Linear Regression on US Housing Price

Linear regression primer

In statistics, linear regression is a linear approach for modeling the relationship between a scalar dependent variable y and one or more explanatory variables (or independent variables) denoted X. The case of one explanatory variable is called simple linear regression. For more than one explanatory variable, the process is called multiple linear regression.

Linear regression models are often fitted using the least squares approach, but they may also be fitted in other ways, such as by minimizing the "lack of fit" in some other norm (as with least absolute deviations regression), or by minimizing a penalized version of the least squares loss function as in ridge regression (L2-norm penalty) and lasso (L1-norm penalty). Conversely, the least squares approach can be used to fit models that are not linear models. Thus, although the terms "least squares" and "linear model" are closely linked, they are not synonymous.

1.Import packages and dataset

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

```
df = pd.read_csv("/content/sample_data/california_housing_train.csv")
df.head()
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value
0	-114.31	34.19	15.0	5612.0	1283.0	1015.0	472.0	1.4936	66900.0
1	-114.47	34.40	19.0	7650.0	1901.0	1129.0	463.0	1.8200	80100.0
2	-114.56	33.69	17.0	720.0	174.0	333.0	117.0	1.6509	85700.0
3	-114.57	33.64	14.0	1501.0	337.0	515.0	226.0	3.1917	73400.0
4	-114.57	33.57	20.0	1454.0	326.0	624.0	262.0	1.9250	65500.0

Next steps: Generate code with df | View recommended plots | New interactive sheet

Check basic info on the data set 'info()' method to check the data types and number

```
df.info(verbose=True)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 17000 entries, 0 to 16999
Data columns (total 9 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   longitude           17000 non-null  float64
 1   latitude            17000 non-null  float64
 2   housing_median_age  17000 non-null  float64
 3   total_rooms         17000 non-null  float64
```

```
 4   total_bedrooms      17000 non-null  float64
 5   population          17000 non-null  float64
 6   households          17000 non-null  float64
 7   median_income       17000 non-null  float64
 8   median_house_value  17000 non-null  float64
dtypes: float64(9)
memory usage: 1.2 MB
```

'describe()' method to get the statistical summary of the various features of the data set

```
df.describe(percentiles=[0.1,0.25,0.5,0.75,0.9])
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value
count	17000.000000	17000.000000	17000.000000	17000.000000	17000.000000	17000.000000	17000.000000	17000.000000	17000.000000
mean	-119.562108	35.625225	28.589353	2643.664412	539.410824	1429.573941	501.221941	3.883578	207300.912353
std	2.005166	2.137340	12.586937	2179.947071	421.499452	1147.852959	384.520841	1.908157	115983.764387
min	-124.350000	32.540000	1.000000	2.000000	1.000000	3.000000	1.000000	0.499900	14999.000000
10%	-122.280000	33.620000	12.000000	949.000000	199.000000	514.000000	185.000000	1.910700	81900.000000
25%	-121.790000	33.930000	18.000000	1462.000000	297.000000	790.000000	282.000000	2.566375	119400.000000
50%	-118.490000	34.250000	29.000000	2127.000000	434.000000	1167.000000	409.000000	3.544600	180400.000000
75%	-118.000000	37.720000	37.000000	3151.250000	648.250000	1721.000000	605.250000	4.767000	265000.000000
90%	-117.240000	38.480000	46.000000	4677.100000	968.000000	2578.200000	893.000000	6.194900	379600.000000
max	-114.310000	41.950000	52.000000	37937.000000	6445.000000	35682.000000	6082.000000	15.000100	500001.000000

'columns' method to get the names of the columns (features)

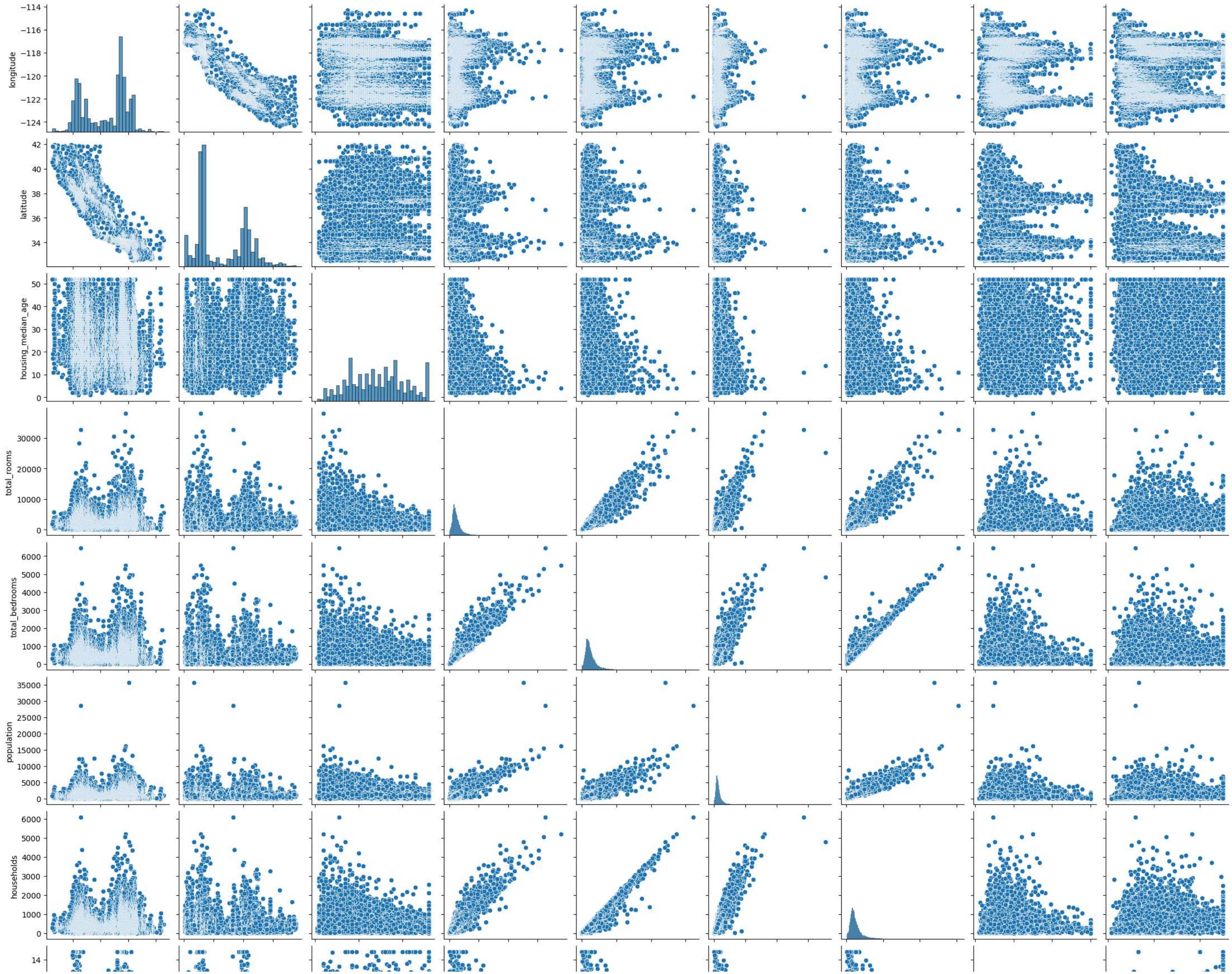
```
df.columns
```

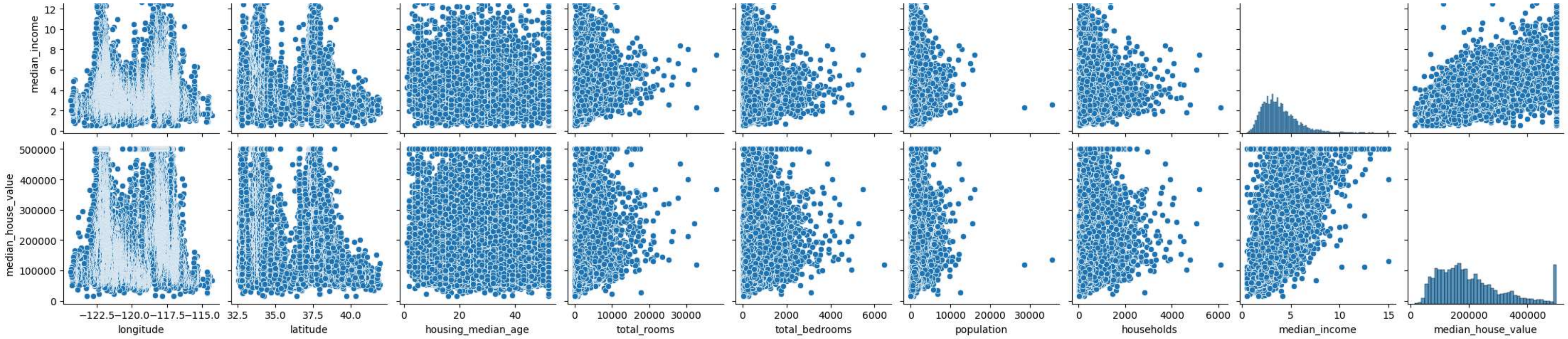
```
Index(['longitude', 'latitude', 'housing_median_age', 'total_rooms',
       'total_bedrooms', 'population', 'households', 'median_income',
       'median_house_value'],
      dtype='object')
```

Basic plotting and visualization on the data set Pairplots using seaborn creates a grid of scatter plots and histograms (or KDEs) that display pairwise relationships between variables in a DataFrame

```
sns.pairplot(df)
```

<seaborn.axisgrid.PairGrid at 0x7f3d38a653f0>

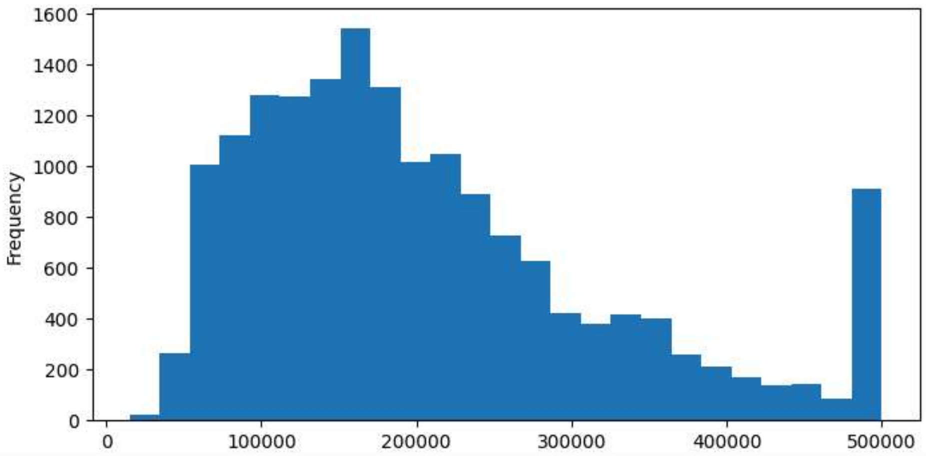



median_income
median_house_value
longitude
latitude
housing_median_age
total_rooms
total_bedrooms
population
households
median_income
median_house_value

Distribution of price (the predicted quantity)

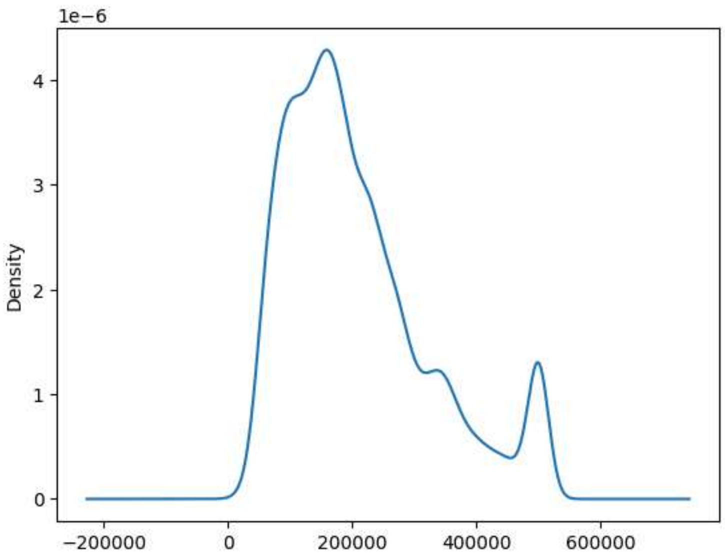
```
df['median_house_value'].plot.hist(bins=25,figsize=(8,4))
```

```
<Axes: ylabel='Frequency'>
```




```
df['median_house_value'].plot.density()
```

```
<Axes: ylabel='Density'>
```



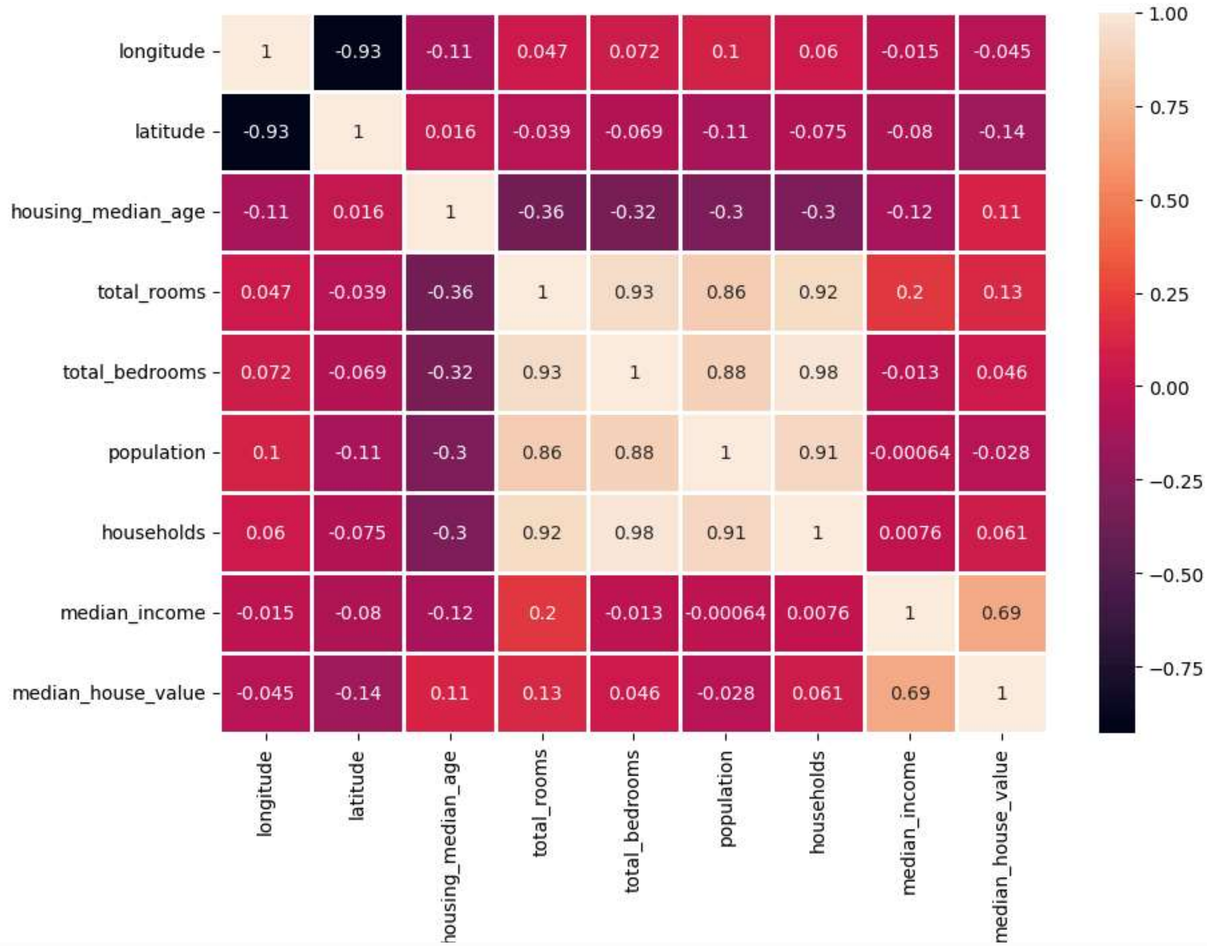
Correlation matrix and heatmap

```
df.corr()
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value
longitude	1.000000	-0.925208	-0.114250	0.047010	0.071802	0.101674	0.059628	-0.015485	-0.044982
latitude	-0.925208	1.000000	0.016454	-0.038773	-0.069373	-0.111261	-0.074902	-0.080303	-0.144917
housing_median_age	-0.114250	0.016454	1.000000	-0.360984	-0.320434	-0.295890	-0.302754	-0.115932	0.106758
total_rooms	0.047010	-0.038773	-0.360984	1.000000	0.928403	0.860170	0.919018	0.195383	0.130991
total_bedrooms	0.071802	-0.069373	-0.320434	0.928403	1.000000	0.881169	0.980920	-0.013495	0.045783
population	0.101674	-0.111261	-0.295890	0.860170	0.881169	1.000000	0.909247	-0.000638	-0.027850
households	0.059628	-0.074902	-0.302754	0.919018	0.980920	0.909247	1.000000	0.007644	0.061031
median_income	-0.015485	-0.080303	-0.115932	0.195383	-0.013495	-0.000638	0.007644	1.000000	0.691871
median_house_value	-0.044982	-0.144917	0.106758	0.130991	0.045783	-0.027850	0.061031	0.691871	1.000000

```
plt.figure(figsize=(10,7))
sns.heatmap(df.corr(),annot=True,linewidths=2)
```

```
<Axes: >
```



Feature and variable sets Make a list of data frame column names

```
l_column = list(df.columns)
len_feature = len(l_column)
l_column
```

```
['longitude',
 'latitude',
 'housing_median_age',
 'total_rooms',
 'total_bedrooms',
 'population',
 'households',
 'median_income',
 'median_house_value']
```

Put all the numerical features in X and Price in y, ignore Address which is string for linear regression

```
X = df[l_column[2:len_feature]]
y = df[l_column[len_feature-1]]

print("Feature set size:",X.shape)
print("Variable set size:",y.shape)
y.head()
```

```
Feature set size: (17000, 7)
Variable set size: (17000,)
```

	median_house_value
0	66900.0
1	80100.0
2	85700.0
3	73400.0
4	65500.0

dtype: float64

```
X.head()
```

	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value
0	15.0	5612.0	1283.0	1015.0	472.0	1.4936	66900.0
1	19.0	7650.0	1901.0	1129.0	463.0	1.8200	80100.0
2	17.0	720.0	174.0	333.0	117.0	1.6509	85700.0
3	14.0	1501.0	337.0	515.0	226.0	3.1917	73400.0
4	20.0	1454.0	326.0	624.0	262.0	1.9250	65500.0

Next steps: Generate code with X | View recommended plots | New interactive sheet

```
y.head()
```

	median_house_value
0	66900.0
1	80100.0
2	85700.0
3	73400.0
4	65500.0

dtype: float64

Test-train split Import train_test_split function from scikit-learn

```
from sklearn.model_selection import train_test_split
```

Create X and y train and test splits in one command using a split ratio and a random seed

```
X_train, X_test, y_train, y_test = train_test_split(X, y,
                            test_size=0.2, random_state=123)
```

```
X_train.shape, X_test.shape
```

```
((13600, 7), (3400, 7))
```

Check the size and shape of train/test splits (it should be in the ratio as per test_size parameter above)

```
print("Training feature set size:",X_train.shape)
print("Test feature set size:",X_test.shape)
print("Training variable set size:",y_train.shape)
print("Test variable set size:",y_test.shape)
```

```
Training feature set size: (13600, 7)
Test feature set size: (3400, 7)
Training variable set size: (13600,)
Test variable set size: (3400,)
```

Model fit and training Import linear regression model estimator from scikit-learn and instantiate

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

```
lm = LinearRegression()
```

Fit the model on to the instantiated object itself

```
lm.fit(X_train,y_train)
```

```
LinearRegression
LinearRegression()
```

```
print("The intercept term of the linear model:", lm.intercept_)
```

```
The intercept term of the linear model: -2.6193447411060333e-10
```

```
print("The coefficients of the linear model:", lm.coef_)
```

```
The coefficients of the linear model: [ 6.95662210e-13 -1.67643677e-14  1.81707080e-13 -7.46278039e-15
 -4.85739921e-14  2.26888723e-11  1.00000000e+00]
```

```
cdf = pd.DataFrame(data=lm.coef_, index=X_train.columns, columns=["Coefficients"])
cdf
```

	Coefficients
housing_median_age	6.956622e-13
total_rooms	-1.676437e-14
total_bedrooms	1.817071e-13
population	-7.462780e-15
households	-4.857399e-14
median_income	2.268887e-11
median_house_value	1.000000e+00

Next steps: Generate code with `cdf` View recommended plots New interactive sheet

Calculation of standard errors and t-statistic for the coefficients

```
y_train.shape
```

```
(13600,)
```

```
n=X_train.shape[0]
k=X_train.shape[1]
dfN = n-k
train_pred=lm.predict(X_train)
train_error = np.square(train_pred - y_train)
sum_error=np.sum(train_error)
se=[0,0,0,0,0,0,0]
for i in range(k):
    r = (sum_error/dfN)
    r = r/np.sum(np.square(X_train[
        list(X_train.columns)[i]]-X_train[list(X_train.columns)[i]].mean()))
    se[i]=np.sqrt(r)
cdf['Standard Error']=se
cdf['t-statistic']=cdf['Coefficients']/cdf['Standard Error']
cdf
```

	Coefficients	Standard Error	t-statistic
housing_median_age	6.956622e-13	7.699549e-14	9.035104e+00
total_rooms	-1.676437e-14	4.473066e-16	-3.747847e+01
total_bedrooms	1.817071e-13	2.321366e-15	7.827594e+01
population	-7.462780e-15	8.878825e-16	-8.405144e+00
households	-4.857399e-14	2.550202e-15	-1.904712e+01
median_income	2.268887e-11	5.103229e-13	4.445984e+01
median_house_value	1.000000e+00	8.407339e-18	1.189437e+17

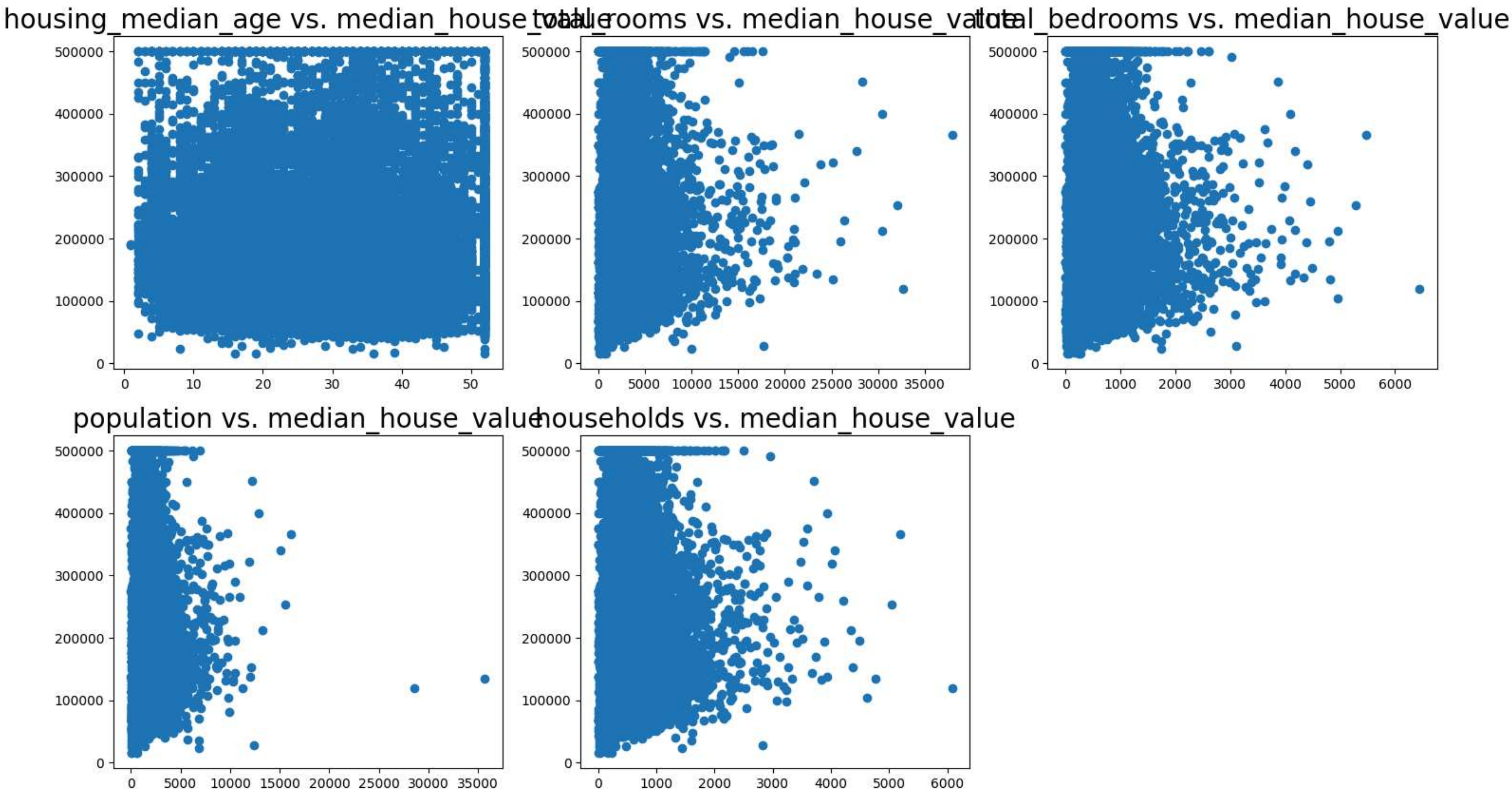
Next steps: Generate code with cdf | View recommended plots | New interactive sheet

```
print("Therefore, features arranged in the order of importance for predicting the house price\n",'-'*90,sep='')
l=list(cdf.sort_values('t-statistic',ascending=False).index)
print(' > \n'.join(l))
```

```
Therefore, features arranged in the order of importance for predicting the house price
------------------------------------------------------------------------------------------
median_house_value >
total_bedrooms >
median_income >
housing_median_age >
population >
households >
total_rooms
```

```
l=list(cdf.index)
from matplotlib import gridspec
fig = plt.figure(figsize=(18, 10))
gs = gridspec.GridSpec(2,3)
ax0 = plt.subplot(gs[0])
ax0.scatter(df[l[0]],df['median_house_value'])
ax0.set_title(l[0]+" vs. median_house_value", fontdict={'fontsize':20})

ax1 = plt.subplot(gs[1])
ax1.scatter(df[l[1]],df['median_house_value'])
ax1.set_title(l[1]+" vs. median_house_value",fontdict={'fontsize':20})

ax2 = plt.subplot(gs[2])
ax2.scatter(df[l[2]],df['median_house_value'])
ax2.set_title(l[2]+" vs. median_house_value",fontdict={'fontsize':20})

ax3 = plt.subplot(gs[3])
ax3.scatter(df[l[3]],df['median_house_value'])
ax3.set_title(l[3]+" vs. median_house_value",fontdict={'fontsize':20})

ax4 = plt.subplot(gs[4])
ax4.scatter(df[l[4]],df['median_house_value'])
ax4.set_title(l[4]+" vs. median_house_value",fontdict={'fontsize':20})
```

```
Text(0.5, 1.0, 'households vs. median_house_value')
```



R-square of the model fit

```
print("R-squared value of this fit:",round(metrics.r2_score(y_train,train_pred),3))
```

```
R-squared value of this fit: 1.0
```

Prediction, error estimate, and regression evaluation matrices Prediction using the lm model

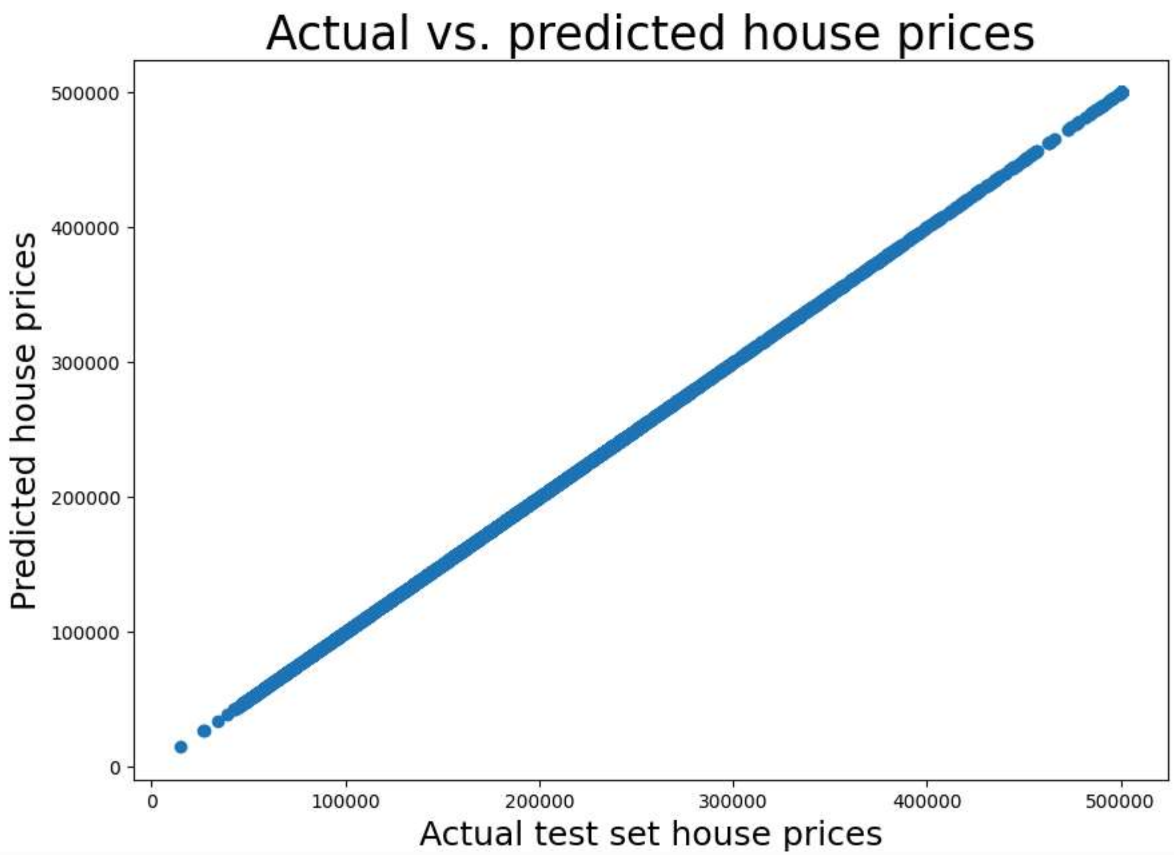
```
predictions = lm.predict(X_test)
print ("Type of the predicted object:", type(predictions))
print ("Size of the predicted object:", predictions.shape)
```

```
Type of the predicted object: <class 'numpy.ndarray'>
Size of the predicted object: (3400,)
```

Scatter plot of predicted price and y_test set to see if the data fall on a 45 degree straight line

```
plt.figure(figsize=(10,7))
plt.title("Actual vs. predicted house prices",fontsize=25)
plt.xlabel("Actual test set house prices",fontsize=18)
plt.ylabel("Predicted house prices", fontsize=18)
plt.scatter(x=y_test,y=predictions)
```

```
<matplotlib.collections.PathCollection at 0x7f3d7c49fc70>
```

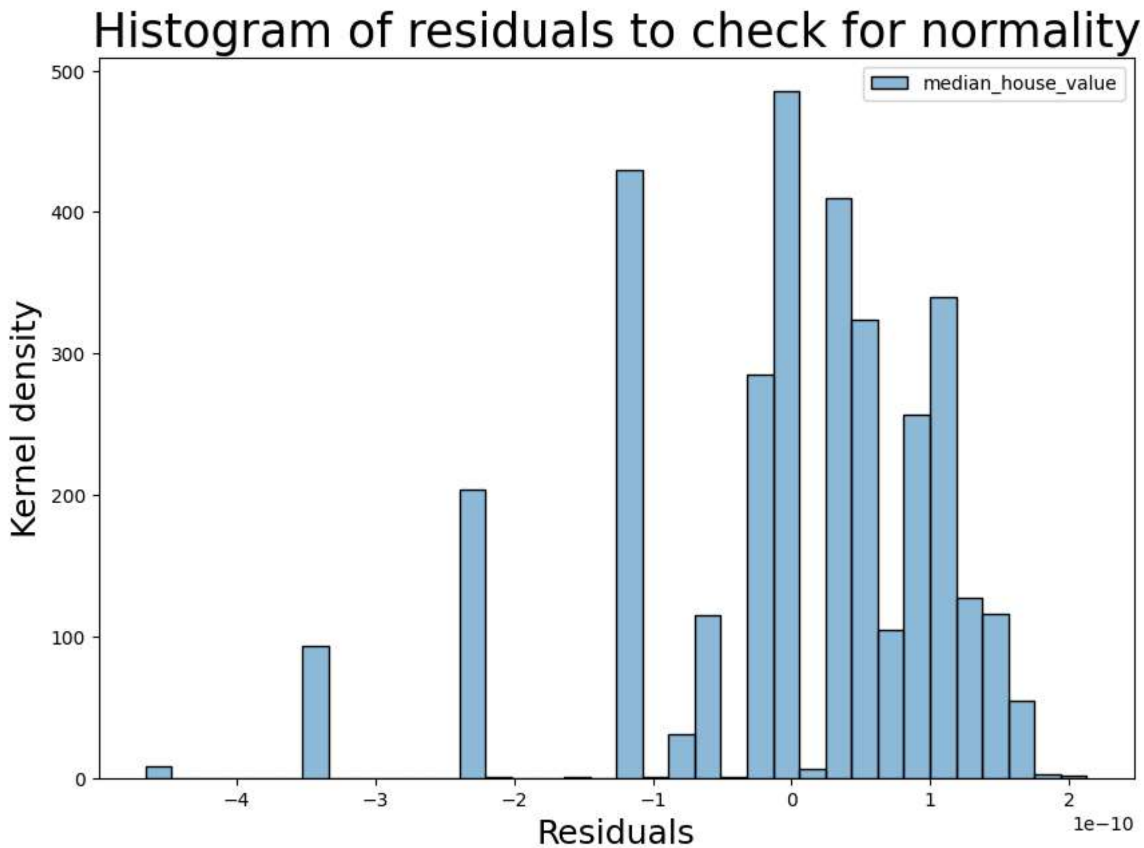


Plotting histogram of the residuals i.e. predicted errors (expect a normally distributed pattern)

```
plt.figure(figsize=(10,7))
plt.title("Histogram of residuals to check for normality",fontsize=25)
```

```
plt.xlabel("Residuals",fontsize=18)
plt.ylabel("Kernel density", fontsize=18)
sns.histplot([y_test-predictions])
```

```
<Axes: title={'center': 'Histogram of residuals to check for normality'}, xlabel='Residuals', ylabel='Kernel density'>
```



Scatter plot of residuals and predicted values (Homoscedasticity)

```
plt.figure(figsize=(10,7))
plt.title("Residuals vs. predicted values plot (Homoscedasticity)\n",fontsize=25)
plt.xlabel("Predicted house prices",fontsize=18)
plt.ylabel("Residuals", fontsize=18)
plt.scatter(x=predictions,y=y_test-predictions)
```