  8499. 12629. 14869. 14489.  6989.  8189.  9279.  9279.  5499.
  7099.  6649.  6849.  7349.  7299.  7799.  7499.  7999.  8249.  8949.
  9549. 13499. 14399. 13499. 17199. 19699. 18399. 11900. 13200. 12440.
 13860. 15580. 16900. 16695. 17075. 16630. 17950. 18150.  5572.  7957.
  6229.  6692.  7609.  8921. 12764. 22018. 32528. 34028. 37028.    nan
  9295.  9895. 11850. 12170. 15040. 15510. 18150. 18620.  5118.  7053.
  7603.  7126.  7775.  9960.  9233. 11259.  7463. 10198.  8013. 11694.
  5348.  6338.  6488.  6918.  7898.  8778.  6938.  7198.  7898.  7788.
  7738.  8358.  9258.  8058.  8238.  9298.  9538.  8449.  9639.  9989.
 11199. 11549. 17669.  8948. 10698.  9988. 10898. 11248. 16558. 15998.
 15690. 15750.  7775.  7975.  7995.  8195.  8495.  9495.  9995. 11595.
  9980. 13295. 13845. 12290. 12940. 13415. 15985. 16515. 18420. 18950.
 16845. 19045. 21485. 22470. 22625.]
```

10..Scale the independent variables

```
columns = df.columns
binary_cols = []
for col in columns:
    if df[col].value_counts().shape[0] == 2:
    binary_cols.append(col)
```