Forward School

Program Code: J620-002-4:2020

Program Name: FRONT-END SOFTWARE DEVELOPMENT

Title : Case Study - Artificial Neural Network using Keras

Name: Chong Mun Chen

IC Number: 960327-07-5097

Date : 2/8/2023

Introduction : Practising with Neural Network learning model on the customer churn and retention dataset using Tensorflow's Keras model.

Conclusion : Succeeded in achieving a linear ROC curve with the AUC number.

Example: Artificial Neural Network using Keras

This exercise is adapted from the book Hands-on Data Science for Marketing: Chapter Customer churn and retention.

Dataset is provided by IBM Watson. You can also find it here: https://www.kaggle.com/zagarsuren/telecom-churn-dataset-ibm-watson-analytics (https://www.kaggle.com/zagarsuren/telecom-churn-dataset-ibm-watson-analytics). A copy of the data is stored in this week's Data folder.

Note for lecturer:

- Link to the book via SafariBooksOnline - https://learning.oreilly.com/library/view/hands-on-data-science/9781789346343/b984726e-af92-4525-8a1f-7343a9b2ac76.xhtml (https://learning.oreilly.com/library/view/hands-on-data-science/9781789346343/b984726e-af92-4525-8a1f-7343a9b2ac76.xhtml)

- Sample answer in github: [https://github.com/yoonhwang/hands-on-data-science-for-marketing/blob/master/ch.11/python/CustomerRetention.ipynb (https://github.com/yoonhwang/hands-on-data-science-for-marketing/blob/master/ch.11/python/CustomerRetention.ipynb)](https://github.com/yoonhwang/hands-on-data-science-for-marketing/blob/master/ch.11/python/CustomerRetention.ipynb)

In [1]:
```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

# Load the dataset into the dataframe df
df = pd.read_excel('../../data_samples2/WA_Fn-UseC_-Telco-Customer-Churn.xl
```

Step 2: Show the first 10 lines of the contents in df

In [2]:
```
df.head(10)
```

Out[2]:

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

10 rows × 21 columns

Q: How many variables/attributes are there in the dataset? which is the target variable?

Step 3: Target variable encoding: As you may have noticed from the data, the target variable, Churn, has two values: Yes and No. Please encode these values as 1 for Yes and 0 for No.

In [3]:
```
reset = {'No': 0, 'Yes': 1}
df = df.replace({'Churn': reset})
df.head()
```

Out[3]:

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLi
0	7590-VHVEG	Female	0	Yes	No	1	No	No ph ser
1	5575-GNVDE	Male	0	No	No	34	Yes	
2	3668-QPYBK	Male	0	No	No	2	Yes	
3	7795-CFOCW	Male	0	No	No	45	No	No ph ser
4	9237-HQITU	Female	0	No	No	2	Yes	

5 rows × 21 columns

Q: What is the overall churn rate? Do you find the churn rate worth paying attention to?

In [4]:
```
df['Churn'].mean()
```

Out[4]:
```
0.2653698707936959
```

Handling missing values in the TotalCharges column: If you looked through the TotalCharges column in the dataset, you may have noticed that there are some records with no TotalCharges values.

Since there are only 11 records with missing TotalCharges values, it is safe to simply ignore and drop those records with missing values.

Step 4: Remove entries that have missing TotalCharges values.

In [5]:
```
df['TotalCharges'] = df['TotalCharges'].replace(' ', np.nan).astype(float)
df.shape
```

Out[5]:
```
(7043, 21)
```

In [6]:
```
df = df.dropna()
df.shape
```

Out[6]:
```
(7032, 21)
```

Transforming continuous variables: The next step is to scale the continuous variables.

Step 5: Take a look at the summary statistics for continuous variables tenure, MonthlyCharges and TotalCharges

In [7]:
```
df[['tenure', 'MonthlyCharges', 'TotalCharges']].describe()
```

Out[7]:

	tenure	MonthlyCharges	TotalCharges
count	7032.000000	7032.000000	7032.000000
mean	32.421786	64.798208	2283.300441
std	24.545260	30.085974	2266.771362
min	1.000000	18.250000	18.800000
25%	9.000000	35.587500	401.450000
50%	29.000000	70.350000	1397.475000
75%	55.000000	89.862500	3794.737500
max	72.000000	118.750000	8684.800000

Step 6: Normalize the variables tenure, MonthlyCharges and TotalCharges so that it has a mean of 0 and standard deviation of 1 (approximately)

In [8]:
```
df['MonthlyCharges'] = np.log(df['MonthlyCharges'])
df['MonthlyCharges'] = (df['MonthlyCharges'] - df['MonthlyCharges'].mean()

df['TotalCharges'] = np.log(df['TotalCharges'])
df['TotalCharges'] = (df['TotalCharges'] - df['TotalCharges'].mean()) / df

df['tenure'] = (df['tenure'] - df['tenure'].mean()) / df['tenure'].std()

df[['tenure', 'MonthlyCharges', 'TotalCharges']].describe()
```

Out[8]:

	tenure	MonthlyCharges	TotalCharges
count	7.032000e+03	7.032000e+03	7.032000e+03
mean	-9.952853e-17	-9.851808e-16	-2.622097e-16
std	1.000000e+00	1.000000e+00	1.000000e+00
min	-1.280157e+00	-1.882268e+00	-2.579056e+00
25%	-9.542285e-01	-7.583727e-01	-6.080585e-01
50%	-1.394072e-01	3.885103e-01	1.950521e-01
75%	9.198605e-01	8.004829e-01	8.382338e-01
max	1.612459e+00	1.269576e+00	1.371323e+00

One-hot encoding categorical variables: As you can see from the data, there are many categorical variables. Let's first take a look at the number of unique values each column has. After that, use one-hot encoding technique to turn these columns into values of 0s and 1s. (Tip: read up One-hot encoding online like this: [https://www.geeksforgeeks.org/ml-one-hot-encoding-of-datasets-in-python/ (https://www.geeksforgeeks.org/ml-one-hot-encoding-of-datasets-in-python/)](https://www.geeksforgeeks.org/ml-one-hot-encoding-of-datasets-in-python/))

Step 7: Find out the number of unique values in each column.

In [9]:
```
# df.nunique()

continuous_vars = list(df.describe().columns)
continuous_vars
```

Out[9]:
```
['SeniorCitizen', 'tenure', 'MonthlyCharges', 'TotalCharges', 'Churn']
```

Q: Which are the variables with 2 to 4 unique values?

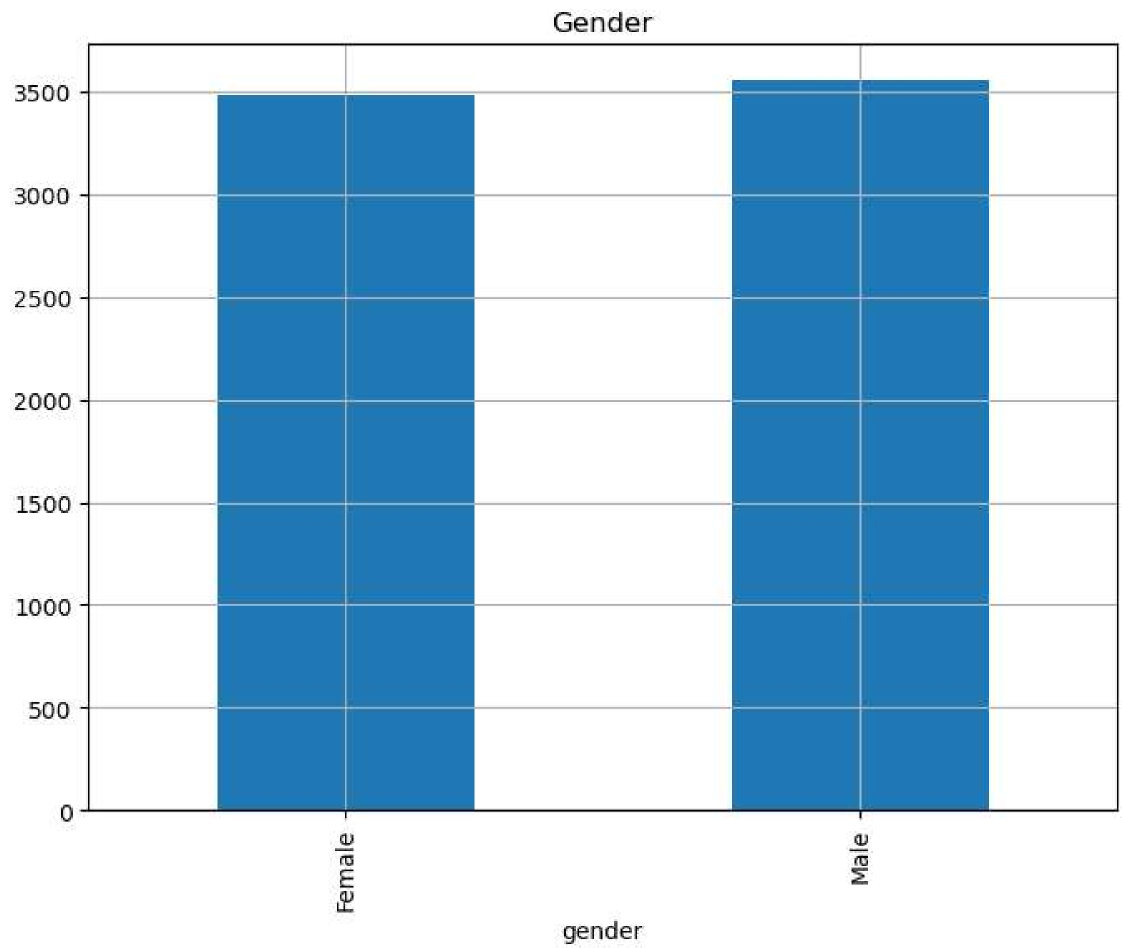
In [10]:
```
# df.nunique()[(df.nunique() >= 2) & (df.nunique() <= 4)]

for col in list(df.columns):
    if 2 <= df[col].nunique() <= 4:
        print(col, df[col].nunique())
```

```
gender 2
SeniorCitizen 2
Partner 2
Dependents 2
PhoneService 2
MultipleLines 3
InternetService 3
OnlineSecurity 3
OnlineBackup 3
DeviceProtection 3
TechSupport 3
StreamingTV 3
StreamingMovies 3
Contract 3
PaperlessBilling 2
PaymentMethod 4
Churn 2
```

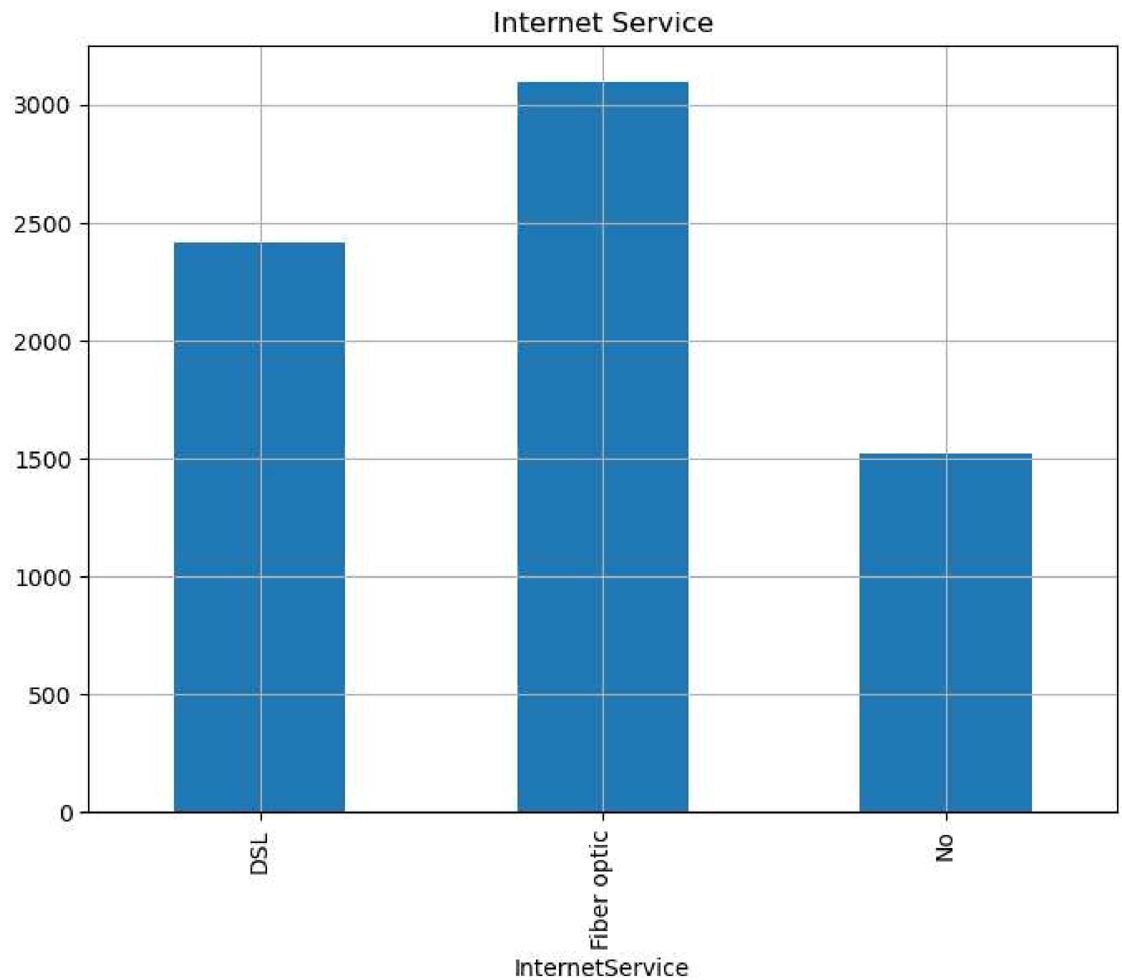
Q: Find out the distributions of the values stored in i) Gender ii) InternetService and iii) PaymentMethod.

In [11]:
```
df.groupby('gender').count()['customerID'].plot(
    kind='bar', grid=True, figsize=(8,6), title='Gender')
plt.show()
```

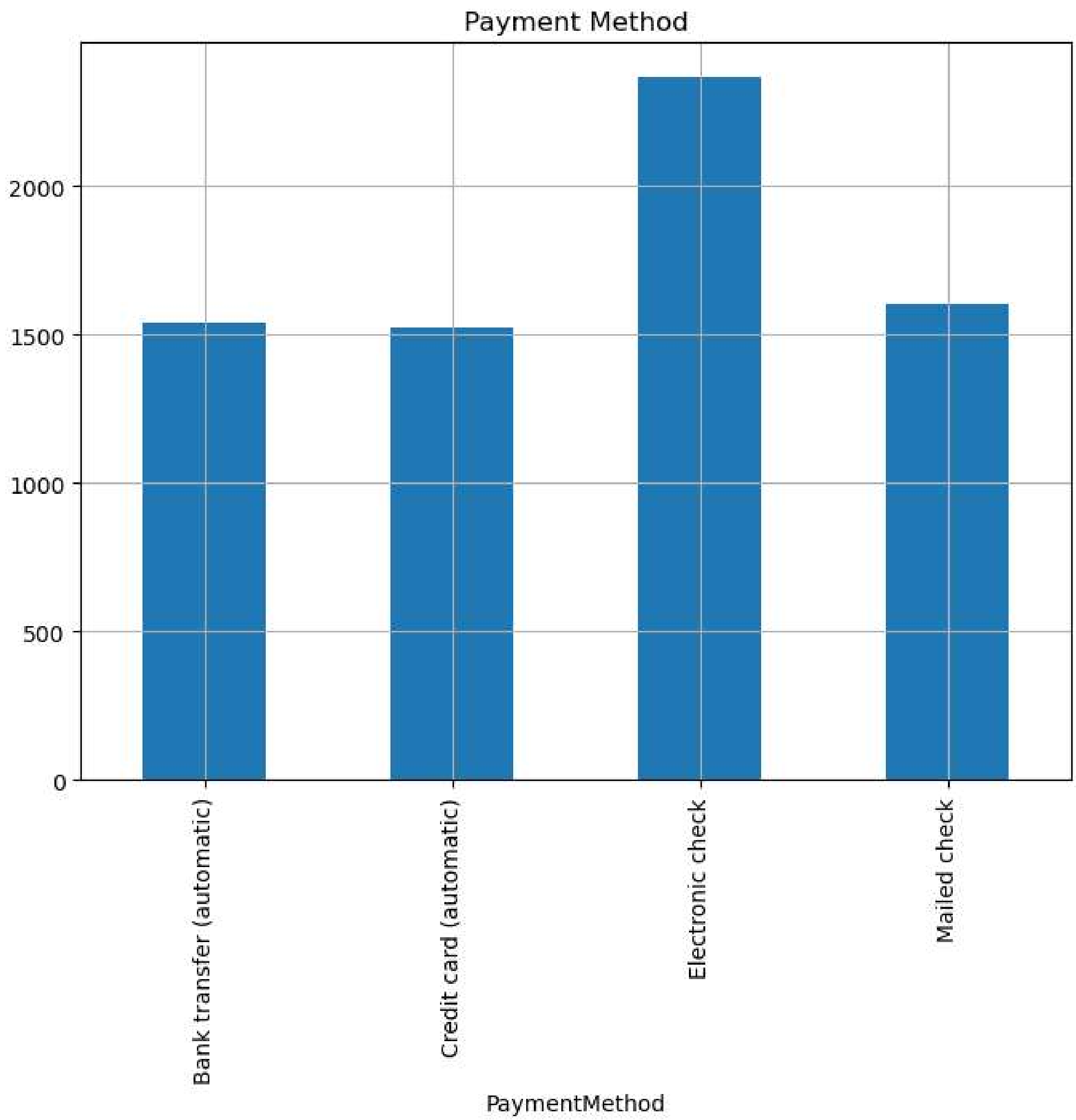


In [12]:

```
df.groupby('InternetService').count()['customerID'].plot(
    kind='bar', grid=True, figsize=(8,6), title='Internet Service')
plt.show()
```



In [13]:
```
df.groupby('PaymentMethod').count()['customerID'].plot(
    kind='bar', grid=True, figsize=(8,6), title='Payment Method')
plt.show()
```



Step 8: Perform One-hot encoding to all columns except tenure, MonthlyCharges, TotalCharges and Churn as well as those with lower than 5 unique values in respective columns.

In [14]:

```
dummy_cols = []

sample_set = df[['tenure', 'MonthlyCharges', 'TotalCharges', 'Churn']].copy

for col in list(df.columns):
    if col not in ['tenure', 'MonthlyCharges', 'TotalCharges', 'Churn'] and
        dummy_vars = pd.get_dummies(df[col])
        dummy_vars.columns = [col + str(x) for x in dummy_vars.columns]
        sample_set = pd.concat([sample_set, dummy_vars], axis=1)

sample_set
```

0	-1.280157	-1.054244	-2.281382	0	1	0
1	0.064298	0.032896	0.389269	0	0	1
2	-1.239416	-0.061298	-1.452520	1	0	1
3	0.512450	-0.467578	0.372439	0	0	1
4	-1.239416	0.396862	-1.234860	1	1	0
...	...	...	...	...	...	...
7038	-0.343113	0.702899	0.422797	0	0	1
7039	1.612459	1.033378	1.265008	0	1	0
7040	-0.872746	-1.068398	-0.702928	0	1	0
7041	-1.157934	0.482708	-0.781604	1	0	1
7042	1.368012	1.072863	1.218001	0	0	1

7032 rows × 47 columns

Step 9: Create features and target_var consisting the correct corresponding column names from sample_set

In [15]:

```
target_var = 'Churn'
features = [x for x in list(sample_set.columns) if x != target_var]
```

ANN with Keras

For building ANN models in Python, we are going to use keras package, which is a high-level neural networks library. For more details, we recommend you visit their official documentation at the following link: [https://keras.io/ (https://keras.io/)](https://keras.io/). Before we can use this package for building ANN models, we need to install two packages: tensorflow and keras. The keras package uses tensorflow as a backend for building neural network models, so we need to install tensorflow first. You can install these two packages using the following pip commands in your Terminal:

```
pip install tensorflow
pip install keras
```

if you wish to use conda to install, then use the following command:

```
conda install keras
```

Step 10: Build a neural network model with one hidden layer using keras. Import Sequential from keras.model and Dense from keras.layers. Create a model using the Sequential model. Use the following parameters:

- select relu as activation function for the input layer (set output units = 16)
- select relu as activation function for the hidden layer (set output units = 8)
- select sigmoid as activation function for the ouput layer (set output units = 1)

In [16]:
```
from keras.models import Sequential
from keras.layers import Dense

model = Sequential()
model.add(Dense(16, input_dim=len(features), activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

Step 11: The final step to build a neural network model with the keras package is to compile this model. Use the adam optimizer. Select binary_crossentropy as the loss function, and the accuracy metric to evaluate the model performance during training.

In [17]:
```
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accur
```

Step 12: Split the dataset to training and testing sample sets. Set 70% for training and 30% for testing.

In [18]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(sample_set[features], s
```

Step 13: Train the neural network model using epochs = 50, and batch size of 100

In [19]:
```
model.fit(X_train, y_train, epochs = 50, batch_size = 100)
```

```
Epoch 1/50
50/50 [==============================] - 1s 3ms/step - loss: 0.5948 - a
ccuracy: 0.6633
Epoch 2/50
50/50 [==============================] - 0s 3ms/step - loss: 0.4661 - a
ccuracy: 0.7702
Epoch 3/50
50/50 [==============================] - 0s 3ms/step - loss: 0.4305 - a
ccuracy: 0.7962
Epoch 4/50
50/50 [==============================] - 0s 2ms/step - loss: 0.4206 - a
ccuracy: 0.8005
Epoch 5/50
50/50 [==============================] - 0s 3ms/step - loss: 0.4155 - a
ccuracy: 0.8043
Epoch 6/50
50/50 [==============================] - 0s 2ms/step - loss: 0.4127 - a
ccuracy: 0.8052
Epoch 7/50
```

Note: As you can see from this output, loss typically decreases and the accuracy (acc) improves in each epoch. However, the rate of model performance improvement decreases over time. As you can see from this output, there are big improvements in the loss and accuracy measures in the first few epochs and the amount of performance gain decreases over time. You can monitor this process and decide to stop when the amount of performance gain is minimal.

Model evaluations

Now that we have built our first neural network model, let's evaluate its performance. We are going to look at the overall accuracy, precision, and recall, as well as the receiver operating characteristic (ROC) curve and area under the curve (AUC). First, execute the following code:

In [20]:
```
y_pred = model.predict(X_test)
y_pred
```

```
66/66 [==============================] - 0s 2ms/step
```

Out[20]:
```
array([[0.75698787],
       [0.50597537],
       [0.45165247],
       ...,
       [0.46415955],
       [0.3830204 ],
       [0.21530871]], dtype=float32)
```

Step 14: Print the following information: Accuracy, precision and recall for the above predictions.

In [21]:
```
y_pred = y_pred.argmax(axis=1)
```

In [22]:
```
from sklearn.metrics import classification_report

print(classification_report(y_test, y_pred, zero_division=1))
```

```
              precision    recall  f1-score   support

           0       0.73      1.00      0.85      1549
           1       1.00      0.00      0.00       561

    accuracy                           0.73      2110
   macro avg       0.87      0.50      0.42      2110
weighted avg       0.80      0.73      0.62      2110
```

Step 15: Compute the AUC numbers

In [23]:
```
from sklearn.metrics import roc_auc_score

score = roc_auc_score(y_test, y_pred)
print(f"AUC: {score:.2f}")
```

```
AUC: 0.50
```

Step 16: visualize this data in the ROC curve

In [24]:
```
from sklearn.metrics import roc_curve

fpr, tpr, _ = roc_curve(y_test,  y_pred)

plt.plot(fpr, tpr)
plt.plot([0, 1], [0, 1], color='gray', linestyle='--')
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```