# Data Science

## Final Project No 7

## House Sales in King Country, USA

In this project, I have to perform as Data Analyst working in a Real Estate Investment Trust. They'd like starting investing in Residential real estate. I'll determine the market price of a house given a set of features and predict housing prices

Élaboré par :
**Smide ALEXIS**

Encadré par :
**Coursera Staff**

Data Science with Python

15 Janvier 2024

---

## Importation des packages

```
import pandas as pd
import seaborn as sbn
import numpy as np
import scipy as sc
import scipy.stats as stats
import matplotlib.pyplot as plt
import pipeline
import tqdm
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.linear_model import LinearRegression, Ridge
from sklearn.preprocessing import StandardScaler, PolynomialFeatures
from sklearn.pipeline import Pipeline
from sklearn.model_selection import train_test_split, cross_val_score, GridSearchCV
import warnings
warnings.filterwarnings("ignore", category=UserWarning)
```

## Importation du dataset

```
filepath=(
  'https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSkill'
  'sNetwork-DA0101EN-SkillsNetwork/labs/FinalModule_Coursera/data/kc_house_data_NaN.csv'
  )
df = pd.read_csv(filepath, header=0)
df.head()
```

```
   Unnamed: 0          id             date  ...     long  sqft_living15  sqft_lot15
0           0  7129300520  20141013T000000  ... -122.257           1340        5650
1           1  6414100192  20141209T000000  ... -122.319           1690        7639
2           2  5631500400  20150225T000000  ... -122.233           2720        8062
3           3  2487200875  20141209T000000  ... -122.393           1360        5000
4           4  1954400510  20150218T000000  ... -122.045           1800        7503

[5 rows x 22 columns]
```

```
df.tail()
```

```
       Unnamed: 0          id  ... sqft_living15  sqft_lot15
21608       21608   263000018  ...          1530        1509
21609       21609  6600060120  ...          1830        7200
21610       21610  1523300141  ...          1020        2007
21611       21611   291310100  ...          1410        1287
21612       21612  1523300157  ...          1020        1357

[5 rows x 22 columns]
```

Display the data types of each column using the function dtypes.

```
df.dtypes
```

```
Unnamed: 0         int64
id                 int64
date              object
price            float64
bedrooms         float64
bathrooms        float64
sqft_living        int64
sqft_lot           int64
floors           float64
waterfront         int64
view               int64
condition          int64
grade              int64
sqft_above         int64
sqft_basement      int64
yr_built           int64
yr_renovated       int64
zipcode            int64
lat              float64
long             float64
sqft_living15      int64
sqft_lot15         int64
dtype: object
```

We use the method describe to obtain a statistical summary of the dataframe.

```
df.describe()
```

```
        Unnamed: 0            id  ...  sqft_living15     sqft_lot15
count  21613.00000  2.161300e+04  ...   21613.000000   21613.000000
mean   10806.00000  4.580302e+09  ...    1986.552492   12768.455652
std     6239.28002  2.876566e+09  ...     685.391304   27304.179631
min        0.00000  1.000102e+06  ...     399.000000     651.000000
25%     5403.00000  2.123049e+09  ...    1490.000000    5100.000000
50%    10806.00000  3.904930e+09  ...    1840.000000    7620.000000
75%    16209.00000  7.308900e+09  ...    2360.000000   10083.000000
max    21612.00000  9.900000e+09  ...    6210.000000  871200.000000

[8 rows x 21 columns]
```

Drop the columns "id" and "Unnamed : 0" from axis 1 using the method drop(), then use the method describe() to obtain a statistical summary of the data. Make sure the inplace parameter is set to True.

```
df.drop(["id","Unnamed: 0"], axis=1, inplace=True)
df.describe()
```

```
              price      bedrooms  ...  sqft_living15     sqft_lot15
count  2.161300e+04  21600.000000  ...   21613.000000   21613.000000
mean   5.400881e+05      3.372870  ...    1986.552492   12768.455652
```

```
std    3.671272e+05      0.926657  ...     685.391304   27304.179631
min    7.500000e+04      1.000000  ...     399.000000     651.000000
25%    3.219500e+05      3.000000  ...    1490.000000    5100.000000
50%    4.500000e+05      3.000000  ...    1840.000000    7620.000000
75%    6.450000e+05      4.000000  ...    2360.000000   10083.000000
max    7.700000e+06     33.000000  ...    6210.000000  871200.000000

[8 rows x 19 columns]
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21613 entries, 0 to 21612
Data columns (total 20 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   date           21613 non-null  object
 1   price          21613 non-null  float64
 2   bedrooms       21600 non-null  float64
 3   bathrooms      21603 non-null  float64
 4   sqft_living    21613 non-null  int64
 5   sqft_lot       21613 non-null  int64
 6   floors         21613 non-null  float64
 7   waterfront     21613 non-null  int64
 8   view           21613 non-null  int64
 9   condition      21613 non-null  int64
 10  grade          21613 non-null  int64
 11  sqft_above     21613 non-null  int64
 12  sqft_basement  21613 non-null  int64
 13  yr_built       21613 non-null  int64
 14  yr_renovated   21613 non-null  int64
 15  zipcode        21613 non-null  int64
 16  lat            21613 non-null  float64
 17  long           21613 non-null  float64
 18  sqft_living15  21613 non-null  int64
 19  sqft_lot15     21613 non-null  int64
dtypes: float64(6), int64(13), object(1)
memory usage: 3.3+ MB
```

We can see we have missing values for the columns bedrooms and bathrooms

```
print("number of NaN values for the column bedrooms:", df['bedrooms'].isnull().sum())
```

```
number of NaN values for the column bedrooms: 13
```

```
print("number of NaN values for the column bathrooms:", df['bathrooms'].isnull().sum())
```

```
number of NaN values for the column bathrooms: 10
```

```
mean=df['bedrooms'].mean()
df['bedrooms'].replace(np.nan,mean, inplace=True)

mean=df['bathrooms'].mean()
df['bathrooms'].replace(np.nan,mean, inplace=True)
```

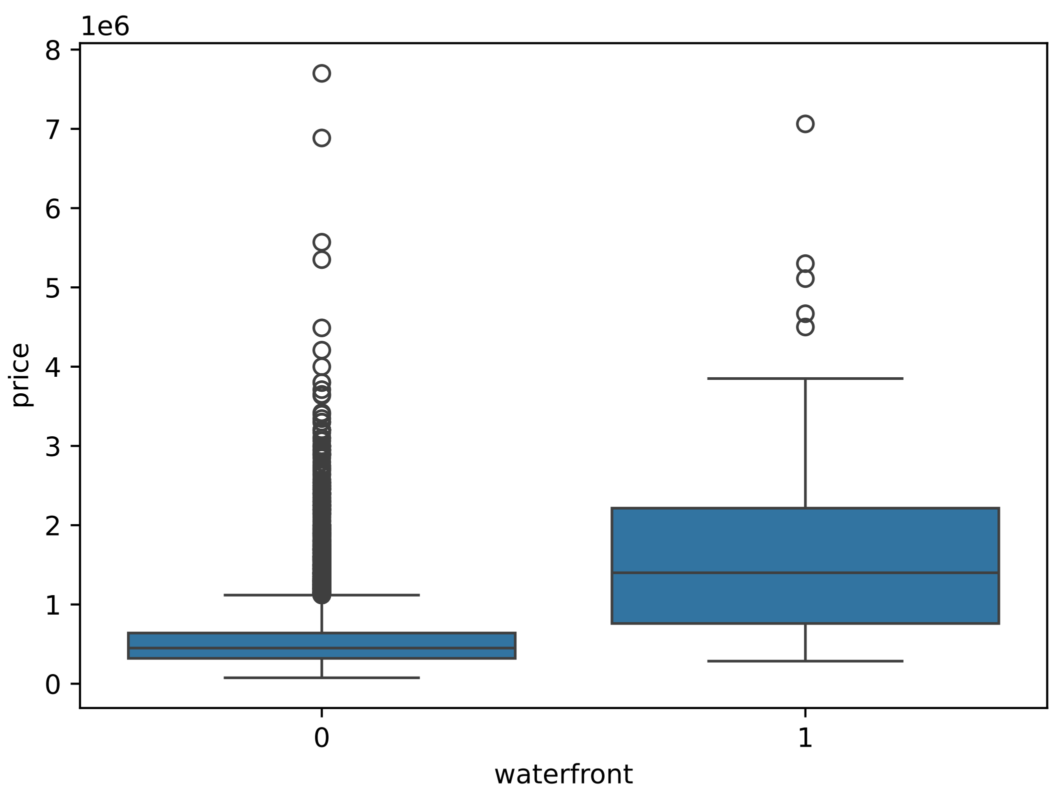
Use the method *value_counts* to count the number of houses with unique floor values, use the method *.to_frame()* to convert it to a data frame.

```
unique_floor_values = df["floors"].value_counts().to_frame()
unique_floor_values
```

```
        count
floors
1.0     10680
2.0      8241
1.5      1910
3.0       613
2.5       161
3.5         8
```

Use the function boxplot in the seaborn library to determine whether houses with a waterfront view or without a waterfront view have more price outliers.

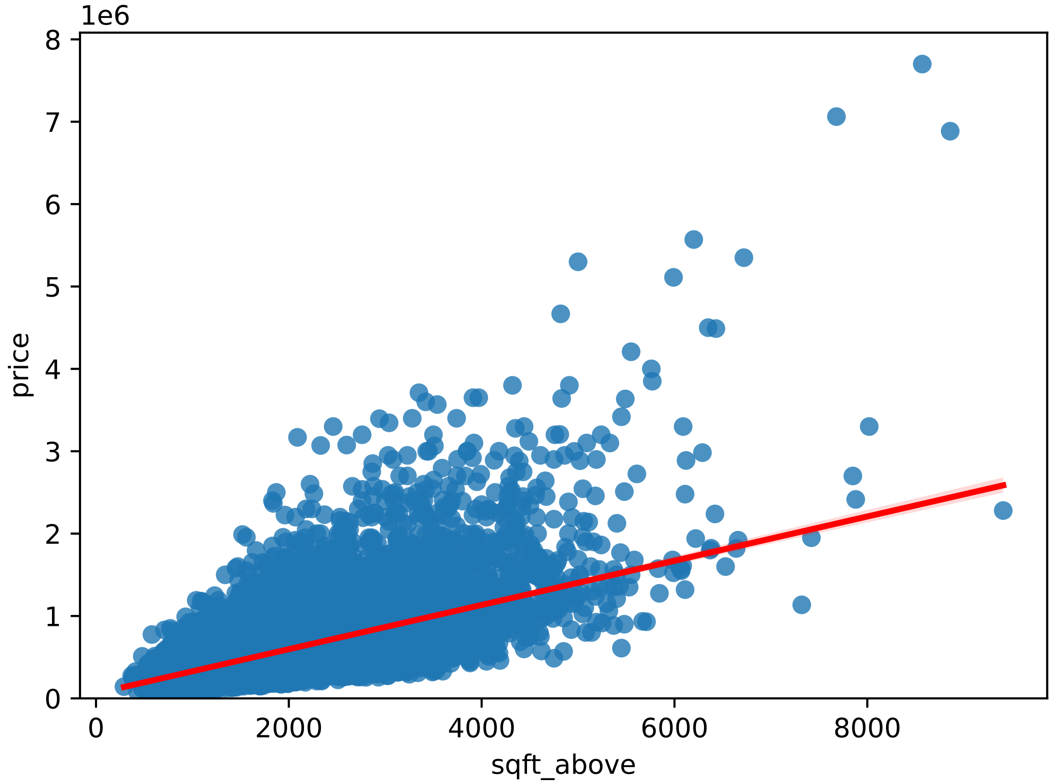
```
sbn.boxplot(x="waterfront", y="price", data=df)
```



Use the function regplot in the seaborn library to determine if the feature sqft_above is negatively or positively correlated with price.

```
sbn.regplot(x="sqft_above", y="price", data=df, line_kws={"color": "red"})
plt.ylim(0,)
```

```
(0.0, 8081250.0)
```




We can use the Pandas method corr() to find the feature other than price that is most correlated with price.

```
# df.corr()['price'].sort_values()
```

We can Fit a linear regression model using the longitude feature 'long' and calculate the $R^2$.

```
X = df[["long"]]
Y = df[["price"]]
lm = LinearRegression()
lm.fit(X,Y)
```

```
LinearRegression()
```

```
print(lm.score(X,Y))
```

```
0.00046769430149007363
```

Fit a linear regression model to predict the 'price' using the feature 'sqft_living' then calculate the R^2.

```
V = df[["sqft_living"]]
Y = df[["price"]]
lm = LinearRegression()
lm.fit(V,Y)
```

```
LinearRegression()
```

```
print(lm.score(V,Y))
```

```
0.4928532179037931
```

Fit a linear regression model to predict the 'price' using the list of features.

```
features =[
  "floors","waterfront","lat","bedrooms","sqft_basement","view",
  "bathrooms","sqft_living15","sqft_above","grade","sqft_living"
  ]
Z = df[features]
lm.fit(Z,Y)
```

```
LinearRegression()
```

```
print(lm.score(Z, Y))
```

```
0.6576868690521125
```

## Pipeline

Create a list of tuples, the first element in the tuple contains the name of the estimator : 'scale', 'polynomial', 'model'. The second element in the tuple contains the model constructor StandardScaler(), PolynomialFeatures(include_bias=False), LinearRegression()

```
Input=[
  ('scale',StandardScaler()), ('polynomial', PolynomialFeatures(include_bias=False)),
  ('model', LinearRegression())
  ]
```

Use the list to create a pipeline object to predict the 'price', fit the object using the features in the list features, and calculate the R^2.

```
pipeline=Pipeline(Input)
Z = Z.astype(float)
pipeline.fit(Z,Y)
```

```
Pipeline(steps=[('scale', StandardScaler()),
                ('polynomial', PolynomialFeatures(include_bias=False)),
                ('model', LinearRegression())])
```

```
ypipe=pipeline.predict(Z)
print(r2_score(Y,ypipe))
```

```
0.751335953931385
```

We will split the data into training and testing sets :

```
features =[
  "floors","waterfront","lat","bedrooms","sqft_basement","view",
  "bathrooms","sqft_living15","sqft_above","grade","sqft_living"
  ]
X = df[features]
Y = df['price']

x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size=0.15, random_state=1)


print("number of test samples:", x_test.shape[0])
```

```
number of test samples: 3242
```

```
print("number of training samples:",x_train.shape[0])
```

```
number of training samples: 18371
```

Create and fit a Ridge regression object using the training data, set the regularization parameter to 0.1, and calculate the $R^2$ using the test data.

```
RidgeModel=Ridge(alpha=0.1)
RidgeModel.fit(x_train, y_train)
```

```
Ridge(alpha=0.1)
```

```
yhat = RidgeModel.predict(x_test)
print(r2_score(y_test,yhat))
```

```
0.6478759163939111
```

Perform a second order polynomial transform on both the training data and testing data. Create and fit a Ridge regression object using the training data, set the regularisation parameter to 0.1, and calculate the $R^2$ utilising the test data provided.

```
pr = PolynomialFeatures(degree=2)
x_train_pr = pr.fit_transform(x_train)
x_test_pr = pr.fit_transform(x_test)
RidgeModel.fit(x_train_pr, y_train)
```

```
Ridge(alpha=0.1)
```

```
y_hat = RidgeModel.predict(x_test_pr)
print(r2_score(y_test,y_hat))
```

```
0.7002744267811738
```