EDA and Prediction

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.ticker as mtick
import matplotlib.pyplot as plt

sns.set(style = 'white')


telecom_cust = pd.read_csv('/content/Telco-Customer-Churn.csv')


telecom_cust.head()
```

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	...
0	7590-VHVEG	Female	0	Yes	No	1	No	No phone service	DSL	No	...
1	5575-GNVDE	Male	0	No	No	34	Yes	No	DSL	Yes	...
2	3668-QPYBK	Male	0	No	No	2	Yes	No	DSL	Yes	...
3	7795-CFOCW	Male	0	No	No	45	No	No phone service	DSL	Yes	...
4	9237-HQITU	Female	0	No	No	2	Yes	No	Fiber optic	No	...

5 rows × 21 columns

```
import pandas as pd
from bokeh.io import output_file, show
from bokeh.plotting import figure
from bokeh.models import ColumnDataSource, BasicTicker, ColorBar
from bokeh.transform import linear_cmap
from bokeh.palettes import RdBu
import numpy as np


df = pd.read_csv('/content/Telco-Customer-Churn.csv')

numeric_df = df.select_dtypes(include=['float64', 'int64'])

corr_matrix = numeric_df.corr()

columns = list(corr_matrix.columns)
rows = list(corr_matrix.index)

data = {'x': [], 'y': [], 'corr': []}
for i, col in enumerate(columns):
    for j, row in enumerate(rows):
        data['x'].append(col)
        data['y'].append(row)
        data['corr'].append(corr_matrix.iloc[i, j])

source = ColumnDataSource(data)

mapper = linear_cmap(field_name='corr', palette=RdBu[9], low=-1, high=1)

p = figure(
    x_range=columns,
    y_range=list(reversed(rows)),
    width=800,
    height=600,
    title='Correlation Heatmap',
    toolbar_location=None,
    tools='',
    x_axis_location='above'
)

p.rect(
    x='x',
    y='y',
    width=1,
    height=1,
    source=source,
    fill_color=mapper,
    line_color=None
)

color_bar = ColorBar(
    color_mapper=mapper['transform'],
    location=(0, 0),
    ticker=BasicTicker(desired_num_ticks=10)
)
p.add_layout(color_bar, 'right')

p.axis.axis_line_color = None
p.axis.major_tick_line_color = None
p.axis.major_label_text_font_size = "8pt"
p.axis.major_label_standoff = 0
p.xaxis.major_label_orientation = np.pi / 3

output_file("correlation_heatmap.html")
show(p)

print(corr_matrix)

print(numeric_df.info())

telecom_cust.columns.values
```

```
                   SeniorCitizen    tenure  MonthlyCharges
SeniorCitizen        1.000000  0.016567        0.220173
tenure               0.016567  1.000000        0.247900
MonthlyCharges       0.220173  0.247900        1.000000
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7043 entries, 0 to 7042
Data columns (total 3 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   SeniorCitizen   7043 non-null   int64
 1   tenure          7043 non-null   int64
```

```
 2   MonthlyCharges  7043 non-null   float64
dtypes: float64(1), int64(2)
memory usage: 165.2 KB
None
array(['customerID', 'gender', 'SeniorCitizen', 'Partner', 'Dependents',
       'tenure', 'PhoneService', 'MultipleLines', 'InternetService',
       'OnlineSecurity', 'OnlineBackup', 'DeviceProtection',
       'TechSupport', 'StreamingTV', 'StreamingMovies', 'Contract',
       'PaperlessBilling', 'PaymentMethod', 'MonthlyCharges',
       'TotalCharges', 'Churn'], dtype=object)
```

Let's explore the data to see if there are any missing values.

```
# Checking the data types of all the columns
telecom_cust.dtypes
```

```
customerID           object
gender               object
SeniorCitizen         int64
Partner              object
Dependents           object
tenure                int64
PhoneService         object
MultipleLines        object
InternetService      object
OnlineSecurity       object
OnlineBackup         object
DeviceProtection     object
TechSupport          object
StreamingTV          object
StreamingMovies      object
Contract             object
PaperlessBilling     object
PaymentMethod        object
MonthlyCharges      float64
TotalCharges         object
Churn                object
dtype: object
```

```
# Converting Total Charges to a numerical data type.
telecom_cust.TotalCharges = pd.to_numeric(telecom_cust.TotalCharges, errors='coerce')
telecom_cust.isnull().sum()
```

```
customerID           0
gender               0
SeniorCitizen        0
Partner              0
Dependents           0
tenure               0
PhoneService         0
MultipleLines        0
InternetService      0
OnlineSecurity       0
OnlineBackup         0
DeviceProtection     0
TechSupport          0
StreamingTV          0
StreamingMovies      0
Contract             0
PaperlessBilling     0
PaymentMethod        0
MonthlyCharges       0
TotalCharges        11
Churn                0
dtype: int64
```

After looking at the above output, we can say that there are 11 missing values for Total Charges. Let us replace remove these 11 rows from our data set

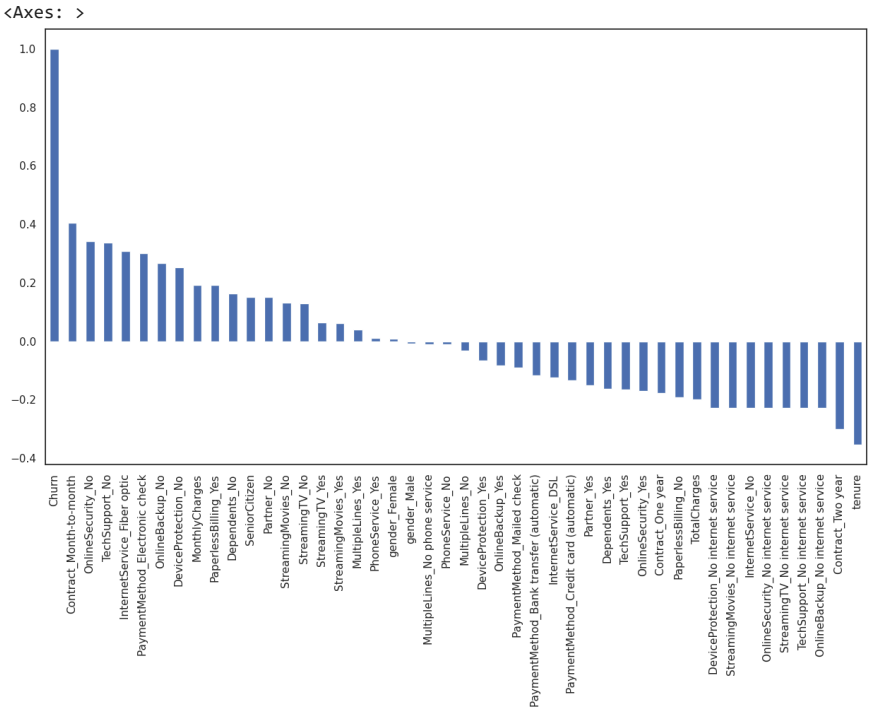
```
#Removing missing values
telecom_cust.dropna(inplace = True)
#Remove customer IDs from the data set
df2 = telecom_cust.iloc[:,1:]
#Convertin the predictor variable in a binary numeric variable
df2['Churn'].replace(to_replace='Yes', value=1, inplace=True)
df2['Churn'].replace(to_replace='No',  value=0, inplace=True)

#Let's convert all the categorical variables into dummy variables
df_dummies = pd.get_dummies(df2)
df_dummies.head()
```

	SeniorCitizen	tenure	MonthlyCharges	TotalCharges	Churn	gender_Female	gender_
0	0	1	29.85	29.85	0	True	F
1	0	34	56.95	1889.50	0	False	
2	0	2	53.85	108.15	1	False	
3	0	45	42.30	1840.75	0	False	
4	0	2	70.70	151.65	1	True	F

5 rows × 46 columns

```
#Get Correlation of "Churn" with other variables:
plt.figure(figsize=(15,8))
df_dummies.corr()['Churn'].sort_values(ascending = False).plot(kind='bar')
```

```
<Axes: >
```



Month to month contracts, absence of online security and tech support seem to be positively correlated with churn. While, tenure, two year contracts seem to be negatively correlated with churn.

Interestingly, services such as Online security, streaming TV, online backup, tech support, etc. without internet connection seem to be negatively related to churn.

We will explore the patterns for the above correlations below before we delve into modelling and identifying the important variables.

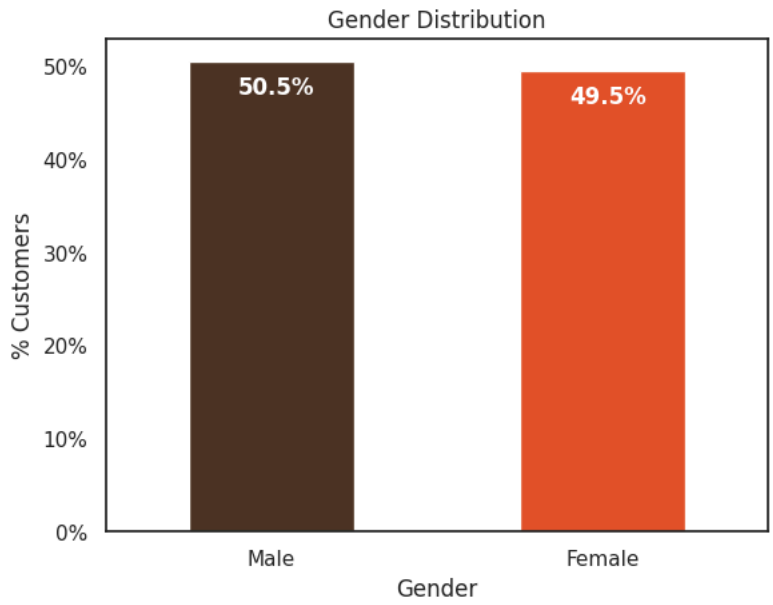
Data Exploration

Let us first start with exploring our data set, to better understand the patterns in the data and potentially form some hypothesis. First we will look at the distribution of individual variables and then slice and dice our data for any interesting trends.

A.) ***Demographics*** - Let us first understand the gender, age range, patner and dependent status of the customers

1. **Gender Distribution** - About half of the customers in our data set are male while the other half are female

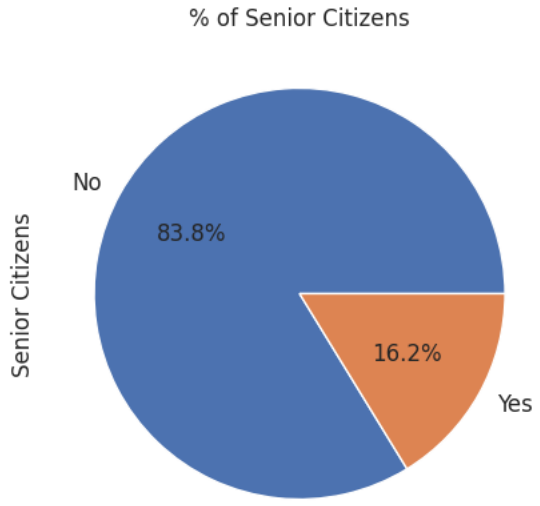
```
colors = ['#4D3425','#E4512B']
ax = (telecom_cust['gender'].value_counts()*100.0 /len(telecom_cust)).plot(kind='bar',
                                                                           stacked = True,
                                                                          rot = 0,
                                                                          color = colors)
ax.yaxis.set_major_formatter(mtick.PercentFormatter())
ax.set_ylabel('% Customers')
ax.set_xlabel('Gender')
ax.set_ylabel('% Customers')
ax.set_title('Gender Distribution')

# create a list to collect the plt.patches data
totals = []

# find the values and append to list
for i in ax.patches:
    totals.append(i.get_width())

# set individual bar lables using above list
total = sum(totals)

for i in ax.patches:
    # get_width pulls left or right; get_y pushes up or down
    ax.text(i.get_x()+.15, i.get_height()-3.5, \
            str(round((i.get_height()/total), 1))+'%',
            fontsize=12,
            color='white',
           weight = 'bold')
```



2. **% Senior Citizens** - There are only 16% of the customers who are senior citizens. Thus most of our customers in the data are younger people.

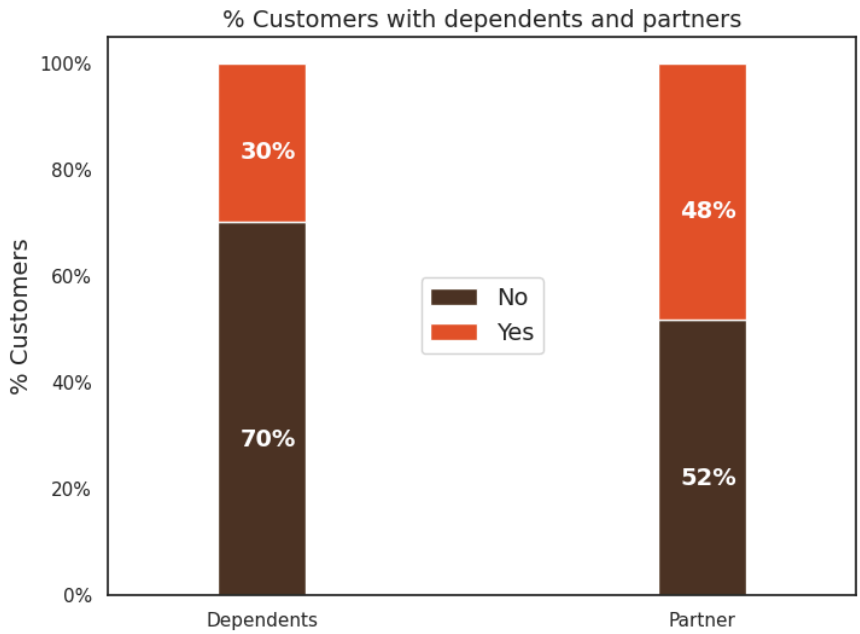
```
ax = (telecom_cust['SeniorCitizen'].value_counts()*100.0 /len(telecom_cust))\
.plot.pie(autopct='%.1f%%', labels = ['No', 'Yes'],figsize =(5,5), fontsize = 12 )
ax.yaxis.set_major_formatter(mtick.PercentFormatter())
ax.set_ylabel('Senior Citizens',fontsize = 12)
ax.set_title('% of Senior Citizens', fontsize = 12)
```

```
Text(0.5, 1.0, '% of Senior Citizens')
```



3. **Partner and dependent status** - About 50% of the customers have a partner, while only 30% of the total customers have dependents.

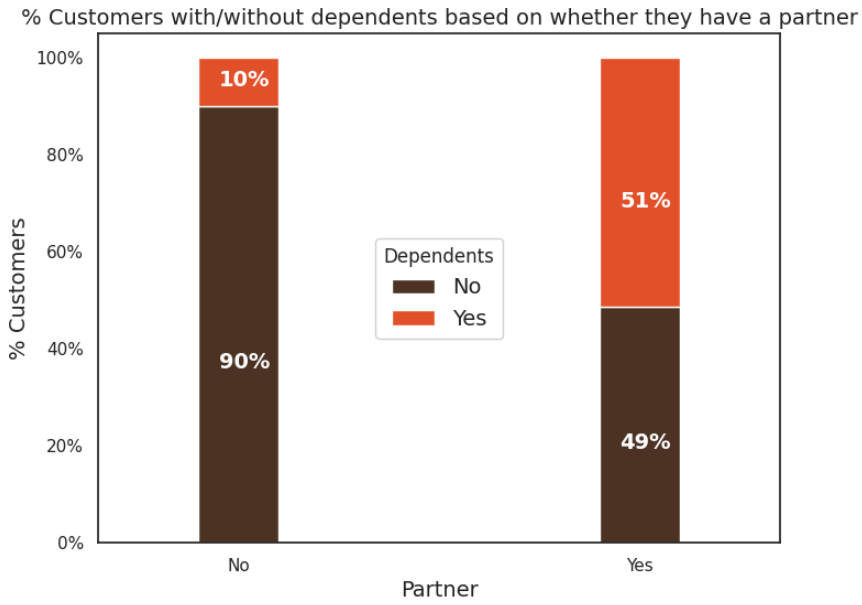
```
df2 = pd.melt(telecom_cust, id_vars=['customerID'], value_vars=['Dependents','Partner'])
df3 = df2.groupby(['variable','value']).count().unstack()
df3 = df3*100/len(telecom_cust)
colors = ['#4D3425','#E4512B']
ax = df3.loc[:,'customerID'].plot.bar(stacked=True, color=colors,
                                      figsize=(8,6),rot = 0,
                                     width = 0.2)

ax.yaxis.set_major_formatter(mtick.PercentFormatter())
ax.set_ylabel('% Customers',size = 14)
ax.set_xlabel('')
ax.set_title('% Customers with dependents and partners',size = 14)
ax.legend(loc = 'center',prop={'size':14})

for p in ax.patches:
    width, height = p.get_width(), p.get_height()
    x, y = p.get_xy()
    ax.annotate('{:.0f}%'.format(height), (p.get_x()+.25*width, p.get_y()+.4*height),
                color = 'white',
               weight = 'bold',
               size = 14)
```



**What would be interesting is to look at the % of customers, who have partners, also have dependents. We will explore this next. **

Interestingly, among the customers who have a partner, only about half of them also have a dependent, while other half do not have any independents. Additionally, as expected, among the customers who do not have any partner, a majority (80%) of them do not have any dependents .

```
colors = ['#4D3425','#E4512B']
partner_dependents = telecom_cust.groupby(['Partner','Dependents']).size().unstack()

ax = (partner_dependents.T*100.0 / partner_dependents.T.sum()).T.plot(kind='bar',
                                                                      width = 0.2,
                                                                      stacked = True,
                                                                      rot = 0,
                                                                      figsize = (8,6),
                                                                      color = colors)
ax.yaxis.set_major_formatter(mtick.PercentFormatter())
ax.legend(loc='center',prop={'size':14},title = 'Dependents',fontsize =14)
ax.set_ylabel('% Customers',size = 14)
ax.set_title('% Customers with/without dependents based on whether they have a partner',size = 14)
ax.xaxis.label.set_size(14)

# Code to add the data labels on the stacked bar chart
for p in ax.patches:
    width, height = p.get_width(), p.get_height()
    x, y = p.get_xy()
    ax.annotate('{:.0f}%'.format(height), (p.get_x()+.25*width, p.get_y()+.4*height),
                color = 'white',
               weight = 'bold',
               size = 14)
```

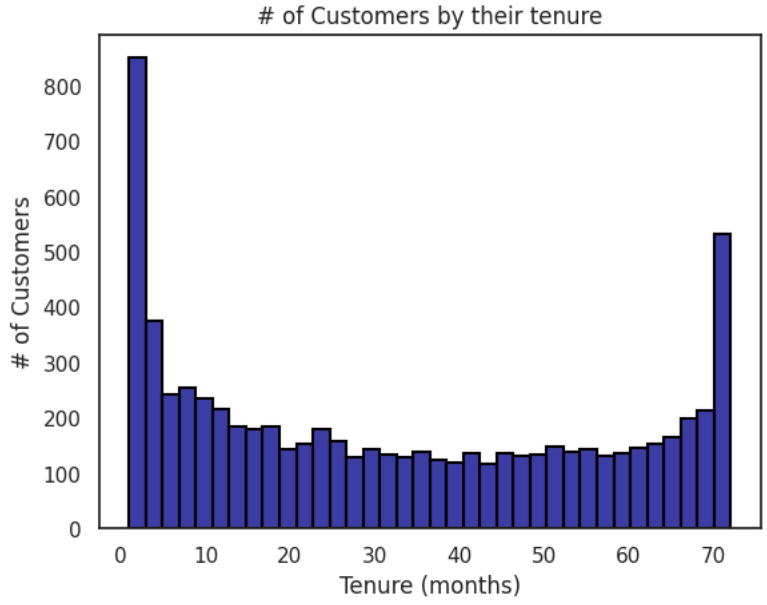


I also looked at any differences between the % of customers with/without dependents and partners by gender. There is no difference in their distribution by gender. Additionally, there is no difference in senior citizen status by gender.

B.) **Customer Account Information**: Let u now look at the tenure, contract

1. Tenure: After looking at the below histogram we can see that a lot of customers have been with the telecom company for just a month, while quite a many are there for about 72 months. This could be potentially because different customers have different contracts. Thus based on the contract they are into it could be more/less easier for the customers to stay/leave the telecom company.

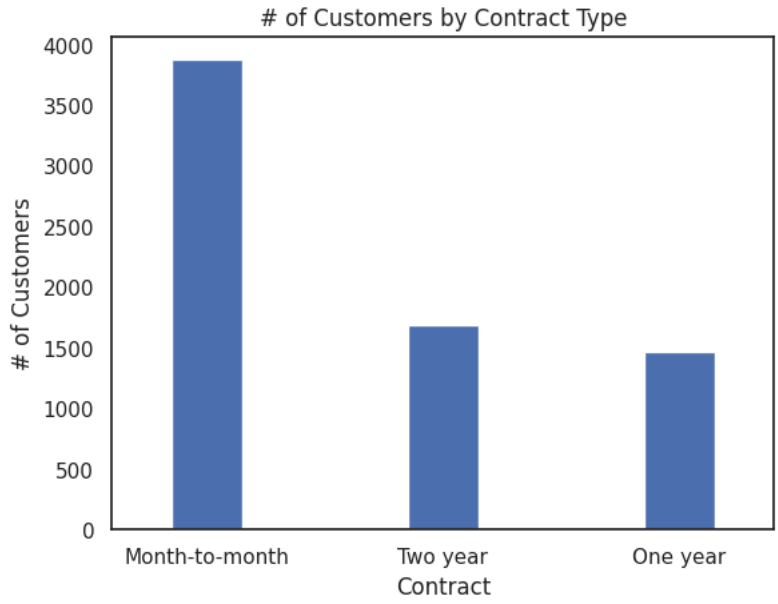
```
import seaborn as sns
import matplotlib.pyplot as plt

ax = sns.histplot(telecom_cust['tenure'], bins=int(180/5), color='darkblue',
                  edgecolor='black', linewidth=1.5)
ax.set_ylabel('# of Customers')
ax.set_xlabel('Tenure (months)')
ax.set_title('# of Customers by their tenure')

plt.show()
```



2. Contracts: To understand the above graph, lets first look at the # of customers by different contracts.

```
ax = telecom_cust['Contract'].value_counts().plot(kind = 'bar',rot = 0, width = 0.3)
ax.set_ylabel('# of Customers')
ax.set_title('# of Customers by Contract Type')
```

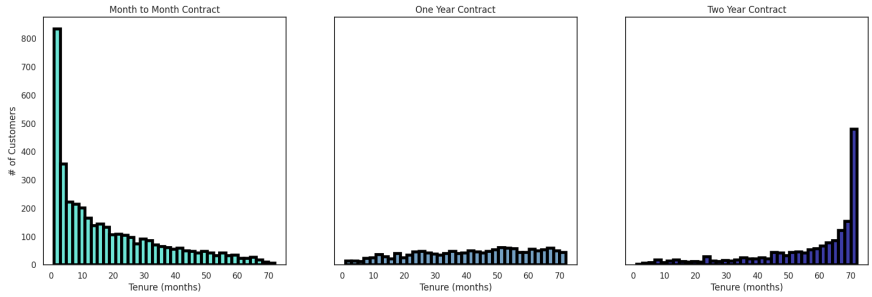
```
Text(0.5, 1.0, '# of Customers by Contract Type')
```



As we can see from this graph most of the customers are in the month to month contract. While there are equal number of customers in the 1 year and 2 year contracts.

Below we will understand the tenure of customers based on their contract type.

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

fig, (ax1, ax2, ax3) = plt.subplots(nrows=1, ncols=3, sharey=True, figsize=(20, 6))

sns.histplot(data=telecom_cust[telecom_cust['Contract'] == 'Month-to-month'], x='tenure',
             bins=int(180/5), color='turquoise', edgecolor='black', linewidth=4,
             ax=ax1)
ax1.set_ylabel('# of Customers')
ax1.set_xlabel('Tenure (months)')
ax1.set_title('Month to Month Contract')

sns.histplot(data=telecom_cust[telecom_cust['Contract'] == 'One year'], x='tenure',
             bins=int(180/5), color='steelblue', edgecolor='black', linewidth=4,
             ax=ax2)
ax2.set_xlabel('Tenure (months)')
ax2.set_title('One Year Contract')

sns.histplot(data=telecom_cust[telecom_cust['Contract'] == 'Two year'], x='tenure',
             bins=int(180/5), color='darkblue', edgecolor='black', linewidth=4,
             ax=ax3)
ax3.set_xlabel('Tenure (months)')
ax3.set_title('Two Year Contract')

plt.show()
```



Interestingly most of the monthly contracts last for 1-2 months, while the 2 year contracts tend to last for about 70 months. This shows that the customers taking a longer contract are more loyal to the company and tend to stay with it for a longer period of time.

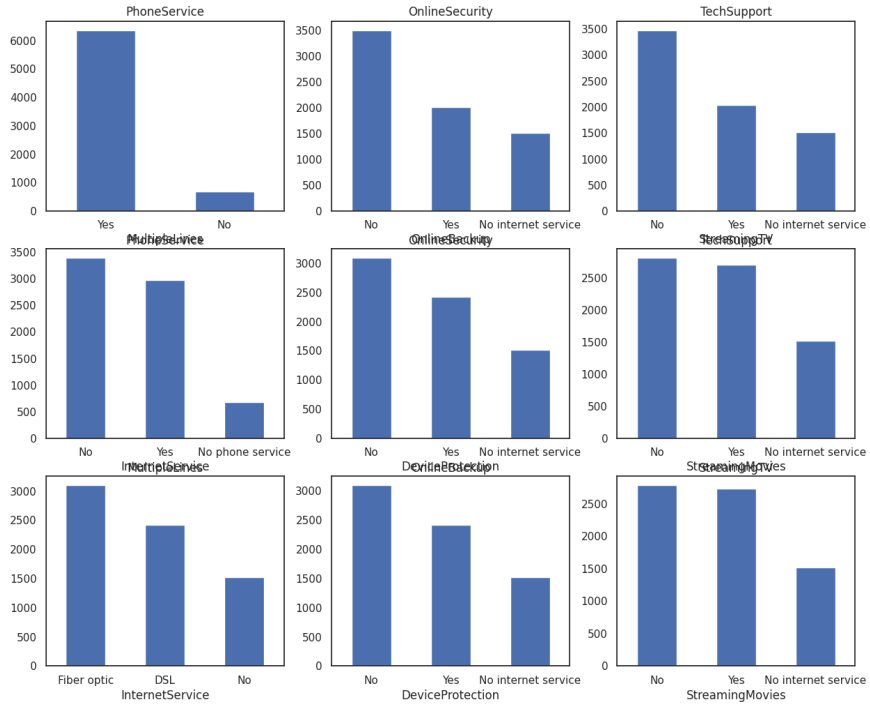
This is also what we saw in the earlier chart on correlation with the churn rate.

C. Let us now look at the distribution of various services used by customers

```
telecom_cust.columns.values
```

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

```
services = ['PhoneService','MultipleLines','InternetService','OnlineSecurity',
           'OnlineBackup','DeviceProtection','TechSupport','StreamingTV','StreamingMovies']

fig, axes = plt.subplots(nrows = 3,ncols = 3,figsize = (15,12))
for i, item in enumerate(services):
    if i < 3:
        ax = telecom_cust[item].value_counts().plot(kind = 'bar',ax=axes[i,0],rot = 0)

    elif i >=3 and i < 6:
        ax = telecom_cust[item].value_counts().plot(kind = 'bar',ax=axes[i-3,1],rot = 0)

    elif i < 9:
        ax = telecom_cust[item].value_counts().plot(kind = 'bar',ax=axes[i-6,2],rot = 0)
    ax.set_title(item)
```

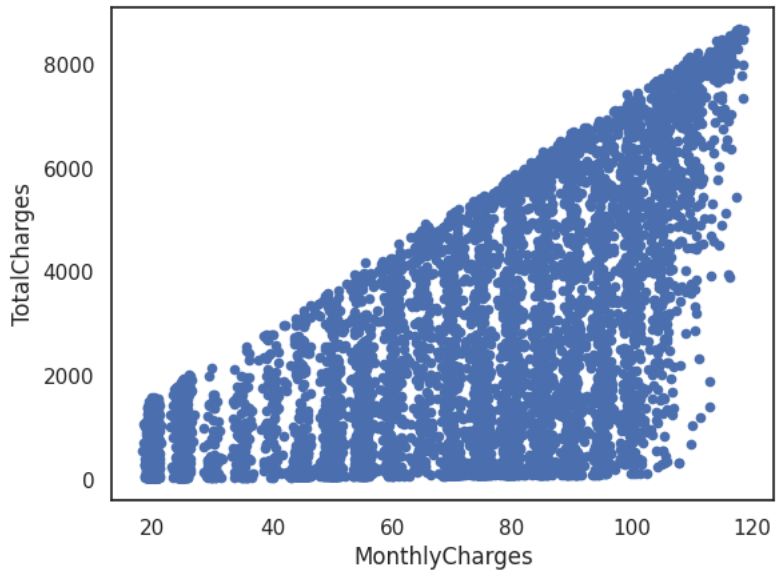


D.) Now let's take a quick look at the relation between monthly and total charges

We will observe that the total charges increases as the monthly bill for a customer increases.

```
telecom_cust[['MonthlyCharges', 'TotalCharges']].plot.scatter(x = 'MonthlyCharges',
                                                              y='TotalCharges')
```

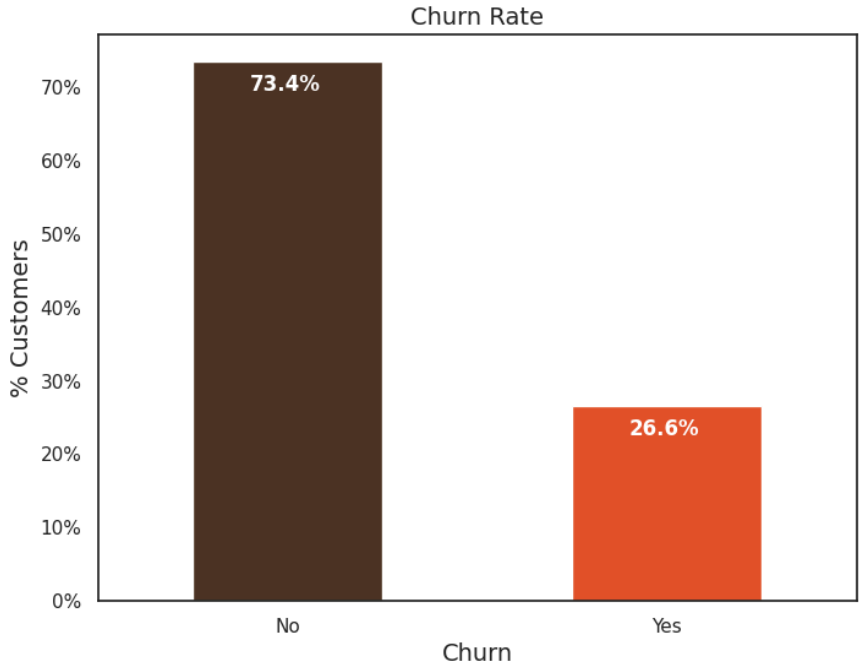
```
<Axes: xlabel='MonthlyCharges', ylabel='TotalCharges'>
```



E.) Finally, let's take a look at out predictor variable (Churn) and understand its interaction with other important variables as was found out in the correlation plot.

1. Lets first look at the churn rate in our data

```
colors = ['#4D3425','#E4512B']
ax = (telecom_cust['Churn'].value_counts()*100.0 /len(telecom_cust)).plot(kind='bar',
                                                                          stacked = True,
                                                                         rot = 0,
                                                                         color = colors,
                                                                        figsize = (8,6))
ax.yaxis.set_major_formatter(mtick.PercentFormatter())
ax.set_ylabel('% Customers',size = 14)
ax.set_xlabel('Churn',size = 14)
ax.set_title('Churn Rate', size = 14)

# create a list to collect the plt.patches data
totals = []

# find the values and append to list
for i in ax.patches:
    totals.append(i.get_width())

# set individual bar lables using above list
total = sum(totals)

for i in ax.patches:
    # get_width pulls left or right; get_y pushes up or down
    ax.text(i.get_x()+.15, i.get_height()-4.0, \
            str(round((i.get_height()/total), 1))+'%',
            fontsize=12,
            color='white',
            weight='bold')
```



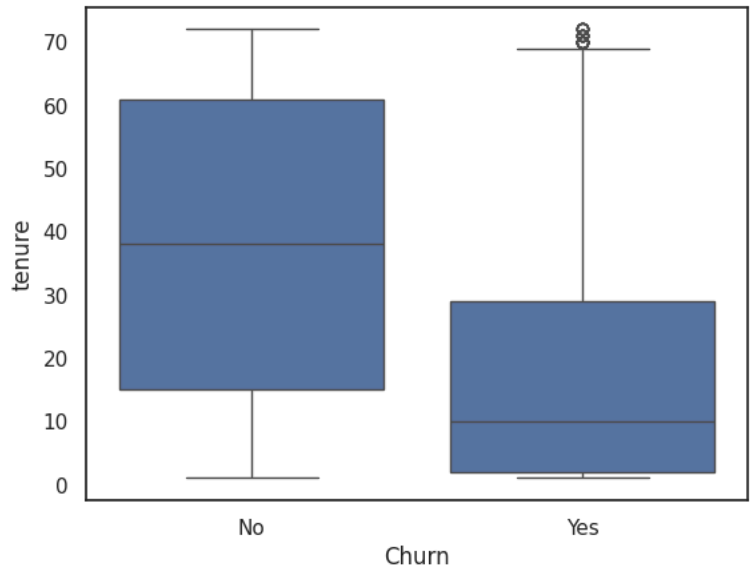
In our data, 74% of the customers do not churn. Clearly the data is skewed as we would expect a large majority of the customers to not churn. This is important to keep in mind for our modelling as skeweness could lead to a lot of false negatives. We will see in the modelling section on how to avoid skewness in the data.

2. Lets now explore the churn rate by tenure, seniority, contract type, monthly charges and total charges to see how it varies by these variables.

i.) Churn vs Tenure: As we can see form the below plot, the customers who do not churn, they tend to stay for a longer tenure with the telecom company.

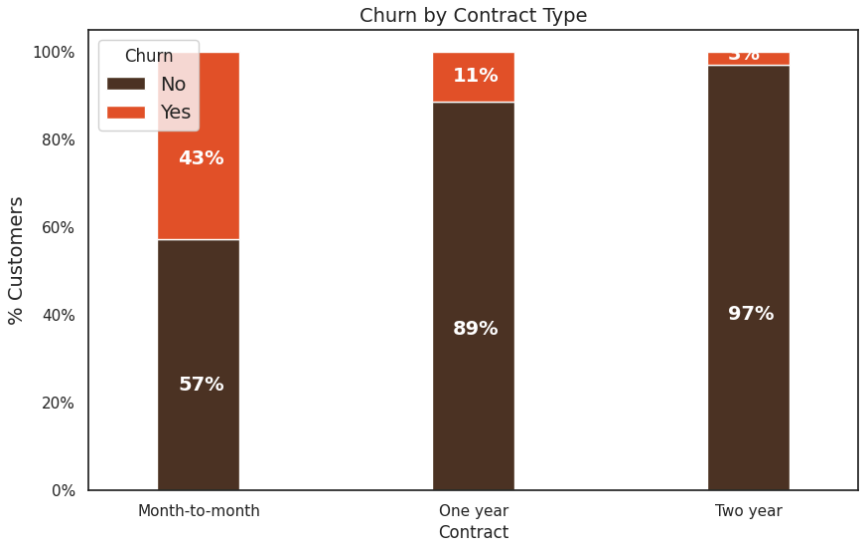
```
sns.boxplot(x = telecom_cust.Churn, y = telecom_cust.tenure)
```

```
<Axes: xlabel='Churn', ylabel='tenure'>
```



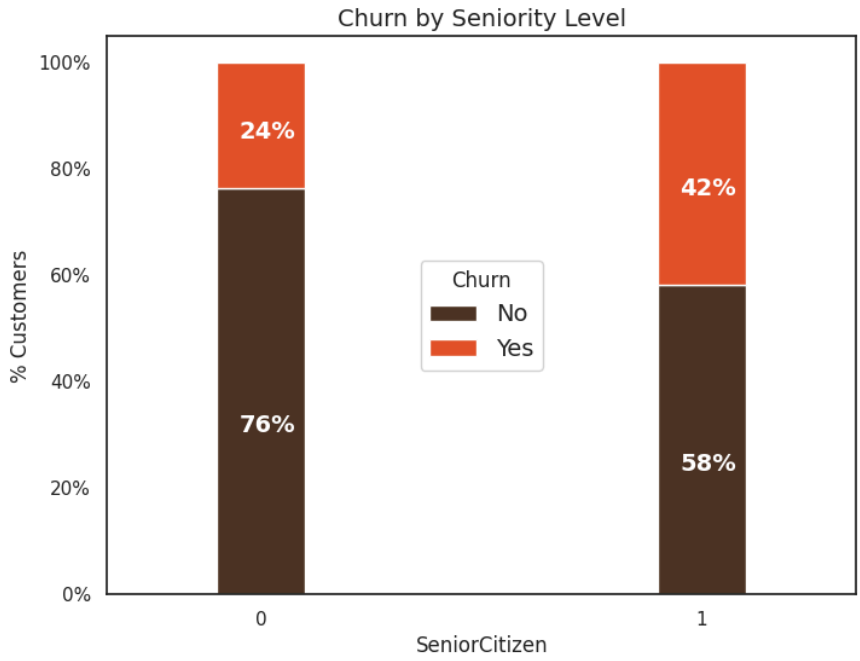
ii.) Churn by Contract Type: Similar to what we saw in the correlation plot, the customers who have a month to month contract have a very high churn rate.

```
colors = ['#4D3425','#E4512B']
contract_churn = telecom_cust.groupby(['Contract','Churn']).size().unstack()

ax = (contract_churn.T*100.0 / contract_churn.T.sum()).T.plot(kind='bar',
                                                              width = 0.3,
                                                              stacked = True,
                                                              rot = 0,
                                                              figsize = (10,6),
                                                              color = colors)
ax.yaxis.set_major_formatter(mtick.PercentFormatter())
ax.legend(loc='best',prop={'size':14},title = 'Churn')
ax.set_ylabel('% Customers',size = 14)
ax.set_title('Churn by Contract Type',size = 14)

# Code to add the data labels on the stacked bar chart
for p in ax.patches:
    width, height = p.get_width(), p.get_height()
    x, y = p.get_xy()
    ax.annotate('{:.0f}%'.format(height), (p.get_x()+.25*width, p.get_y()+.4*height),
                color = 'white',
               weight = 'bold',
               size = 14)
```



iii.) Churn by Seniority: Senior Citizens have almost double the churn rate than younger population.

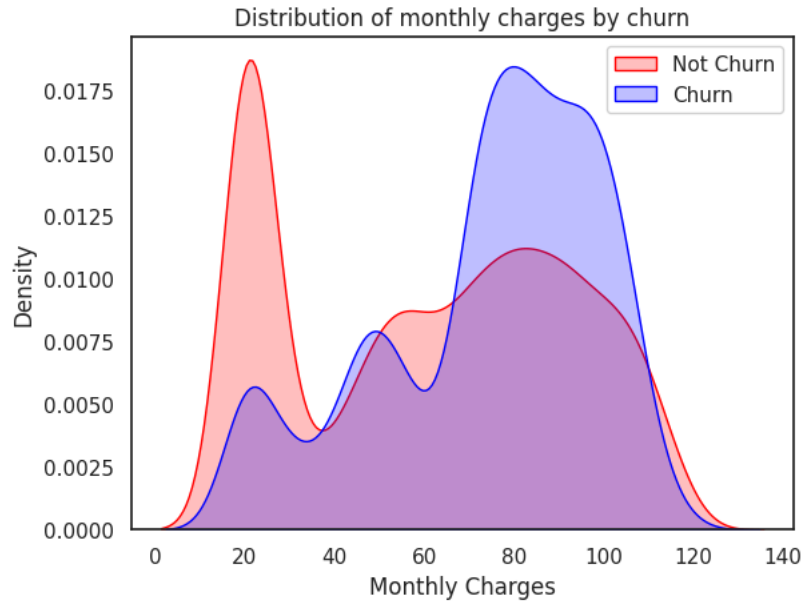
```
colors = ['#4D3425','#E4512B']
seniority_churn = telecom_cust.groupby(['SeniorCitizen','Churn']).size().unstack()

ax = (seniority_churn.T*100.0 / seniority_churn.T.sum()).T.plot(kind='bar',
                                                              width = 0.2,
                                                              stacked = True,
                                                              rot = 0,
                                                              figsize = (8,6),
                                                              color = colors)
ax.yaxis.set_major_formatter(mtick.PercentFormatter())
ax.legend(loc='center',prop={'size':14},title = 'Churn')
ax.set_ylabel('% Customers')
ax.set_title('Churn by Seniority Level',size = 14)

# Code to add the data labels on the stacked bar chart
for p in ax.patches:
    width, height = p.get_width(), p.get_height()
    x, y = p.get_xy()
    ax.annotate('{:.0f}%'.format(height), (p.get_x()+.25*width, p.get_y()+.4*height),
                color = 'white',
               weight = 'bold',size =14)
```



iv.) Churn by Monthly Charges: Higher % of customers churn when the monthly charges are high.

```
import seaborn as sns

ax = sns.kdeplot(telecom_cust.MonthlyCharges[(telecom_cust["Churn"] == 'No')],
                 color="Red", fill=True)
ax = sns.kdeplot(telecom_cust.MonthlyCharges[(telecom_cust["Churn"] == 'Yes')],
                 ax=ax, color="Blue", fill=True)
ax.legend(["Not Churn", "Churn"], loc='upper right')
ax.set_ylabel('Density')
ax.set_xlabel('Monthly Charges')
ax.set_title('Distribution of monthly charges by churn')
```

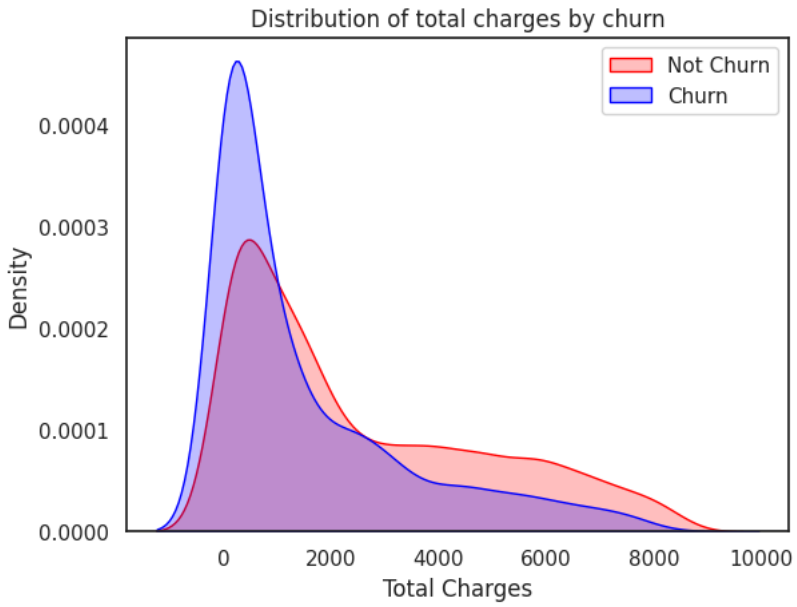
```
Text(0.5, 1.0, 'Distribution of monthly charges by churn')
```



v.) Churn by Total Charges: It seems that there is higer churn when the total charges are lower.

```
import seaborn as sns

ax = sns.kdeplot(telecom_cust.TotalCharges[(telecom_cust["Churn"] == 'No')],
                 color="Red", fill=True)
ax = sns.kdeplot(telecom_cust.TotalCharges[(telecom_cust["Churn"] == 'Yes')],
                 ax=ax, color="Blue", fill=True)
ax.legend(["Not Churn", "Churn"], loc='upper right')
ax.set_ylabel('Density')
ax.set_xlabel('Total Charges')
ax.set_title('Distribution of total charges by churn')
```

```
Text(0.5, 1.0, 'Distribution of total charges by churn')
```



After going through the above EDA we will develop some predictive models and compare them.

We will develop Logistic Regression, Random Forest, SVM, ADA Boost and XG Boost

1. Logistic Regression

```
# We will use the data frame where we had created dummy variables
y = df_dummies['Churn'].values
X = df_dummies.drop(columns = ['Churn'])

# Scaling all the variables to a range of 0 to 1
from sklearn.preprocessing import MinMaxScaler
features = X.columns.values
scaler = MinMaxScaler(feature_range = (0,1))
scaler.fit(X)
X = pd.DataFrame(scaler.transform(X))
X.columns = features


print(df.columns)
df.head(10)
```

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

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	Mult
0	7590-VHVEG	Female	0	Yes	No	1	No	
1	5575-GNVDE	Male	0	No	No	34	Yes	
2	3668-QPYBK	Male	0	No	No	2	Yes	
3	7795-CFOCW	Male	0	No	No	45	No	
4	9237-HQITU	Female	0	No	No	2	Yes	
5	9305-CDSKC	Female	0	No	No	8	Yes	
6	1452-KIOVK	Male	0	No	Yes	22	Yes	
7	6713-OKOMC	Female	0	No	No	10	No	
8	7892-POOKP	Female	0	Yes	No	28	Yes	
9	6388-TABGU	Male	0	No	Yes	62	Yes	

10 rows × 21 columns

It is important to scale the variables in logistic regression so that all of them are within a range of 0 to 1. This helped me improve the accuracy from 79.7% to 80.7%. Further, you will notice below that the importance of variables is also aligned with what we are seeing in Random Forest algorithm and the EDA we conducted above.

```
# Create Train & Test Data
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=101)
```

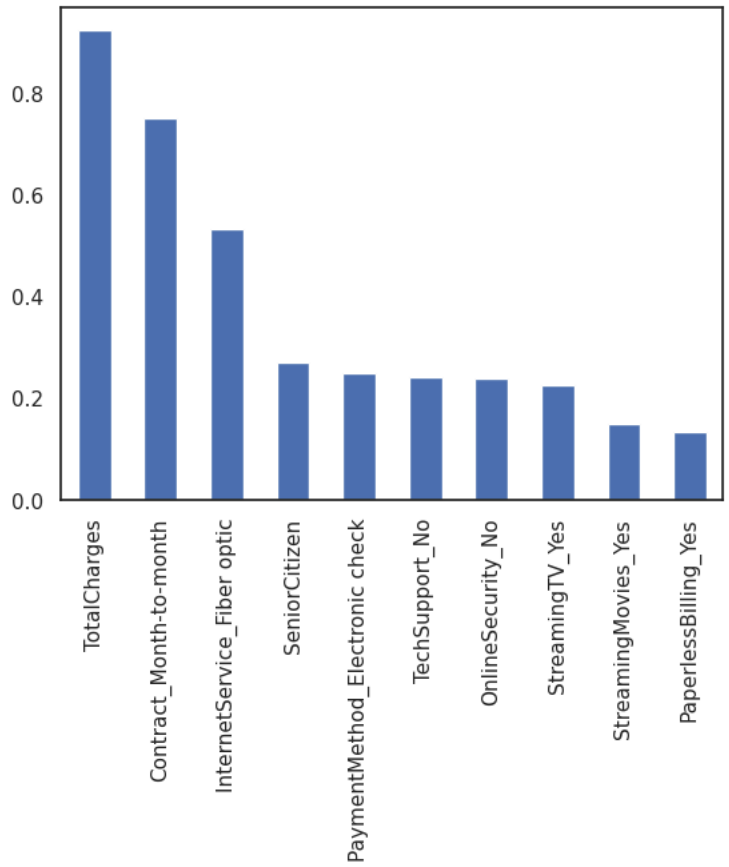
```
# Running logistic regression model
from sklearn.linear_model import LogisticRegression
model = LogisticRegression()
result = model.fit(X_train, y_train)
```

```
from sklearn import metrics
prediction_test = model.predict(X_test)
# Print the prediction accuracy
print (metrics.accuracy_score(y_test, prediction_test))
```

```
0.8075829383886256
```

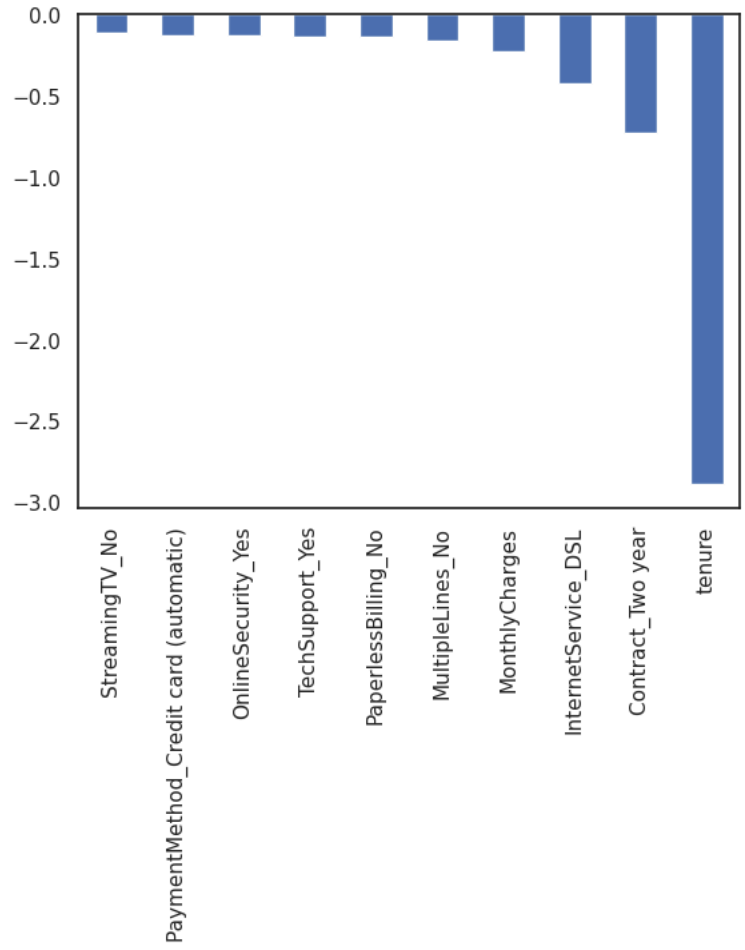
```
# To get the weights of all the variables
weights = pd.Series(model.coef_[0],
                 index=X.columns.values)
print (weights.sort_values(ascending = False)[:10].plot(kind='bar'))
```

```
Axes(0.125,0.11;0.775x0.77)
```




```
print(weights.sort_values(ascending = False)[-10:].plot(kind='bar'))
```

```
Axes(0.125,0.11;0.775x0.77)
```



Observations

We can see that some variables have a negative relation to our predicted variable (Churn), while some have positive relation. Negative relation means that likeliness of churn decreases with that variable. Let us summarize some of the interesting features below:

- As we saw in our EDA, having a 2 month contract reduces chances of churn. 2 month contract along with tenure have the most negative relation with Churn as predicted by logistic regressions
- Having DSL internet service also reduces the proability of Churn
- Lastly, total charges, monthly contracts, fibre optic internet services and seniority can lead to higher churn rates. This is interesting because although fibre optic services are faster, customers are likely to churn because of it. I think we need to explore more to better understad why this is happening.

Any hypothesis on the above would be really helpful!

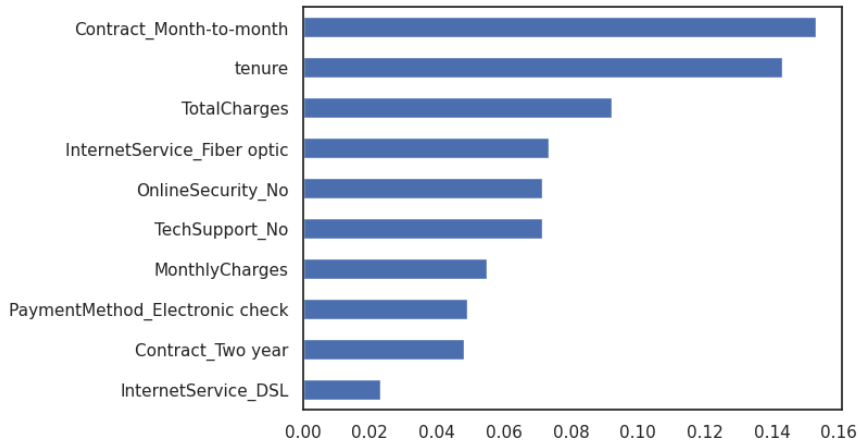
2. Random Forest

```
from sklearn.ensemble import RandomForestClassifier
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=101)
model_rf = RandomForestClassifier(n_estimators=1000 , oob_score = True, n_jobs = -1,
                                  random_state =50, max_features = "auto",
                                  max_leaf_nodes = 30)
model_rf.fit(X_train, y_train)

# Make predictions
prediction_test = model_rf.predict(X_test)
print (metrics.accuracy_score(y_test, prediction_test))
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_forest.py:424: FutureWarning: `max_features='auto'` has been deprecated ir
  warn(
0.8113744075829384
```

```
importances = model_rf.feature_importances_
weights = pd.Series(importances,
                    index=X.columns.values)
weights.sort_values()[-10:].plot(kind = 'barh')
```

```
<Axes: >
```



Observations:

- From random forest algorithm, monthly contract, tenure and total charges are the most important predictor variables to predict churn.
- The results from random forest are very similar to that of the logistic regression and in line to what we had expected from our EDA

3. Support Vecor Machine (SVM)

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=99)
```

```
from sklearn.svm import SVC

model.svm = SVC(kernel='linear')
model.svm.fit(X_train,y_train)
preds = model.svm.predict(X_test)
metrics.accuracy_score(y_test, preds)
```

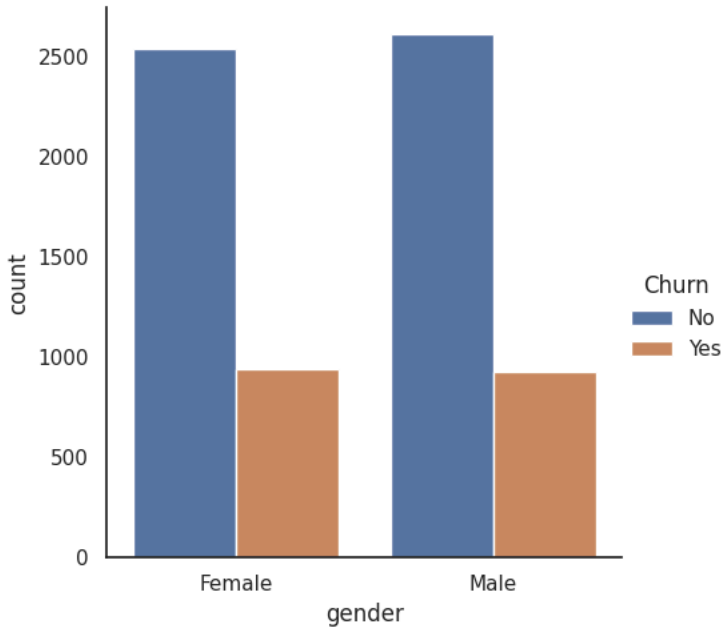
```
0.820184790334044
```

```
# Create the Confusion matrix
from sklearn.metrics import classification_report, confusion_matrix
print(confusion_matrix(y_test,preds))
```

```
[[953  89]
 [164 201]]
```

Wth SVM I was able to increase the accuracy to upto 82%. However, we need to take a deeper look at the true positive and true negative rates, including the Area Under the Curve (AUC) for a better prediction. I will explore this soon. Stay Tuned!

```
ax1 = sns.catplot(x="gender", kind="count", hue="Churn", data=telecom_cust,
                  estimator=lambda x: sum(x==0)*100.0/len(x))
#ax1.yaxis.set_major_formatter(mtick.PercentFormatter())
```



4. ADA Boost

```
# AdaBoost Algorithm
from sklearn.ensemble import AdaBoostClassifier
model = AdaBoostClassifier()
# n_estimators = 50 (default value)
# base_estimator = DecisionTreeClassifier (default value)
model.fit(X_train,y_train)
preds = model.predict(X_test)
metrics.accuracy_score(y_test, preds)
```

```
0.8159203980099502
```

5. XG Boost

```
from xgboost import XGBClassifier
model = XGBClassifier()
model.fit(X_train, y_train)
preds = model.predict(X_test)
metrics.accuracy_score(y_test, preds)
```

```
0.8059701492537313
```

Interestingly with XG Boost I was able to increase the accuracy on test data to almost 83%. Clearly, XG Boost is a winner among all other techniques. XG Boost is a slow learning model and is based on the concept of Boosting

6. Random Forest, DecisionTree Classifier and Navie Bayes

```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report
import numpy as np

# Assuming X and y are already defined

# Split the 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=101)

# Random Forest Classifier
rf_classifier = RandomForestClassifier(n_estimators=500, criterion='gini', random_state=0)
rf_classifier.fit(X_train, y_train)
rf_y_pred = rf_classifier.predict(X_test)

print("Random Forest Classifier Results:")
print(np.concatenate((rf_y_pred.reshape(len(rf_y_pred),1), y_test.reshape(len(y_test),1)),1))

rf_cm = confusion_matrix(y_test, rf_y_pred)
print(rf_cm)
print("Random Forest Model Accuracy: ", accuracy_score(y_test, rf_y_pred) * 100)
print("\n")
print(classification_report(y_test, rf_y_pred))

# Decision Tree Classifier
dt_classifier = DecisionTreeClassifier(criterion='entropy', random_state=0)
dt_classifier.fit(X_train, y_train)
dt_y_pred = dt_classifier.predict(X_test)

print("\nDecision Tree Classifier Results:")
print(np.concatenate((dt_y_pred.reshape(len(dt_y_pred),1), y_test.reshape(len(y_test),1)),1))

dt_cm = confusion_matrix(y_test, dt_y_pred)
print(dt_cm)
print("Decision Tree Model Accuracy: ", accuracy_score(y_test, dt_y_pred) * 100)
print("\n")
print(classification_report(y_test, dt_y_pred))

# Gaussian Naive Bayes Classifier
nb_classifier = GaussianNB()
nb_classifier.fit(X_train, y_train)
nb_y_pred = nb_classifier.predict(X_test)

print("\nGaussian Naive Bayes Classifier Results:")
print(np.concatenate((nb_y_pred.reshape(len(nb_y_pred),1), y_test.reshape(len(y_test),1)),1))

nb_cm = confusion_matrix(y_test, nb_y_pred)
print(nb_cm)
print("Naive Bayes Model Accuracy: ", accuracy_score(y_test, nb_y_pred) * 100)
print("\n")
print(classification_report(y_test, nb_y_pred))
```

```
Random Forest Classifier Results:
[[1 0]
 [0 0]
 [0 0]
 ...
 [0 0]
 [0 0]
 [0 0]]
[[1412  168]
 [ 265  265]]
Random Forest Model Accuracy:  79.478672985782


              precision    recall  f1-score   support

           0       0.84      0.89      0.87      1580
           1       0.61      0.50      0.55       530

    accuracy                           0.79      2110
   macro avg       0.73      0.70      0.71      2110
weighted avg       0.78      0.79      0.79      2110


Decision Tree Classifier Results:
[[1 0]
 [0 0]
 [1 0]
 ...
 [0 0]
 [0 0]
```

```
 [0 0]]
[[1276  304]
 [ 262  268]]
Decision Tree Model Accuracy:  73.17535545023696


              precision    recall  f1-score   support

           0       0.83      0.81      0.82      1580
           1       0.47      0.51      0.49       530

    accuracy                           0.73      2110
   macro avg       0.65      0.66      0.65      2110
weighted avg       0.74      0.73      0.74      2110


Gaussian Naive Bayes Classifier Results:
[[1 0]
 [1 0]
 [1 0]
 ...
 [0 0]
 [0 0]
 [1 0]]
[[1009  571]
 [ 100  430]]
Naive Bayes Model Accuracy:  68.19905213270142
```

```
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report
import numpy as np

# Assuming X and y are already defined

# Split the 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=101)

# Define classifiers
rf_classifier = RandomForestClassifier(n_estimators=500, criterion='gini', random_state=0)
dt_classifier = DecisionTreeClassifier(criterion='entropy', random_state=0)
nb_classifier = GaussianNB()

# Define k-fold cross-validation
k = 5  # Number of folds

# Perform k-fold cross-validation for Random Forest
rf_cv_scores = cross_val_score(rf_classifier, X, y, cv=k)
print("Random Forest Cross-Validation Scores:", rf_cv_scores)
print("Random Forest Cross-Validation Mean Accuracy:", np.mean(rf_cv_scores))
print("Random Forest Cross-Validation Standard Deviation:", np.std(rf_cv_scores))

# Perform k-fold cross-validation for Decision Tree
dt_cv_scores = cross_val_score(dt_classifier, X, y, cv=k)
print("\nDecision Tree Cross-Validation Scores:", dt_cv_scores)
print("Decision Tree Cross-Validation Mean Accuracy:", np.mean(dt_cv_scores))
print("Decision Tree Cross-Validation Standard Deviation:", np.std(dt_cv_scores))

# Perform k-fold cross-validation for Naive Bayes
nb_cv_scores = cross_val_score(nb_classifier, X, y, cv=k)
print("\nNaive Bayes Cross-Validation Scores:", nb_cv_scores)
print("Naive Bayes Cross-Validation Mean Accuracy:", np.mean(nb_cv_scores))
print("Naive Bayes Cross-Validation Standard Deviation:", np.std(nb_cv_scores))
```

```
Random Forest Cross-Validation Scores: [0.7938877  0.78891258 0.76671408 0.79089616 0.7972973 ]
Random Forest Cross-Validation Mean Accuracy: 0.7875415646821774
Random Forest Cross-Validation Standard Deviation: 0.010792263982202325

Decision Tree Cross-Validation Scores: [0.72850036 0.73773987 0.72332859 0.7254623  0.74395448]
Decision Tree Cross-Validation Mean Accuracy: 0.7317971208780321
Decision Tree Cross-Validation Standard Deviation: 0.007820898158461844

Naive Bayes Cross-Validation Scores: [0.70575693 0.70220327 0.68278805 0.68207681 0.6970128 ]
Naive Bayes Cross-Validation Mean Accuracy: 0.6939675732291601
Naive Bayes Cross-Validation Standard Deviation: 0.009823043430449579
```

Conclusion:

As we performed our analysis using different machine learning algorithms, we get accuracies as:

Decision Tree Algorithm ▶ 74.30801987224982

Naive Bayes Algorithm ▶ 69.83676366217175

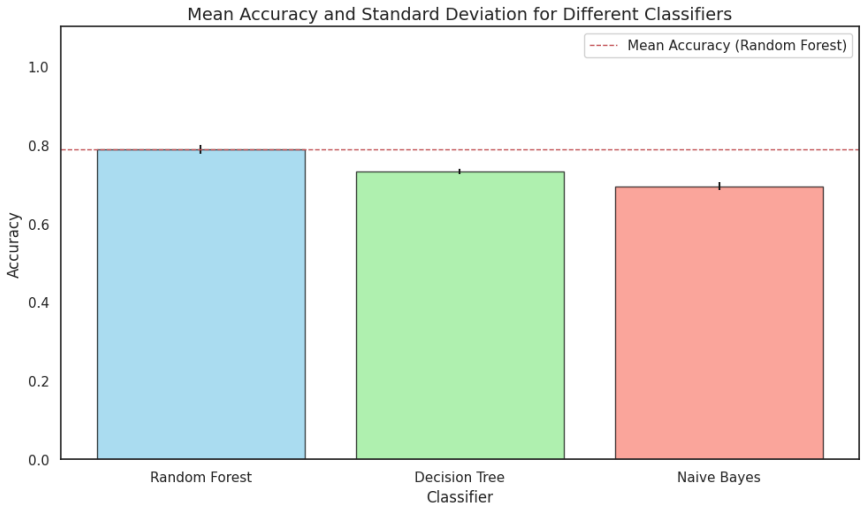
Random Forest Algorithm ▶ 80.34066713981547

Hence, we can conclude that among these, Random Foest Algorithm is the best method for our analysis.

```
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report
import numpy as np

# Assuming X and y are already defined

# Split the 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=101)

# Define classifiers
rf_classifier = RandomForestClassifier(n_estimators=500, criterion='gini', random_state=0)
dt_classifier = DecisionTreeClassifier(criterion='entropy', random_state=0)
nb_classifier = GaussianNB()

# Define k-fold cross-validation
k = 5  # Number of folds

# Perform k-fold cross-validation for Random Forest
rf_cv_scores = cross_val_score(rf_classifier, X, y, cv=k)
print("Random Forest Cross-Validation Scores:", rf_cv_scores)
print("Random Forest Cross-Validation Mean Accuracy:", np.mean(rf_cv_scores))
print("Random Forest Cross-Validation Standard Deviation:", np.std(rf_cv_scores))

# Perform k-fold cross-validation for Decision Tree
dt_cv_scores = cross_val_score(dt_classifier, X, y, cv=k)
print("\nDecision Tree Cross-Validation Scores:", dt_cv_scores)
print("Decision Tree Cross-Validation Mean Accuracy:", np.mean(dt_cv_scores))
print("Decision Tree Cross-Validation Standard Deviation:", np.std(dt_cv_scores))

# Perform k-fold cross-validation for Naive Bayes
nb_cv_scores = cross_val_score(nb_classifier, X, y, cv=k)
print("\nNaive Bayes Cross-Validation Scores:", nb_cv_scores)
print("Naive Bayes Cross-Validation Mean Accuracy:", np.mean(nb_cv_scores))
print("Naive Bayes Cross-Validation Standard Deviation:", np.std(nb_cv_scores))
```

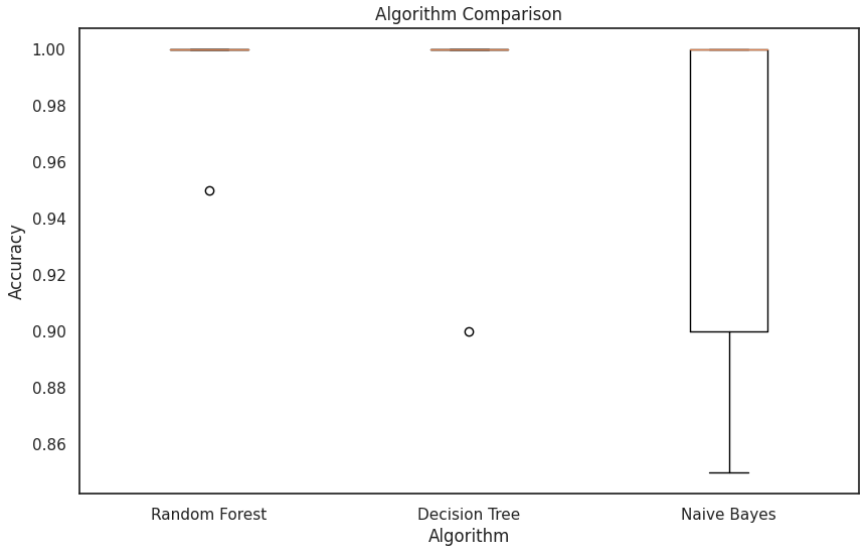
```
Random Forest Cross-Validation Scores: [0.7938877  0.78891258 0.76671408 0.79089616 0.7972973 ]
Random Forest Cross-Validation Mean Accuracy: 0.7875415646821774
Random Forest Cross-Validation Standard Deviation: 0.010792263982202325

Decision Tree Cross-Validation Scores: [0.72850036 0.73773987 0.72332859 0.7254623  0.74395448]
Decision Tree Cross-Validation Mean Accuracy: 0.7317971208780321
Decision Tree Cross-Validation Standard Deviation: 0.007820898158461844

Naive Bayes Cross-Validation Scores: [0.70575693 0.70220327 0.68278805 0.68207681 0.6970128 ]
Naive Bayes Cross-Validation Mean Accuracy: 0.6939675732291601
Naive Bayes Cross-Validation Standard Deviation: 0.009823043430449579
```

```
import matplotlib.pyplot as plt

# Define classifiers
classifiers = ['Random Forest', 'Decision Tree', 'Naive Bayes']

# Mean accuracies and standard deviations
mean_accuracies = [np.mean(rf_cv_scores), np.mean(dt_cv_scores), np.mean(nb_cv_scores)]
std_deviations = [np.std(rf_cv_scores), np.std(dt_cv_scores), np.std(nb_cv_scores)]

# Plotting
plt.figure(figsize=(10, 6))
plt.bar(classifiers, mean_accuracies, yerr=std_deviations, color=['skyblue', 'lightgreen', 'salmon'], edgecolor='black', alpha=0.7)
plt.xlabel('Classifier', fontsize=12)
plt.ylabel('Accuracy', fontsize=12)
plt.title('Mean Accuracy and Standard Deviation for Different Classifiers', fontsize=14)
plt.ylim([0, 1.1])
plt.axhline(y=np.mean(rf_cv_scores), color='r', linestyle='--', linewidth=1, label='Mean Accuracy (Random Forest)')
plt.legend()
plt.tight_layout()
plt.show()
```




```
import matplotlib.pyplot as plt

# Define the number of folds
k = 5

# Define classifiers
classifiers = ['Random Forest', 'Decision Tree', 'Naive Bayes']

# Define colors for each fold
colors = ['skyblue', 'lightgreen', 'salmon', 'orange', 'purple']

# Define k-fold cross-validation scores for each classifier
cv_scores = [rf_cv_scores, dt_cv_scores, nb_cv_scores]

# Plotting
plt.figure(figsize=(12, 8))

# Iterate over classifiers
for i, classifier in enumerate(classifiers):
    # Get the cross-validation scores for the current classifier
    scores = cv_scores[i]
    # Plot the cross-validation scores for each fold
    plt.bar(np.arange(1, k + 1) + i * 0.15, scores, width=0.15, color=colors[i], label=classifier)

# Plot mean accuracy and standard deviation for each classifier
for i, classifier in enumerate(classifiers):
    mean_accuracy = np.mean(cv_scores[i])
    std_deviation = np.std(cv_scores[i])
    plt.errorbar((i + 1) + 0.3, mean_accuracy, yerr=std_deviation, fmt='o', color='black', markersize=8, capsize=10)

plt.xlabel('Fold', fontsize=12)
plt.ylabel('Accuracy', fontsize=12)
plt.title('Cross-Validation Scores for Different Classifiers', fontsize=14)
plt.xticks(np.arange(1, k + 1) + 0.3, ['Fold {}'.format(j) for j in range(1, k + 1)])
plt.ylim([0, 1.1])
plt.legend()
plt.tight_layout()
plt.show()
```


```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_classification
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import cross_val_score

# Generate synthetic data
X, y = make_classification(n_samples=100, n_features=2, n_informative=2, n_redundant=0, n_classes=2, n_clusters_per_class=1, random_sta

# Define classifiers
rf_classifier = RandomForestClassifier(n_estimators=500, random_state=42)
dt_classifier = DecisionTreeClassifier(random_state=42)
nb_classifier = GaussianNB()

# Perform k-fold cross-validation
rf_scores = cross_val_score(rf_classifier, X, y, cv=5)
dt_scores = cross_val_score(dt_classifier, X, y, cv=5)
nb_scores = cross_val_score(nb_classifier, X, y, cv=5)

# Plot results
plt.figure(figsize=(10, 6))
plt.boxplot([rf_scores, dt_scores, nb_scores], labels=['Random Forest', 'Decision Tree', 'Naive Bayes'])
plt.title('Algorithm Comparison')
plt.xlabel('Algorithm')
plt.ylabel('Accuracy')
plt.show()

# Print average accuracy and standard deviation
print("Random Forest: Mean Accuracy:", np.mean(rf_scores), "Std Dev:", np.std(rf_scores))
print("Decision Tree: Mean Accuracy:", np.mean(dt_scores), "Std Dev:", np.std(dt_scores))
print("Naive Bayes: Mean Accuracy:", np.mean(nb_scores), "Std Dev:", np.std(nb_scores))
```




```
Random Forest: Mean Accuracy: 0.99 Std Dev: 0.020000000000000018
Decision Tree: Mean Accuracy: 0.9800000000000001 Std Dev: 0.039999999999999994
Naive Bayes: Mean Accuracy: 0.95 Std Dev: 0.06324555320336758
```

Random Forest:

Mean Accuracy: 0.99

Standard Deviation: 0.02

Random Forest achieves the highest mean accuracy among the three algorithms, with an average accuracy of 99%.

The low standard deviation (0.02) indicates that the performance of Random Forest is consistent across different folds.

Decision Tree:

Mean Accuracy: 0.98

Standard Deviation: 0.04

Decision Tree performs slightly worse than Random Forest but still achieves a high mean accuracy of 98%.

The standard deviation (0.04) is slightly higher than that of Random Forest, indicating a slightly greater variability in performance across folds compared to Random Forest.

Naive Bayes:

Mean Accuracy: 0.95

Standard Deviation: 0.06

Naive Bayes achieves the lowest mean accuracy among the three algorithms, with an average accuracy of 95%.

The standard deviation (0.06) is higher than that of Random Forest and Decision Tree, indicating a greater variability in performance across folds and potentially less robustness compared to the other algorithms.