LESSON18 SCALING AND ENCODING Last update June 1,2021

Scaling is used to normalize all input data into a predefined range between 0 to 1. Scaling lets the classifiers be more accurate. Unscaled data can also slow down or even prevent the convergence of many gradient-based estimators.

We will use the scalers from sklearn. Sklearn has many scalers:

Standard Scaler

A **StandardScaler** removes the mean and scales the data to unit variance. StandardScaler are very sensitive to the presence of outliers. Outliers are observations that lie outside the distance from other values in a in a data sample.

The standard score of a sample x is calculated as:

z = (x - u) / s

Where **u** is the mean of the training samples or 0 if the parameter **with_mean**=False, and **s** is the standard deviation of the training samples or 1 if the parameter **with_std**=False in the following Scaler constructor.

Constructor

```
StandardScaler(*, copy=True, with_mean=True, with_std=True)
```

Constructor paramaters

Parameter	Description
Copy Boolean	Optional If False try to avoid a copy and do inplace scaling instead.
with_mean Boolean	True by default, If True, center the data before scaling.
with_std Boolean	True by default, If True, scale the data to unit variance (or equivalently, unit standard deviation).

Attributes

Attribute	Description
scale_ndarray or None, shape (n_features,)	Per feature relative scaling of the data. This is calculated using `np.sqrt(var_)`. Equal to `None` when `with_std=False`.
mean_ ndarray or None, shape (n_features,)	The mean value when `with_mean=False`.for each feature in the training set. Equal to `None`
var_ndarray or None, shape (n_features,)	The variance for each feature in the training set. Used to compute `scale_`. Equal to `None` when `with_std=False`.
n_samples_seen_int or array, shape (n_features,)	The number of samples processed by the estimator for each feature. If there are not missing samples, the `n_samples_seen` will be an integer, otherwise it will be an array. Will be reset on new calls to fit, but increments across `partial_fit` calls.

Methods

Method	Description
`fit`(X[, y])	Compute the mean and std to be used for later scaling.
`fit_transform`(X[, y])	Fit to data, then transform it.
`get_params`([deep])	Get parameters for this estimator.
`inverse_transform`(X[, copy])	Scale back the data to the original representation
`partial_fit`(X[, y])	Online computation of mean and std on X for later scaling.
`set_params`(params)**	Set the parameters of this estimator.
`transform`(X[, copy])	Perform standardization by centering and scaling

Example program using Standard Scaller

```
# standard scaler
from sklearn.preprocessing import StandardScaler

# data of x,y pairs
data = [[20, 50], [15, 80], [10, 70], [30, 80]]

# make scaler
scaler = StandardScaler()

# fit data
scaler.fit(data)

# get mean and print
mean = scaler.mean_
print("mean:",mean)
```

```
mean: [18.75 70.  ]
```

```
# transform data
transform = scaler.transform(data)
print(transform)
```

```
unscaled data: [[20, 50], [15, 80], [10, 70], [30, 80]]
scaled data:
[[ 0.16903085 -1.63299316]
 [-0.50709255  0.81649658]
 [-1.18321596  0.        ]
 [ 1.52127766  0.81649658]]
```

```
# test scaler
test1 = scaler.transform([[12, 60]])
print("test known data [15, 80]:",test1)

test2 = scaler.transform([[12, 60]])
print("test unknown data [12, 60]:",test1)
```

```
test known data [15, 80]: [[-0.9127666  -0.81649658]]
test unknown data [12, 60]: [[-0.9127666  -0.81649658]]
```

Here is the complete program:

```
# standard scaler
from sklearn.preprocessing import StandardScaler

# data of x,y pairs
data = [[20, 50], [15, 80], [10, 70], [30, 80]]

# make scaler
scaler = StandardScaler()
```

```
# fit data
scaler.fit(data)

# get mean and print
mean = scaler.mean_
print("mean:",mean)

# transform data
transform = scaler.transform(data)

# test scaler
test1 = scaler.transform([[12, 60]])
print("test known data [15, 80]:",test1)
test2 = scaler.transform([[12, 60]])
print("test unknown data [12, 60]:",test1)
```

OTHER SCALERS

MinMax Scaler

MinMaxScaler rescales the data set such that all feature values are in the range [0, 1] MinMaxScaler are also is very sensitive to the presence of outliers.

MaxAbsScaler

MaxAbsScaler is same as MinMax Scaler such that the absolute values are mapped in the range [0, 1].

ENCODING

Categorial Features

Categorical features (non numeric) can only take on a limited, and usually fixed, number of possible values. For example, if a dataset is about information related to users, then you will typically find features like country, gender, age group, etc. Alternatively, if the data you're working with is related to products, you will find features like product type, manufacturer, seller and so on.

These features are typically stored as text values which represent various traits of the observations. For example, gender is described as Male (M) or Female (F), product type could be described as electronics, apparels, food etc.

Nominal Features

Nominal features are features where the categories are labeled without any order of precedence.

Ordinal Features

Ordinal features are features which have some order associated with them. For example, a feature like economic status, with three categories: low, medium and high, which have an order associated with them.

Continuous Features

Continuous Features are numeric variables that have an infinite number of values between any two values. A continuous variable can be numeric or a date/time.

DATA CONSTRAINTS

- Categorical features may have a very large number of levels, known as high cardinality, (for example, cities or URLs), where most of the levels appear in a relatively small number of instances.

- Many machine learning models, such as regression or SVM, are algebraic. This means that their input must be numerical. To use these models, categories must be transformed into numbers first, before you can apply the learning algorithm on them.

- While some Machine Learning packages or libraries might transform categorical data to numeric automatically based on some default embedding method, many other Machine Learning packages don't support such inputs.

- For machine learning, categorical data doesn't contain the same context or information that humans can easily associate and understand. For example, when looking at a feature called City with three cities New York, New Jersey and New Delhi, humans can infer that New York is closely related to New Jersey as they are from same country, while New York and New Delhi are much different. But for the model, New York, New Jersey and New Delhi, are just three different levels (possible values) of the same feature City. If you don't specify the additional contextual information, it will be impossible for the model to differentiate between highly different levels.

ENCODING CATEGORIAL DATA

- Replacing values
- Encoding labels
- One-Hot encoding
- Binary encoding
- Backward difference encoding
- Miscellaneous features

Replacing Values

We replace known labels with a numeric value using the pandas **replace** function. We replace labels with sequential values. We can use a hard coded method where we manually assign a number to a known label. We can also use a automatic method where we assign ascending sequential numbers to unknown label set.

In this example we will assign numbers to a column of country names. We first make a one column data frame of column names.

```
# replacing values
import pandas as pd

# make data frame that has a column of countries
df = pd.DataFrame({'countries':['Canada','USA','Europe', 'India','China']})
print(df)
```

```
  countries
0    Canada
1       USA
2    Europe
3     India
4     China
```

We make a dictionary of county names and assigned numbers.

```
# replacing labels with sequential number
country_map = {'countries': {'Canada': 1, 'USA': 2, 'Europe': 3, 'India': 4,'China': 5}}
```

We can also make the country map from a list of country categories and assigning sequential numbers to them.

```
# alternatively you can also look up countries automatically
# and assign a sequential number to each
labels = df['countries'].astype('category').cat.categories.tolist()
country_map = {'countries' : {k: v for k,v in zip(labels,list(range(1,len(labels)+1)))}}

# print country map
print(country_map)
```

```
{'countries': {'Canada': 1, 'USA': 2, 'Europe': 3, 'India': 4, 'China': 5}}
```

```
# we now replace the country labels with codes
df.replace(country_map, inplace=True)
print(df)
```

```
   countries
0          1
1          2
2          3
3          4
4          5
```

It's a good practice to typecast categorical features to a **category** dtype because they make the operations on such columns much faster than the **object** dtype. Here is the complete program:

```
# replacing values

import pandas as pd
# make data frame that has a column of countries
df = pd.DataFrame({'countries':['Canada','USA','Europe', 'India','China']})
print(df)

# replacing labels with sequential number
country_map = {'countries': {'Canada': 1, 'USA': 2, 'Europe': 3, 'India': 4,'China': 5}}

# alternatively you can also look up countries automatically
# and assign a sequential number to each

labels = df['countries'].astype('category').cat.categories.tolist()
country_map = {'countries' : {k: v for k,v in zip(labels,list(range(1,len(labels)+1)))}}

# print country map
print(country_map)

# we now replace the country labels with codes
df.replace(country_map, inplace=True)
print(df)
```

Label Encoding

Label Encoding allows you to convert each category label in a column to a number. Numerical labels are always between 0 and number of categories-1.

In the following **Gender** column we have made a new column called **GenderCodes,** where we have assigned the number 1 to Male and the number 0 to Female using the sklearn LabelEncoder

```
# using label encoder  to encode gender to numbers
from sklearn.preprocessing import LabelEncoder

# make 1 column data frame of genders
df = pd.DataFrame({'gender':['Male','Male','Female', 'Male']})
print(df)
```

```
   gender
0    Male
1    Male
2  Female
3    Male
```

```
# make label encoder
labelEncoder = LabelEncoder()

# make gender encoded column
df['gender_code'] = labelEncoder.fit_transform(df['gender'])
print(df)
```

```
   gender  gender_code
0    Male            1
1    Male            1
2  Female            0
3    Male            1
```

Here is the complete program:

```
# using label encoder  to encode gender to numbers
from sklearn.preprocessing import LabelEncoder

# make 1 column data frame of genders
df = pd.DataFrame({'gender':['Male','Male','Female', 'Male']})
print(df)

# make label encoder
labelEncoder = LabelEncoder()

# make gender encoded column
df['gender_code'] = labelEncoder.fit_transform(df['gender'])
print(df)
```

One-Hot encoding using the pandas get_dummies() function

The basic strategy is to convert each category value into a new column and assign a 1 or 0 (True/False) values to the column. This has the benefit of not weighting a value improperly. There are no columns that have a 1 in them for a particular row.

There are many libraries out there that support one-hot encoding but the simplest one is using pandas **get_dummies()** method.

This function is named this way because it creates dummy/indicator variables (1 or 0). The **get_dummies()** method has three important arguments.

get_dummies(dataframe,column,prefix)

The first one is the **DataFrame** you want to encode on, second **'column'** is the column you want to do encoding on, and third, the **'prefix'** argument which lets you specify the prefix for the new columns that will be created after encoding.

We first make a 1 column dataframe of education then apply the **getDummies** method to the **education** column .

```
# one hot encoding using get_dummies method

# make data frame that has a column of countries
df = pd.DataFrame({'education':['High_School','College','University', 'University']})
print(df)
```

```
     education
0  High_School
1      College
2   University
3   University
```

```
# one hot encodeing using get dummies
df = pd.get_dummies(df, columns=['education'], prefix = ['education'])
print(df)
```

```
   education_College  education_High_School  education_University
0                  0                      1                     0
1                  1                      0                     0
2                  0                      0                     1
3                  0                      0                     1
```

The original education column has now been replaced with the encoded column's prefixed with "education"

Here is the complete program:

```
# one hot encoding using get_dummies method

# make data frame that has a column of countries
df = pd.DataFrame({'education':['High_School','College','University', 'University']})
print(df)

# one hot encoding using get dummies
df = pd.get_dummies(df, columns=['education'], prefix = ['education'])
print(df)
```

One Hot Encoding using sklearn OneHotEncoder

The One Hot Encoder produces a similar result as the panda **getDummies()** function. We again make a 1 column dataframe of education then apply the sklearn **OneHotDecoder**.

```
# one hot encoding using sklearn
import pandas as pd
from sklearn.preprocessing import OneHotEncoder

# make data frame that has a column of countries
df = pd.DataFrame({'education':['High_School','College','University', 'University']})
print(df)
```

```
     education
0  High_School
1      College
2   University
3   University
```

We then make the oneHotEncoder specifying handle_**unknown='ignore'** so it can ignore missing data. We use the OneHotEncoder **fit_transform** method on the dataframe Education column that produces a sparse matrix of row column coordinates and corresponding values

```
# make one hot encodes, ignore unknown column values handle_unknown='ignore'
oneHotEncoder = OneHotEncoder(handle_unknown='ignore')

# do one hot decoding
encoding = oneHotEncoder.fit_transform(df[['education']])

#  print row-column coordinates with corresponding values
print(encoding)
```

```
  (0, 1)    1.0
  (1, 0)    1.0
  (2, 2)    1.0
  (3, 2)    1.0
```

We then convert the sparse matrix to an array showing row-column values

```
# encoding as an array
encoding = encoding.toarray()
print(encoding)
```

```
[[0. 1. 0.]
 [1. 0. 0.]
 [0. 0. 1.]
 [0. 0. 1.]]
```

We now make another data frame with our encoded results

```
# make a data frame of encoded results
encoded_education = pd.DataFrame(encoding)
print(encodes_education)
```

```
     0    1    2
0  0.0  1.0  0.0
1  1.0  0.0  0.0
2  0.0  0.0  1.0
3  0.0  0.0  1.0
```

Our data frame has no column names just column numbers. We can get the encoded column names from the oneHot Decoder **get_feature_names** function then assign the column names to our data frame.

```
# get encoded column names
encoded_columns = oneHotEncoder.get_feature_names(['education'])
print(encoded_columns )
```

```
['education_College' 'education_High_School' 'education_University']
```

```
# assign encoded column names
encoded_education.columns=encoded_columns
print(encoded_education)
```

```
   education_College  education_High_School  education_University
0                0.0                    1.0                   0.0
1                1.0                    0.0                   0.0
2                0.0                    0.0                   1.0
3                0.0                    0.0                   1.0
```

Finally we add the encoded columns back to our original data frame.

```
# add encoded columns to dataframe
df = df.join(encoded_education)
print(df)
```

```
     education  education_College  education_High_School  education_University
0  High_School                0.0                    1.0                   0.0
1      College                1.0                    0.0                   0.0
2   University                0.0                    0.0                   1.0
3   University                0.0                    0.0                   1.0
```

Here is the Complete Program:

```
# one hot encoding using sklearn

import pandas as pd
from sklearn.preprocessing import OneHotEncoder

# make data frame that has a column of countries
df = pd.DataFrame({'education':['High_School','College','University', 'University']})
print(df)
# make one hot encodes, ignore unknown column values handle_unknown='ignore'
oneHotEncoder = OneHotEncoder(handle_unknown='ignore')

# do one hot decoding
encoding = oneHotEncoder.fit_transform(df[['education']])

#  print row-column coordinates with corresponding values
print(encoding)

# encoding as an array
encoding = encoding.toarray()
print(encoding)

# make a data frame of encoded results
encoded_education = pd.DataFrame(encoding)
print(encodes_education)

# get encoded column names
encoded_columns = oneHotEncoder.get_feature_names(['education'])
print(encoded_columns )


# assign encoded column names
encoded_education.columns=encoded_columns
print(encoded_education)


# add encoded columns to dataframe
df = df.join(encoded_education)
print(df)
```

EXAMPLE USING SCALING AND ENCODING

We start with a data set with the following columns:

Name	age	gender	Country	Education	salary	range

We want to be able predict salary range as Low, Medium or High depending on their age, gender, country, education and salary.

We have made a small csv file with some of the data called salaries.csv. Intentially we have left some blank values so we can test for NaN'S. Note: You actually need at least 50 enntries to get accurate results.

salaries.csv

```
name,age,gender,country,education,salary,range
tom smith,34,Male,Canada,University,75000,High
bob jones,36,Male,USA,High School,25000,Low
dave smith,24,Male,Europe,College,,Medium
bill smith,28,Female,India,High School,25000,Medium
jay smith,23,Male,China,High School,25000,Low
sue smith,44,Female,USA,University,55000,High
sid jones,42,Male,China,High School,25000,Medium
mary smith,34,Male,India,College,,Medium
dodo smith,34,Male,Europe,College,20000,Low
sid smith,24,Male,Europe,College,60000,High
```

We first read in the salaries.csv into a dataframe

```
# example using scaling and encoding

import pandas as pd;
import matplotlib.pyplot as plt
import seaborn as sns

# read in salaries.csv file
df = pd.read_csv("salaries.csv")

# Printing data frame
print(df)
```

```
          name  age  gender country    education   salary   range
0    tom smith   34    Male  Canada   University  75000.0    High
1    bob jones   36    Male     USA  High School  25000.0     Low
2   dave smith   24    Male  Europe      College      NaN  Medium
3   bill smith   28  Female   India  High School  25000.0  Medium
4    jay smith   23    Male   China  High School  25000.0     Low
5    sue smith   44  Female     USA   University  55000.0    High
6    sid jones   42    Male   China  High School  25000.0  Medium
7   mary smith   34    Male   India      College      NaN  Medium
8   dodo smith   34    Male  Europe      College  20000.0     Low
9    sid smith   24    Male  Europe      College  60000.0    High
```

Next we check for NaNs. NaNs are empty cells in the csv file

```
# check data frame for nulls
print("Number Nulls: ",df.isnull().values.sum())
```

```
Number Nulls:  2
```

```
# check columns for null
print(df.isnull().sum())
```

```
name          0
age           0
gender        0
country       0
education     0
salary        2
range         0
dtype: int64
```

We have found 2 NaN's in our data frame, both in the salary column. The best thing to do is replace the NaN'S with an average value of the **country** column, where the NaN's were found. Since salaries in each country have a similar rate.

```
# replace any null salaries with average of the country
df['salary'] = df['salary'].fillna(df.groupby('country')['salary'].transform('mean'))
```

We again verify all names have been replaced by a mean value corresponding to the country that the NaN's were found**.**

```
# check data frame for nulls
print("Number Nulls: ",df.isnull().values.sum())
```

```
Number Nulls:  0
```

```
# check columns for null
print(df.isnull().sum())
```

```
name          0
age           0
gender        0
country       0
education     0
salary        0
range         0
dtype: int64
```

```
print(df)
```

```
          name  age  gender country    education   salary   range
0   tom smith   34    Male  Canada   University  75000.0    High
1    bob jones   36    Male     USA  High School  25000.0     Low
2  dave smith   24    Male  Europe      College  40000.0  Medium
3  bill smith   28  Female   India  High School  25000.0  Medium
4   jay smith   23    Male   China  High School  25000.0     Low
5   sue smith   44  Female     USA   University  55000.0    High
6   sid jones   42    Male   China  High School  25000.0  Medium
7  mary smith   34    Male   India      College  25000.0  Medium
8  dodo smith   34    Male  Europe      College  20000.0     Low
9   sid smith   24    Male  Europe      College  60000.0    High
```

All NaN’s are now gone and replace by a mean value accordingly to the country in which the NaN’s have been found. Next we drop the name column since the person’s name has nothing to do with salaries.

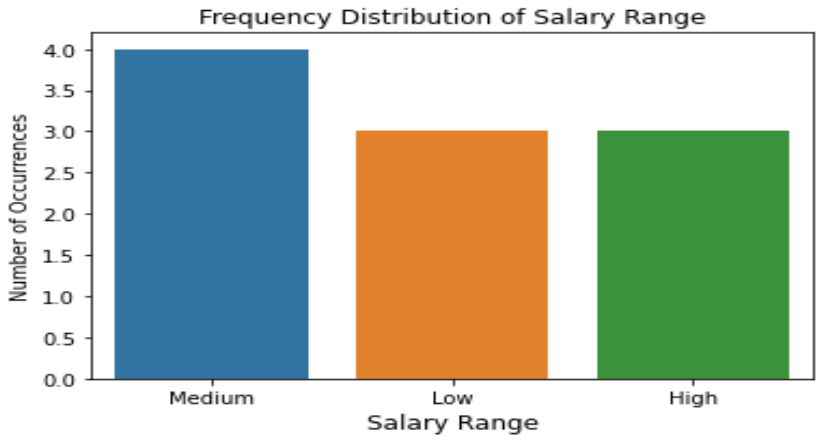
```
# we need to drop name column
df = df.drop("name",axis='columns')
print(df)
```

```
   age  gender country    education   salary   range
0   34    Male  Canada   University  75000.0    High
1   36    Male     USA  High School  25000.0     Low
2   24    Male  Europe      College  40000.0  Medium
3   28  Female   India  High School  25000.0  Medium
4   23    Male   China  High School  25000.0     Low
5   44  Female     USA   University  55000.0    High
6   42    Male   China  High School  25000.0  Medium
7   34    Male   India      College  25000.0  Medium
8   34    Male  Europe      College  20000.0     Low
9   24    Male  Europe      College  60000.0    High
```

Plotting Salary Distribution

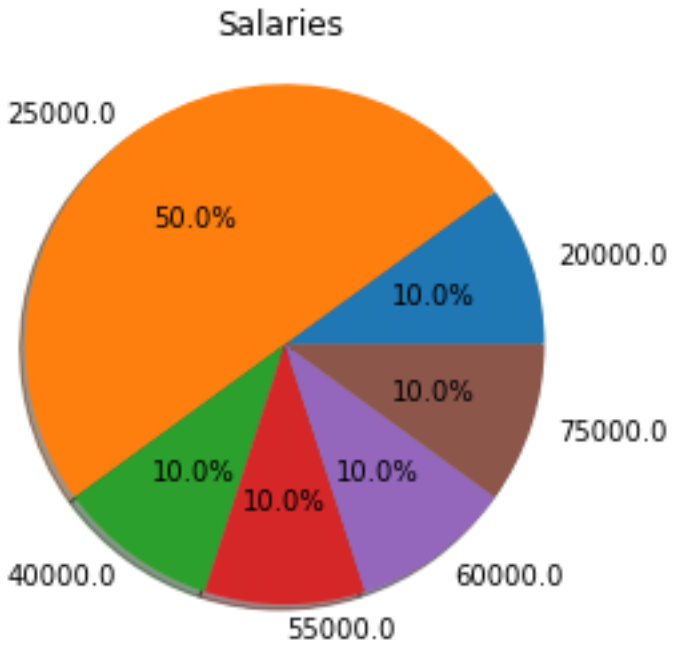
We plot a barplot of salary categories of Low ,Medium and High using the seaborn **barplot** function.

```
# plotting salary distribution as a bar chart
salary_range_counts = df['range'].value_counts()
sns.barplot(x=salary_range_counts.index,y= salary_range_counts)
plt.title('Frequency Distribution of Salary Range')
plt.ylabel('Number of Occurrences')
plt.xlabel('Salary Range', fontsize=12)
plt.show()
```



We then show a pie chart of salaries using the **pie** function:

```
# plotting salary distribution as a  pie chart
labels = df['salary'].astype('category').cat.categories.tolist()
counts = df['salary'].value_counts()
sizes = [counts[var_cat] for var_cat in labels]
fig1, ax1 = plt.subplots()
ax1.pie(sizes, labels=labels, autopct='%1.1f%%', shadow=True) #autopct is show the %
on plot
ax1.axis('equal')
plt.show()
```



Encoding the Data

We use the hard coding method to encode country data using a dictionary of known countries in the country column and assigning a number to each one of them.

```
# encoding the data

# replacing country labels with sequential number

# make map of country codes
country_map = {'country': {'Canada': 1, 'USA': 2, 'Europe': 3, 'India': 4,'China': 5}}
```

We can also make the country map automatically from a list of country categories and assigning sequential numbers to them.

```
# look up countries automatically and assign a sequential number to each
labels = df['country'].astype('category').cat.categories.tolist()
country_map = {'country' : {k: v for k,v in zip(labels,list(range(1,len(labels)+1)))}}
```

```
{'country': {'Canada': 1, 'China': 2, 'Europe': 3, 'India': 4, 'USA': 5}}
```

We then replace the country column with the assigned numbers using the pandas replace function and the dictionary mapping country names to numbers.

```
#  replace the country labels with codes
df.replace(country_map, inplace=True)
print(df)
```

```
   age  gender  country    education   salary   range
0   34    Male        1   University  75000.0    High
1   36    Male        5  High School  25000.0     Low
2   24    Male        3      College  40000.0  Medium
3   28  Female        4  High School  25000.0  Medium
4   23    Male        2  High School  25000.0     Low
5   44  Female        5   University  55000.0    High
6   42    Male        2  High School  25000.0  Medium
7   34    Male        4      College  25000.0  Medium
8   34    Male        3      College  20000.0     Low
9   24    Male        3      College  60000.0    High
```

Replace Gender Column with 1 and 0

We now replace the Gender column with 1 and 0 to represent Male and Female. We first make a **LabelEncoder**

```
# replace Gender column with 1 and 0

 # make LabelEncoder
 from sklearn.preprocessing import LabelEncoder

# gender to numbers
labelEncoder = LabelEncoder()
df['gender_code'] = labelEncoder.fit_transform(df['gender'])
```

This make a new column called gender_code and next we drop the original gender column and print out the data frame.

```
df.drop('gender', axis='columns',inplace=True)
print(df)
```

```
   age  country    education   salary   range  gender_code
0   34        1   University  75000.0    High            1
1   36        5  High School  25000.0     Low            1
2   24        3      College  40000.0  Medium            1
3   28        4  High School  25000.0  Medium            0
4   23        2  High School  25000.0     Low            1
5   44        5   University  55000.0    High            0
6   42        2  High School  25000.0  Medium            1
7   34        4      College  25000.0  Medium            1
8   34        3      College  20000.0     Low            1
9   24        3      College  60000.0    High            1
```

Replace Education Column with One Hot Encoding

Next we use one hot encoding to replace the Education column with one hot encoding columns**.** We first do **One-Hot encoding** using the get_dummies() function.

```
# replace education column

# one hot encoding using get dummies
df2 = pd.get_dummies(df, columns=['education'], prefix = ['education'])

print(df2)
```

```
   education_College  education_High School  education_University
0                  0                      0                     1
1                  0                      1                     0
2                  1                      0                     0
3                  0                      1                     0
4                  0                      1                     0
5                  0                      0                     1
6                  0                      1                     0
7                  1                      0                     0
8                  1                      0                     0
9                  1                      0                     0
```

This has replaced the original education column with the new column of one hot encoded columns

We will also can do OneHotEncoding using sklearn **OneHotEncoder.**

```
# using one hot encoder
from sklearn.preprocessing import OneHotEncoder

# make one hot encodes, ignore unknown column values
oneHotEncoder = OneHotEncoder(handle_unknown='ignore')

# do one hot encoding
encoded_education =
pd.DataFrame(oneHotEncoder.fit_transform(df[['education']]).toarray())

# get encoded column names
encoded_columns = oneHotEncoder.get_feature_names(['education'])

# assign encoded column names
encoded_education.columns=encoded_columns
print(encoded_education)
```

```
[10 rows x 9 columns]
   education_College  education_High School  education_University
0                0.0                    0.0                   1.0
1                0.0                    1.0                   0.0
2                1.0                    0.0                   0.0
3                0.0                    1.0                   0.0
4                0.0                    1.0                   0.0
5                0.0                    0.0                   1.0
6                0.0                    1.0                   0.0
7                1.0                    0.0                   0.0
8                1.0                    0.0                   0.0
9                1.0                    0.0                   0.0
```

```
# add encoded columns to dataframe
df = df.join(encoded_education)

# print our education columns for verification
print(df[['education'] + df.columns[-3:].tolist()])
```

```
     education  education_College  education_High School  education_University
0   University                0.0                    0.0                   1.0
1  High School                0.0                    1.0                   0.0
2      College                1.0                    0.0                   0.0
3  High School                0.0                    1.0                   0.0
4  High School                0.0                    1.0                   0.0
5   University                0.0                    0.0                   1.0
6  High School                0.0                    1.0                   0.0
7      College                1.0                    0.0                   0.0
8      College                1.0                    0.0                   0.0
9      College                1.0                    0.0                   0.0
```

We drop the education column because it is no longer needed.

```
df.drop('education', axis='columns',inplace=True)
```

Encode Range Column

We do not need to encode the target salary range column, the classifier can handle this. But we still need a coded range for the scoring and accuracy calculations

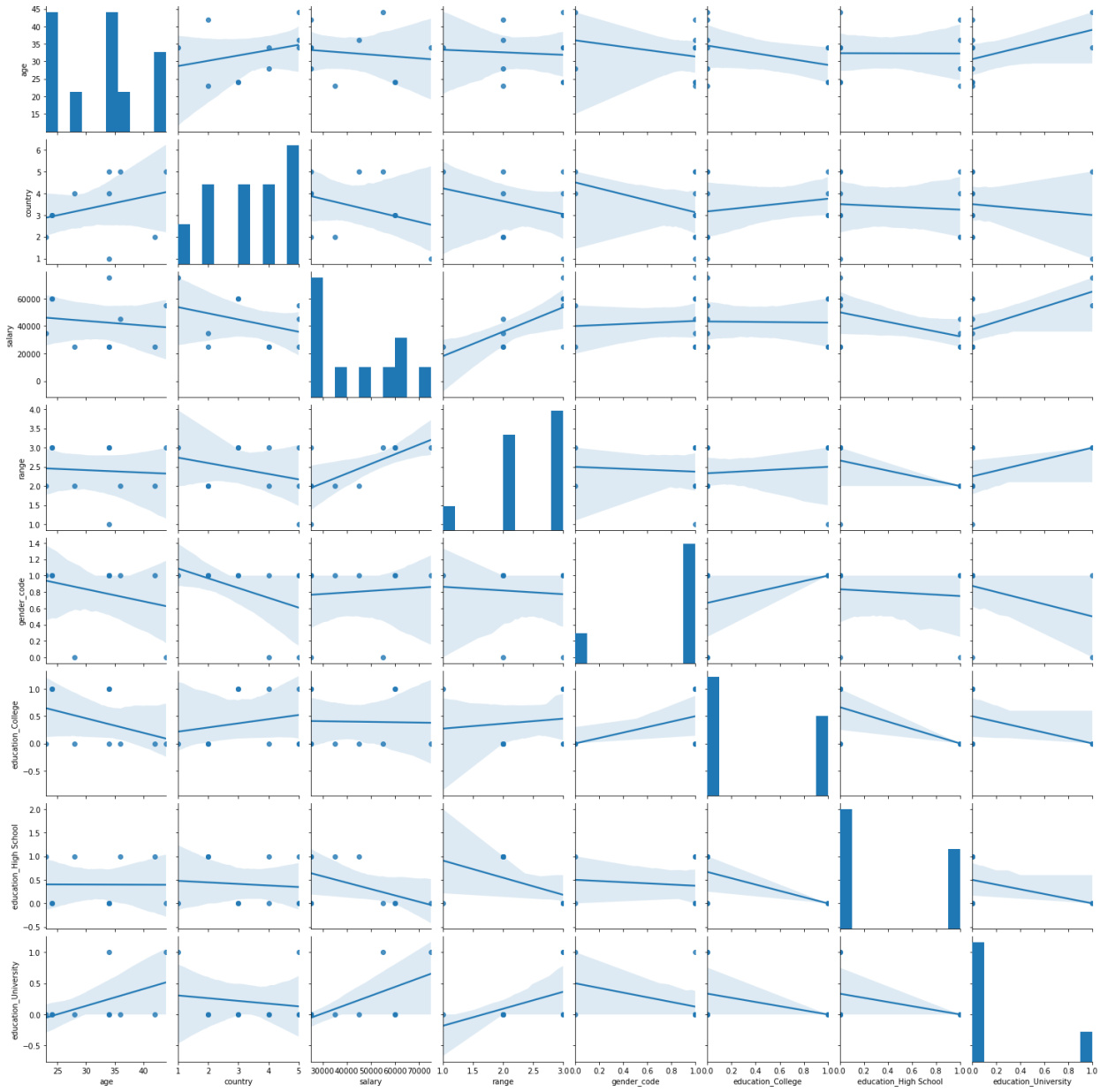
```
# encode range column
labelEncoder = LabelEncoder()
df['range_code'] = labelEncoder.fit_transform(df['range'])
```

```
    range  range_coded
0    High            0
1     Low            1
2  Medium            2
3  Medium            2
4     Low            1
5    High            0
6  Medium            2
7  Medium            2
8     Low            1
9    High            0
```

Identify trends using a Pair Plot

A Pair plot helps you identify upward and downward trends, and helps you identify patterns in the data set. For example higher education goes with larger salary.

```
# plot pair plot with regression line
sns.pairplot(df,kind="reg")
plt.show()
```

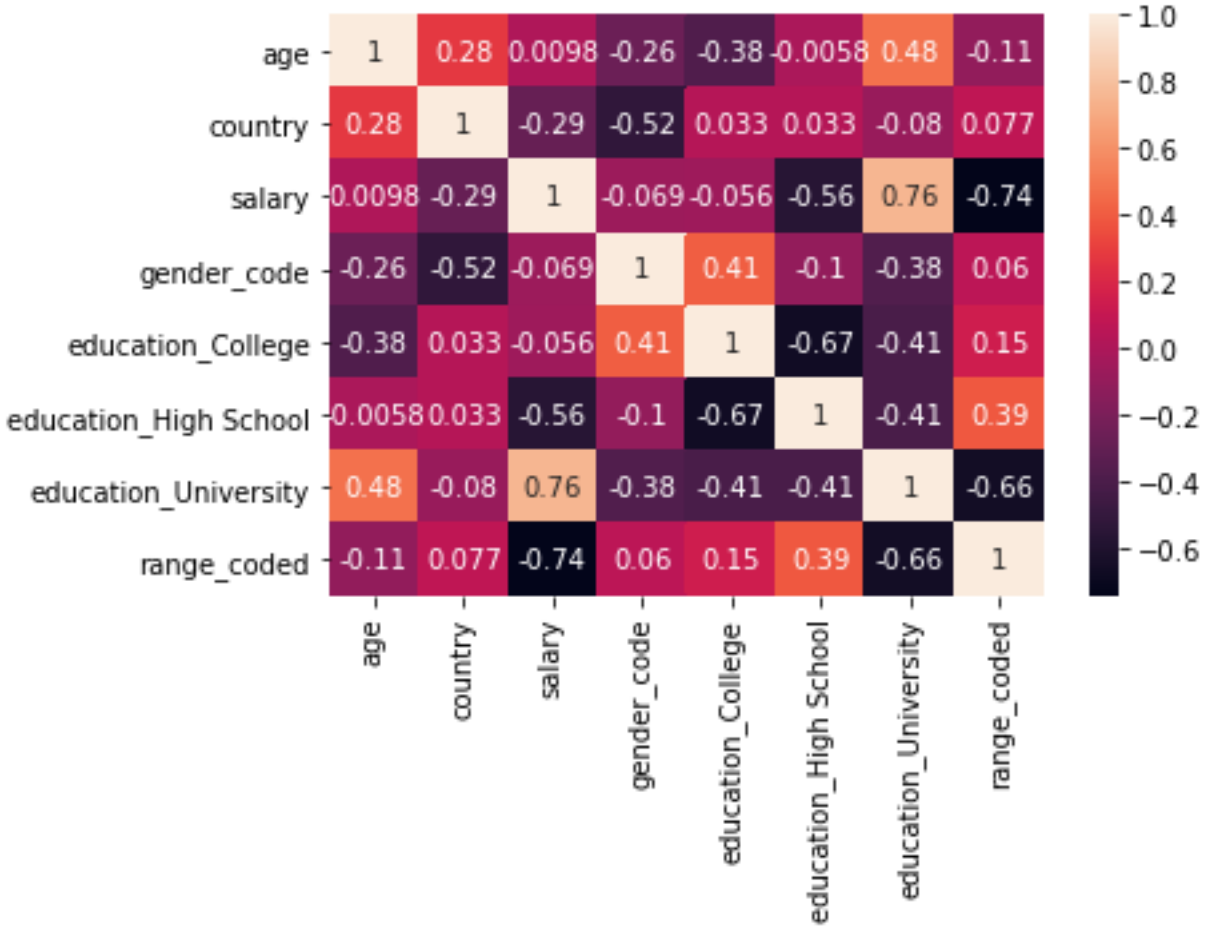


Plot Correlation Heat Map

We can also plot the correlation for each column pair, where correlation measures the trends, upwards, downwards or consolidated (flat)

```
# plot correlation heat map
# calculate the correlation matrix
corr = df.corr()
corr.style.background_gradient(cmap='coolwarm')
```

```
# plot the heatmap
sns.heatmap(corr, xticklabels=corr.columns,yticklabels=corr.columns,annot = True)
plt.show()
```



Scaling and splitting data into training and test sets.

Next we set the X feature columns and y target to columns to **'range'** and **'range_coded'**. The X feature columns are the independent variables of known data features like age,gender,country,education and salary. The y target columns are the independent variable y what we are going to predict like salary range from the X known dependent data.

```
# scaling the data

#set feature
X = df[['age','gender_code','country','education_College','education_High
School','education_University','salary']]
```

```
#target is salary range
y = df[['range','range_coded']]
```

The y data includes both range and range coded data. Our classifier can handle label categories but our error and accuracy calculations need numeric data that is contained in the range_coded column.

Scaling X Data

We scale the X data using the sklearn **StandardScaler** and then print out the results.

```
from sklearn.preprocessing import StandardScaler

# scale X data
scaler = StandardScaler()
scaler.fit(X)
X_scaled = scaler.transform(X)
print("scaled x")
print(X_scaled)
```

```
scaled x
[[ 0.24135945  0.5  -1.76140969 -0.81649658 -0.81649658  2.  2.06040846]
 [ 0.52531174  0.5   1.44115338 -0.81649658  1.22474487 -0.5  -0.68680282]
 [-1.17840202  0.5  -0.16012815  1.22474487 -0.81649658 -0.5 0.13736056]
 [-0.61049743 -2.    0.64051262 -0.81649658  1.22474487 -0.5 -0.68680282]
 [-1.32037817  0.5  -0.96076892 -0.81649658  1.22474487 -0.5 -0.68680282]
 [ 1.66112092 -2.    1.44115338 -0.81649658 -0.81649658  2.  0.96152395]
 [ 1.37716863  0.5   -0.96076892 -0.81649658  1.22474487 -0.5  -0.68680282]
 [ 0.24135945  0.5   0.64051262  1.22474487 -0.81649658 -0.5 -0.68680282]
 [ 0.24135945  0.5  -0.16012815  1.22474487 -0.81649658 -0.5  -0.96152395]
 [-1.17840202  0.5   -0.16012815  1.22474487 -0.81649658 -0.5   1.23624508]]
```

Splitting Data

We now split the data into train and test sets. Using sklearn function **train_test_split**

 # splitting the data

from sklearn.model_selection import train_test_split

 # split data
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.5, random_state=42)
print("X train data")
print(X_train)

```
X train data
[[-1.17840202  0.5  -0.16012815  1.22474487 -0.81649658 -0.5 0.13736056]
 [-1.17840202  0.5  -0.16012815  1.22474487 -0.81649658 -0.5  1.23624508]
 [-1.32037817  0.5  -0.96076892 -0.81649658  1.22474487 -0.5 -0.68680282]
 [-0.61049743 -2.    0.64051262 -0.81649658  1.22474487 -0.5 -0.68680282]
 [ 1.37716863  0.5 -0.96076892 -0.81649658  1.22474487 -0.5 -0.68680282]]
```

print("y train data")
print(y_train)

```
y train data
    range  range_coded
2  Medium            2
9    High            0
4     Low            1
3  Medium            2
6  Medium            2
```

print("X test data")
print(X_test)

```
X test data
[[ 0.24135945  0.5      -0.16012815  1.22474487 -0.81649658 -0.5   -0.96152395]
 [ 0.52531174  0.5       1.44115338 -0.81649658  1.22474487 -0.5  -0.68680282]
 [ 1.66112092 -2.        1.44115338 -0.81649658 -0.81649658  2.  0.96152395]
 [ 0.24135945  0.5      -1.76140969 -0.81649658 -0.81649658  2. 2.06040846]
 [ 0.24135945  0.5       0.64051262  1.22474487 -0.81649658 -0.5 -0.68680282]]
```

print("y test data")
print(y_test)

```
y test data
    range  range_coded
8     Low            1
1     Low            1
5    High            0
0    High            0
7  Medium            2
```

The training set is usually larger than the test set. But in our case we have made is 50/50 since we do not have too much data.

Classifying the Data

We first use the train data then use the test data to measure the accuracy. We make a sklearn **stochastic gradient descent** SGD classifier. We first train the data then get the predicted data from the x test data set. Finally we print out the y_test data and compare to the y predicted data

```
# classifying the data

from sklearn.linear_model import SGDClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error

import numpy as np

# make sgd clasifier
classifier = SGDClassifier(random_state=42, max_iter=30, tol=1e-3)

# fit the model
classifier.fit(X_train,y_train['range_coded'])

# predict data
y_pred = classifier.predict(X_test)

# print test and predicted scores
print("actual    predicted");
for a,p in zip(y_test['range'],y_pred):
```

print(a,p)

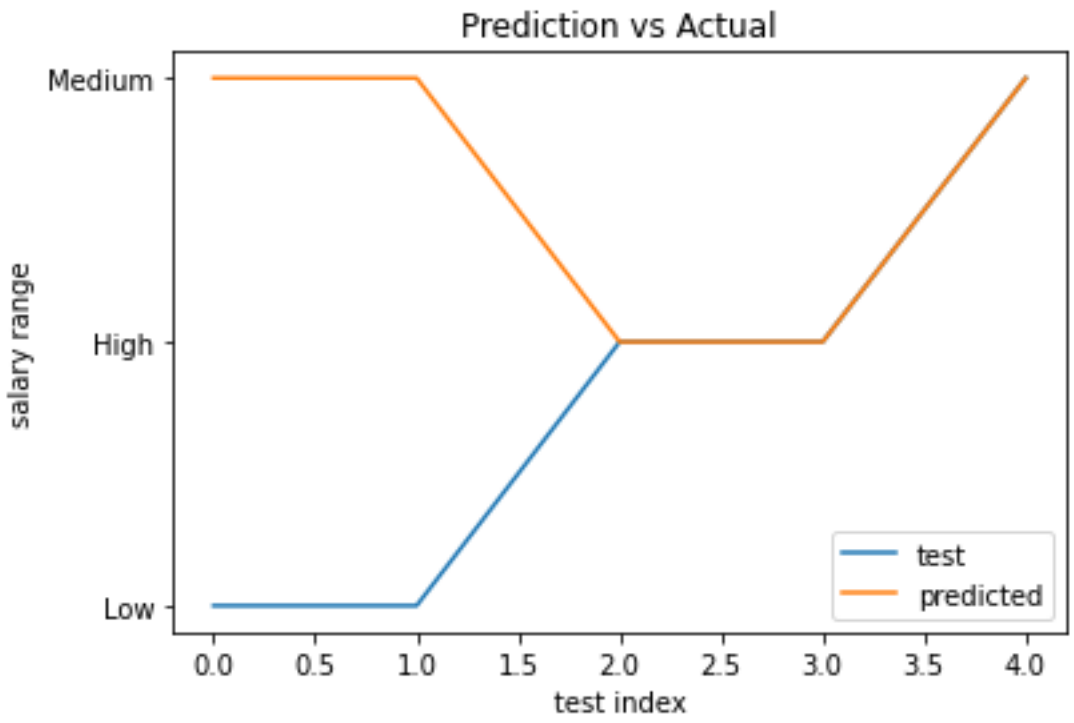
```
actual   predicted
Low        Medium
Low        Medium
High       High
High       High
Medium     Medium
```

It seems the classifier has problems predicting the low salary range, That's okay, we do not have a lot of training data, also our purpose is demonstrating not perfection.

Next we plot out the actual vs prediction results.

```
# plot actual vs prediction results

# plot ypred to ytest
plt.plot(y_test['range'].tolist(),label='test')
plt.plot(y_pred.tolist(),label='predicted')
plt.title('Prediction vs Actual')
plt.ylabel('salary range')
plt.xlabel('test index')
plt.legend()
plt.show()
```



Calculate Error and Accuracy

We now calculate error and accuracy. For these calculation we use the range_encoded column that has the prediction numeric values.

```
# calculate error and accuracy

# calculate mean square error mse
mse = mean_squared_error(y_test['range_coded'], y_pred)
print('mse =', mse)

# calculate root mean square error rmse
rmse = np.sqrt(mse)
print('rmse =', rmse)

# calculate mean absolute error mae
mae = mean_absolute_error(y_test['range_coded'], y_pred)
print('mae =', mae)

# calculate accuracy score
accuracy = classifier.score(X_test, y_test['range_coded'])
print("accuracy = ", accuracy )
```

```
mse = 0.4
rmse = 0.6324555320336759
mae = 0.4
accuracy =  0.6
```

Print Confusion Matrix

A confusion matrix count the number of times instances of class A are classified as class B. Each row in a confusion matrix represents an actual class, while each column represents a predicted class. The top is the predicted results and the left side the actual results. The diagonals are the true values: True High True Medium and True Low. Where every where else are the False values. False Low, False Medium and False High.

```
# print confusion matrix
from sklearn.metrics import confusion_matrix
cnf_matrix = confusion_matrix(y_test['range_coded'], y_pred)
print('Confusion matrix: [[TH FL FM][FH TL FM][FH FL TM]]')
print(cnf_matrix)
print( cnf_matrix)
```

```
Confusion matrix: [[TH FL FM][FH TL FM][FH FL TM]]
 [[2 0 0]
  [0 0 2]
  [0 0 1]]
```

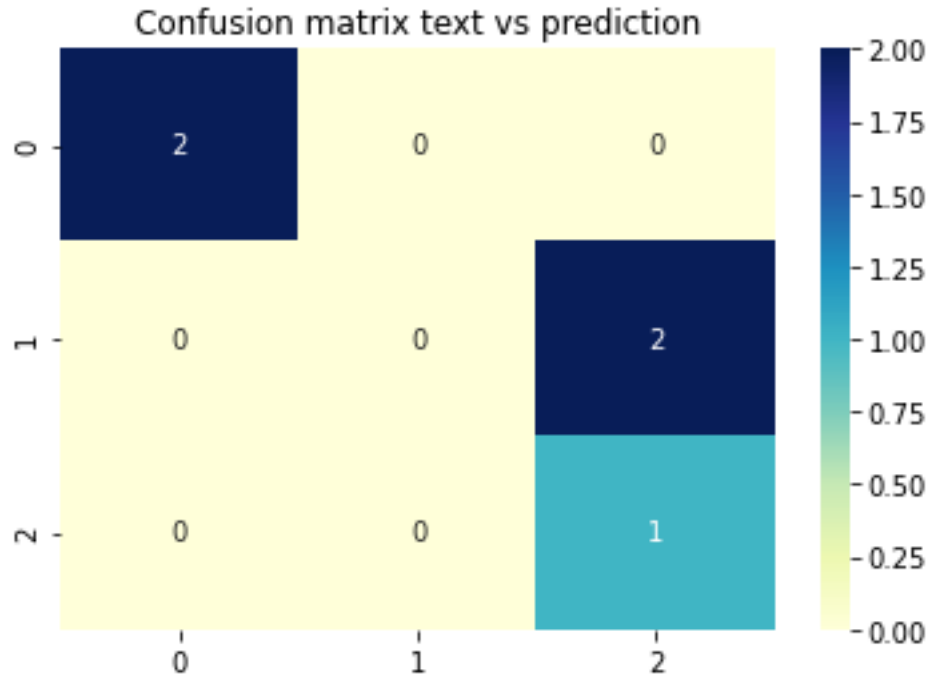
Where the entries are as follows:

	Predicted			
A		**H**	**L**	**M**
C	**H**	TH	FL	FM
T	**L**	FH	TL	FM
U	**M**	FH	FL	TM
A				
L				

Plot confusion matrix

We can put the confusing matrix id a seaborn heat map:

```
# Plot confusion matrix
sns.heatmap(cnf_matrix,annot = True,cmap="YlGnBu")
plt.title('Confusion matrix text vs prediction')
plt.show()
```



Calculating Precision, Recall and F1-Score

Precision talks about how precise/accurate your model is out of those predicted positive, how many of them are actual results.

Recall actually calculates how many of the Actual Positives our model capture through labeling it as Positive (True Positive). Recall is the model metric we use to select our best model when there is a high cost associated with False Negative results.

For instance, in fraud detection or sick patient detection. If a fraudulent transaction (Actual Positive) is predicted as non-fraudulent (Predicted Negative), the consequence can be very bad for the bank.

Similarly, in sick patient detection. If a sick patient (Actual Positive) goes through the test and predicted as not sick (Predicted Negative). The cost associated with False Negative will be extremely high if the sickness is contagious.

F1 is a function of Precision and Recall. F1 Score is needed when you want to seek a balance between Precision and Recall.

```
# calculate precision, recall and f1_score
from sklearn.metrics import precision_score, recall_score, f1_score
print('Precision:', precision_score(y_test['range_coded'].values,
y_pred,average='weighted'))
print('Recall:', recall_score(y_test['range_coded'].values, y_pred,average='weighted'))

# Compute F1 Score
from sklearn.metrics import f1_score
print('F1 Score:', f1_score(y_test['range_coded'].values, y_pred,average='weighted'))
```

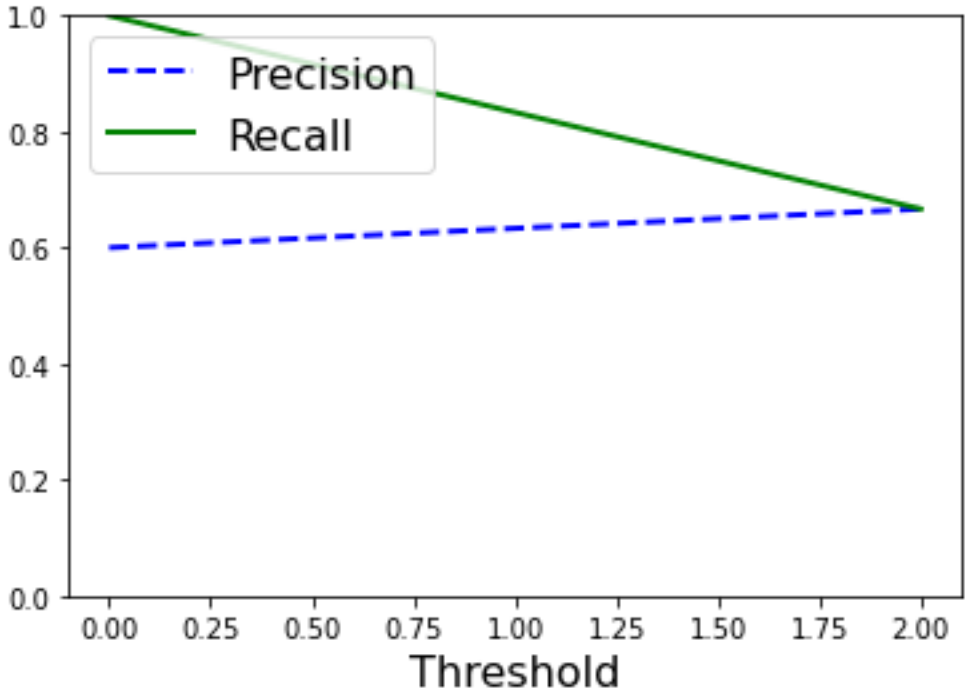
```
Precision: 0.4666666666666667
Recall: 0.6
F1 Score: 0.5
```

Plot Precision-Recall Curve

The precision-recall curve shows the tradeoff between precision and recall for different threshold. A high area under the curve represents both high recall and high precision, where high precision relates to a low false positive rate, and high recall relates to a low false negative rate. High scores for both show that the classifier is returning accurate results (high precision), as well as returning a majority of all positive results (high recall).

```
# Plot Precision-Recall Curve

from sklearn.metrics import precision_recall_curve
precisions, recalls, thresholds = precision_recall_curve(y_test['range_coded'],
y_pred,pos_label=2)
plt.plot(thresholds, precisions[:-1], "b--", label="Precision", linewidth=2)
plt.plot(thresholds, recalls[:-1], "g-", label="Recall", linewidth=2)
plt.xlabel("Threshold", fontsize=16)
plt.title("Precision Recall Curve")
plt.legend(loc="upper left", fontsize=16)
plt.ylim([0, 1])
plt.show()
```

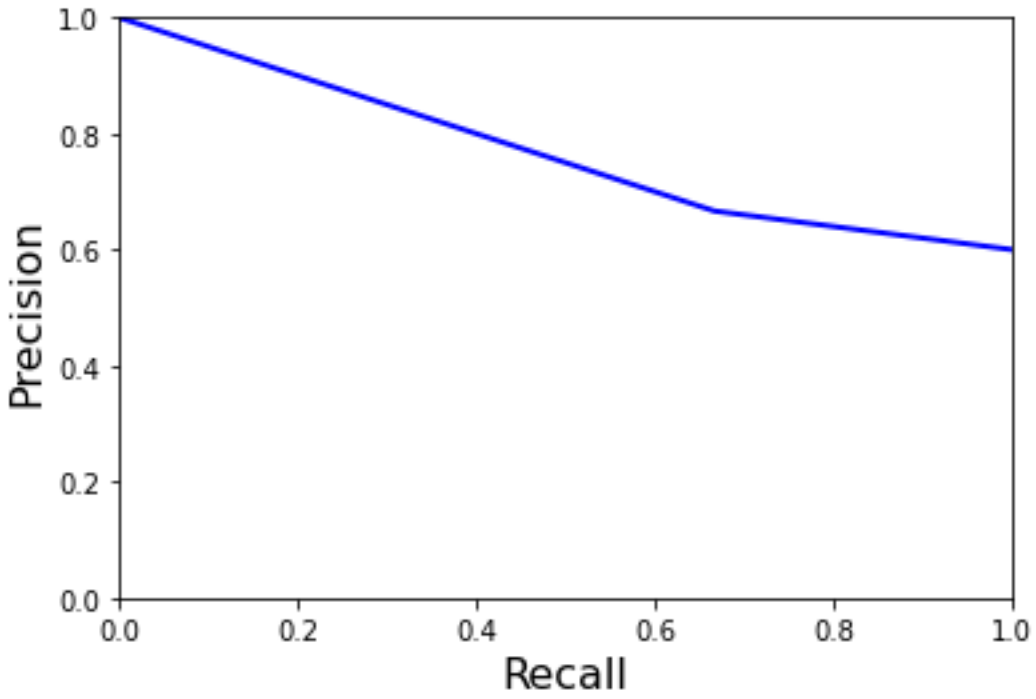


Plotting Recalls and Precisions

A system with high recall but low precision returns many results, but most of its predicted labels are incorrect when compared to the training labels. A system with high precision but low recall is just the opposite, returning very few results, but most of its predicted labels are correct when compared to the training labels.

An ideal system with high precision and high recall will return many results, with all results labeled correctly.

```
# plot recalls vs precisions,
plt.plot(recalls, precisions, "b-", linewidth=2)
plt.xlabel("Recall", fontsize=16)
plt.ylabel("Precision", fontsize=16)
plt.title("Recall vs Precision")
plt.axis([0, 1, 0, 1])
plt.show()
```



Plot ROC Curve

A **receiver operating characteristic** (ROC), or simply ROC curve, is a graphical plot which illustrates the performance of a binary classifier system as its discrimination threshold is varied. It is created by plotting the fraction of true positives out of the positives (TPR = true positive rate) vs. the fraction of false positives out of the negatives (FPR = false positive rate), at various threshold settings. TPR is also known as sensitivity, and FPR is one minus the specificity or true negative rate.

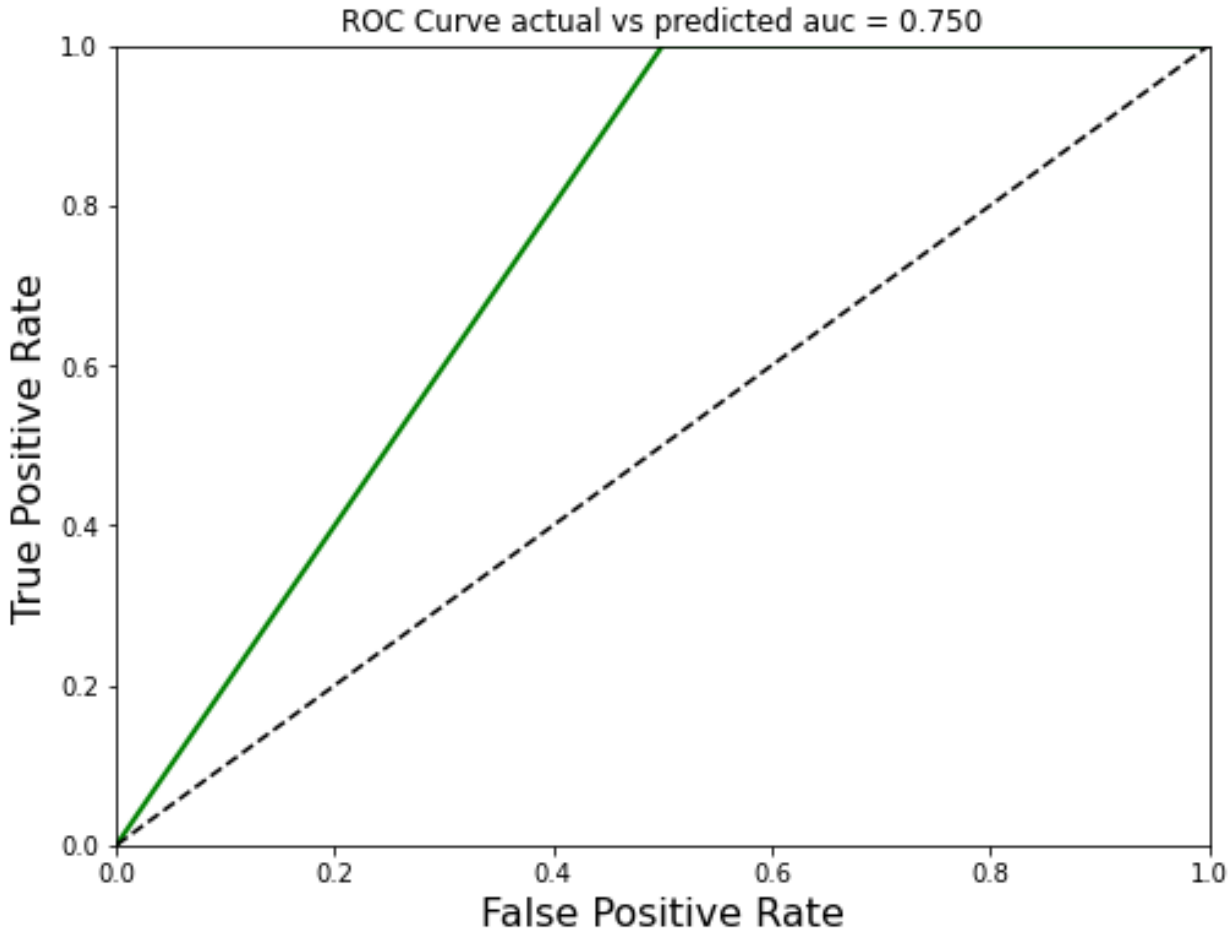
The **roc_auc_score** function computes the area under the receiver operating characteristic (ROC) curve, which is also denoted by AUC or AUROC. By computing the area under the roc curve, the curve information is summarized in one number.

```
# plot roc curve

from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score
from sklearn.metrics import auc
fpr, tpr, thresholds = roc_curve(y_test['range_coded'], y_pred,pos_label=2)
auc =  auc(fpr, tpr)
print("fpr: ", fpr)
print("tpr: ", tpr)
print("thresholds : ", thresholds)
print("auc : ", auc)
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, 'g-', linewidth=2, label='SGD')
plt.plot([0, 1], [0, 1], 'k--')
plt.axis([0, 1, 0, 1])
plt.xlabel('False Positive Rate', fontsize=16)
plt.ylabel('True Positive Rate', fontsize=16)
plt.title("ROC Curve actual vs predicted auc = %.3f" % (auc))
plt.show()

# Plotting ROC Curve from cross validation scores

from sklearn.metrics import roc_curve
from sklearn.metrics import auc
fpr, tpr, thresholds = roc_curve(y_train['range_coded'], y_scores,pos_label=2)
auc =  auc(fpr, tpr)
print("fpr: ", fpr)
print("tpr: ", tpr)
print("thresholds : ", thresholds)
print("auc : ", auc)
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, 'g-', linewidth=2, label='SGD')
plt.plot([0, 1], [0, 1], 'k--')
plt.axis([0, 1, 0, 1])
plt.xlabel('False Positive Rate', fontsize=16)
plt.ylabel('True Positive Rate', fontsize=16)
plt.title("ROC Curve actual vs predicted auc = %.3f" % (auc))
plt.show()
```



Cross Validation Value on Training Set

Cross validation divides the training set into smaller sub sets called folds (k – folds) The best subset is selected and used in the training data.

We use the sklearn the cross_val_score helper function on the estimator and the dataset to estimate the accuracy of a classifier by splitting the data, fitting a model and computing the score 5 consecutive times (with different splits each time):

```
# Cross Validation Value on Training Set

# use cross-validation and print the scores for each fold
from sklearn.model_selection import cross_val_score
import sklearn.metrics

#  print cross validation prediction scores for each fold
scores = cross_val_score(classifier, X_train, y_train['range_coded'], cv=3)
print('Score for each fold:', scores)
```

```
Score for each fold: [0.  0.5 0. ]
```

Cross Validation Predict on Training Set

We use the sklearn **cross_val_predict** function to generate cross-validated estimates for each input data point. The data is split according to the **cv** parameter in the **cross_val_predict** function. Each sample belongs to exactly one test set, and its prediction is computed with an estimator fitted on the corresponding training set. We use the training set on the classifier training data rather than the test set,

```
# cross validation prediction on Training Set

from sklearn.model_selection import cross_val_predict

y_train_pred = cross_val_predict(classifier, X_train, y_train['range_coded'], cv=3)
print("y train pred:" ,y_train_pred)

# print confusion matrix from cross_val_predict results
from sklearn.metrics import confusion_matrix
print('Confusion matrix: [[TN, FP],[FN,TP]', confusion_matrix(y_train['range_coded'],
y_train_pred))

# Obtain Precision and Recall
from sklearn.metrics import precision_score, recall_score
```

```
print('Precision:', precision_score(y_train['range_coded'].values,
y_train_pred,average='weighted'))
print('Recall:', recall_score(y_train['range_coded'].values,
y_train_pred,average='weighted'))


# Compute F1 Score
from sklearn.metrics import f1_score
print('F1 Score:', f1_score(y_train['range_coded'].values,
y_train_pred,average='weighted'))
```

```
Confusion matrix: [[TH FL FM][FH TL FM][FH FL TM]]
[[0 1 0]
 [0 0 1]
 [0 2 1]]
Precision: 0.3
Recall: 0.2
F1 Score: 0.24000000000000005
```

Test a Random Person

We pick a random person from our X Train set to print out the probability scores per class using the classifier function decision_function

```
# test a random person

# pick random person
random_person = np.random.randint(0, len(X_train))

# get probabilities
random_person_scores =
classifier.decision_function((X_train[random_person,0:].reshape(1,-1)))

# print scores and classes
print('Score per class:', random_person_scores)
print('Classes:',classifier.classes_)
```

```
Score per class: [[-62.11598648 -39.25580622   8.43517366]]
Classes: [0 1 2]
```

precision_recall_curve on Cross Validated Training Data

We first get the y predicted scores from the **cross_val_predict** function

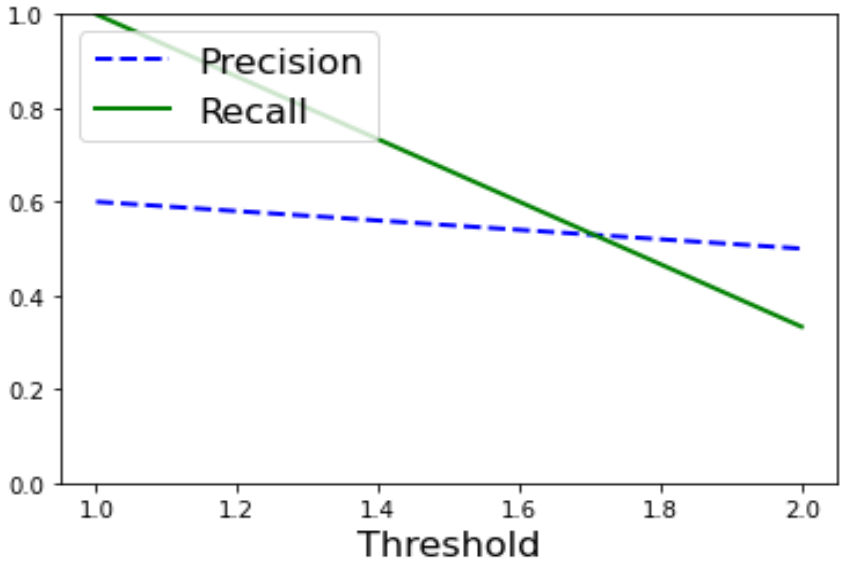
```
# precision_recall_curve   on Cross Validated Training Data

#  print cross validation prediction scores for each fold
y_scores = cross_val_predict(classifier, X_train,y_train['range_coded'], cv=3)
print('Score for each fold:', y_scores)
```

```
Score for each fold: [1 1 2 2 1]
```

We use the y scores on the precision_recall_curve.

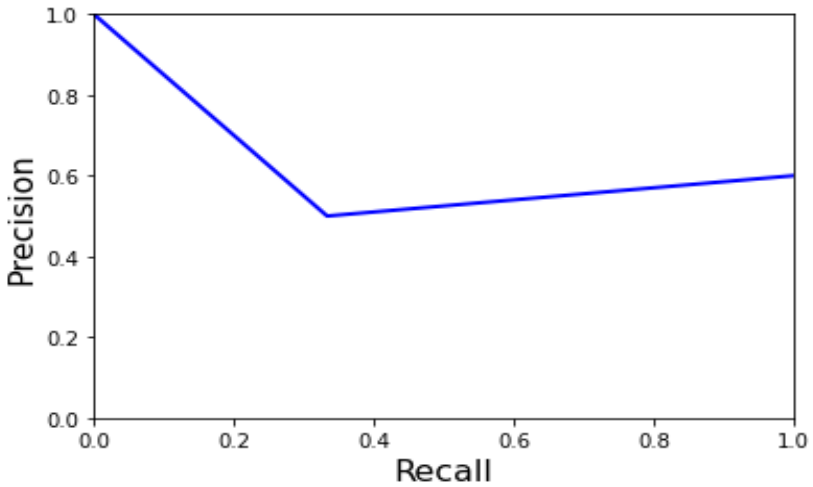
```
from sklearn.metrics import precision_recall_curve
precisions, recalls, thresholds = precision_recall_curve(y_train['range_coded'],
y_scores,pos_label=2)
plt.plot(thresholds, precisions[:-1], "b--", label="Precision", linewidth=2)
plt.plot(thresholds, recalls[:-1], "g-", label="Recall", linewidth=2)
plt.xlabel("Threshold", fontsize=16)
plt.title("Precision Recall Curve")
plt.legend(loc="upper left", fontsize=16)
plt.ylim([0, 1])
plt.show()
```



Plotting Recall vs Precision

We can plot precision vs recall from the results obtained from the **precision_recall_curve** function.

```
# plotting recall vs precision
plt.plot(recalls, precisions, "b-", linewidth=2)
plt.xlabel("Recall", fontsize=16)
plt.ylabel("Precision", fontsize=16)
plt.title("Recall vs Precision")
plt.axis([0, 1, 0, 1])
plt.show()
```

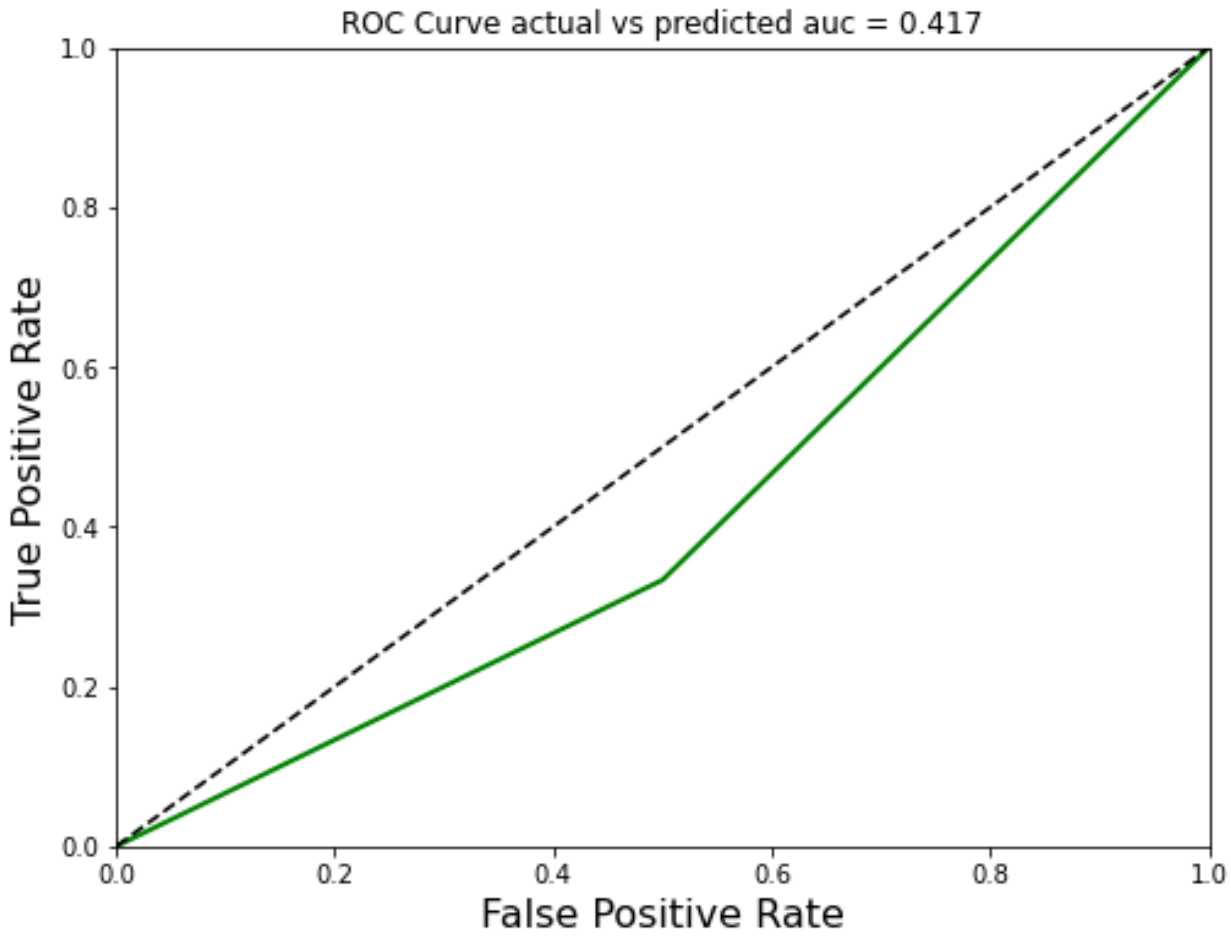


Plotting ROC Curve from cross validation scores

We can plot the roc curve using the y training actual values and the cross validation prediction probability results.

```
# Plotting ROC Curve from cross validation scores

from sklearn.metrics import roc_curve
from sklearn.metrics import auc
fpr, tpr, thresholds = roc_curve(y_train['range_coded'], y_scores,pos_label=2)
auc =  auc(fpr, tpr)
print("fpr: ", fpr)
print("tpr: ", tpr)
print("thresholds : ", thresholds)
print("auc : ", auc)
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, 'g-', linewidth=2, label='SGD')
plt.plot([0, 1], [0, 1], 'k--')
plt.axis([0, 1, 0, 1])
plt.xlabel('False Positive Rate', fontsize=16)
plt.ylabel('True Positive Rate', fontsize=16)
plt.title("ROC Curve actual vs predicted auc = %.3f" % (auc))
plt.show()
```



Summary

We have made a multiclass classifier, we have scaled the data, encoded the data and tested for errors accuracy. We have plotted the precision and recall and ROC curve's and calculated the area under the ROC curve AUC. We used cross validation and prediction to compare our test results to the training data and to choose the best training data

To do

Type in or copy paste in the above code and get it running. Add more data to the csv file . tweak the classifier max_iter=30, tol=1e-3 to increase the accuracy

```
classifier = SGDClassifier(random_state=42, max_iter=30, tol=1e-3)
```

Ignore all the warnings when you run the program, There errors are caused by insufficient test data.

Here is the complete program:

```
# example using scaling and encoding

import pandas as pd;
import matplotlib.pyplot as plt
import seaborn as sns

# read in salaries.csv file
df = pd.read_csv("salaries.csv")

# check data frame for nulls
print("Number Nulls: ",df.isnull().values.sum())

# check columns for null
print(df.isnull().sum())

# replace any null salaries with average of the country
df['salary'] = df['salary'].fillna(df.groupby('country')['salary'].transform('mean'))

# check data frame for nulls
print("Number Nulls: ",df.isnull().values.sum())

# check columns for null
print(df.isnull().sum())

# print data frame again
print(df)

# we need to drop name column
df = df.drop("name",axis='columns')
print(df)

# plotting salary distribution
salary_range_counts = df['range'].value_counts()
sns.barplot(x=salary_range_counts.index,y= salary_range_counts)
plt.title('Frequency Distribution of Salary Range')
plt.ylabel('Number of Occurrences')
plt.xlabel('Salary Range', fontsize=12)
```

```
plt.show()

# plotting salary distribution as a  pie chart
labels = df['salary'].astype('category').cat.categories.tolist()
counts = df['salary'].value_counts()
sizes = [counts[var_cat] for var_cat in labels]
fig1, ax1 = plt.subplots()
ax1.pie(sizes, labels=labels, autopct='%1.1f%%', shadow=True) #autopct is show the %
on plot
ax1.axis('equal')
plt.show()

# encoding the data

# replacing country labels with sequential number

# make map of country codes
country_map = {'country': {'Canada': 1, 'USA': 2, 'Europe': 3, 'India': 4,'China': 5}}

# look up countries automatically and assign a sequential number to each
labels = df['country'].astype('category').cat.categories.tolist()
country_map = {'country' : {k: v for k,v in zip(labels,list(range(1,len(labels)+1)))}}

# print country map
print(country_map)

#  replace the country labels with codes
df.replace(country_map, inplace=True)
print(df)

# replace Gender column with 1 and 0

 # make LabelEncoder
from sklearn.preprocessing import LabelEncoder

# gender to numbers
labelEncoder = LabelEncoder()
df['gender_code'] = labelEncoder.fit_transform(df['gender'])

df.drop('gender', axis='columns',inplace=True)
print(df)
```

```
# replace education column

# one hot encoding using get dummies
df2 = pd.get_dummies(df, columns=['education'], prefix = ['education'])

print(df2)

# using one hot encoder
from sklearn.preprocessing import OneHotEncoder

# make one hot encodes, ignore unknown column values
oneHotEncoder = OneHotEncoder(handle_unknown='ignore')

# do one hot encoding
encoded_education =
pd.DataFrame(oneHotEncoder.fit_transform(df[['education']]).toarray())

# get encoded column names
encoded_columns = oneHotEncoder.get_feature_names(['education'])

# assign encoded column names
encoded_education.columns=encoded_columns
print(encoded_education)

# add encoded columns to dataframe
df = df.join(encoded_education)

# print our education columns for verification
print(df[['education'] + df.columns[-3:].tolist()])

# drop education column
df.drop('education', axis='columns',inplace=True)

# encode range column
labelEncoder = LabelEncoder()
df['range_coded'] = labelEncoder.fit_transform(df['range'])


# plot pair plot with regression line
sns.pairplot(df,kind="reg")
plt.show()
```

```
# plot correlation heat map
# calculate the correlation matrix
corr = df.corr()
corr.style.background_gradient(cmap='coolwarm')

# plot the heatmap
sns.heatmap(corr, xticklabels=corr.columns,yticklabels=corr.columns,annot = True)
plt.show()

 # scaling and splitting the data

#set feature
X = df[['age','gender_code','country','education_College','education_High
School','education_University','salary']]

#target is salary range
y = df[['range','range_coded']]

from sklearn.preprocessing import StandardScaler

# scale X data
scaler = StandardScaler()
scaler.fit(X)
X_scaled = scaler.transform(X)
print("scaled x")
print(X_scaled)

# splitting the data

from sklearn.model_selection import train_test_split

# split data
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.5, random_state=42)
print("X train data")
print(X_train)

print("y train data")
print(y_train)

 print("X test data")
 print(X_test)
```

```
print("y test data")
print(y_test)

# classifying the data

from sklearn.linear_model import SGDClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error

import numpy as np

# make sgd clasifier
classifier = SGDClassifier(random_state=42, max_iter=30, tol=1e-3)

# fit the model
classifier.fit(X_train,y_train['range_coded'])

# predict data
y_pred = classifier.predict(X_test)

# print test and predicted scores
print("actual    predicted");
for a,p in zip(y_test['range'],y_pred):
    print(a,p)

# plot actual vs prediction results

# plot ypred to ytest
plt.plot(y_test['range'].tolist(),label='test')
plt.plot(y_pred.tolist(),label='predicted')
plt.title('Prediction vs Actual')
plt.ylabel('salary range')
plt.xlabel('test index')
plt.legend()
plt.show()

# calculate error and accuracy

# calculate mean square error mse
mse = mean_squared_error(y_test['range_coded'], y_pred)
print('mse =', mse)
```

```
# calculate root mean square error rmse
rmse = np.sqrt(mse)
print('rmse =', rmse)

# calculate mean absolute error mae
mae = mean_absolute_error(y_test['range_coded'], y_pred)
print('mae =', mae)

# calculate accuracy score
accuracy = classifier.score(X_test, y_test['range_coded'])
print("accuracy = ", accuracy )
# print confusion matrix
from sklearn.metrics import confusion_matrix
cnf_matrix = confusion_matrix(y_test['range_coded'], y_pred)
print('Confusion matrix: [[TH FL FM][FH TL FM][FH FL TM]]')
print(cnf_matrix)
print( cnf_matrix)

# Plot confusion matrix
sns.heatmap(cnf_matrix,annot = True,cmap="YlGnBu")
plt.title('Confusion matrix text vs prediction')
plt.show()
# calculate precision, recall and f1_score
from sklearn.metrics import precision_score, recall_score, f1_score
print('Precision:', precision_score(y_test['range_coded'].values,
y_pred,average='weighted'))
print('Recall:', recall_score(y_test['range_coded'].values, y_pred,average='weighted'))

# Compute F1 Score
from sklearn.metrics import f1_score
print('F1 Score:', f1_score(y_test['range_coded'].values, y_pred,average='weighted'))

# Plot Precision-Recall Curve

from sklearn.metrics import precision_recall_curve
precisions, recalls, thresholds = precision_recall_curve(y_test['range_coded'],
y_pred,pos_label=2)
plt.plot(thresholds, precisions[:-1], "b--", label="Precision", linewidth=2)
plt.plot(thresholds, recalls[:-1], "g-", label="Recall", linewidth=2)
plt.xlabel("Threshold", fontsize=16)
plt.title("Precision Recall Curve")
plt.legend(loc="upper left", fontsize=16)
```

```
plt.ylim([0, 1])
plt.show()

# plot recalls vs precisions,
plt.plot(recalls, precisions, "b-", linewidth=2)
plt.xlabel("Recall", fontsize=16)
plt.ylabel("Precision", fontsize=16)
plt.title("Recall vs Precision")
plt.axis([0, 1, 0, 1])
plt.show()

# plot roc curve

from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score
from sklearn.metrics import auc
fpr, tpr, thresholds = roc_curve(y_test['range_coded'], y_pred,pos_label=2)
auc =  auc(fpr, tpr)
print("fpr: ", fpr)
print("tpr: ", tpr)
print("thresholds : ", thresholds)
print("auc : ", auc)
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, 'g-', linewidth=2, label='SGD')
plt.plot([0, 1], [0, 1], 'k--')
plt.axis([0, 1, 0, 1])
plt.xlabel('False Positive Rate', fontsize=16)
plt.ylabel('True Positive Rate', fontsize=16)
plt.title("ROC Curve actual vs predicted auc = %.3f" % (auc))
plt.show()


# Cross Validation Value on Training Set

# use cross-validation and print the scores for each fold
from sklearn.model_selection import cross_val_score
import sklearn.metrics

#  print cross validation prediction scores for each fold
scores = cross_val_score(classifier, X_train, y_train['range_coded'], cv=3)
print('Score for each fold:', scores)
```

```
# cross validation prediction on Training Set

from sklearn.model_selection import cross_val_predict

y_train_pred = cross_val_predict(classifier, X_train, y_train['range_coded'], cv=3)
print("y train pred:" ,y_train_pred)

# print confusion matrix from cross_val_predict results
from sklearn.metrics import confusion_matrix
print('Confusion matrix: [[TN, FP],[FN,TP]', confusion_matrix(y_train['range_coded'],
y_train_pred))

# Obtain Precision and Recall
from sklearn.metrics import precision_score, recall_score

print('Precision:', precision_score(y_train['range_coded'].values,
y_train_pred,average='weighted'))
print('Recall:', recall_score(y_train['range_coded'].values,
y_train_pred,average='weighted'))


# Compute F1 Score
from sklearn.metrics import f1_score
print('F1 Score:', f1_score(y_train['range_coded'].values,
y_train_pred,average='weighted'))

# test a random person

# pick random person
random_person = np.random.randint(0, len(X_train))

# get probabilities
random_person_scores =
classifier.decision_function((X_train[random_person,0:].reshape(1,-1)))

# print scores and classes
print('Score per class:', random_person_scores)
print('Classes:',classifier.classes_)

# precision_recall_curve   on Cross Validated Training Data
```

```
#  print cross validation prediction scores for each fold
y_scores = cross_val_predict(classifier, X_train,y_train['range_coded'], cv=3)
print('Score for each fold:', y_scores)

from sklearn.metrics import precision_recall_curve
precisions, recalls, thresholds = precision_recall_curve(y_train['range_coded'],
y_scores,pos_label=2)
plt.plot(thresholds, precisions[:-1], "b--", label="Precision", linewidth=2)
plt.plot(thresholds, recalls[:-1], "g-", label="Recall", linewidth=2)
plt.xlabel("Threshold", fontsize=16)
plt.title("Precision Recall Curve")
plt.legend(loc="upper left", fontsize=16)
plt.ylim([0, 1])
plt.show()

# Plotting Recall vs Precision

plt.plot(recalls, precisions, "b-", linewidth=2)
plt.xlabel("Recall", fontsize=16)
plt.ylabel("Precision", fontsize=16)
plt.title("Recall vs Precision")
plt.axis([0, 1, 0, 1])
plt.show()

# Plotting ROC Curve from cross validation scores

from sklearn.metrics import roc_curve
from sklearn.metrics import auc
fpr, tpr, thresholds = roc_curve(y_train['range_coded'], y_scores,pos_label=2)
auc =  auc(fpr, tpr)
print("fpr: ", fpr)
print("tpr: ", tpr)
print("thresholds : ", thresholds)
print("auc : ", auc)
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, 'g-', linewidth=2, label='SGD')
plt.plot([0, 1], [0, 1], 'k--')
plt.axis([0, 1, 0, 1])
plt.xlabel('False Positive Rate', fontsize=16)
plt.ylabel('True Positive Rate', fontsize=16)
plt.title("ROC Curve actual vs predicted auc = %.3f" % (auc))
plt.show()
```

HOMEWORK

Simple weather prediction

Make a multiclass model to predict weather using a classifier of your choice. You will predict weather accordingly to temperature, atmospheric pressure, wind, humidity, precipitation, and cloudiness. For columns wind and Cloudily use labels to describe the condition. For the other columns use numeric values but do not worry about the units.

temperature	atmospheric_pressure	wind	humidity	precipitation	cloudiness	weather
40	45	Very	67	56	sparse	raining
45	45	None	78	90	heavy	snowing
30	66	slightly	67	67	few	cold
80	77	breezy	78	78	many	hot
45	88	moderate	86	99	light	cloudy
45	56	None	78	66	many	windy

Things to a accomplish

(1) Make a csv file similar to the table above, you will need 20 to 30 rows of data, put in some Nan's in columns of your choice

(2) Read in csv file into a data frame

(3) Check for NaN's, Fill in NaN's with average data dependent of some other column.

For example if precipitation is missing, take the average of the wind column category where the missing value is found.

(4) Encode the label data as numbers using an encoder of your choice. You may want to have the target column as category and as a separate numeric column

(5) set X feature and y target data

(6) Scale the data

(7) plot a pair plot and correlation heatmap

(8) identify some trends and pattern into your data using the pair-plot and correlation plot

(9) Split the data into training and test sets

(10) make and fit the classifier

(11) predict data
print test and predicted scores

(12) plot out the actual vs prediction results.

(13) Calculate mae, mse, rmse and accuracy

(14) print and maybe plot out confusion matrix

(15) Calculate precision , recall and F1-Score

(16) plot the prescion-recall curve for the test data

(17) plot precision vs recall for the test data

(18) Plot the roc curves and calculate auc for the test data

(19) print out the cross validation folds on your test data

(20) Do a cross validation prediction on your test data

(21) Pick a random weather sample row and do the test score on it

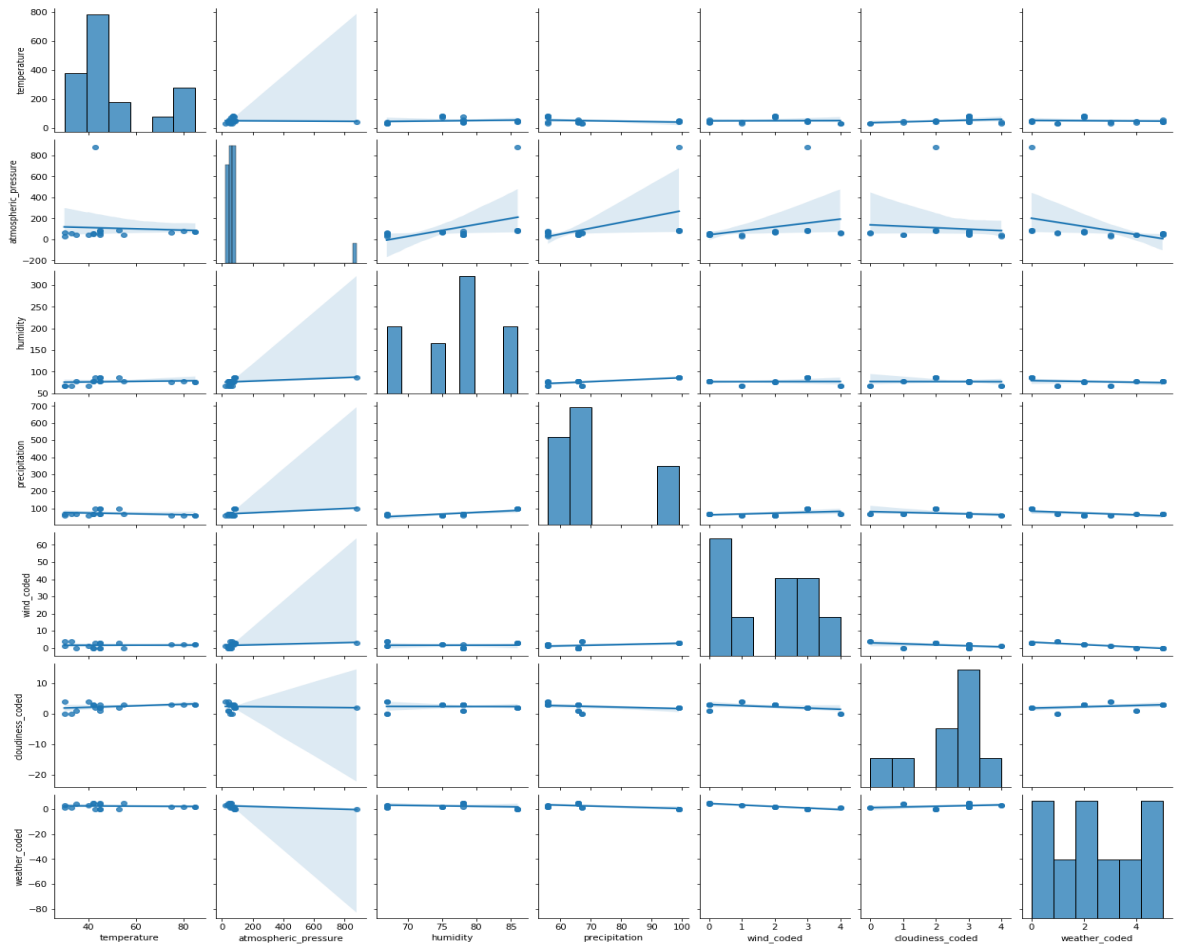
(22) Plot the recall precision curve for the train data

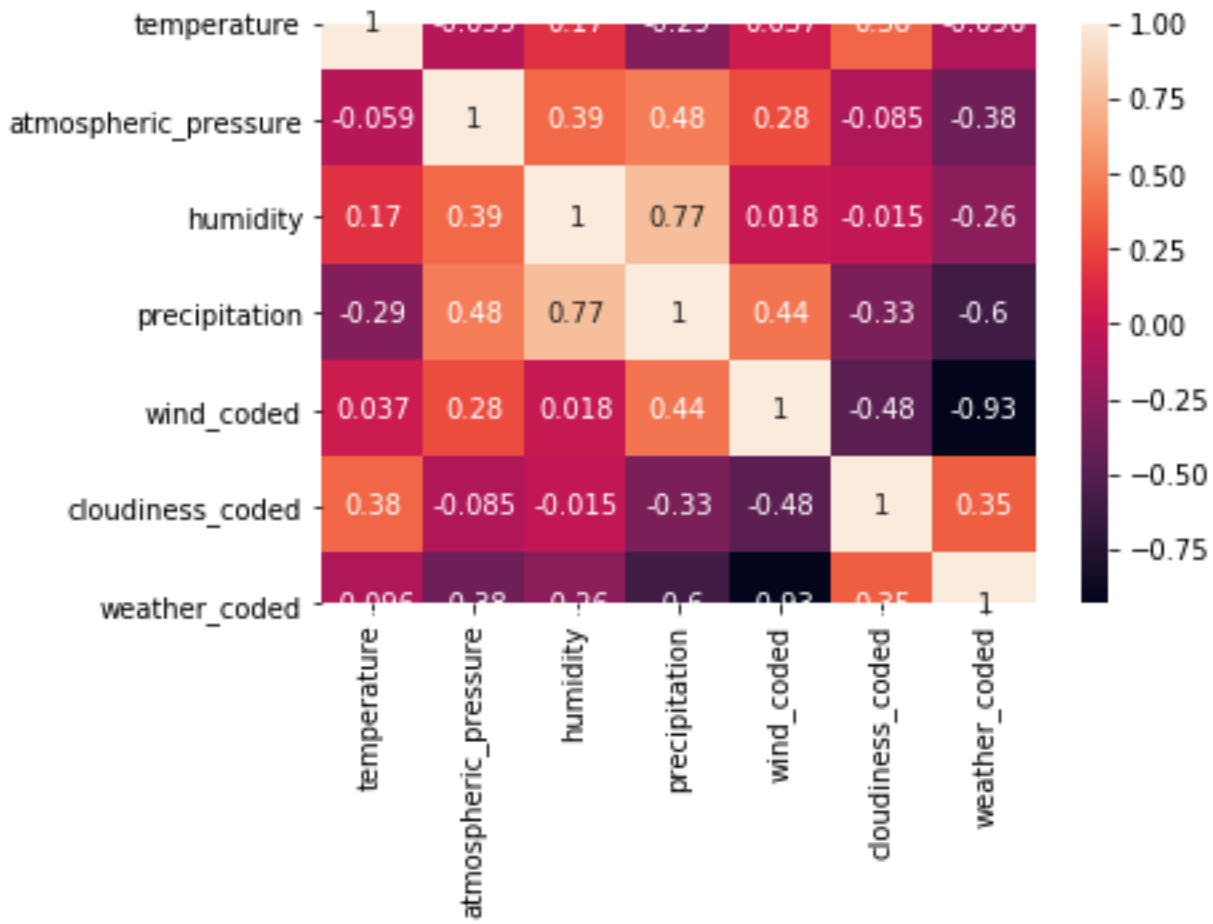
(23) plot precision vs recall for the train data

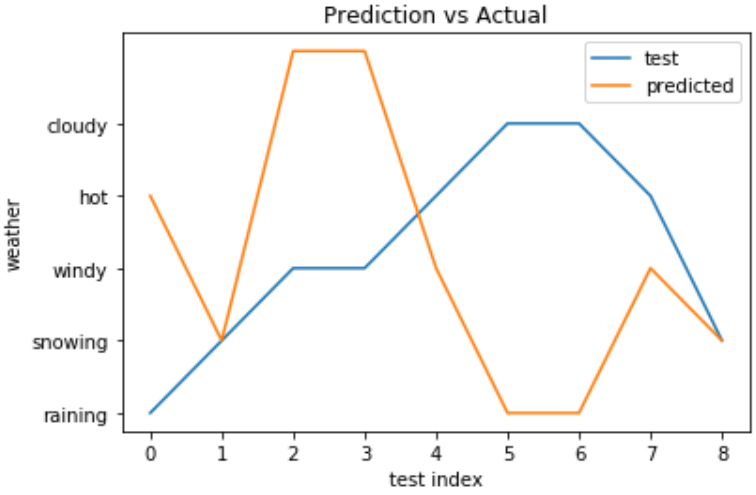
(24) Plot the roc curves and calculate auc for the train data

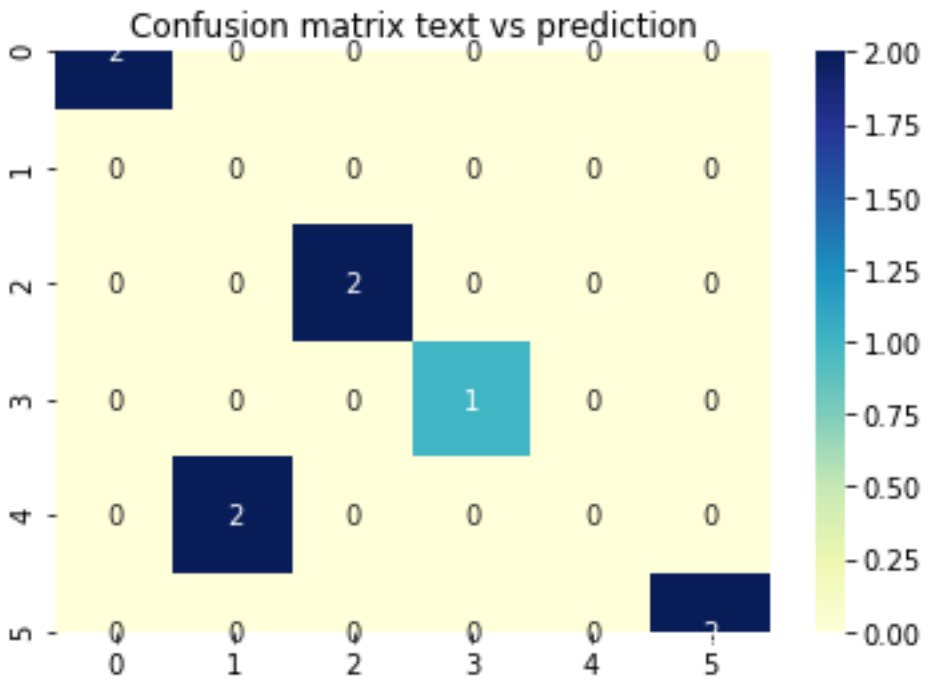
Ignore warnings, these warnings mean you do not have enough weather data.
Name your py file scaling_encoding_homework.py and your csv file weather.csv

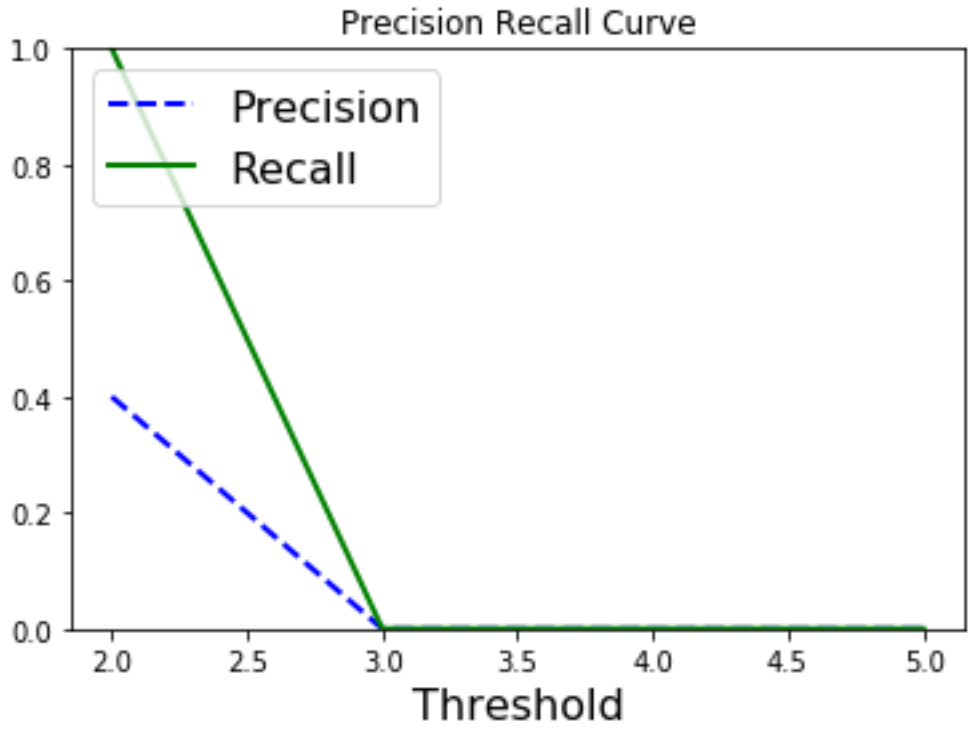
You should get something like this:

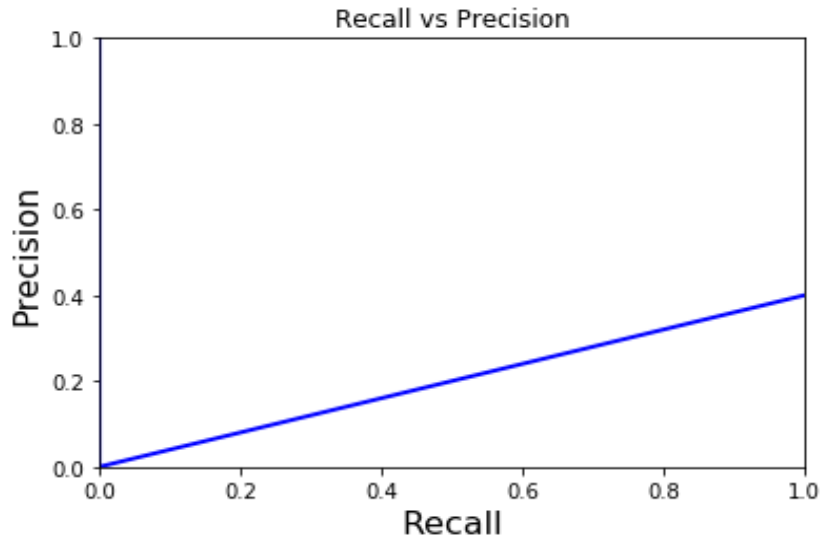


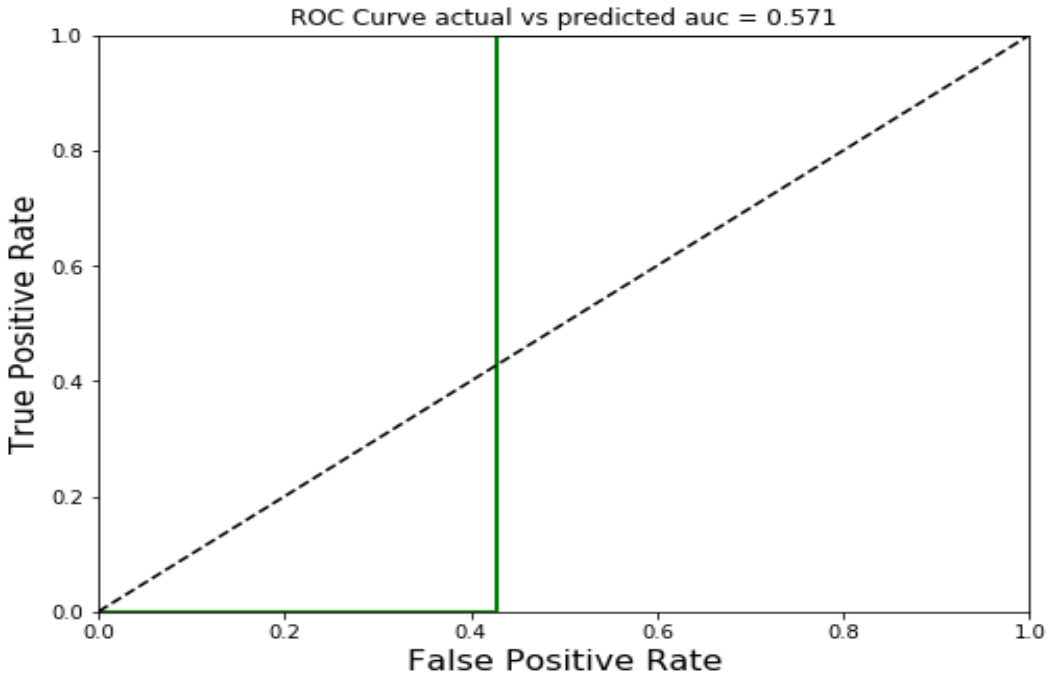


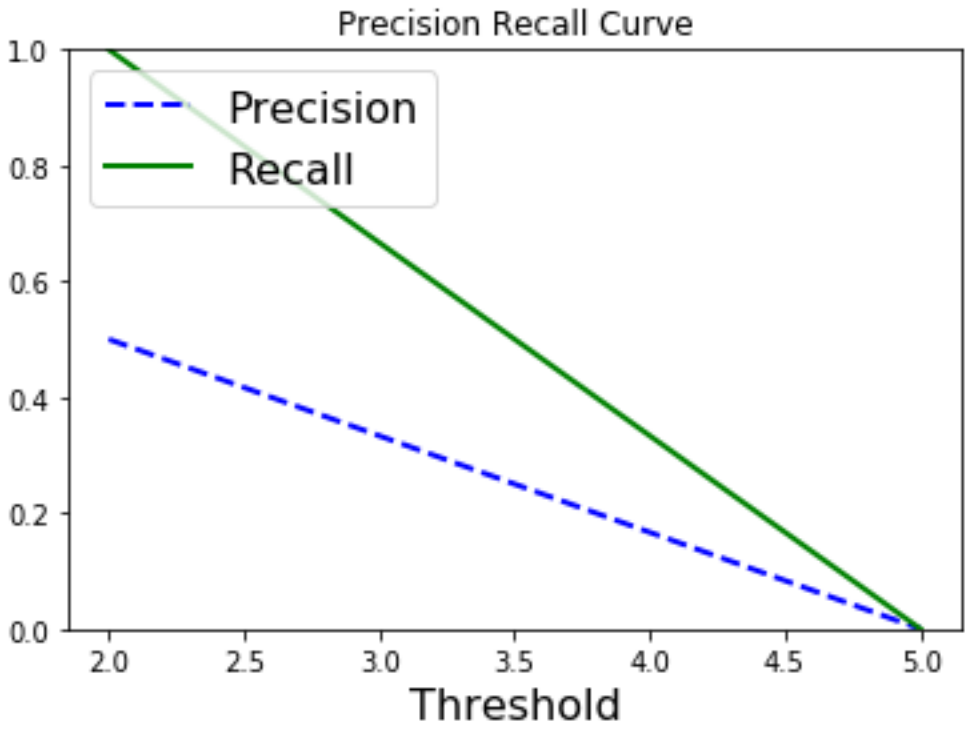


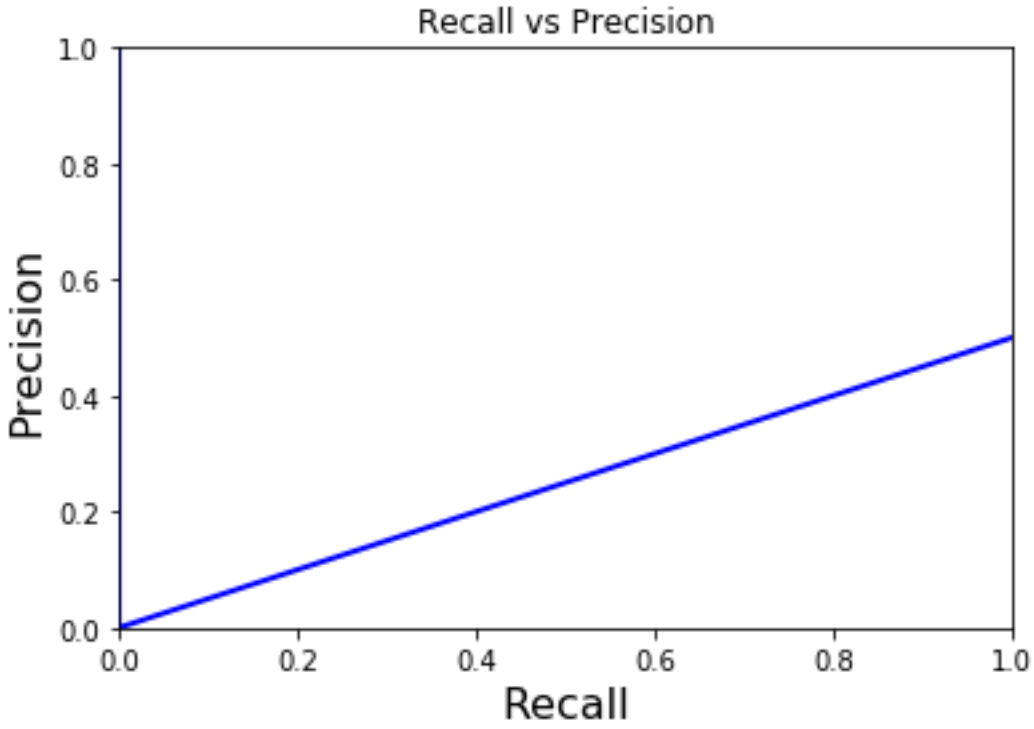


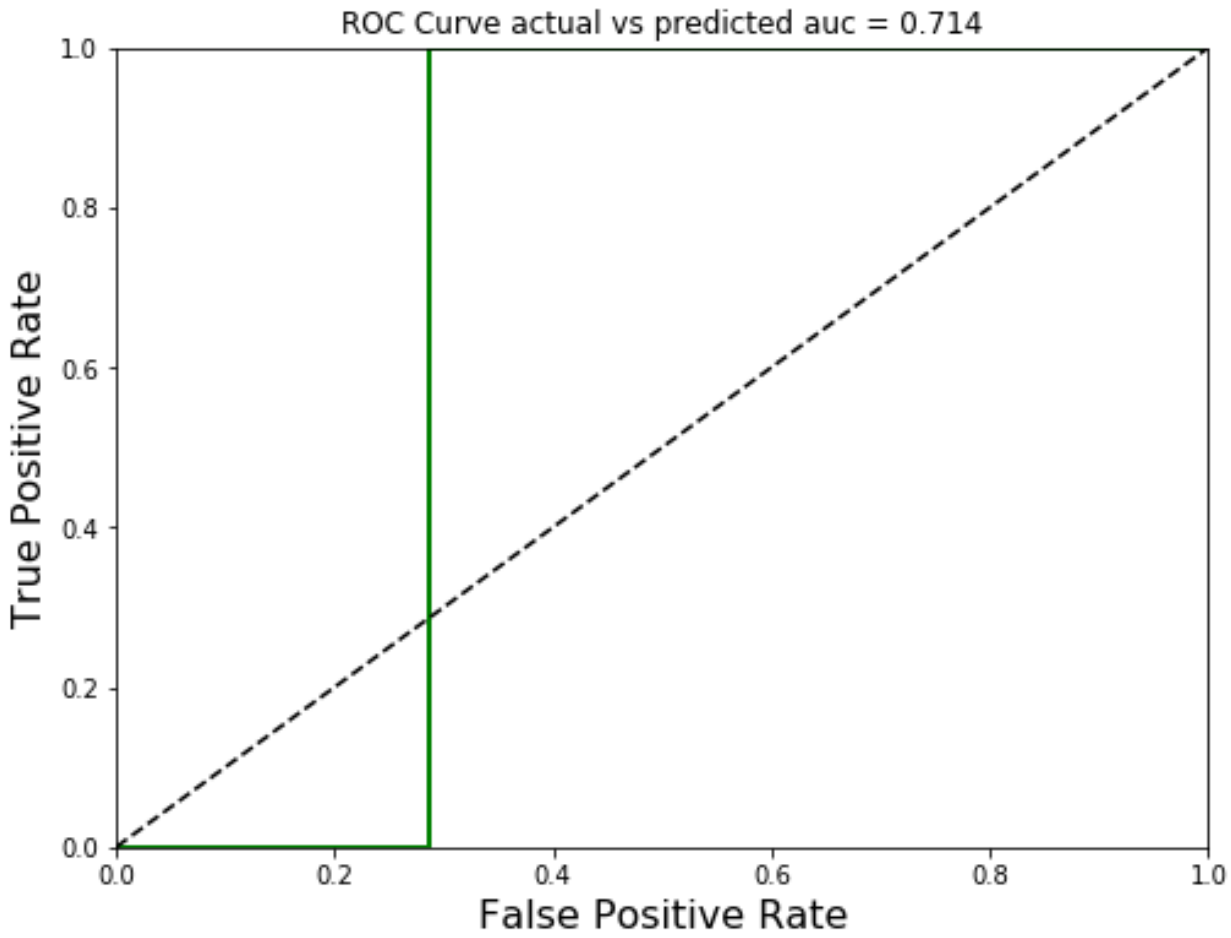











ROC Curve actual vs predicted auc = 0.714
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0

END