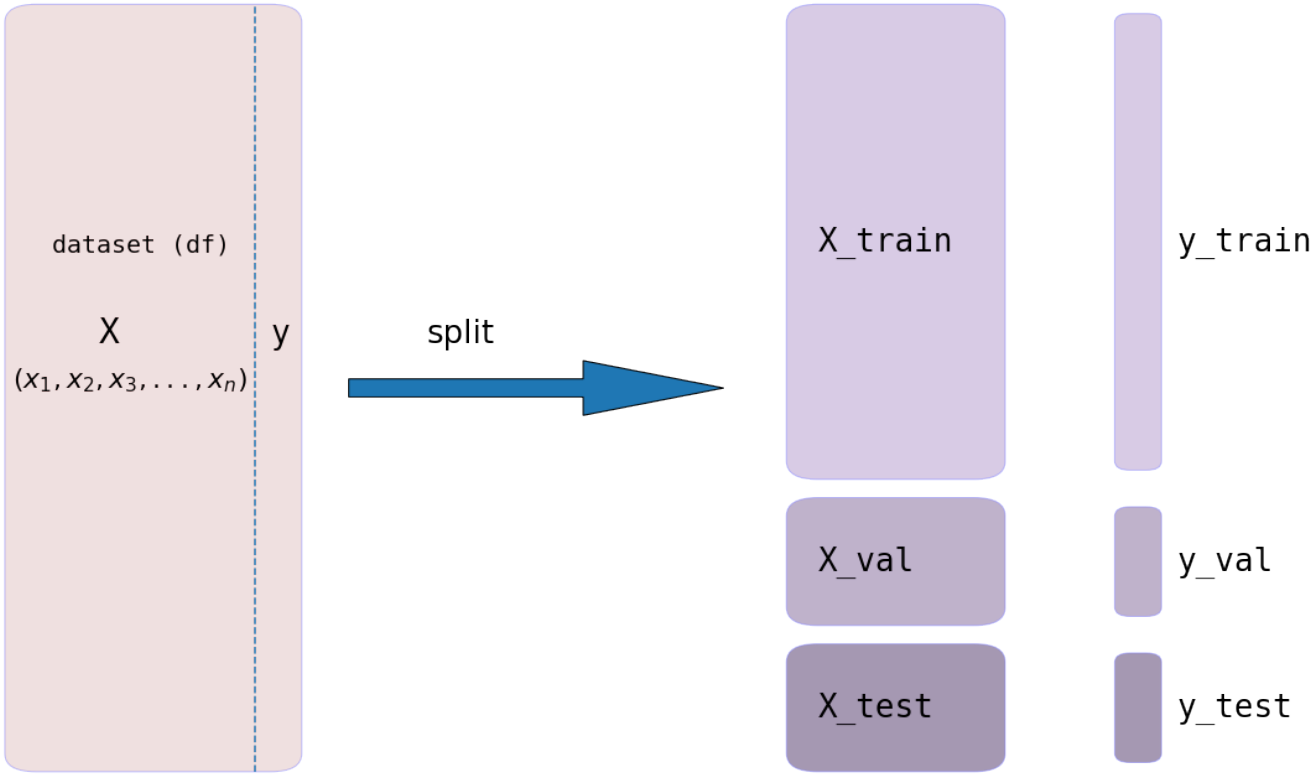
Model performance evaluation

Carl McBride Ellis

carl.mcbride@u-tad.com

Dataset splitting: train, validation, and final test

`df` $\rightarrow$ `X_train`, `y_train`, `X_val`, `y_val`, `X_test`, `y_test`



Important: the `X_test` `y_test` data should only be used once!

Scikit has routines to help with the task of splitting datasets

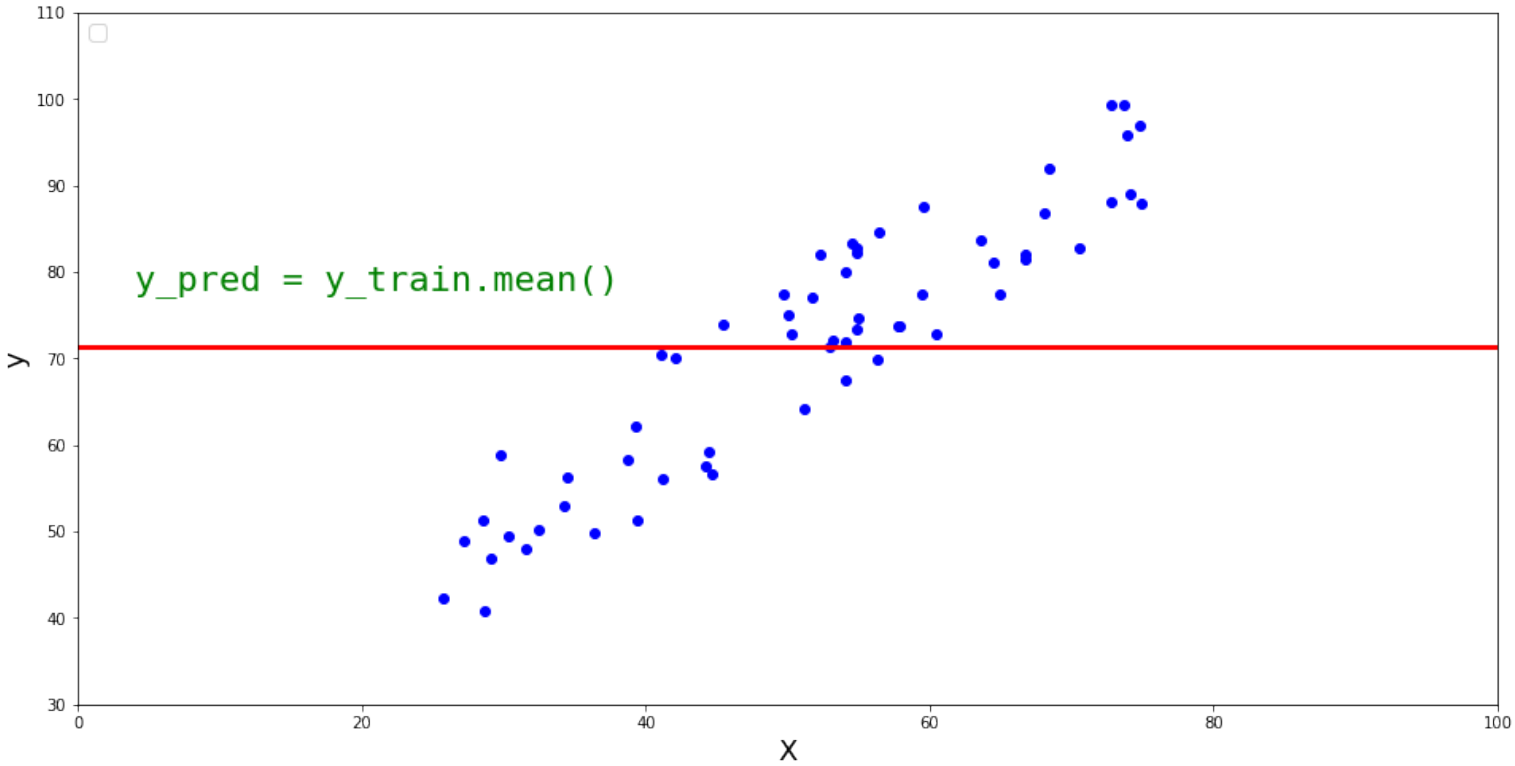
- `from sklearn.model_selection import train_test_split`

example of performing the `train_test_split`

```
X = dataset
y = X.pop('target')

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test  = train_test_split(
                                X, y,
                                test_size=0.20, # 20% test size
                                random_state=42)
```

The very first thing we should do is calculate the 'out-of-sample' performance of the 'baseline' model:

$$\hat{y} = \mathbb{E}[f(x)] = \bar{y} \quad \forall\ \hat{y}$$



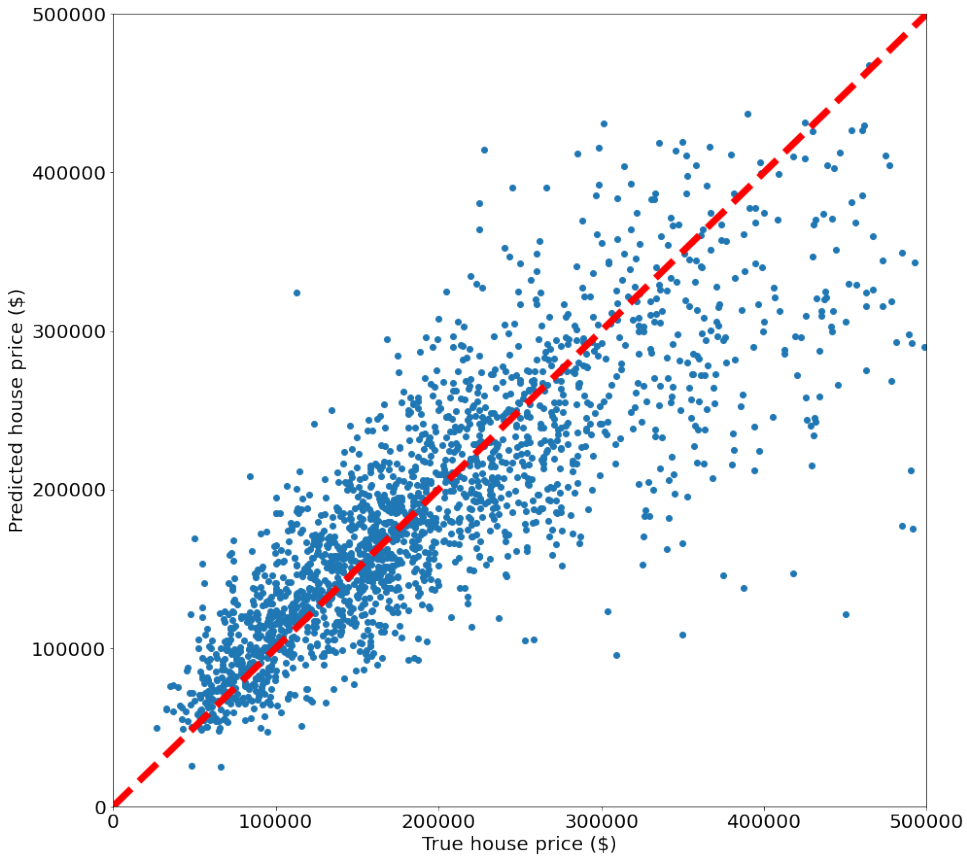
example: calculating the baseline score

```
from sklearn.metrics import root_mean_squared_error as metric

y_train_mean = np.full( len(y_test), np.mean(y_train) )
metric(y_test, y_train_mean)
```

where the `root_mean_squared_error` is given by

$$\mathrm{RMSE} = \sqrt{\varepsilon^2} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

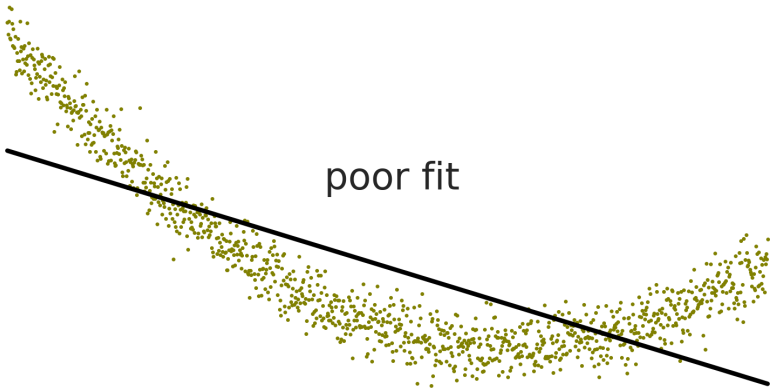
How well is our model doing?: 'True' vs 'predicted' plot

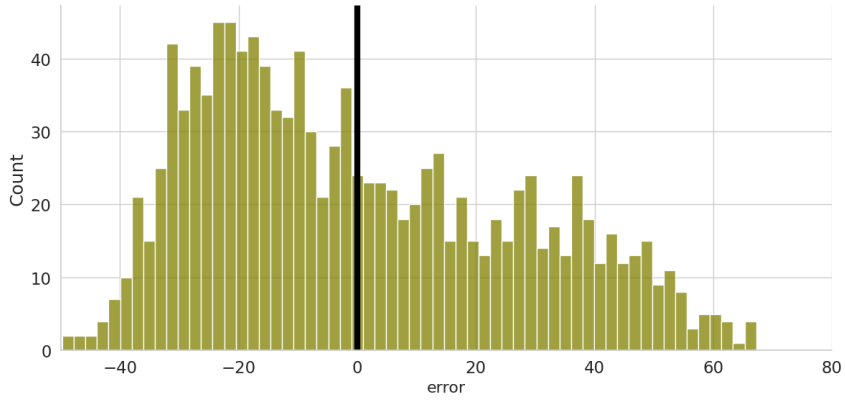


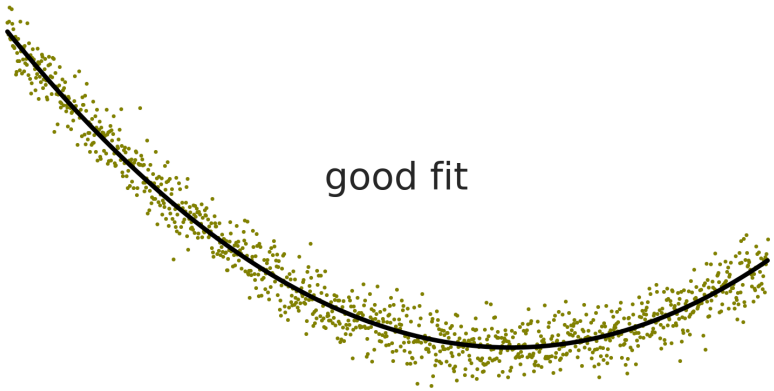
(`sklearn.metrics.PredictionErrorDisplay`)

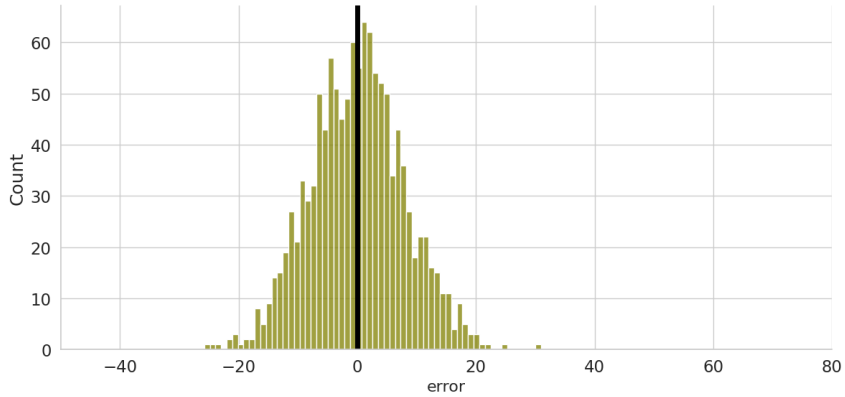
```
fig, ax = plt.subplots(figsize=(7, 7))
ax.axline([0, 0], [1, 1],
            color = "red",
            linestyle='--',
            lw=3, zorder=3)
ax.scatter(x=y_validation, y=y_pred, c='blue', label="data")

minimum = min(y_pred.min(), y_validation.min())
plt.xlim(minimum)
plt.ylim(minimum)
plt.xlabel('True')
plt.ylabel('Predicted')
plt.show();
```

The residuals of a good fit should have a symmetric distribution:









```
residual_mean = (y_validation - y_pred).mean()

sns.displot( (y_validation - y_pred), height=6, aspect=2)
plt.xlabel('residual')
# add a mean value indicator
plt.axvline(residual_mean, color="red", lw=1);
```

Regression performance metrics

The most natural metric if we are using the L2 loss is the *root mean squared error* (RMSE)

$$\text{RMSE} = \sqrt{\overline{\varepsilon^2}} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

`sklearn.metrics.mean_squared_error`

The most natural metric if we are using the L1 loss is the *mean absolute error* (MAE)

$$\text{MAE} = \overline{|\varepsilon|} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

`sklearn.metrics.mean_absolute_error`

The data for train, validation, and final test:

- `train`: use the training data to find the model parameters
- `validation`: (or *hold-out*) to find the hyperparameters
- `test`: a final evaluation of the model performance

It is a good idea for the validation set to be the same size as the test set when comparing performance: for example roughly $\approx$ 80:10:10
(a "*Pareto*" split)

We want that each column of the three datasets are independent and identically distributed (iid).
Solution: it is usually sufficient to perform a random split (in classification we also use `stratify=y`).

Potential problems:

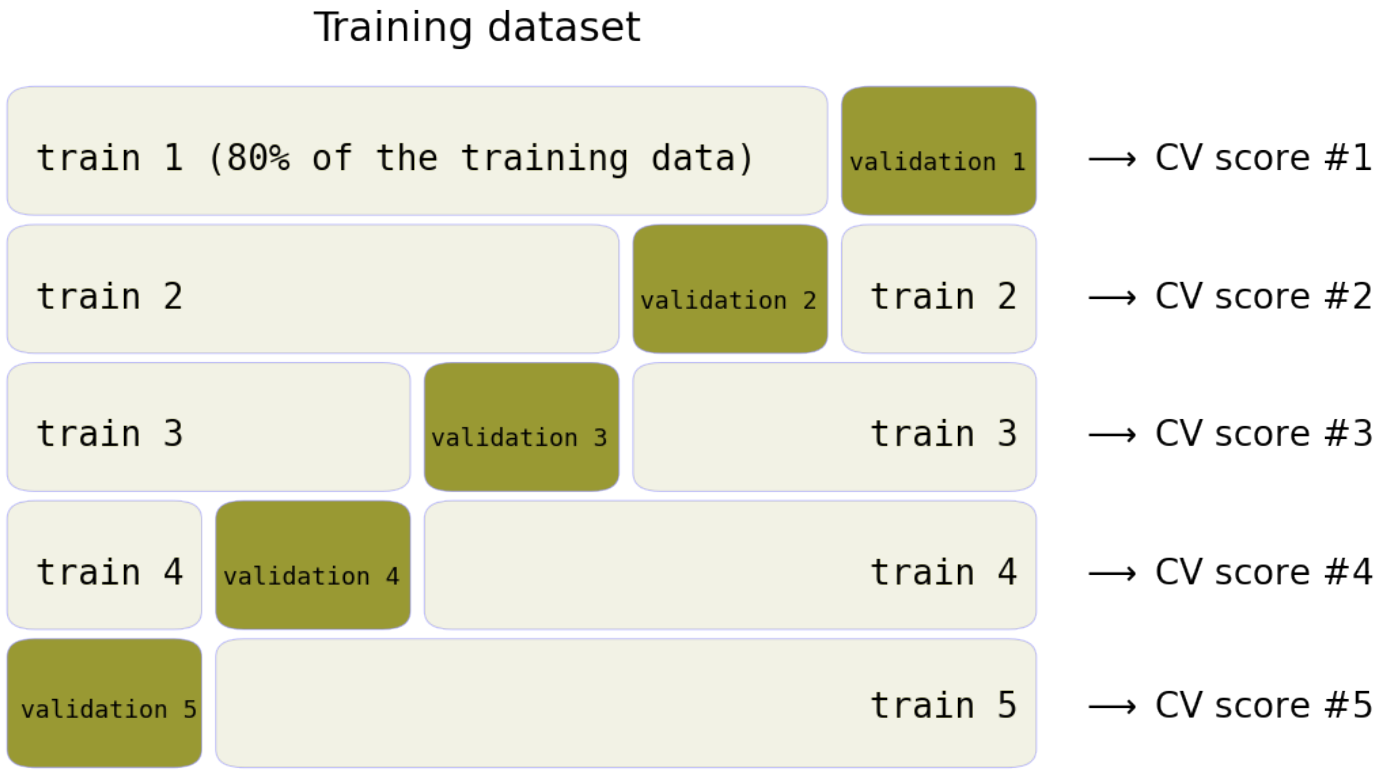
- **Covariate shift** $P(X)$: The `X` in `X_train` are not the same as the `X` in `X_test`.
  For example we modeled houses of 70m^2 to 120m^2 and we are asked to predict prices of hoses in the range 120m^2 to 170m^2
- **Concept drift** $P(y|X)$: The `y` in train are not the same as those in the test.
  For example we train with prices from the year 1980 but we are predicting in the year 2024. The houses have not changed, but the market has.

Solution: re-train our model with new data.

Advanced material: What is Adversarial Validation?

Advanced material: Kolmogórov-Smirnov test

What happens if we have very little data?
k-fold cross-validation (here $k = 5$):



More information: Cross-validation: evaluating estimator performance

(Advanced material: Nested cross-validation en *"Cross-validation: What does it estimate and how well does it do it?"*)

Scikit makes it easy to calculate the results of a cross-validation

- `from sklearn.model_selection import cross_val_score`

example: calculating the CV score

```
from sklearn.model_selection import cross_val_score, KFold
# Regression using the L2 loss
from sklearn.metrics import root_mean_squared_error
from sklearn.metrics import make_scorer

RMSE = make_scorer(root_mean_squared_error)
kf = KFold(n_splits=5, shuffle=True, random_state=42)
CV_scores = cross_val_score(regressor,
                            X_train, y_train,
                            cv=kf,
                            scoring=RMSE,
                            n_jobs=-1)

print("CV scores: ", CV_scores)
print("RMS of CV RMSE: ",np.sqrt(np.mean(np.square(CV_scores))))
print("Std. dev of CV score: ", np.std(CV_scores))
```

(Note: the scikit learn `cross_val_score` admits the possibility of data leakage. For a more precise evaluation see this code).

Danger! *"data leakage"*

- **target leakage**: between columns
- **future leakage**: between rows
  when our model has seen the 'future'
  - transformations: for example scaling, mean imputation

Problem: leakage results in overly optimistic results

How do we avoid leakage: Cleaning the dataframes `X_train`, `X_validation` and `X_test` equally, but individually.

for example, to avoid future leakage via `NaN` imputation:

```
X_train = X_train.fillna( X_train.mean() )
X_validation = X_validation.fillna( X_train.mean() )
X_test = X_test.fillna( X_train.mean() )
```

we always fill the missing values with the mean that was obtained from the training data

The (in)famous leakage by Andrew Ng (paper)

3. Data

3.1. Training

We use the ChestX-ray14 dataset released by Wang et al. (2017) which contains 112,120 frontal-view X-ray images of 30,805 unique patients. Wang et al. (2017) annotate each image with up to 14 different thoracic pathology labels using automatic extraction methods on radiology reports. We label images that have pneumonia as one of the annotated pathologies as positive examples and label all other images as negative examples for the pneumonia detection task. We randomly split the entire dataset into 80% training, and 20% validation.

CheXNet: Radiologist-Level Pneumonia Detection on Chest X-Rays with Deep Learning

---

Pranav Rajpurkar [*1] Jeremy Irvin [*1] Kaylie Zhu [1] Brandon Yang [1] Hershel Mehta [1]
Tony Duan [1] Daisy Ding [1] Aarti Bagul [1] Curtis Langlotz [2] Katie Shpanskaya [2]
Matthew P. Lungren [2] Andrew Y. Ng [1]

¿can you spot the problem?

Practical session

We are going to do an end-to-end linear regression project

`notebook_ML_regression_concrete_template.ipynb`

tasks:

- split the dataset (`df`) into a data matrix and target vector
  i.e. into `X` and `y`
- split your data into disjoint train and test sets
  i.e. into `X_train, y_train` and `X_test, y_test`
- re-scale the datasets
- look at the β's, *i.e.* identify the most important features
- estimate the baseline model RMSE performance
- calculate the `LinearRegression` model CV performance
- produce a final fit to all of `X_train`
  and calulate the performance for `X_test`