housePricing-TP2

November 15, 2020

###

House Pricing

0.1 Step 1 : Reading Data

[3]:
```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
```

[4]:
```
# loading the data
df = pd.read_csv('datasets/housing.csv')
df.head()
```

[4]:
```
   longitude  latitude  housing_median_age  total_rooms  total_bedrooms  \
0    -122.23     37.88                41.0        880.0           129.0
1    -122.22     37.86                21.0       7099.0          1106.0
2    -122.24     37.85                52.0       1467.0           190.0
3    -122.25     37.85                52.0       1274.0           235.0
4    -122.25     37.85                52.0       1627.0           280.0

   population  households  median_income  median_house_value ocean_proximity
0       322.0       126.0         8.3252            452600.0        NEAR BAY
1      2401.0      1138.0         8.3014            358500.0        NEAR BAY
2       496.0       177.0         7.2574            352100.0        NEAR BAY
3       558.0       219.0         5.6431            341300.0        NEAR BAY
4       565.0       259.0         3.8462            342200.0        NEAR BAY
```

0.2 Step 2 : Data Preparation

[5]:
```
df['ocean_proximity'].value_counts()
```

[5]:
```
<1H OCEAN     9136
INLAND        6551
NEAR OCEAN    2658
```

```
NEAR BAY      2290
ISLAND           5
Name: ocean_proximity, dtype: int64
```

[6]:
```
ocean = pd.get_dummies(df.ocean_proximity)
ocean.columns = ['ocean_1','ocean_2','ocean_3','ocean_4','ocean_5']
df = df.drop("ocean_proximity",axis = 1)
full_df = pd.concat([df,ocean],axis = 1)
```

[7]:
```
full_df[['ocean_1','ocean_2','ocean_3','ocean_4','ocean_5']] =␣
↪full_df[['ocean_1','ocean_2','ocean_3','ocean_4','ocean_5']].astype(float)
```

[8]:
```
full_df.head()
```

[8]:
```
   longitude  latitude  housing_median_age  total_rooms  total_bedrooms  \
0    -122.23     37.88                41.0        880.0           129.0
1    -122.22     37.86                21.0       7099.0          1106.0
2    -122.24     37.85                52.0       1467.0           190.0
3    -122.25     37.85                52.0       1274.0           235.0
4    -122.25     37.85                52.0       1627.0           280.0

   population  households  median_income  median_house_value  ocean_1  \
0       322.0       126.0         8.3252            452600.0      0.0
1      2401.0      1138.0         8.3014            358500.0      0.0
2       496.0       177.0         7.2574            352100.0      0.0
3       558.0       219.0         5.6431            341300.0      0.0
4       565.0       259.0         3.8462            342200.0      0.0

   ocean_2  ocean_3  ocean_4  ocean_5
0      0.0      0.0      1.0      0.0
1      0.0      0.0      1.0      0.0
2      0.0      0.0      1.0      0.0
3      0.0      0.0      1.0      0.0
4      0.0      0.0      1.0      0.0
```

[ ]:

0.2.1 here we splitted our data to train data and test data

[9]:
```
train_df = full_df[:16512]
test_df = full_df[16512:]
```

0.2.2 here we we'll see how many missing rows do we have

[10]:
```
train_df.isna().sum()
```

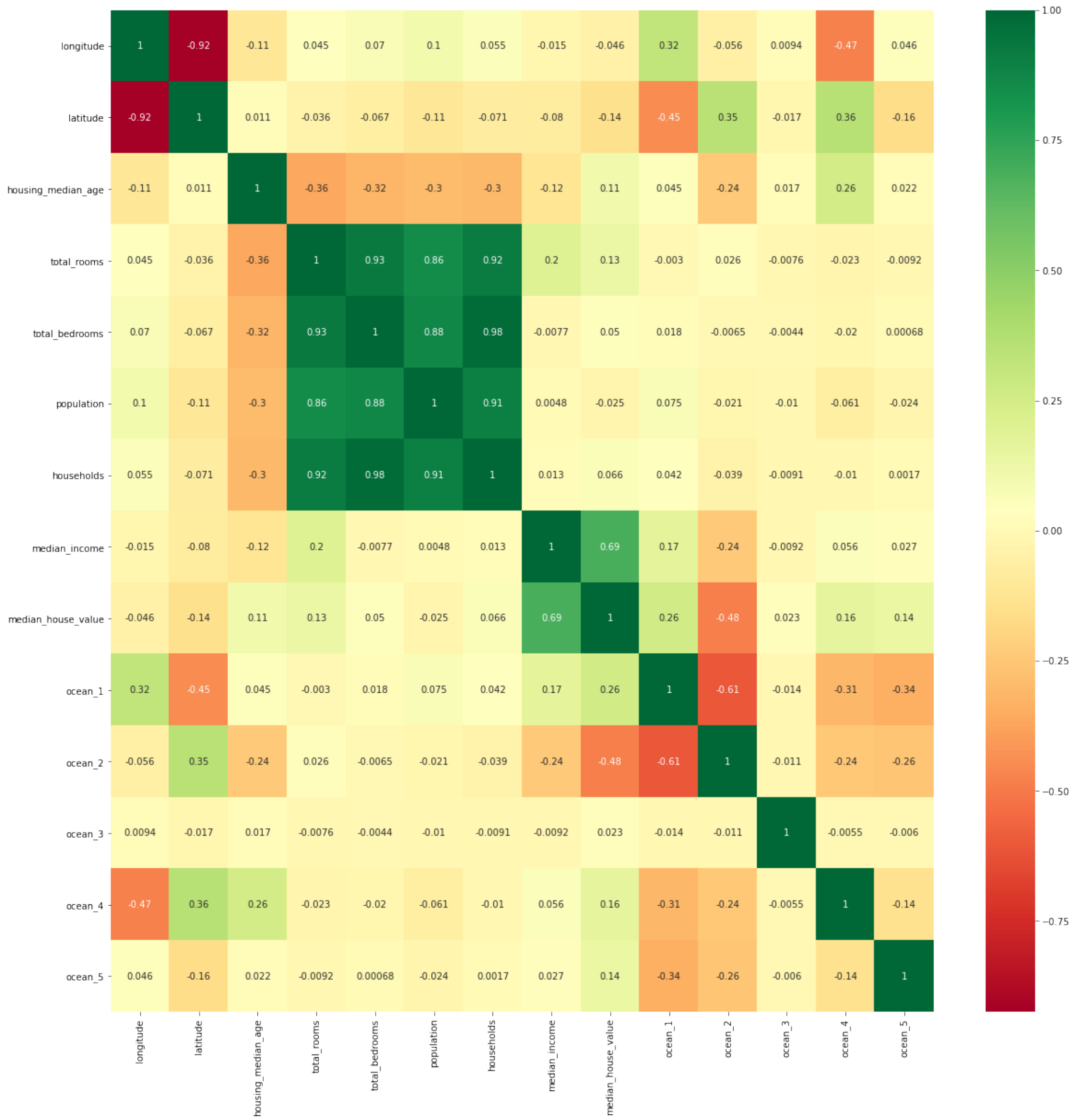
[10]:
```
longitude              0
latitude               0
housing_median_age     0
total_rooms            0
total_bedrooms       159
population             0
households             0
median_income          0
median_house_value     0
ocean_1                0
ocean_2                0
ocean_3                0
ocean_4                0
ocean_5                0
dtype: int64
```

0.2.3 we see here that we have 159 nan samples in totoal_bedrooms column , the number of unknow values is not that big compared to the total number of samples

0.2.4 therefore we'll see its correlation with the output (all features correlations we'll be shown in a heat map matrix

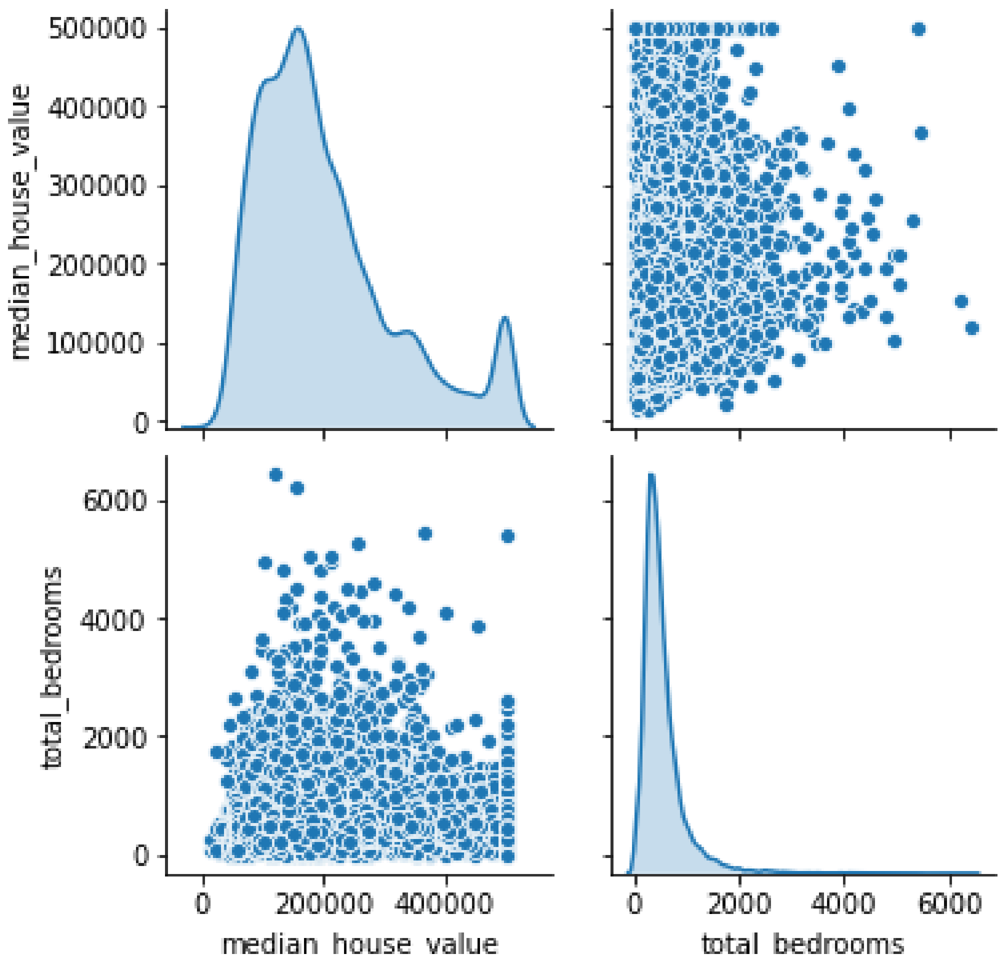
and we decide what we'll do later

[11]:
```
corrmat = full_df.corr()
top_corr_features = corrmat.index
plt.figure(figsize=(20,20))
#plot heat map
g=sns.heatmap(full_df[top_corr_features].corr(),annot=True,cmap="RdYlGn")
```



0.2.5 the correlation between the median__house__value and the total__bedrooms plotted below

[12]:
```
cols = ['median_house_value','total_bedrooms']
sns.pairplot(train_df[cols],diag_kind="kde")
plt.show()
```



since total_bedrooms has a weak correlation with the output variable it is better to delete it

we'll delete also some other non correlated features

[13]:
```
full_df = full_df.
↪drop(['ocean_1','ocean_2','ocean_3','ocean_4','ocean_5','total_bedrooms'],␣
↪axis = 1)
```

[14]:
```
full_df.columns
```

[14]:
```
Index(['longitude', 'latitude', 'housing_median_age', 'total_rooms',
       'population', 'households', 'median_income', 'median_house_value'],
      dtype='object')
```

[15]:
```
train_df = full_df[:16512]
test_df = full_df[16512:]
```

[16]:
```
test_df.shape
```

[16]: (4128, 8)

[ ]:
```

```

[17]:
```
# extraction of X_train (n_samples, n_features) and y_train (target variable)
X_train = train_df.drop("median_house_value", axis=1)
y_train = train_df["median_house_value"].to_numpy()
print('X_train:', X_train.shape, '; y_train:', np.shape(y_train))
```

```
X_train: (16512, 7) ; y_train: (16512,)
```

[18]:
```
# loading the training data
# extraction of X_test and y_test
X_test= test_df.drop("median_house_value", axis=1)
y_test = test_df["median_house_value"].to_numpy()
print('X_test:', X_test.shape, '; y_test:', np.shape(y_test))
```

```
X_test: (4128, 7) ; y_test: (4128,)
```

0.3 Step 3 : Building and testing Model

0.4 Here we'll try a bunch of models on our data using a pipeline contains GridSearchCV which will make it easy for us to train each model on several parameters and then get the best of them

[25]:
```
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
from sklearn.linear_model import ElasticNet
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import GradientBoostingRegressor
```

0.4.1 Note : you can get all parameters of each algorithm with the cell below

[26]:
```
# here we can see how many paremters we can deal with iin each algorithm
#print(LinearRegression().get_params())
#print(Ridge().get_params())
#print(Lasso().get_params())
```

```
#print(ElasticNet().get_params())
#print(DecisionTreeRegressor().get_params())
#print(RandomForestRegressor().get_params())
```

[34]:
```
# K-fold cross-validation and GridSearchCV
pipelines = []
params = []
names = []

#
# add LinearRegression
pipelines.append(Pipeline([('clf', LinearRegression())])) ### LinearRegression
params.append({'clf__normalize':[True]})
names.append('LinearRegression')
# add Ridge regression
pipelines.append(Pipeline([('clf', Ridge())])) ### LinearRegression
params.append({'clf__alpha':[0.1,1,10]})
names.append('Ridge regression')
# add Lasso regression
pipelines.append(Pipeline([('clf', Lasso())])) ### LinearRegression
params.append({'clf__alpha':[0.1,1,10],'clf__selection' : ['cyclic', 'random']})
names.append('Lasso regression')

# add ElasticNet regression
pipelines.append(Pipeline([('clf', ElasticNet())])) ### LinearRegression
params.append({'clf__l1_ratio':[0,0.5,1]})
names.append('ElasticNet regression')

# add DecisionTreeRegressor
pipelines.append(Pipeline([('clf', DecisionTreeRegressor())])) ##␣
↪DecisionTreeRegressor
params.append({'clf__max_depth':np.linspace(5, 15, 5)})
names.append('DecisionTreeRegressor')

# add  RandomForestRegressor
pipelines.append(Pipeline([('clf', RandomForestRegressor())])) ##␣
↪RandomForestRegressor
params.append({'clf__n_estimators': [100,200]})
names.append('RandomForestRegressor')
# add GradientBoostingRegressor
pipelines.append(Pipeline([('clf', GradientBoostingRegressor())])) ##␣
↪RandomForestRegressor
params.append({'clf__loss': ['ls', 'lad', 'huber',␣
↪'quantile'],'clf__learning_rate':[0.1,0.01,0.001]})
names.append('GradientBoostingRegressor')
```

[35]:
```
from sklearn.model_selection import KFold, GridSearchCV, cross_val_score

def model(pipeline, parameters, name, X, y):
    cv = KFold(n_splits=5, shuffle=True, random_state=32)
    grid_obj = GridSearchCV(estimator=pipeline, param_grid=parameters, cv=cv,␣
 ↪scoring='r2', n_jobs=-1)
    grid_obj.fit(X,y)
    print(name, 'R2:', grid_obj.best_score_)
    print(name, 'best parameters:', grid_obj.best_params_)
    estimator = grid_obj.best_estimator_
    estimator.fit(X,y) # training sur tout training dataset
    return estimator
estimators = []
for i in range(len(pipelines)):
    estimators.append(model(pipelines[i], params[i], names[i], X_train,␣
 ↪y_train))
```

```
LinearRegression R2: 0.6095220945555696
LinearRegression best parameters: {'clf__normalize': True}
Ridge regression R2: 0.6095233381077149
Ridge regression best parameters: {'clf__alpha': 10}
Lasso regression R2: 0.6095225522208908
Lasso regression best parameters: {'clf__alpha': 10, 'clf__selection': 'random'}
ElasticNet regression R2: 0.6095221461148423
ElasticNet regression best parameters: {'clf__l1_ratio': 1}
DecisionTreeRegressor R2: 0.7325740042740059
DecisionTreeRegressor best parameters: {'clf__max_depth': 10.0}
RandomForestRegressor R2: 0.8300119482727247
RandomForestRegressor best parameters: {'clf__n_estimators': 200}
GradientBoostingRegressor R2: 0.7808791283670714
GradientBoostingRegressor best parameters: {'clf__learning_rate': 0.1,
'clf__loss': 'ls'}
```

0.4.2 Seems like Ransom forest Regressor is the best model that performs well on training data but we need to see the model performance on test data to judge which is the best model in our case

[36]:
```
from sklearn.metrics import mean_squared_error, r2_score

# extraction of X_test and y_test
X_test= test_df.drop("median_house_value", axis=1)
y_test = test_df["median_house_value"].to_numpy()
print('X_test:', X_test.shape, '; y_test:', np.shape(y_test))

# Evaluation
```

```
for i, estimator in enumerate(estimators):
    print('\nPerformance :', names[i])
    y_pred = estimator.predict(X_test)
    #print('\n mean_squared_error :', mean_squared_error(y_test, y_pred))
    print('\n r2_score :', r2_score(y_test, y_pred))
```

```
X_test: (4128, 7) ; y_test: (4128,)

Performance : LinearRegression

 r2_score : 0.6921662617541259

Performance : Ridge regression

 r2_score : 0.6921214717405941

Performance : Lasso regression

 r2_score : 0.6921489611912213

Performance : ElasticNet regression

 r2_score : 0.6921645514650492

Performance : DecisionTreeRegressor

 r2_score : 0.4761774870204921

Performance : RandomForestRegressor

 r2_score : 0.608204499045217

Performance : GradientBoostingRegressor

 r2_score : 0.6335292522447636
```

0.4.3 lasso regression might be the best choice here because it has the lowest overfitting , note that a model overfits when it perofrms well on train data and it has bad performance on test data

0.4.4 in the cell below we'll save each of our models for future uses

[37]:
```
import pickle
# save the classifier
for i, estimator in enumerate(estimators):
    with open(names[i]+".pkl", 'wb') as fid:
```

```
        pickle.dump(estimator, fid)
```

[ ]:

```

```