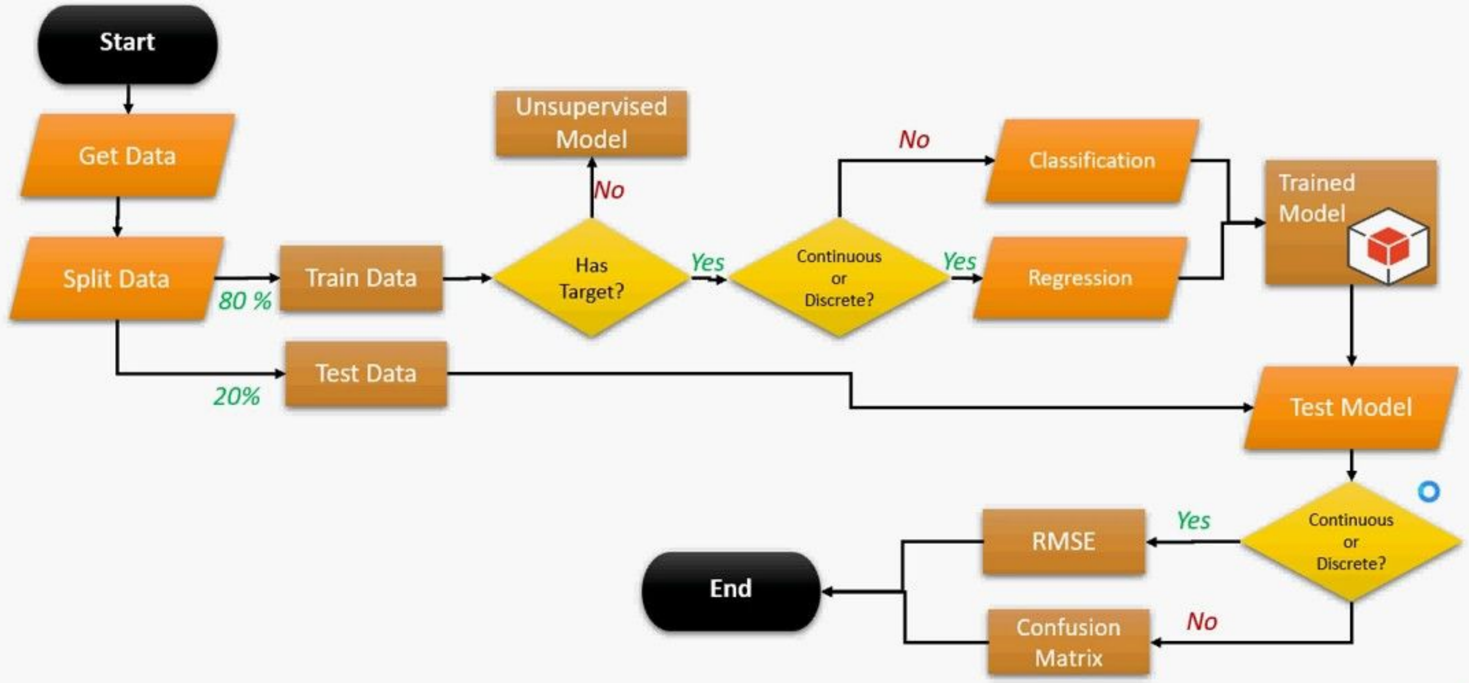
The Machine Learning Pipeline (workflow)

Marc Berghouse and Lazaro Perez

The Machine Learning Pipeline

- The pipeline is composed of all of the computation and analysis required to get from the beginning to the end of a machine learning project
- Gather data
- Clean and preprocess data
- Exploratory data analysis
- Split data into training, validation and testing sets (holdout vs cross-validation)
- Develop/import model
- Train model
- Test model
- Hyperparameter tuning
- Report and visualize results

Machine Learning – Model Flowchart



https://www.youtube.com/watch?v=1wVgtINZIT4

Gather Data

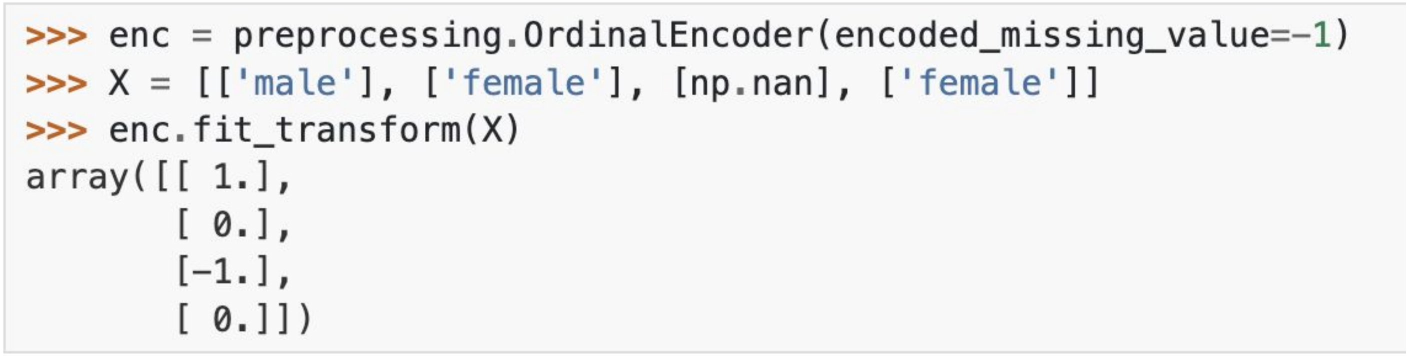
- We need data
- Good data is hard to find
  - No cure all method to easily finding and extracting data
  - Be persistent
- Data almost always needs to be cleaned and preprocessed
  - Think about, how can I make this data look like an Excel spreadsheet/Pandas dataframe
- Generally try to read in data with pd.read_csv()

Clean and Preprocess data

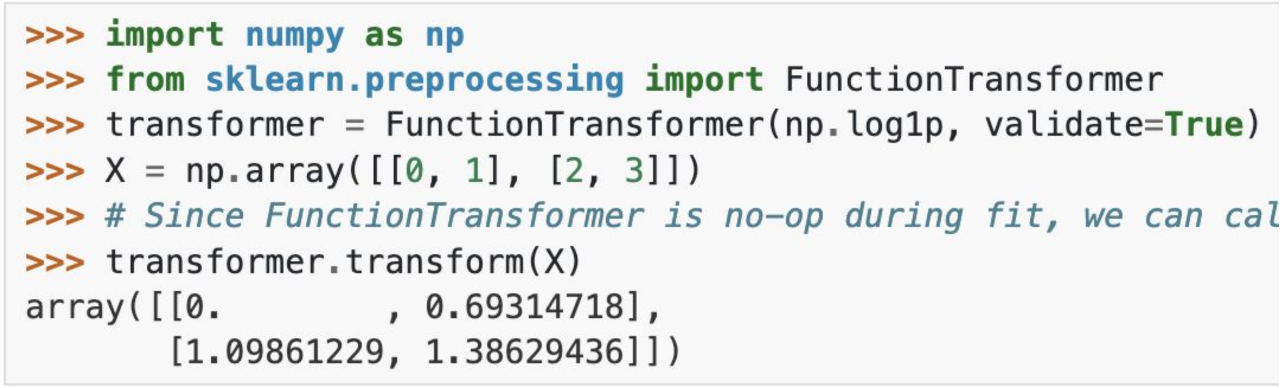
- Read data and parse into a pandas dataframe or numpy array
- Take care of NAN values and outliers
  - Imputation or dropping
  - NaN values propagate through calculations
  - Outlier detection (kNN, support vector machine, covariance)
- Use sklearn’s StandardScalar() to scale your data to mean of 0 with a standard deviation of 1
- Make necessary data transformations
  - For dates, pd.to_datetime() should work, but I’ve had issues with it
  - You can also count number of days or hours from the beginning of the dataset and use this as the date/time variable
  - Units should be in the same system (kg, m, s; lb, ft, s)
  - Encode string-type variables
  - Transformations should be done before using StandardScalar()

Encoding

- Convert categorical variables (strings) to numbers

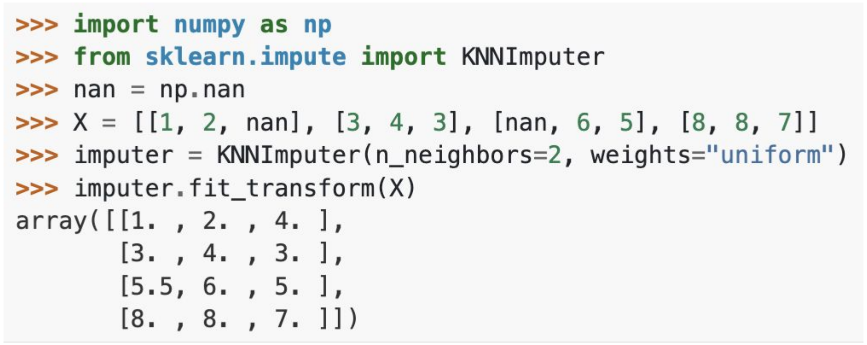


```
>>> enc = preprocessing.OrdinalEncoder(encoded_missing_value=-1)
>>> X = [['male'], ['female'], [np.nan], ['female']]
>>> enc.fit_transform(X)
array([[ 1.],
       [ 0.],
       [-1.],
       [ 0.]])
```

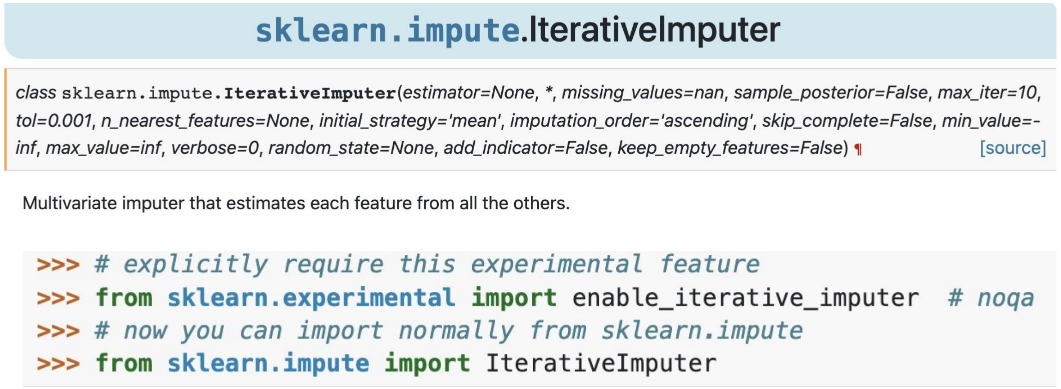


```
>>> import numpy as np
>>> from sklearn.preprocessing import FunctionTransformer
>>> transformer = FunctionTransformer(np.log1p, validate=True)
>>> X = np.array([[0, 1], [2, 3]])
>>> # Since FunctionTransformer is no-op during fit, we can cal
>>> transformer.transform(X)
array([[0.        , 0.69314718],
       [1.09861229, 1.38629436]])
```

Fill Missing or NaN Values (sklearn)



```
>>> import numpy as np
>>> from sklearn.impute import KNNImputer
>>> nan = np.nan
>>> X = [[1, 2, nan], [3, 4, 3], [nan, 6, 5], [8, 8, 7]]
>>> imputer = KNNImputer(n_neighbors=2, weights="uniform")
>>> imputer.fit_transform(X)
array([[1. , 2. , 4. ],
       [3. , 4. , 3. ],
       [5.5, 6. , 5. ],
       [8. , 8. , 7. ]])
```

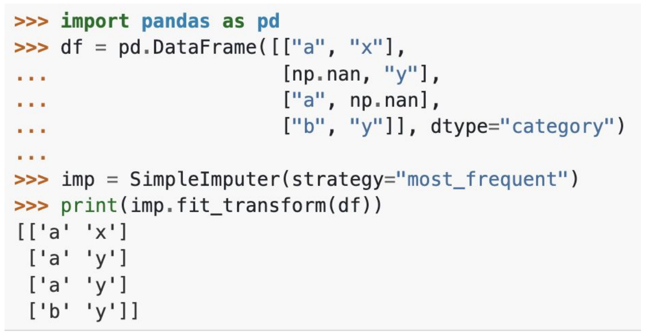


sklearn.impute.IterativeImputer

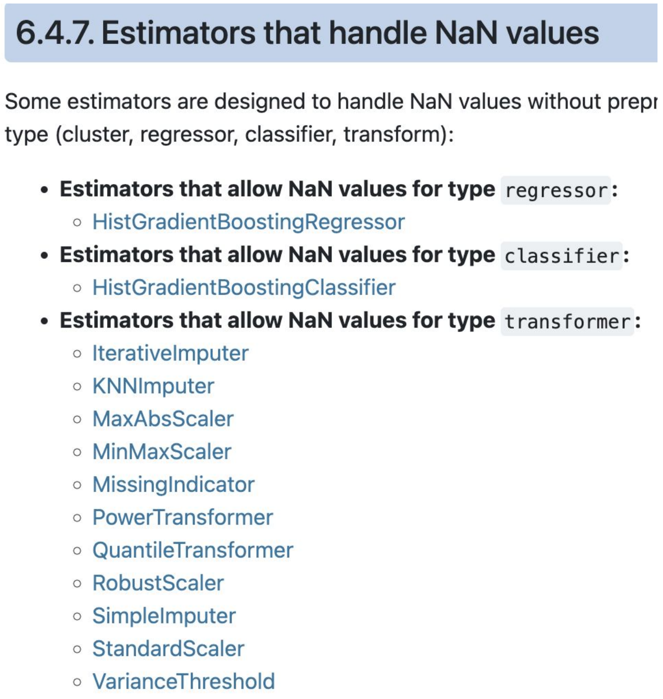
class sklearn.impute.**IterativeImputer**(*estimator=None, *, missing_values=nan, sample_posterior=False, max_iter=10, tol=0.001, n_nearest_features=None, initial_strategy='mean', imputation_order='ascending', skip_complete=False, min_value=-inf, max_value=inf, verbose=0, random_state=None, add_indicator=False, keep_empty_features=False*) ¶ [source]

Multivariate imputer that estimates each feature from all the others.

```
>>> # explicitly require this experimental feature
>>> from sklearn.experimental import enable_iterative_imputer  # noqa
>>> # now you can import normally from sklearn.impute
>>> from sklearn.impute import IterativeImputer
```



```
>>> import pandas as pd
>>> df = pd.DataFrame([["a", "x"],
...                    [np.nan, "y"],
...                    ["a", np.nan],
...                    ["b", "y"]], dtype="category")
...
>>> imp = SimpleImputer(strategy="most_frequent")
>>> print(imp.fit_transform(df))
[['a' 'x']
 ['a' 'y']
 ['a' 'y']
 ['b' 'y']]
```

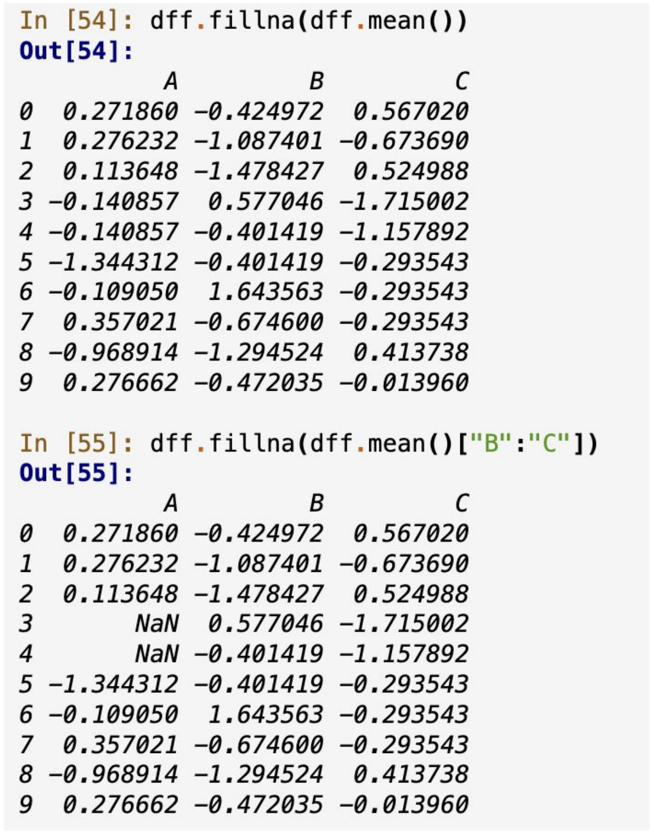


6.4.7. Estimators that handle NaN values

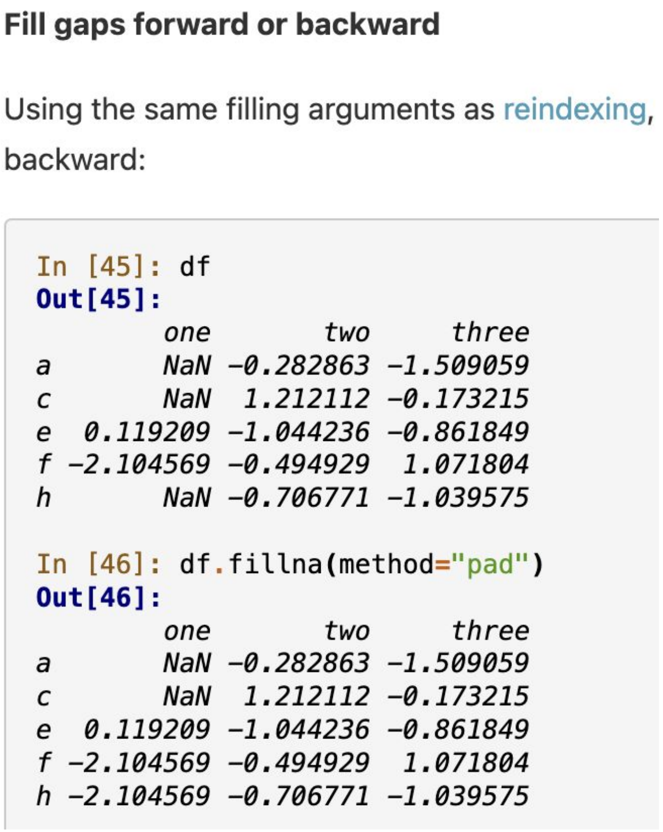
Some estimators are designed to handle NaN values without prepr
type (cluster, regressor, classifier, transform):

- **Estimators that allow NaN values for type** `regressor`**:**
  - HistGradientBoostingRegressor
- **Estimators that allow NaN values for type** `classifier`**:**
  - HistGradientBoostingClassifier
- **Estimators that allow NaN values for type** `transformer`**:**
  - IterativeImputer
  - KNNImputer
  - MaxAbsScaler
  - MinMaxScaler
  - MissingIndicator
  - PowerTransformer
  - QuantileTransformer
  - RobustScaler
  - SimpleImputer
  - StandardScaler
  - VarianceThreshold

Fill Missing or NaN Values (Pandas)



```
In [54]: dff.fillna(dff.mean())
Out[54]:
          A         B         C
0  0.271860 -0.424972  0.567020
1  0.276232 -1.087401 -0.673690
2  0.113648 -1.478427  0.524988
3 -0.140857  0.577046 -1.715002
4 -0.140857 -0.401419 -1.157892
5 -1.344312 -0.401419 -0.293543
6 -0.109050  1.643563 -0.293543
7  0.357021 -0.674600 -0.293543
8 -0.968914 -1.294524  0.413738
9  0.276662 -0.472035 -0.013960

In [55]: dff.fillna(dff.mean()["B":"C"])
Out[55]:
          A         B         C
0  0.271860 -0.424972  0.567020
1  0.276232 -1.087401 -0.673690
2  0.113648 -1.478427  0.524988
3       NaN  0.577046 -1.715002
4       NaN -0.401419 -1.157892
5 -1.344312 -0.401419 -0.293543
6 -0.109050  1.643563 -0.293543
7  0.357021 -0.674600 -0.293543
8 -0.968914 -1.294524  0.413738
9  0.276662 -0.472035 -0.013960
```

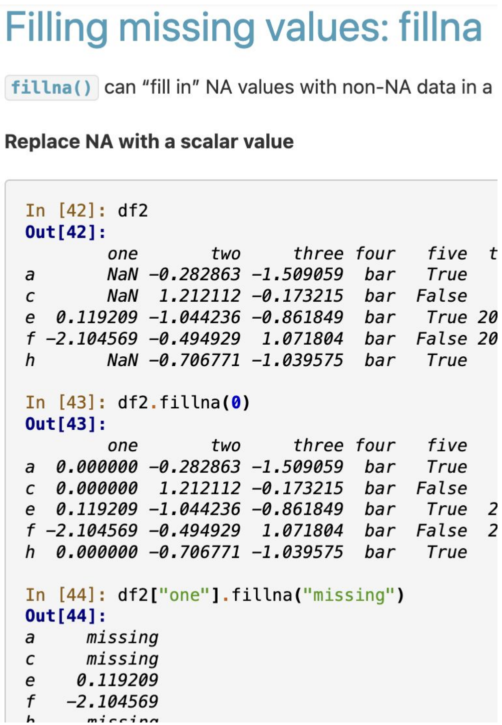


Fill gaps forward or backward

Using the same filling arguments as reindexing,

backward:

```
In [45]: df
Out[45]:
        one       two     three
a       NaN -0.282863 -1.509059
c       NaN  1.212112 -0.173215
e  0.119209 -1.044236 -0.861849
f -2.104569 -0.494929  1.071804
h       NaN -0.706771 -1.039575

In [46]: df.fillna(method="pad")
Out[46]:
        one       two     three
a       NaN -0.282863 -1.509059
c       NaN  1.212112 -0.173215
e  0.119209 -1.044236 -0.861849
f -2.104569 -0.494929  1.071804
h -2.104569 -0.706771 -1.039575
```



Filling missing values: fillna

`fillna()` can “fill in” NA values with non-NA data in a

Replace NA with a scalar value

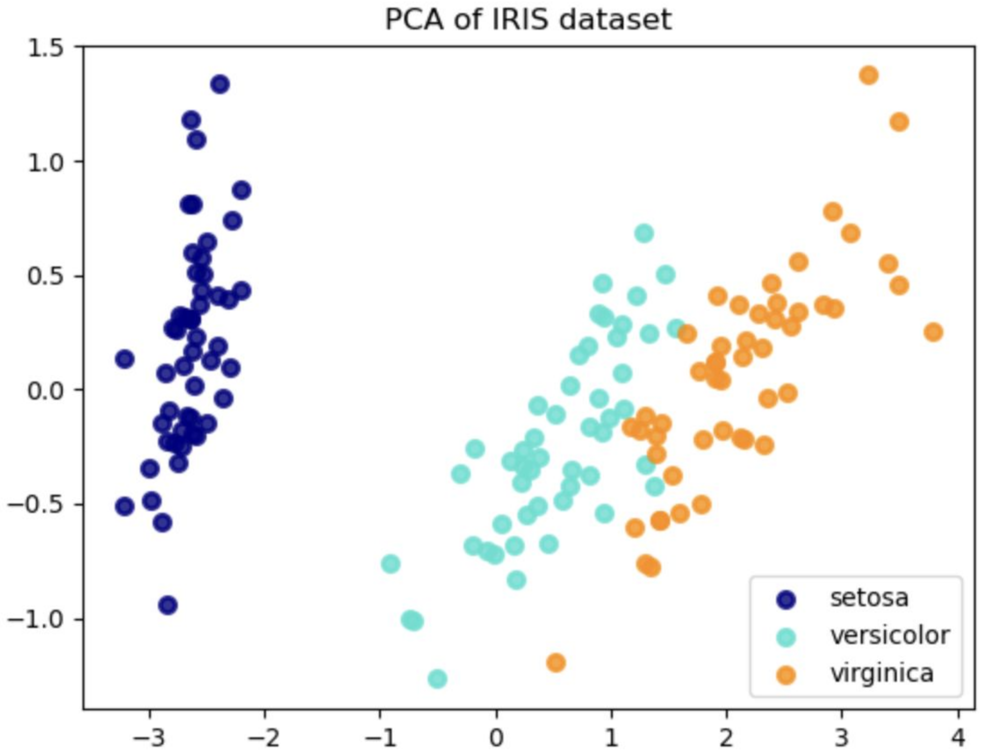
```
In [42]: df2
Out[42]:
        one       two     three four   five  t
a       NaN -0.282863 -1.509059  bar   True
c       NaN  1.212112 -0.173215  bar  False
e  0.119209 -1.044236 -0.861849  bar   True 20
f -2.104569 -0.494929  1.071804  bar  False 20
h       NaN -0.706771 -1.039575  bar   True

In [43]: df2.fillna(0)
Out[43]:
        one       two     three four   five
a  0.000000 -0.282863 -1.509059  bar   True
c  0.000000  1.212112 -0.173215  bar  False
e  0.119209 -1.044236 -0.861849  bar   True  2
f -2.104569 -0.494929  1.071804  bar  False  2
h  0.000000 -0.706771 -1.039575  bar   True

In [44]: df2["one"].fillna("missing")
Out[44]:
a      missing
c      missing
e     0.119209
f    -2.104569
h      missing
```

Exploratory Data Analysis

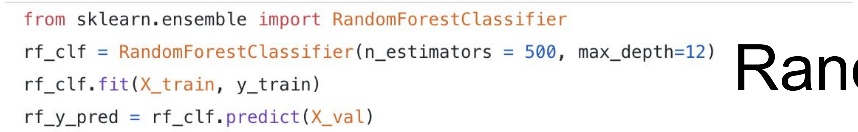
- Understand and communicate what is going on with your data at a general level
- Pandas profiler
    - Basic stats for whole dataframe and individual variables
    - Scatter plots and correlation heatmap of all variables
    - If you can’t install for whatever reason, you can use matplotlib to generate the same plots
- Feature Analysis
    - Unsupervised methods
        - Principal Component Analysis (PCA)
        - Outlier detection
    - Supervised methods
        - Linear Discriminant Analysis (LDA)
        - Random Forest
    - PCA, LDA, and factor analysis all look for linear combinations of variables that best explain the data

Principal Component Analysis (PCA)

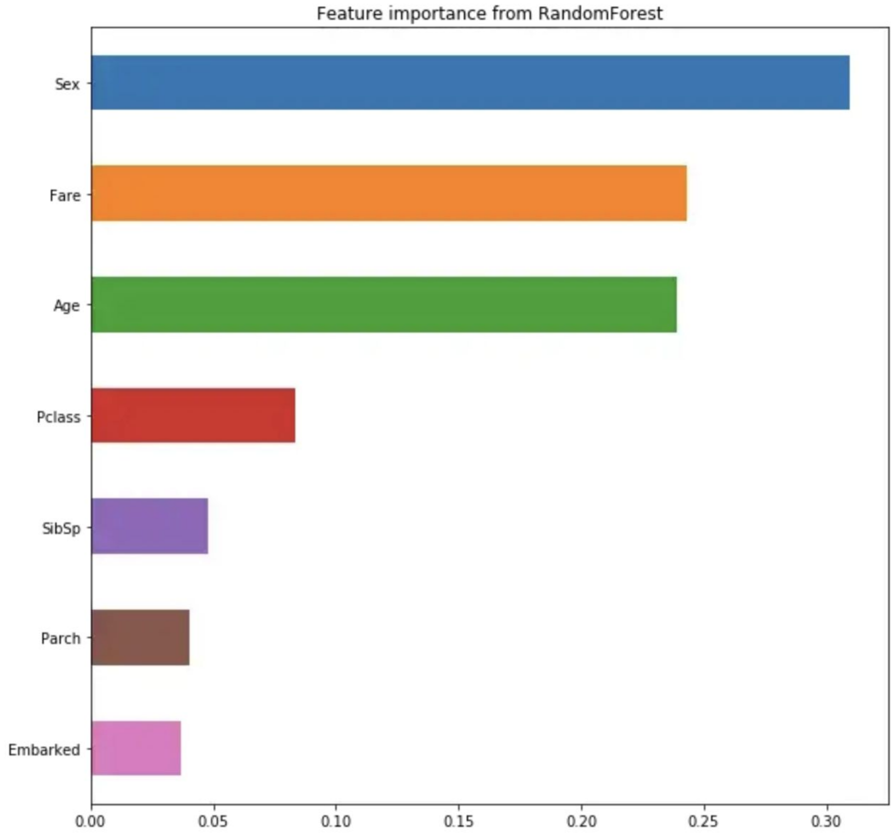
- “Linear dimensionality reduction using Singular Value Decompositio of the data to project it to a lower dimensional space”
- Separates data into new dimension based on the **combinations** of attributes that account for most of the variance in the data (explanator power)
- **LDA** is like PCA but it looks for attributes that explain the most **variance between classes**



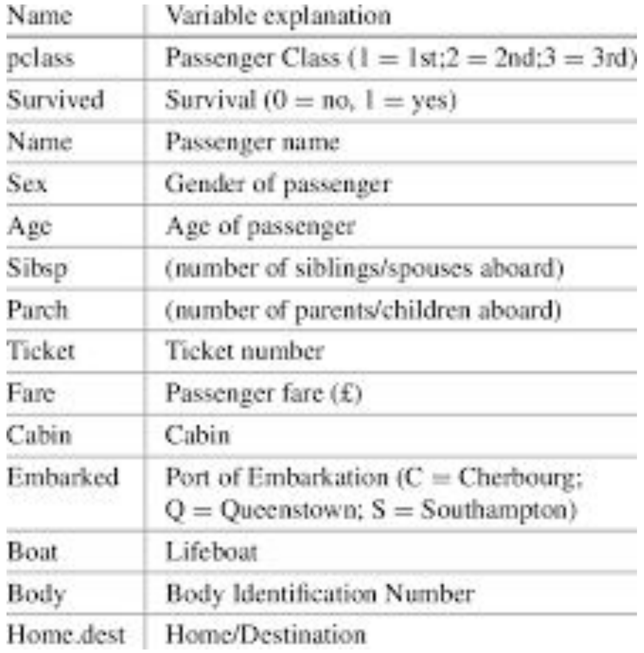
Random Forest Feature Engineering



```
from sklearn.ensemble import RandomForestClassifier
rf_clf = RandomForestClassifier(n_estimators = 500, max_depth=12)
rf_clf.fit(X_train, y_train)
rf_y_pred = rf_clf.predict(X_val)
```



Titanic Dataset

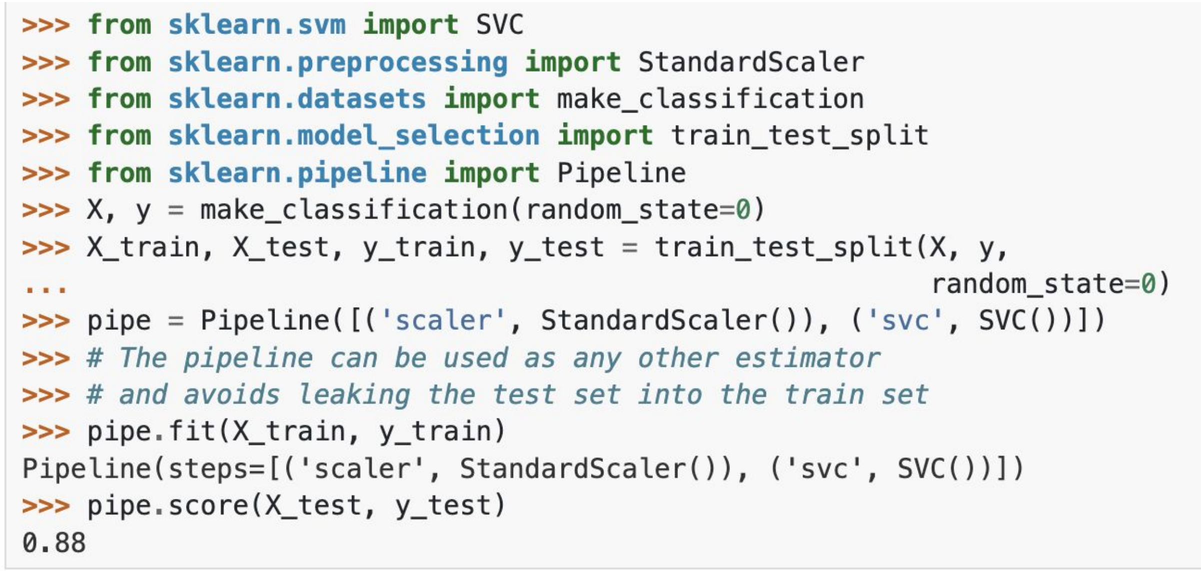


Name	Variable explanation
pclass	Passenger Class (1 = 1st;2 = 2nd;3 = 3rd)
Survived	Survival (0 = no, 1 = yes)
Name	Passenger name
Sex	Gender of passenger
Age	Age of passenger
Sibsp	(number of siblings/spouses aboard)
Parch	(number of parents/children aboard)
Ticket	Ticket number
Fare	Passenger fare (£)
Cabin	Cabin
Embarked	Port of Embarkation (C = Cherbourg; Q = Queenstown; S = Southampton)
Boat	Lifeboat
Body	Body Identification Number
Home.dest	Home/Destination

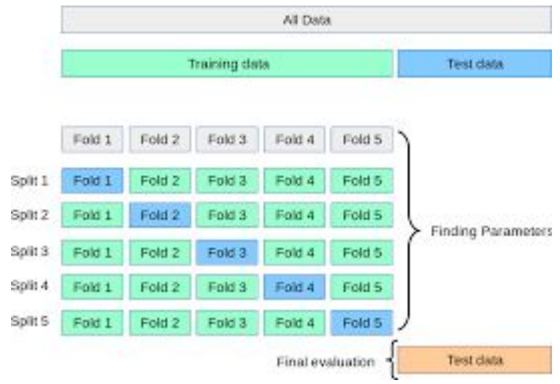
https://towardsdatascience.com/a-starter-pack-to-exploratory-data-analysis-with-python-pandas-seaborn-and-scikit-learn-a77889485baf

Training and testing splits

- We use training data to fit the model
- We use testing data (unseen during training) to test the model
  - Data leakage happens when the model somehow has access to testing samples during training
- Holdout
  - Only one train-test split
- Cross-Validation (CV)
  - Divide dataset into 6 parts
  - Train on five parts, test on 6th
  - Switch training/testing parts until all pairs are complete



```
>>> from sklearn.svm import SVC
>>> from sklearn.preprocessing import StandardScaler
>>> from sklearn.datasets import make_classification
>>> from sklearn.model_selection import train_test_split
>>> from sklearn.pipeline import Pipeline
>>> X, y = make_classification(random_state=0)
>>> X_train, X_test, y_train, y_test = train_test_split(X, y,
...                                                     random_state=0)
>>> pipe = Pipeline([('scaler', StandardScaler()), ('svc', SVC())])
>>> # The pipeline can be used as any other estimator
>>> # and avoids leaking the test set into the train set
>>> pipe.fit(X_train, y_train)
Pipeline(steps=[('scaler', StandardScaler()), ('svc', SVC())])
>>> pipe.score(X_test, y_test)
0.88
```

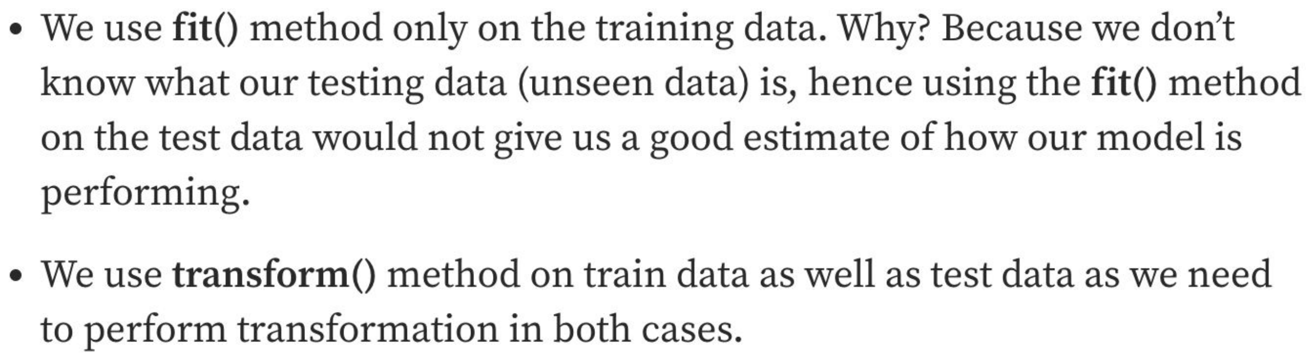


Model Development

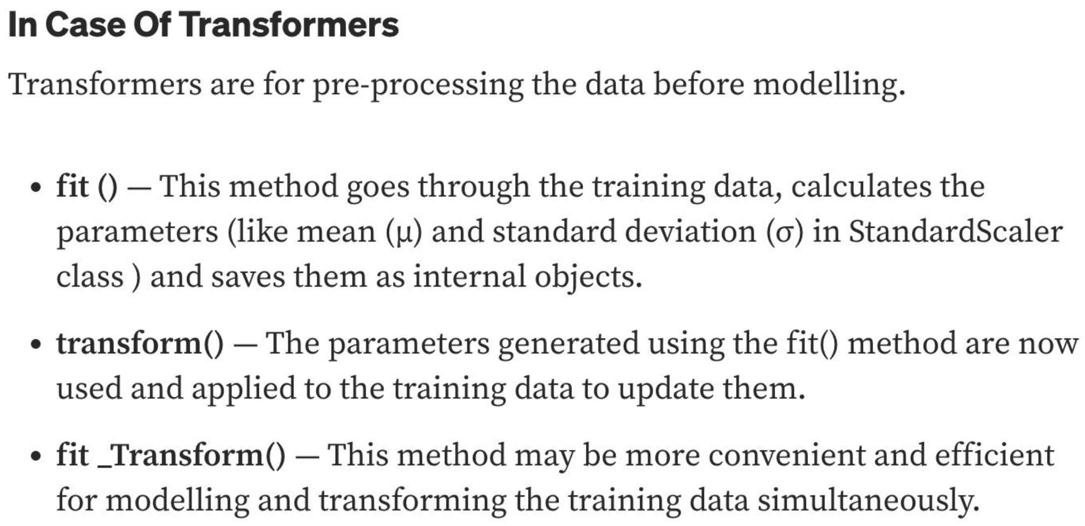
- Choose the appropriate model to solve the problem
- Whole lecture on this later
- Classification or Regression?
- Supervised, unsupervised or reinforcement learning?
    - https://scikit-learn.org/stable/supervised_learning.html
    - https://scikit-learn.org/stable/unsupervised_learning.html
- Read the Sklearn documentation for the model you pick to make sure you are using it in the right situation and passing the correct arguments to the model function
- Hyperparameter tuning (more on this later)
    - https://scikit-learn.org/stable/modules/grid_search.html

Model Training

We use the .fit() method for all training



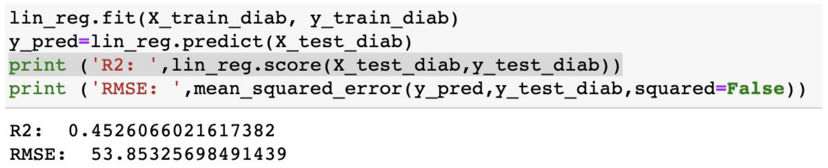
- We use **fit()** method only on the training data. Why? Because we don't know what our testing data (unseen data) is, hence using the **fit()** method on the test data would not give us a good estimate of how our model is performing.
- We use **transform()** method on train data as well as test data as we need to perform transformation in both cases.



In Case Of Transformers

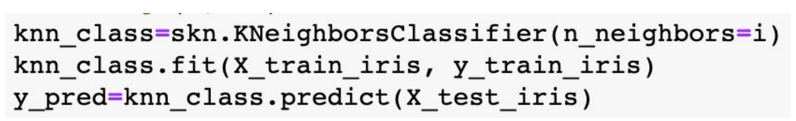
Transformers are for pre-processing the data before modelling.

- **fit ()** — This method goes through the training data, calculates the parameters (like mean (μ) and standard deviation (σ) in StandardScaler class) and saves them as internal objects.
- **transform()** — The parameters generated using the fit() method are now used and applied to the training data to update them.
- **fit _Transform()** — This method may be more convenient and efficient for modelling and transforming the training data simultaneously.



```
lin_reg.fit(X_train_diab, y_train_diab)
y_pred=lin_reg.predict(X_test_diab)
print ('R2: ',lin_reg.score(X_test_diab,y_test_diab))
print ('RMSE: ',mean_squared_error(y_pred,y_test_diab,squared=False))
```

```
R2:  0.4526066021617382
RMSE:  53.85325698491439
```

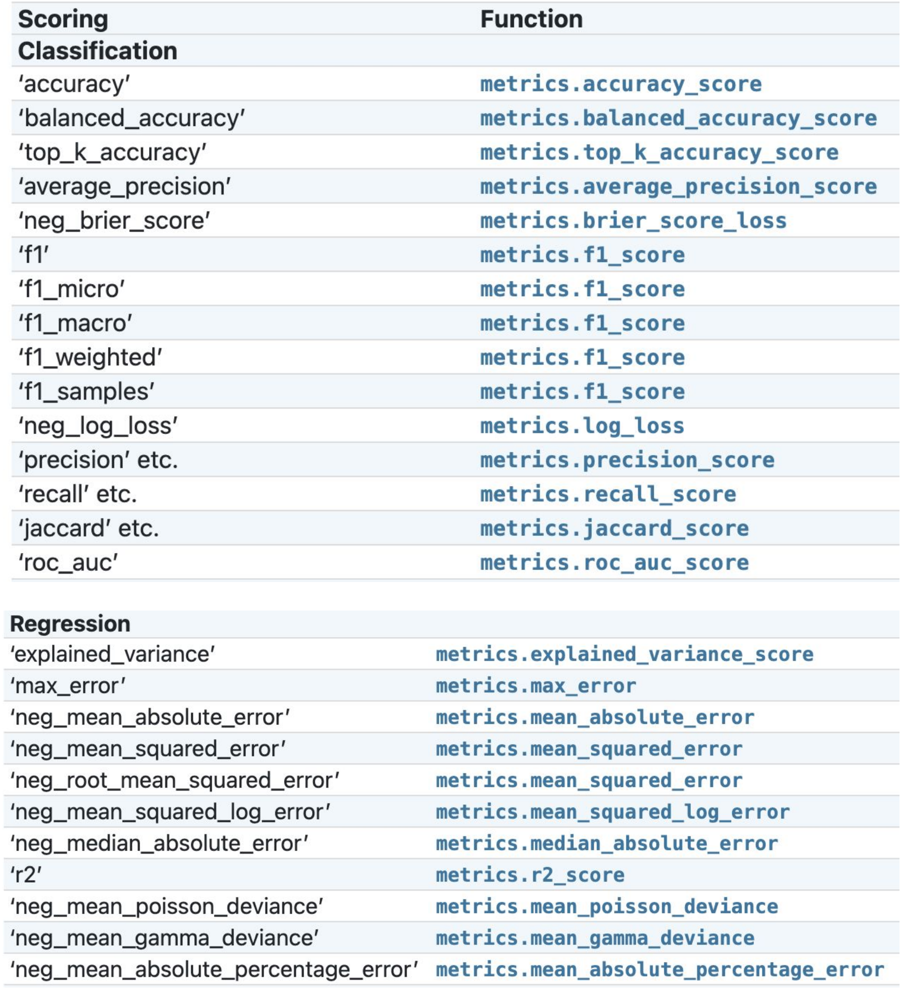


```
knn_class=skn.KNeighborsClassifier(n_neighbors=i)
knn_class.fit(X_train_iris, y_train_iris)
y_pred=knn_class.predict(X_test_iris)
```

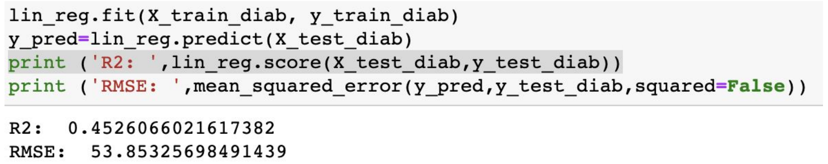
https://medium.com/nerd-for-tech/difference-fit-transform-and-fit-transform-method-in-scikit-learn-b0a4efcab804

Model Testing/Scoring

- We use the .predict() method to generate predictions of our testing data
- If we use .predict(), we then have to generate a score by comparing the predicted data (“y_pred”) with the true data (“y_test”)
- If we use .score(), Sklearn does the prediction and score generation together
- The .score() method uses the arguments (x_test,y_test), but if you call one of the specific scoring functions on the right, the arguments are (y_pred,y_test)



Scoring	Function
Classification	
‘accuracy’	`metrics.accuracy_score`
‘balanced_accuracy’	`metrics.balanced_accuracy_score`
‘top_k_accuracy’	`metrics.top_k_accuracy_score`
‘average_precision’	`metrics.average_precision_score`
‘neg_brier_score’	`metrics.brier_score_loss`
‘f1’	`metrics.f1_score`
‘f1_micro’	`metrics.f1_score`
‘f1_macro’	`metrics.f1_score`
‘f1_weighted’	`metrics.f1_score`
‘f1_samples’	`metrics.f1_score`
‘neg_log_loss’	`metrics.log_loss`
‘precision’ etc.	`metrics.precision_score`
‘recall’ etc.	`metrics.recall_score`
‘jaccard’ etc.	`metrics.jaccard_score`
‘roc_auc’	`metrics.roc_auc_score`
Regression	
‘explained_variance’	`metrics.explained_variance_score`
‘max_error’	`metrics.max_error`
‘neg_mean_absolute_error’	`metrics.mean_absolute_error`
‘neg_mean_squared_error’	`metrics.mean_squared_error`
‘neg_root_mean_squared_error’	`metrics.mean_squared_error`
‘neg_mean_squared_log_error’	`metrics.mean_squared_log_error`
‘neg_median_absolute_error’	`metrics.median_absolute_error`
‘r2’	`metrics.r2_score`
‘neg_mean_poisson_deviance’	`metrics.mean_poisson_deviance`
‘neg_mean_gamma_deviance’	`metrics.mean_gamma_deviance`
‘neg_mean_absolute_percentage_error’	`metrics.mean_absolute_percentage_error`



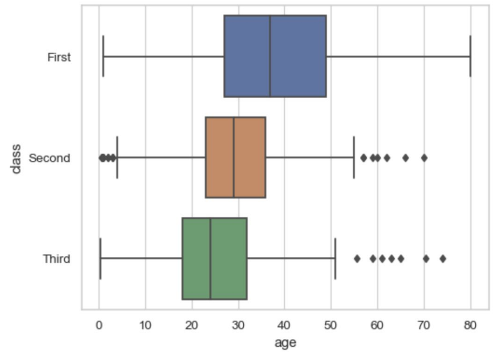
```
lin_reg.fit(X_train_diab, y_train_diab)
y_pred=lin_reg.predict(X_test_diab)
print ('R2: ',lin_reg.score(X_test_diab,y_test_diab))
print ('RMSE: ',mean_squared_error(y_pred,y_test_diab,squared=False))
```

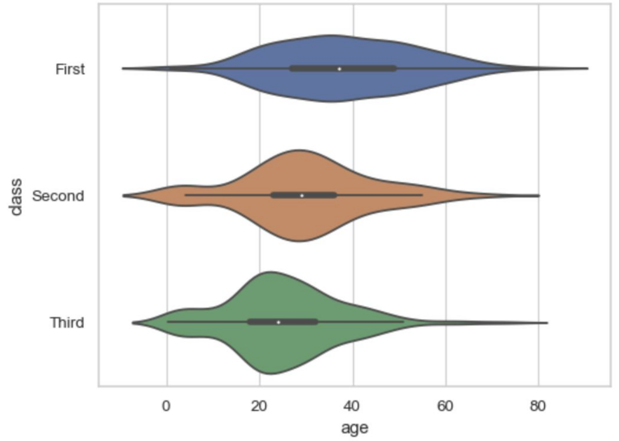
```
R2:  0.4526066021617382
RMSE:  53.85325698491439
```

Report and Visualize Results (Regression)

- Regression
  - Scatter plots with line of best fit
  - For multiple linear regression, you can use 3D plots (x,y, and z coordinates), color, and scatter point size to add extra dimensions to the plot
  - Correlation matrix/heatmap (already done with Pandas profiler)
  - Box or violin plots of scores for different models/experiments/runs
    - Test different models under a range of changing variables/parameters (range of scores for each model)
    - Test the range of results for probabilistic models (different result after each run, like with neural nets)

```
sns.boxplot(data=df, x="age", y="class")
```

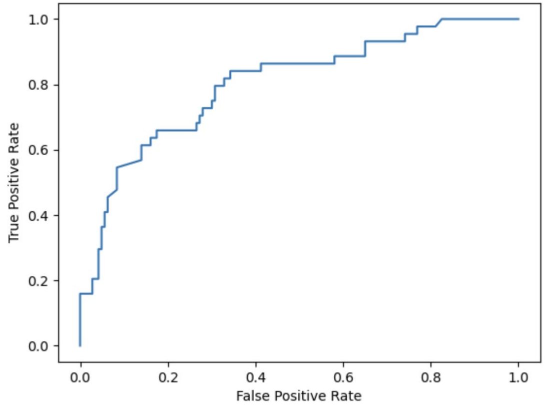


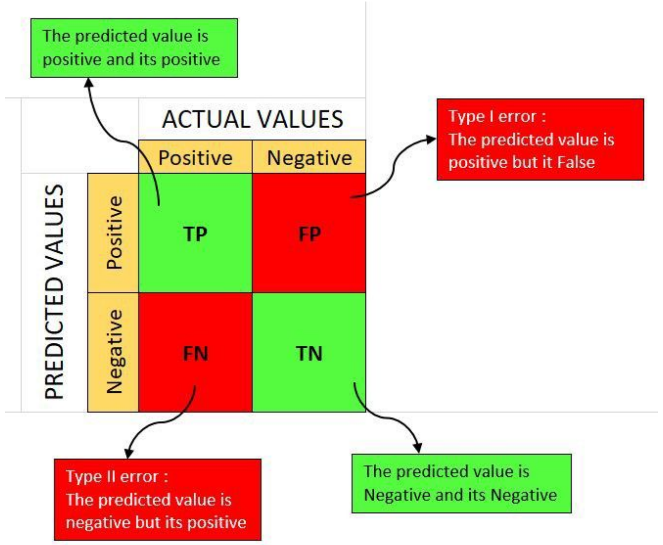


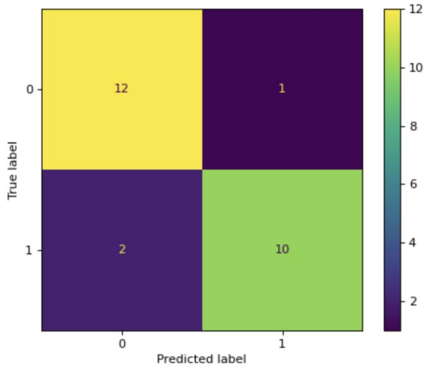
Report and Visualize Results (Classification)

- Confusion Matrix
- ROC-AUC curve
  - False positive rate (FPR) vs true positive rate (TPR)

```python
from sklearn.metrics import roc_curve
from sklearn.metrics import RocCurveDisplay

y_score = clf.decision_function(X_test)

fpr, tpr, _ = roc_curve(y_test, y_score, pos_label=clf.classes_[1])
roc_display = RocCurveDisplay(fpr=fpr, tpr=tpr).plot()
```

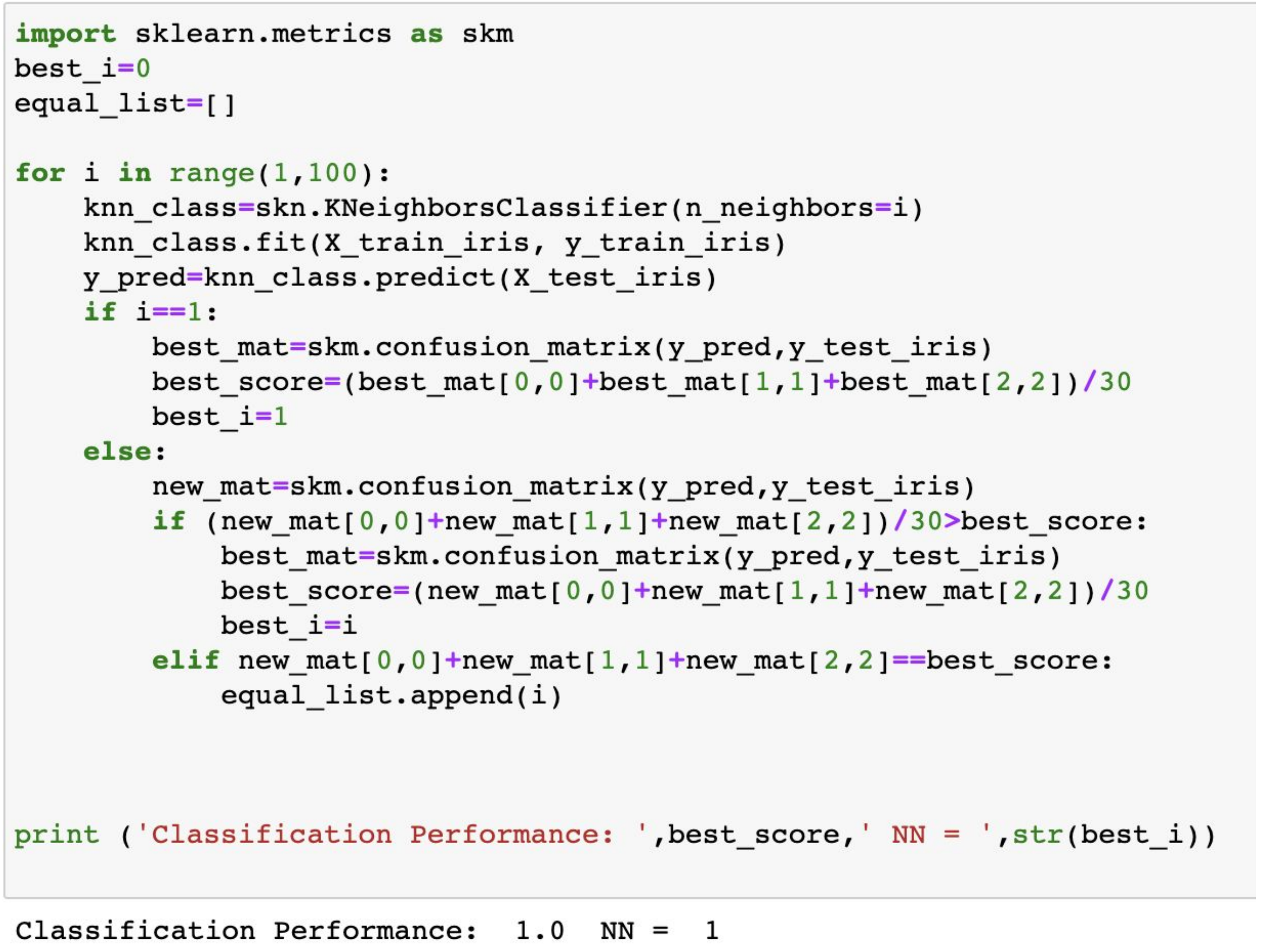



The predicted value is positive and its positive
ACTUAL VALUES
Positive
Negative
PREDICTED VALUES
Positive
Negative
TP
FP
FN
TN
Type I error : The predicted value is positive but it False
Type II error : The predicted value is negative but its positive
The predicted value is Negative and its Negative



https://medium.com/analytics-vidhya/what-is-a-confusion-matrix-d1c0f8feda5

Example Walkthrough of Iris dataset confusion matrix and best nearest neighbors searcher



```python
import sklearn.metrics as skm
best_i=0
equal_list=[]

for i in range(1,100):
    knn_class=skn.KNeighborsClassifier(n_neighbors=i)
    knn_class.fit(X_train_iris, y_train_iris)
    y_pred=knn_class.predict(X_test_iris)
    if i==1:
        best_mat=skm.confusion_matrix(y_pred,y_test_iris)
        best_score=(best_mat[0,0]+best_mat[1,1]+best_mat[2,2])/30
        best_i=1
    else:
        new_mat=skm.confusion_matrix(y_pred,y_test_iris)
        if (new_mat[0,0]+new_mat[1,1]+new_mat[2,2])/30>best_score:
            best_mat=skm.confusion_matrix(y_pred,y_test_iris)
            best_score=(new_mat[0,0]+new_mat[1,1]+new_mat[2,2])/30
            best_i=i
        elif new_mat[0,0]+new_mat[1,1]+new_mat[2,2]==best_score:
            equal_list.append(i)



print ('Classification Performance: ',best_score,' NN = ',str(best_i))
```

```
Classification Performance:  1.0  NN =  1
```