Customer Churn Prediction

Imports

In [ ]:
```
# Tools
import pandas as pd
import numpy as np
import tensorflow as tf
from sklearn.model_selection import GridSearchCV, train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, precision_score, recall_score

# Models
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from lifelines import KaplanMeierFitter, CoxPHFitter
from xgboost import XGBClassifier

# Visualisation
import matplotlib.pyplot as plt
from lime import lime_tabular
import seaborn as sns
```

In [ ]:
```
# Dataset Loading
dataset = pd.read_csv('dataset.csv').iloc[:, 1:]
```

In [ ]:
```
dataset.head()
```

Out[ ]:

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	Inte
0	Female	0	Yes	No	1	No	No phone service	
1	Male	0	No	No	34	Yes	No	
2	Male	0	No	No	2	Yes	No	
3	Male	0	No	No	45	No	No phone service	
4	Female	0	No	No	2	Yes	No	

In [ ]:
```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7043 entries, 0 to 7042
Data columns (total 20 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   gender            7043 non-null   object 
 1   SeniorCitizen     7043 non-null   int64  
 2   Partner           7043 non-null   object 
 3   Dependents        7043 non-null   object 
 4   tenure            7043 non-null   int64  
 5   PhoneService      7043 non-null   object 
 6   MultipleLines     7043 non-null   object 
 7   InternetService   7043 non-null   object 
 8   OnlineSecurity    7043 non-null   object 
 9   OnlineBackup      7043 non-null   object 
 10  DeviceProtection  7043 non-null   object 
 11  TechSupport       7043 non-null   object 
 12  StreamingTV       7043 non-null   object 
 13  StreamingMovies   7043 non-null   object 
 14  Contract          7043 non-null   object 
 15  PaperlessBilling  7043 non-null   object 
 16  PaymentMethod     7043 non-null   object 
 17  MonthlyCharges    7043 non-null   float64
 18  TotalCharges      7043 non-null   object 
 19  Churn             7043 non-null   object 
dtypes: float64(1), int64(2), object(17)
memory usage: 1.1+ MB
```

In [ ]:
```
dataset.describe()
```

Out[ ]:

	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

Data Cleaning

In [ ]:
```
dataset = dataset[dataset['TotalCharges'] != ' ']
dataset['TotalCharges'] = dataset['TotalCharges'].astype(float)
```

In [ ]:
```
dataset['MultipleLines'] = dataset['MultipleLines'].replace(to_replace='N
dataset.loc[:, ['OnlineSecurity', 'OnlineBackup', 'DeviceProtection', 'Te
```

In [ ]:
```
# Checking for Null Values
dataset.isna().sum()
```

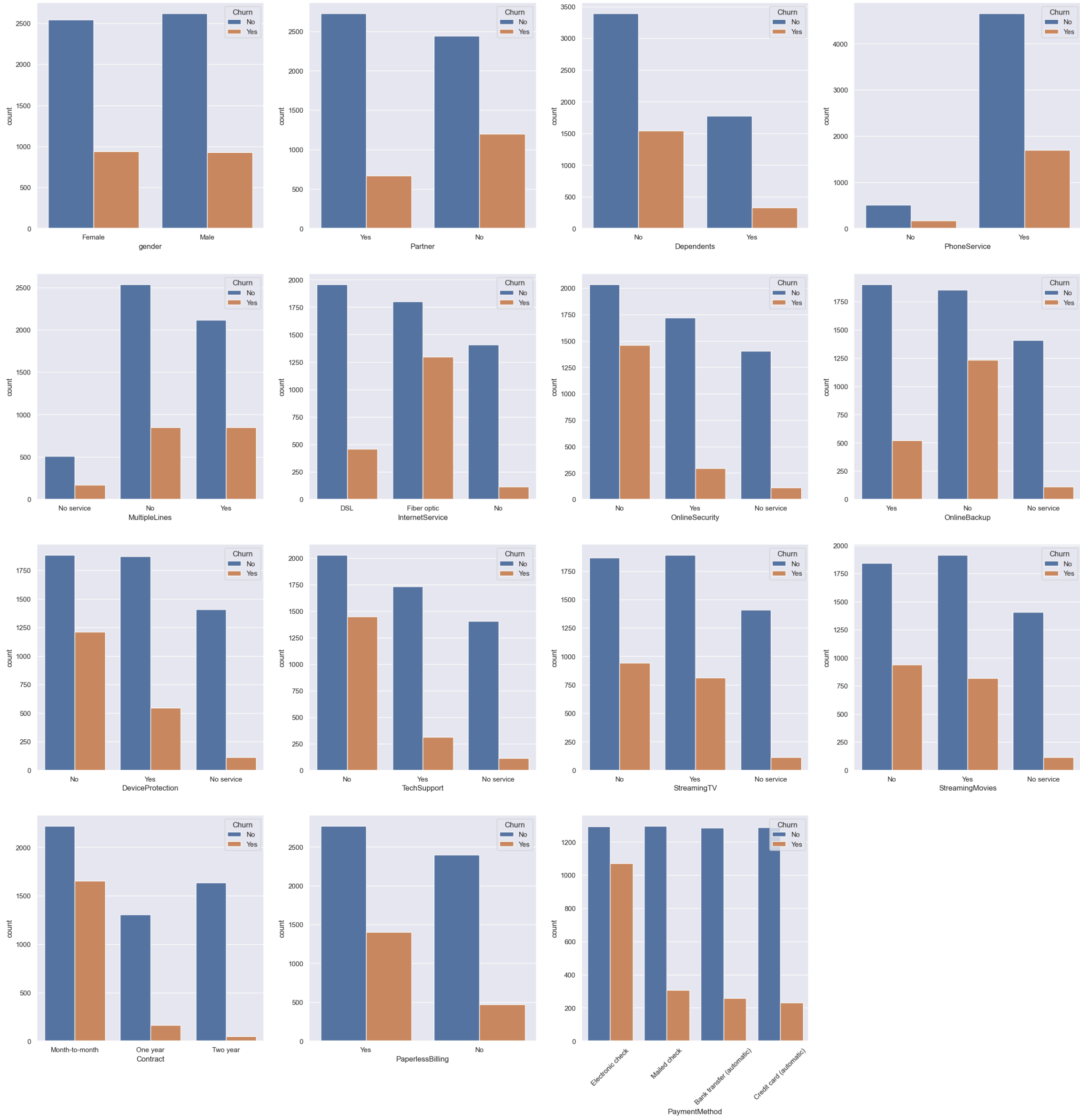
Out[ ]:
```
gender              0
SeniorCitizen       0
Partner             0
Dependents          0
tenure              0
PhoneService        0
MultipleLines       0
InternetService     0
OnlineSecurity      0
OnlineBackup        0
DeviceProtection    0
TechSupport         0
StreamingTV         0
StreamingMovies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

In [ ]:
```python
# Seperating Categorical and Numerical columns
categorical_columns = dataset.select_dtypes(include=['object']).columns.t
numerical_columns = [col for col in dataset.columns if col not in categor
```

Data Visualisation

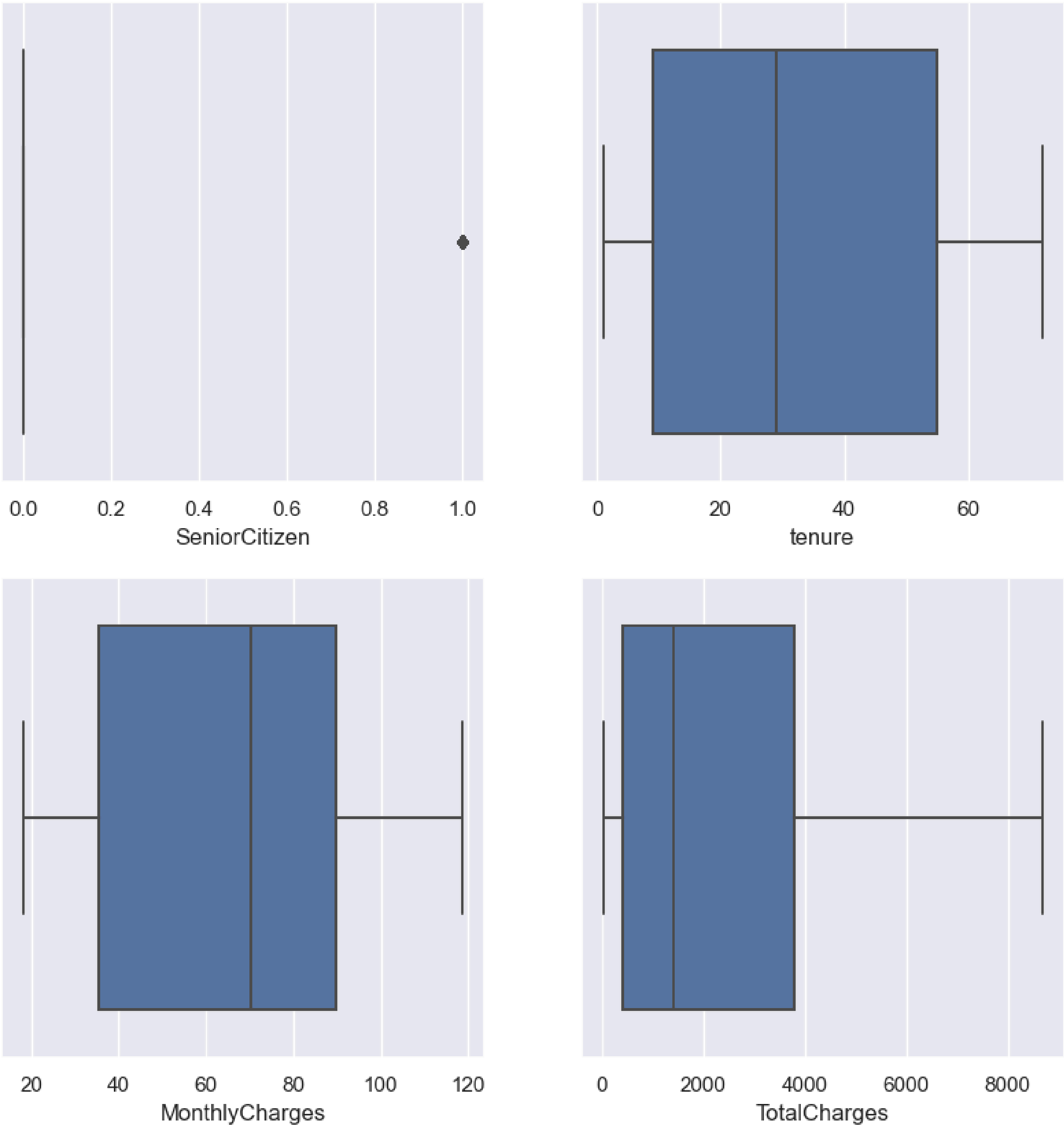
In [ ]:
```python
# Setting Seaborn Theme Style
sns.set_theme(style='darkgrid')
```

In [ ]:
```python
# Churn Distribution by Category
plt.figure(figsize=(30, 30))
for index, col in enumerate(categorical_columns):
    if col != 'Churn':
        plt.subplot(4, 4, index+1)
        sns.countplot(data=dataset, x=col, hue='Churn')
        if(len(dataset[col].unique()) > 3):
            plt.xticks(rotation=45)
plt.show()
```

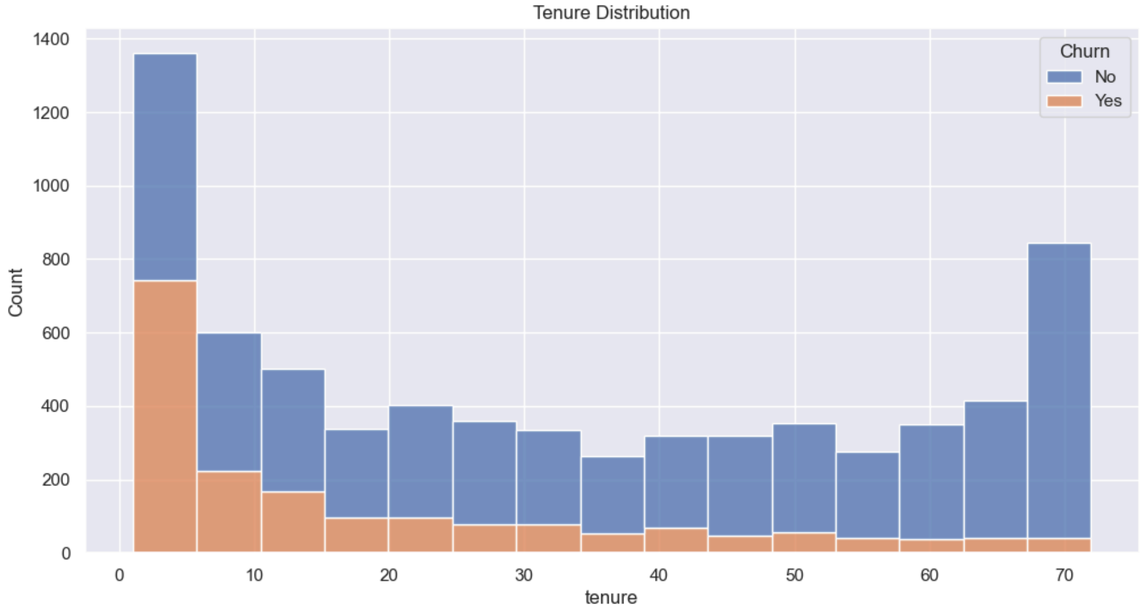


In [ ]:

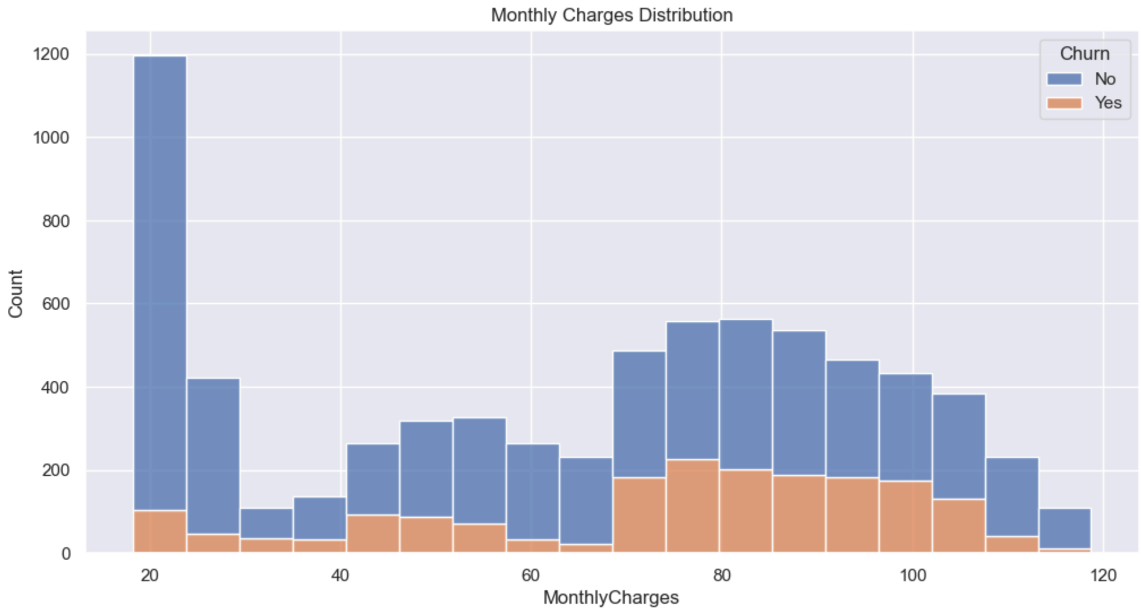
```python
# Numerical Columns Distribution
plt.figure(figsize=(10, 10))
for index, col in enumerate(numerical_columns):
    plt.subplot(2, 2, index+1)
    sns.boxplot(data=dataset, x=col)
plt.show()
```



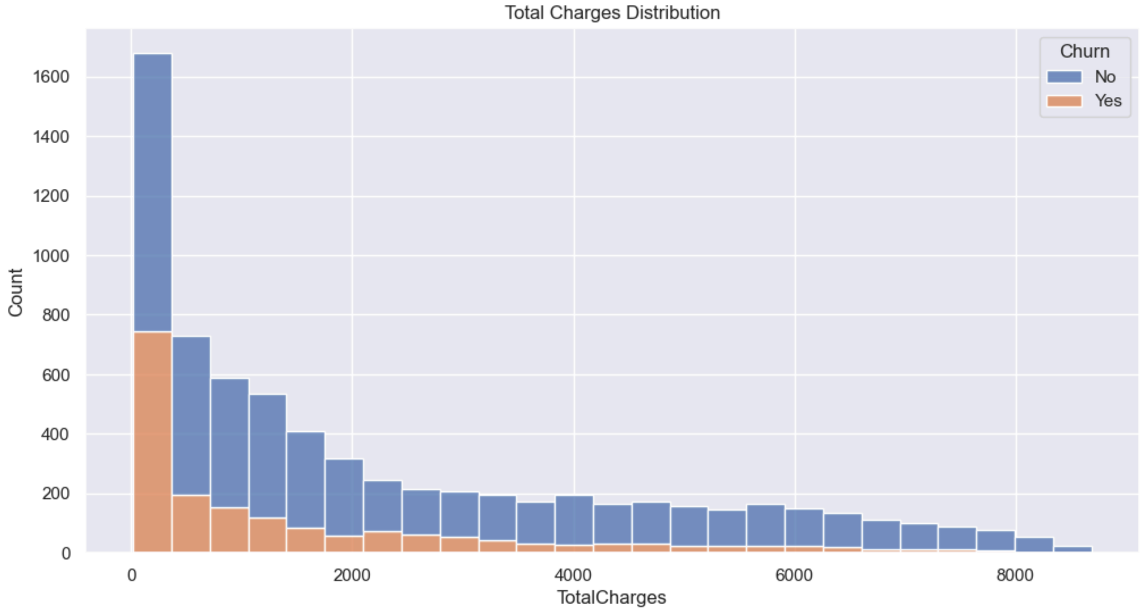
In [ ]:
```python
# Tenure Distribution
plt.figure(figsize=(12, 6))
sns.histplot(data=dataset, x='tenure', hue='Churn', multiple='stack')
plt.title('Tenure Distribution')
plt.show()
```



In [ ]:
```
# Monthly Charges Distribution
plt.figure(figsize=(12, 6))
sns.histplot(data=dataset, x='MonthlyCharges', hue='Churn', multiple='sta
plt.title('Monthly Charges Distribution')
plt.show()
```



In [ ]:
```
# Total Charges Distribution
plt.figure(figsize=(12, 6))
sns.histplot(data=dataset, x='TotalCharges', hue='Churn', multiple='stack
plt.title('Total Charges Distribution')
plt.show()
```



In [ ]:
```
# Extracting Unique Values for each Categorical Column
for col in categorical_columns:
    print(f'{col}: {dataset[col].unique()}')
```

```
gender: ['Female' 'Male']
Partner: ['Yes' 'No']
Dependents: ['No' 'Yes']
PhoneService: ['No' 'Yes']
MultipleLines: ['No service' 'No' 'Yes']
InternetService: ['DSL' 'Fiber optic' 'No']
OnlineSecurity: ['No' 'Yes' 'No service']
OnlineBackup: ['Yes' 'No' 'No service']
DeviceProtection: ['No' 'Yes' 'No service']
TechSupport: ['No' 'Yes' 'No service']
StreamingTV: ['No' 'Yes' 'No service']
StreamingMovies: ['No' 'Yes' 'No service']
Contract: ['Month-to-month' 'One year' 'Two year']
PaperlessBilling: ['Yes' 'No']
PaymentMethod: ['Electronic check' 'Mailed check' 'Bank transfer (automa
tic)'
 'Credit card (automatic)']
Churn: ['No' 'Yes']
```

Ordinal Encoding

In [ ]:
```
# Seperating Categorical Columns based on Mapping Values
categorical_columns_ordinal_1 = [
    'Partner',
    'Dependents',
    'PhoneService',
    'PaperlessBilling',
    'Churn'
]
categorical_columns_ordinal_2 = [
    'MultipleLines',
    'OnlineSecurity',
    'OnlineBackup',
    'DeviceProtection',
    'TechSupport',
    'StreamingTV',
```

```
    'StreamingMovies'
]
```

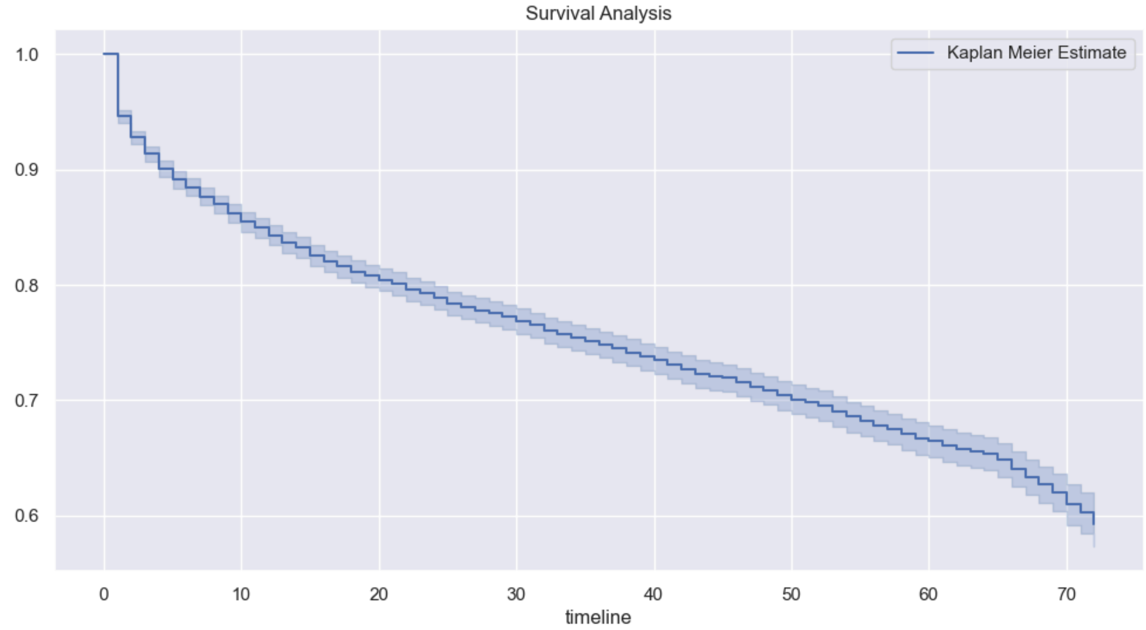
In [ ]:
```
# Mapping Values
ordinal_map_1 = {
    'Yes': 1,
    'No': 0
}
ordinal_map_2 = {
    'Yes': 1,
    'No': 0,
    'No service': -1
}
ordinal_map_3 = {
    'Fiber optic': 1,
    'DSL': 0,
    'No': -1
}
ordinal_map_4 = {
    'Two year': 2,
    'One year': 1,
    'Month-to-month': 0
}
ordinal_map_5 = {
    'Female': 1,
    'Male': 0
}
```

In [ ]:
```
# Mapping Values for Different Categorical Columns
for col in categorical_columns_ordinal_1:
    dataset[col] = dataset[col].map(ordinal_map_1)

for col in categorical_columns_ordinal_2:
    dataset[col] = dataset[col].map(ordinal_map_2)

dataset['InternetService'] = dataset['InternetService'].map(ordinal_map_3
dataset['Contract'] = dataset['Contract'].map(ordinal_map_4)
dataset['gender'] = dataset['gender'].map(ordinal_map_5)
```

Survival Analysis

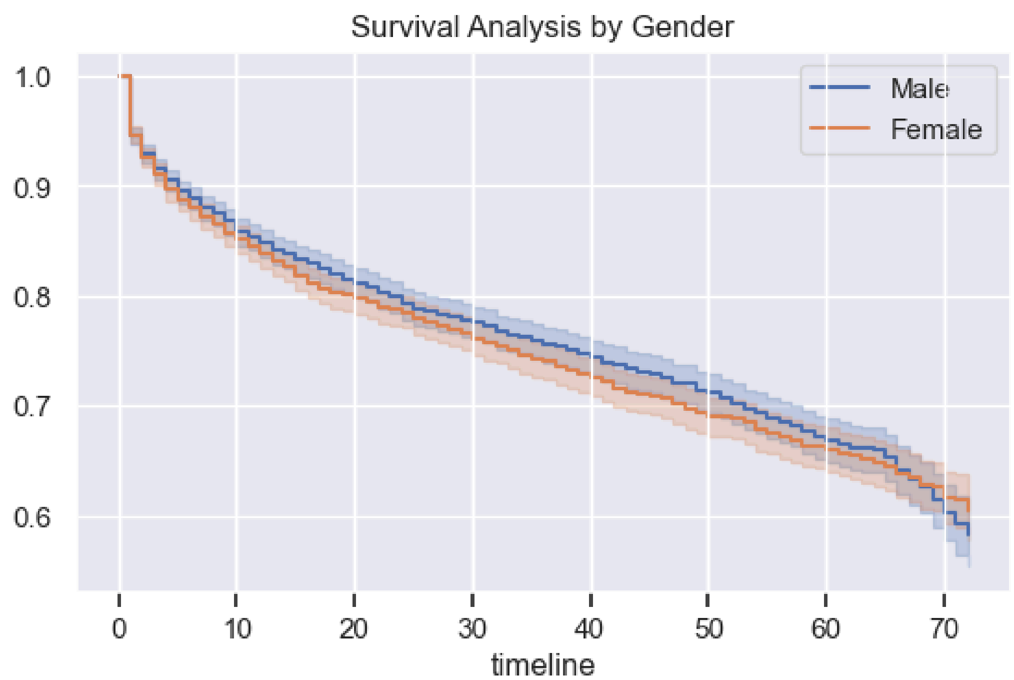
In [ ]:
```
# Kaplan Meier
km = KaplanMeierFitter()

tenure = dataset['tenure']
churn = dataset['Churn']
```

In [ ]:
```
# Kaplan Meier Estimate
plt.figure(figsize=(12, 6))
km.fit(tenure, churn,label='Kaplan Meier Estimate')
km.plot_survival_function()
plt.title('Survival Analysis')
plt.show()
```



In [ ]:

```
# Survival Analysis by Gender
ax = plt.subplot()
m = (dataset['gender'] == 0)
km.fit(durations = tenure[m], event_observed = churn[m], label = "Male")
km.plot_survival_function(ax = ax)
km.fit(tenure[~m], event_observed = churn[~m], label = "Female")
km.plot_survival_function(ax = ax, at_risk_counts = True)
plt.title('Survival Analysis by Gender')
plt.show()
```



Female								
At risk	3483	2519	2052	1684	1371	1022	674	261
Censored	0	472	787	1069	1307	1596	1906	2287
Events	0	492	644	730	805	865	903	935

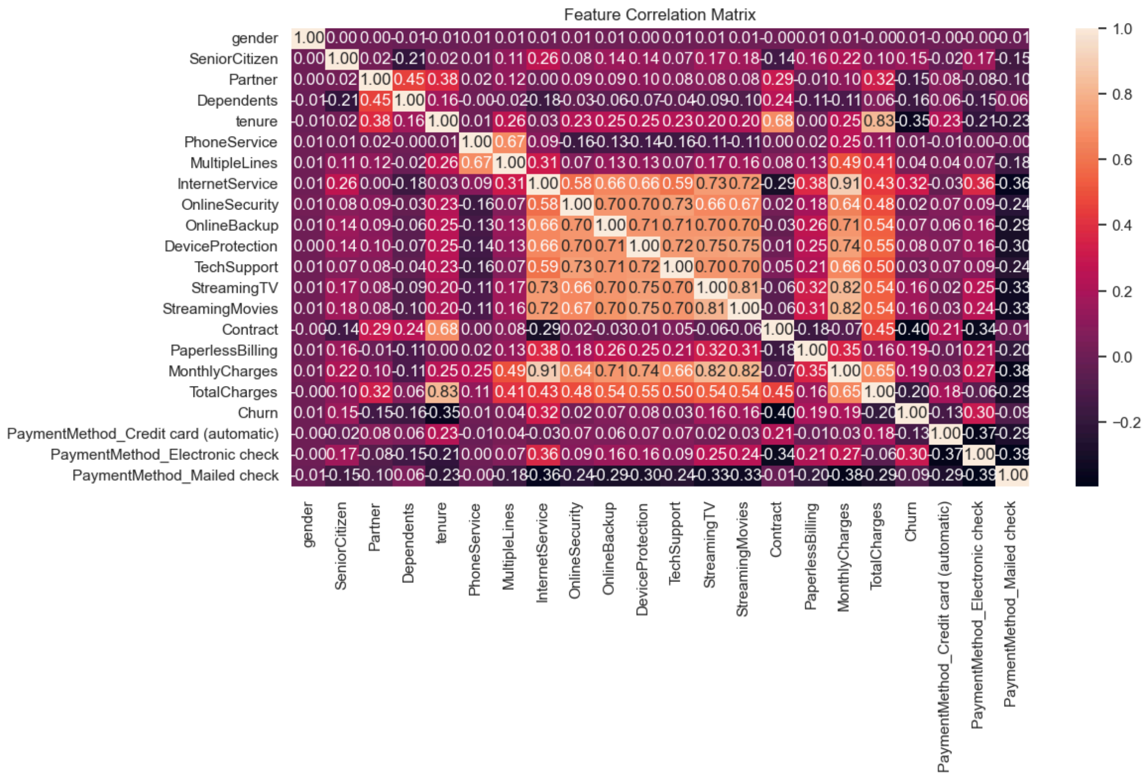
Label Encoding

In [ ]:
```
# Using Pandas Library to Encode Payment Method Column
dataset = pd.get_dummies(dataset, columns=['PaymentMethod'], drop_first=T
```

Correlation Heatmap

In [ ]:
```
# Correlation Matrix
correlation_matrix = dataset.corr()
```

In [ ]:
```
# Plotting Heatmap
plt.figure(figsize=(12, 6))
sns.heatmap(correlation_matrix, annot=True, fmt=".2f")
plt.title('Feature Correlation Matrix')
plt.show()
```



Cox Proportional Hazard Model

In [ ]:
```
cph = CoxPHFitter()
cph.fit(dataset, 'tenure', 'Churn')
cph.print_summary()
```

model	lifelines.CoxPHFitter
duration col	'tenure'
event col	'Churn'
baseline estimation	breslow
number of observations	7032
number of events observed	1869
partial log-likelihood	-12673.77
time fit was run	2024-04-25 20:27:07 UTC

	coef	**exp(coef)**	**se(coef)**	**coef lower 95%**	**coef upper 95%**	**exp(coef) lower 95%**	**exp(coel uppe 95%**
gender	0.03	1.04	0.05	-0.06	0.13	0.94	1.1
SeniorCitizen	0.03	1.03	0.06	-0.08	0.14	0.92	1.1
Partner	-0.18	0.83	0.06	-0.29	-0.08	0.75	0.9
Dependents	-0.09	0.92	0.07	-0.22	0.05	0.80	1.0
PhoneService	1.12	3.08	0.17	0.78	1.47	2.19	4.3
MultipleLines	0.20	1.22	0.06	0.08	0.32	1.09	1.3
InternetService	1.63	5.08	0.14	1.36	1.89	3.89	6.6
OnlineSecurity	-0.07	0.93	0.06	-0.19	0.05	0.83	1.0
OnlineBackup	0.08	1.08	0.05	-0.03	0.18	0.97	1.2
DeviceProtection	0.23	1.25	0.06	0.12	0.34	1.12	1.4
TechSupport	0.05	1.05	0.06	-0.07	0.18	0.93	1.1
StreamingTV	0.54	1.71	0.08	0.39	0.69	1.47	1.9
StreamingMovies	0.54	1.71	0.08	0.39	0.69	1.48	1.9
Contract	-1.61	0.20	0.08	-1.77	-1.45	0.17	0.2
PaperlessBilling	0.15	1.16	0.06	0.04	0.26	1.04	1.3
MonthlyCharges	-0.01	0.99	0.01	-0.02	-0.00	0.98	1.0
TotalCharges	-0.00	1.00	0.00	-0.00	-0.00	1.00	1.0
PaymentMethod_Credit card (automatic)	-0.01	0.99	0.09	-0.19	0.16	0.83	1.1
PaymentMethod_Electronic check	0.39	1.47	0.07	0.24	0.53	1.28	1.7
PaymentMethod_Mailed check	0.51	1.66	0.09	0.34	0.68	1.40	1.9

Concordance	0.93
Partial AIC	25387.55
log-likelihood ratio test	5958.53 on 20 df
-log2(p) of ll-ratio test	inf

Extracting Variables

In [ ]:
```
# Independent Variables
X = dataset.drop(columns=['Churn'], axis=1)

# Dependent Variable
y = dataset.loc[:, 'Churn']
```

Splitting Data

In [ ]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
```

Feature Scaling

In [ ]:
```
# Standard Scaler will Scale Values between -3 to +3
sc = StandardScaler()

X_train_scaled = sc.fit_transform(X_train)
X_test_scaled = sc.transform(X_test)
```

GridSearchCV

In [ ]:
```
# Models
models = {
    'LogisticRegression': LogisticRegression(max_iter=1000),
    'KNeighborsClassifier': KNeighborsClassifier(),
    'SVC': SVC(),
    'GaussianNB': GaussianNB(),
    'DecisionTreeClassifier': DecisionTreeClassifier(),
    'RandomForestClassifier': RandomForestClassifier()
}

# Model Parameters
param_grids = {
    'RandomForestClassifier': {
        'n_estimators': [100, 300, 500, 800, 1000],
        'max_depth': [10, 20, 30, None]
    },
    'SVC': {
        'C': [0.1, 1, 10],
        'gamma': ['scale', 'auto']
    },
    'LogisticRegression': {
        'C': [0.001, 0.01, 0.1, 1, 10, 100]
    },
    'KNeighborsClassifier': {
        'n_neighbors': [3, 5, 7, 9],
        'weights': ['uniform', 'distance'],
        'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute']
    },
    'DecisionTreeClassifier': {
        'max_depth': [None, 10, 20, 30],
        'min_samples_split': [2, 5, 10],
        'min_samples_leaf': [1, 2, 4]
```

```
    },
    'GaussianNB': {}
}

# Grid Search with Cross Validation
for key, value in models.items():
    grid_search = GridSearchCV(estimator=value, param_grid=param_grids[ke
    grid_search.fit(X=X_train_scaled, y=y_train)
```

GridSearch Results

In [ ]:
```
results = pd.DataFrame(grid_search.cv_results_)
relevant_columns = ['params', 'mean_test_accuracy', 'std_test_accuracy',
results = results.loc[:, relevant_columns]

results
```

Out[ ]:

	params	mean_test_accuracy	std_test_accuracy	mean_test_precision	std_tes
0	{'max_depth': 10, 'n_estimators': 100}	0.800889	0.011593	0.653024	
1	{'max_depth': 10, 'n_estimators': 300}	0.801601	0.010678	0.657279	
2	{'max_depth': 10, 'n_estimators': 500}	0.802312	0.012857	0.660438	
3	{'max_depth': 10, 'n_estimators': 800}	0.800889	0.010831	0.656056	
4	{'max_depth': 10, 'n_estimators': 1000}	0.800712	0.012561	0.656346	
5	{'max_depth': 20, 'n_estimators': 100}	0.787559	0.015453	0.619179	
6	{'max_depth': 20, 'n_estimators': 300}	0.790400	0.016606	0.628641	
7	{'max_depth': 20, 'n_estimators': 500}	0.791466	0.018031	0.630349	
8	{'max_depth': 20, 'n_estimators': 800}	0.791643	0.016300	0.630804	
9	{'max_depth': 20, 'n_estimators': 1000}	0.789155	0.020305	0.624870	
10	{'max_depth': 30, 'n_estimators': 100}	0.788622	0.013918	0.625051	
11	{'max_depth': 30, 'n_estimators': 300}	0.789511	0.015492	0.626874	
12	{'max_depth': 30, 'n_estimators': 500}	0.790222	0.017228	0.627363	
13	{'max_depth': 30,	0.789511	0.020096	0.626802	

	params	mean_test_accuracy	std_test_accuracy	mean_test_precision	std_tes
	'n_estimators': 800}				
14	{'max_depth': 30, 'n_estimators': 1000}	0.789690	0.017461	0.627121	
15	{'max_depth': None, 'n_estimators': 100}	0.789157	0.015753	0.625907	
16	{'max_depth': None, 'n_estimators': 300}	0.788266	0.019372	0.624414	
17	{'max_depth': None, 'n_estimators': 500}	0.791643	0.016571	0.631914	
18	{'max_depth': None, 'n_estimators': 800}	0.789866	0.017943	0.626633	
19	{'max_depth': None, 'n_estimators': 1000}	0.790044	0.020039	0.627586	

In [ ]:
```
best_score = grid_search.best_score_
best_params = grid_search.best_params_
model = grid_search.best_estimator_

print("Best Score:", best_score)
print("Best Parameters:", best_params)
print("Best Model:", model)
```

```
Best Score: 0.8023115870116243
Best Parameters: {'max_depth': 10, 'n_estimators': 500}
Best Model: RandomForestClassifier(max_depth=10, n_estimators=500)
```

Random Forest

In [ ]:
```
# Fitting Model
model.fit(X_train_scaled, y_train)

# Making Prediction
y_pred = model.predict(X_test_scaled)

# Metrics
model_accuracy = accuracy_score(y_pred, y_test)
model_precision = precision_score(y_pred, y_test)
model_recall = recall_score(y_pred, y_test)
model_cm = confusion_matrix(y_pred, y_test)
```

XGBoost

In [ ]:
```
# Initialising Model
xgb = XGBClassifier(
    n_estimators=100,
    learning_rate=0.01,
    max_depth=5,
    random_state=21
)

# Fitting Model
xgb.fit(X_train_scaled, y_train, eval_set=[(X_test_scaled, y_test)], verb

# Making Prediction
y_pred = xgb.predict(X_test_scaled)

# Metrics
xgb_accuracy = accuracy_score(y_pred, y_test)
xgb_precision = precision_score(y_pred, y_test)
xgb_recall = recall_score(y_pred, y_test)
xgb_cm = confusion_matrix(y_pred, y_test)
```

Artificial Neural Network

In [ ]:
```
# Initialising Model
ann = tf.keras.models.Sequential([
    tf.keras.layers.Dense(10, activation='relu'),
    tf.keras.layers.Dense(8, activation='relu'),
    tf.keras.layers.Dense(6, activation='relu'),
    tf.keras.layers.Dense(4, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])

# Compiling Model
ann.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accur

# Fitting Model
ann.fit(X_train, y_train, epochs=50, batch_size=32)

# Making Prediction
y_pred = ann.predict(X_test) > 0.5

# Metrics
ann_accuracy = accuracy_score(y_pred, y_test)
ann_precision = precision_score(y_pred, y_test)
ann_recall = recall_score(y_pred, y_test)
ann_cm = confusion_matrix(y_pred, y_test)
```

```
Epoch 1/50
176/176 [==============================] - 0s 684us/step - loss: 1.6228
- accuracy: 0.7061
Epoch 2/50
176/176 [==============================] - 0s 630us/step - loss: 0.5977
- accuracy: 0.7604
Epoch 3/50
176/176 [==============================] - 0s 631us/step - loss: 0.5569
- accuracy: 0.7664
Epoch 4/50
176/176 [==============================] - 0s 624us/step - loss: 0.5366
- accuracy: 0.7717
Epoch 5/50
176/176 [==============================] - 0s 621us/step - loss: 0.5256
- accuracy: 0.7730
Epoch 6/50
176/176 [==============================] - 0s 617us/step - loss: 0.5188
- accuracy: 0.7781
Epoch 7/50
176/176 [==============================] - 0s 618us/step - loss: 0.5147
- accuracy: 0.7801
Epoch 8/50
176/176 [==============================] - 0s 613us/step - loss: 0.5109
- accuracy: 0.7808
Epoch 9/50
176/176 [==============================] - 0s 615us/step - loss: 0.5041
- accuracy: 0.7851
Epoch 10/50
176/176 [==============================] - 0s 702us/step - loss: 0.5016
- accuracy: 0.7819
Epoch 11/50
176/176 [==============================] - 0s 614us/step - loss: 0.5017
- accuracy: 0.7813
Epoch 12/50
176/176 [==============================] - 0s 615us/step - loss: 0.5073
- accuracy: 0.7792
Epoch 13/50
176/176 [==============================] - 0s 609us/step - loss: 0.4973
- accuracy: 0.7838
Epoch 14/50
176/176 [==============================] - 0s 617us/step - loss: 0.4994
- accuracy: 0.7797
Epoch 15/50
176/176 [==============================] - 0s 609us/step - loss: 0.4972
- accuracy: 0.7803
Epoch 16/50
176/176 [==============================] - 0s 617us/step - loss: 0.5013
- accuracy: 0.7787
Epoch 17/50
176/176 [==============================] - 0s 616us/step - loss: 0.5054
- accuracy: 0.7808
Epoch 18/50
176/176 [==============================] - 0s 614us/step - loss: 0.4920
- accuracy: 0.7824
Epoch 19/50
176/176 [==============================] - 0s 615us/step - loss: 0.4979
- accuracy: 0.7756
Epoch 20/50
176/176 [==============================] - 0s 617us/step - loss: 0.4937
- accuracy: 0.7810
```

```
Epoch 21/50
176/176 [==============================] - 0s 616us/step - loss: 0.4958
- accuracy: 0.7824
Epoch 22/50
176/176 [==============================] - 0s 617us/step - loss: 0.5021
- accuracy: 0.7760
Epoch 23/50
176/176 [==============================] - 0s 611us/step - loss: 0.4833
- accuracy: 0.7865
Epoch 24/50
176/176 [==============================] - 0s 614us/step - loss: 0.4941
- accuracy: 0.7776
Epoch 25/50
176/176 [==============================] - 0s 615us/step - loss: 0.4872
- accuracy: 0.7838
Epoch 26/50
176/176 [==============================] - 0s 612us/step - loss: 0.4868
- accuracy: 0.7788
Epoch 27/50
176/176 [==============================] - 0s 721us/step - loss: 0.4767
- accuracy: 0.7879
Epoch 28/50
176/176 [==============================] - 0s 613us/step - loss: 0.4836
- accuracy: 0.7851
Epoch 29/50
176/176 [==============================] - 0s 613us/step - loss: 0.4804
- accuracy: 0.7803
Epoch 30/50
176/176 [==============================] - 0s 615us/step - loss: 0.4771
- accuracy: 0.7803
Epoch 31/50
176/176 [==============================] - 0s 614us/step - loss: 0.4702
- accuracy: 0.7877
Epoch 32/50
176/176 [==============================] - 0s 615us/step - loss: 0.4792
- accuracy: 0.7879
Epoch 33/50
176/176 [==============================] - 0s 616us/step - loss: 0.4858
- accuracy: 0.7847
Epoch 34/50
176/176 [==============================] - 0s 610us/step - loss: 0.4775
- accuracy: 0.7883
Epoch 35/50
176/176 [==============================] - 0s 610us/step - loss: 0.4810
- accuracy: 0.7863
Epoch 36/50
176/176 [==============================] - 0s 615us/step - loss: 0.4875
- accuracy: 0.7799
Epoch 37/50
176/176 [==============================] - 0s 613us/step - loss: 0.4738
- accuracy: 0.7899
Epoch 38/50
176/176 [==============================] - 0s 609us/step - loss: 0.4799
- accuracy: 0.7797
Epoch 39/50
176/176 [==============================] - 0s 612us/step - loss: 0.4740
- accuracy: 0.7911
Epoch 40/50
176/176 [==============================] - 0s 616us/step - loss: 0.4742
- accuracy: 0.7868
```

```
Epoch 41/50
176/176 [==============================] - 0s 611us/step - loss: 0.4706
- accuracy: 0.7840
Epoch 42/50
176/176 [==============================] - 0s 713us/step - loss: 0.4741
- accuracy: 0.7842
Epoch 43/50
176/176 [==============================] - 0s 614us/step - loss: 0.4732
- accuracy: 0.7881
Epoch 44/50
176/176 [==============================] - 0s 615us/step - loss: 0.4746
- accuracy: 0.7876
Epoch 45/50
176/176 [==============================] - 0s 615us/step - loss: 0.4715
- accuracy: 0.7879
Epoch 46/50
176/176 [==============================] - 0s 612us/step - loss: 0.4782
- accuracy: 0.7813
Epoch 47/50
176/176 [==============================] - 0s 611us/step - loss: 0.4618
- accuracy: 0.7852
Epoch 48/50
176/176 [==============================] - 0s 615us/step - loss: 0.4719
- accuracy: 0.7847
Epoch 49/50
176/176 [==============================] - 0s 612us/step - loss: 0.4562
- accuracy: 0.7899
Epoch 50/50
176/176 [==============================] - 0s 612us/step - loss: 0.4675
- accuracy: 0.7897
44/44 [==============================] - 0s 476us/step
```

Metrics Dataframe

In [ ]:
```
metrics_df = pd.DataFrame([[model_accuracy, model_precision, model_recall
```

In [ ]:
```
print(metrics_df)
```

```
   random_forest_accuracy  random_forest_precision  random_forest_recall
\
0                0.791045                   0.5125              0.674342

   xgboost_accuracy  xgboost_precision  xgboost_recall  \
0          0.771855               0.26        0.806202

   artificial_neural_network_accuracy  artificial_neural_network_precisi
on  \
0                            0.781095                                 0.38
25

   artificial_neural_network_recall
0                          0.714953
```

Confusion Matrix

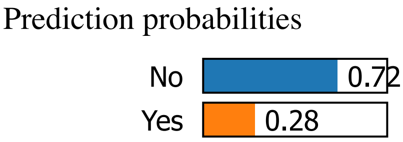
In [ ]:
```
print('Random Forest:', model_cm)
print('XGBoost:', xgb_cm)
print('Artificial Neural Network:', ann_cm)
```

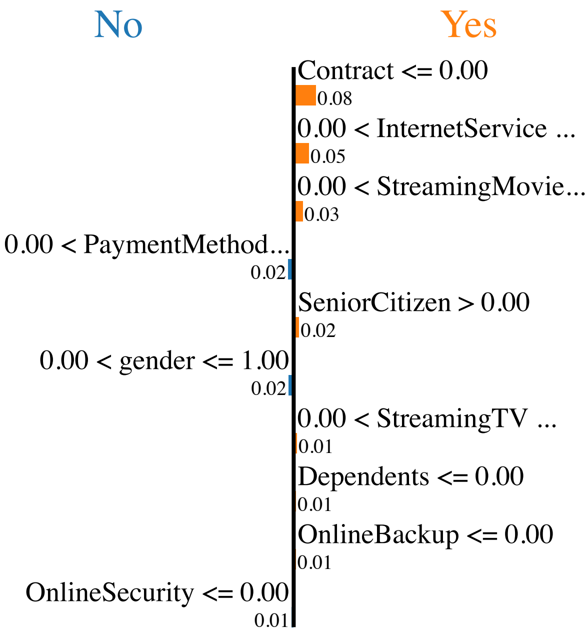
```
Random Forest: [[908 195]
 [ 99 205]]
XGBoost: [[982 296]
 [ 25 104]]
Artificial Neural Network: [[946 247]
 [ 61 153]]
```

LIME(Local Interpretable Model-agnostic Explanations)

In [ ]:
```
# Initialising Explainer
explainer = lime_tabular.LimeTabularExplainer(
    training_data=np.array(X_train),
    feature_names=X_train.columns,
    class_names=['No', 'Yes'],
    mode='classification'
)
```

In [ ]:
```
# Plotting Explainer
exp = explainer.explain_instance(
    data_row=X_test.iloc[1],
    predict_fn=model.predict_proba
)
exp.show_in_notebook(show_table=True)
```





Feature	Value
Contract	0.00
InternetService	1.00
StreamingMovies	1.00
PaymentMethod_Electronic check	1.00
SeniorCitizen	1.00
gender	1.00
StreamingTV	1.00
Dependents	0.00
OnlineBackup	0.00
OnlineSecurity	0.00