Preprocessing Data using SKlearn

Dealing with categorical features

- Scikit-learn will not accept categorical features by default
- Need to encode categorical features numerically
- Convert to ‘dummy variables’
  - 0: Observation was NOT that category
  - 1: Observation was that category

Dealing with categorical features in Python

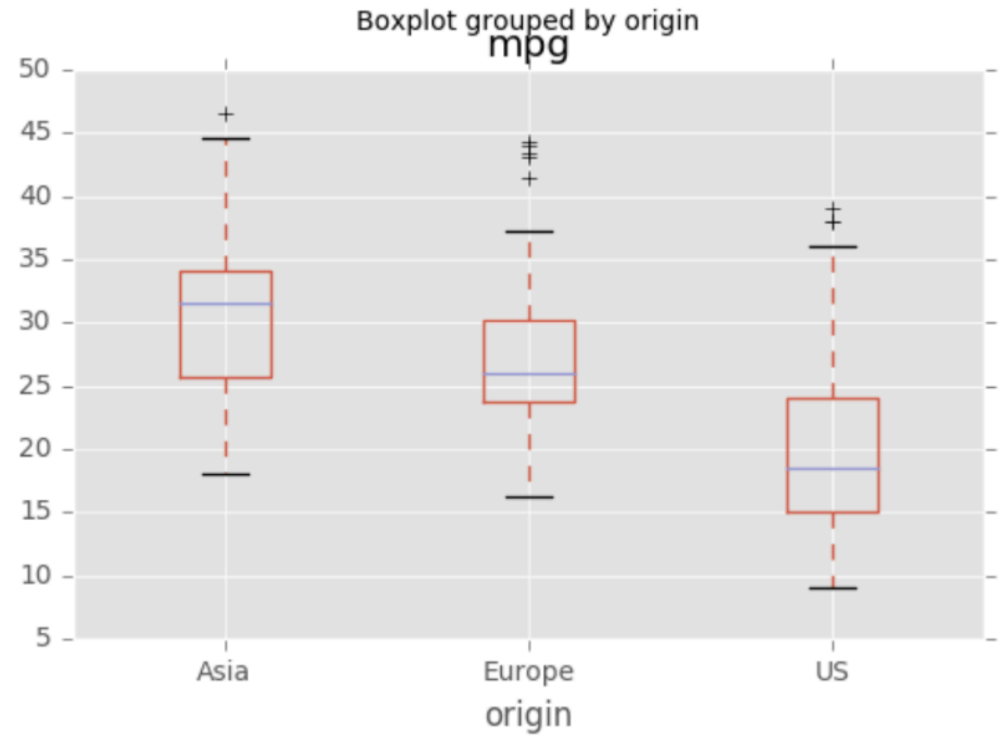
- scikit-learn: OneHotEncoder()
- pandas: get_dummies()

Automobile dataset

- mpg: Target Variable
- Origin: Categorical Feature

	mpg	displ	hp	weight	accel	origin	size
0	18.0	250.0	88	3139	14.5	US	15.0
1	9.0	304.0	193	4732	18.5	US	20.0
2	36.1	91.0	60	1800	16.4	Asia	10.0
3	18.5	250.0	98	3525	19.0	US	15.0
4	34.3	97.0	78	2188	15.8	Europe	10.0

EDA w/ categorical feature



Encoding dummy variables

```
import pandas as pd
df = pd.read_csv('auto.csv')
df_origin = pd.get_dummies(df)
print(df_origin.head())
```

```
    mpg  displ   hp  weight  accel  size  origin_Asia  origin_Europe  \\
0  18.0  250.0   88    3139   14.5  15.0            0              0
1   9.0  304.0  193    4732   18.5  20.0            0              0
2  36.1   91.0   60    1800   16.4  10.0            1              0
3  18.5  250.0   98    3525   19.0  15.0            0              0
4  34.3   97.0   78    2188   15.8  10.0            0              1
   origin_US
0          1
1          1
2          0
3          1
4          0
```

pandas get_dummies method turns the categorical column into a one hot representation.

Encoding dummy variables

```
df_origin = df_origin.drop('origin_Asia', axis=1)
print(df_origin.head())
```

```
    mpg  displ   hp  weight  accel  size  origin_Europe  origin_US
0  18.0  250.0   88    3139   14.5  15.0              0          1
1   9.0  304.0  193    4732   18.5  20.0              0          1
2  36.1   91.0   60    1800   16.4  10.0              0          0
3  18.5  250.0   98    3525   19.0  15.0              0          1
4  34.3   97.0   78    2188   15.8  10.0              1          0
```

We can drop the origin_Asia column as the data is present in the other two columns and if 1 value is not present in the other two columns it should be present in the origin_Asia column.

# Create dummy variables with drop_first=True: df_region
df_region = pd.get_dummies(df, drop_first = True)

drop_first removes the extra column which we have defined earlier.

Handling missing data

PIMA Indians dataset

```
df = pd.read_csv('diabetes.csv')
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
pregnancies    768 non-null int64
glucose        768 non-null int64
diastolic      768 non-null int64
triceps        768 non-null int64
insulin        768 non-null int64
bmi            768 non-null float64
dpf            768 non-null float64
age            768 non-null int64
diabetes       768 non-null int64
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
None
```

According to the info method there are no missing values, but missing values can be encoded in a variety of different ways. - 0, ?, -1

PIMA Indians dataset

```
print(df.head())
```

```
   pregnancies  glucose  diastolic  triceps  insulin   bmi    dpf  age  \
0            6      148         72       35        0  33.6  0.627   50
1            1       85         66       29        0  26.6  0.351   31
2            8      183         64        0        0  23.3  0.672   32
3            1       89         66       23       94  28.1  0.167   21
4            0      137         40       35      168  43.1  2.288   33
   diabetes
0         1
1         0
2         1
3         0
4         1
```

Dropping missing data

```
df.insulin.replace(0, np.nan, inplace=True)
df.triceps.replace(0, np.nan, inplace=True)
df.bmi.replace(0, np.nan, inplace=True)
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
pregnancies    768 non-null int64
glucose        768 non-null int64
diastolic      768 non-null int64
triceps        541 non-null float64
insulin        394 non-null float64
bmi            757 non-null float64
dpf            768 non-null float64
age            768 non-null int64
diabetes       768 non-null int64
dtypes: float64(4), int64(5)
memory usage: 54.1 KB
```

Dropping missing data

```
df = df.dropna()
df.shape
```

```
(393, 9)
```

We lost around 50% of our data, when we removed the missing values, as this is a huge number, we will think of other ways to deal with this.

Technique 1: Imputing Missing Data: Here imputing mean

Imputing missing data

- Making an educated guess about the missing values
- Example: Using the mean of the non-missing entries

```
from sklearn.preprocessing import Imputer
imp = Imputer(missing_values='NaN', strategy='mean', axis=0)
imp.fit(X)
X = imp.transform(X)
```

Here, by making use of the imputer we will fill the missing the missing values with the mean. axis = 0, we will take the mean of the column and then fill in the value along the column.

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import Imputer
imp = Imputer(missing_values = 'NaN', strategy = 'mean', axis = 0)
logreg = LogisticRegression
# Next we will define the steps to be taken in the pipeline
steps = [ ("Imputation", imp), ("Logistic Regression", logreg)
]
# Here first item in the tuple refers to the step name and the second parameter
refers to the object name.
pipeline = Pipeline(steps)
```

Imputing within a pipeline

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import Imputer
imp = Imputer(missing_values='NaN', strategy='mean', axis=0)
logreg = LogisticRegression()
steps = [('imputation', imp),
             ('logistic_regression', logreg)]
pipeline = Pipeline(steps)
X_train, X_test, y_train, y_test = train_test_split(X, y,
    test_size=0.3, random_state=42)
```

```
pipeline.fit(X_train, y_train)
y_pred = pipeline.predict(X_test)
pipeline.score(X_test, y_test)
```

```
0.75324675324675328
```

Note: In the pipeline the last step must be transformer(We can transform data on it)/classifier.

```
# Import necessary modules
from sklearn.preprocessing import Imputer
from sklearn.pipeline import Pipeline
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split

# Setup the pipeline steps: steps
steps = [('imputation', Imputer(missing_values='NaN', strategy='most_frequent', 
axis=0)),
        ('SVM', SVC())]

# Create the pipeline: pipeline
pipeline = Pipeline(steps)

# Create training and test sets
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state = 42)

# Fit the pipeline to the train set
pipeline.fit(X_train, y_train)

# Predict the labels of the test set
y_pred = pipeline.predict(X_test)

# Compute metrics
print(classification_report(y_test, y_pred))
```

Centering and scaling

- **Range varies**

Why scale your data?

```
print(df.describe())
```

```
     fixed acidity  free sulfur dioxide  total sulfur dioxide      density  \\
count  1599.000000          1599.000000           1599.000000  1599.000000
mean      8.319637            15.874922             46.467792     0.996747
std       1.741096            10.460157             32.895324     0.001887
min       4.600000             1.000000              6.000000     0.990070
25%       7.100000             7.000000             22.000000     0.995600
50%       7.900000            14.000000             38.000000     0.996750
75%       9.200000            21.000000             62.000000     0.997835
max      15.900000            72.000000            289.000000     1.003690
                pH    sulphates      alcohol      quality
count  1599.000000  1599.000000  1599.000000  1599.000000
mean      3.311113     0.658149    10.422983     0.465291
std       0.154386     0.169507     1.065668     0.498950
min       2.740000     0.330000     8.400000     0.000000
25%       3.210000     0.550000     9.500000     0.000000
50%       3.310000     0.620000    10.200000     0.000000
75%       3.400000     0.730000    11.100000     1.000000
max       4.010000     2.000000    14.900000     1.000000
```

Why scale your data?

- Many models use some form of distance to inform them
- Features on larger scales can unduly influence the model
- Example: k-NN uses distance explicitly when making predictions
- We want features to be on a similar scale
- Normalizing (or scaling and centering)

Ways to normalize your data

- Standardization: Subtract the mean and divide by variance
- All features are centered around zero and have variance one
- Can also subtract the minimum and divide by the range
- Minimum zero and maximum one
- Can also normalize so the data ranges from -1 to +1
- See scikit-learn docs for further details

Scaling in scikit-learn

```
from sklearn.preprocessing import scale
X_scaled = scale(X)
```

```
np.mean(X), np.std(X)
```

```
(8.13421922452, 16.7265339794)
```

```
np.mean(X_scaled), np.std(X_scaled)
```

```
(2.54662653149e-15, 1.0)
```

Scaling in a pipeline

```
from sklearn.preprocessing import StandardScaler
steps = [('scaler', StandardScaler()),
             ('knn', KNeighborsClassifier())]
pipeline = Pipeline(steps)
X_train, X_test, y_train, y_test = train_test_split(X, y,
    test_size=0.2, random_state=21)
knn_scaled = pipeline.fit(X_train, y_train)
y_pred = pipeline.predict(X_test)
accuracy_score(y_test, y_pred)
```

```
0.956
```

```
knn_unscaled = KNeighborsClassifier().fit(X_train, y_train)
knn_unscaled.score(X_test, y_test)
```

```
0.928
```

CV and scaling in a pipeline

```
steps = [('scaler', StandardScaler()),
        (('knn', KNeighborsClassifier())]
pipeline = Pipeline(steps)
parameters = {knn__n_neighbors: np.arange(1, 50)}
X_train, X_test, y_train, y_test = train_test_split(X, y,
    test_size=0.2, random_state=21)
cv = GridSearchCV(pipeline, param_grid=parameters)
cv.fit(X_train, y_train)
y_pred = cv.predict(X_test)
```

Scaling and CV in a pipeline

```
print(cv.best_params_)
```

```
{'knn__n_neighbors': 41}
```

```
print(cv.score(X_test, y_test))
```

```
0.956
```

```
print(classification_report(y_test, y_pred))
```

```
             precision    recall  f1-score   support
          0       0.97      0.90      0.93        39
          1       0.95      0.99      0.97        75
avg / total       0.96      0.96      0.96       114
```