Predicting house prices

In this challenge, you'll be using your newly acquired Linear Regression skills to try **to predict house prices in Ames, Iowa!**

You'll have to do some more complex preprocessing and when you try to model, things might not go your way...

Let's get started!

+ Code + Text

Import the Data

Start by importing the data from this link!

[Iowa_housing_prices.csv]

Load it into a dataframe!

```
import pandas as pd
df = pd.read_csv('5.Iowa_housing_prices.csv')
```

Cleaning

```
df.isna().any()
```

```
Id               False
MSSubClass       False
MSZoning         False
LotFrontage       True
LotArea          False
                 ...
MoSold           False
YrSold           False
SaleType         False
SaleCondition    False
SalePrice        False
Length: 81, dtype: bool
```

Handle NA values

Unlike the previous challenge, this dataset has not been cleaned!

Most important thing to take care of are NA values!

Which columns have missing values?

```
threshold = len(df) * 0.7
```

```
df.dropna(axis=1, thresh=threshold, inplace=True)
```

```
df.isna().any()
```

```
Id               False
MSSubClass       False
MSZoning         False
LotFrontage       True
LotArea          False
                 ...
MoSold           False
YrSold           False
SaleType         False
SaleCondition    False
SalePrice        False
Length: 75, dtype: bool
```

To drop or to fill?

Some columns miss many more values than others!

Drop columns that have more than 30% missing values and fill the others with the mean strategy!

At this stage, if you check for NaN values, you'll still find some, particularly in categorical features. You have the option to either drop these features or proceed with preprocessing to handle them.

Picking X and y

After cleaning, we are left with 76 columns/features. That's a lot to choose from! If we were experts in real estate, we could use our domain knowledge and pick out features we know are important!

However, we're not taking that approach today. We'll use all of the features to try to reach a prediction, all 76 of them!

Assign `X` and `y` appropriately! Keep in mind that we are trying to predict house prices!

```
df.head()
```

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	LotShape	LandContou
0	1	60	RL	65.0	8450	Pave	Reg	Lv
1	2	20	RL	80.0	9600	Pave	Reg	Lv
2	3	60	RL	68.0	11250	Pave	IR1	Lv
3	4	70	RL	60.0	9550	Pave	IR1	Lv
4	5	60	RL	84.0	14260	Pave	IR1	Lv

5 rows × 75 columns

```
df['SaleCondition'].unique()
```

```
array(['Normal', 'Abnorml', 'Partial', 'AdjLand', 'Alloca', 'Family'],
      dtype=object)
```

```
#picking x and y
x = df[['LotFrontage','LotArea','Street','LandContour','YrSold']]
y = df['SalePrice']
```

```
# doing my own steps because it's better >> will convert, then split
numeric_features = x[['LotFrontage','LotArea','YrSold']]
non_numeric_features = x[['Street','LandContour']]
```

```
#ohe is used to quantify qualitative variables

from sklearn.preprocessing import OneHotEncoder
ohe = OneHotEncoder(handle_unknown='ignore', sparse_output=False)
```

```
non_numeric_features_ohe = ohe.fit_transform(non_numeric_features[['Street','LandContour']])
non_numeric_features_ohe
```

```
array([[0., 1., 0., 0., 0., 1.],
       [0., 1., 0., 0., 0., 1.],
       [0., 1., 0., 0., 0., 1.],
       ...,
       [0., 1., 0., 0., 0., 1.],
       [0., 1., 0., 0., 0., 1.],
       [0., 1., 0., 0., 0., 1.]])
```

```
non_numeric_features_ohe = pd.DataFrame(non_numeric_features_ohe, columns=ohe.get_feature_names_out())
non_numeric_features_ohe.head()
```

	Street_Grvl	Street_Pave	LandContour_Bnk	LandContour_HLS	LandContour_Low	LandContour_Lvl
0	0.0	1.0	0.0	0.0	0.0	1.0
1	0.0	1.0	0.0	0.0	0.0	1.0
2	0.0	1.0	0.0	0.0	0.0	1.0
3	0.0	1.0	0.0	0.0	0.0	1.0
4	0.0	1.0	0.0	0.0	0.0	1.0

```
numeric_features
```

	LotFrontage	LotArea	YrSold
0	65.0	8450	2008
1	80.0	9600	2007
2	68.0	11250	2008
3	60.0	9550	2006
4	84.0	14260	2008
...	...	...	...
1455	62.0	7917	2007
1456	85.0	13175	2010
1457	66.0	9042	2010
1458	68.0	9717	2010
1459	75.0	9937	2008

1460 rows × 3 columns

```
# combine

# x_combined = pd.concat([numeric_features, non_numeric_features_ohe], join='inner',axis=1, ignore_index=True)

x_combined = numeric_features.join(non_numeric_features_ohe, how='inner')
```

```
x_combined.fillna(method='ffill', inplace=True)
```

```
# # then split
# x_combined.dropna(axis=1, thresh=threshold, inplace=True)
# x_combined.shape[0]
```

```
x_combined
```

	LotFrontage	LotArea	YrSold	Street_Grvl	Street_Pave	LandContour_Bnk	LandContour_HLS	LandContour_Low	LandContou
0	65.0	8450	2008	0.0	1.0	0.0	0.0	0.0	
1	80.0	9600	2007	0.0	1.0	0.0	0.0	0.0	
2	68.0	11250	2008	0.0	1.0	0.0	0.0	0.0	
3	60.0	9550	2006	0.0	1.0	0.0	0.0	0.0	
4	84.0	14260	2008	0.0	1.0	0.0	0.0	0.0	
...	...	...	...	...	...	...	...	...	
1455	62.0	7917	2007	0.0	1.0	0.0	0.0	0.0	
1456	85.0	13175	2010	0.0	1.0	0.0	0.0	0.0	
1457	66.0	9042	2010	0.0	1.0	0.0	0.0	0.0	
1458	68.0	9717	2010	0.0	1.0	0.0	0.0	0.0	
1459	75.0	9937	2008	0.0	1.0	0.0	0.0	0.0	

1460 rows × 9 columns

```
y.shape[0]
```

```
1460
```

Train test split

As always, we need to split the data into train and test!

```
# here we train the model and then test it
from sklearn.model_selection import train_test_split
import numpy as np
```

```
x_train, x_test, y_train, y_test = train_test_split(x_combined, y, test_size=0.2, random_state=0)
```

Normalization

We can't skip this step! However, unlike the previous challenge, we now have non-numeric columns as well that we need to take care of!

Numeric Values

You have to do it only on numerical data!

▶ Hint:

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
```

```
# numeric_features_train_scaled = scaler.fit_transform(numeric_features_train)
# numeric_features_test_scaled = scaler.transform(numeric_features_test)
```

Non-Numeric Values

Now let's try to do the same for the non-numeric columns! Use `select_dtypes` again and maybe change that `include` to something else? Have a look at the documentation!

After you've selected the non-numeric columns, use `OneHotEncoder` to encode the data!

```
# non_numeric_features_train = x_train[['Street','LandContour']]
# non_numeric_features_test = x_test[['Street','LandContour']]

# so we don't need to use scaler on categorical because after we use one hot encoder, the values are not that extreme
# why do we normalize only numerical values? because sometimes they can have very large ranges, non-numerical values that ha
```

```
# from sklearn.preprocessing import OneHotEncoder
# enc = OneHotEncoder(handle_unknown='ignore', sparse_output=False)
```

```
# non_numeric_features_train_ohe = enc.fit_transform(non_numeric_features_train)
# non_numeric_features_test_ohe = enc.transform(non_numeric_features_test)
```

```
# non_numeric_features_train_ohe = pd.DataFrame(data=non_numeric_features_train_ohe)
# non_numeric_features_test_ohe = pd.DataFrame(data=non_numeric_features_test_ohe)
```

Have a look at your encoded columns.

Recreate X

Recreate `X` now by combining (concatenating) the numeric and non-numeric normalized columns together! Call it `X_normalized`!

```
# import pandas as pd
```

```
# x_normalized = pd.concat([numeric_features_train_scaled, non_numeric_features_train_ohe], ignore_index=True)
```

Try a Linear Regression

Let's try to use a Linear Regression to model house prices! Instantiate and fit a model!

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
```

```
x_train_scaled = scaler.fit_transform(x_train)
x_test_scaled = scaler.transform(x_test)
```

```
from sklearn.linear_model import LinearRegression
lin_reg = LinearRegression()
```

```
# x_combined.fillna(method='ffill', inplace=True)
```

```
x_combined.isnull().sum()
```

```
LotFrontage       0
LotArea           0
YrSold            0
Street_Grvl       0
Street_Pave       0
LandContour_Bnk   0
LandContour_HLS   0
LandContour_Low   0
LandContour_Lvl   0
dtype: int64
```

```
lin_reg.fit(x_train_scaled, y_train)
```

```
▾ LinearRegression
LinearRegression()
```

Calculate the MAE

Let's now calculate the mean absolute error of the model on the test set.

```
from sklearn.metrics import mean_absolute_error
y_prediction = lin_reg.predict(x_train_scaled)
mae = mean_absolute_error(y_train, y_prediction)
mae
```

```
53360.72721898664
```

That's quite a large number and it represents the amount, in the dollars, by which we were wrong about house prices! **Ouch!**

What went wrong?

Predicting house prices is, believe it or not, a very complex endeavour! There's not one single quality that determines house prices well, it's one large complex soup of features.

Furthermore, there is a good probability that this is a **non-linear** task! Which would mean that our Linear Regression is ill-suited to handle it.

Whenever you encounter the limitations of a Linear Regression, there's a couple of things that you could try:

- See if there is not a *numerical* data that are *categorical*
- Remove colinear features
- Apply some regularization techniques
- Try non-linear models

Improve our model

Let's try with fewer features. Select features that are relevant to predict the SalePrice.

▸ Answer:

```
X = df[['LotArea', 'LotConfig', 'LotShape', 'MSZoning', 'BldgType', 'Neighborhood', 'GarageCars']]
Y = df['SalePrice']
```

```
X = pd.DataFrame(X)
X.head()
```

	LotArea	LotConfig	LotShape	MSZoning	BldgType	Neighborhood	GarageCars
0	8450	Inside	Reg	RL	1Fam	CollgCr	2
1	9600	FR2	Reg	RL	1Fam	Veenker	2
2	11250	Inside	IR1	RL	1Fam	CollgCr	2
3	9550	Corner	IR1	RL	1Fam	Crawfor	3
4	14260	FR2	IR1	RL	1Fam	NoRidge	3

```
X_numeric = X[['LotArea','GarageCars']]
X_non_numeric = X[['LotConfig','LotShape','MSZoning','BldgType','Neighborhood']]
```

```
X_non_numeric
```

	LotConfig	LotShape	MSZoning	BldgType	Neighborhood
0	Inside	Reg	RL	1Fam	CollgCr
1	FR2	Reg	RL	1Fam	Veenker
2	Inside	IR1	RL	1Fam	CollgCr
3	Corner	IR1	RL	1Fam	Crawfor
4	FR2	IR1	RL	1Fam	NoRidge
...	...	...	...	...	...
1455	Inside	Reg	RL	1Fam	Gilbert
1456	Inside	Reg	RL	1Fam	NWAmes
1457	Inside	Reg	RL	1Fam	Crawfor
1458	Inside	Reg	RL	1Fam	NAmes
1459	Inside	Reg	RL	1Fam	Edwards

1460 rows × 5 columns

```
X_non_numeric_ohe = ohe.fit_transform(X_non_numeric[['LotConfig','LotShape','MSZoning','BldgType','Neighborhood']])
```

```
X_non_numeric_ohe = pd.DataFrame(X_non_numeric_ohe, columns=ohe.get_feature_names_out())
```

```
X_non_numeric_ohe
```

	LotConfig_Corner	LotConfig_CulDSac	LotConfig_FR2	LotConfig_FR3	LotConfig_Inside	LotShape_IR1	LotShape_IR2	Lot:
0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	
1	0.0	0.0	1.0	0.0	0.0	0.0	0.0	
2	0.0	0.0	0.0	0.0	1.0	1.0	0.0	
3	1.0	0.0	0.0	0.0	0.0	1.0	0.0	
4	0.0	0.0	1.0	0.0	0.0	1.0	0.0	
...	...	...	...	...	...	...	...	
1455	0.0	0.0	0.0	0.0	1.0	0.0	0.0	
1456	0.0	0.0	0.0	0.0	1.0	0.0	0.0	
1457	0.0	0.0	0.0	0.0	1.0	0.0	0.0	
1458	0.0	0.0	0.0	0.0	1.0	0.0	0.0	
1459	0.0	0.0	0.0	0.0	1.0	0.0	0.0	

1460 rows × 44 columns

```
X_combined = X_numeric.join(X_non_numeric_ohe, how='inner')
```

```
X_combined.head()
```

	LotArea	GarageCars	LotConfig_Corner	LotConfig_CulDSac	LotConfig_FR2	LotConfig_FR3	LotConfig_Inside	LotShape_IR1
0	8450	2	0.0	0.0	0.0	0.0	1.0	0.0
1	9600	2	0.0	0.0	1.0	0.0	0.0	0.0
2	11250	2	0.0	0.0	0.0	0.0	1.0	1.0
3	9550	3	1.0	0.0	0.0	0.0	0.0	1.0
4	14260	3	0.0	0.0	1.0	0.0	0.0	1.0

5 rows × 46 columns

Split you data.

```
X_train, X_test, Y_train, Y_test = train_test_split(X_combined, Y, test_size = 0.2, random_state = 0)
```

Be sure that you normalize only data you need to normalize.

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
```

```
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Encode your categorical features.

Let's try again with a Linear regression.

```
lin_reg.fit(X_train_scaled, Y_train)
```

```
▾ LinearRegression
LinearRegression()
```

```
lin_reg.score(X_train_scaled, Y_test)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-382-fd6e352a0baa> in <cell line: 1>()
----> 1 lin_reg.score(X_train_scaled, Y_test)

                              3 frames
/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py in check_consistent_length(*arrays)
    395     uniques = np.unique(lengths)
    396     if len(uniques) > 1:
--> 397         raise ValueError(
    398             "Found input variables with inconsistent numbers of samples: %r"
    399             % [int(l) for l in lengths]

ValueError: Found input variables with inconsistent numbers of samples: [292, 1168]
```

If you choose the right columns, you could see an improvement between `$30,000` and `$40,00`. That's a significant enhancement.

So, it is very important to understand the data you use.

Optional

All of the above are out of scope for the DA bootcamp, but it's worth knowing about them! We've coded a cell below that uses a **Random Forest Model** to predict house prices! Try to see if you can make some sense of it.

P.S.: You might need to adjust some variable names if we weren't able to guess them right.

```
from sklearn.ensemble import RandomForestRegressor

X_train = # your code
X_test = # your code
y_train = # your code
y_test = # your code
```