Training and Test Errors

```
import pandas as pd
import numpy as np
```

Try to use scikit-learn whenever possible.

Ames Housing Data

```
df_ames = pd.read_csv("http://dlsun.github.io/pods/data/AmesHousing.txt", sep = "\t")

df_ames
```

	Order	PID	MS SubClass	MS Zoning	Lot Frontage	Lot Area	Street	Alley	Lot Shape	Con
0	1	526301100	20	RL	141.0	31770	Pave	NaN	IR1	
1	2	526350040	20	RH	80.0	11622	Pave	NaN	Reg	
2	3	526351010	20	RL	81.0	14267	Pave	NaN	IR1	
3	4	526353030	20	RL	93.0	11160	Pave	NaN	Reg	
4	5	527105010	60	RL	74.0	13830	Pave	NaN	IR1	
...	...	...	...	...	...	...	...	...	...	
2925	2926	923275080	80	RL	37.0	7937	Pave	NaN	IR1	
2926	2927	923276100	20	RL	NaN	8885	Pave	NaN	IR1	
2927	2928	923400125	85	RL	62.0	10441	Pave	NaN	Reg	
2928	2929	924100070	20	RL	77.0	10010	Pave	NaN	Reg	
2929	2930	924151050	60	RL	74.0	9627	Pave	NaN	Reg	

2930 rows × 82 columns

1. Fit a 10-nearest neighbors model to predict **SalePrice** using **Bldg Type** as the only feature.

```
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsRegressor
from sklearn.compose import make_column_transformer
from sklearn.preprocessing import OneHotEncoder

df_ames_knn = df_ames

X_train = df_ames_knn[["Bldg Type"]]
y_train = df_ames_knn["SalePrice"]

ct = make_column_transformer(
    (OneHotEncoder(), ["Bldg Type"]),
    remainder="passthrough"  # all other columns in X will be passed through unchanged
)

pipeline = make_pipeline(
    ct,
    KNeighborsRegressor(n_neighbors=10)
)

pipeline.fit(X=X_train, y=y_train)
```

```
▸ Pipeline
  ▸ columntransformer: ColumnTransformer
```

2. Calculate the **training error** of this model. Try a few different performance metrics.

```
from sklearn.metrics import mean_squared_error
import numpy as np

y_train_ = pipeline.predict(X=X_train)
y_train_
```

```
array([185170., 185170., 185170., ..., 185170., 185170., 185170.])
```

```
mse = mean_squared_error(y_train, y_train_)
mse
```

```
6193813781.924573
```

```
rmse = np.sqrt(mse)
rmse
```

```
78700.78641236422
```

3. Repeat the above process to calculate the training error for $k = 1, 2, \ldots, 10$. Which value of k gives the smallest training error? Does that necessarily mean this is the best value of k? Discuss with your partner.

```
for k in range(1, 11):
  pipeline = make_pipeline(
    ct,
    KNeighborsRegressor(n_neighbors=k)
  )
  pipeline.fit(X=X_train, y=y_train)

  y_train_ = pipeline.predict(X=X_train)
  print(np.sqrt(mean_squared_error(y_train, y_train_)))
```

```
83869.12582123668
82461.56991333194
81459.84807307809
79077.36953503109
78841.98384073898
78718.47565629905
78666.84969111855
78813.2596795962
78781.76304262652
78700.78641236422
```

`k = 7` has the smallest training error. This does not necessarily mean that it has the best value of k, as neighbors with k > 10 may have a smaller training error.

4. Return to the model in part 1. Now estimate the **test error of** the model using cross-validation. Try a few different performance metrics.

```
from sklearn.model_selection import cross_val_score

scores = cross_val_score(pipeline,
                         X=X_train,
                         y=y_train,
                         scoring="neg_mean_squared_error",
                         cv=5)
scores
```

```
array([-8.24099337e+09, -5.50326008e+09, -5.70148673e+09, -6.64332148e+09,
       -5.60974944e+09])
```

```
np.sqrt(-scores).mean() # average test error
```

```
79375.38036425046
```

5. Now, define a 10-nearest neighbors model to predict **SalePrice** using **Neighborhood** as the only feature. Try to estimate the test error of this model using cross validation.

You will get an error. Can you figure out why this error occurs? Can you figure out how to fix it?

```
X_train2 = df_ames_knn[["Neighborhood"]]
y_train2 = df_ames_knn["SalePrice"]

ct2 = make_column_transformer(
    (OneHotEncoder(handle_unknown="ignore"), ["Neighborhood"]),
    remainder="drop"  # all other columns in X will be passed through unchanged
)

pipeline2 = make_pipeline(
    ct2,
    KNeighborsRegressor(n_neighbors=10)
)

pipeline2.fit(X=X_train2, y=y_train2)
```

```
▸ Pipeline
▸ columntransformer: ColumnTransformer
▸ onehotencoder
▸ OneHotEncoder
▸ KNeighborsRegressor
```

```
scores2 = cross_val_score(pipeline2,
                          X=X_train2,
                          y=y_train2,
                          scoring="neg_mean_squared_error",
                          cv=5)
np.sqrt(-scores2)
```

```
array([64251.79816865, 50256.6188725 , 52896.91087655, 60271.46679312,
       50588.36441503])
```

```
np.sqrt(-scores2).mean()
```

```
55653.03182517042
```

The OneHotEncoder converts all the Neighborhoods in the training set into dummies. There's a mismatch between the columns in the training set and the validation set, which causes the error.

6. Recall that in a previous notebook we fit a 10-nearest neighbors regression model that predicts the price (just **SalePrice**, not log) of a home using square footage (**Gr Liv Area**), number of bedrooms (**Bedroom AbvGr**), number of full bathrooms (**Full Bath**), number of half bathrooms (**Half Bath**), and **Neighborhood**. Fit this model and estimate its test error using cross-validation. Try a few different performance metrics.

```
X_train3 = df_ames_knn[["Gr Liv Area", "Bedroom AbvGr", "Full Bath", "Half Bath", "Neighborhood"]]
y_train3 = df_ames_knn["SalePrice"]

ct3 = make_column_transformer(
    (OneHotEncoder(handle_unknown="ignore"), ["Neighborhood"]),
    (StandardScaler(), ["Gr Liv Area", "Bedroom AbvGr", "Full Bath", "Half Bath"]),
    remainder="drop"  # don't make model on other categories
)

pipeline3 = make_pipeline(
    ct3,
    KNeighborsRegressor(n_neighbors=10)
)

pipeline3.fit(X=X_train3, y=y_train3)
```

```
Pipeline
columntransformer: ColumnTransformer
onehotencoder      standardscaler
OneHotEncoder      StandardScaler
```

```
scores3 = cross_val_score(pipeline3,
                          X=X_train3,
                          y=y_train3,
                          scoring="neg_mean_squared_error",
                          cv=5)
np.sqrt(-scores3)
```

```
array([42378.60222257, 36959.4675814 , 43940.36822748, 42894.83382625,
       32686.68232439])
```

```
np.sqrt(-scores3).mean()
```

```
39771.99083641717
```

7. Repeat the process in part 6 to fit k-nearest neighbors regression models for several values of k (say $k = 1, \ldots, 20$). Which value of k produces the best test error? Try a few different performance metrics; does the best value of k depend on the metric?

```
for k in range(1, 21):
  pipeline4 = make_pipeline(
    ct3,
    KNeighborsRegressor(n_neighbors=k)
  )

  scores4 = cross_val_score(pipeline4,
                          X=X_train3,
                          y=y_train3,
                          scoring="neg_mean_squared_error",
                          cv=5)
  print(np.sqrt(-scores4).mean())
```

```
45858.774794573845
41189.47159704959
40114.64822571264
39685.6408173541
39232.787241678474
38990.71285714399
39331.08053802313
39396.75027806344
39606.901035557974
39771.99083641717
39986.09104434125
40132.144439508585
40264.00249925615
40370.261239407315
40494.60786398502
40594.778757171836
40758.49489656655
40886.24629835865
41108.988827075766
41171.836109168304
```