churn-notebook

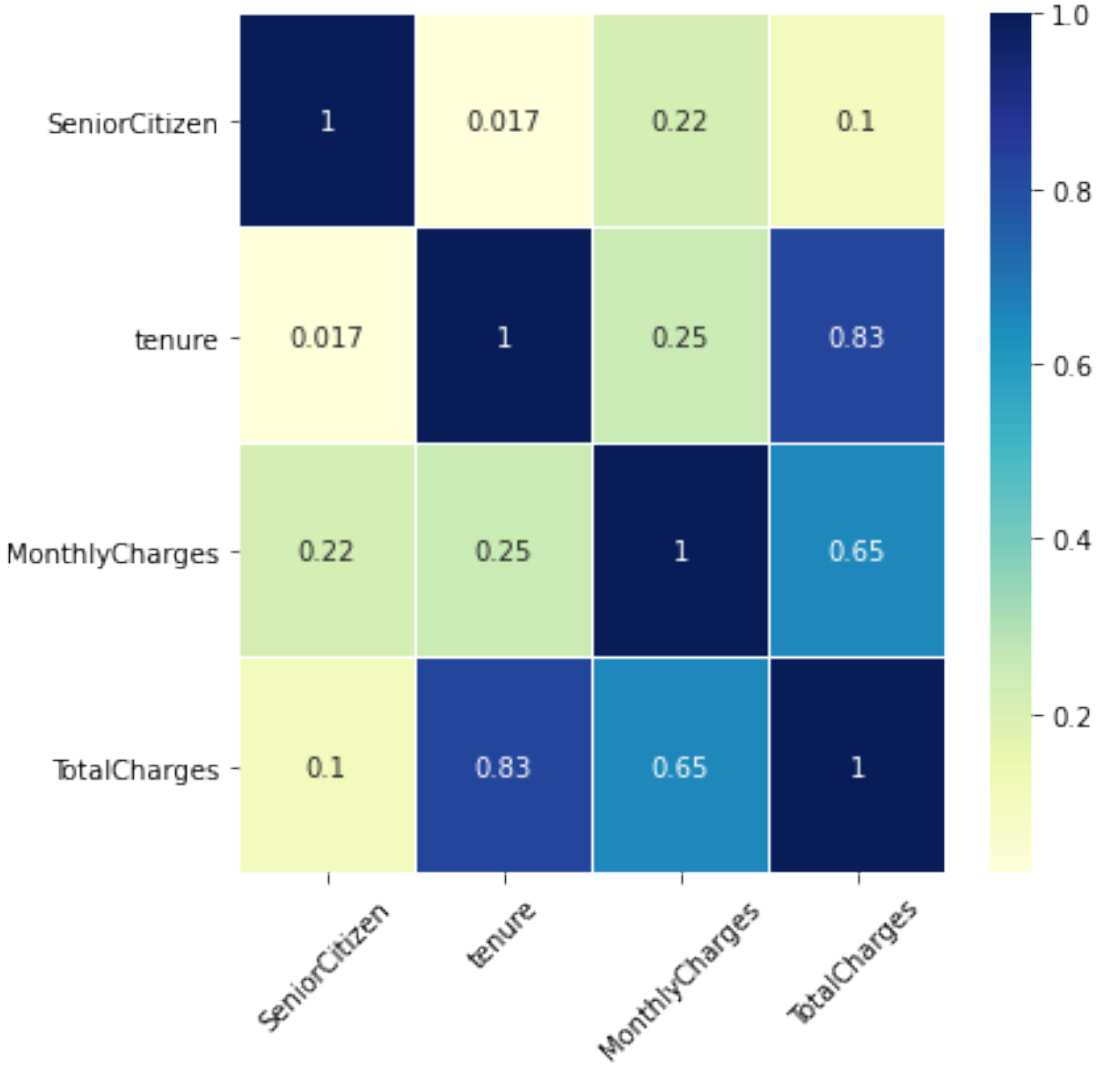
December 13, 2020

[10]:
```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import plotly.graph_objects as go
import seaborn as sns
import tensorflow as tf
from joblib import dump, load
from keras import optimizers
from keras.layers import Dense, Dropout
from plotly.offline import iplot, init_notebook_mode
from sklearn import metrics
from sklearn.decomposition import IncrementalPCA, PCA
from sklearn.ensemble import AdaBoostClassifier, RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_auc_score, confusion_matrix  # import metrics␣
↪from sklearn
from sklearn.model_selection import GridSearchCV, train_test_split, ShuffleSplit
from sklearn.preprocessing import MinMaxScaler

from sklearn.tree import DecisionTreeClassifier

from tensorflow import keras
from tensorflow.keras import Sequential

from imblearn.over_sampling import SMOTE


init_notebook_mode()
```

[11]:
```
dataset = pd.read_csv('./data/WA_Fn-UseC_-Telco-Customer-Churn.csv')
dataset.drop(columns=['customerID'], axis=1, inplace=True)
# Converting Total Charges to a numerical data type.
dataset.TotalCharges = pd.to_numeric(dataset.TotalCharges, errors='coerce')
dataset.shape
```

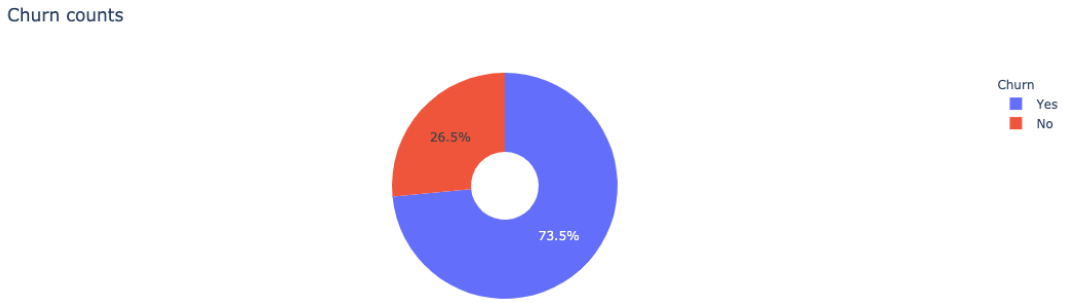
[11]:
```
(7043, 20)
```

[12]:
```
# Feature Correlation
plt.figure(figsize=(6,6))
feat_corr = dataset.drop(['Churn'], axis=1).corr()
g = sns.heatmap(feat_corr, xticklabels=feat_corr.columns, linewidths=.01,␣
↪cmap="YlGnBu", annot=True)
_ = g.set_yticklabels(g.get_yticklabels(), rotation = 0)
_ = g.set_xticklabels(g.get_xticklabels(), rotation = 45)
```



[13]:
```
values=dataset['Churn'].value_counts().to_list()
labels = ['Yes','No']
layout={'title':"Churn counts",'legend_title_text':'Churn','width':500,'height':
↪400}
trace=go.Pie(labels=labels, values=values, hole=.3)
```

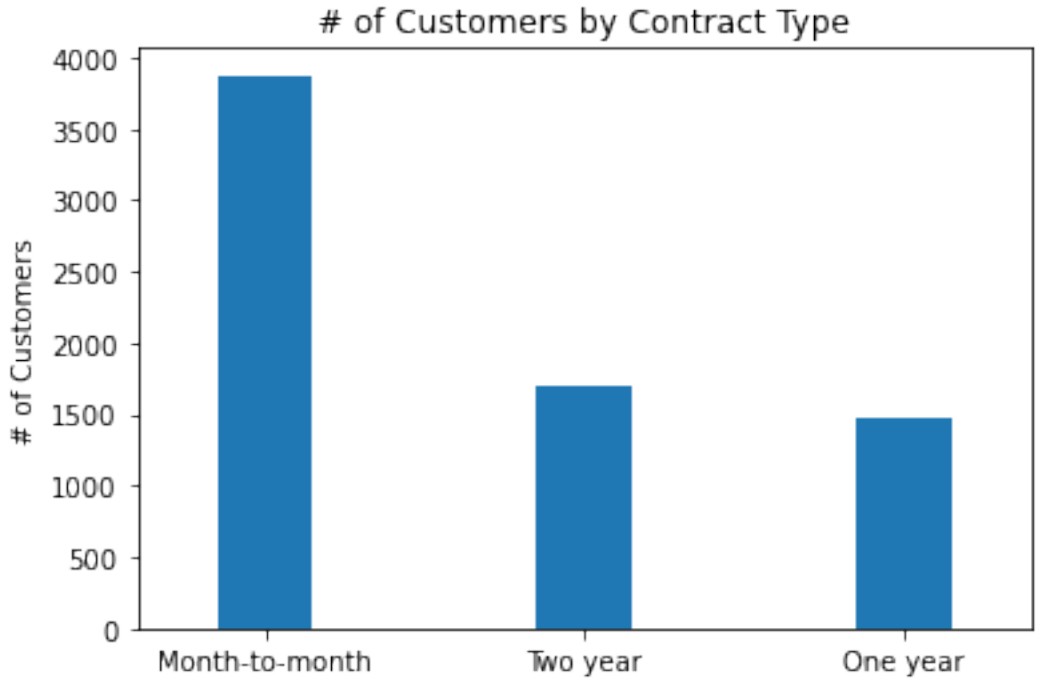
```
data=[trace]
fig = go.Figure(data=data,layout=layout)
iplot(fig)
```



We will need to balance the dataset later

[14]:
```
ax = dataset['Contract'].value_counts().plot(kind = 'bar',rot = 0, width = 0.3)
ax.set_ylabel('# of Customers')
ax.set_title('# of Customers by Contract Type')
```

[14]: Text(0.5, 1.0, '# of Customers by Contract Type')



[15]:
```
# Removing missing values
dataset.dropna(inplace = True)

# Convert the predictor variable in a binary numeric variable
dataset['Churn'].replace(to_replace='Yes', value=1, inplace=True)
dataset['Churn'].replace(to_replace='No',  value=0, inplace=True)

# Convert all the categorical variables into dummy variables
df_dummies = pd.get_dummies(dataset)
df_dummies.head()

dataset = df_dummies.copy()

# Put Churn at the front
churn = dataset.pop('Churn')
dataset.insert(0, 'Churn', churn)

print("Dataset loaded:", dataset.shape)
```

```
Dataset loaded: (7032, 46)
```

[16]:
```
def plot_confusion_matrix(y_true, y_pred, classes,
                          normalize=True,
                          title=None,
                          cmap=plt.cm.Blues):
    if not title:
        if normalize:
            title = 'Normalized confusion matrix'
        else:
            title = 'Confusion matrix, without normalization'

    # Compute confusion matrix
    cm = confusion_matrix(y_true, y_pred)
    # Only use the labels that appear in the data
    #classes = classes[unique_labels(y_true, y_pred)]
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

#     print(cm)

    fig, ax = plt.subplots()
    im = ax.imshow(cm, interpolation='nearest', cmap=cmap)
    ax.figure.colorbar(im, ax=ax)
```

```
    # We want to show all ticks...
    ax.set(xticks=np.arange(cm.shape[1]),
           yticks=np.arange(cm.shape[0]),
           title=title,
           ylabel='True label',
           xlabel='Predicted label')

    # Rotate the tick labels and set their alignment.
    plt.setp(ax.get_xticklabels(), rotation=45, ha="right",
             rotation_mode="anchor")

    # Loop over data dimensions and create text annotations.
    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i in range(cm.shape[0]):
        for j in range(cm.shape[1]):
            ax.text(j, i, format(cm[i, j], fmt),
                    ha="center", va="center",
                    color="white" if cm[i, j] > thresh else "black")
    fig.tight_layout()
    return ax
```

In the interest of customer privacy, we will drop customer ID and their Surname.

[17]:
```
dataset.columns
```

[17]:
```
Index(['Churn', 'SeniorCitizen', 'tenure', 'MonthlyCharges', 'TotalCharges',
       'gender_Female', 'gender_Male', 'Partner_No', 'Partner_Yes',
       'Dependents_No', 'Dependents_Yes', 'PhoneService_No',
       'PhoneService_Yes', 'MultipleLines_No',
       'MultipleLines_No phone service', 'MultipleLines_Yes',
       'InternetService_DSL', 'InternetService_Fiber optic',
       'InternetService_No', 'OnlineSecurity_No',
       'OnlineSecurity_No internet service', 'OnlineSecurity_Yes',
       'OnlineBackup_No', 'OnlineBackup_No internet service',
       'OnlineBackup_Yes', 'DeviceProtection_No',
       'DeviceProtection_No internet service', 'DeviceProtection_Yes',
       'TechSupport_No', 'TechSupport_No internet service', 'TechSupport_Yes',
       'StreamingTV_No', 'StreamingTV_No internet service', 'StreamingTV_Yes',
       'StreamingMovies_No', 'StreamingMovies_No internet service',
       'StreamingMovies_Yes', 'Contract_Month-to-month', 'Contract_One year',
       'Contract_Two year', 'PaperlessBilling_No', 'PaperlessBilling_Yes',
       'PaymentMethod_Bank transfer (automatic)',
       'PaymentMethod_Credit card (automatic)',
       'PaymentMethod_Electronic check', 'PaymentMethod_Mailed check'],
      dtype='object')
```

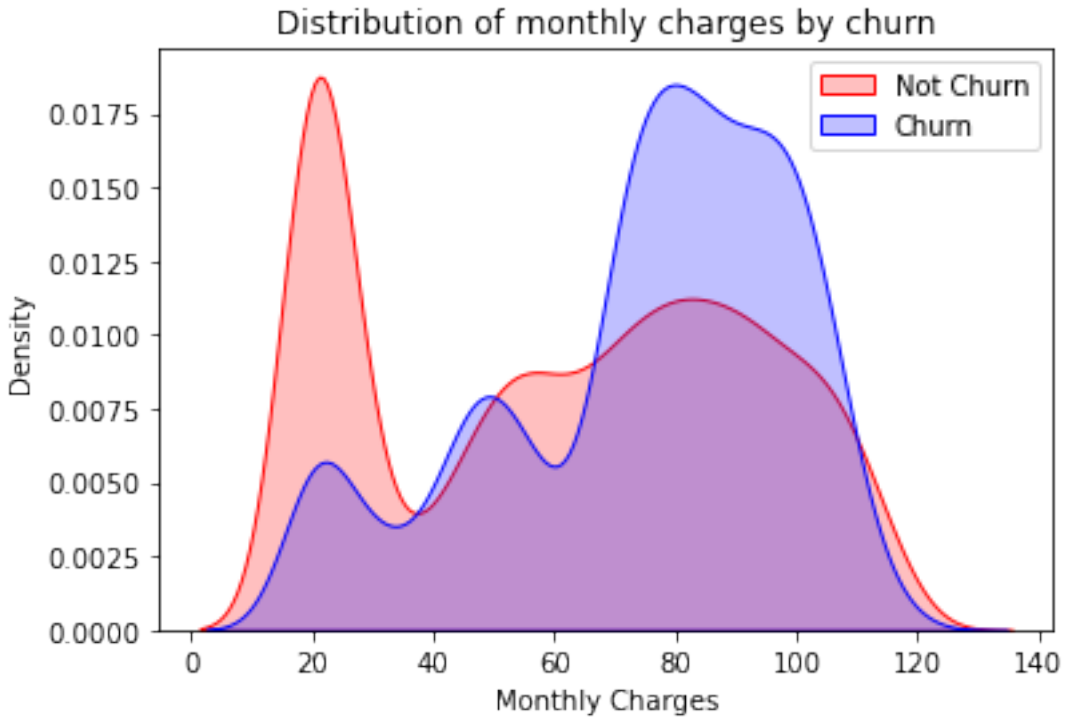
[18]:
```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 7032 entries, 0 to 7042
Data columns (total 46 columns):
 #   Column                                  Non-Null Count  Dtype
---  ------                                  --------------  -----
 0   Churn                                   7032 non-null   int64
 1   SeniorCitizen                           7032 non-null   int64
 2   tenure                                  7032 non-null   int64
 3   MonthlyCharges                          7032 non-null   float64
 4   TotalCharges                            7032 non-null   float64
 5   gender_Female                           7032 non-null   uint8
 6   gender_Male                             7032 non-null   uint8
 7   Partner_No                              7032 non-null   uint8
 8   Partner_Yes                             7032 non-null   uint8
 9   Dependents_No                           7032 non-null   uint8
 10  Dependents_Yes                          7032 non-null   uint8
 11  PhoneService_No                         7032 non-null   uint8
 12  PhoneService_Yes                        7032 non-null   uint8
 13  MultipleLines_No                        7032 non-null   uint8
 14  MultipleLines_No phone service          7032 non-null   uint8
 15  MultipleLines_Yes                       7032 non-null   uint8
 16  InternetService_DSL                     7032 non-null   uint8
 17  InternetService_Fiber optic             7032 non-null   uint8
 18  InternetService_No                      7032 non-null   uint8
 19  OnlineSecurity_No                       7032 non-null   uint8
 20  OnlineSecurity_No internet service      7032 non-null   uint8
 21  OnlineSecurity_Yes                      7032 non-null   uint8
 22  OnlineBackup_No                         7032 non-null   uint8
 23  OnlineBackup_No internet service        7032 non-null   uint8
 24  OnlineBackup_Yes                        7032 non-null   uint8
 25  DeviceProtection_No                     7032 non-null   uint8
 26  DeviceProtection_No internet service    7032 non-null   uint8
 27  DeviceProtection_Yes                    7032 non-null   uint8
 28  TechSupport_No                          7032 non-null   uint8
 29  TechSupport_No internet service         7032 non-null   uint8
 30  TechSupport_Yes                         7032 non-null   uint8
 31  StreamingTV_No                          7032 non-null   uint8
 32  StreamingTV_No internet service         7032 non-null   uint8
 33  StreamingTV_Yes                         7032 non-null   uint8
 34  StreamingMovies_No                      7032 non-null   uint8
 35  StreamingMovies_No internet service     7032 non-null   uint8
 36  StreamingMovies_Yes                     7032 non-null   uint8
 37  Contract_Month-to-month                 7032 non-null   uint8
 38  Contract_One year                       7032 non-null   uint8
 39  Contract_Two year                       7032 non-null   uint8
```

```
 40  PaperlessBilling_No                          7032 non-null   uint8
 41  PaperlessBilling_Yes                         7032 non-null   uint8
 42  PaymentMethod_Bank transfer (automatic)      7032 non-null   uint8
 43  PaymentMethod_Credit card (automatic)        7032 non-null   uint8
 44  PaymentMethod_Electronic check               7032 non-null   uint8
 45  PaymentMethod_Mailed check                   7032 non-null   uint8
dtypes: float64(2), int64(3), uint8(41)
memory usage: 611.2 KB
```

[19]:
```
ax = sns.kdeplot(dataset.MonthlyCharges[(dataset["Churn"] == 0) ],
                 color="Red", shade = True)
ax = sns.kdeplot(dataset.MonthlyCharges[(dataset["Churn"] == 1) ],
                 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')
```

[19]:
```
Text(0.5, 1.0, 'Distribution of monthly charges by churn')
```

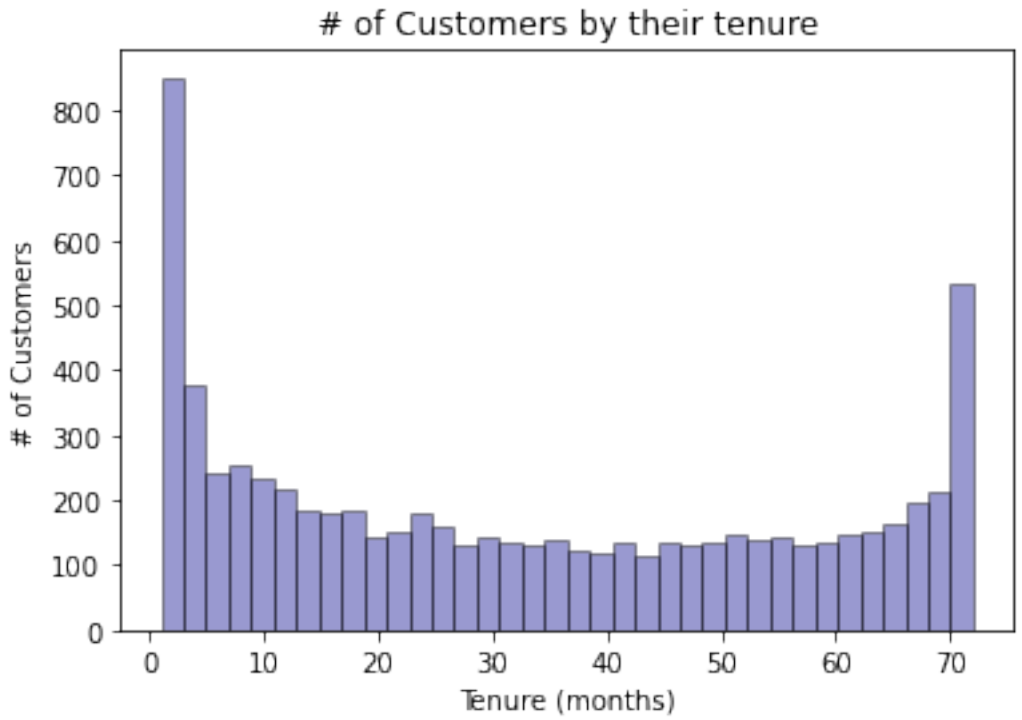


[20]:
```
ax = sns.distplot(dataset['tenure'], hist=True, kde=False,
             bins=int(180/5), color = 'darkblue',
             hist_kws={'edgecolor':'black'},
             kde_kws={'linewidth': 4})
```

```
ax.set_ylabel('# of Customers')
ax.set_xlabel('Tenure (months)')
ax.set_title('# of Customers by their tenure')
```

```
/opt/conda/lib/python3.7/site-packages/seaborn/distributions.py:2551:
FutureWarning:

`distplot` is a deprecated function and will be removed in a future version.
Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).
```

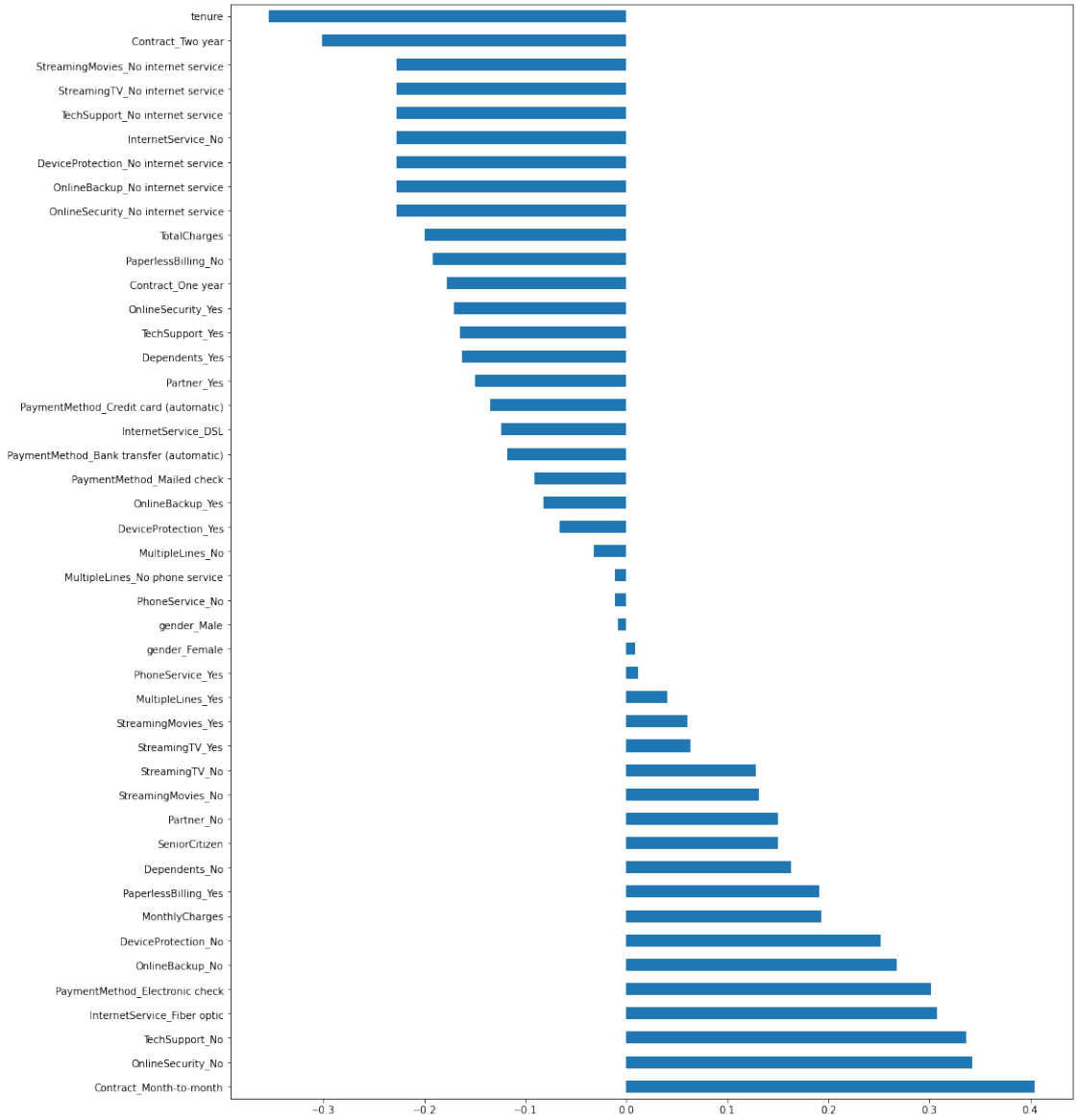
[20]:
```
Text(0.5, 1.0, '# of Customers by their tenure')
```



Correlation of Churn with other variables:

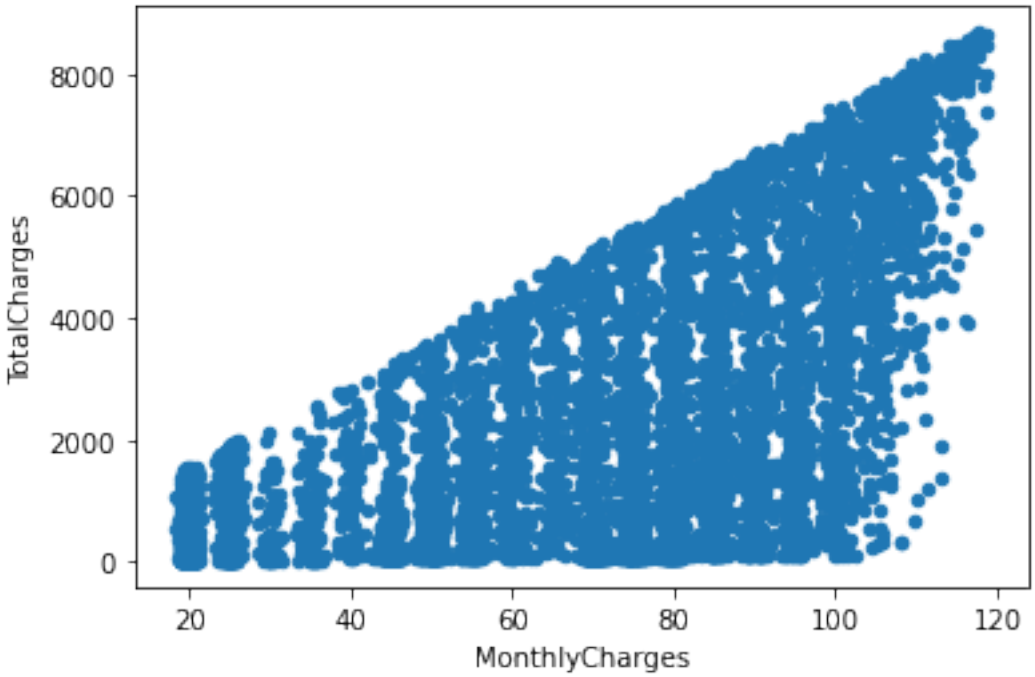
[21]:
```
plt.figure(figsize=(15,20))
df_plot = dataset.corr()['Churn'].sort_values(ascending = False)
df_plot = df_plot.drop(labels=['Churn'])
df_plot.plot(kind='barh')
```

[21]:
```
<AxesSubplot:>
```



[22]:
```
dataset[['MonthlyCharges', 'TotalCharges']].plot.scatter(x =␣
↪'MonthlyCharges',y='TotalCharges')
```

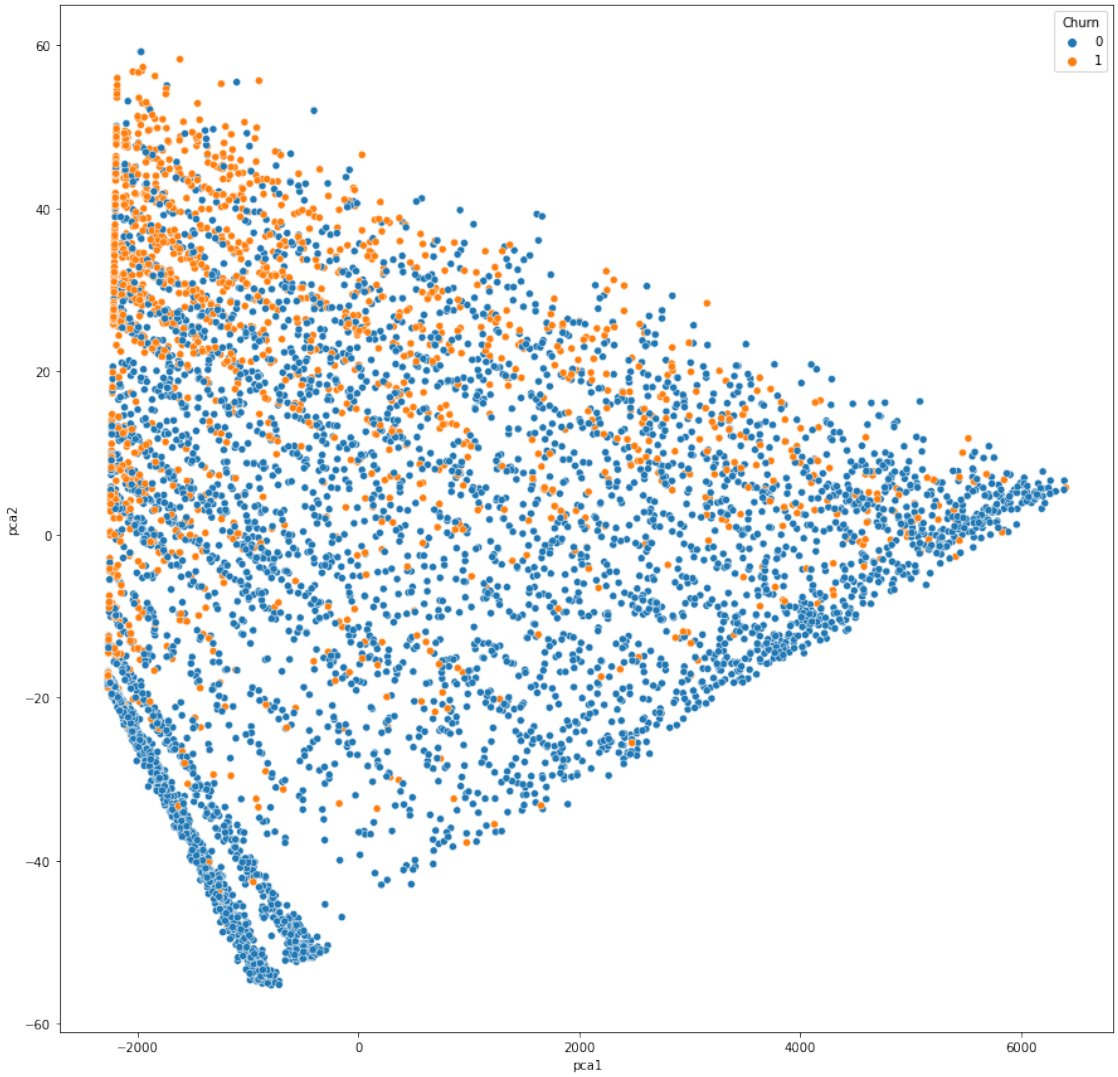
[22]: <AxesSubplot:xlabel='MonthlyCharges', ylabel='TotalCharges'>



We will use the data frame where we had created dummy variables

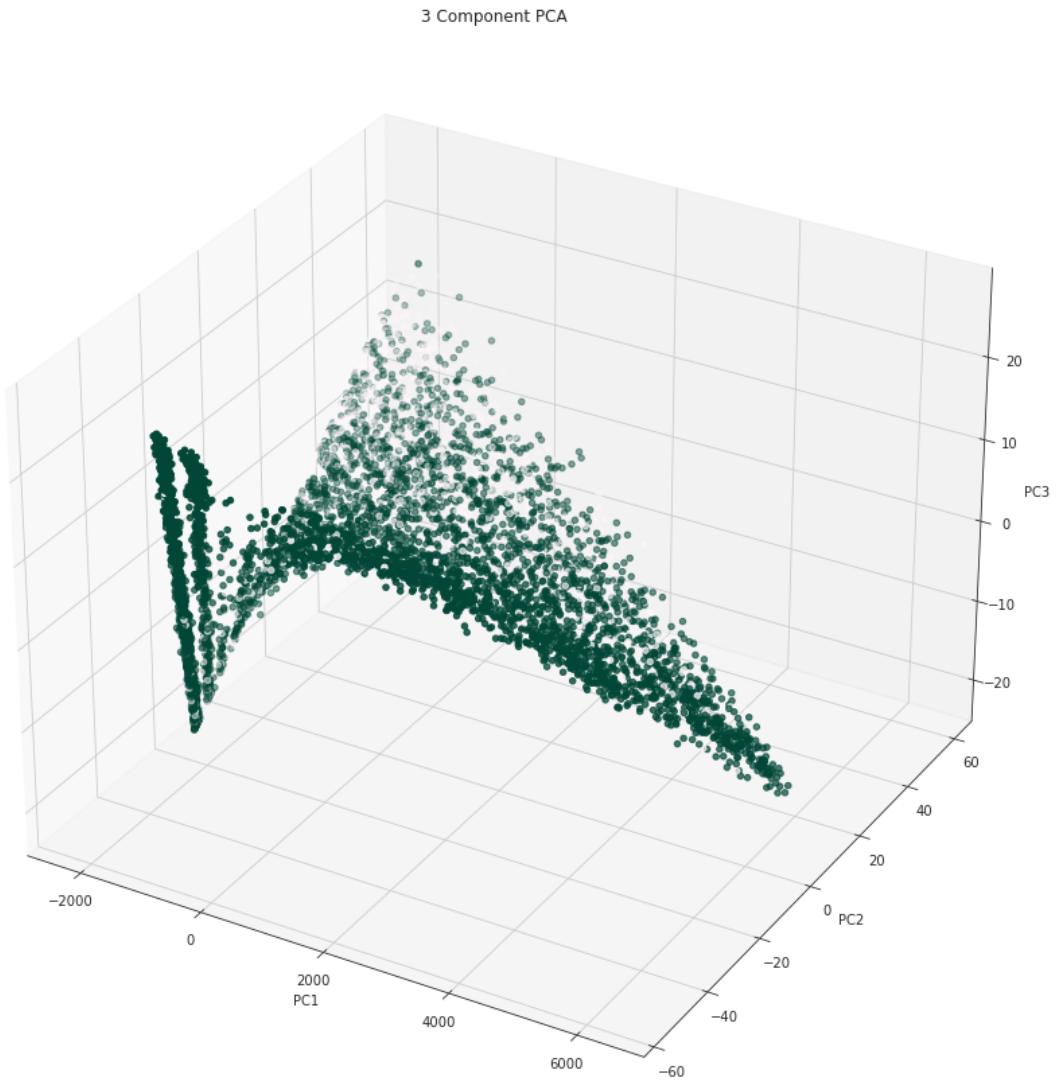
[23]:
```
# dataset = dataset.head(100)

y = dataset['Churn'].values
X = dataset.drop(columns = ['Churn'])
```

[24]:
```
pca = PCA(n_components=2, svd_solver='full')
np_pca = pca.fit_transform(dataset)

d = {'pca1': np_pca[:, 0], 'pca2': np_pca[:, 1], 'Churn': y}
df_pca = pd.DataFrame(data=d)

fig = plt.figure(1, figsize=(15, 15))
sns.scatterplot(data=df_pca, x="pca1", y="pca2", hue="Churn",␣
 ↪cmap=['red','blue'])
plt.show()
```



[27]:
```
sns.set_style("white")

# Run The PCA
pca = PCA(n_components=3)
pca.fit(dataset)

# Store results of PCA in a data frame
result=pd.DataFrame(pca.transform(dataset), columns=['PCA%i' % i for i in␣
 ↪range(3)], index=dataset.index)

# # Plot initialisation
fig = plt.figure(1, figsize=(15, 15))
ax = fig.add_subplot(111, projection='3d')
ax.scatter(result['PCA0'],result['PCA1'], result['PCA2'],
```

```
            c=dataset['Churn'],
            cmap='PuBuGn_r')


# label the axes
ax.set_xlabel("PC1")
ax.set_ylabel("PC2")
ax.set_zlabel("PC3")
ax.set_title("3 Component PCA")
plt.show()
```



No obvious clustering here. From testing with/without PCA, all models performed better without PCA.

We do still need to scale and balance the dataset.

[28]:
```
features = X.columns.values
scaler = MinMaxScaler(feature_range = (0,1))
scaler.fit(X)
X = pd.DataFrame(scaler.transform(X))
X.columns = features
```

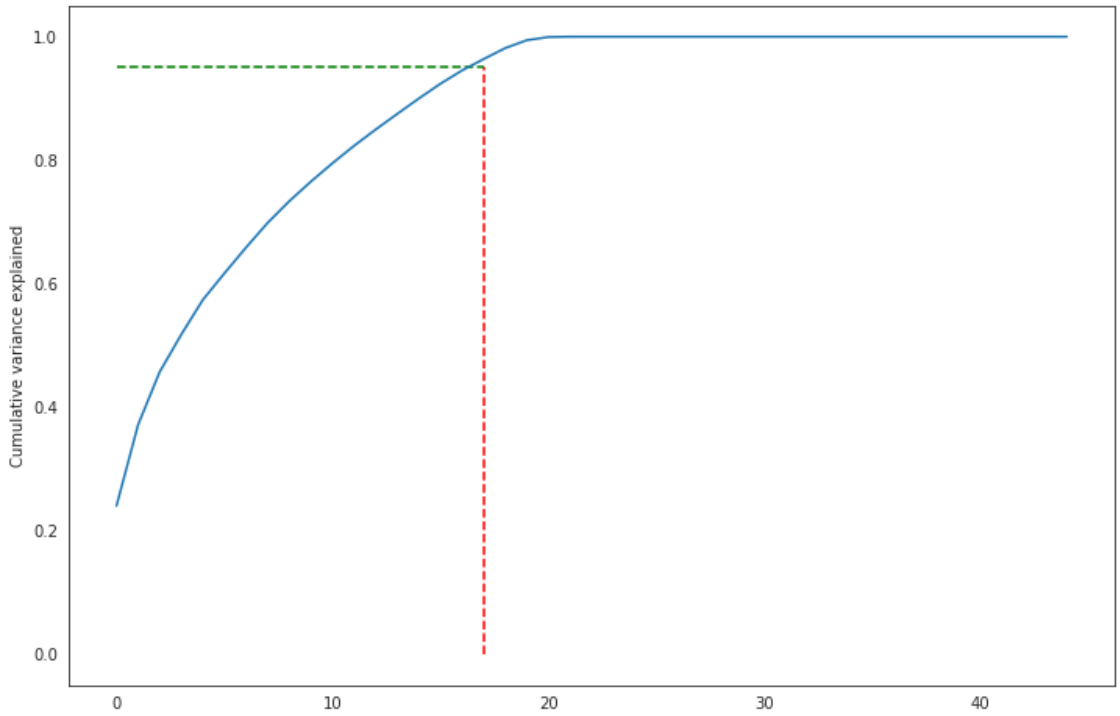
1 Create Train & Test Data

[29]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5,␣
↪random_state=42)
```

1.1 PCA Analysis

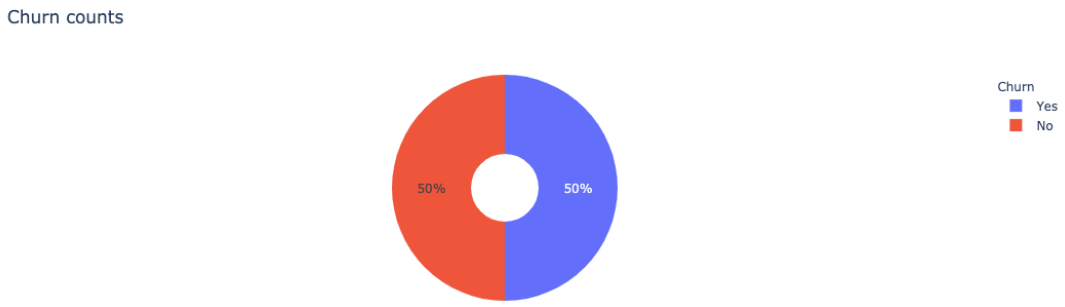
[30]:
```
# Running pca with default parameters.
pca = IncrementalPCA()

# Fitting the model
X_train_pca = pca.fit_transform(X_train)
X_test_pca = pca.transform(X_test)

# cumulative variance
var_cumu = np.cumsum(pca.explained_variance_ratio_)

CUM_VAR = 0.95

num_comps = len(np.argwhere(var_cumu < CUM_VAR))
print(CUM_VAR, "confidence with", num_comps, "components")

# code for Scree plot
fig = plt.figure(figsize=[12,8])
plt.vlines(x=num_comps, ymax=CUM_VAR, ymin=0, colors="r", linestyles="--")
plt.hlines(y=CUM_VAR, xmax=num_comps, xmin=0, colors="g", linestyles="--")
plt.plot(var_cumu)
plt.ylabel("Cumulative variance explained")
plt.show()
```

```
0.95 confidence with 17 components
```



Interestingly, 95% variance is explained by 17 dimensions.

[31]:
```
# Balancing the dataSet using SMOTE method




sm = SMOTE(sampling_strategy='auto', random_state=100)
X_bal, y_bal = sm.fit_sample(X, y)
X = X_bal.copy()
y = y_bal.copy()

distinct, values = np.unique(y, return_counts=True)

labels = ['Yes','No']
layout={'title':"Churn counts",'legend_title_text':'Churn','width':500,'height':
 ↪400}
trace=go.Pie(labels=labels, values=values, hole=.3)
data=[trace]
fig = go.Figure(data=data,layout=layout)
iplot(fig)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5,␣
 ↪random_state=42)
```



2 Hyperparameter Tuning of various models

[32]:
```
fits = {}
cv_sets = ShuffleSplit(random_state = 42) # shuffling our data for␣
↪cross-validation
```

2.1 Decision Trees

[33]:
```
dtc = DecisionTreeClassifier()
tree_param = {'criterion':['gini','entropy'],'max_depth':range(1,15),
              "min_samples_split":range(1,10),
              "min_samples_leaf":range(1,5)}

gs = GridSearchCV(estimator=dtc,
                  param_grid= tree_param,
                  n_jobs= -1,
                  cv= cv_sets)

grid_result = gs.fit(X_train, y_train)
dump(grid_result.best_estimator_, 'models/DecisionTreeClassifier.
↪joblib',compress = 1)

fits['DecisionTreeClassifier'] = grid_result

print("DecisionTreeClassifier best: %f using %s" % (grid_result.best_score_,␣
↪grid_result.best_params_))
```

```
DecisionTreeClassifier best: 0.805609 using {'criterion': 'entropy',
'max_depth': 10, 'min_samples_leaf': 2, 'min_samples_split': 5}
```

2.2 Adaboost

[34]:
```
ada_classifier = AdaBoostClassifier(random_state=42)

parameters = {'n_estimators': [int(x) for x in np.linspace(start = 200, stop =␣
↪300, num = 10)],
              'learning_rate': np.linspace(0.01,0.5,10,endpoint=False)}

gs = GridSearchCV(estimator=ada_classifier,
                  param_grid= parameters,
                  n_jobs= -1,
                  cv= cv_sets)

grid_result = gs.fit(X_train, y_train)
dump(grid_result.best_estimator_, 'models/AdaBoostClassifier.joblib',compress =␣
↪1)

print("AdaBoostClassifier best: %f using %s" % (grid_result.best_score_,␣
↪grid_result.best_params_))
```

```
AdaBoostClassifier best: 0.823404 using {'learning_rate': 0.451, 'n_estimators':
300}
```

2.3 Logistic Regression

[35]:
```
log_reg = LogisticRegression(random_state = 42, n_jobs=-1, verbose=2)

# Creating hyper parameter grid
parameters = {'solver': ['newton-cg', 'lbfgs','sag'],
                  'penalty': ['l1', 'l2', 'elasticnet'],
                  'C': np.linspace(0.05,0.1,20,endpoint=False)}

gs = GridSearchCV(estimator=log_reg,
                  param_grid= parameters,
                  n_jobs= -1,
                  cv= cv_sets)

grid_result = gs.fit(X_train, y_train)
dump(grid_result.best_estimator_, 'models/LogisticRegression.joblib',compress =␣
↪1)

fits['LogisticRegression'] = grid_result

# Finding the best model
print("LogisticRegression best: %f using %s" % (grid_result.best_score_,␣
↪grid_result.best_params_))
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 64 concurrent workers.
```

```
[Parallel(n_jobs=-1)]: Done   1 out of   1 | elapsed:    0.1s finished
```

```
LogisticRegression best: 0.776789 using {'C': 0.0825, 'penalty': 'l2', 'solver':
'newton-cg'}
```

2.4 Random Forest

[36]:
```
rfc = RandomForestClassifier(random_state = 42, n_jobs=-1, verbose=1)

# Create the random parameter grid
parameters = {
    'n_estimators': [int(x) for x in np.linspace(500,1000,20,endpoint=False)],
    'max_features': ['auto', 'sqrt', 'log2']
}

# Searching across different combinations for best model parameters
gs = GridSearchCV(estimator=rfc,
                              param_grid=parameters,
                              cv = cv_sets,
                              n_jobs=-1)


grid_result = gs.fit(X_train, y_train)
dump(grid_result.best_estimator_, 'models/RandomForestClassifier.
 ↪joblib',compress = 1)

print("RandomForestClassifier best: %f using %s" % (grid_result.best_score_,␣
 ↪grid_result.best_params_))
```

```
[Parallel(n_jobs=-1)]: Using backend ThreadingBackend with 64 concurrent
workers.
[Parallel(n_jobs=-1)]: Done  72 tasks      | elapsed:    0.4s
[Parallel(n_jobs=-1)]: Done 322 tasks      | elapsed:    1.1s
[Parallel(n_jobs=-1)]: Done 725 out of 725 | elapsed:    2.2s finished
```

```
RandomForestClassifier best: 0.849903 using {'max_features': 'auto',
'n_estimators': 725}
```

2.5 TensorFlow Neural Network

[37]:
```
model = Sequential()
model.add(Dense(512, input_shape=(X_train.shape[1],), activation='relu'))
model.add(Dropout(0.3))
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.3))
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.3))
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.3))
```

```
model.add(Dense(32, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(16, activation='relu'))
model.add(Dropout(0.3))
model.add(Dense(1, activation='sigmoid'))

callback = tf.keras.callbacks.EarlyStopping(monitor='loss', patience=3)

# Compile model
model.compile(loss='binary_crossentropy', optimizer=optimizers.RMSprop(lr=0.
 ↪0001), metrics=['accuracy'])

model.fit(X_train, y_train, epochs=100, batch_size=50, validation_data=(X_test,␣
 ↪y_test), verbose=0)
model.save('sequentialnetmodel')
```

```
WARNING:tensorflow:From /opt/conda/lib/python3.7/site-
packages/tensorflow/python/training/tracking/tracking.py:111:
Model.state_updates (from tensorflow.python.keras.engine.training) is deprecated
and will be removed in a future version.
Instructions for updating:
This property should not be used in TensorFlow 2.0, as updates are applied
automatically.
WARNING:tensorflow:From /opt/conda/lib/python3.7/site-
packages/tensorflow/python/training/tracking/tracking.py:111: Layer.updates
(from tensorflow.python.keras.engine.base_layer) is deprecated and will be
removed in a future version.
Instructions for updating:
This property should not be used in TensorFlow 2.0, as updates are applied
automatically.
INFO:tensorflow:Assets written to: sequentialnetmodel/assets
```

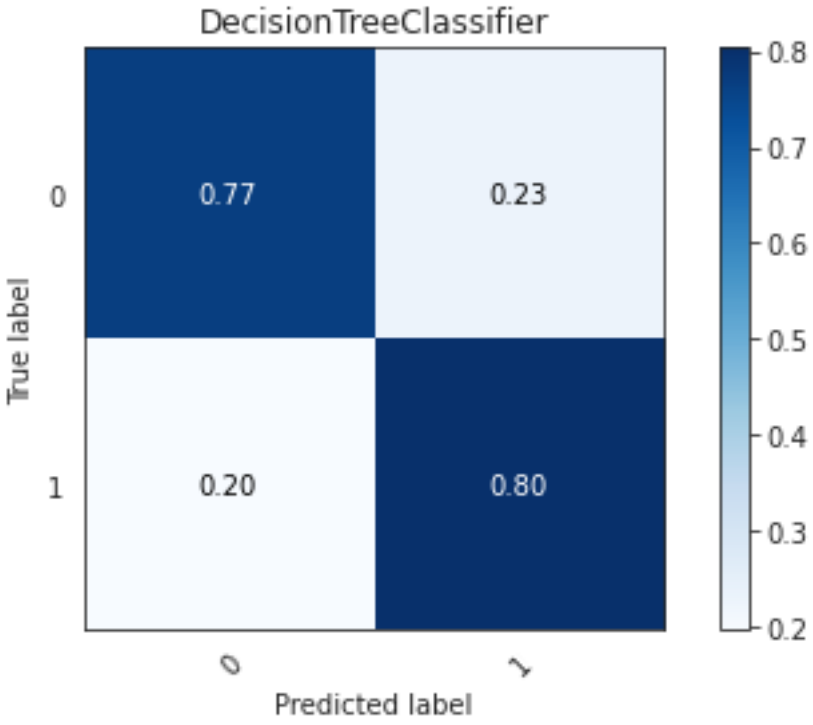
3 Evaluating Models with their best parameters

3.0.1 Decision Trees

[38]:
```
model = load('models/DecisionTreeClassifier.joblib')
model.fit(X_train,y_train)
preds = model.predict(X_test)

print(metrics.accuracy_score(y_test, preds))
plot_confusion_matrix(y_test, preds, classes=["loyal", "churn"],␣
 ↪title="DecisionTreeClassifier")
```

```
0.7867518884369552
Normalized confusion matrix
```

[38]: <AxesSubplot:title={'center':'DecisionTreeClassifier'}, xlabel='Predicted label', ylabel='True label'>

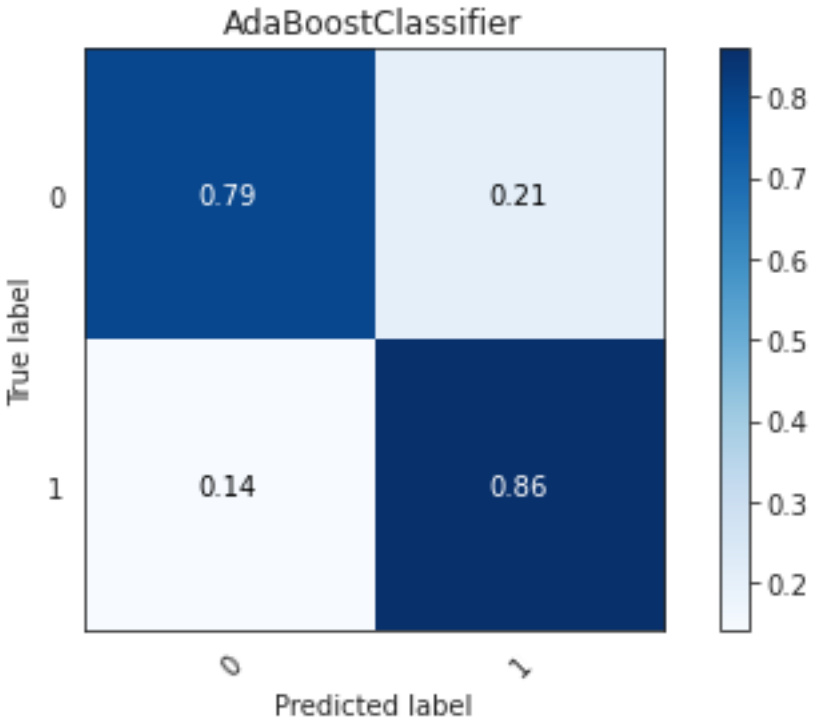


3.0.2 AdaBoost

[39]:
```
model = load('models/AdaBoostClassifier.joblib')
model.fit(X_train,y_train)
preds = model.predict(X_test)

print(metrics.accuracy_score(y_test, preds))
plot_confusion_matrix(y_test, preds, classes=["loyal", "churn"],␣
 ↪title="AdaBoostClassifier")
```

```
0.8249079992252566
Normalized confusion matrix
```

[39]: <AxesSubplot:title={'center':'AdaBoostClassifier'}, xlabel='Predicted label', ylabel='True label'>



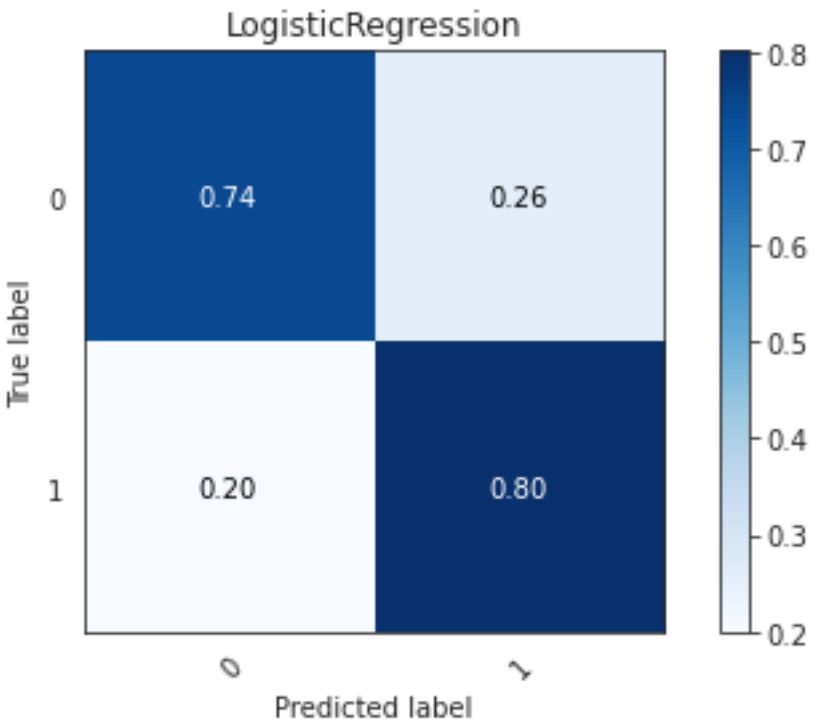
3.0.3 Logistic Regression

[40]:
```
model = load('models/LogisticRegression.joblib')
model.fit(X_train,y_train)
preds = model.predict(X_test)

print(metrics.accuracy_score(y_test, preds))
plot_confusion_matrix(y_test, preds, classes=["loyal", "churn"],␣
↪title="LogisticRegression")
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 64 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 out of   1 | elapsed:    0.1s finished
```

```
0.7714507069533217
Normalized confusion matrix
```

[40]:
```
<AxesSubplot:title={'center':'LogisticRegression'}, xlabel='Predicted label',
ylabel='True label'>
```

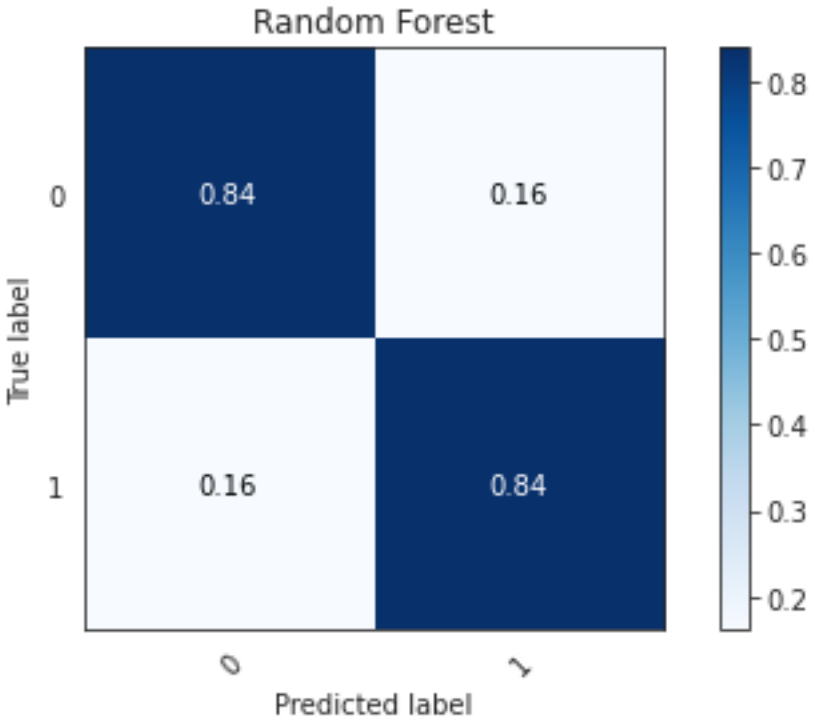


3.0.4 Random Forest

[41]:
```
model = load('models/RandomForestClassifier.joblib')
model.fit(X_train,y_train)
preds = model.predict(X_test)

print(metrics.accuracy_score(y_test, preds))
plot_confusion_matrix(y_test, preds, classes=["loyal", "churn"], title="Random␣
↪Forest")
```

```
[Parallel(n_jobs=-1)]: Using backend ThreadingBackend with 64 concurrent
workers.
[Parallel(n_jobs=-1)]: Done  72 tasks      | elapsed:    0.4s
[Parallel(n_jobs=-1)]: Done 322 tasks      | elapsed:    1.2s
[Parallel(n_jobs=-1)]: Done 725 out of 725 | elapsed:    2.2s finished
[Parallel(n_jobs=64)]: Using backend ThreadingBackend with 64 concurrent
workers.
[Parallel(n_jobs=64)]: Done  72 tasks      | elapsed:    0.1s
[Parallel(n_jobs=64)]: Done 322 tasks      | elapsed:    0.2s
[Parallel(n_jobs=64)]: Done 725 out of 725 | elapsed:    0.4s finished

0.8380786364516754
Normalized confusion matrix
```

[41]: <AxesSubplot:title={'center':'Random Forest'}, xlabel='Predicted label', ylabel='True label'>

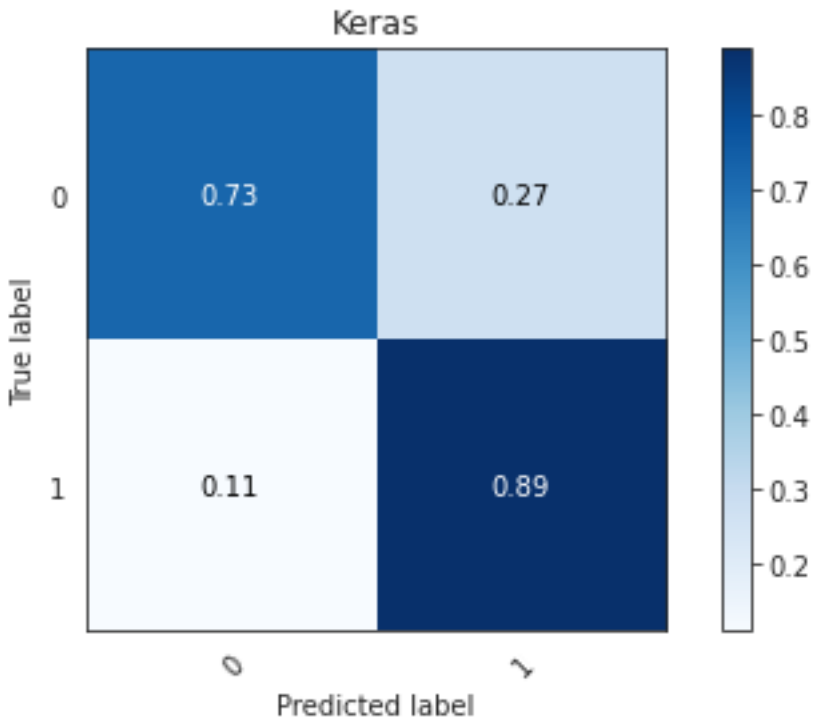


3.0.5 Tensorflow Sequential Neural Network

[42]:
```
model = keras.models.load_model('sequentialnetmodel')
preds = model.predict(X_test)
y_predict = (preds>0.5)
print(metrics.accuracy_score(y_test, y_predict))
plot_confusion_matrix(y_test, y_predict, classes=["loyal", "churn"],␣
↪title="Keras")
```

```
0.8063141584350184
Normalized confusion matrix
```

[42]: <AxesSubplot:title={'center':'Keras'}, xlabel='Predicted label', ylabel='True label'>


Keras
0.73
0.27
0.11
0.89
0
1
True label
0
1
Predicted label
0.8
0.7
0.6
0.5
0.4
0.3
0.2