Project-1

Census Income

Problem Definition:

The data was obtained from the 1994 Census bureau database by Ronny Kohavi and Barry Becker (Data Mining and Visualization, Silicon Graphics). *The goal is to train a classifier to predict whether a person makes over $50K a year* or not so it has two possible outcomes ">50K" and "<50K". The data contain information such as "Age"," Workclass"," Fnlwgt"," Education" etc. Dataset has 32560 instances and 15 attributes containing a blend of categorical and numerical values.

Data Analysis:

Before moving to model building and evaluation phase, data analysis helps in understanding the data and deriving insights about dataset. Data analysis require cleaning, transforming and modelling of data to identify useful information from data and taking decision on basis of derived result.

First, import the required libraries

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

Now, Load the data into pandas DataFrame using read_csv function.

```
df=pd.read_csv("census_income.csv")
```

```
df.shape
```

Shape attribute provide the information about the dataset containing 32500 rows and 15 columns

Info function provide concise summary of the DataFrame including the datatype

```
df.info ()
```

```
Data columns (total 15 columns):
Age               32560 non-null int64
Workclass         32560 non-null object
Fnlwgt            32560 non-null int64
Education         32560 non-null object
Education_num     32560 non-null int64
Marital_status    32560 non-null object
Occupation        32560 non-null object
```

```
Relationship      32560 non-null object
Race              32560 non-null object
Sex               32560 non-null object
Capital_gain      32560 non-null int64
Capital_loss      32560 non-null int64
Hours_per_week    32560 non-null int64
Native_country    32560 non-null object
Income            32560 non-null object
```

It can be observed that no missing values can be identified with attributes a combination of numerical and categorical datatypes.

Handling numerical columns:

```
df.describe()
```

	Age	Fnlwgt	Education_num	Capital_gain	Capital_loss	Hours_per_week
count	32560.000000	3.256000e+04	32560.000000	32560.000000	32560.000000	32560.000000
mean	38.581634	1.897818e+05	10.080590	1077.615172	87.306511	40.437469
std	13.640642	1.055498e+05	2.572709	7385.402999	402.966116	12.347618
min	17.000000	1.228500e+04	1.000000	0.000000	0.000000	1.000000
25%	28.000000	1.178315e+05	9.000000	0.000000	0.000000	40.000000
50%	37.000000	1.783630e+05	10.000000	0.000000	0.000000	40.000000
75%	48.000000	2.370545e+05	12.000000	0.000000	0.000000	45.000000
max	90.000000	1.484705e+06	16.000000	99999.000000	4356.000000	99.000000

Describe function give us summary of statistics regarding the DataFrame columns. It displays the mean, std and IQR values for numeric columns.

It can be identified that Capital_gain and Capital_loss has median as 0 and huge difference between median and mean thus indicating skewness.

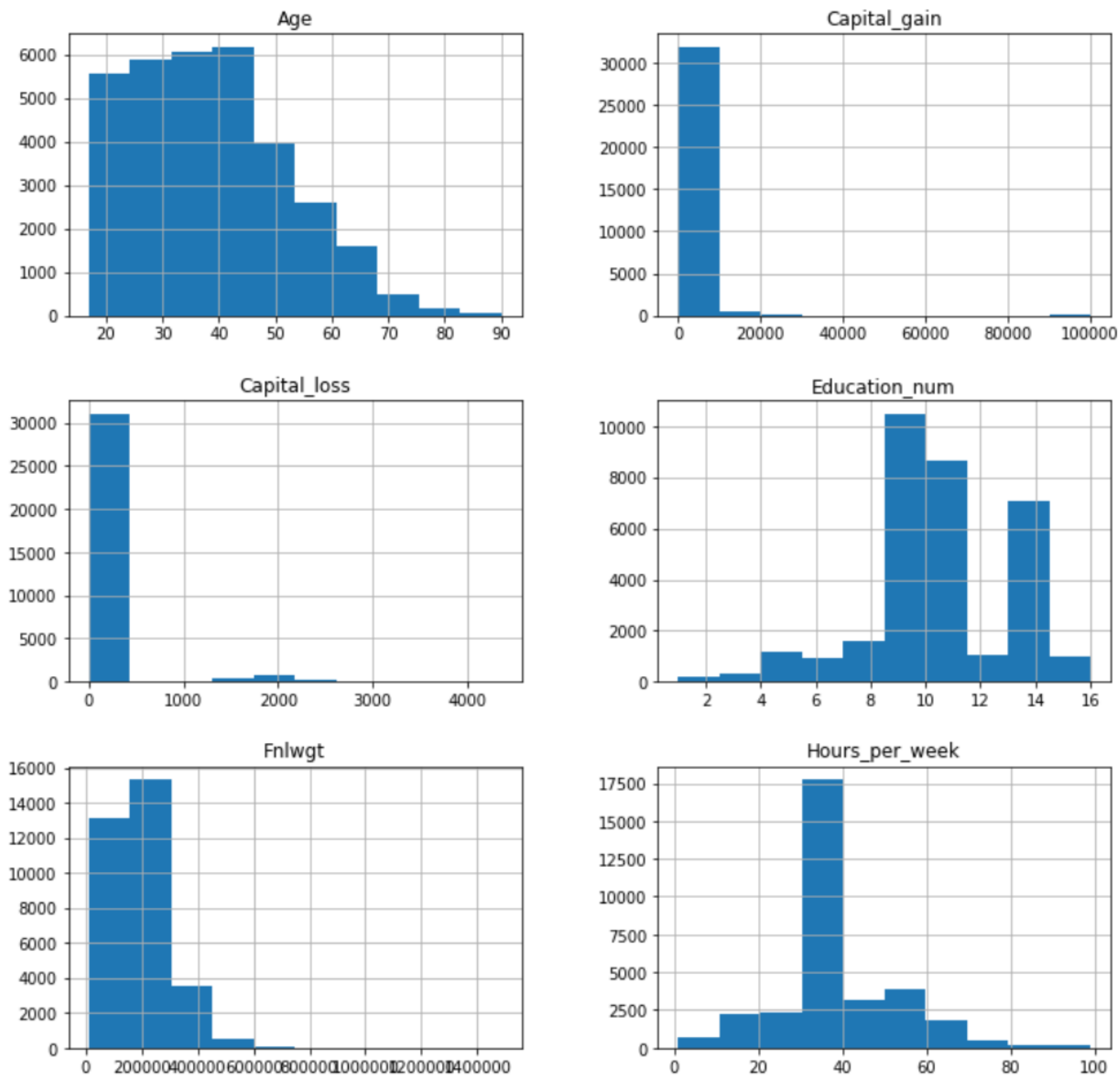
```
df["Workclass"].value_counts()
df["Occupation"].value_counts()
df["Native_country"].value_counts()
```

Using the value_counts()function it can be noticed that "Workclass","Occupation", "Native_country" have “?” as the value which need to be replaced by “unknown”.

```
edit_cols = ['Native_country','Occupation','Workclass']
# Replace ? with Unknown
for col in edit_cols:
    df.loc[df[col] == '?', col] = 'unknown'
```

Visualization of numerical columns

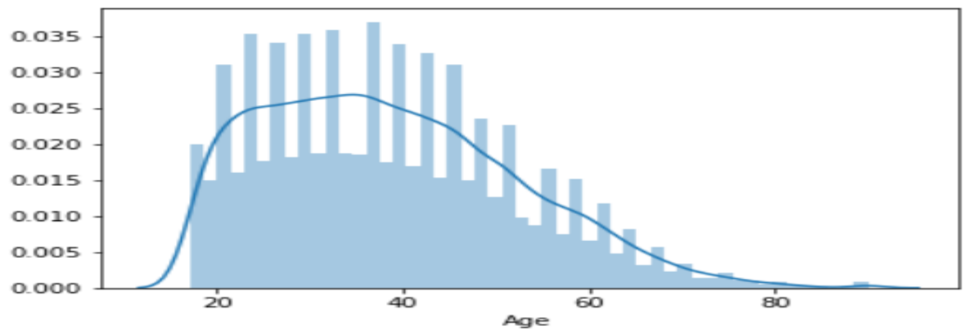
```
#Separate categorical and numerical columns
cat_set = df.dtypes[df. dtypes == 'object']
num_set = df.dtypes[df.dtypes != 'object']

df[list(num_set.index)].hist(figsize = (12,12))
```

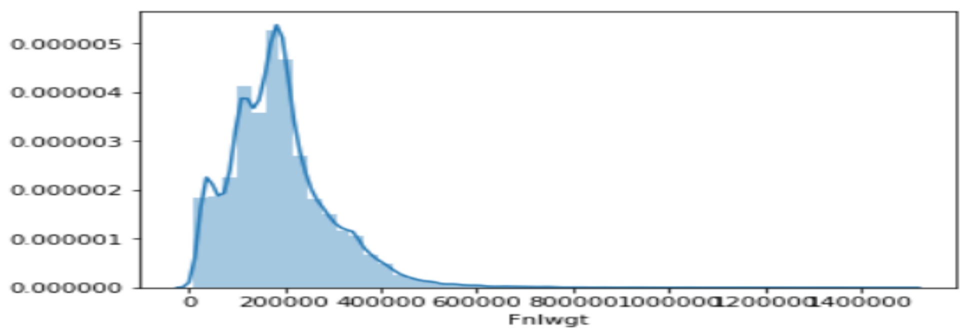


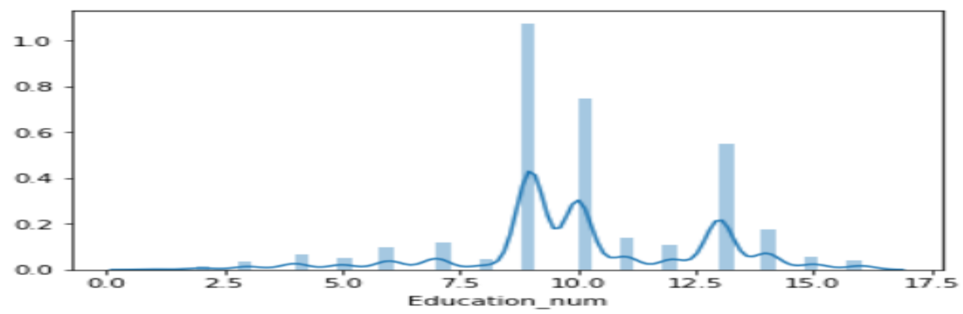
Histogram generate bin of the ranges, then distributes the values into a series of intervals and count the values which fall into each of the intervals. It provides information about frequency distribution.

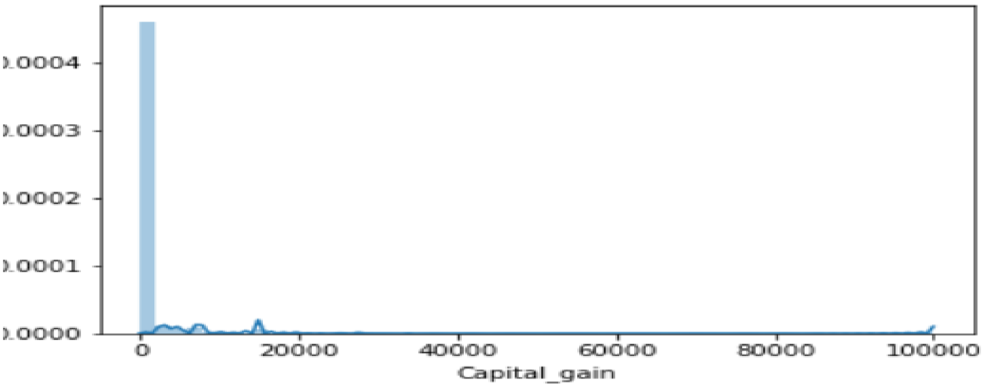
It can be deducted from the plot that for “Age”,” Capital_gain”,” Capital_loss”,” Education_num”,” Fnlwgt”,” Hours_per_week” data is not uniformly distributed.

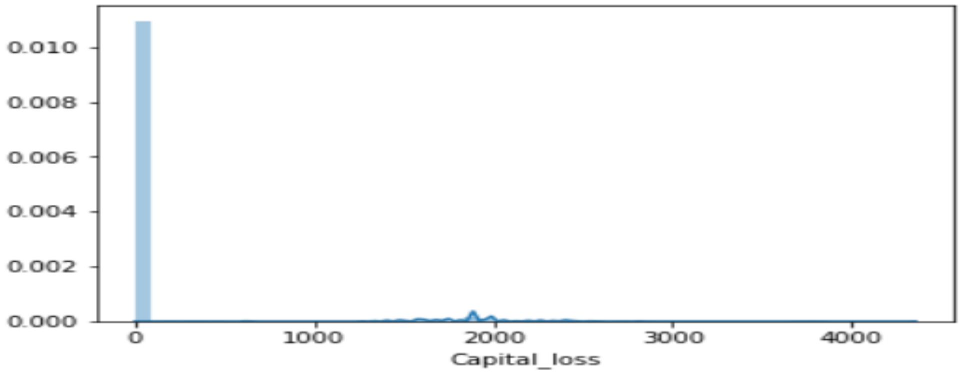
```
sns.distplot(df["Age"])
sns.distplot(df["Fnlwgt"])
sns.distplot(df["Education_num"])
sns.distplot(df["Capital_gain"])
sns.distplot(df["Capital_loss"])
sns.distplot(df["Hours_per_week"])
```

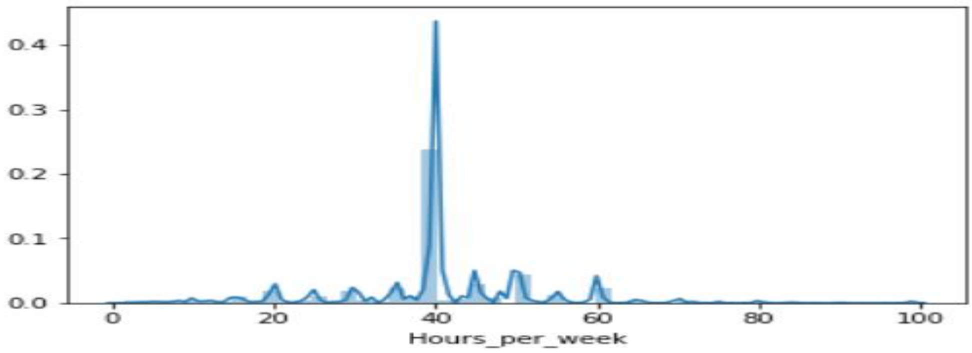










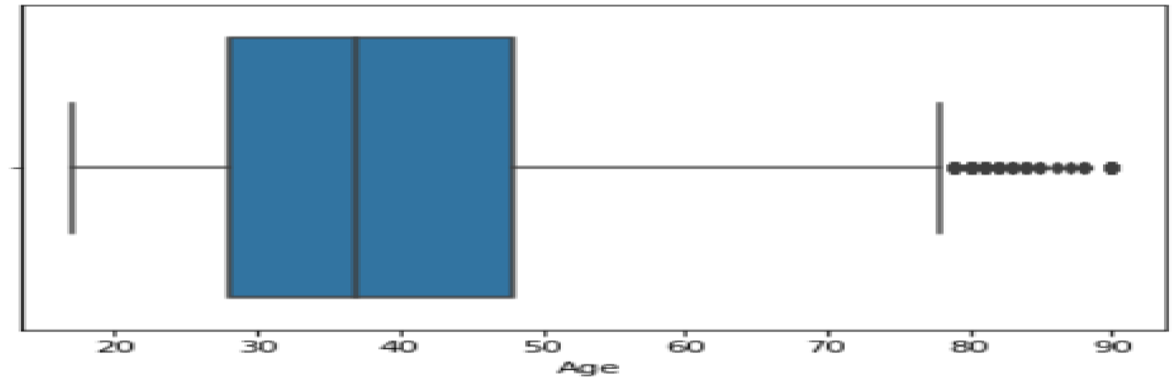


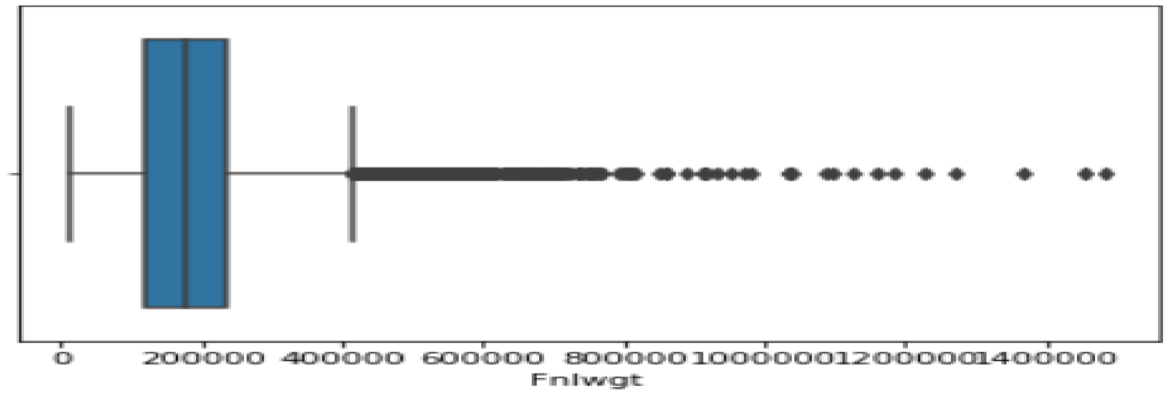

```
df.skew()
```

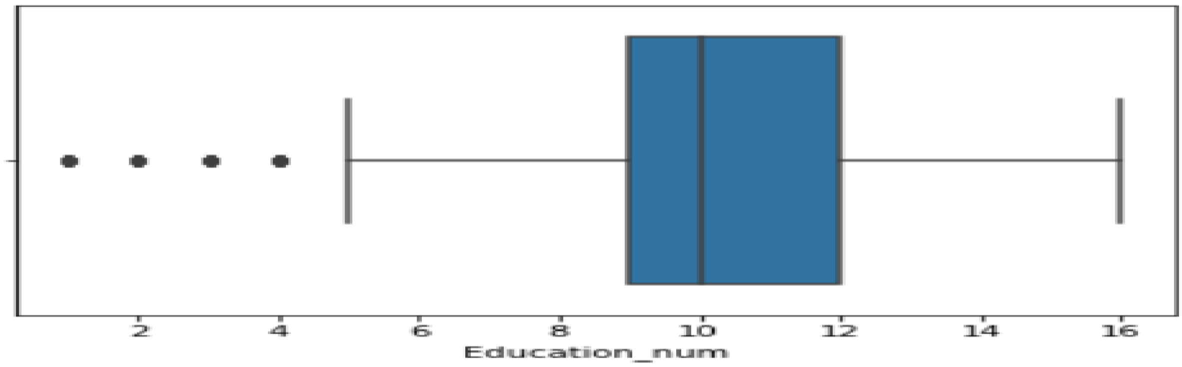
```
Age                0.558738
Fnlwgt             1.446972
Education_num     -0.311630
Capital_gain      11.953690
Capital_loss       4.594549
Hours_per_week     0.227636
```

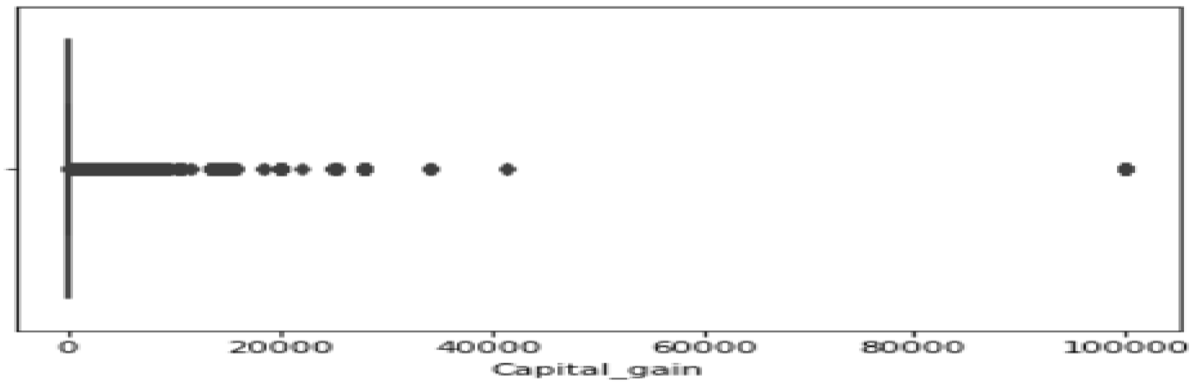
Using the distribution plot variation in data distribution can be identified. Also it helps us understanding the skewness in the data. It can be observed that "Age"," Capital_gain"," Capital_loss"," Fnlwgt" have skewed data. Skewness can be reduced using Power Transform, Log Transform, Square root Transform, Box-Cox Transform.

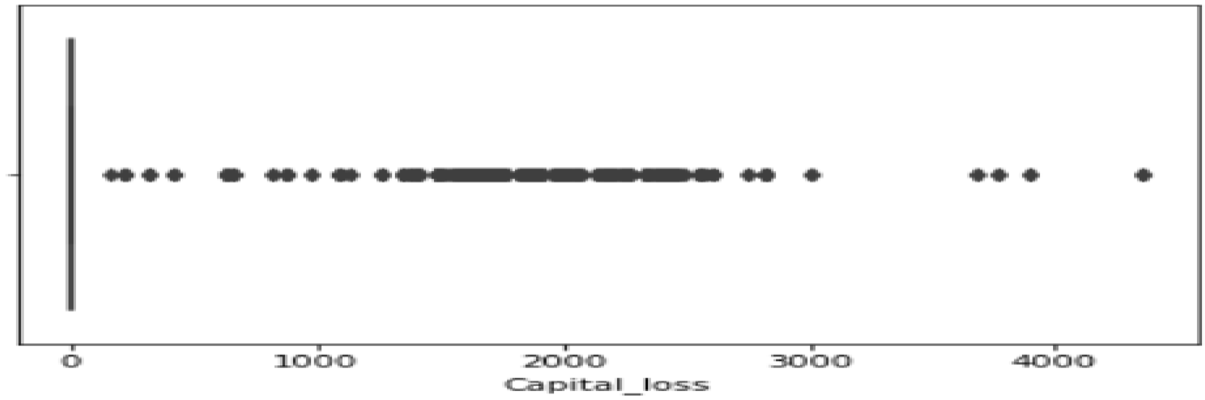
Along with skewness, outliers must also be considered so that model accuracy is not affected. To check the outliers, boxplot can be used and outliers can be removed using zscore.

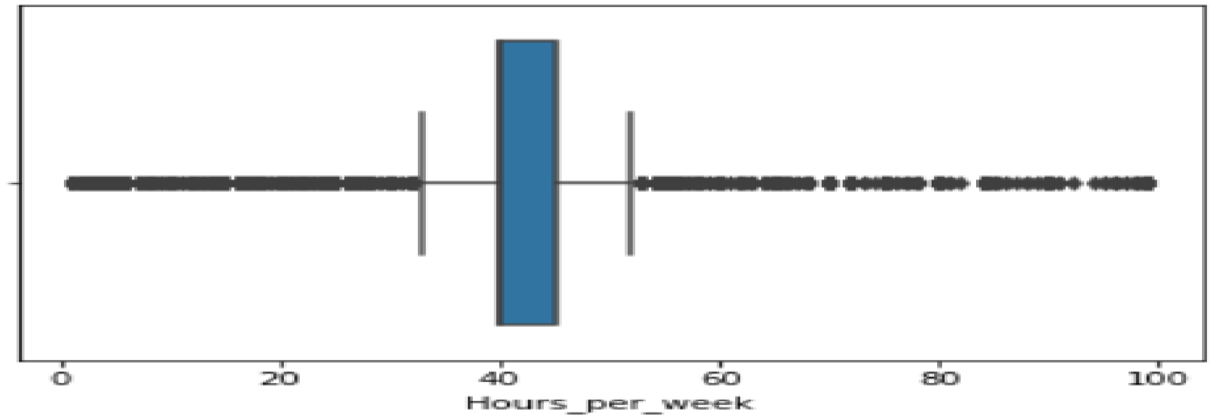










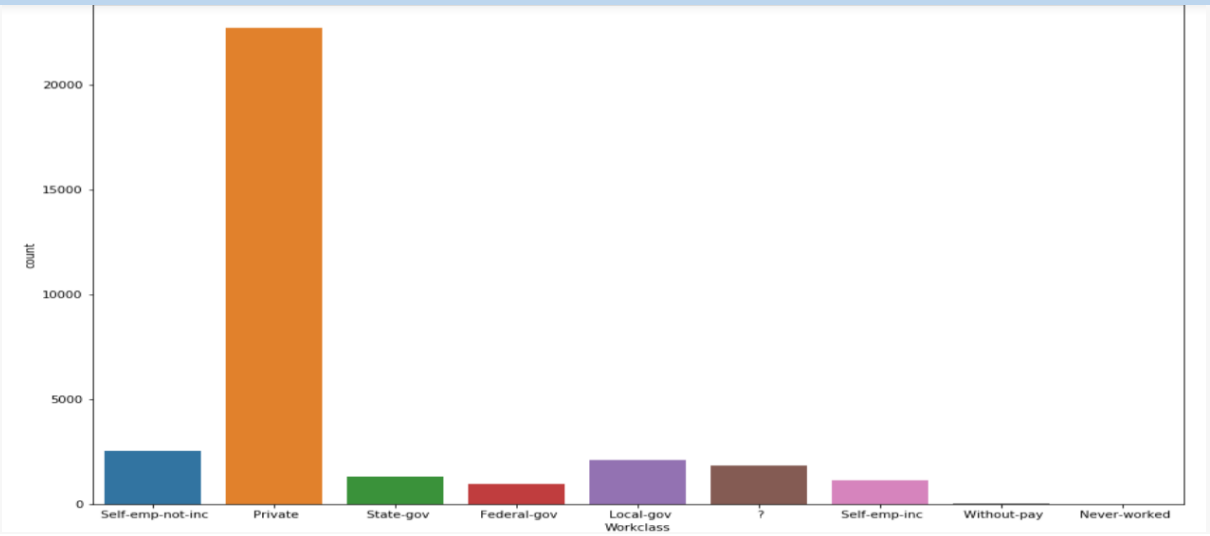


Looking at the boxplots, it can be identified that “Age”,” Capital_gain”,” Capital_loss”,” Education_num”,” Fnlwgt”,” Hours_per_week” have outlier values.

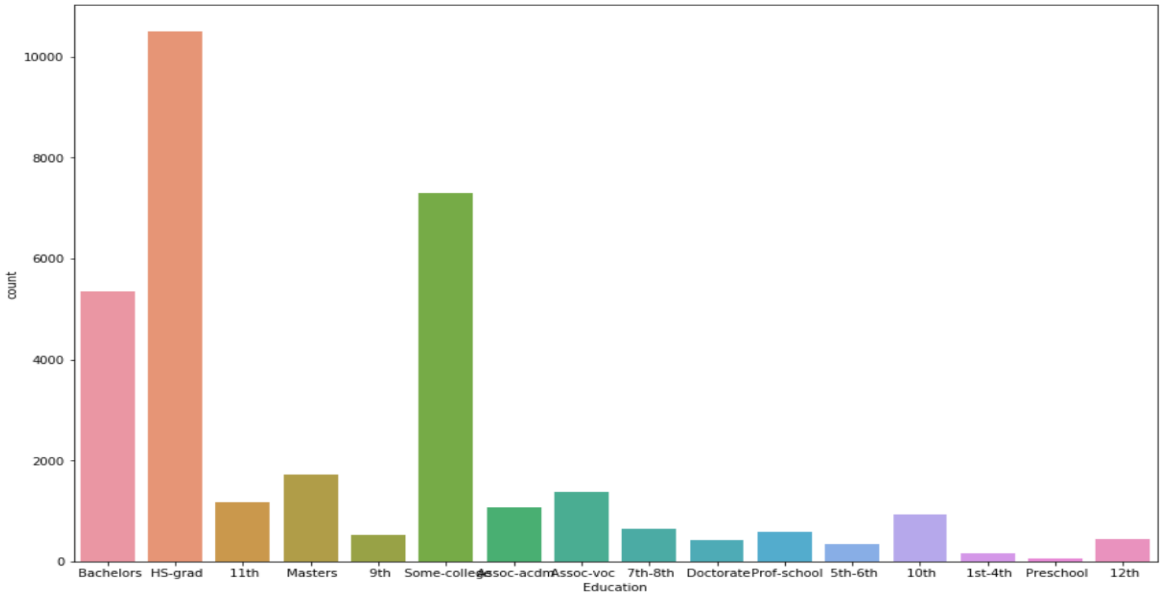
Handling Categorical columns

Categorical columns can be visualized using countplot as it provides the counts of observations in each categorical bin using bars.

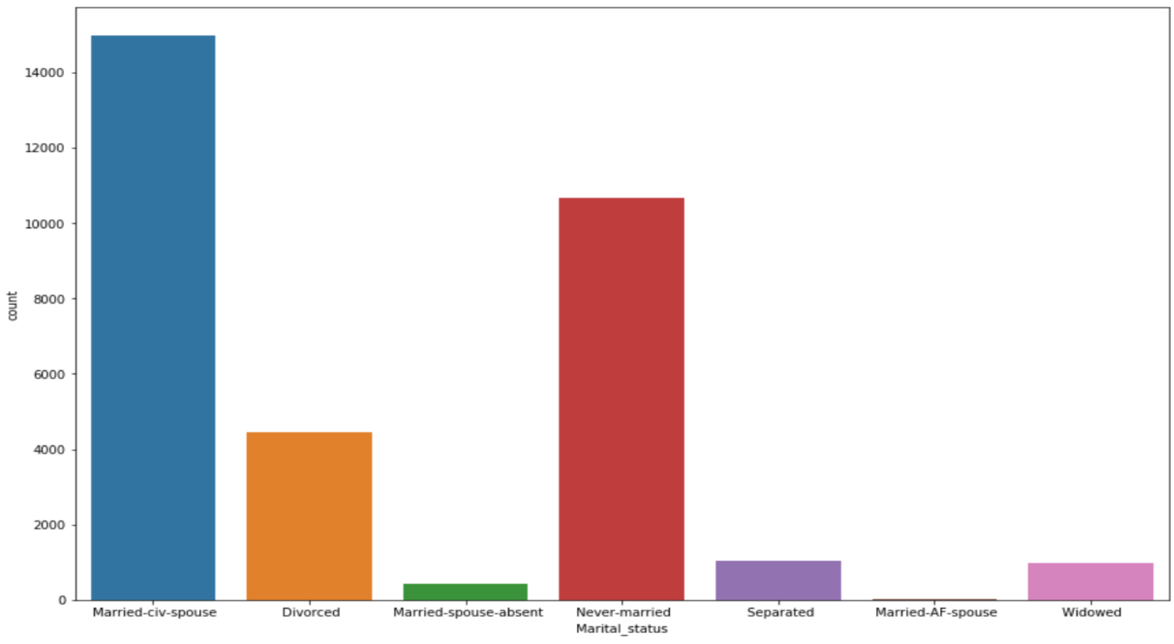
```
plt.figure(figsize=(15,10),facecolor="white")
sns.countplot(x="Workclass”, data=df)
```



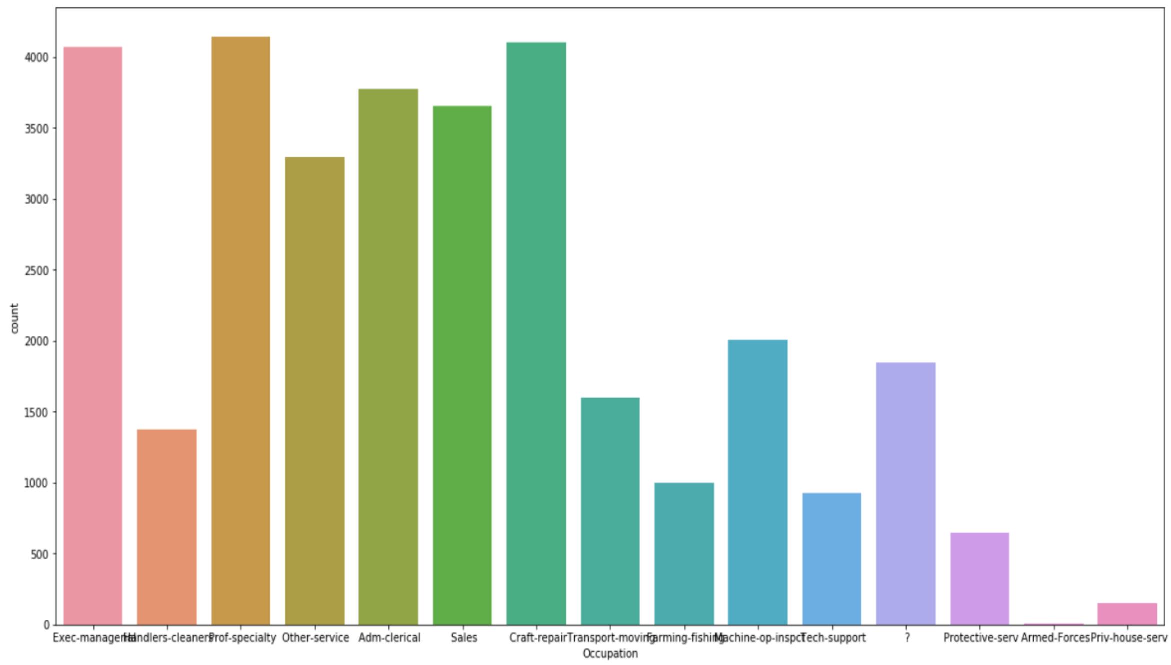
```
plt.figure(figsize=(15,10),facecolor="white")
sns.countplot(x="Education",data=df)
```



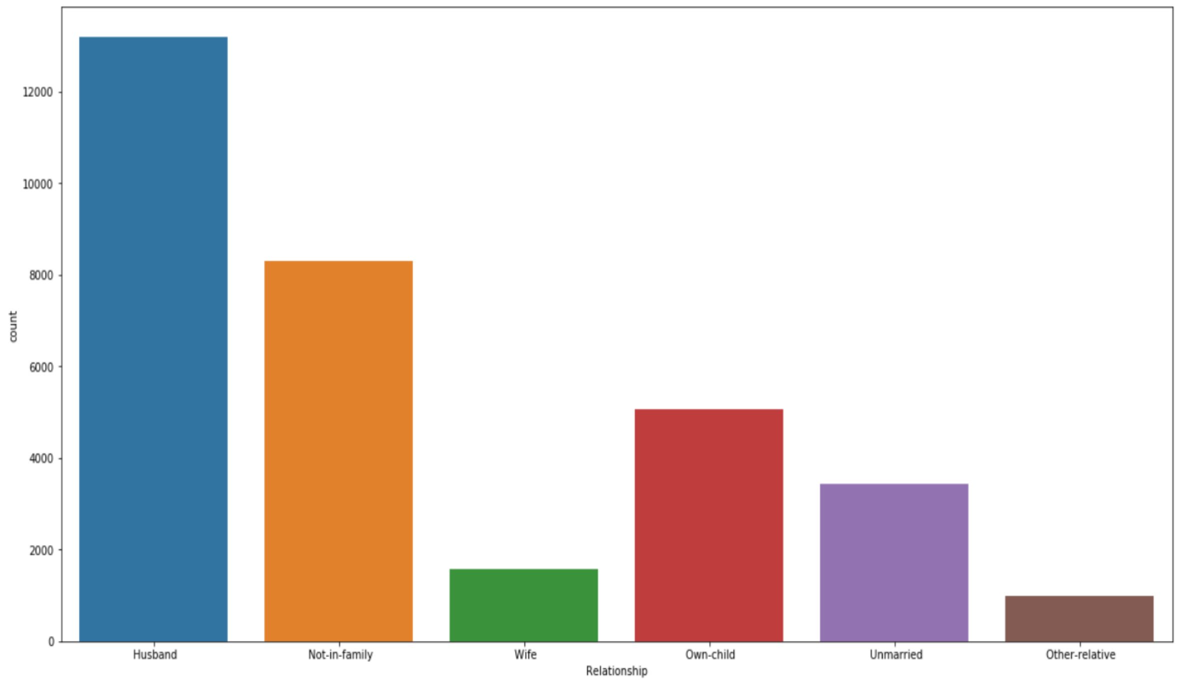

```
plt.figure(figsize=(15,10),facecolor="white")
sns.countplot(x="Marital_status",data=df)
```



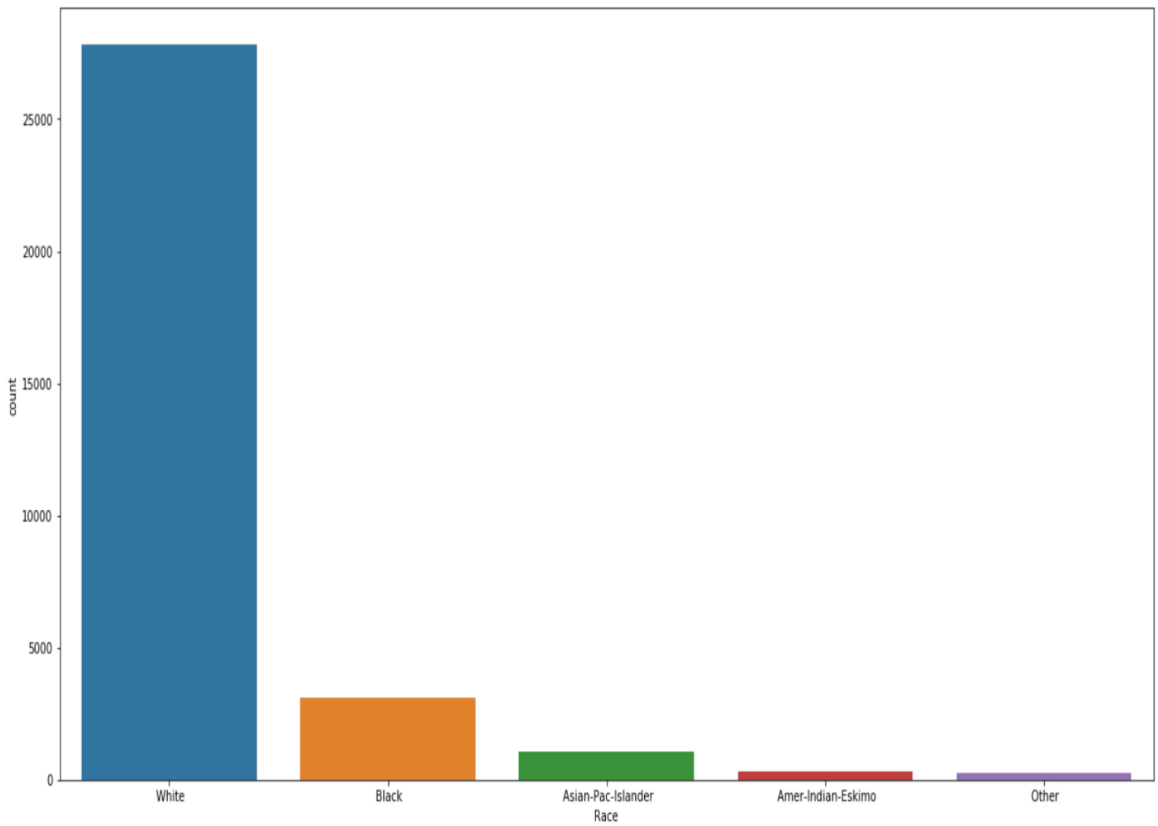

```
plt.figure(figsize=(20,10),facecolor="white")
sns.countplot(x="Occupation",data=df)
```



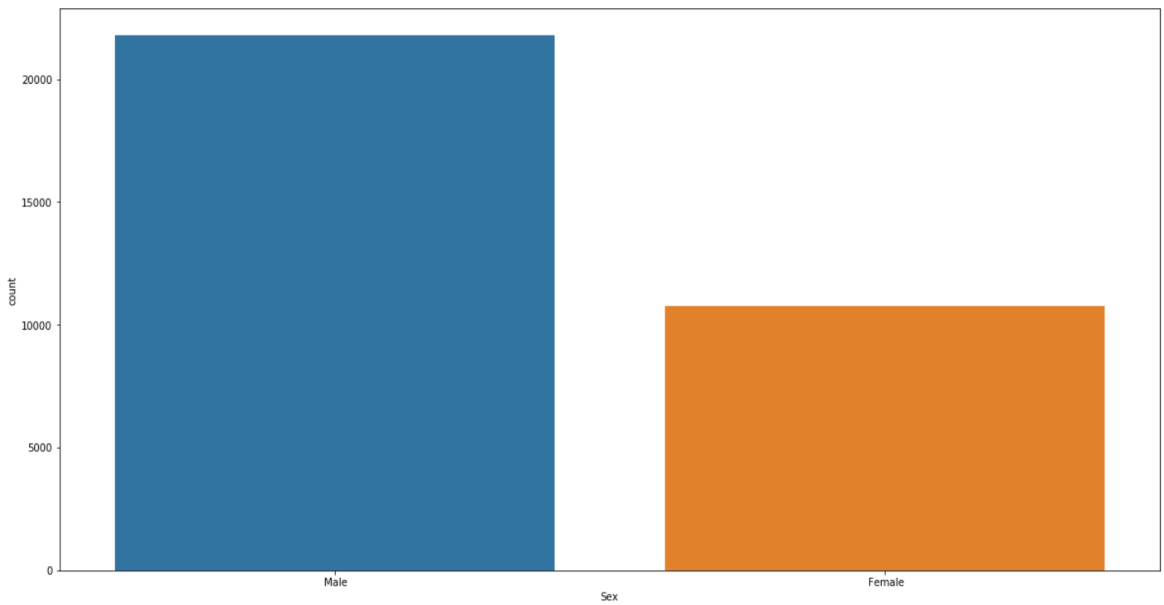

```
plt.figure(figsize=(20,10),facecolor="white")
sns.countplot(x="Relationship",data=df)
```



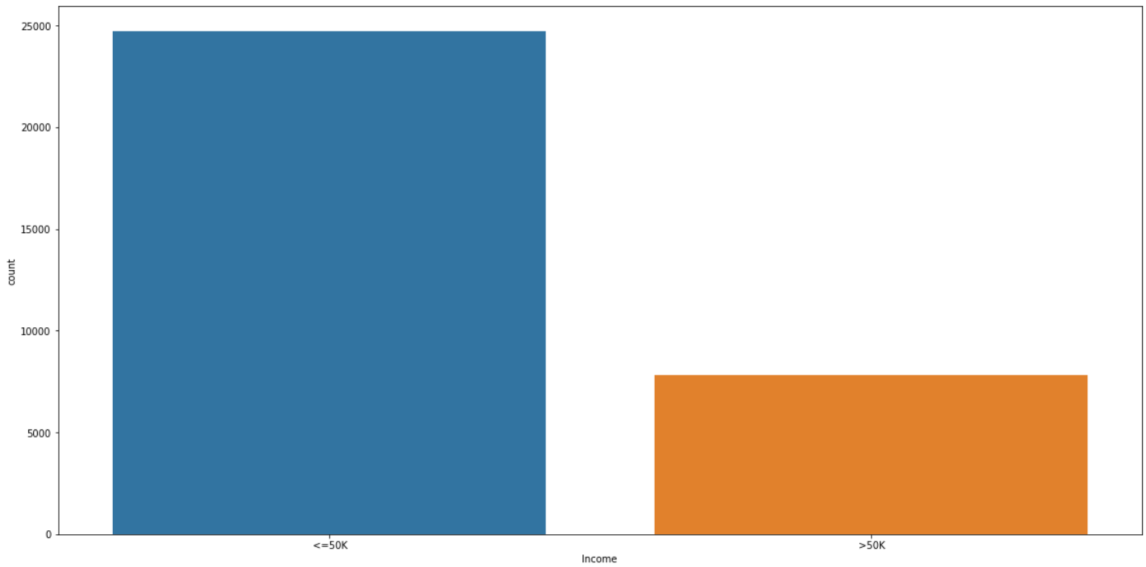

```
plt.figure(figsize=(20,10),facecolor="white")
sns.countplot(x="Race",data=df)
```




```
plt.figure(figsize=(20,10),facecolor="white")
sns.countplot(x="Sex",data=df)
```



```
plt.figure(figsize=(20,10),facecolor="white")
sns.countplot(x="Income",data=df)
```



Using the countplot it can be noticed that columns “Workclass”, "Education", "Marital_status", "Occupation", "Relationship", "Race", "Sex" have class imbalance problem which can be resolved using undersampling or oversampling technique.

Correlation

Correlation is really important to identify which columns are affecting the target column positively or negatively. Through correlation it can also be identified that which columns are strongly related to target.

```
df.corr()
```

For plotting correlation heatmap can be used to understand the correlation more easily.

```
import matplotlib.pyplot as plt
plt.figure(figsize=(30,15))
sns.heatmap(df.corr(),annot=True,linewidths=0.1,linecolor="black",fmt=".2f")
```

EDA Concluding Remark:

- Dataset contains 32500 rows and 15 columns with no missing values
- Dataset comprise of column of categorical as well as numerical type.
- "Workclass”,” Occupation”, Native_country" have “?” as the value which need to be

replaced by “unknown”.

- For columns “Age”,” Capital_gain”,” Capital_loss”,” Education_num”,” Fnlwgt”,” Hours_per_week” data is not uniformly distributed.

- Capital_gain and Capital_loss has median as 0 and huge difference between median and mean thus indicating skewness. “Age”,” Capital_gain”,” Capital_loss”,” Fnlwgt” have skewed data. Power Transform can be used to remove skewness.

- It can be identified that “Age”,” Capital_gain”,” Capital_loss”,” Education_num”,” Fnlwgt”,” Hours_per_week” have outlier values. Outliers can be removed using zscore.

- For columns “Workclass”, "Education", "Marital_status", "Occupation", "Relationship", "Race", "Sex”, it can be noticed that columns have class imbalance problem which can be resolved using undersampling or oversampling technique

- MinMaxScaler can be used to normalize data column by column.

Pre-Processing:

To handle the numerical and categorical attributes differently. Numerical attributes need to be scaled, whereas for categorical columns missing values need to be filled and then encode the categorical values into numerical values.

We have lot of features which are categorical and for building a model, categorical column need to encoded. OrdicalEncoder can be used to encode the categorical columns

```
from sklearn.preprocessing import OrdinalEncoder
enc=OrdinalEncoder()
for i in df.columns:
   if df[i].dtypes=="object":
      df[i]=enc.fit_transform(df[i].values.reshape(-1,1))
```

Before proceeding to model fitting, data transformation must be performed. Properly formatted and validated data improves data quality. Data transformation allow scaling each feature to a given range. For the given dataset MinMaxScaler can be used which scales all the data features in the range [0,1] or else in the range [-1,1].

Here, MinMaxScaler will be used to normalize the data column by column.

```
from sklearn.preprocessing import MinMaxScaler
scale=MinMaxScaler()
df["Age"]=scale.fit_transform(df["Age"].values.reshape(-1,1))
```

```
from sklearn.preprocessing import MinMaxScaler
scale=MinMaxScaler()
df["Fnlwgt"]=scale.fit_transform(df["Fnlwgt"].values.reshape(-1,1))
```

```
from sklearn.preprocessing import MinMaxScaler
scale=MinMaxScaler()
df["Education_num"]=scale.fit_transform(df["Education_num"].values.reshape(-1,1))
```

```
from sklearn.preprocessing import MinMaxScaler
scale=MinMaxScaler()
df["Capital_gain"]=scale.fit_transform(df["Capital_gain"].values.reshape(-1,1))
```

```
from sklearn.preprocessing import MinMaxScaler
scale=MinMaxScaler()
df["Capital_loss"]=scale.fit_transform(df["Capital_loss"].values.reshape(-1,1))
```

```
from sklearn.preprocessing import MinMaxScaler
scale=MinMaxScaler()
df["Hours_per_week"]=scale.fit_transform(df["Hours_per_week"].values.reshape(-1,1))
```

Building Machine Learning Models:

Model Building involves splitting of original dataset into input(x) and output (y) columns.

```
y=df["Income"]
x=df.drop("Income", axis=1)
```

x and y will be used to divide the dataset into training dataset and testing dataset.

Training dataset can be used for model fitting and statistical method can be used to estimate the accuracy of the model using the unseen data. Testing set can be used for evaluating the model accuracy.

Loaded dataset will be spitted into two, 70% of which will be used to train and 30% will be used to test the model.

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score,classification_report
from sklearn.metrics import confusion_matrix, roc_auc_score
from sklearn.model_selection import train_test_split
import warnings
warnings. filterwarnings("ignore")
```

Random state allows to generate a reproducible split. Scikit-learn use random permutations to generate the splits. Random state provided is a seed to generate random number and also ensure that generated numbers are in same order.

Maximum accuracy_score can be used to identify the best random state from 1 to 200 and will proceed with same random state to fit the model using different algorithms.

```
Maxr2=0
BestRs=0
for i in range(1,200):
   xtrain,xtest,ytrain,ytest=train_test_split(x,y,test_size=.30,random_state=i)
   lr=LogisticRegression()
   lr.fit(xtrain,ytrain)
   pred=lr.predict(xtest)
   accu=accuracy_score(pred,ytest)
   if accu>Maxr2:
      Maxr2=accu
      BestRs=i
print("with random state as",BestRs,"max accuracy is",Maxr2)
```

```
with random state as 56 max accuracy is 0.8223791973791974
```

So maximum accuracy score is .82237 which mean 82.237% accuracy at random state 56.

Following are the metrics we'll use to evaluate our predictive accuracy:

- Sensitivity = True Positive Rate (TP/TP+FN) – It says, 'out of all the positive (majority class) values, how many have been predicted correctly'.
- Specificity = True Negative Rate (TN/TN +FP) – It says, 'out of all the negative (minority class) values, how many have been predicted correctly'.
- Precision = (TP/TP+FP)
- Recall = Sensitivity
- F score = 2 * (Precision * Recall)/ (Precision + Recall) – It is the harmonic mean of precision and recall. It is used to compare several models side-by-side. Higher the better.

Logistic Regression Model

So we will fit logistic regression model with random state=56

```
xtrain,xtest,ytrain,ytest=train_test_split(x,y,test_size=.30,random_state=56)
lr=LogisticRegression()
lr.fit(xtrain,ytrain)
pred=lr.predict(xtest)
accu=accuracy_score(pred,ytest)
print(classification_report(pred,ytest))
print(accu)
print(confusion_matrix(pred,ytest))
```

```
              precision    recall  f1-score   support

         0.0       0.94      0.85      0.89      8318
         1.0       0.44      0.69      0.53      1450

    accuracy                           0.82      9768
   macro avg       0.69      0.77      0.71      9768
weighted avg       0.86      0.82      0.84      9768

accuracy score = 0.8223791973791974

confusion matrix = [[7036 1282]
                    [ 453  997]]
```

Outcome of the logistic regression model gave us accuracy score (0.822) and f1-score as .82

Random Forest Model

So, fitting a Random Forest model with random state 56

```
from sklearn.ensemble import RandomForestClassifier

RFC=RandomForestClassifier()
RFC.fit(xtrain,ytrain)
pred=RFC.predict(xtest)
accu=accuracy_score(pred,ytest)
print(classification_report(pred,ytest))
print(accu)
print(confusion_matrix(pred,ytest))
```

```
               precision     recall  f1-score    support

         0.0       0.93      0.89      0.91      7764
```

```
         1.0       0.64      0.73      0.68      2004

    accuracy                           0.86      9768
   macro avg       0.78      0.81      0.80      9768
weighted avg       0.87      0.86      0.86      9768

accuracy score = 0.8606674856674856

confusion matrix = [[6946  818]
                    [ 543 1461]]
```

Outcome of the Random forest classifier model gave us accuracy score (0.86) and f1-score is .86

Decision Tree Model

So, fitting a Decision Tree model with random state 56

```
from sklearn.tree import DecisionTreeClassifier
dtc=DecisionTreeClassifier()
dtc.fit(xtrain,ytrain)
pred=dtc.predict(xtest)
accu=accuracy_score(pred,ytest)
print(classification_report(pred,ytest))
print(accu)
print(confusion_matrix(pred,ytest))
```

```
               precision    recall  f1-score   support

         0.0       0.87      0.89      0.88      7323
         1.0       0.63      0.59      0.61      2445

    accuracy                           0.81      9768
   macro avg       0.75      0.74      0.74      9768
weighted avg       0.81      0.81      0.81      9768

accuracy score = 0.812039312039312

confusion matrix = [[6488  835]
                    [1001 1444]]
```

Outcome of the Decision Tree Classifier model gave us accuracy score (0.812) and f1-score as .81

Support Vector Machine Model

So, fitting a support vector machine classifier model with random state 56

```
from sklearn.svm import SVC
svc=SVC()
svc.fit(xtrain,ytrain)
pred=svc.predict(xtest)
accu=accuracy_score(pred,ytest)
print(classification_report(pred,ytest))
print(accu)
print(confusion_matrix(pred,ytest))
```

```
              precision    recall  f1-score   support

         0.0       1.00      0.77      0.87      9768
         1.0       0.00      0.00      0.00         0

    accuracy                           0.77      9768
   macro avg       0.50      0.38      0.43      9768
weighted avg       1.00      0.77      0.87      9768

accuracy score = 0.7666871416871417

confusion matrix = [[7489 2279]
                    [   0    0]]
```

Outcome of the support vector classifier model gave us accuracy score (0.766) and f1-score (.77)

Cross validation

Cross Validation is a technique for assessing how the statistical analysis generalises to an independent dataset. It is a technique for evaluating machine learning models by training several models on subsets of the available input data and evaluating them on the complementary subset of the data

For a k fold cross-validation, k refers to the number of groups a given data sample is to be split into and the procedure is often called k-fold cross-validation. When a specific value for k is chosen, then it may be used such as k=5

```
from sklearn.model_selection import cross_val_score

model=[lr,dtc,RFC,svc]

for model in model:
   print(cross_val_score(model, x, y, cv=5).mean())
```

```
cross_val_score for logistic regression = 0.8126535626535626
cross_val_score for decision tree = 0.8073095823095823
cross_val_score for random forest = 0.8570331695331694
cross_val_score for support vector = 0.7591830466830467
```

After comparing the accuracy_score and cross_val_score for logistic regression, decision tree classifier, random forest classifier and support vector classifier, it can be noticed that difference in accuracy_score and cross_val_score for random forest classifier is lowest implying that random forest classifier model is more accurate for prediction.

So we will proceed with random forest classifier and will try to tune hyperparameter using GridSearchCV

Hyperparameter tuning

Hyperparameter is a parameter which control the learning rate. Hyperparameters tuning is crucial as they control the overall behavior of a machine learning model. Every machine learning models will have different hyperparameters that can be set. A hyperparameter is a parameter whose value is set before the learning process begins

Tuning the parameters of the Random Forest in order to obtain the best possible parameters for model building.

```
from sklearn.model_selection import GridSearchCV
parameter={"n_estimators":np.arange(10,100),"max_depth":np.arange(2,10),"criterion":["g
ini", "entropy"],"max_features":["auto","sqrt","log2"]}
grid=GridSearchCV(estimator=RFC,param_grid=parameter,cv=5)
grid.fit(xtrain,ytrain)
print(grid.best_score_)
print(grid.best_params_)
```

```
Best score = 0.8548175493632748

Best parameter = {'criterion': 'gini', 'max_depth': 9, 'max_features':
'auto', 'n_estimators': 34}
```

Using gridsearchcv best parameters are identified which can used to develop final random forest classifier model.Best score which is obtained using the best parameters is .85481.

So we will be developing final model using the best parameters.

```
{'criterion': 'gini', 'max_depth': 9, 'max_features': 'auto', 'n_estima
tors': 34}
```

Final Model

```
finalrfc=RandomForestClassifier(n_estimators=34,criterion='gini',max_depth=9,max_feature
s='auto')
finalrfc.fit(xtrain,ytrain)
pred=finalrfc.predict(xtest)
accu=accuracy_score(pred,ytest)
print(classification_report(pred,ytest))
print(accu)
print(confusion_matrix(pred,ytest))
```

```
              precision    recall  f1-score   support

         0.0       0.96      0.88      0.92      8199
         1.0       0.56      0.81      0.66      1569

    accuracy                           0.87      9768
   macro avg       0.76      0.84      0.79      9768
weighted avg       0.89      0.87      0.87      9768

0.8654791154791155
[[7187 1012]
 [ 302 1267]]
```

Final model which is finalrfc has an accuracy_score of .8654 and f1_score of .87. Now we will be saving the final model and 87% accuracy means that it can be predicted that a person has a salary above $50K or lower than $50 K with 87% accuracy.

Saving the final model

Saving the model using joblib library

```
import joblib
joblib.dump(finalrfc,"census_income_pred.obj")
```

Concluding Remarks:

Here by using sklearn, we have built a preliminary machine learning tool and we have used the accuracy_score to select the random forest classifier model as the best model for predicting the person's salary whether it is above $50 K or below $50 K. Our final model has an accuracy of 87% using the parameter tuning. We can further try to improve the model prediction by working on preprocessing and roc_auc_score. Of course, there is always tools and analysis you can do further in order to make it more accurate, and better to use.

Project-2

Customer Churn

Problem Definition:

To examine customer data from IBM Sample Data Sets with the aim of building and compari ng several customer churn prediction models. The data contain information such as 'custom erID', 'gender', 'SeniorCitizen', 'Partner', etc. Dataset has 7043 instances and 21 attributes co ntaining a blend of categorical and numerical values.

Data Analysis:

Before moving to model building and evaluation phase, data analysis helps in understanding the data and deriving insights about dataset. Data analysis require cleaning, transforming and modelling of data to identify useful information from data and taking decision on basis of derived result.

First, import the required libraries

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

Now, Load the data into pandas DataFrame using read_csv function.

```
df=pd.read_csv(“"Customer_churn_analysis.csv"")
```

```
df.shape
```

Shape attribute provide the information about the dataset containing 7043 rows and 21 columns

Info function provide concise summary of the DataFrame including the datatype

```
df.info ()
```

```
customerID          7043 non-null object
gender              7043 non-null object
SeniorCitizen       7043 non-null int64
Partner             7043 non-null object
Dependents          7043 non-null object
tenure              7043 non-null int64
PhoneService        7043 non-null object
MultipleLines       7043 non-null object
InternetService     7043 non-null object
OnlineSecurity      7043 non-null object
OnlineBackup        7043 non-null object
DeviceProtection    7043 non-null object
TechSupport         7043 non-null object
StreamingTV         7043 non-null object
StreamingMovies     7043 non-null object
Contract            7043 non-null object
PaperlessBilling    7043 non-null object
PaymentMethod       7043 non-null object
MonthlyCharges      7043 non-null float64
TotalCharges        7043 non-null object
Churn               7043 non-null object
```

It can be observed that no missing values can be identified with attributes a combination of numerical and categorical datatypes.

Handling numerical columns:

```
df.describe()
```

	SeniorCitizen	tenure	MonthlyCharges
count	7043.000000	7043.000000	7043.000000
mean	0.162147	32.371149	64.761692
std	0.368612	24.559481	30.090047
min	0.000000	0.000000	18.250000
25%	0.000000	9.000000	35.500000
50%	0.000000	29.000000	70.350000
75%	0.000000	55.000000	89.850000
max	1.000000	72.000000	118.750000

Describe function give us summary of statistics regarding the DataFrame columns. It displays the mean, std and IQR values for numeric columns.

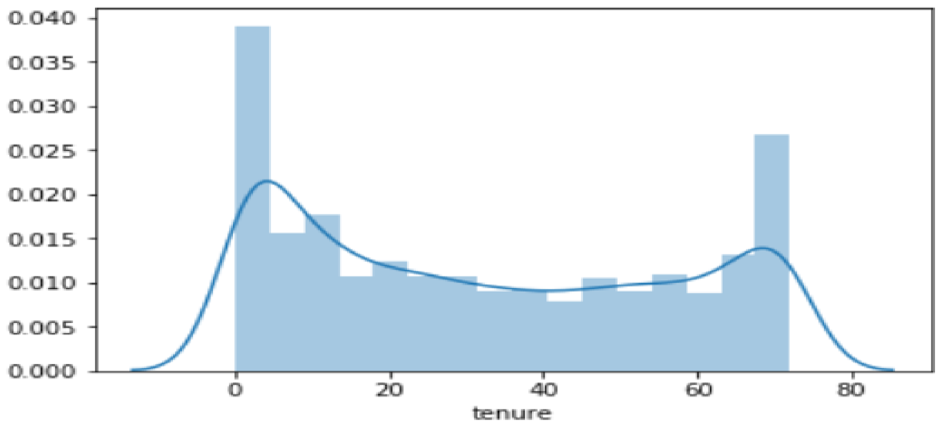
It can be identified that SeniorCitizen has median as 0 and mean as 0. 1621.Similarly mean for tenure is 32.37 and for MonthlyCharges it is 64.76.

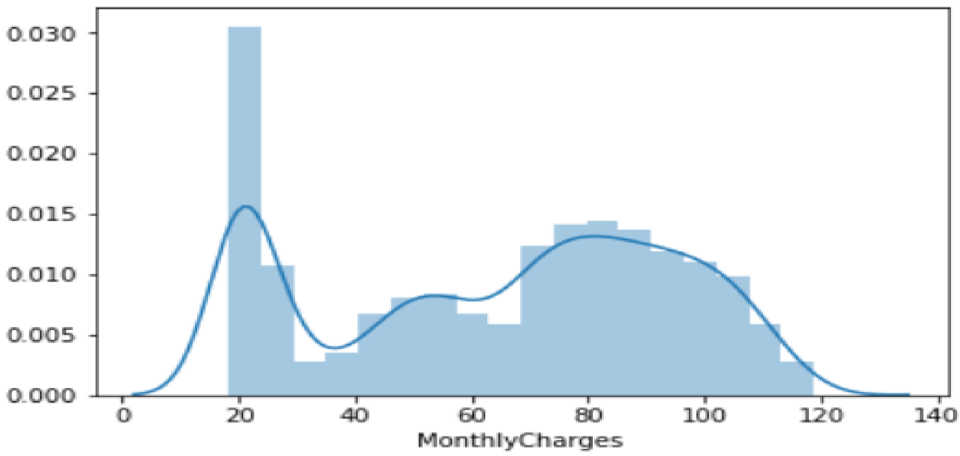
```
df["TotalCharges"].value_counts()
df["MonthlyCharges"].value_counts()
df["tenure"].value_counts()
```

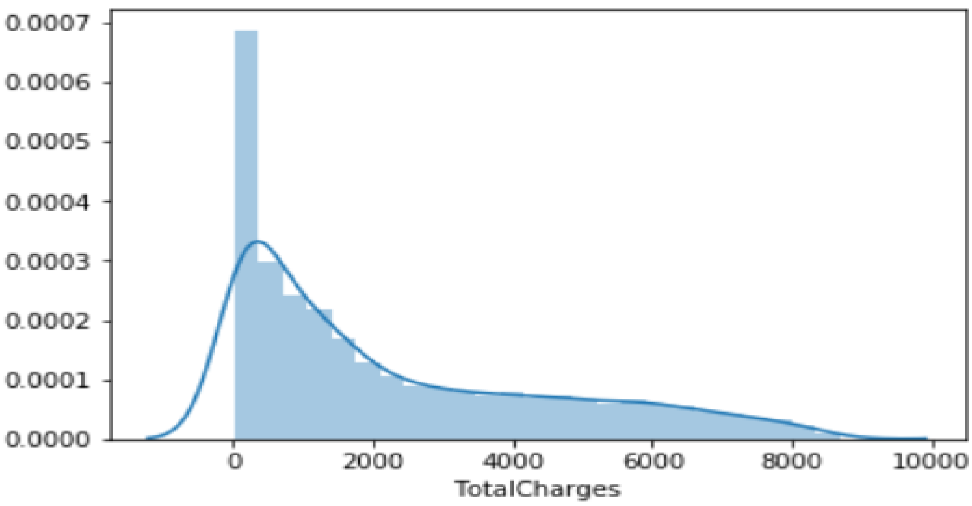
Using the value_counts() function it can easily be noticed that “Totalcharges” have datatype as object but values are float type so need to convert datatype

Visualization of numerical columns

```
sns.distplot(df["tenure"])
sns.distplot(df["MonthlyCharges"])
sns.distplot(df["TotalCharges"])
```



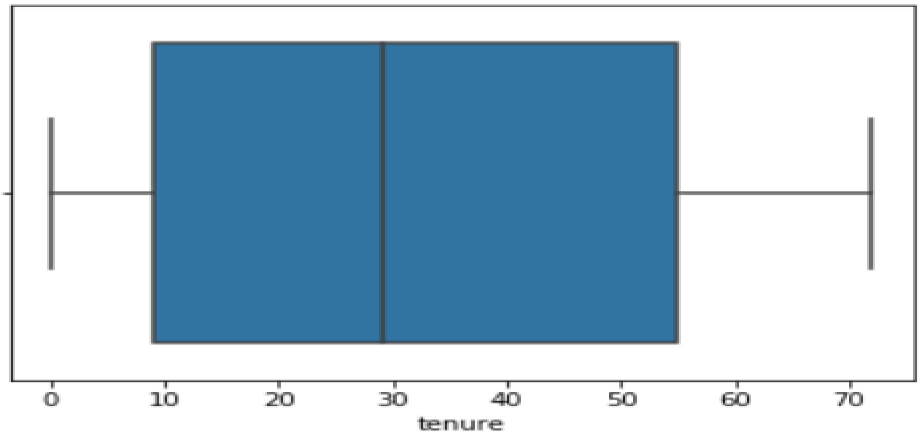




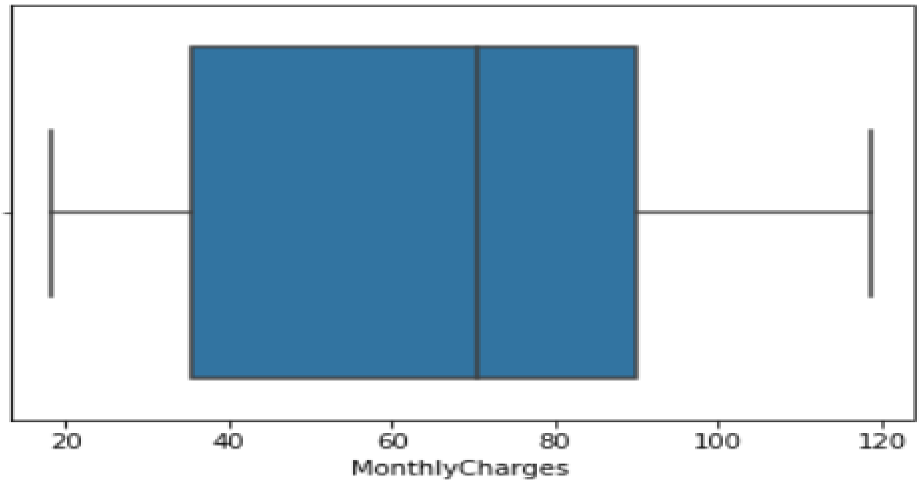
Using the distribution plot variation in data distribution can be identified. Also it helps us understanding the skewness in the data. From distribution plot it can be identified that “tenure”,” MonthlyCharges”,”TotalCharges” are not equally distributed and are skewed. Skewness can be reduced using Power Transform, Log Transform, Square root Transform, Box-Cox Transform.

Along with skewness, outliers must also be considered so that model accuracy is not affected. To check the outliers, boxplot can be used and outliers can be removed using zscore.

```
sns.boxplot(x="tenure",data=df)
```




```
sns.boxplot(x="MonthlyCharges",data=df)
```

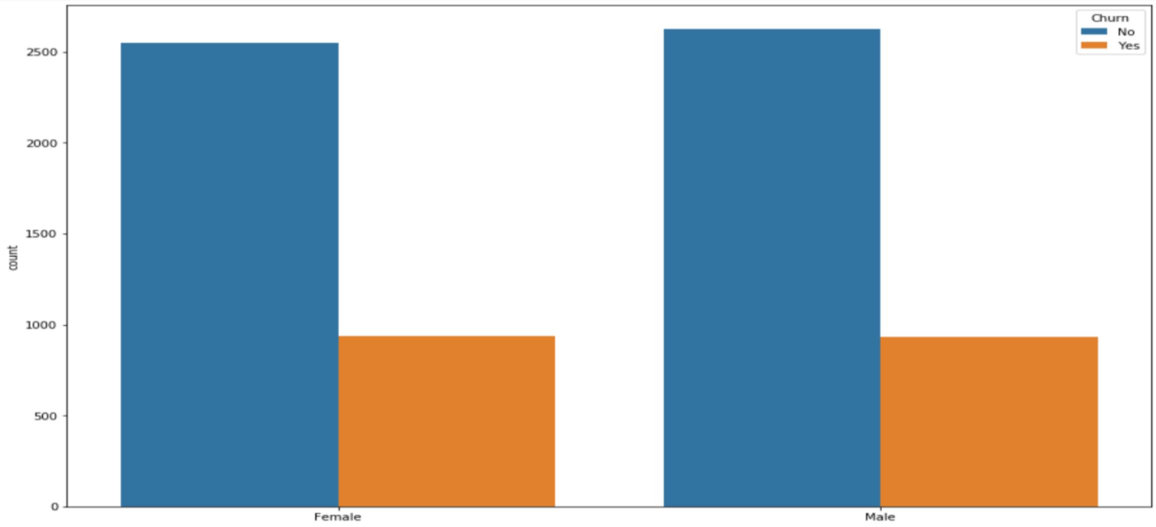


Looking at the boxplots, it can be identified that “tenure”,” MonthlyCharges”, do not have outlier values.

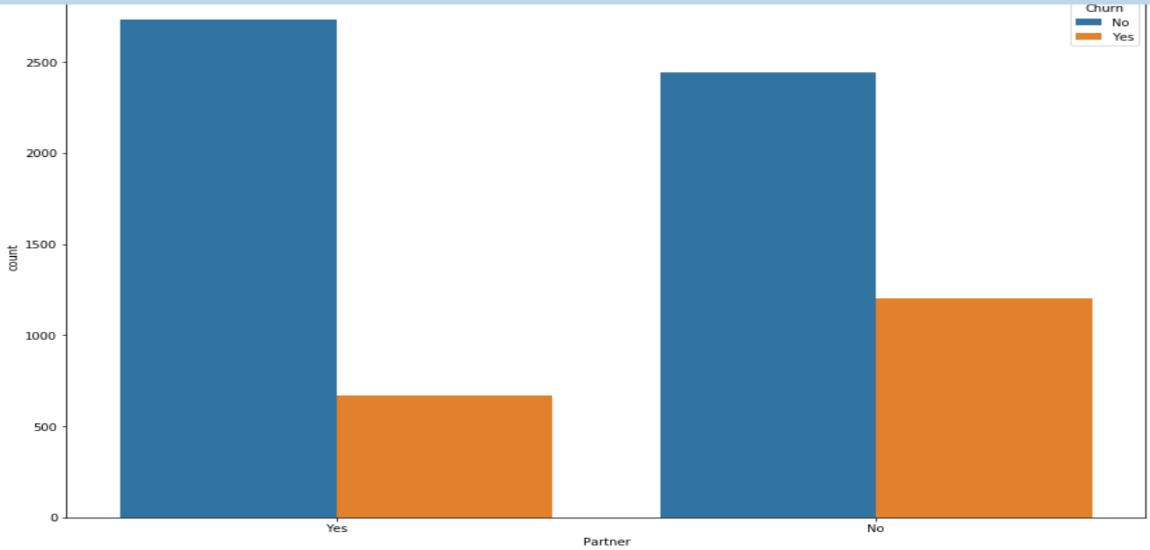
Handling Categorical columns

Categorical columns can be visualized using countplot as it provides the counts of observations in each categorical bin using bars.

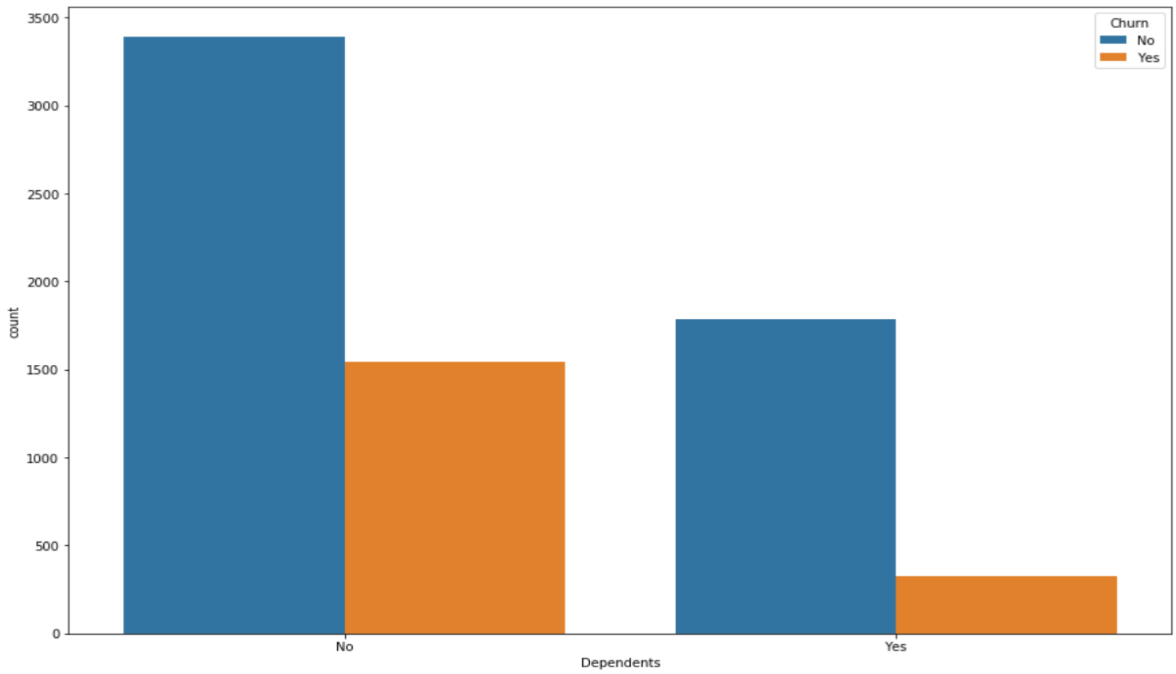
```
plt.figure(figsize=(15,10),facecolor="white")
sns.countplot(x="gender",data=df,hue="Churn")
```



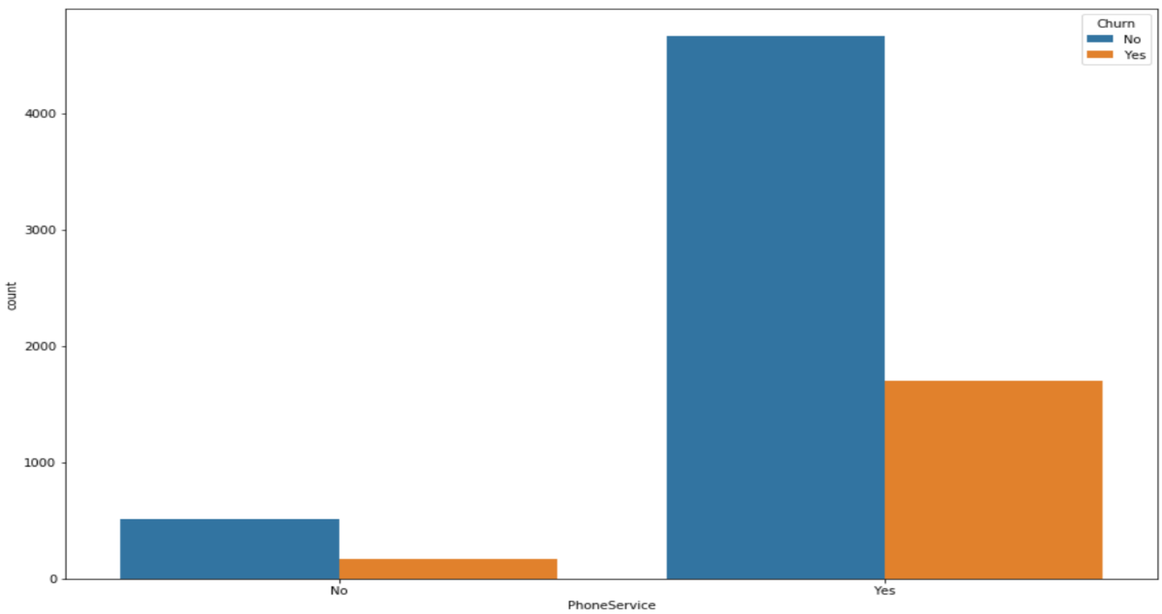

```
plt.figure(figsize=(15,10),facecolor="white")
sns.countplot(x="Partner",data=df,hue="Churn")
```



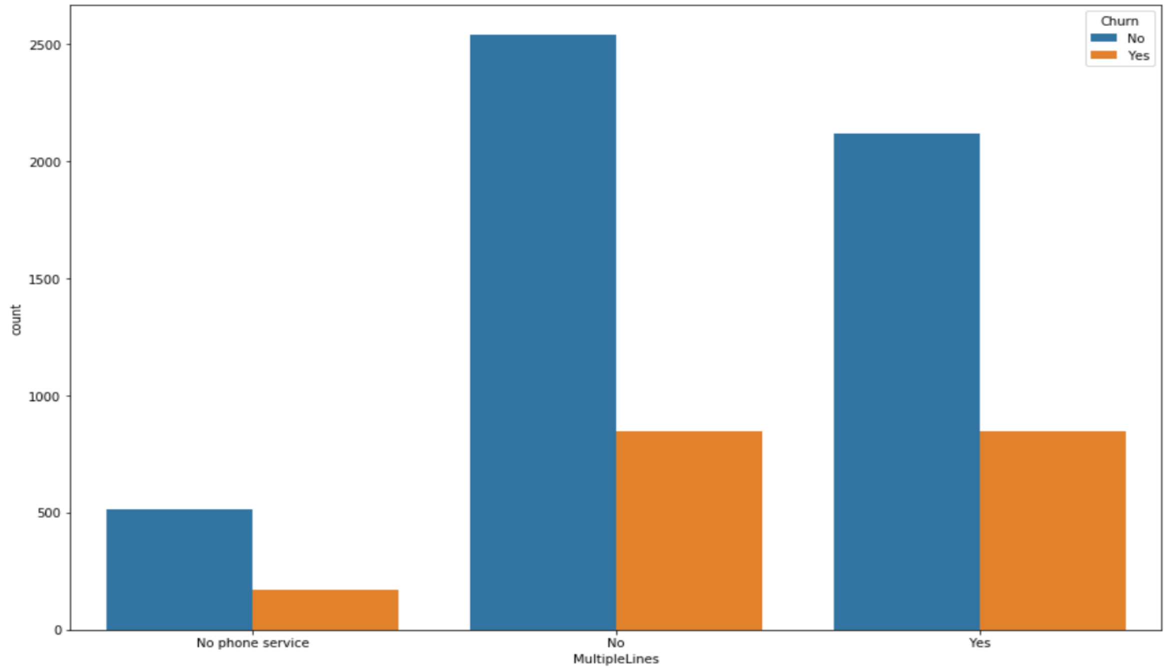

```
plt.figure(figsize=(15,10),facecolor="white")
sns.countplot(x="Dependents",data=df,hue="Churn")
```



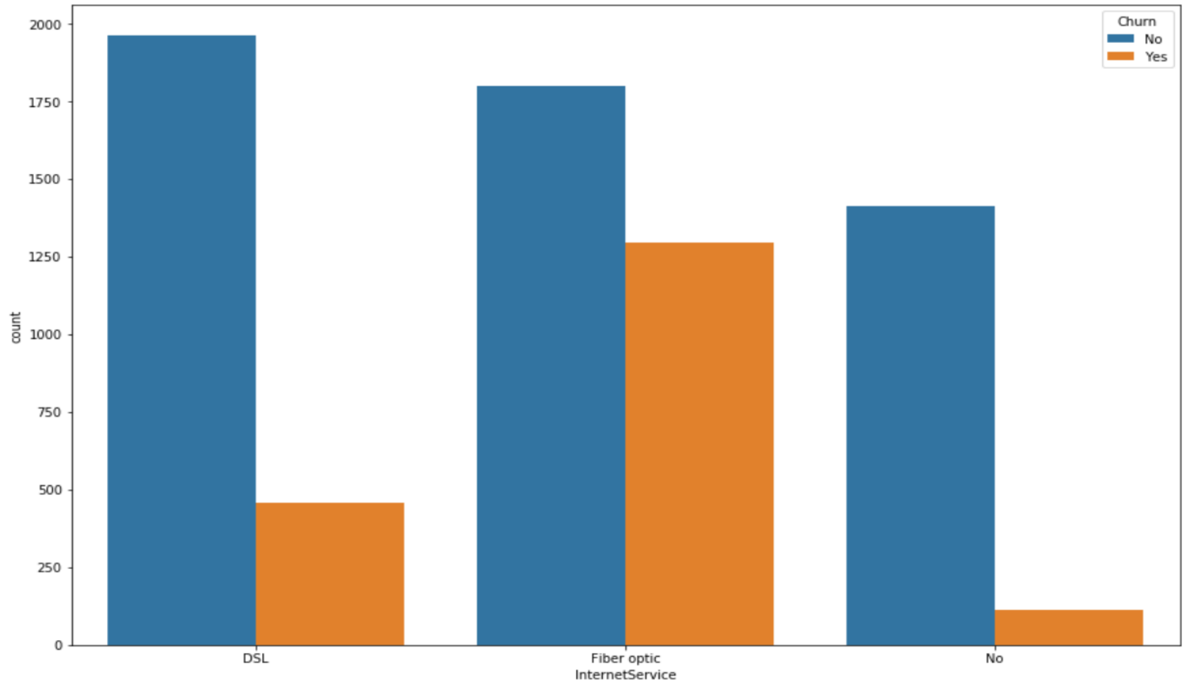

```
plt.figure(figsize=(15,10),facecolor="white")
sns.countplot(x="PhoneService",data=df,hue="Churn")
```



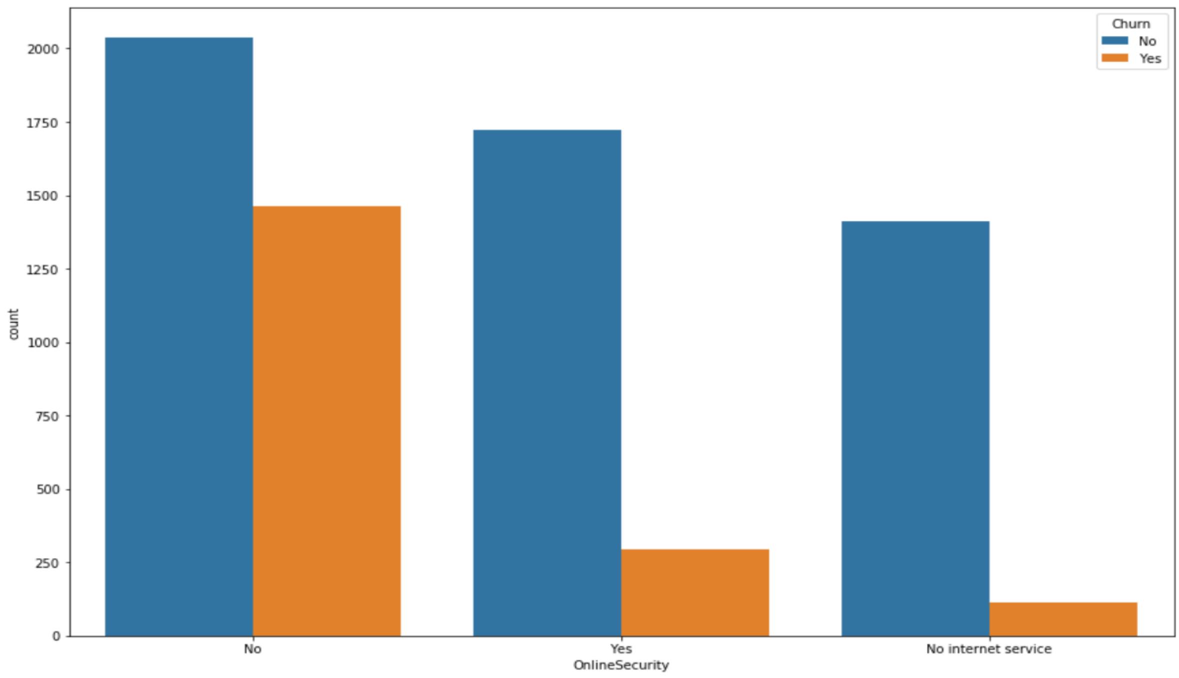

```
plt.figure(figsize=(15,10),facecolor="white")
sns.countplot(x="MultipleLines",data=df,hue="Churn")
```



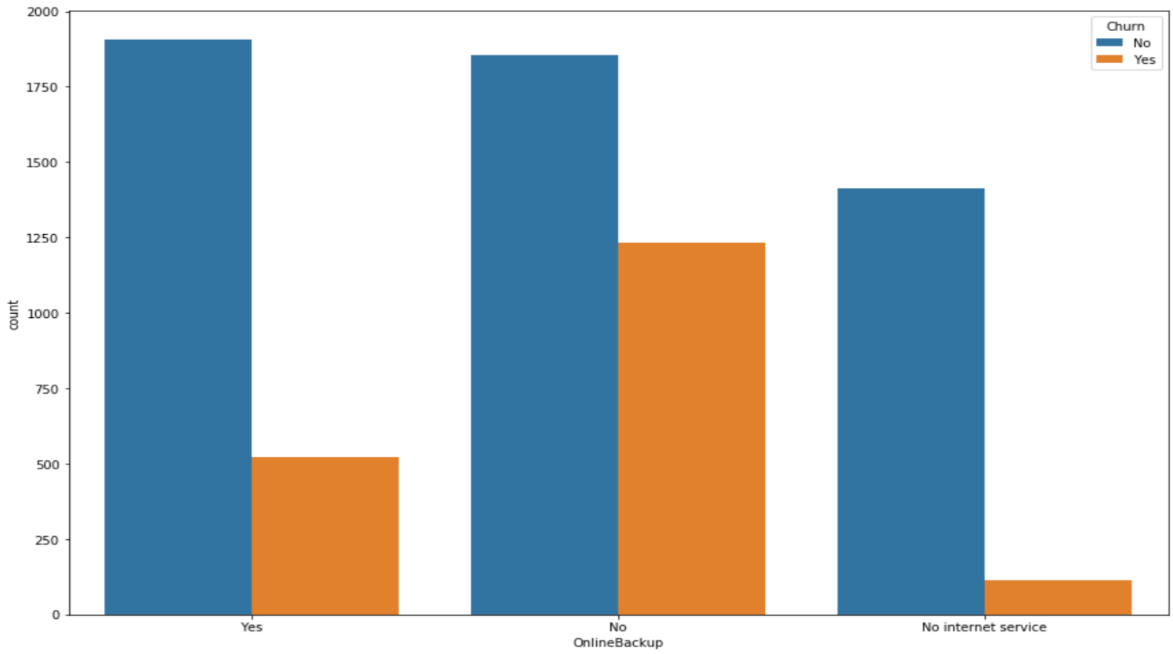

```
plt.figure(figsize=(15,10),facecolor="white")
sns.countplot(x="InternetService",data=df,hue="Churn")
```



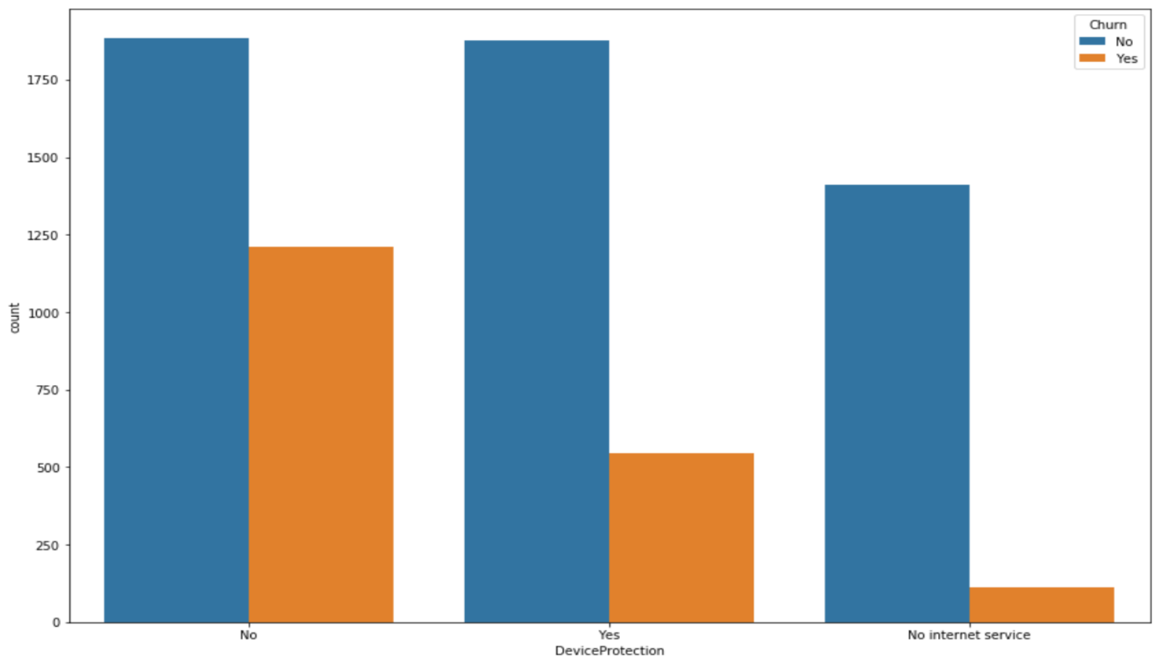

```
plt.figure(figsize=(15,10),facecolor="white")
sns.countplot(x="OnlineSecurity",data=df,hue="Churn")
```



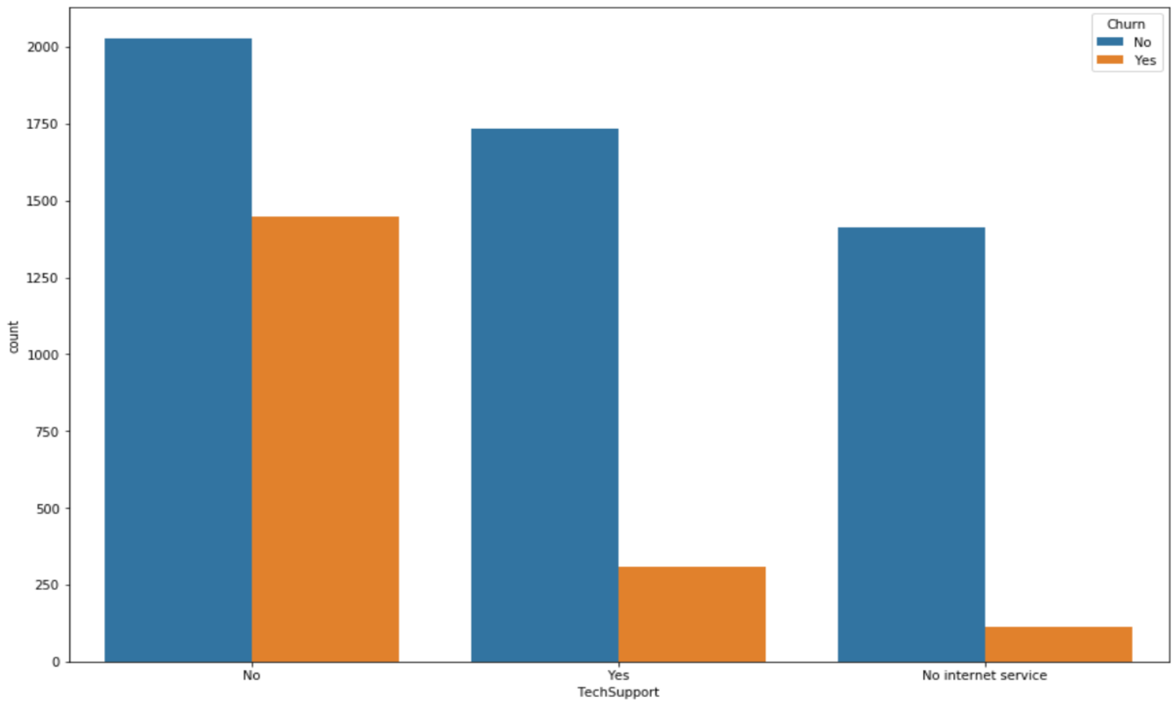

```
plt.figure(figsize=(15,10),facecolor="white")
sns.countplot(x="OnlineBackup",data=df,hue="Churn")
```



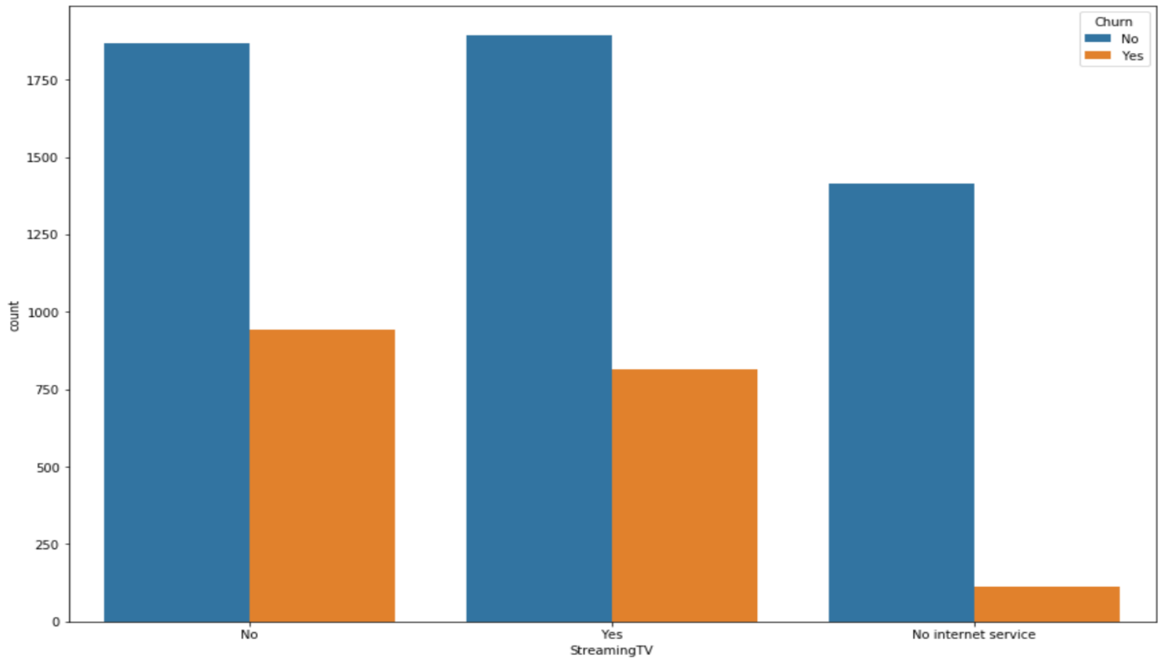

```
plt.figure(figsize=(15,10),facecolor="white")
sns.countplot(x="DeviceProtection",data=df,hue="Churn")
```



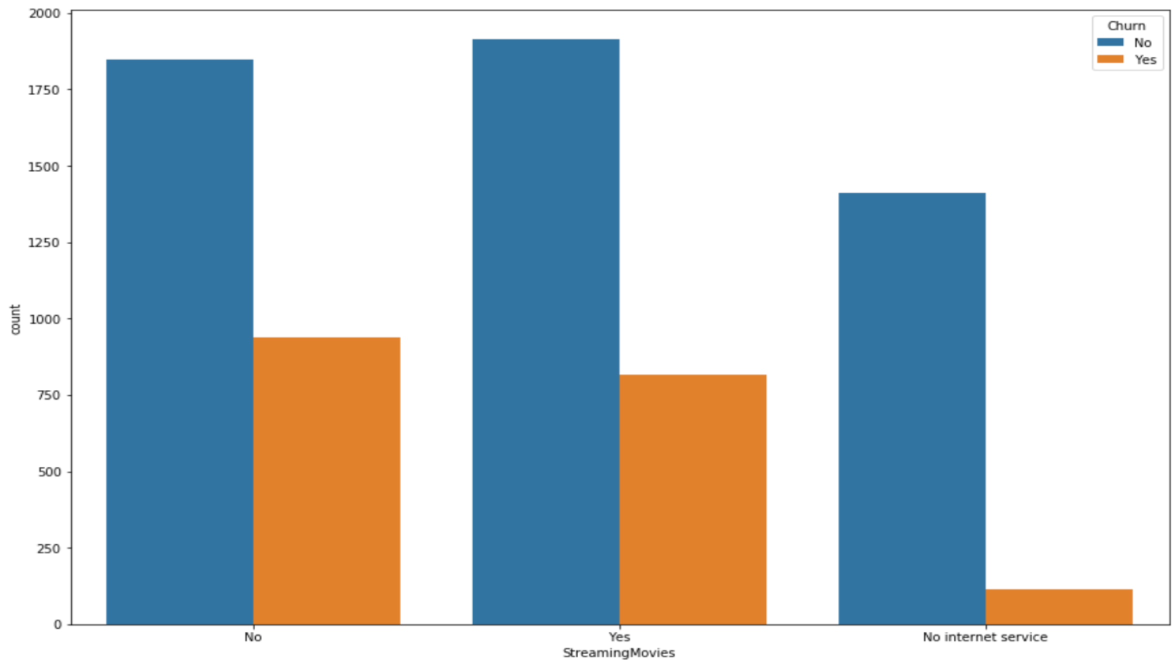

```
plt.figure(figsize=(15,10),facecolor="white")
sns.countplot(x="TechSupport",data=df,hue="Churn")
```



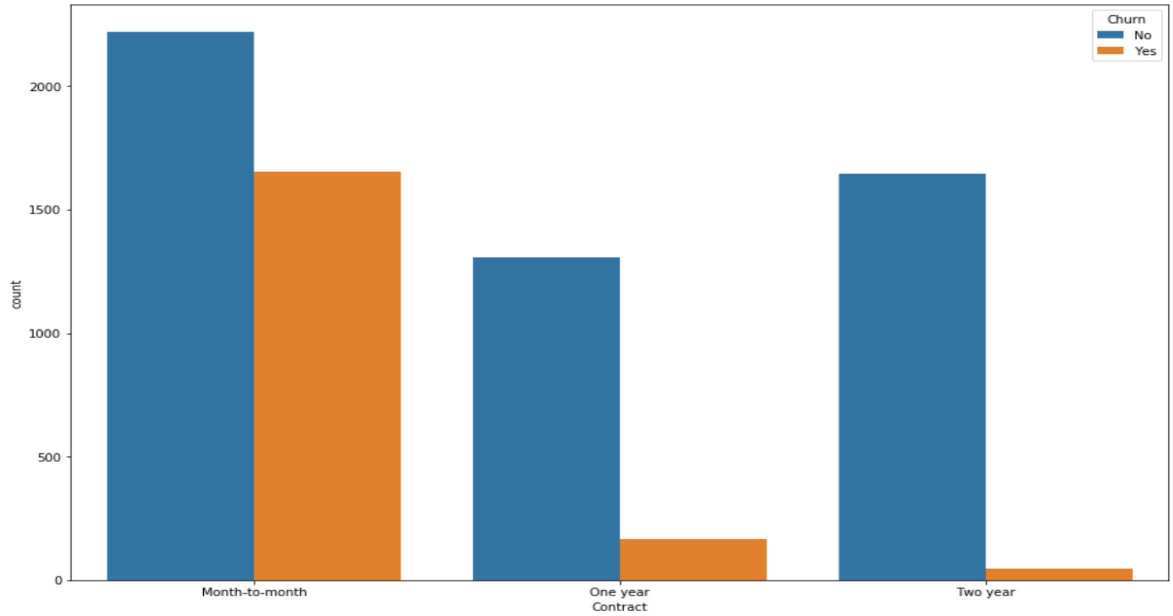

```
plt.figure(figsize=(15,10),facecolor="white")
sns.countplot(x="StreamingTV",data=df,hue="Churn")
```



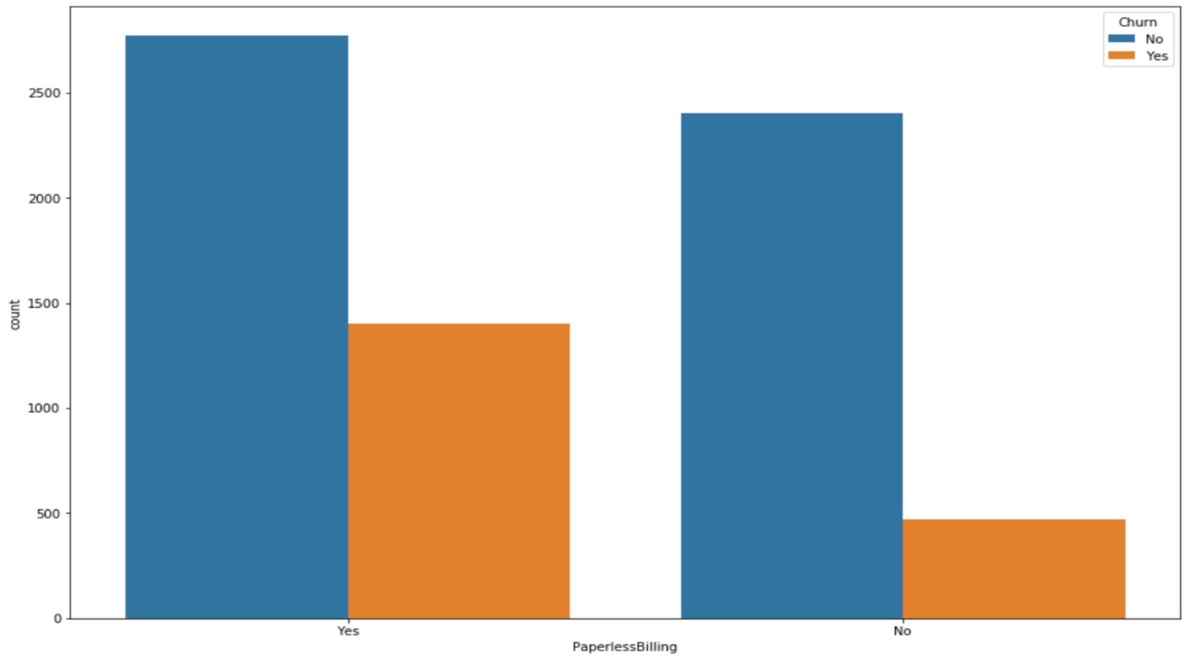

```
plt.figure(figsize=(15,10),facecolor="white")
sns.countplot(x="StreamingMovies",data=df,hue="Churn")
```



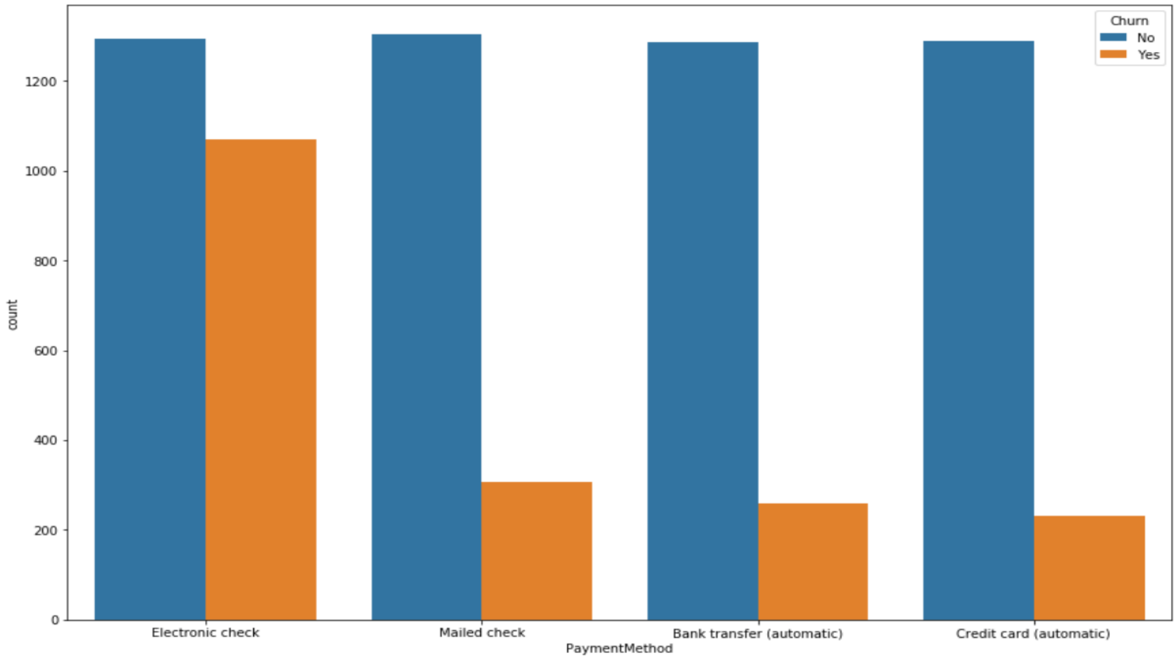

```
plt.figure(figsize=(15,10),facecolor="white")
sns.countplot(x="Contract",data=df,hue="Churn")
```



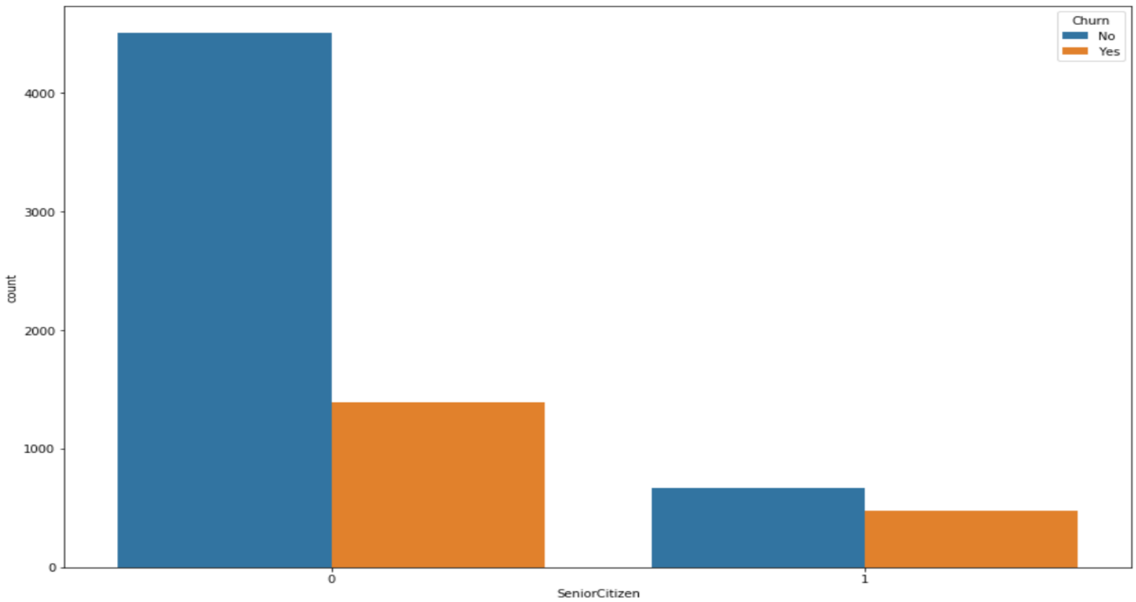

```
plt.figure(figsize=(15,10),facecolor="white")
sns.countplot(x="PaperlessBilling",data=df,hue="Churn")
```




```
plt.figure(figsize=(15,10),facecolor="white")
sns.countplot(x="PaymentMethod",data=df,hue="Churn")
```



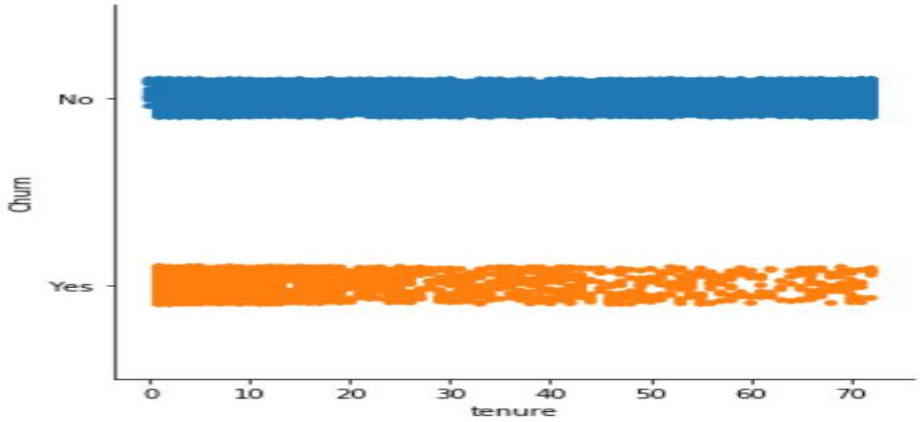

```
plt.figure(figsize=(15,10),facecolor="white")
sns.countplot(x="SeniorCitizen",data=df,hue="Churn")
```

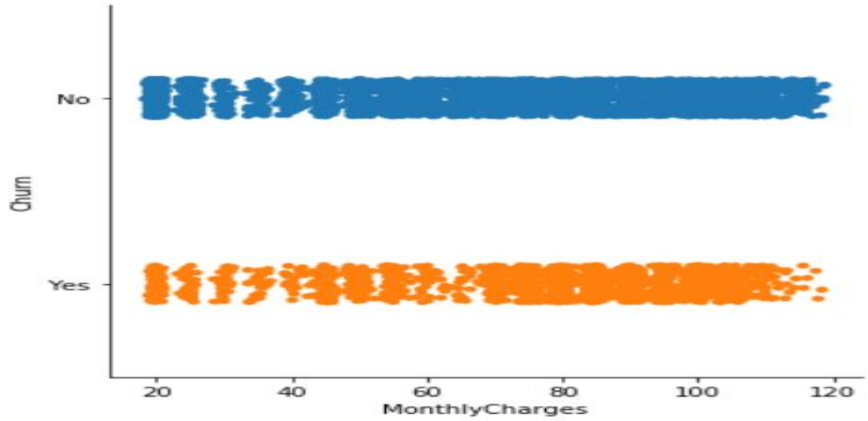


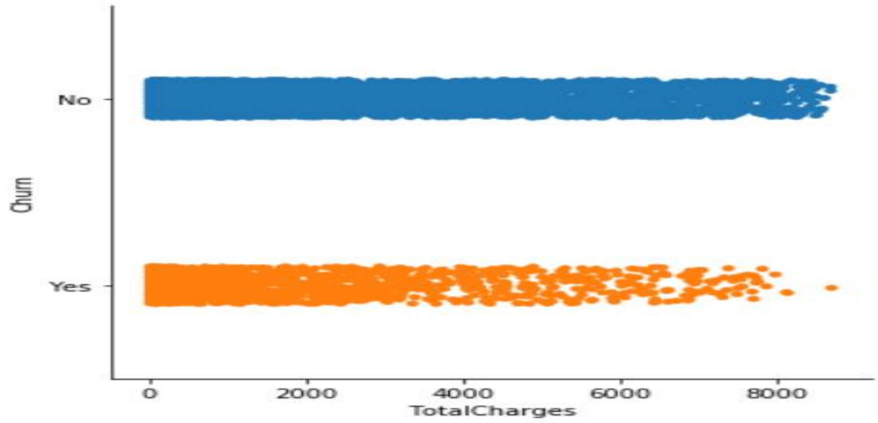
Using the countplot it can be noticed that columns "PhoneService", “MultipleLines”, "SeniorCitizen", have class imbalance problem which can be resolved using undersampling or oversampling technique.

Catplot

```
sns.catplot(x="tenure",y="Churn",data=df)
sns.catplot(x="MonthlyCharges",y="Churn",data=df)
sns.catplot(x="TotalCharges",y="Churn",data=df)
```







Correlation

Correlation is really important to identify which columns are affecting the target column positively or negatively. Through correlation it can also be identified that which columns are strongly related to target.

```
df.corr()
```

For plotting correlation heatmap can be used to understand the correlation more easily.

```
import matplotlib.pyplot as plt
plt.figure(figsize=(30,15))
sns.heatmap(df.corr(),annot=True,linewidths=0.1,linecolor="black",fmt=".2f")
```

Through correlation it can be identified that **tenure, Contract** have strong negative correlation with **churn.**

EDA Concluding Remark:

- Dataset contains 7043 rows and 21 columns with no missing values
- Dataset comprise of column of categorical as well as numerical type.
- It can be observed that no missing values can be identified
- It can easily be noticed that “Totalcharges” have datatype as object but values are continuous so need to convert datatype.
- “customerID” will not add any significance to the model building process so it must be dropped.
- It can be identified that “tenure”,” MonthlyCharges”,”TotalCharges” are not equally distributed and are skewed.
- It can be identified that “tenure”,” MonthlyCharges”, do not have any outlier values.
- it can be noticed that columns "PhoneService", “MultipleLines”, "SeniorCitizen", have class imbalance problem which can be resolved using undersampling or oversampling technique.
- It can be identified that **tenure, Contract** have strong negative correlation with **churn.**
- MinMaxScaler can be used for the normalization of data.

Pre-Processing:

Converting the datatype of “Totalcharges” from object to float type

```
df["TotalCharges”] =df["TotalCharges"].str.strip()
df["TotalCharges"]=pd.to_numeric(df["TotalCharges"])
```

Dropping the column “customerID”

```
df.drop(columns=["customerID"],inplace=True)
```

To handle the numerical and categorical attributes differently. Numerical attributes need to be scaled, whereas for categorical columns missing values need to be filled and then encode the categorical values into numerical values.

We have lot of features which are categorical and for building a model, categorical column need to encoded. OrdicalEncoder can be used to encode the categorical columns

```
from sklearn.preprocessing import OrdinalEncoder
enc=OrdinalEncoder()
for i in df.columns:
  if df[i].dtypes=="object":
    df[i]=enc.fit_transform(df[i].values.reshape(-1,1))
```

Before proceeding to model fitting, data transformation must be performed. Properly formatted and validated data improves data quality. Data transformation allow scaling each feature to a given range. For the given dataset MinMaxScaler can be used which scales all the data features in the range [0,1] or else in the range [-1,1].

Here, MinMaxScaler will be used to normalize the data column by column.

```
from sklearn.preprocessing import MinMaxScaler
scale=MinMaxScaler()
df["tenure"]=scale.fit_transform(df["tenure"].values.reshape(-1,1))
```

```
from sklearn.preprocessing import MinMaxScaler
scale=MinMaxScaler()
df["MonthlyCharges"]=scale.fit_transform(df["MonthlyCharges"].values.reshape(-1,1))
```

```
from sklearn.preprocessing import MinMaxScaler
scale=MinMaxScaler()
df["TotalCharges"]=scale.fit_transform(df["TotalCharges"].values.reshape(-1,1))
```

Building Machine Learning Models:

Model Building involves splitting of original dataset into input(x) and output (y) columns.

```
y=df["Churn"]
x=df.drop("Churn",axis=1)
```

x and y will be used to divide the dataset into training dataset and testing dataset.

Training dataset can be used for model fitting and statistical method can be used to estimate the accuracy of the model using the unseen data. Testing set can be used for evaluating the model accuracy.

Loaded dataset will be spitted into two, 70% of which will be used to train and 30% will be used to test the model.

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report, roc_auc_score
from sklearn.metrics import confusion_matrix, roc_auc_score
from sklearn.model_selection import train_test_split
import warnings
warnings. filterwarnings("ignore")
```

Random state allows to generate a reproducible split. Scikit-learn use random permutations to generate the splits. Random state provided is a seed to generate random number and also ensure that generated numbers are in same order.

Maximum accuracy_score can be used to identify the best random state from 1 to 200 and will proceed with same random state to fit the model using different algorithms.

```
Maxr2=0
BestRs=0
for i in range(1,200):
    xtrain,xtest,ytrain,ytest=train_test_split(x,y,test_size=.30,random_state=i)
    lr=LogisticRegression()
    lr.fit(xtrain,ytrain)
    pred=lr.predict(xtest)
    accu=accuracy_score(pred,ytest)
    print(accu)
    roc_score=roc_auc_score(pred,ytest)
    if roc_score>Maxr2:
        Maxr2=roc_score
        BestRs=i
print("with random state as",BestRs,"max accuracy is",Maxr2)
```

```
with random state as 80 max accuracy is 0.7773657420152327
```

So maximum roc_auc_score is .7773 which mean 77.73% accuracy at random state 80.

Following are the metrics we'll use to evaluate our predictive accuracy:

- Sensitivity = True Positive Rate (TP/TP+FN) – It says, 'out of all the positive (majority class) values, how many have been predicted correctly'.
- Specificity = True Negative Rate (TN/TN +FP) – It says, 'out of all the negative (minority class) values, how many have been predicted correctly'.
- Precision = (TP/TP+FP)
- Recall = Sensitivity
- F score = 2 * (Precision * Recall)/ (Precision + Recall) – It is the harmonic mean of precision and recall. It is used to compare several models side-by-side. Higher the better.

Logistic Regression Model

So we will fit logistic regression model with random state=80

```
xtrain,xtest,ytrain,ytest=train_test_split(x,y,test_size=.30,random_state=80)
lr=LogisticRegression()
lr.fit(xtrain,ytrain)
pred=lr.predict(xtest)
accu=accuracy_score(pred,ytest)
print(classification_report(pred,ytest))
print(accu)
print(confusion_matrix(pred,ytest))
print(roc_auc_score(pred,ytest))
```

```
              precision    recall  f1-score   support

         0.0       0.92      0.84      0.88      1669
         1.0       0.55      0.71      0.62       444

    accuracy                           0.82      2113
   macro avg       0.73      0.78      0.75      2113
weighted avg       0.84      0.82      0.82      2113

accuracy score = 0.8154283009938476

confusion matrix = [[1407  262]
                    [ 128  316]]

roc_auc_score = 0.7773657420152327
```

Outcome of the logistic regression model gave us accuracy score (0.815) and roc_auc_score as .777

Random Forest Model

So, fitting a Random Forest model with random state 80

```
from sklearn.ensemble import RandomForestClassifier

RFC=RandomForestClassifier()
RFC.fit(xtrain,ytrain)
pred=RFC.predict(xtest)
accu=accuracy_score(pred,ytest)
print(classification_report(pred,ytest))
print(accu)
print(confusion_matrix(pred,ytest))
print(roc_auc_score(pred,ytest))
```

```
              precision    recall  f1-score   support

         0.0       0.91      0.82      0.86      1692
         1.0       0.48      0.67      0.56       421

    accuracy                           0.79      2113
   macro avg       0.70      0.74      0.71      2113
weighted avg       0.82      0.79      0.80      2113

accuracy score = 0.792238523426408

confusion matrix = [[1394  298]
                    [ 141  280]]

roc_auc_score = 0.7444801019749219
```

Outcome of the Random forest classifier model gave us accuracy score (0.792) and roc_auc_ score is .744 .

Decision Tree Model

So, fitting a Decision Tree model with random state 80

```
from sklearn.tree import DecisionTreeClassifier

dtc=DecisionTreeClassifier()
dtc.fit(xtrain,ytrain)
pred=dtc.predict(xtest)
accu=accuracy_score(pred,ytest)
print(classification_report(pred,ytest))
print(accu)
print(confusion_matrix(pred,ytest))
```

```
print(roc_auc_score(pred,ytest))
```

```
              precision    recall  f1-score   support

         0.0       0.81      0.81      0.81      1521
         1.0       0.51      0.49      0.50       592

    accuracy                           0.72      2113
   macro avg       0.66      0.65      0.65      2113
weighted avg       0.72      0.72      0.72      2113

accuracy score = 0.7236157122574538

confusion matrix = [[1236  285]
                   [ 299  293]]

roc_auc_score = 0.653777853297084
```

Outcome of the Decision Tree Classifier model gave us accuracy score (0.723) and roc_auc_score as 0 .653 .

Support Vector Machine Model

So, fitting a support vector machine classifier model with random state 80

```
from sklearn.svm import SVC

svc=SVC()
svc.fit(xtrain,ytrain)
pred=svc.predict(xtest)
accu=accuracy_score(pred,ytest)
print(classification_report(pred,ytest))
print(accu)
print(confusion_matrix(pred,ytest))
print(roc_auc_score(pred,ytest))
```

```
              precision    recall  f1-score   support

         0.0       0.93      0.82      0.87      1735
         1.0       0.47      0.71      0.56       378

    accuracy                           0.80      2113
   macro avg       0.70      0.77      0.72      2113
weighted avg       0.85      0.80      0.82      2113

accuracy score = 0.8021769995267393

confusion matrix = [[1426  309]
                   [ 109  269]]
```

```
0.7667711144656388
```

Outcome of the support vector classifier model gave us accuracy score (0.802) and auc_roc_score (0.766)

Cross validation

Cross Validation is a technique for assessing how the statistical analysis generalises to an independent dataset. It is a technique for evaluating machine learning models by training several models on subsets of the available input data and evaluating them on the complementary subset of the data

For a k fold cross-validation, k refers to the number of groups a given data sample is to be split into and the procedure is often called k-fold cross-validation. When a specific value for k is chosen, then it may be used such as k=5

```
from sklearn.model_selection import cross_val_score
model=[lr,dtc,RFC,svc]

for model in model:
   print(cross_val_score(model,x,y,cv=5,scoring="roc_auc").mean())
```

```
0.8436054857675283
0.652881963774847
0.8216564299596982
0.804040467355892
```

After comparing the roc_auc_score and cross_val_score for logistic regression, decision tree classifier, random forest classifier and support vector classifier, it can be noticed that difference in roc_auc_score and cross_val_score for SVC (Support Vector Classifier) is lowest implying that Support Vector Classifier model is more accurate for prediction.

So we will proceed with Support Vector Classifier and will try to tune hyperparameter using GridSearchCV

Hyperparameter tuning

Hyperparameter is a parameter which control the learning rate. Hyperparameters tuning is crucial as they control the overall behavior of a machine learning model. Every machine learning models will have different hyperparameters that can be set. A hyperparameter is a parameter whose value is set before the learning process begins

Tuning the parameters of the Support Vector Classifier in order to obtain the best possible parameters for model building.

```
from sklearn.model_selection import GridSearchCV
```

```
parameter={"kernel":["linear","poly","rbf","sigmoid"],"gamma":["scale","auto"]}
grid=GridSearchCV(estimator=svc,param_grid=parameter,cv=5)
grid.fit(xtrain,ytrain)
print(grid.best_score_)
print(grid.best_params_)
```

```
Best score = 0.79026369168357

Best parameter = {'gamma': 'scale', 'kernel': 'linear'}
```

Using gridsearchcv best parameters are identified which can used to develop final Support Vector Classifier model. Best score which is obtained using the best parameters is .7902 .

So we will be developing final model using the best parameters.

```
{'gamma': 'scale', 'kernel': 'linear'}
```

Final Model

```
finalsvc=SVC(kernel='linear',gamma="scale")
finalsvc.fit(xtrain,ytrain)
pred=finalsvc.predict(xtest)
print("classification_report :",classification_report(pred,ytest))
print("accuracy_score :",accuracy_score(pred,ytest))
print("confusion_matrix :",confusion_matrix(pred,ytest))
print("roc_auc_score :",roc_auc_score(pred,ytest))
```

```
              precision    recall  f1-score   support

         0.0       0.90      0.84      0.87      1644
         1.0       0.54      0.67      0.60       469

    accuracy                           0.80      2113
   macro avg       0.72      0.75      0.73      2113
weighted avg       0.82      0.80      0.81      2113

accuracy_score : 0.8017037387600567

confusion_matrix : [[1380  264]
                   [ 155  314]]

roc_auc_score : 0.7544628266384449
```

Final model which is finalsvc has an accuracy_score of .8017 and roc_auc_score of .7544. Now we will be saving the final model and 75.44% accuracy means that it can be predicted that a customer will stop doing business with company or not *with 75.44% accuracy*.

Saving the final model

Saving the model using joblib library

```
import joblib
joblib.dump(finalsvc,"customer_churn.obj")
```

Concluding Remarks:

Here by using sklearn, we have built a preliminary machine learning tool and we have used the roc_auc_score to select the support vector classifier model as the best model for predicting that a customer will stop doing business with company or not. Our final model has an accuracy of 75.44% using the parameter tuning. We can further try to improve the model prediction by working on preprocessing and roc_auc_score. Of course, there is always tools and analysis you can do further in order to make it more accurate, and better to use.