Accuracy is not always a useful metric to validate models

We can prevent Class imbalance, an uneven frequency of classes, through a confusion matrix.

	Predicted: Legitimate	Predicted: Fraudulent
Actual: Legitimate	True Negative	False Positive
Actual: Fraudulent	False Negative	True Positive

TP = correctly predicted positives
TN = correctly predicted negatives
FP = predicted positive, but actually negative
FN = predicted negative, but actually positive

$$\frac{tp + tn}{tp + tn + fp + fn}$$

precision:

$$\frac{tp}{tp + fp}$$

high precision = lower false positive rate

recall:

$$\frac{tp}{tp + fn}$$

high recall = lower false negative rate

F1: harmonic mean of precision and recall

$$2 * \frac{precision * recall}{precision + recall}$$

```
from sklearn.metrics import classification_report, confusion_matrix
knn = KNeighborsClassifier(n_neighbors=7)
X_train, X_test, y_trian, y_test = train_Test_split(X, y, test_size = 0.4,
random_state=42)
knn.fit(X_train, y_train)
```

```
y_pred = knn.predict(X_test)
print(confusion_matrix(y_test, y_pred)) # 1106 tn, 183 fn, 11 fp, 34 tp
print(classification_report(y_test, y_pred))
```

```
print(confusion_matrix(y_test, y_pred))
```

```
[[1106   11]
 [ 183   34]]
```

```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.86      0.99      0.92      1117
           1       0.76      0.16      0.26       217

    accuracy                           0.85      1334
   macro avg       0.81      0.57      0.59      1334
weighted avg       0.84      0.85      0.81      1334
```

Logistic regression and the ROC curve

Used for classification problems
Output probabilities that the data is in a class
If p>0.5, data is labelled as 1. Else, labelled as 0

```
from sklearn.linear_model import LogisticRegression
logreg = LogisticRegression()
# split data
# logreg.fit
# predict
```

```
y_pred_probs = logreg.predict_proba(X_test)[:, 1]
print(y_pred_probs[0])
# default threshold is 0.5
```

The threshold can be varied. The ROC curve shows threshold changes affect results

```
from sklearn.metrics import roc_curve
fpr, tpr, thresholders = roc_curve(y_test, y_pred_probs)
plt.plot([0,1], [0, 1], 'k--')
plt.plot(fpr, tpr) # false positive rate, true positive rate
plt.show()
```

Area under Curve (AUC)

```
from sklearn.metrics import roc_auc_score
print(roc_auc_score(y_test, y_pred_probs)) # 0.67
```

Hyperparameter tuning

Hyperparameters such as alpha or n_neighbors
How to choose correct parameters?

1. try different values
2. fit all separately
3. see how they perform
4. choose the best values

Important to use CV when hyperparameter tuning

```
from sklearn.model_selection import GridSearchCV
kf = KFold(n_splits=5, shuffle=True, random_state=42)
param_grid = {"alpha": np.arange(0.0001, 1, 10),
                "solver": ["sag", "lsqr"]}
ridge = Ridge()
ridge_cv = GridSearchCV(ridge, param_grid, cv=kf)
ridge_cv.fit(X_train, y_train
print(ridge_cv.best_params_, ridge_cv.best_score_)
```

The number of fits = number of hyperparameters *values* folders
This doesn't scale well.

Ex:
3-fold, 1 hyperparam, 10 values = 30 fits
10-fold, 3 hyperparams, 30 values = 900 fits

Or use **RandomizedSearchCV**

```
from sklearn.model_selection import RandomizedSearchCV
kf = KFold(n_splits=5, shuffle=True, random_state=42)
param_grid = {"alpha": np.arange(0.0001, 1, 10),
                "solver": ["sag", "lsqr"]}
```

```
ridge = Ridge()
ridge_cv = GridSearchCV(ridge, param_grid, cv=kf, n_iter=2) # n_iter is the
number of hyperparams tested
ridge_cv.fit(X_train, y_train)
print(ridge_cv.best_params_, ridge_cv.best_score_)

test_score = ridge_cv.score(X_test, y_test)
print(test_score)
```