Complete machine Learning & Data Science Boot Camp |udemy

Project on heart disease predication using RandomForestclassifier with prediction of accuracy score through Scikit-Learn.

Project on the cars sales prediction with all missing data using RandomForestRegressor with predication of model score through Scikit-Learn.

Elevating a ML model with Estimators built-in score () method, the scoring parameter and specific metric function for housing and heart disease through Scikit-Learn.

1. Get Data ready

In [8]:
```
import pandas as pd
heart_disease = pd.read_csv("heart-disease.csv")
heart_disease
```

Out[8]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	1
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2	1
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2	1
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
298	57	0	0	140	241	0	1	123	1	0.2	1	0	3	0
299	45	1	3	110	264	0	1	132	0	1.2	1	0	3	0
300	68	1	0	144	193	1	1	141	0	3.4	1	2	3	0
301	57	1	0	130	131	0	1	115	1	1.2	1	1	3	0
302	57	0	1	130	236	0	0	174	0	0.0	1	1	2	0

303 rows × 14 columns

In [9]:
```
# Create x (Feature Matrix)
x = heart_disease.drop("target", axis=1)

# create y (labels)
y = heart_disease["target"]
```

In [10]:
```
# 2. choose the wright model and hyperparameters
from sklearn.ensemble import RandomForestClassifier
clf = RandomForestClassifier(n_estimators=100)

clf.get_params()
```

Out[10]:
```
{'bootstrap': True,
 'ccp_alpha': 0.0,
 'class_weight': None,
 'criterion': 'gini',
 'max_depth': None,
 'max_features': 'sqrt',
 'max_leaf_nodes': None,
 'max_samples': None,
 'min_impurity_decrease': 0.0,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'n_estimators': 100,
 'n_jobs': None,
 'oob_score': False,
 'random_state': None,
 'verbose': 0,
 'warm_start': False}
```

In [11]:
```
from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2)
```

In [12]:
```
clf.fit(x_train, y_train);
```

In [13]:
```
y_preds = clf.predict(x_test)
y_preds
```

Out[13]:
```
array([0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0,
       1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0], dtype=int64)
```

In [14]:
```
y_test
```

Out[14]:
```
179    0
228    0
111    1
246    0
60     1
      ..
249    0
104    1
300    0
193    0
184    0
Name: target, Length: 61, dtype: int64
```

In [15]:
```
# 4. Evalute the model on the trainning data and test data
clf.score(x_train, y_train)
```

Out[15]:
```
1.0
```

In [16]:
```
clf.score(x_test, y_test)
```

Out[16]:
```
0.8524590163934426
```

In [17]:
```
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
print (classification_report(y_test, y_preds))
```

```
              precision    recall  f1-score   support

           0       0.86      0.83      0.84        29
           1       0.85      0.88      0.86        32

    accuracy                           0.85        61
   macro avg       0.85      0.85      0.85        61
weighted avg       0.85      0.85      0.85        61
```

In [18]:
```
confusion_matrix(y_test, y_preds)
```

Out[18]:
```
array([[24,  5],
       [ 4, 28]], dtype=int64)
```

In [19]:
```
accuracy_score(y_test, y_preds)
```

Out[19]:
```
0.8524590163934426
```

In [20]:
```
# imporve a model
# try different amount of n_estimators
import numpy  as np
np.random.seed(42)
for i in range(10, 100, 10):
    print(f"Trying model with {i} estimators...")
    clf = RandomForestClassifier (n_estimators=i).fit(x_train, y_train)
    print(f"Model accuracy on the test set:{clf.score(x_test, y_test)*100:.2f}%")
    print("")
```

```
Trying model with 10 estimators...
Model accuracy on the test set:85.25%

Trying model with 20 estimators...
Model accuracy on the test set:80.33%

Trying model with 30 estimators...
Model accuracy on the test set:83.61%

Trying model with 40 estimators...
Model accuracy on the test set:80.33%

Trying model with 50 estimators...
Model accuracy on the test set:86.89%

Trying model with 60 estimators...
Model accuracy on the test set:83.61%

Trying model with 70 estimators...
Model accuracy on the test set:83.61%

Trying model with 80 estimators...
Model accuracy on the test set:83.61%

Trying model with 90 estimators...
Model accuracy on the test set:81.97%
```

In [21]:
```
# 6. save a model and load it
import pickle
pickle.dump(clf, open("random_forest_model_1.pkl", "wb"))
```

In [22]:
```
loaded_model  = pickle.load(open("random_forest_model_1.Pkl","rb"))
loaded_model.score(x_test, y_test)
```

Out[22]:
```
0.819672131147541
```

1. getting our data ready to be used with machune learning Three main thing we have to do
2. Split the Data into features and labels (usuallys 'x' and 'y')
3. Filling (also called imputing) or disregarding missing values
4. converting non-numerical values to numerical values (also called feature coding)

In [23]:
```
heart_disease.head()
```

Out[23]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	1
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2	1
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2	1
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2	1

In [24]:
```
y = heart_disease["target"]
```

```
y.head()
```

Out[24]:
```
0    1
1    1
2    1
3    1
4    1
Name: target, dtype: int64
```

In [25]:
```
#split the data into training and test sets
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2)
```

In [26]:
```
x_train.shape, x_test.shape, y_train.shape, y_test.shape
```

Out[26]:
```
((242, 13), (61, 13), (242,), (61,))
```

In [27]:
```
x.shape
```

Out[27]:
```
(303, 13)
```

In [28]:
```
len(heart_disease)
```

Out[28]:
```
303
```

1.1 Make sure it's all Numerical

In [29]:
```
car_sales = pd.read_csv("car-sales-extended.csv")
car_sales.head()
```

Out[29]:

	Make	Colour	Odometer (KM)	Doors	Price
0	Honda	White	35431	4	15323
1	BMW	Blue	192714	5	19943
2	Honda	White	84714	4	28343
3	Toyota	White	154365	4	13434
4	Nissan	Blue	181577	3	14043

In [30]:
```
car_sales["Doors"].value_counts()
```

Out[30]:
```
4    856
5     79
3     65
Name: Doors, dtype: int64
```

In [31]:
```
len(car_sales)
```

Out[31]:
```
1000
```

In [32]:
```
# split into x/y
x = car_sales.drop("Price", axis=1)
y = car_sales["Price"]
```

```
# split into training and test
x_train, X_test, Y_train, Y_test = train_test_split(x, y, test_size= 0.2)
```

In [34]:
```
x
```

Out[34]:

	Make	Colour	Odometer (KM)	Doors
0	Honda	White	35431	4
1	BMW	Blue	192714	5
2	Honda	White	84714	4
3	Toyota	White	154365	4
4	Nissan	Blue	181577	3
...	...	...	...	...
995	Toyota	Black	35820	4
996	Nissan	White	155144	3
997	Nissan	Blue	66604	4
998	Honda	White	215883	4
999	Toyota	Blue	248360	4

1000 rows × 4 columns

In [35]:
```
#turn the catogories into Number
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer

categorical_features = ["Make", "Colour", "Doors"]
one_hot = OneHotEncoder()
transformer = ColumnTransformer([("one_hot",
                                  one_hot,
                                  categorical_features)],
                                  remainder="passthrough")
transformed_x = transformer.fit_transform(x)
transformed_x
```

Out[35]:
```
array([[0.00000e+00, 1.00000e+00, 0.00000e+00, ..., 1.00000e+00,
        0.00000e+00, 3.54310e+04],
       [1.00000e+00, 0.00000e+00, 0.00000e+00, ..., 0.00000e+00,
        1.00000e+00, 1.92714e+05],
       [0.00000e+00, 1.00000e+00, 0.00000e+00, ..., 1.00000e+00,
        0.00000e+00, 8.47140e+04],
       ...,
       [0.00000e+00, 0.00000e+00, 1.00000e+00, ..., 1.00000e+00,
        0.00000e+00, 6.66040e+04],
       [0.00000e+00, 1.00000e+00, 0.00000e+00, ..., 1.00000e+00,
        0.00000e+00, 2.15883e+05],
       [0.00000e+00, 0.00000e+00, 0.00000e+00, ..., 1.00000e+00,
        0.00000e+00, 2.48360e+05]])
```

In [36]:
```
x.head()
```

Out[36]:

	Make	Colour	Odometer (KM)	Doors
0	Honda	White	35431	4
1	BMW	Blue	192714	5
2	Honda	White	84714	4
3	Toyota	White	154365	4
4	Nissan	Blue	181577	3

In [37]:
```
pd.DataFrame(transformed_x)
```

Out[37]:

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0	35431.0
1	1.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	192714.0
2	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0	84714.0
3	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0	154365.0
4	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	181577.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...
995	0.0	0.0	0.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	35820.0
996	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	1.0	0.0	0.0	155144.0
997	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0	66604.0
998	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0	215883.0
999	0.0	0.0	0.0	1.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0	248360.0

1000 rows × 13 columns

In [38]:
```
dummies = pd.get_dummies(car_sales[["Make", "Colour", "Doors"]])
dummies
```

Out[38]:

	Doors	Make_BMW	Make_Honda	Make_Nissan	Make_Toyota	Colour_Black	Colour_Blue	Colou
0	4	0	1	0	0	0	0	
1	5	1	0	0	0	0	1	
2	4	0	1	0	0	0	0	
3	4	0	0	0	1	0	0	
4	3	0	0	1	0	0	1	
...	...	...	...	...	...	...	...	
995	4	0	0	0	1	1	0	
996	3	0	0	1	0	0	0	
997	4	0	0	1	0	0	1	
998	4	0	1	0	0	0	0	
999	4	0	0	0	1	0	1	

1000 rows × 10 columns

In [39]:
```
# Let's refit the model
np.random.seed(42)
x_train, x_test, y_train, y_test = train_test_split(transformed_x, y, test_size=0.2)
model.fit(x_train, y_train)
```

Out[39]:

▾ RandomForestRegressor

RandomForestRegressor()

In [40]:
```
model.score(x_test, y_test)
```

Out[40]:
```
0.3235867221569877
```

1.2 what if there were missing values?

1.Fill then with some value (also known as imputation)

2.Remove the sample with missing Data together

In [41]:
```
#import cars sell missing Data
cars_sales_missing = pd.read_csv("car-sales-extended-missing-data.csv")
cars_sales_missing.head()
```

Out[41]:

	Make	Colour	Odometer (KM)	Doors	Price
0	Honda	White	35431.0	4.0	15323.0
1	BMW	Blue	192714.0	5.0	19943.0
2	Honda	White	84714.0	4.0	28343.0
3	Toyota	White	154365.0	4.0	13434.0
4	Nissan	Blue	181577.0	3.0	14043.0

In [42]:
```
cars_sales_missing.isna().sum()
```

Out[42]:
```
Make             49
Colour           50
Odometer (KM)    50
Doors            50
Price            50
dtype: int64
```

In [43]:
```
cars_sales_missing
```

Out[43]:

	Make	Colour	Odometer (KM)	Doors	Price
0	Honda	White	35431.0	4.0	15323.0
1	BMW	Blue	192714.0	5.0	19943.0
2	Honda	White	84714.0	4.0	28343.0
3	Toyota	White	154365.0	4.0	13434.0
4	Nissan	Blue	181577.0	3.0	14043.0
...	...	...	...	...	...
995	Toyota	Black	35820.0	4.0	32042.0
996	NaN	White	155144.0	3.0	5716.0
997	Nissan	Blue	66604.0	4.0	31570.0
998	Honda	White	215883.0	4.0	4001.0
999	Toyota	Blue	248360.0	4.0	12732.0

1000 rows × 5 columns

In [44]:
```
cars_sales_missing["Doors"].value_counts()
```

Out[44]:
```
4.0    811
5.0     75
3.0     64
Name: Doors, dtype: int64
```

option 1: Fill missing data with Pandas

In [45]:
```
#Fill the "make" column
cars_sales_missing["Make"].fillna("missing", inplace=True)
```

```
#Fill the "Colour" column
cars_sales_missing["Colour"].fillna("missing", inplace=True)

#Fill the "Odometer(KM)" column
cars_sales_missing["Odometer (KM)"].fillna(cars_sales_missing["Odometer (KM)"].mean(),

#Fill the "doors" column
cars_sales_missing["Doors"].fillna(4, inplace=True)
```

In [46]:
```
# check out DataFrame again()
cars_sales_missing.isna().sum()
```

Out[46]:
```
Make             0
Colour           0
Odometer (KM)    0
Doors            0
Price           50
dtype: int64
```

In [47]:
```
# Remove rows with missing Price.value
cars_sales_missing.dropna(inplace =True)
```

In [48]:
```
cars_sales_missing.isna().sum()
```

Out[48]:
```
Make            0
Colour          0
Odometer (KM)   0
Doors           0
Price           0
dtype: int64
```

In [49]:
```
len(cars_sales_missing)
```

Out[49]:
```
950
```

In [50]:
```
x = cars_sales_missing.drop("Price", axis=1)
y = cars_sales_missing["Price"]
```

In [51]:
```
#turn the catogories into Number
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer

categorical_features = ["Make", "Colour", "Doors"]
one_hot = OneHotEncoder()
transformer = ColumnTransformer([("one_hot",
                                  one_hot,
                                  categorical_features)],
                                  remainder="passthrough")
transformed_x = transformer.fit_transform(cars_sales_missing)
transformed_x
```

Out[51]:
```
array([[0.00000e+00, 1.00000e+00, 0.00000e+00, ..., 0.00000e+00,
        3.54310e+04, 1.53230e+04],
       [1.00000e+00, 0.00000e+00, 0.00000e+00, ..., 1.00000e+00,
        1.92714e+05, 1.99430e+04],
       [0.00000e+00, 1.00000e+00, 0.00000e+00, ..., 0.00000e+00,
        8.47140e+04, 2.83430e+04],
       ...,
       [0.00000e+00, 0.00000e+00, 1.00000e+00, ..., 0.00000e+00,
        6.66040e+04, 3.15700e+04],
       [0.00000e+00, 1.00000e+00, 0.00000e+00, ..., 0.00000e+00,
        2.15883e+05, 4.00100e+03],
       [0.00000e+00, 0.00000e+00, 0.00000e+00, ..., 0.00000e+00,
        2.48360e+05, 1.27320e+04]])
```

Option 2:Fill misisng values with Scikit learn

In [52]:
```
cars_sales_missing= pd.read_csv("car-sales-extended-missing-data.csv")
cars_sales_missing.head()
```

Out[52]:

	Make	Colour	Odometer (KM)	Doors	Price
0	Honda	White	35431.0	4.0	15323.0
1	BMW	Blue	192714.0	5.0	19943.0
2	Honda	White	84714.0	4.0	28343.0
3	Toyota	White	154365.0	4.0	13434.0
4	Nissan	Blue	181577.0	3.0	14043.0

In [53]:
```
cars_sales_missing.isna().sum()
```

Out[53]:
```
Make             49
Colour           50
Odometer (KM)    50
Doors            50
Price            50
dtype: int64
```

In [54]:
```
#Drop the rows with no labels
cars_sales_missing.dropna(subset=["Price"], inplace=True)
cars_sales_missing.isna().sum()
```

Out[54]:
```
Make             47
Colour           46
Odometer (KM)    48
Doors            47
Price             0
dtype: int64
```

In [55]:
```
# split in x and y
x = cars_sales_missing.drop("Price", axis=1)
y = cars_sales_missing["Price"]
```

In [56]:
```
#Fill missing values with the scikit-learn
from sklearn.impute import SimpleImputer
from sklearn.compose import ColumnTransformer

#Fill categorical value with "missing" & Numerical values with mean
```

```
cat_imputer = SimpleImputer(strategy="constant", fill_value="missing")
door_imputer = SimpleImputer(strategy="constant", fill_value=4)
num_imputer = SimpleImputer(strategy = "mean")

# define columns
cat_features = ["Make","Colour"]
door_features = ["Doors"]
num_features = ["Odometer (KM)"]

#create an imputer (something that fills missing data)
imputer = ColumnTransformer([
    ("cat_imputer", cat_imputer, cat_features),
    ("door_imputer", door_imputer, door_features),
    ("num_imputer", num_imputer, num_features)
])

# transform the data
filled_x = imputer.fit_transform(x)
filled_x
```

Out[56]:

```
array([['Honda', 'White', 4.0, 35431.0],
       ['BMW', 'Blue', 5.0, 192714.0],
       ['Honda', 'White', 4.0, 84714.0],
       ...,
       ['Nissan', 'Blue', 4.0, 66604.0],
       ['Honda', 'White', 4.0, 215883.0],
       ['Toyota', 'Blue', 4.0, 248360.0]], dtype=object)
```

In [57]:

```
cars_sales_filled = pd.DataFrame(filled_x,
                                 columns=["Make", "Colour", "Doors", "Odometer (KM)"])
cars_sales_filled.head()
```

Out[57]:

	Make	Colour	Doors	Odometer (KM)
0	Honda	White	4.0	35431.0
1	BMW	Blue	5.0	192714.0
2	Honda	White	4.0	84714.0
3	Toyota	White	4.0	154365.0
4	Nissan	Blue	3.0	181577.0

In [58]:

```
cars_sales_filled.isna().sum()
```

Out[58]:

```
Make             0
Colour           0
Doors            0
Odometer (KM)    0
dtype: int64
```

In [59]:

```
#turn the catogories into Number
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer

categorical_features = ["Make", "Colour", "Doors"]
one_hot = OneHotEncoder()
transformer = ColumnTransformer([("one_hot",
                                  one_hot,
                                  categorical_features)],
```

```
                                    remainder="passthrough")
transformed_x = transformer.fit_transform(cars_sales_filled)
transformed_x
```

Out[59]:
```
<950x15 sparse matrix of type '<class 'numpy.float64'>'
        with 3800 stored elements in Compressed Sparse Row format>
```

In [60]:
```
# Now we've gotten our data numbers and filled(no missing values)
# Let's fit a model
import numpy as np
np.random.seed(42)
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(transformed_x,
                                                    y,
                                                    test_size=0.2)
model = RandomForestRegressor(n_estimators =100)
model.fit(x_train, y_train)
model.score(x_test, y_test)
```

Out[60]:
```
0.21990196728583944
```

In [61]:
```
len(cars_sales_filled), len(car_sales)
```

Out[61]:
```
(950, 1000)
```

Note: The 50 less values in the transformed data is because we dropped the rows(50 total) with missing values with price columns

2. choosing the right estimator/algorithm for your problem

some things to note: sklearn refers to machine learning models, algorithms, estimators

classification problem-predicting a catagory (heart disease or not)

sometimes you will see clf (short for classifier) used as classification estimator

Regression Problem-Predicting a number(selling price of a car)

2.1 Picking a machine learning model for a regression model

Let of use a california data set https://scikit-learn.org/stable/modules/generated/sklearn.datasets.fetch_california_housing.html

In [62]:
```
# Get california housing Data set
from sklearn.datasets import fetch_california_housing
housing = fetch_california_housing()
housing
```

Out[62]:
{'data': array([[8.3252   ,   41.   ,   6.98412698, ...,   2.55555556,
           37.88   , -122.23   ],
        [8.3014   ,   21.   ,   6.23813708, ...,   2.10984183,
           37.86   , -122.22   ],
        [7.2574   ,   52.   ,   8.28813559, ...,   2.80225989,
           37.85   , -122.24   ],
        ...,
        [1.7   ,   17.   ,   5.20554273, ...,   2.3256351 ,
           39.43   , -121.22   ],
        [1.8672   ,   18.   ,   5.32951289, ...,   2.12320917,
           39.43   , -121.32   ],
        [2.3886   ,   16.   ,   5.25471698, ...,   2.61698113,
           39.37   , -121.24   ]]),
 'target': array([4.526, 3.585, 3.521, ..., 0.923, 0.847, 0.894]),
 'frame': None,
 'target_names': ['MedHouseVal'],
 'feature_names': ['MedInc',
  'HouseAge',
  'AveRooms',
  'AveBedrms',
  'Population',
  'AveOccup',
  'Latitude',
  'Longitude'],
 'DESCR': '.. _california_housing_dataset:\n\nCalifornia Housing dataset\n----------
---------------\n\n**Data Set Characteristics:**\n\n   :Number of Instances: 20640\n
\n   :Number of Attributes: 8 numeric, predictive attributes and the target\n\n   :
Attribute Information:\n   - MedInc   median income in block group\n
- HouseAge   median house age in block group\n   - AveRooms   average numb
er of rooms per household\n   - AveBedrms   average number of bedrooms per hou
sehold\n   - Population   block group population\n   - AveOccup   avera
ge number of household members\n   - Latitude   block group latitude\n
- Longitude   block group longitude\n\n   :Missing Attribute Values: None\n\nThis
dataset was obtained from the StatLib repository.\nhttps://www.dcc.fc.up.pt/~ltorgo/R
egression/cal_housing.html\n\nThe target variable is the median house value for Calif
ornia districts,\nexpressed in hundreds of thousands of dollars ($100,000).\n\nThis d
ataset was derived from the 1990 U.S. census, using one row per census\nblock group.
A block group is the smallest geographical unit for which the U.S.\nCensus Bureau pub
lishes sample data (a block group typically has a population\nof 600 to 3,000 peopl
e).\n\nAn household is a group of people residing within a home. Since the average\nn
umber of rooms and bedrooms in this dataset are provided per household, these\ncolumn
s may take surpinsingly large values for block groups with few households\nand many e
mpty houses, such as vacation resorts.\n\nIt can be downloaded/loaded using the\n:fun
c:`sklearn.datasets.fetch_california_housing` function.\n\n.. topic:: References\n\n
- Pace, R. Kelley and Ronald Barry, Sparse Spatial Autoregressions,\n   Statistics
and Probability Letters, 33 (1997) 291-297\n'}

In [63]:
```
housing_df = pd.DataFrame(housing["data"], columns=housing["feature_names"])
housing_df
```

Out[63]:

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude
0	8.3252	41.0	6.984127	1.023810	322.0	2.555556	37.88	-122.23
1	8.3014	21.0	6.238137	0.971880	2401.0	2.109842	37.86	-122.22
2	7.2574	52.0	8.288136	1.073446	496.0	2.802260	37.85	-122.24
3	5.6431	52.0	5.817352	1.073059	558.0	2.547945	37.85	-122.25
4	3.8462	52.0	6.281853	1.081081	565.0	2.181467	37.85	-122.25
...	...	...	...	...	...	...	...	...
20635	1.5603	25.0	5.045455	1.133333	845.0	2.560606	39.48	-121.09
20636	2.5568	18.0	6.114035	1.315789	356.0	3.122807	39.49	-121.21
20637	1.7000	17.0	5.205543	1.120092	1007.0	2.325635	39.43	-121.22
20638	1.8672	18.0	5.329513	1.171920	741.0	2.123209	39.43	-121.32
20639	2.3886	16.0	5.254717	1.162264	1387.0	2.616981	39.37	-121.24

20640 rows × 8 columns

In [64]:
```
housing_df["target"]=housing["target"]

housing_df["MedHouseVal"]=housing["target"]
housing_df.head()
```

Out[64]:

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude	target
0	8.3252	41.0	6.984127	1.023810	322.0	2.555556	37.88	-122.23	4.526
1	8.3014	21.0	6.238137	0.971880	2401.0	2.109842	37.86	-122.22	3.585
2	7.2574	52.0	8.288136	1.073446	496.0	2.802260	37.85	-122.24	3.521
3	5.6431	52.0	5.817352	1.073059	558.0	2.547945	37.85	-122.25	3.413
4	3.8462	52.0	6.281853	1.081081	565.0	2.181467	37.85	-122.25	3.422

In [65]:
```
housing_df = housing_df.drop("MedHouseVal", axis=1)
housing_df
```

Out[65]:

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude	targ
0	8.3252	41.0	6.984127	1.023810	322.0	2.555556	37.88	-122.23	4.5
1	8.3014	21.0	6.238137	0.971880	2401.0	2.109842	37.86	-122.22	3.5
2	7.2574	52.0	8.288136	1.073446	496.0	2.802260	37.85	-122.24	3.5
3	5.6431	52.0	5.817352	1.073059	558.0	2.547945	37.85	-122.25	3.4
4	3.8462	52.0	6.281853	1.081081	565.0	2.181467	37.85	-122.25	3.4
...	...	...	...	...	...	...	...	...	
20635	1.5603	25.0	5.045455	1.133333	845.0	2.560606	39.48	-121.09	0.7
20636	2.5568	18.0	6.114035	1.315789	356.0	3.122807	39.49	-121.21	0.7
20637	1.7000	17.0	5.205543	1.120092	1007.0	2.325635	39.43	-121.22	0.9
20638	1.8672	18.0	5.329513	1.171920	741.0	2.123209	39.43	-121.32	0.8
20639	2.3886	16.0	5.254717	1.162264	1387.0	2.616981	39.37	-121.24	0.8

20640 rows × 9 columns

In [66]:
```
#Import algorithm/estimator
from sklearn.linear_model import Ridge
from sklearn.model_selection import train_test_split
import numpy as np
# setup random seed
np.random.seed(42)

# create the  data
x = housing_df.drop("target", axis=1)
y = housing_df["target"] # medium house price is $1000,000s

# split in train and test sets
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2)

#Instantiate and fit the model (on the trainning set)
model = Ridge()
model.fit(x_train, y_train)

#check the score of the model (on the test set)
model.score(x_test, y_test)
```

Out[66]:
```
0.5758549611440126
```

In [67]:
```
#import the RandomForestRegressor class from the ensemble module
from sklearn.ensemble import RandomForestRegressor

#setup random seed
np.random.seed(42)

# create the data
x = housing_df.drop ("target", axis=1)
y = housing_df["target"]
```

```
# split into train and test sets
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2)


# create Random forest model
model = RandomForestRegressor()
model.fit(x_train, y_train)

#check the score of the model(on the test set)
model.score(x_test, y_test)
```

Out[67]:
```
0.8065734772187598
```

2.2 choosing estimator for classification problem Let's go the map...https://scikit-learn.org/stable/tutorial/machine_learning_map/index.html

In [68]:
```
heart_disease = pd.read_csv("heart-disease.csv")
heart_disease.head()
```

Out[68]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	1
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2	1
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2	1
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2	1

In [69]:
```
len(heart_disease)
```

Out[69]:
```
303
```

consulting the map that says to try LinearSVC

In [70]:
```
#import the LinearSVC class
from sklearn.svm import LinearSVC

#setup random seed
np.random.seed(42)

# create the data
x = heart_disease.drop ("target", axis=1)
y = heart_disease["target"]

# split into train and test sets
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2)


# create LinearSVC model
clf = LinearSVC()
clf.fit(x_train, y_train)

#check the score of the model(on the test set)
clf.score(x_test, y_test)
```

```
C:\Users\Dipranjan\anaconda3\lib\site-packages\sklearn\svm\_base.py:1244: Convergence
Warning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
```

Out[70]:
```
0.8688524590163934
```

In [71]:
```
heart_disease["target"].value_counts()
```

Out[71]:
```
1    165
0    138
Name: target, dtype: int64
```

In [72]:
```
#import the RandomForestRegressor class from the ensemble module
from sklearn.ensemble import RandomForestClassifier

#setup random seed
np.random.seed(42)

# create the data
x = heart_disease.drop ("target", axis=1)
y = heart_disease["target"]

# split into train and test sets
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2)


# create Random forest model
clf = RandomForestClassifier()
clf.fit(x_train, y_train)

#check the score of the model(on the test set)
clf.score(x_test, y_test)
```

Out[72]:
```
0.8524590163934426
```

Fit the model/algorithm on our data and use it to make it prediciton

3.1 fitting the model to the data

Different names for:

- x = features, features variable, data
- y = labels, targets, target variables

In [73]:
```
#import the RandomForestRegressor class from the ensemble module
from sklearn.ensemble import RandomForestClassifier

#setup random seed
np.random.seed(42)

# create the data
x = heart_disease.drop ("target", axis=1)
y = heart_disease["target"]
```

```
# split into train and test sets
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2)


# create Random forest model
clf = RandomForestClassifier(n_estimators = 100)

# Fit the model to the data (training the machine learning model)
clf.fit(x_train, y_train)

#check the score of the model(on the test set) (use the patterns the model have learne
clf.score(x_test, y_test)
```

Out[73]:
```
0.8524590163934426
```

In [74]:
```
x.head()
```

Out[74]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2

In [75]:
```
y.tail()
```

Out[75]:
```
298    0
299    0
300    0
301    0
302    0
Name: target, dtype: int64
```

3.2 Make predicitons using a machine learning model

2 ways to make predicitons

1. predict()

2. predict_proba()

In [76]:
```
clf.predict(x_test)
```

Out[76]:
```
array([0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0,
       1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0], dtype=int64)
```

In [77]:
```
np.array([y_test])
```

Out[77]:
```
array([[0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0,
        0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0]], dtype=int64)
```

In [78]:
```
#compare the predictions to the true labels to elevate the model
y_preds = clf.predict(x_test)
np.mean(y_preds == y_test)
```

Out[78]:
```
0.8524590163934426
```

In [79]:
```
clf.score(x_test, y_test)
```

Out[79]:
```
0.8524590163934426
```

In [80]:
```
from sklearn.metrics import accuracy_score
accuracy_score(y_test, y_preds)
```

Out[80]:
```
0.8524590163934426
```

make the predictions with predict_proba()

In [81]:
```
# predict_proba() returns probabilites of a classification label
clf.predict_proba(x_test[:5])
```

Out[81]:
```
array([[0.89, 0.11],
       [0.49, 0.51],
       [0.43, 0.57],
       [0.84, 0.16],
       [0.18, 0.82]])
```

In [82]:
```
# Let predict on the same data...
clf.predict(x_test[:5])
```

Out[82]:
```
array([0, 1, 1, 0, 1], dtype=int64)
```

In [83]:
```
heart_disease["target"].value_counts()
```

Out[83]:
```
1    165
0    138
Name: target, dtype: int64
```

In [84]:
```
x_test[:5]
```

Out[84]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal
179	57	1	0	150	276	0	0	112	1	0.6	1	1	1
228	59	1	3	170	288	0	0	159	0	0.2	1	0	3
111	57	1	2	150	126	1	1	173	0	0.2	2	1	3
246	56	0	0	134	409	0	0	150	1	1.9	1	2	3
60	71	0	2	110	265	1	0	130	0	0.0	2	1	2

predict() can be used for predcitons model too

In [85]:
```
housing_df.head()
```

Out[85]:

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude	target
0	8.3252	41.0	6.984127	1.023810	322.0	2.555556	37.88	-122.23	4.526
1	8.3014	21.0	6.238137	0.971880	2401.0	2.109842	37.86	-122.22	3.585
2	7.2574	52.0	8.288136	1.073446	496.0	2.802260	37.85	-122.24	3.521
3	5.6431	52.0	5.817352	1.073059	558.0	2.547945	37.85	-122.25	3.413
4	3.8462	52.0	6.281853	1.081081	565.0	2.181467	37.85	-122.25	3.422

In [86]:
```
#import the RandomForestRegressor class from the ensemble module
from sklearn.ensemble import RandomForestRegressor

#setup random seed
np.random.seed(42)

# create the data
x = housing_df.drop ("target", axis=1)
y = housing_df["target"]

# split into train and test sets
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2)


# create Random forest model
model = RandomForestRegressor()

# fit the model
model.fit(x_train, y_train)

# make predicitions
y_preds = model.predict(x_test)
```

In [87]:
```
y_preds[:10]
```

Out[87]:
```
array([0.49384  , 0.75494  , 4.9285964, 2.54316  , 2.33176  , 1.6525301,
       2.34323  , 1.66182  , 2.47489  , 4.8344779])
```

In [88]:
```
np.array(y_test[:10])
```

Out[88]:
```
array([0.477  , 0.458  , 5.00001, 2.186  , 2.78   , 1.587  , 1.982  ,
       1.575  , 3.4    , 4.466  ])
```

In [89]:
```
len(y_test)
```

Out[89]:
```
4128
```

In [90]:
```
len(y_preds)
```

Out[90]:
```
4128
```

In [91]:
```
#compare the predicition to the truth
from sklearn.metrics import mean_absolute_error
mean_absolute_error(y_test, y_preds)
```

Out[91]:
```
0.32659871732073664
```

In [92]:
```
housing_df["target"]
```

Out[92]:
```
0        4.526
1        3.585
2        3.521
3        3.413
4        3.422
         ...
20635    0.781
20636    0.771
20637    0.923
20638    0.847
20639    0.894
Name: target, Length: 20640, dtype: float64
```

4. Evaluating a machine learning model

Three ways to evaluate scikit-learning models / estimators

1. Estimators built-in score () method
2. the scoring parameter
3. Problem specific metric functions

4.1 Evaluating a model with score method

In [93]:
```
#import the RandomForestRegressor class from the ensemble module
from sklearn.ensemble import RandomForestClassifier

#setup random seed
np.random.seed(42)

# create the data
x = heart_disease.drop ("target", axis=1)
y = heart_disease["target"]

# split into train and test sets
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2)


# create Random forest model
clf = RandomForestClassifier(n_estimators = 100)

# Fit the model to the data (training the machine learning model)
clf.fit(x_train, y_train)
```

Out[93]:
```
RandomForestClassifier
RandomForestClassifier()
```

In [94]:
```
# The highest value of the score method is 1.0 and lowest is 0.0
clf.score(x_train, y_train)
```

Out[94]:
```
1.0
```

In [95]:
```
clf.score(x_test, y_test)
```

Out[95]:
```
0.8524590163934426
```

Let use the score on the regression problem

In [96]:
```
#import the RandomForestRegressor class from the ensemble module
from sklearn.ensemble import RandomForestRegressor

#setup random seed
np.random.seed(42)

# create the data
x = housing_df.drop ("target", axis=1)
y = housing_df["target"]

# split into train and test sets
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2)


# create Random forest model
model = RandomForestRegressor(n_estimators =100)

# fit the model
model.fit(x_train, y_train)
```

Out[96]:
```
RandomForestRegressor
RandomForestRegressor()
```

In [97]:
```
#The default score() evaluation metrics is r_squared for regression algortithm
model.score(x_train, y_train)
```

Out[97]:
```
0.9736801960414609
```

In [98]:
```
model.score(x_test, y_test)
```

Out[98]:
```
0.8065734772187598
```

4.2 Evaluating a model using the scoring parameter

In [100…
```
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import RandomForestClassifier

np.random.seed(42)

x = heart_disease.drop("target", axis=1)
y = heart_disease["target"]

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2)

clf = RandomForestClassifier(n_estimators =100)

clf.fit(x_train, y_train);
```

In [101…
```
import pandas as pd
import numpy as np
cross_val_score(clf, x, y, cv=5)
```

Out[101]:
```
array([0.81967213, 0.86885246, 0.81967213, 0.78333333, 0.76666667])
```

In [102…
```
clf.score(x_test,y_test)
```

Out[102]:
```
0.8524590163934426
```

In [103…
```
cross_val_score(clf, x, y, cv=10)
```

Out[103]:
```
array([0.90322581, 0.80645161, 0.87096774, 0.9       , 0.86666667,
       0.8       , 0.73333333, 0.86666667, 0.73333333, 0.8       ])
```

In [104…
```
np.random.seed(42)

#single training and test split score
clf_single_score = clf.score(x_test, y_test)

# Take the mean of 5-fold  cross validation score
clf_cross_val_score = np.mean(cross_val_score(clf, x, y, cv=5))

# compare the two
clf_single_score, clf_cross_val_score
```

Out[104]:
```
(0.8524590163934426, 0.8248087431693989)
```

In [108…
```
# scoring paramter set to none to default
cross_val_score(clf, x, y, cv=5, scoring=None)
```

Out[108]:
```
array([0.78688525, 0.86885246, 0.80327869, 0.78333333, 0.76666667])
```

Default scoring parameter of classifier = mean accuracy

clf.score()

4.2.1 Classification model evaluation metrics

1. accuracy
2. Area under ROC curve
3. Confusion Matrix
4. Classification Report

Accuracy

In [120…
```
heart_disease.head()
```

Out[120]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	1
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2	1
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2	1
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2	1

In [121…
```
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import RandomForestClassifier

np.random.seed(42)

x = heart_disease.drop("target", axis=1)
y = heart_disease["target"]

clf =RandomForestClassifier(n_estimators=100)
cross_val_score(clf, x, y, cv=5)
```

Out[121]:
```
array([0.81967213, 0.90163934, 0.83606557, 0.78333333, 0.78333333])
```

In [122…
```
np.mean(cross_val_score(clf, x, y, cv=5))
```

Out[122]:
```
0.8018032786885245
```

In [123…
```
print(f"Heart Disease Classifier Accuracy:{np.mean(cross_val_score(clf, x, y, cv=5)) *
```

```
Heart Disease Classifier Accuracy:81.17%
```

In [ ]: