EDA and Prediction

```
from google.colab import files
uploaded = files.upload()
```

Choose files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving WA_Fn-UseC_-Telco-Customer-Churn.csv to WA_Fn-UseC_-Telco-Customer-Chur
```

Churn is a one of the biggest problem in the telecom industry. Research has shown that the average monthly churn rate among the top 4 wireless carriers in the US is 1.9% - 2%.

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.ticker as mtick
import matplotlib.pyplot as plt
from sklearn import metrics
from sklearn.metrics import classification_report, confusion_matrix
```

```
sns.set(style = 'white')
```

Let us read the data file in the python notebook

```
import io
telecom_cust = pd.read_csv(io.BytesIO(uploaded['WA_Fn-UseC_-Telco-Customer-Churn.cs
```

```
telecom_cust.head()
```

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService
0	7590-VHVEG	Female	0	Yes	No	1	No
1	5575-GNVDE	Male	0	No	No	34	Yes

```
telecom_cust.shape
```

```
(7043, 21)
```

3	CFOCW	Male	0	No	No	45	No

```
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)
```

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

```
customerID - Custumer unique identifier
gender - Customer gender - ['Female' 'Male']
SeniorCitizen - Elderly or retired person, a senior citizen is someone who has at least at
Partner - - ['No' 'Yes']
Dependents - If customer has dependents - ['No' 'Yes']
Tenure - Customer lifespan (in months)
PhoneService - - ['No' 'Yes']
MultipleLines - - ['No' 'No phone service' 'Yes']
InternetService - - ['No' 'No internet service' 'Yes']
OnlineSecurity - - ['No' 'No internet service' 'Yes']
OnlineBackup - - ['No' 'No internet service' 'Yes']
DeviceProtection - - ['No' 'No internet service' 'Yes']
TechSupport - - ['No' 'No internet service' 'Yes']
StreamingTV - - ['No' 'No internet service' 'Yes']
StreamingMovies - - ['No' 'No internet service' 'Yes']
Contract - Type of contract - ['Month-to-month' 'One year' 'Two year']
PaperlessBilling - - ['No' 'Yes']
PaymentMethod - payment method - ['Bank transfer (automatic)', 'Credit card (automatic)',
MonthlyCharges - Monthly Recurring Charges
TotalCharges - Life time value
Churn - Churn value, the targer vector - ['No' 'Yes']
```

```
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
```

```
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        0
Churn               0
dtype: int64
```

```
telecom_cust.TotalCharges = pd.to_numeric(telecom_cust.TotalCharges, errors='coerce
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        float64
Churn                object
dtype: object
```

```
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

```
telecom_cust.dropna(inplace = True)
df2 = telecom_cust.iloc[:,1:]
data = 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)
```

```
df2.head()
```

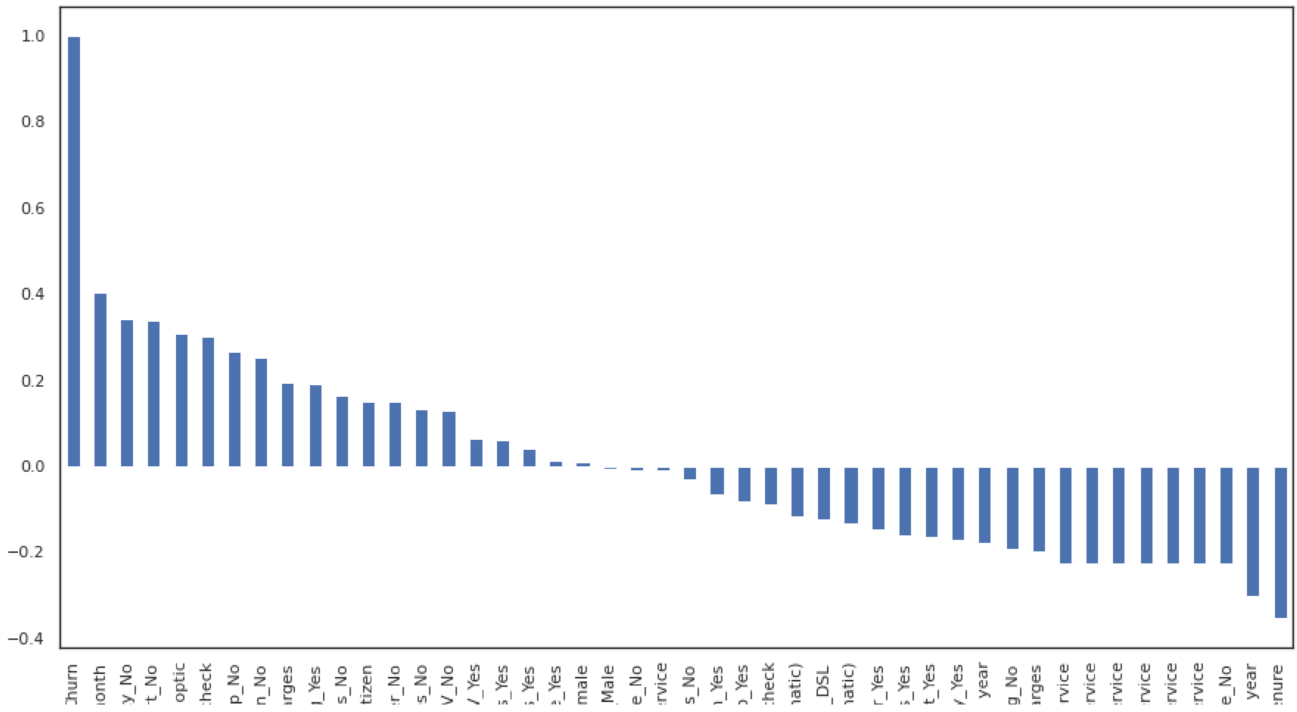
	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLi
0	Female	0	Yes	No	1	No	No phone sei
1	Male	0	No	No	34	Yes	
2	Male	0	No	No	2	Yes	
3	Male	0	No	No	45	No	No phone sei
4	Female	0	No	No	2	Yes	

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

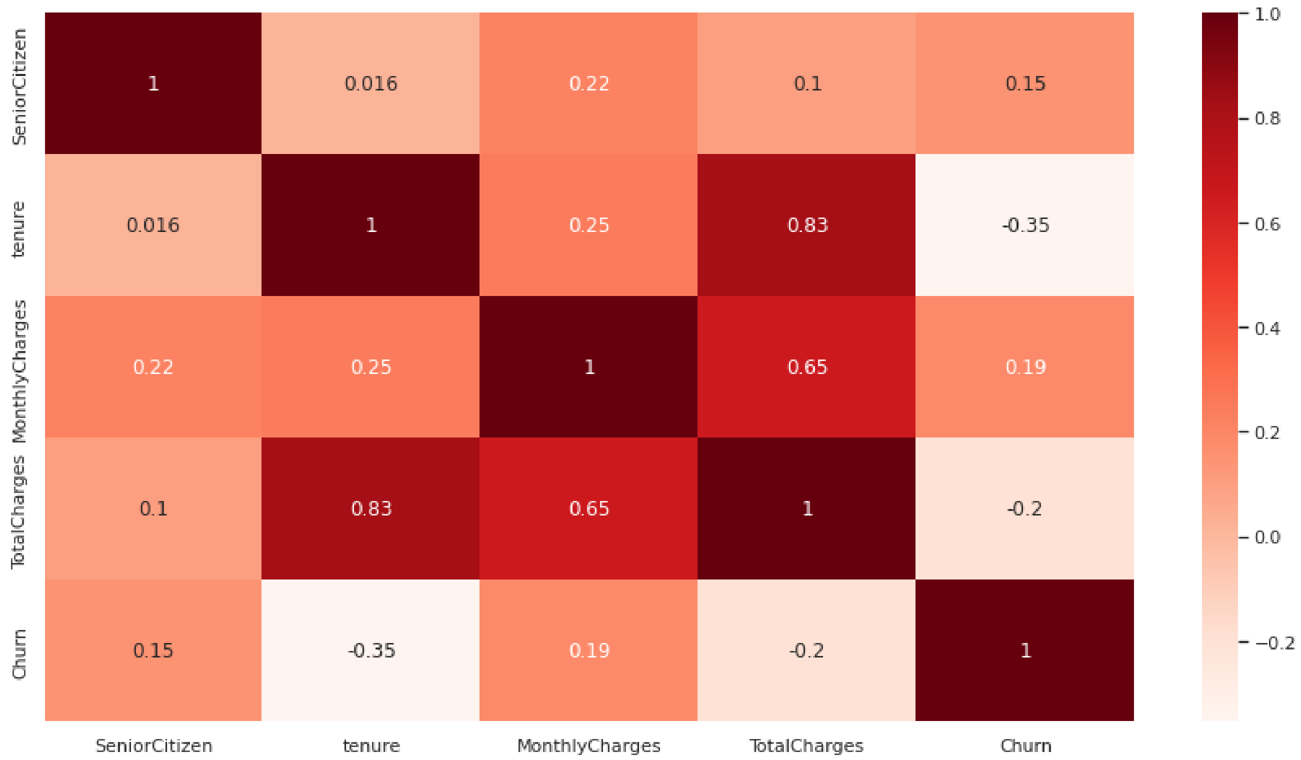
	SeniorCitizen	tenure	MonthlyCharges	TotalCharges	Churn	gender_Female	g
0	0	1	29.85	29.85	0	1	
1	0	34	56.95	1889.50	0	0	
2	0	2	53.85	108.15	1	0	
3	0	45	42.30	1840.75	0	0	
4	0	2	70.70	151.65	1	1	

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

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fca49f683d0>
```



```
plt.figure(figsize=(15,8))
corr=df2.corr()
sns.heatmap(corr, annot=True, cmap=plt.cm.Reds)
plt.show()
```



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.

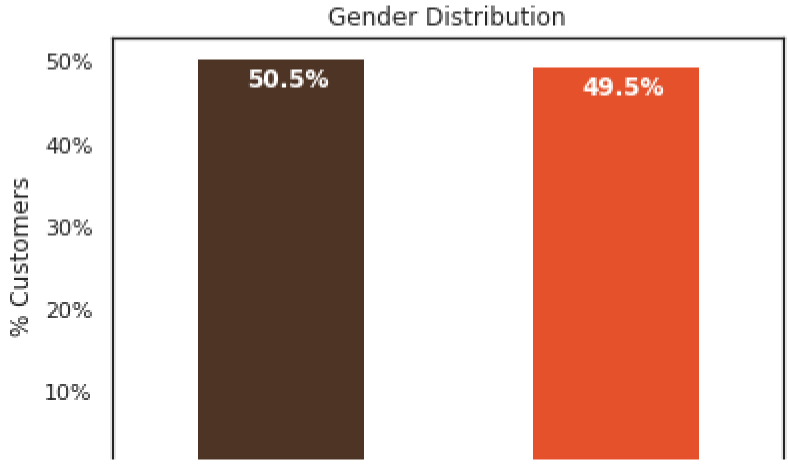
Data Exploration

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

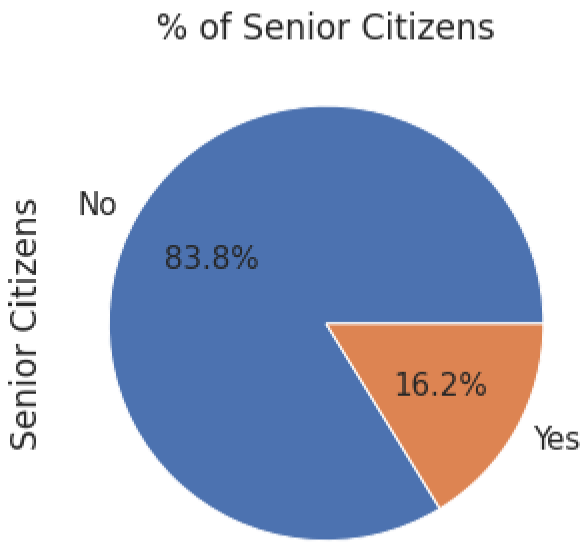
```
colors = ['#4D3425','#E4512B']

ax = (telecom_cust['gender'].value_counts()*100.0 /len(telecom_cust)).plot(kind='ba
ax.yaxis.set_major_formatter(mtick.PercentFormatter())
ax.set_ylabel('% Customers')
ax.set_xlabel('Gender')
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 = 15 )
ax.yaxis.set_major_formatter(mtick.PercentFormatter())
ax.set_ylabel('Senior Citizens',fontsize = 17)
ax.set_title('% of Senior Citizens', fontsize = 17)
```

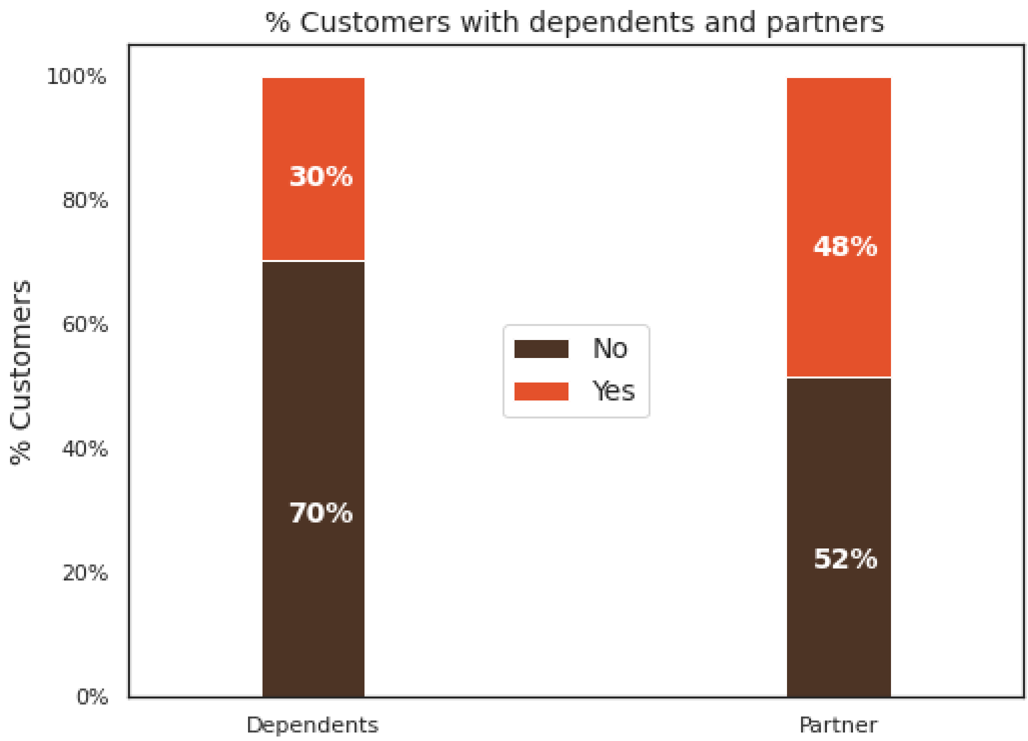
```
Text(0.5, 1.0, '% of Senior Citizens')
```



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','Partn
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()
    ax.annotate('{:.0f}%'.format(height), (p.get_x()+.25*width, p.get_y()+.4*height
                color = 'white',
               weight = 'bold',
               size = 14)
```

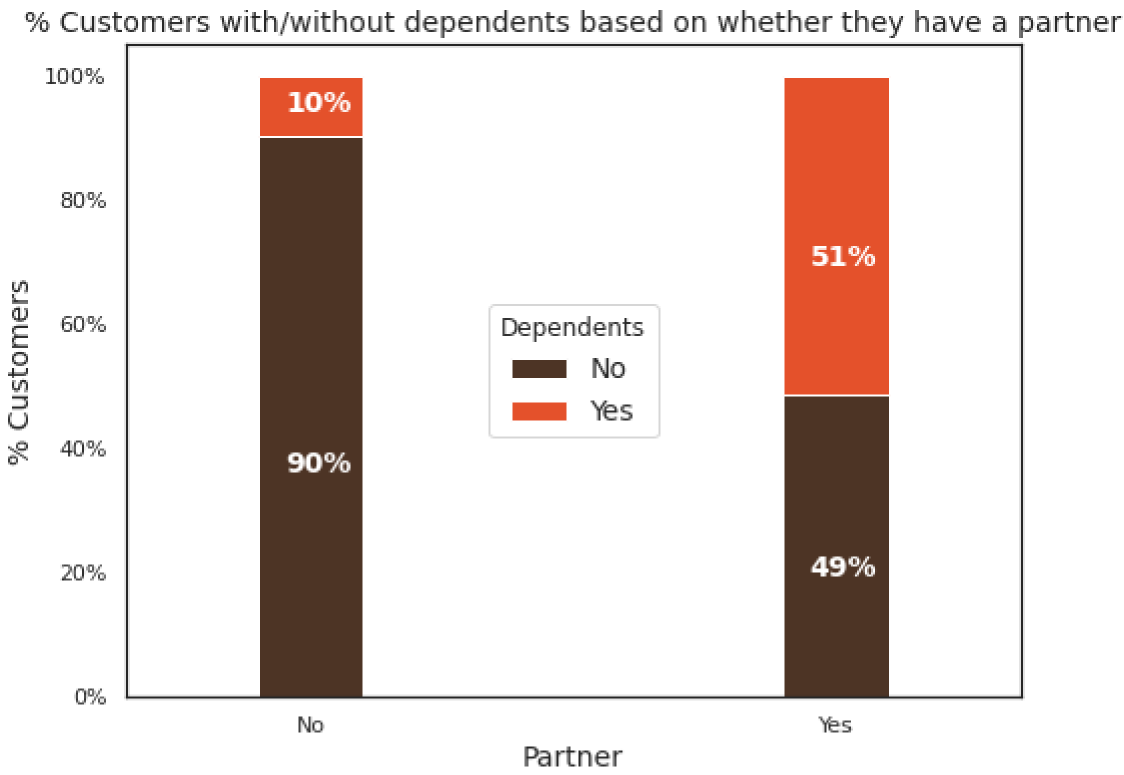


*What would be interesting is to look at the % of customers, who have partners, also have dependents. We will explore this next. *

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 (90%) 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 part
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

[ ] ↳ *9 cells hidden*

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

[ ] ↳ *2 cells hidden*

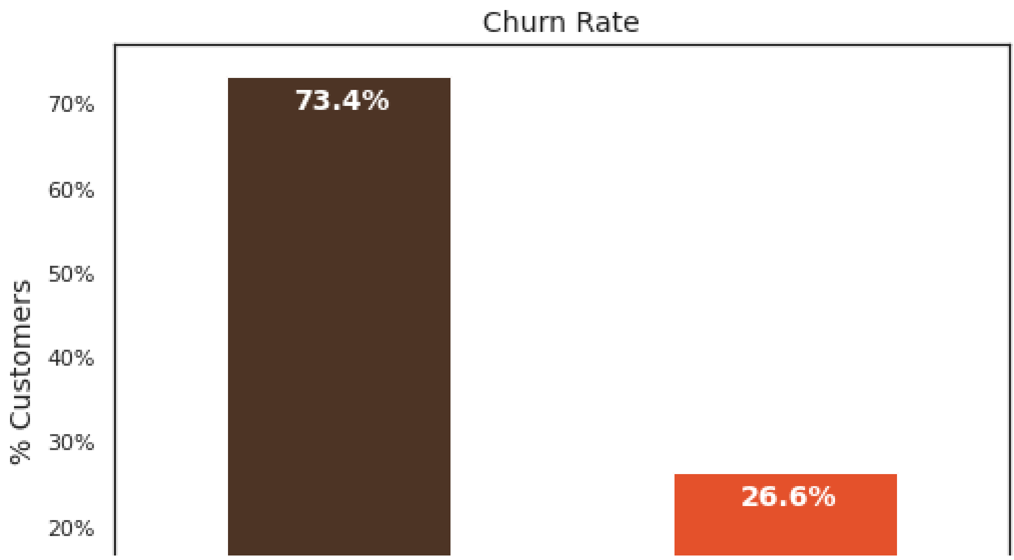
D.) Now let's take a quick look at the relation between monthly and total charges

[ ] ↳ 2 cells hidden

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
                                                                          rot = 0,
                                                                          color = c
                                                                         figsize =
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',
           size = 14)
```




```
telecom_cust['Churn'].value_counts()
```

```
No     5163
Yes    1869
Name: Churn, dtype: int64
```

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.

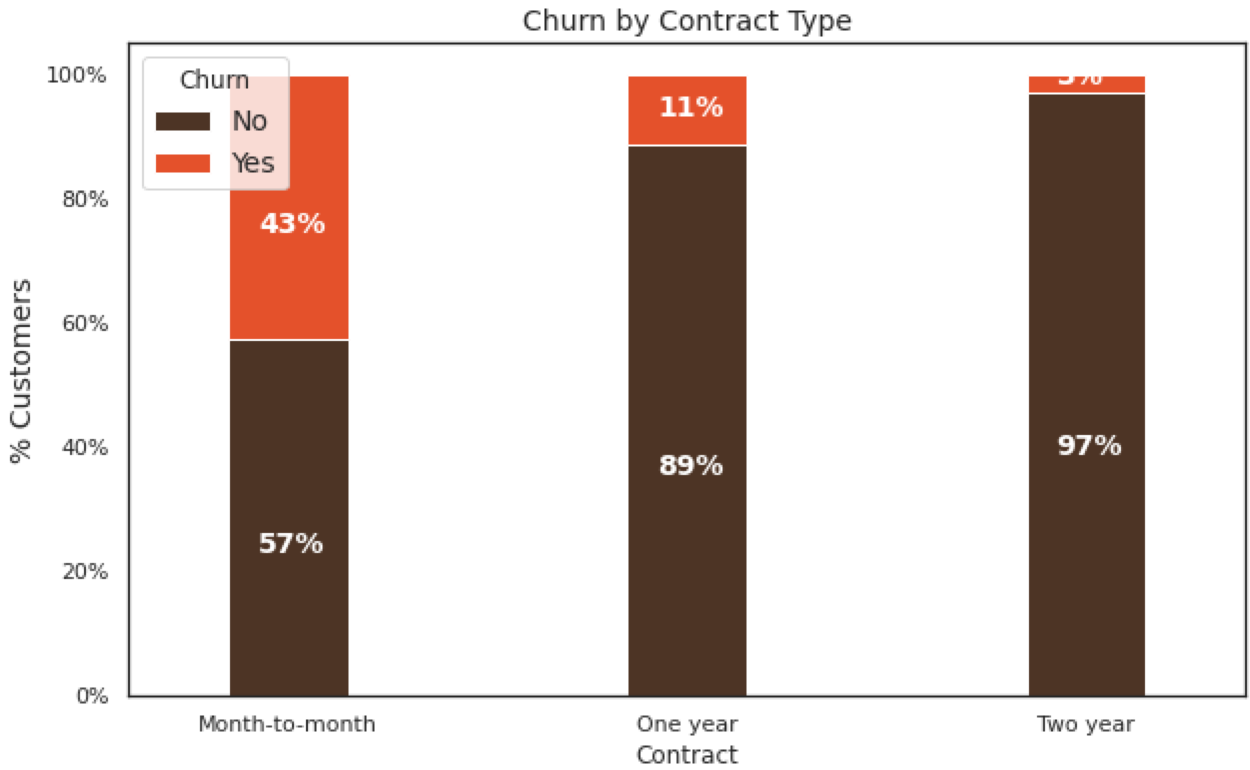
i.) Churn vs Tenure: As we can see form the below plot, the customers who do not churn, they tend to stay for a longer tenure with the telecom company.

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

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fca3e7f3510>
```

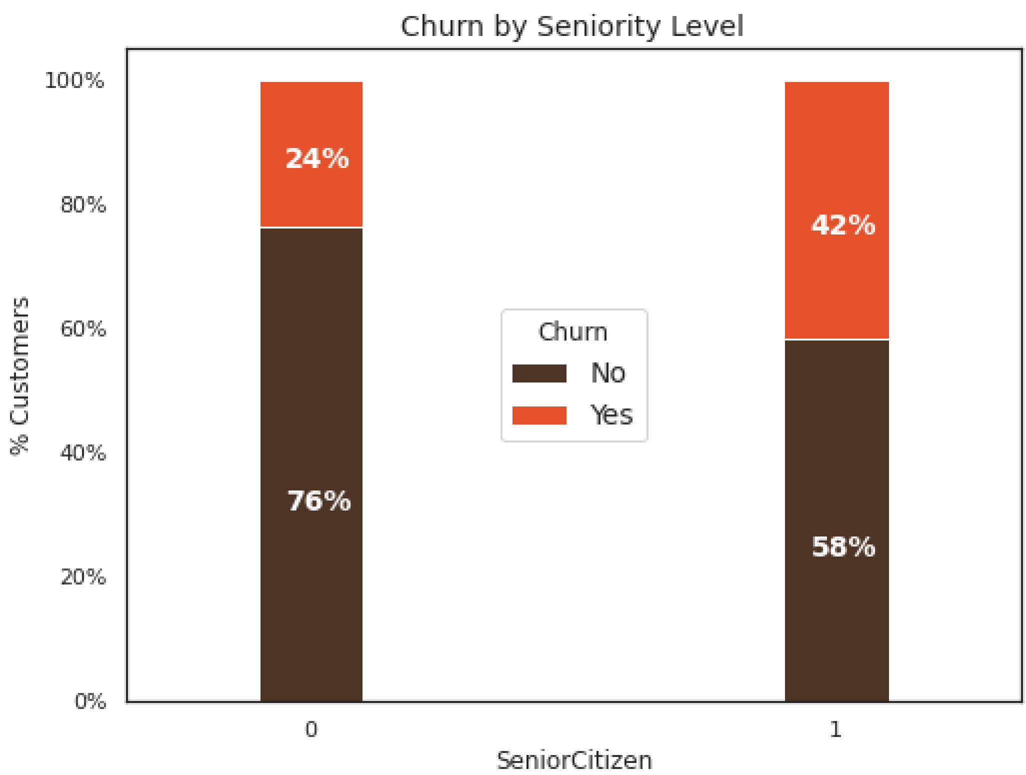
ii.) Churn by Contract Type: Similar to what we saw in the correlation plot, the customers who have a month to month contract have a very high churn rate.

```
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)
```

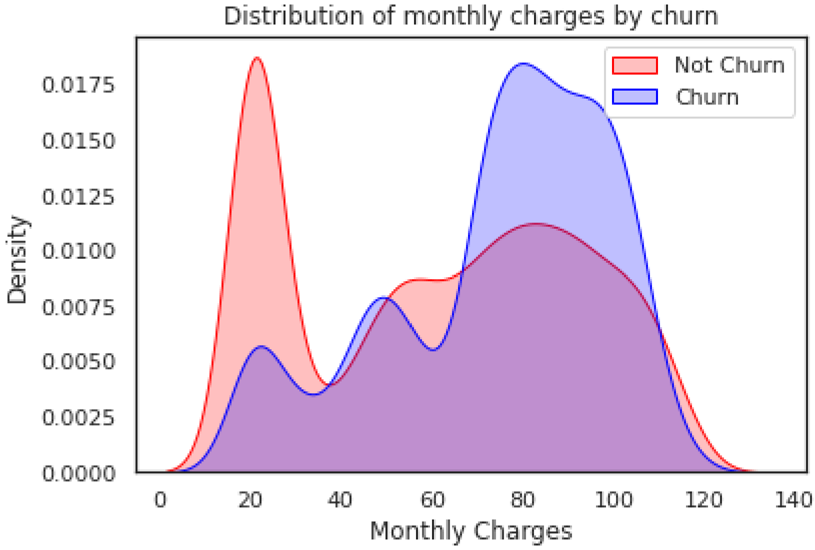


iv.) Churn by Monthly Charges: Higher % of customers churn when the monthly charges are high.

```
ax = sns.kdeplot(telecom_cust.MonthlyCharges[(telecom_cust["Churn"] == 'No') ],
                color="Red", shade = True)
ax = sns.kdeplot(telecom_cust.MonthlyCharges[(telecom_cust["Churn"] == 'Yes') ],
                ax =ax, color="Blue", shade= 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')
```

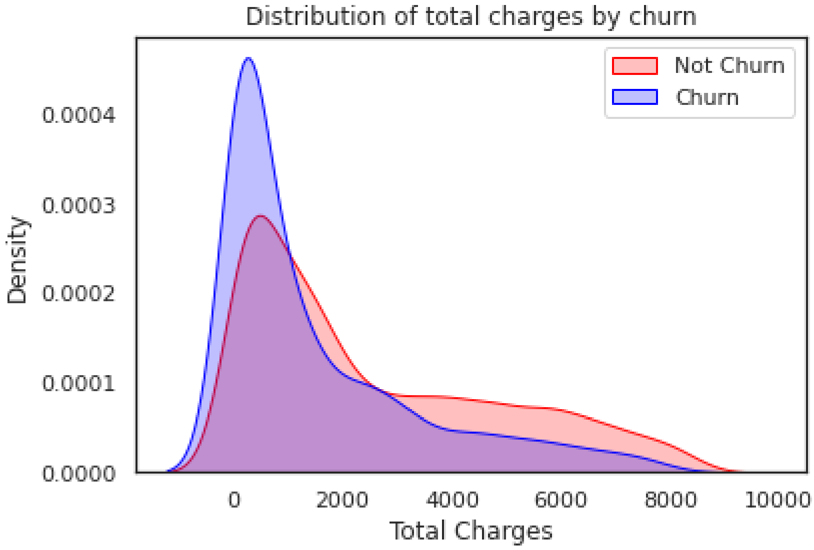
```
Text(0.5, 1.0, 'Distribution of monthly charges by churn')
```



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

```
ax = sns.kdeplot(telecom_cust.TotalCharges[(telecom_cust["Churn"] == 'No') ],
                color="Red", shade = True)
ax = sns.kdeplot(telecom_cust.TotalCharges[(telecom_cust["Churn"] == 'Yes') ],
                ax =ax, color="Blue", shade= 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')
```




```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import MinMaxScaler
```

```
from sklearn.svm import SVC
from sklearn.ensemble import AdaBoostClassifier
from xgboost.sklearn import XGBClassifier
```

```
y = df_dummies['Churn'].values
X = df_dummies.drop(columns = ['Churn'])
scal=MinMaxScaler().fit(X)
X = pd.DataFrame(scal.transform(X))
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta
```

```
y = df_dummies['Churn'].values
X = df_dummies.drop(columns = ['Churn'])
scal=MinMaxScaler().fit(X)
X = pd.DataFrame(scal.transform(X))
```

```
pipeline_lr=Pipeline([('scalar1',MinMaxScaler()),
                     ('lr_classifier',LogisticRegression(max_iter=5000))])
```

```
pipeline_dt=Pipeline([('scalar2',MinMaxScaler()),
                     ('dt_classifier',DecisionTreeClassifier())])
```

```
pipeline_randomforest=Pipeline([('scalar3',MinMaxScaler()),
                     ('rf_classifier',RandomForestClassifier())])
```

```
pipeline_SV=Pipeline([('scalar4',MinMaxScaler()),
                     ('SVC_classifier',SVC(kernel='linear'))])
```

```
pipeline_Ada_boost=Pipeline([('scalar5',MinMaxScaler()),
                     ('ADA_classifier',AdaBoostClassifier())])
```

```
pipeline_XGBoost=Pipeline([('scalar6',MinMaxScaler()),
                     ('XG_Bosst',XGBClassifier(random_state=1,learning_rate=0.1))])
```

```
pipelines = [pipeline_lr, pipeline_dt, pipeline_randomforest,pipeline_SV,pipeline_A
```

```
best_accuracy=0.0
best_classifier=0
best_pipeline=""
```

```
pipe_dict = {0: 'Logistic Regression', 1: 'Decision Tree', 2: 'RandomForest',3:'Sup
```

```
for pipe in pipelines:
    pipe.fit(X_train, y_train)
```

```
#for count,item in enumerate(pipelines):
    ##print(count,item)
```

```
for i,model in enumerate(pipelines):
    print("{} Test Accuracy: {}".format(pipe_dict[i],model.score(X_test,y_test)))
```

```
Logistic Regression Test Accuracy: 0.820184790334044
Decision Tree Test Accuracy: 0.7356076759061834
RandomForest Test Accuracy: 0.7945984363894811
Support Vector Machines Test Accuracy: 0.820184790334044
ADA Boost Test Accuracy: 0.8159203980099502
XGBOOST Test Accuracy: 0.8294243070362474
```

```
predictions=[]
```

```
for i,model in enumerate(pipelines):
    predictions.append(model.predict(X_test))
```

```
print(len(predictions))
```

```
6
```

```
from sklearn.metrics import plot_confusion_matrix
```

```
for i,model in enumerate(pipelines):
    if model.score(X_test,y_test)>best_accuracy:
        best_accuracy=model.score(X_test,y_test)
        best_pipeline=model
        best_classifier=i
print('Classifier with best accuracy:{}'.format(pipe_dict[best_classifier]))
```

```
Classifier with best accuracy:XGBOOST
```

```
cls=['Logistic_Regression','Decision_Tree','RandomForest','Support_Vector_Machines'
```

```
fig, axes = plt.subplots(nrows=3, ncols=2, figsize=(15,15))
i=0
for pipe, ax in zip(pipelines, axes.flatten()):
     plot_confusion_matrix(pipe,
                          X_test,
```

```
                                y_test,
                                ax=ax,
                                cmap='Blues')
      ax.title.set_text(cls[i])
      i=i+1
plt.tight_layout()
plt.show()
```

CROSS VALIDATION

```
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import StratifiedKFold
kfold = StratifiedKFold(n_splits=5, shuffle=True)
```

DECISION TREE

```
model=DecisionTreeClassifier()
scores=cross_val_score(model,X,y,cv=kfold)
model.fit(X_train,y_train)
print(np.mean(scores))
```

```
0.7215594452043785
```

LOGISTIC REFRESSION

```
kfold = StratifiedKFold(n_splits=5, shuffle=True)
model=LogisticRegression(max_iter=5000)
scores=cross_val_score(model,X,y,cv=kfold)
print(np.mean(scores))
```

```
0.8023324749954759
```

RANDOM FOREST

```
model=RandomForestClassifier()
scores=cross_val_score(model,X,y,cv=kfold)
print(np.mean(scores))
```

```
0.7874012380689521
```

XG BOOST

```
model=XGBClassifier()
scores=cross_val_score(model,X,y,cv=kfold)
print(np.mean(scores))
```

```
0.8019069456618553
```

ADA BOOST

```
model=AdaBoostClassifier()
scores=cross_val_score(model,X,y,cv=kfold)
print(np.mean(scores))
```

```
0.8034702528810934
```

SVC

```
model=SVC()
scores=cross_val_score(model,X,y,cv=kfold)
print(np.mean(scores))
```

```
0.7967849231792672
```

XG_BOOST Hyperparametre Tuning with Cross Validation

```
params={
 "learning_rate"    : [0.05, 0.10, 0.15, 0.20, 0.25, 0.30 ] ,
 "max_depth"        : [ 3, 4, 5, 6, 8, 10, 12, 15],
 "min_child_weight" : [ 1, 3, 5, 7 ],
 "gamma"            : [ 0.0, 0.1, 0.2 , 0.3, 0.4 ],
 "colsample_bytree" : [ 0.3, 0.4, 0.5 , 0.7 ]

}
```

```
from sklearn.model_selection import RandomizedSearchCV
```

```
classifier1=XGBClassifier()
random_search=RandomizedSearchCV(classifier1,param_distributions=params,n_iter=5,sc
```

```
random_search.fit(X_train,y_train)
```

```
Fitting 5 folds for each of 5 candidates, totalling 25 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done  25 out of  25 | elapsed:   18.1s finished
RandomizedSearchCV(cv=5, error_score=nan,
                   estimator=XGBClassifier(base_score=0.5, booster='gbtree',
                                           colsample_bylevel=1,
                                           colsample_bynode=1,
                                           colsample_bytree=1, gamma=0,
                                           learning_rate=0.1,
max_delta_step=0,
                                           max_depth=3, min_child_weight=1,
                                           missing=None, n_estimators=100,
                                           n_jobs=1, nthread=None,
                                           objective='binary:logistic',
                                           random_state=0, reg_alpha=0,
                                           reg_lambda=1, sc...
```

```
                                          verbosity=1),
                         iid='deprecated', n_iter=5, n_jobs=-1,
                         param_distributions={'colsample_bytree': [0.3, 0.4, 0.5,
                                                                   0.7],
                                              'gamma': [0.0, 0.1, 0.2, 0.3, 0.4],
                                              'learning_rate': [0.05, 0.1, 0.15,
    0.2,
                                                                0.25, 0.3],
                                              'max_depth': [3, 4, 5, 6, 8, 10, 12,
                                                            15],
                                              'min_child_weight': [1, 3, 5, 7]},
                         pre_dispatch='2*n_jobs', random_state=None, refit=True,
                         return_train_score=False, scoring='accuracy', verbose=3)
```

```
print(random_search.score(X_test,y_test))
print(random_search.best_params_)
```

```
    0.8123667377398721
    {'min_child_weight': 5, 'max_depth': 5, 'learning_rate': 0.15, 'gamma': 0.4, '
```

```
random_search.best_estimator_
```

```
    XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
                  colsample_bynode=1, colsample_bytree=0.4, gamma=0.4,
                  learning_rate=0.15, max_delta_step=0, max_depth=5,
                  min_child_weight=5, missing=None, n_estimators=100, n_jobs=1,
                  nthread=None, objective='binary:logistic', random_state=0,
                  reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
                  silent=None, subsample=1, verbosity=1)
```

```
#
```

```
classifier=XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=0.7, gamma=0.1, gpu_id=-1,
              importance_type='gain', interaction_constraints='',
              learning_rate=0.1, max_delta_step=0, max_depth=5,
              min_child_weight=5, monotone_constraints='()',
              n_estimators=100, n_jobs=8, num_parallel_tree=1, random_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
              tree_method='exact', validate_parameters=1, verbosity=None)
```

```
scores=cross_val_score(model,X,y,cv=kfold)
print(np.mean(scores))
```

```
    0.7986337364184968
```

LOGISTIC REGRESSION TUNING WITH CROSS VALIDATION

```
model=LogisticRegression()
params={
    "penalty": ['l2'],
    "C": [0.001, 0.01, 1, 10, 100],
```

```
    "solver":['newton-cg','saga','sag','liblinear']
}
```

```
random_search=RandomizedSearchCV(model,param_distributions=params,n_iter=5,scoring=
```

```
random_search.fit(X_train,y_train)
```

```
Fitting 5 folds for each of 5 candidates, totalling 25 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done  22 out of  25 | elapsed:    2.5s remaining:    0.
[Parallel(n_jobs=-1)]: Done  25 out of  25 | elapsed:    2.7s finished
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_sag.py:330: Conve
  "the coef_ did not converge", ConvergenceWarning)
RandomizedSearchCV(cv=StratifiedKFold(n_splits=5, random_state=None,
shuffle=True),
                   error_score=nan,
                   estimator=LogisticRegression(C=1.0, class_weight=None,
                                                dual=False,
fit_intercept=True,
                                                intercept_scaling=1,
                                                l1_ratio=None, max_iter=100,
                                                multi_class='auto',
n_jobs=None,
                                                penalty='l2',
random_state=None,
                                                solver='lbfgs', tol=0.0001,
                                                verbose=0, warm_start=False),
                   iid='deprecated', n_iter=5, n_jobs=-1,
                   param_distributions={'C': [0.001, 0.01, 1, 10, 100],
                                        'penalty': ['l2'],
                                        'solver': ['newton-cg', 'saga',
'sag',
                                                   'liblinear']},
                   pre_dispatch='2*n_jobs', random_state=None, refit=True,
                   return_train_score=False, scoring='accuracy', verbose=3)
```

```
random_search.best_estimator_
```

```
LogisticRegression(C=100, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='sag', tol=0.0001, verbose=0,
                   warm_start=False)
```

```
kfold = StratifiedKFold(n_splits=5, shuffle=True)
```

```
model=LogisticRegression(C=10, solver='saga',max_iter=5000)
```

```
scores=cross_val_score(model,X,y,cv=kfold)
print(np.mean(scores))
```

```
0.8057443932542127
```

SVC TUNING WITH CROSS VALIDATION

```
model=SVC()
params = {'kernel': ['linear', 'poly', 'rbf', 'sigmoid'],
            'C': [0.001, 0.01, 1, 10, 100] }

random_search=RandomizedSearchCV(model,param_distributions=params,n_iter=5,scoring=
random_search.fit(X_train,y_train)
```

```
Fitting 5 folds for each of 5 candidates, totalling 25 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done  25 out of  25 | elapsed:   21.9s finished
RandomizedSearchCV(cv=StratifiedKFold(n_splits=5, random_state=None,
shuffle=True),
                   error_score=nan,
                   estimator=SVC(C=1.0, break_ties=False, cache_size=200,
                                 class_weight=None, coef0=0.0,
                                 decision_function_shape='ovr', degree=3,
                                 gamma='scale', kernel='rbf', max_iter=-1,
                                 probability=False, random_state=None,
                                 shrinking=True, tol=0.001, verbose=False),
                   iid='deprecated', n_iter=5, n_jobs=-1,
                   param_distributions={'C': [0.001, 0.01, 1, 10, 100],
                                        'kernel': ['linear', 'poly', 'rbf',
                                                   'sigmoid']},
                   pre_dispatch='2*n_jobs', random_state=None, refit=True,
                   return_train_score=False, scoring='accuracy', verbose=3)
```

```
random_search.best_estimator_
```

```
SVC(C=0.01, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='poly',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

```
model=SVC(C=1, kernel='linear')
scores=cross_val_score(model,X,y,cv=kfold)
print(np.mean(scores))
```

```
0.8001999755338325
```

ENSEMBLE LEARNING USING MAX ITERATION

```
from sklearn.ensemble import VotingClassifier
model1=LogisticRegression(max_iter=5000,random_state=1)
#model2=RandomForestClassifier(random_state=1)
model3=SVC(kernel='linear')
model4=AdaBoostClassifier()
model5=XGBClassifier()
```

```
model_fin = VotingClassifier(estimators=[('lr', model1), ('svc',model3), ('xgb',mod
model_fin.fit(X_train,y_train)
print(model_fin.score(X_test,y_test))
```

```
0.8244491826581379
```

```
import pickle
```

```
saved_file=pickle.dumps(model_fin)
```

```
print(model)
```

```
SVC(C=1, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='linear',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

```
model_pickle=pickle.loads(saved_file)
```

```
model_pickle.score(X_test,y_test)
```

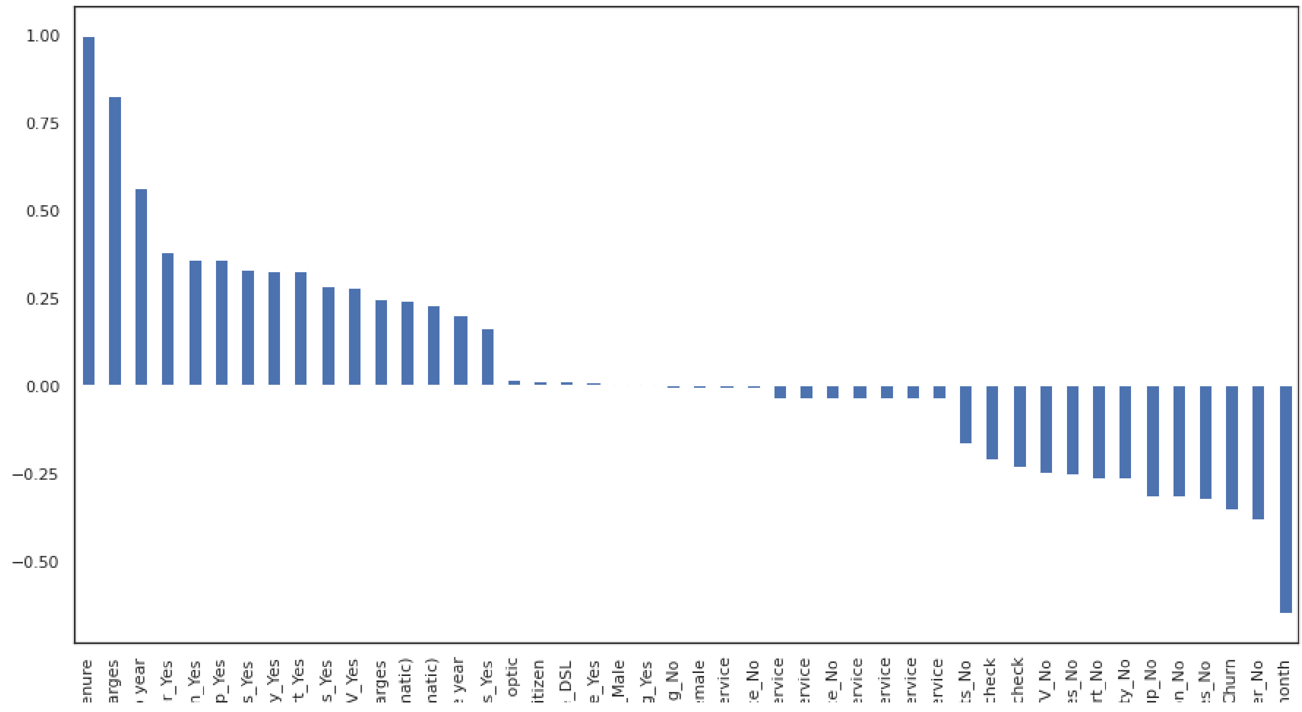
```
0.8244491826581379
```

DOING REGRESSION ANALYSIS BY TAKING MONTHLY CHARGES AND TOTAL CHARGES AS INDEPENDENT VARIABLES

```
from sklearn.linear_model import LinearRegression
from xgboost.sklearn import XGBClassifier
from sklearn.linear_model import LogisticRegression
```

```
plt.figure(figsize=(15,8))
df_dummies.corr()['tenure'].sort_values(ascending = False).plot(kind='bar')
```

<matplotlib.axes._subplots.AxesSubplot at 0x7fca2fe938d0>




```
import pickle
final_model=pickle.dumps(model_fin)
```

```
print(df_dummies.corr()['tenure'].sort_values(ascending = False))
```

```
tenure                                      1.000000
TotalCharges                                0.825880
Contract_Two year                           0.563801
Partner_Yes                                 0.381912
DeviceProtection_Yes                        0.361520
OnlineBackup_Yes                            0.361138
MultipleLines_Yes                           0.332399
OnlineSecurity_Yes                          0.328297
TechSupport_Yes                             0.325288
StreamingMovies_Yes                         0.285402
StreamingTV_Yes                             0.280264
MonthlyCharges                              0.246862
PaymentMethod_Bank transfer (automatic)     0.243822
PaymentMethod_Credit card (automatic)       0.232800
Contract_One year                           0.202338
Dependents_Yes                              0.163386
InternetService_Fiber optic                 0.017930
SeniorCitizen                               0.015683
InternetService_DSL                         0.013786
PhoneService_Yes                            0.007877
gender_Male                                 0.005285
PaperlessBilling_Yes                        0.004823
PaperlessBilling_No                        -0.004823
gender_Female                              -0.005285
MultipleLines_No phone service             -0.007877
PhoneService_No                            -0.007877
OnlineSecurity_No internet service         -0.037529
OnlineBackup_No internet service           -0.037529
InternetService_No                         -0.037529
DeviceProtection_No internet service       -0.037529
TechSupport_No internet service            -0.037529
```

```
StreamingTV_No internet service          -0.037529
StreamingMovies_No internet service      -0.037529
Dependents_No                            -0.163386
PaymentMethod_Electronic check           -0.210197
PaymentMethod_Mailed check               -0.232181
StreamingTV_No                           -0.246814
StreamingMovies_No                       -0.252890
TechSupport_No                           -0.264363
OnlineSecurity_No                        -0.265987
OnlineBackup_No                          -0.314769
DeviceProtection_No                      -0.314820
MultipleLines_No                         -0.323891
Churn                                    -0.354049
Partner_No                               -0.381912
Contract_Month-to-month                  -0.649346
Name: tenure, dtype: float64
```

```
data = telecom_cust.iloc[:,1:]
data.drop(columns=['SeniorCitizen','MultipleLines','InternetService','StreamingMovi
```

```
data['gender'].replace(to_replace='Male',value=1,inplace=True)
data['gender'].replace(to_replace='Female',value=0,inplace=True)
data['Partner'].replace(to_replace='Yes',value=1,inplace=True)
data['Partner'].replace(to_replace='No',  value=0, inplace=True)
data['PhoneService'].replace(to_replace=['Yes','No phone service','No'],value=[1,0,
data['OnlineSecurity'].replace(to_replace=['No internet service','Yes','No'],value=
data['OnlineBackup'].replace(to_replace=['No internet service','Yes','No'],value=[0
data['DeviceProtection'].replace(to_replace=['No internet service','Yes','No'],valu
data['TechSupport'].replace(to_replace=['No internet service','Yes','No'],value=[0,
data['Contract'].replace(to_replace='Month-to-month',  value=0, inplace=True)
data['Contract'].replace(to_replace='One year',  value=1, inplace=True)
data['Contract'].replace(to_replace='Two year',  value=2, inplace=True)
data['Contract'].replace(to_replace='Two year',  value=2, inplace=True)
data['StreamingTV'].replace(to_replace='No',  value=0, inplace=True)
data['StreamingTV'].replace(to_replace='Yes',  value=1, inplace=True)
data['StreamingTV'].replace(to_replace='No internet service',  value=0, inplace=Tru
```

```
data.head()
```

gender Partner tenure PhoneService OnlineSecurity OnlineBackup DeviceP

```
X=data.drop(columns=["tenure"])
y=data["tenure"].values
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta
model_LR=LinearRegression()
model_LG=LogisticRegression()

model_LR.fit(X_train,y_train)
model_LG.fit(X_train,y_train)
model_XGB.fit(X_train,y_train)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:940:
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regress
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-96-4f224c108532> in <module>()
      8 model_LR.fit(X_train,y_train)
      9 model_LG.fit(X_train,y_train)
---> 10 model_XGB.fit(X_train,y_train)

NameError: name 'model_XGB' is not defined
```

SEARCH STACK OVERFLOW

```
from sklearn.metrics import accuracy_score

y_pred = model_LR.predict(X_test)
predictions = [round(value) for value in y_pred]
# evaluate predictions
accuracy = accuracy_score(y_test, predictions)
print("Accuracy: %.2f%%" % (accuracy * 100.0))
```

```
model_LR.score(X_test,y_test)
```

```
model_LG.score(X_test,y_test)
```

```
X.shape
```

```
model_XGB.score(X_test,y_test)
```

```
saved_model=pickle.dumps(model_XGB)
```

```
from sklearn.metrics import accuracy_score

y_pred = model_XGB.predict(X_test)
predictions = [round(value) for value in y_pred]
# evaluate predictions
accuracy = accuracy_score(y_test, predictions)
print("Accuracy: %.2f%%" % (accuracy * 100.0))
```

```
X.columns
```

```
loadmodel=pickle.loads(saved_model)
```

```
print("enter the user data")
a=list(map(int,input().split())
newArray = numpy.append (a, [10, 11, 12])
arr = np.array(a)
pred=loadmodel.predict(arr)#this will predict the tenure.
print(pred[0])
if pred[0]>=12 and pred[0]<=24:
          print("10% discount")
elif pred[0] >24:
          print("15% discount")
else:
          print("No discount")
```