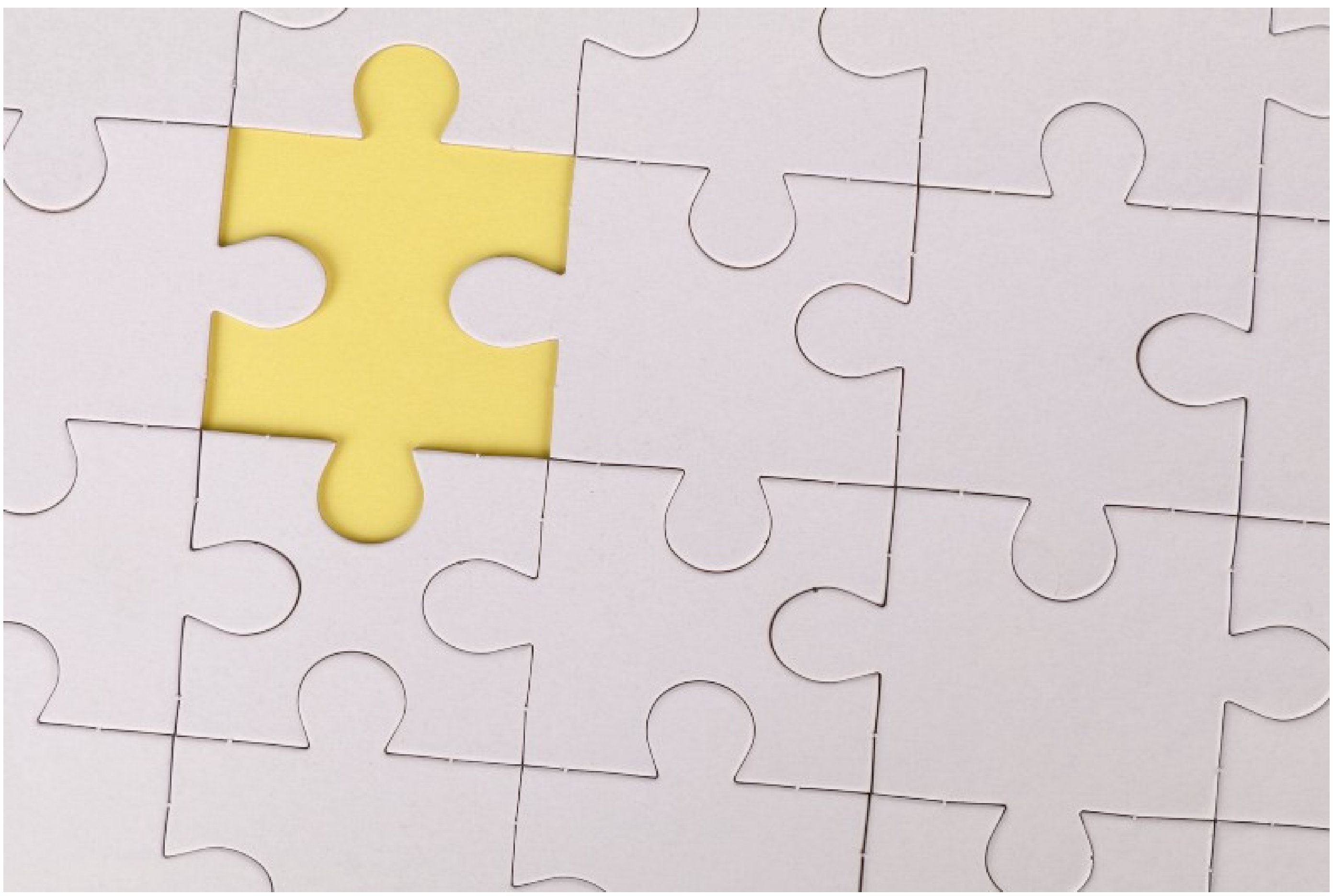
Published in Towards Data Science

Sadrach Pierre, Ph.D. Follow

Aug 20, 2020 · 6 min read ★

Predicting Missing Values with Python

Building Models for Data Imputation



Source

For data scientists, handling missing data is an important part of the data cleaning and

Let’s get started!

For our purposes, we will be working with the wines dataset which can be found here.

To start, let’s read the data into a Pandas data frame:

```
import pandas as pd
df = pd.read_csv("winemag-data-130k-v2.csv")
```

Next, let’s print the first five rows of data:

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

```
    country                                        description  \
0     Italy  Aromas include tropical fruit, broom, brimston...
1  Portugal  This is ripe and fruity, a wine that is smooth...
2        US  Tart and snappy, the flavors of lime flesh and...
3        US  Pineapple rind, lemon pith and orange blossom ...
4        US  Much like the regular bottling from 2012, this...

                          designation  points  price           province  \
0                        Vulkà Bianco      87    NaN  Sicily & Sardinia
1                            Avidagos      87   15.0              Douro
2                                 NaN      87   14.0             Oregon
3                Reserve Late Harvest      87   13.0           Michigan
4  Vintner's Reserve Wild Child Block      87   65.0             Oregon

              region_1           region_2         taster_name  \
0                 Etna                NaN       Kerin O’Keefe
1                  NaN                NaN          Roger Voss
2    Willamette Valley  Willamette Valley        Paul Gregutt
3  Lake Michigan Shore                NaN  Alexander Peartree
4    Willamette Valley  Willamette Valley        Paul Gregutt

  taster_twitter_handle                                              title  \
0          @kerinokeefe                  Nicosia 2013 Vulkà Bianco  (Etna)
1            @vossroger      Quinta dos Avidagos 2011 Avidagos Red (Douro)
2           @paulgwine       Rainstorm 2013 Pinot Gris (Willamette Valley)
```

0	White Blend	Nicosia
1	Portuguese Red	Quinta dos Avidagos
2	Pinot Gris	Rainstorm
3	Riesling	St. Julian
4	Pinot Noir	Sweet Cheeks

Let’s take a random sample of 500 records from this data. This will help with speeding up model training and testing, though it can easily be modified by the reader:

```
import pandas as pd
df = pd.read_csv("winemag-data-130k-v2.csv").sample(n=500,
random_state = 42)
```

Now, let’s print the info corresponding to our data which will give us an idea of which columns have missing values:

```
print(df.info())
```

```
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   country                500 non-null    object
 1   description            500 non-null    object
 2   designation            369 non-null    object
 3   points                 500 non-null    int64
 4   price                  471 non-null    float64
 5   province               500 non-null    object
 6   region_1               424 non-null    object
 7   region_2               188 non-null    object
 8   taster_name            407 non-null    object
 9   taster_twitter_handle  385 non-null    object
 10  title                  500 non-null    object
 11  variety                500 non-null    object
 12  winery                 500 non-null    object
```

Several columns have less than 500 non-null values, which correspond to missing values. First let's consider building a model that imputes missing 'price' values using the 'points'. To start, let's print the correlation between 'price' and 'points':

```
print("Correlation: ", df['points'].corr(df['price']))
```

```
Correlation:  0.4161667418606222
```

We see that there is a weak positive correlation. Let's build a linear regression model that uses 'points' to predict the 'price'. First, let's import the 'LinearRegresssion' module from 'scikit-learn':

```
from sklearn.linear_model import LinearRegression
```

filter out the missing values by selecting only positive price values:

```
import numpy as np
df_filter = df[df['price'] > 0].copy()
```

Let's also initialize lists we will use to store our predictions and actual values:

```
y_pred = []
y_true = []
```

We will use K-fold cross validation to validate our model. Let's import the 'KFolds' module from 'scikit-learn'. We will use 10 folds to validate our model:

```
from sklearn.model_selection import KFold
kf = KFold(n_splits=10, random_state = 42)
for train_index, test_index in kf.split(df_filter):
    df_test = df_filter.iloc[test_index]
    df_train = df_filter.iloc[train_index]
```

We can now define our input and output:

```
for train_index, test_index in kf.split(df_filter):
    ...
    X_train = np.array(df_train['points']).reshape(-1, 1)
    y_train = np.array(df_train['price']).reshape(-1, 1)
    X_test = np.array(df_test['points']).reshape(-1, 1)
    y_test = np.array(df_test['price']).reshape(-1, 1)
```

And fit our linear regression model:

```
    model = LinearRegression()
    model.fit(X_train, y_train)
```

Now let's generate and store our predictions:

```
for train_index, test_index in kf.split(df_filter):
    ...
    y_pred.append(model.predict(X_test)[0])
    y_true.append(y_test[0])
```

Now let's evaluate the performance of our model. Let's use mean squared error to evaluate the performance of our model:

```
print("Mean Square Error: ", mean_squared_error(y_true, y_pred))
```

```
Mean Square Error:  590.0704441239659
```

We see that the performance isn't too great. We can improve this by training on prices bound by the mean price plus one standard deviation:

```
df_filter = df[df['price'] <= df['price'].mean() + df['price'].std()
].copy()
...
print("Mean Square Error: ", mean_squared_error(y_true, y_pred))
```

```
Mean Square Error:  83.31517408259938
```

While this significantly improves performance this comes at the price of not being able to accurately impute values for highly priced wines. Instead of using a regression model of a wine ‘price’. First, let’s convert the categorical variables into categorical codes that can be handled by the random forests model:

```
df['country_cat'] = df['country'].astype('category')
df['country_cat'] = df['country_cat'].cat.codes

df['province_cat'] = df['province'].astype('category')
df['province_cat'] = df['province_cat'].cat.codes

df['winery_cat'] = df['winery'].astype('category')
df['winery_cat'] = df['winery_cat'].cat.codes

df['variety_cat'] = df['variety'].astype('category')
df['variety_cat'] = df['variety_cat'].cat.codes
```

Let’s increase the random sample size to 5000 :

```
df = pd.read_csv("winemag-data-130k-v2.csv").sample(n=5000,
random_state = 42)
```

Next, let’s import the random forest regressor module from scikit-learn. Let’s also define the list of features we will use to train our model:

```
from sklearn.ensemble import RandomForestRegressor
features = ['points', 'country_cat', 'province_cat', 'winery_cat',
'variety_cat']
```

Let’s train our model using a random forest with 1000 estimators and a max depth of 1000. Let’s then generate predictions and append them to a new list:

```
for train_index, test_index in kf.split(df_filter):
```

```
    y_test = np.array(df_test['price'])
    model = RandomForestRegressor(n_estimators = 1000, max_depth =
1000, random_state = 42)
    model.fit(X_train, y_train)

    y_pred_rf.append(model.predict(X_test)[0])
    y_true_rf.append(y_test[0])
```

Finally, let's evaluate the mean squared error for both the random forest and the linear regression models:

```
print("Mean Square Error (Linear Regression): ",
mean_squared_error(y_true, y_pred))
print("Mean Square Error (Random Forest): ",
mean_squared_error(y_pred_rf, y_true_rf))
```

```
Mean Square Error (Linear Regression):  104.60465409290084
Mean Square Error (Random Forest):  40.199467115694446
```

We see that the random forests model has superior performance. Now, let's predict the missing price values using our models and display sample predictions:

```
df_missing = df[df['price'].isnull()].copy()

X_test_lr = np.array(df_missing['points']).reshape(-1, 1)
X_test_rf = np.array(df_missing[features])

X_train_lr = np.array(df_filter['points']).reshape(-1, 1)
y_train_lr = np.array(df_filter['price']).reshape(-1, 1)

X_train_rf = np.array(df_filter[features])
y_train_rf = np.array(df_filter['price'])

model_lr = LinearRegression()
model_lr.fit(X_train_lr, y_train_lr)
```

```
print("Random forests regression predictions: ",
model_rf.predict(X_test_rf)[0])
```

```
Linear regression predictions:  28.05582905629467
Random forests regression predictions:  24.94
```

I'll stop here but I encourage you to play around with feature selection and hyper-parameter tuning to see if you can improve performance. Further, I encourage you to extend this data imputation model to impute missing values in categorical fields such as 'region_1' and 'designation'. Here you can build a tree-based classification model trained on categorical and numerical features to predict the missing values for the categories listed.

CONCLUSIONS

To summarize, in this post we discussed how to build machine learning models that we can use to impute missing values in data. First, we built a linear regression model trained on 'points' for reviewed wines to predict the price of wines. We then built a random forest model trained on 'points' and additional categorical variables to predict wine prices. We saw that the random forests model significantly outperformed the linear regression based data imputation model. I hope you found this post useful/interesting. The code from this post is available on GitHub. Thank you for reading!

Open in app Get started