DS 6003 Assignment 1

Jiangxue Han (jh6rg)

Motivation

- The dataset contains 1000 camera models with 18 features including basic information and price.
- The model is designed to find the relationship between camera's price and camera's property.
- Price is the target feature. Both classification and regression techniques have been implemented. For the classification, a new variable is created indicating whether the price of corresponding camera is above the average price.
- download link: https://www.kaggle.com/crawford/1000-cameras-dataset/

Code snippet and explanation

- Create a new feature “isExpensive” indicating the whether the price is above average

In [55]:
```
from pyspark.sql.functions import *
df = df.withColumn('isExpensive',when(df.Price > 458,1).otherwise(0)).drop(df.Price)
df = df.drop(df.Model).dropna()
df
```

Out[55]:
```
DataFrame[date: bigint, Max_Resolution: double, Low_Resolution: double, Effective_pixels: double, Zoom_wide: double,
Zoom_tele: double, Normal_focus_range: double, Macro_focus_range: double, Storage_included: double, Weight: double, D
imensions: double, isExpensive: int]
```

- Correlation analysis: It seems the features don’t have a strong relationship with the target feature.

In [20]:
```
print("Pearson's r(Effective pixels,Max resolution) = {}".format(df.corr("Effective_pixels", "Max_Resolution")))
print("Pearson's r(Normal focus range,Min resolution) = {}".format(df.corr("Normal_focus_range", "Low_Resolution")))
print("Pearson's r(Dimensions,Weight) = {}".format(df.corr("Dimensions","Weight")))
```

```
Pearson's r(Effective pixels,Max resolution) = 0.9538458570187706
Pearson's r(Normal focus range,Min resolution) = -0.12543611622804537
Pearson's r(Dimensions,Weight) = 0.6778848089757021
```

In [63]:
```
print("Pearson's r(Date,Price) = {}".format(df.corr("date", "Price")))
print("Pearson's r(Max Resolution,Price) = {}".format(df.corr("Max_Resolution", "Price")))
print("Pearson's r(Low Resolution,Price) = {}".format(df.corr("Low_Resolution", "Price")))
print("Pearson's r(Zoom wide,Price) = {}".format(df.corr("Zoom_wide", "Price")))
print("Pearson's r(Normal focus range,Price) = {}".format(df.corr("Normal_focus_range", "Price")))
print("Pearson's r(Macro focus range,Price) = {}".format(df.corr("Macro_focus_range", "Price")))
print("Pearson's r(Effective pixels,Price) = {}".format(df.corr("Effective_pixels", "Price")))
print("Pearson's r(Dimension,Price) = {}".format(df.corr("Dimensions", "Price")))
```

```
Pearson's r(Date,Price) = -0.022522217044592785
Pearson's r(Max Resolution,Price) = 0.18420092652935718
Pearson's r(Low Resolution,Price) = 0.15420406063216538
Pearson's r(Zoom wide,Price) = -0.4591033999208599
Pearson's r(Normal focus range,Price) = -0.27385393615399717
Pearson's r(Macro focus range,Price) = -0.12757671236241522
Pearson's r(Effective pixels,Price) = 0.190284007565066
Pearson's r(Dimension,Price) = 0.2642561651698723
```

- Build Linear Regression model

7]:
```
df = df.drop(df.Low_Resolution).drop(df.Effective_pixels)
```

8]:
```
stages = []
df = df.withColumnRenamed("Price", "label")
assemblerInputs = df.columns[:-1]
assembler = VectorAssembler(inputCols=assemblerInputs, outputCol="features")
stages += [assembler]
```

9]:
```
pipeline = Pipeline(stages = stages)
pipelineModel = pipeline.fit(df)
df3 = pipelineModel.transform(df)
selectedCols = ['label', 'features']
df3 = df3.select(selectedCols)
```

0]:
```
# create train/test sets
seed = 42
(testDF, trainingDF) = df3.randomSplit((0.20, 0.80), seed=seed)
print ('training set N = {}, test set N = {}'.format(trainingDF.count(),testDF.count()))
```

```
training set N = 843, test set N = 193
```

1]:
```
lr = LinearRegression()
lrModel = lr.fit(trainingDF)
```

2]:
```
predictionsAndLabelsDF = lrModel.transform(testDF)
```

- Build Logistic Regression model

In [152]:
```
df.groupBy().avg('label').collect()
```

Out[152]:
```
[Row(avg(label)=457.9218146718147)]
```

In [154]:
```
# Create a new feature "isExpensive" indicating whether Price(label) is above average
df = df.withColumn('isExpensive',when(df.label > 458,1).otherwise(0)).drop(df.label)
df
```

Out[154]:
```
DataFrame[date: bigint, Max_Resolution: double, Zoom_wide: double, Zoom_tele: double, Normal_focus_range: double, Mac
ro_focus_range: double, Storage_included: double, Weight: double, Dimensions: double, isExpensive: int]
```

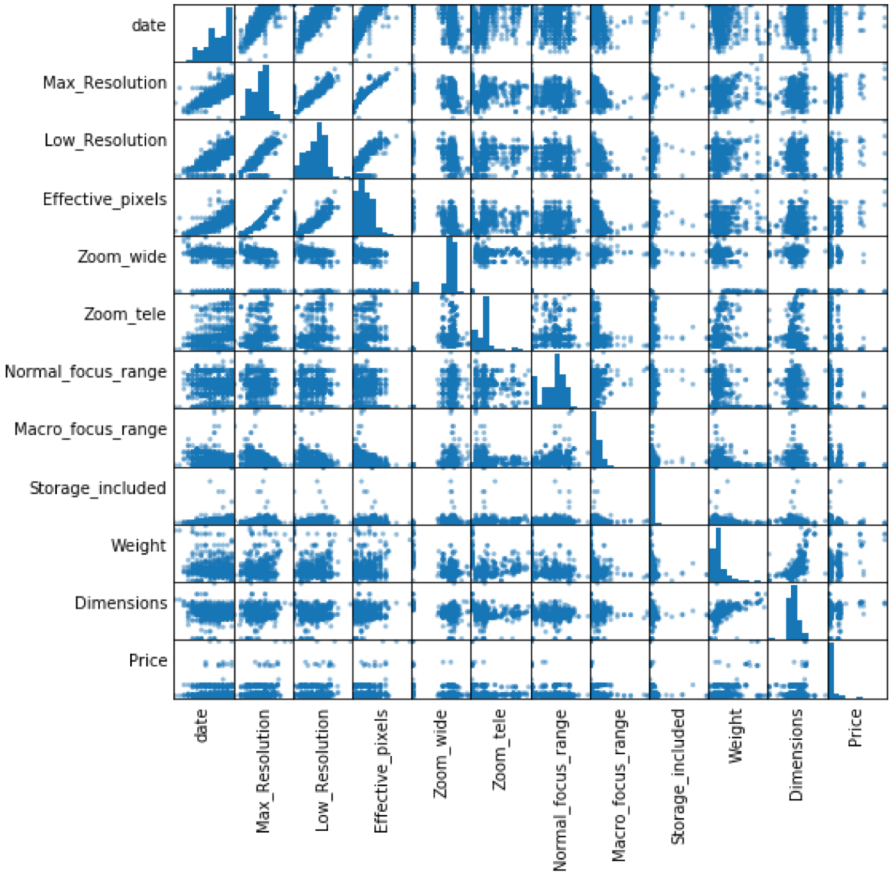
In [161]:
```
stages = []
label_stringIdx = StringIndexer(inputCol = 'isExpensive', outputCol = 'label')
stages += [label_stringIdx]
assemblerInputs = df.columns[:-3]
assembler = VectorAssembler(inputCols=assemblerInputs, outputCol="features")
stages += [assembler]
```

In [168]:
```
pipeline = Pipeline(stages = stages)
pipelineModel = pipeline.fit(df)
df3 = pipelineModel.transform(df)
selectedCols = ['label', 'features']
df3 = df3.select(selectedCols)
```

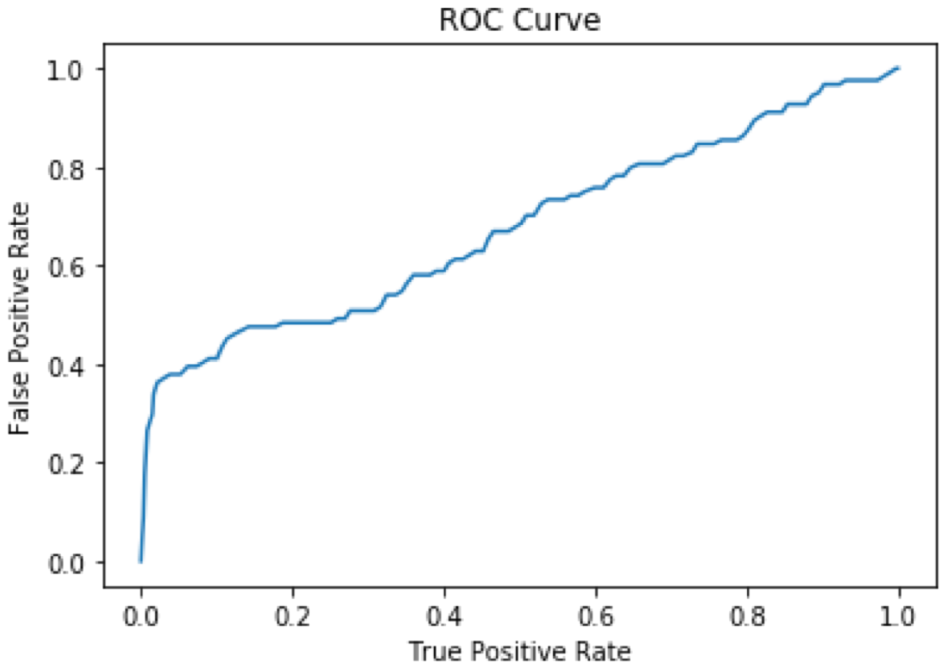
In [169]:
```
trainDF, testDF = df3.randomSplit([0.7, 0.3], seed = 2019)
print ('training set N = {}, test set N = {}'.format(train_df.count(),test_df.count()))
```

Visualizations

Correlation map



ROC Curve



AUC: 0.64