Data Collection

```
import numpy as np
import pandas as pd
pd.set_option('display.max_columns', None)

import plotly.express as px
import matplotlib.pyplot as plt

data_df = pd.read_csv("/content/customer_churn_large_dataset.csv")

#Get overview of the data
def dataoveriew(df, message):
    print(f'{message}:n')
    print('Number of rows: ', df.shape[0])
    print("nNumber of features:", df.shape[1])
    print("nData Features:")
    print(df.columns.tolist())
    print("nMissing values:", df.isnull().sum().values.sum())
    print("nUnique values:")
    print(df.nunique())

dataoveriew(data_df, 'Overview of the dataset')
```

```
Overview of the dataset:n
Number of rows:  100000
nNumber of features: 9
nData Features:
['CustomerID', 'Name', 'Age', 'Gender', 'Location', 'Subscription_Length_Months', 'Monthly_Bill', 'Total_Usage_GB
nMissing values: 0
nUnique values:
CustomerID                    100000
Name                          100000
Age                               53
Gender                             2
Location                           5
Subscription_Length_Months        24
Monthly_Bill                    7001
Total_Usage_GB                   451
Churn                              2
dtype: int64
```

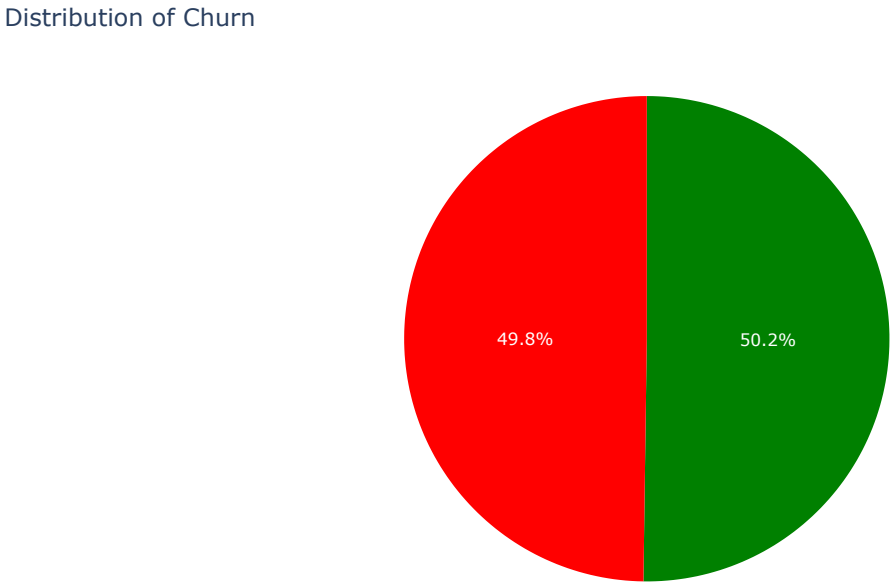
```
data_df
```

	CustomerID	Name	Age	Gender	Location	Subscription_Length_Months	Monthly_Bill	Total_Usage_GB
0	1	Customer_1	63	Male	Los Angeles	17	73.36	236
1	2	Customer_2	62	Female	New York	1	48.76	172
2	3	Customer_3	24	Female	Los Angeles	5	85.47	460
3	4	Customer_4	36	Female	Miami	3	97.94	297
4	5	Customer_5	46	Female	Miami	19	58.14	266
...	...	...	...	...	...	...	...	...
99995	99996	Customer_99996	33	Male	Houston	23	55.13	226
99996	99997	Customer_99997	62	Female	New York	19	61.65	351
99997	99998	Customer_99998	64	Male	Chicago	17	96.11	251
99998	99999	Customer_99999	51	Female	New York	20	49.25	434
99999	100000	Customer_100000	27	Female	Los Angeles	19	76.57	173

100000 rows × 9 columns

Data Visualisation

```
target_instance = data_df["Churn"].value_counts().to_frame()
target_instance = target_instance.reset_index()
target_instance = target_instance.rename(columns={'index': 'Category'})
fig = px.pie(target_instance, values='Churn', names='Category', color_discrete_sequence=["green", "red"],
             title='Distribution of Churn')
fig.show()
```

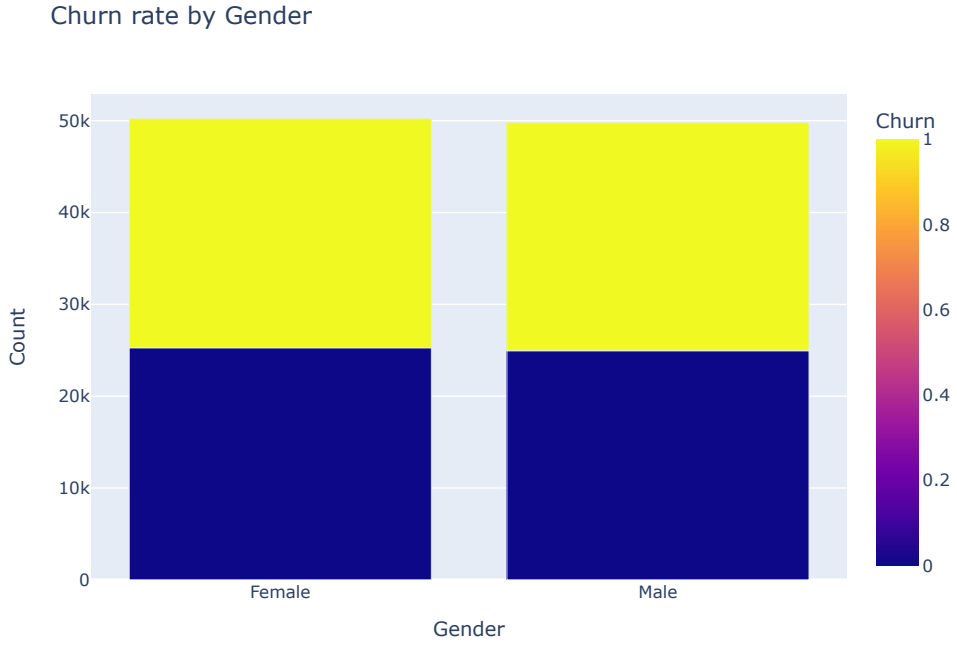



```
def bar(feature, df=data_df ):
    temp_df = df.groupby([feature, 'Churn']).size().reset_index()
    temp_df = temp_df.rename(columns={0:'Count'})

    value_counts_df = df[feature].value_counts().to_frame().reset_index()
    categories = [cat[1][0] for cat in value_counts_df.iterrows()]

    num_list = [num[1][1] for num in value_counts_df.iterrows()]
    div_list = [element / sum(num_list) for element in num_list]
    percentage = [round(element * 100,1) for element in div_list]

    def num_format(list_instance):
        formatted_str = ''
        for index,num in enumerate(list_instance):
            if index < len(list_instance)-2:
                formatted_str=formatted_str+f'{num}%, '
            elif index == len(list_instance)-2:
                formatted_str=formatted_str+f'{num}% & '
            else:
                formatted_str=formatted_str+f'{num}%'
        return formatted_str

    def str_format(list_instance):
        formatted_str = ''
        for index, cat in enumerate(list_instance):
            if index < len(list_instance)-2:
                formatted_str=formatted_str+f'{cat}, '
            elif index == len(list_instance)-2:
                formatted_str=formatted_str+f'{cat} & '
            else:
                formatted_str=formatted_str+f'{cat}'
        return formatted_str

    num_str = num_format(percentage)
    cat_str = str_format(categories)


    fig = px.bar(temp_df, x=feature, y='Count', color='Churn', title=f'Churn rate by {feature}', barmode="group", color
    fig.add_annotation(
                text=f'Value count of distribution of {cat_str} are<br>{num_str} percentage respectively.',
                align='left',
                showarrow=False,
```

```
                xref='paper',
                yref='paper',
                x=1.4,
                y=1.3,
                bordercolor='black',
                borderwidth=1)
    fig.update_layout(
        margin=dict(r=400),
    )

    return fig.show()
```

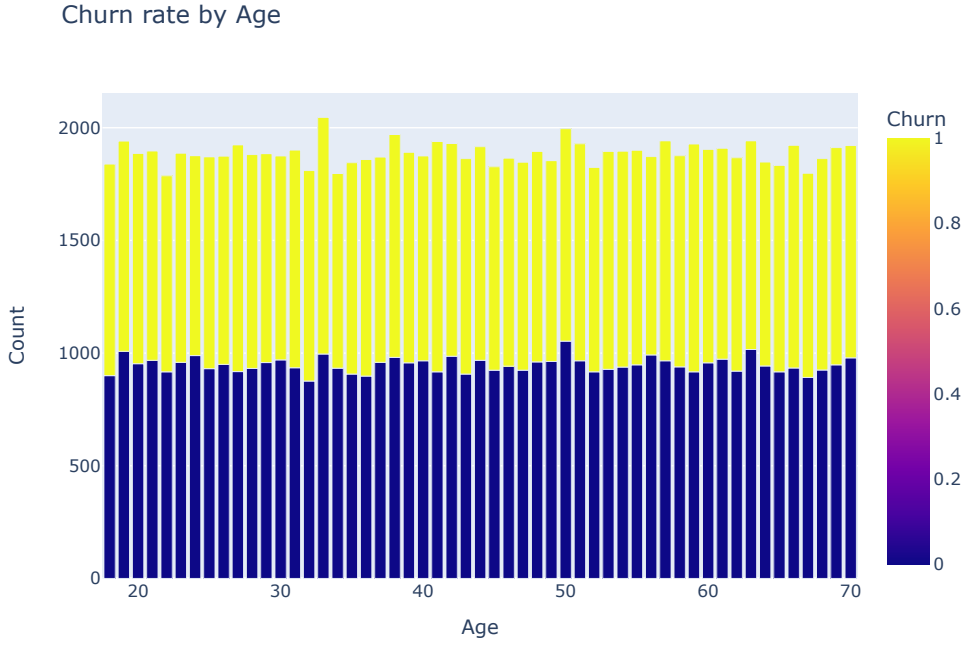
```
bar('Gender')
```

Value count of distribution of Female & Male are 50.2% & 49.8% percentage respectively.




```
bar('Age')
```

%, 1.9%, 1.9%, 1.9%, 1.9%, 1.9%, 1.8%, 1.8%, 1.8%, 1.8%, 1.8%, 1.8%, 1.8%, 1.8%, 1.8%, 1.8% & 1.8% percentage respectively.

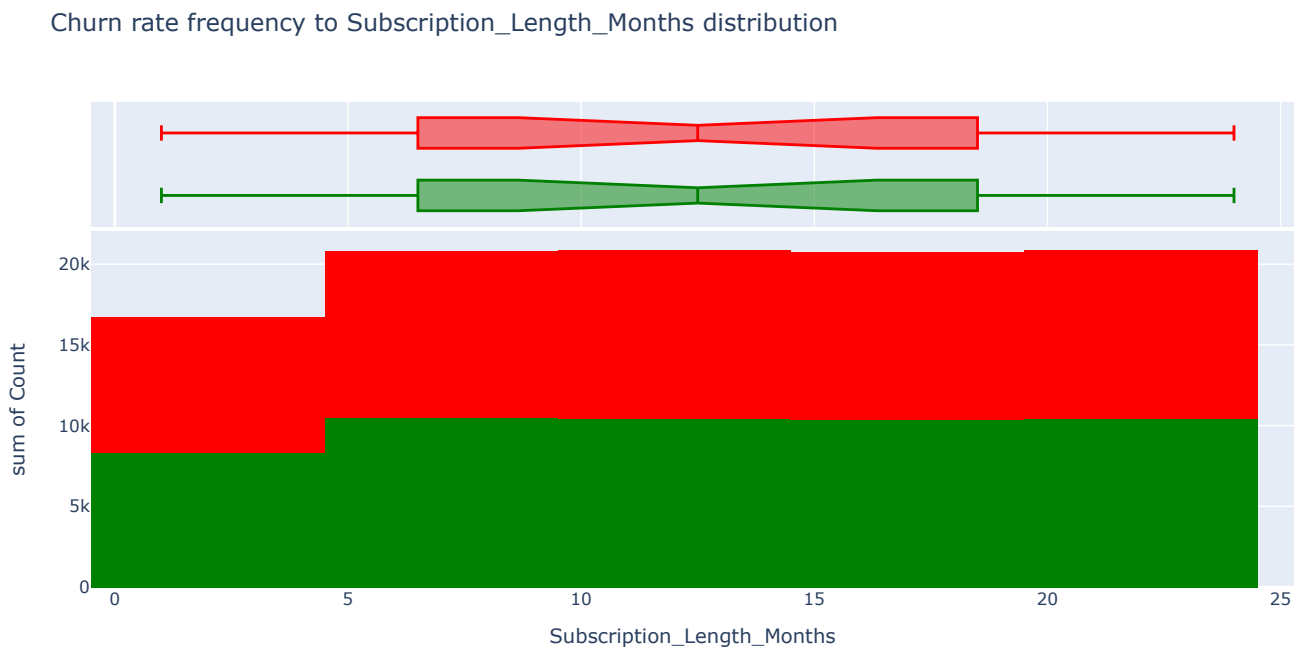



```
data_df.dtypes
```

```
CustomerID                    int64
Name                         object
Age                           int64
Gender                       object
Location                     object
Subscription_Length_Months    int64
Monthly_Bill                float64
Total_Usage_GB                int64
Churn                         int64
dtype: object
```

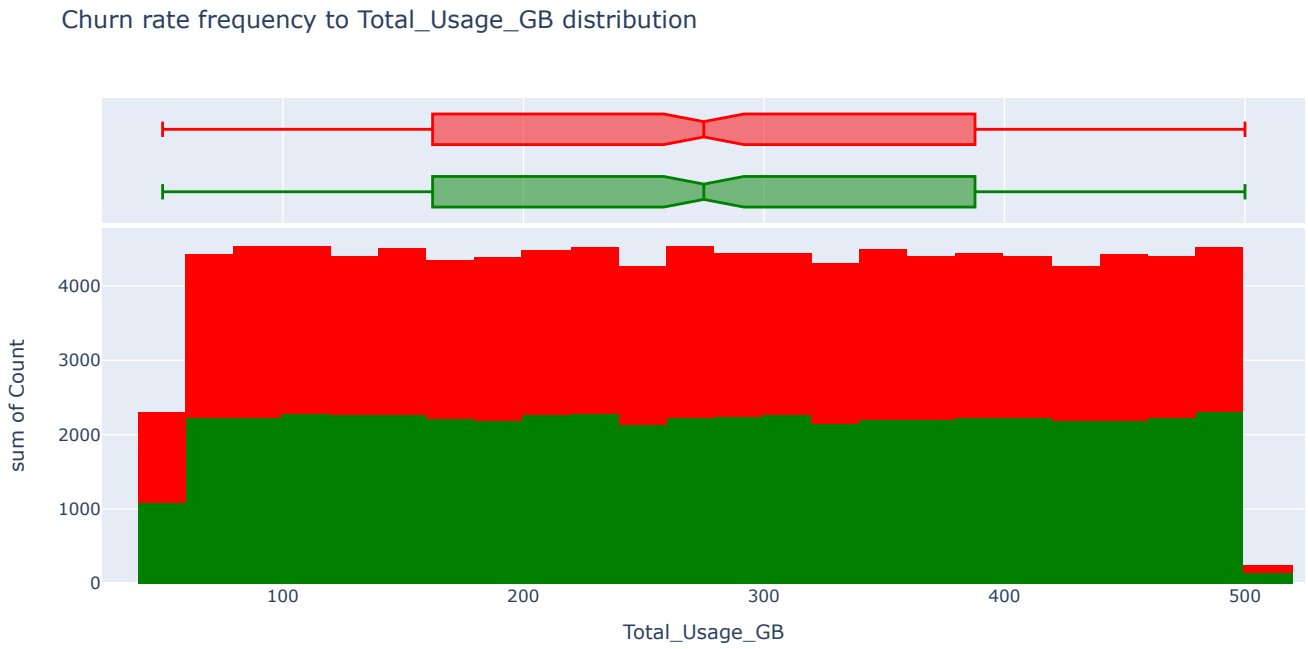
```
def hist(feature):
    group_df = data_df.groupby([feature, 'Churn']).size().reset_index()
    group_df = group_df.rename(columns={0: 'Count'})
    fig = px.histogram(group_df, x=feature, y='Count', color='Churn', marginal='box', title=f'Churn rate frequency to {
    fig.show()
```

```
hist('Subscription_Length_Months')
```



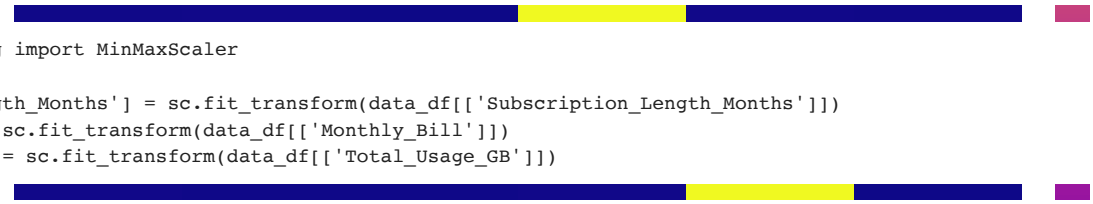

```
hist('Monthly_Bill')
```

```
hist('Total_Usage_GB')
```




```
data_df.drop(["CustomerID", "Name", "Location"],axis=1,inplace = True)

def binary_map(feature):
    return feature.map({'Yes':1, 'No':0})

data_df['Gender'] = data_df['Gender'].map({'Male':1, 'Female':0})

data_df = pd.get_dummies(data_df, drop_first=True)


corr = data_df.corr()

fig = px.imshow(corr,width=1000, height=1000)
fig.show()
```

Data Cleaning/Structuring



```
from sklearn.preprocessing import MinMaxScaler
sc = MinMaxScaler()
data_df['Subscription_Length_Months'] = sc.fit_transform(data_df[['Subscription_Length_Months']])
data_df['Monthly_Bill'] = sc.fit_transform(data_df[['Monthly_Bill']])
data_df['Total_Usage_GB'] = sc.fit_transform(data_df[['Total_Usage_GB']])
```



```
data_df
```

	Age	Gender	Subscription_Length_Months	Monthly_Bill	Total_Usage_GB	Churn
0	63	1	0.695652	0.619429	0.413333	0
1	62	0	0.000000	0.268000	0.271111	0
2	24	0	0.173913	0.792429	0.911111	0
3	36	0	0.086957	0.970571	0.548889	1
4	46	0	0.782609	0.402000	0.480000	0
...	...	...	...	...	...	...
99995	33	1	0.956522	0.359000	0.391111	1
99996	62	0	0.782609	0.452143	0.668889	0
99997	64	1	0.695652	0.944429	0.446667	1
99998	51	0	0.826087	0.275000	0.853333	1
99999	27	0	0.782609	0.665286	0.273333	1

100000 rows × 6 columns

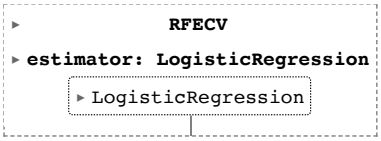
Model Selection

```
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB

from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

data_df

from sklearn.model_selection import train_test_split
X = data_df.drop('Churn', axis=1)
y = data_df['Churn']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=50)
```

```
def modeling(alg, alg_name, params={}):
    model = alg(**params)
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)

    def print_scores(alg, y_true, y_pred):
        print(alg_name)
        acc_score = accuracy_score(y_true, y_pred)
        print("accuracy: ",acc_score)
        pre_score = precision_score(y_true, y_pred)
        print("precision: ",pre_score)
        rec_score = recall_score(y_true, y_pred)
        print("recall: ",rec_score)
        f_score = f1_score(y_true, y_pred, average='weighted')
        print("f1_score: ",f_score)

    print_scores(alg, y_test, y_pred)
    return model

log_model = modeling(LogisticRegression, 'Logistic Regression')
```

```
Logistic Regression
accuracy:  0.5036
precision:  0.49765094832086304
recall:  0.28885971114028886
f1_score:  0.4800498499176361
```

```
from sklearn.feature_selection import RFECV
from sklearn.model_selection import StratifiedKFold
log = LogisticRegression()
rfecv = RFECV(estimator=log, cv=StratifiedKFold(10, random_state=50, shuffle=True), scoring="accuracy")
rfecv.fit(X, y)
```




```
X_rfe = X.iloc[:, rfecv.support_]

print("X dimension: {}".format(X.shape))
print("X column list:", X.columns.tolist())
print("X_rfe dimension: {}".format(X_rfe.shape))
print("X_rfe column list:", X_rfe.columns.tolist())
```

```
X dimension: (100000, 5)
X column list: ['Age', 'Gender', 'Subscription_Length_Months', 'Monthly_Bill', 'Total_Usage_GB']
X_rfe dimension: (100000, 1)
X_rfe column list: ['Total_Usage_GB']
```

```
svc_model = modeling(SVC, 'SVC Classification')
```

```
SVC Classification
accuracy:  0.50495
precision:  0.0
recall:  0.0
f1_score:  0.33884780557493605
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning:

Precision is ill-defined and being set to 0.0 due to no predicted samples. Use `zero_division` parameter to contro
```

```
#Random forest
rf_model = modeling(RandomForestClassifier, "Random Forest Classification")
```

```
Random Forest Classification
accuracy:  0.49975
precision:  0.4946080464537536
recall:  0.48176951823048175
f1_score:  0.49960368243714137
```

```
#Decision tree
dt_model = modeling(DecisionTreeClassifier, "Decision Tree Classification")
```

```
Decision Tree Classification
accuracy:  0.49935
precision:  0.4944223107569721
recall:  0.5013634986365013
f1_score:  0.49936026466595324
```

```
#Naive bayes
nb_model = modeling(GaussianNB, "Naive Bayes Classification")
```

```
Naive Bayes Classification
accuracy:  0.5028
precision:  0.4964076858813701
recall:  0.3000706999293001
f1_score:  0.4818899883073551
```

```
# define model
model = LogisticRegression()

from sklearn.model_selection import RepeatedStratifiedKFold
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)

from scipy.stats import loguniform
space = dict()
space['solver'] = ['newton-cg', 'lbfgs', 'liblinear']
space['penalty'] = ['none', 'l1', 'l2', 'elasticnet']
space['C'] = loguniform(1e-5, 1000)

from sklearn.model_selection import RandomizedSearchCV
search = RandomizedSearchCV(model, space, n_iter=500, scoring='accuracy', n_jobs=-1, cv=cv, random_state=1)

result = search.fit(X_rfe, y)
params = result.best_params_

log_model = modeling(LogisticRegression, 'Logistic Regression Classification', params=params)
```

```
 0.5017667  0.5017067         nan         nan         nan         nan
        nan 0.50179667 0.50199333         nan 0.50156667         nan
 0.50176667         nan         nan 0.50172           nan         nan
        nan         nan 0.50177333         nan 0.50176667 0.50170667
 0.50178            nan 0.50213333         nan 0.50176667         nan
 0.50176667         nan]

Logistic Regression Classification
accuracy:  0.5016
precision:  0.49098250336473753
recall:  0.1842238157761842
f1 score:  0.44681134405426065
```

```
import joblib

filename = 'model.sav'
joblib.dump(log_model, filename)
```

```
['model.sav']
```

✓ 0s completed at 10:45 PM