Scikit-Learn provides a package of built-in functions to fetch datasets commonly used by the machine learning community.

https://scikit-learn.org/stable/datasets/index.html#california-housing-dataset

(a) Download the above dataset using sklearn.datasets.fetch_california_housing function.

```
import numpy as np
import pandas as pd
from sklearn import datasets
from math import sqrt, sin, cos, radians
```

```
housing = datasets.fetch_california_housing()
```

(b) From the downloaded dataset, examine the features and define a pandas DataFrame to hold the district records as rows. Add the target median housing price as a new column in the data frame. The shape of the data frame should be (20640, 9).

```
print(housing.DESCR)
```

```
.. _california_housing_dataset:

California Housing dataset
--------------------------

**Data Set Characteristics:**

    :Number of Instances: 20640

    :Number of Attributes: 8 numeric, predictive attributes and the target

    :Attribute Information:
        - MedInc        median income in block group
        - HouseAge      median house age in block group
        - AveRooms      average number of rooms per household
        - AveBedrms     average number of bedrooms per household
        - Population    block group population
        - AveOccup      average number of household members
        - Latitude      block group latitude
        - Longitude     block group longitude

    :Missing Attribute Values: None

This dataset was obtained from the StatLib repository.
https://www.dcc.fc.up.pt/~ltorgo/Regression/cal_housing.html

The target variable is the median house value for California districts,
```

```
    expressed in hundreds of thousands of dollars ($100,000).

    This dataset was derived from the 1990 U.S. census, using one row per census
    block group. A block group is the smallest geographical unit for which the U.S.
    Census Bureau publishes sample data (a block group typically has a population
    of 600 to 3,000 people).

    An household is a group of people residing within a home. Since the average
    number of rooms and bedrooms in this dataset are provided per household, these
    columns may take surpinsingly large values for block groups with few households
    and many empty houses, such as vacation resorts.

    It can be downloaded/loaded using the
    :func:`sklearn.datasets.fetch_california_housing` function.

    .. topic:: References

        - Pace, R. Kelley and Ronald Barry, Sparse Spatial Autoregressions,
          Statistics and Probability Letters, 33 (1997) 291-297
```

```
housing_df = pd.DataFrame(data=housing.data, columns=housing.feature_names)
```

```
housing_df.head()
```

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Lo
0	8.3252	41.0	6.984127	1.023810	322.0	2.555556	37.88	
1	8.3014	21.0	6.238137	0.971880	2401.0	2.109842	37.86	
2	7.2574	52.0	8.288136	1.073446	496.0	2.802260	37.85	
3	5.6431	52.0	5.817352	1.073059	558.0	2.547945	37.85	
4	3.8462	52.0	6.281853	1.081081	565.0	2.181467	37.85	

```
print(housing.target)
```

```
    [4.526 3.585 3.521 ... 0.923 0.847 0.894]
```

```
housing_df['MedianPrice'] = housing.target
```

```
housing_df.shape
```

```
    (20640, 9)
```

```
housing_df
```

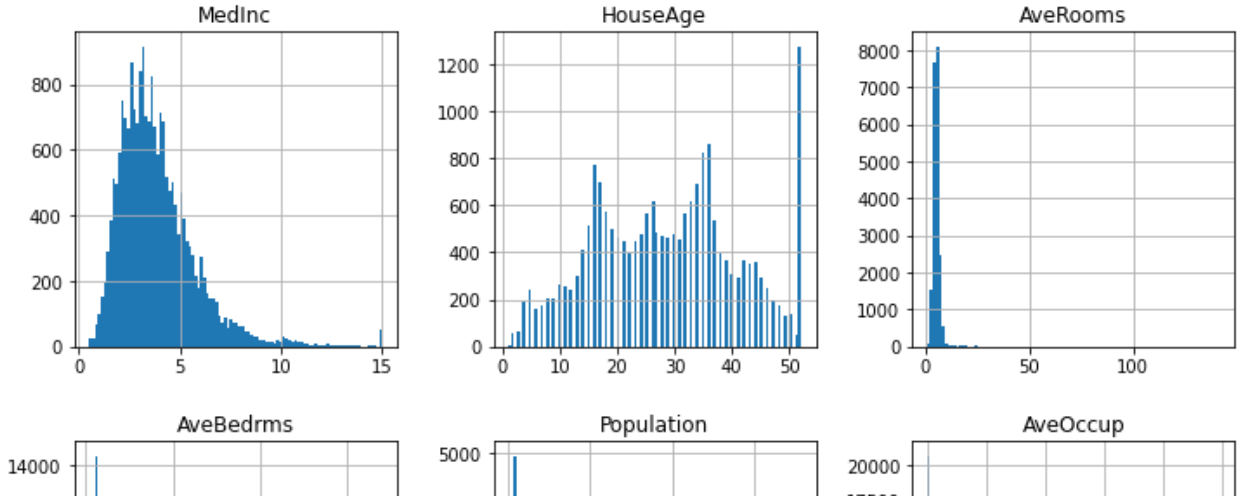
	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude
0	8.3252	41.0	6.984127	1.023810	322.0	2.555556	37.88
1	8.3014	21.0	6.238137	0.971880	2401.0	2.109842	37.86
2	7.2574	52.0	8.288136	1.073446	496.0	2.802260	37.85
3	5.6431	52.0	5.817352	1.073059	558.0	2.547945	37.85
4	3.8462	52.0	6.281853	1.081081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

20640 rows × 9 columns

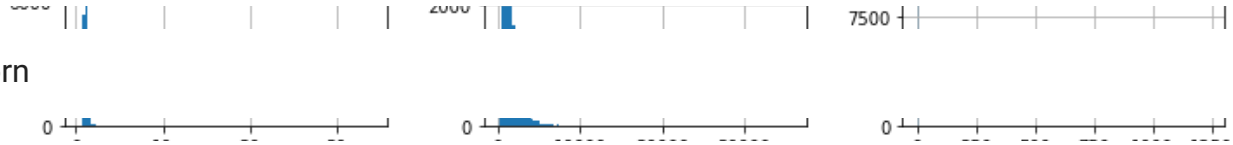
(c) Plot a histogram for each attribute in the data frame. Use bins=100.

```
import matplotlib.pyplot as plt
housing_df.hist(figsize=(12, 12), bins=100)
```

```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x7f37031ab610>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f3703941cd0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f3703b11310>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7f3703ac2890>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f3703a76e90>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f3703aa6b10>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7f3703a71590>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f3703a289d0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f3703a28a10>]],
      dtype=object)
```



(d) Plot a scatter diagram to show the location (latitude-longitude) of the districts in the records. Use the median housing price as a scale to color the points in the plot.

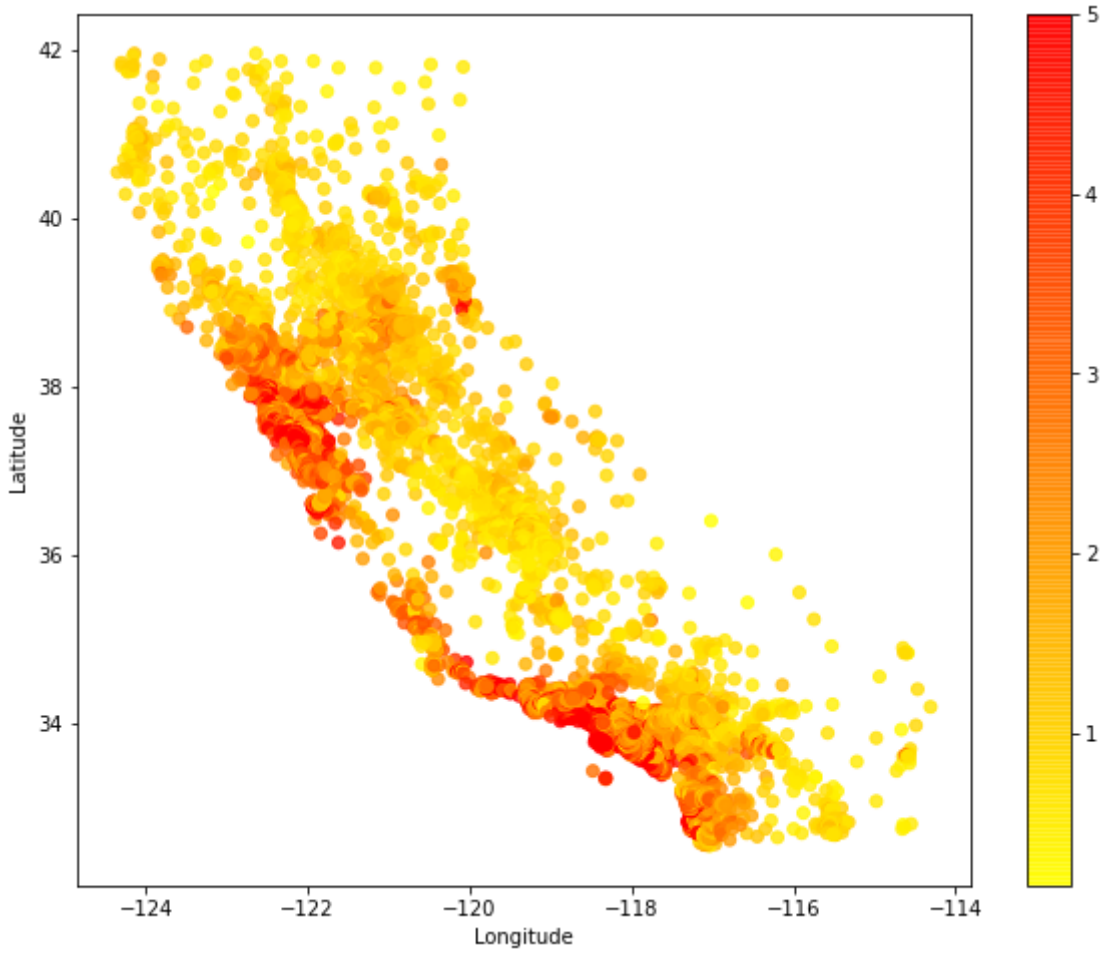


Seaborn


```
import seaborn as sns
ax = sns.scatterplot(data=housing_df, x="Longitude", y="Latitude", hue="MedianPrice")
norm = plt.Normalize(housing_df['MedianPrice'].min(), housing_df['MedianPrice'].max())
sm = plt.cm.ScalarMappable(cmap="RdBu", norm=norm)
ax.get_legend().remove()
ax.figure.colorbar(sm)
plt.show()
```

Matplotlib


```
plt.figure(figsize=(10, 8))
#plot the data with housing price as scale to color the points
plt.scatter(housing_df['Longitude'],housing_df['Latitude'],
            cmap = 'autumn_r', c = housing_df['MedianPrice'],
            alpha = 0.8)
plt.xlabel('Longitude')
plt.ylabel('Latitude')
plt.colorbar()
plt.show();
```



(e) Write a function to return the distance between two points where the function inputs are the latitude and longitude of the two points.

```
# import math

# def degree_conversion(value):
#   return value*111/6378.137
```

```
# def distance(lat1, lat2, log1, log2):
#   lat1 = degree_conversion(lat1)
#   lat2 = degree_conversion(lat2)
#   log1 = degree_conversion(log1)
#   log2 = degree_conversion(log2)
#   r =6378.137
#   distance = 2*r* np.arcsin(math.sqrt(math.sin((lat2-lat1)/2)**2 + math.cos(lat1)*ma
#   return distance

def distance(point1, point2):
  '''
  Calculate the distance between two points based on distance formula in https://en.wi

  Outputs:
    distance : The distance(km) between two points
  '''
  point1_lat, point1_long = point1
  point2_lat, point2_long = point2
  #radius of the Earth
  r = 6378.137
  #convert the degree into radians
  phi_1, phi_2 = radians(point1_lat), radians(point2_lat)
  lambda_1, lambda_2 = radians(point1_long),radians(point2_long)
  #compute the inside square root
  inside_sqrt = sin((phi_2-phi_1)/2)**2 + cos(phi_1)*cos(phi_2)*sin((lambda_2 - lambda
  #put all pieces together
  distance = 2 * r * np.arcsin(np.sqrt(inside_sqrt))
  return distance


#An examination: distance from LA to SF
distance((33.93, -118.40), (37.62, -122.38))
```

```
545.7177050542751
```

(f) For each city given in figure 1, compute the distance of the California districts from the city using your function in (e).

```
#create a dataframe that contains the logitude and latitude of cities given in figure
data = {'Lat':[33.93, 37.62, 37.37, 32.57, 34.43],
        'Lon':[-118.40, -122.38, -121.92, -116.98, -119.83]}
cities_degree = pd.DataFrame(data, index=['LA', 'SF', 'SJ', 'SD', 'SB'])
# cities_degree = pd.DataFrame(data)
cities_degree
```

	Lat	Lon
LA	33.93	−118.40
SF	37.62	−122.38
SJ	37.37	−121.92

```
type(housing_df)
```

```
pandas.core.frame.DataFrame
```

```
dis = dict({})
#use a for loop to generate a new dataframe
# for i in range(len(housing_df)):
#   for j in range(len(cities_degree)):
#     dis.append(distance(housing_df.loc[i,'Latitude'], cities_degree.loc[j, 'Lat'],
#                         housing_df.loc[i, 'Longitude'], cities_degree.loc[j, 'Lon']))

for city in cities_degree.index:
  #get the city point from cities_degree
  city_point = tuple(cities_degree.loc[city, ['Lat', 'Lon']])
  #print(city_point)
  #apply distance on city_point and housing_df to calculate the distances
  housing_city_dis = housing_df.apply(lambda point: distance(city_point, (point.Latitu
  #print(housing_city_dis)
  #organize housing_city_dis into list, dis
  dis['Dist_'+city] = housing_city_dis


dis = pd.DataFrame(dis)
dis
```

	Dist_LA	Dist_SF	Dist_SJ	Dist_SD	Dist_SB
0	558.991206	31.812202	63.009434	759.520447	440.444012

(g) Now that you have (f), add a column (with label 'CityProximity') to the data frame in part (b) where each number in the column represents the shortest distance of the corresponding district to the given cities in figure 1.

```
# shortest_dis = []
# temp = []
# while(dis!=[]):
#   for i in range(5):
#     temp.append(dis.pop(0))
#   shortest_dis.append(min(temp))


# shortest_dis[0:2]
```

```
[31.728894424216165, 30.124988471759767]
```

```
housing_df['CityProximity']=dis.min(axis=1)


housing_df.head()
```

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Lo
0	8.3252	41.0	6.984127	1.023810	322.0	2.555556	37.88	
1	8.3014	21.0	6.238137	0.971880	2401.0	2.109842	37.86	
2	7.2574	52.0	8.288136	1.073446	496.0	2.802260	37.85	
3	5.6431	52.0	5.817352	1.073059	558.0	2.547945	37.85	
4	3.8462	52.0	6.281853	1.081081	565.0	2.181467	37.85	

(h) From the data frame prepared in (g), create a new data frame where the median housing price is less than 5.0 and median house age is less than 52.0.

```
housing_df1 = housing_df[housing_df['MedianPrice']<5]
housing_df1 = housing_df[housing_df['HouseAge']<52]


len(housing_df1)
```

```
19367
```

(i) Split the data frame in (h) into a training and test dataset. Use the first 14000 rows as the training dataset and the rest as the test dataset. Split the target dataset in the same manner. (Note that the data frame in (h) has the target median housing price column which should be removed in the training set)

```
# X = housing_df1[{'AveBedrms', 'MedInc', 'HouseAge', 'AveRooms', 'Population',
                      #  'AveOccup', 'Latitude', 'Longitude', 'CityProximity'}]
X = housing_df1.drop('median_housing_px',axis=1)
X_train = X[:14000]
X_train.shape
```

```
(14000, 9)
```

```
X_test = X[14000:]
X_test.shape
```

```
(5367, 9)
```

```
Y = housing_df1['MedianPrice']
Y_train = Y[:14000]
Y_train.shape
```

```
(14000,)
```

```
Y_test = Y[14000:]
Y_test.shape
```

```
(5367,)
```

(j) Use RandomForestRegressor (with 20 estimators) in Scikit-Learn to fit a random forest on the training dataset. Use cross_val_score to compute the average of the root mean squared error (RMSE) on the 5-fold cross-validation.

A standard way to evaluate model accuracy on continuous data is to compare the mean squared error (MSE) of your candidate models.

```
from sklearn.ensemble import RandomForestRegressor
model = RandomForestRegressor(n_estimators=20)
model.fit(X_train, Y_train)
```

```
RandomForestRegressor(n_estimators=20)
```

```
from sklearn.model_selection import cross_val_score
```

```
cv_scores = cross_val_score(model, X_train, Y_train, cv=5, scoring='neg_root_mean_squa
cv_scores.mean()
```

```
-0.6392572036215326
```

(k) Use GridSearchCV (with 5-fold cross-validation) to fine tune the hyper-parameters of the Random Forest Regressor model. (You may choose the upper bound for n_estimators to be 50)

```
from sklearn.model_selection import GridSearchCV

param_grid = {'n_estimators':[10, 20, 30, 40, 50]}
gs = GridSearchCV(estimator=model, param_grid=param_grid, cv=5)
gs.fit(X_train, Y_train)
```

```
GridSearchCV(cv=5, estimator=RandomForestRegressor(n_estimators=20),
             param_grid={'n_estimators': [10, 20, 30, 40, 50]})
```

```
gs.best_params_
```

```
{'n_estimators': 50}
```

```
best_model = gs.best_estimator_
```

(l) Evaluate the RMSE of the fine-tuned model on the test dataset.

```
best_model.fit(X_test, Y_test)
```

```
RandomForestRegressor(n_estimators=50)
```

```
cv_scores = cross_val_score(best_model, X_test, Y_test, cv=5, scoring='neg_root_mean_s
cv_scores.mean()
```

```
-0.7483414371086954
```

✓ 0 秒 完成时间：16:56