¿A que voy yo y lo encuentro…? Pipelines en Scikit-Learn

Irene Rodríguez Luján

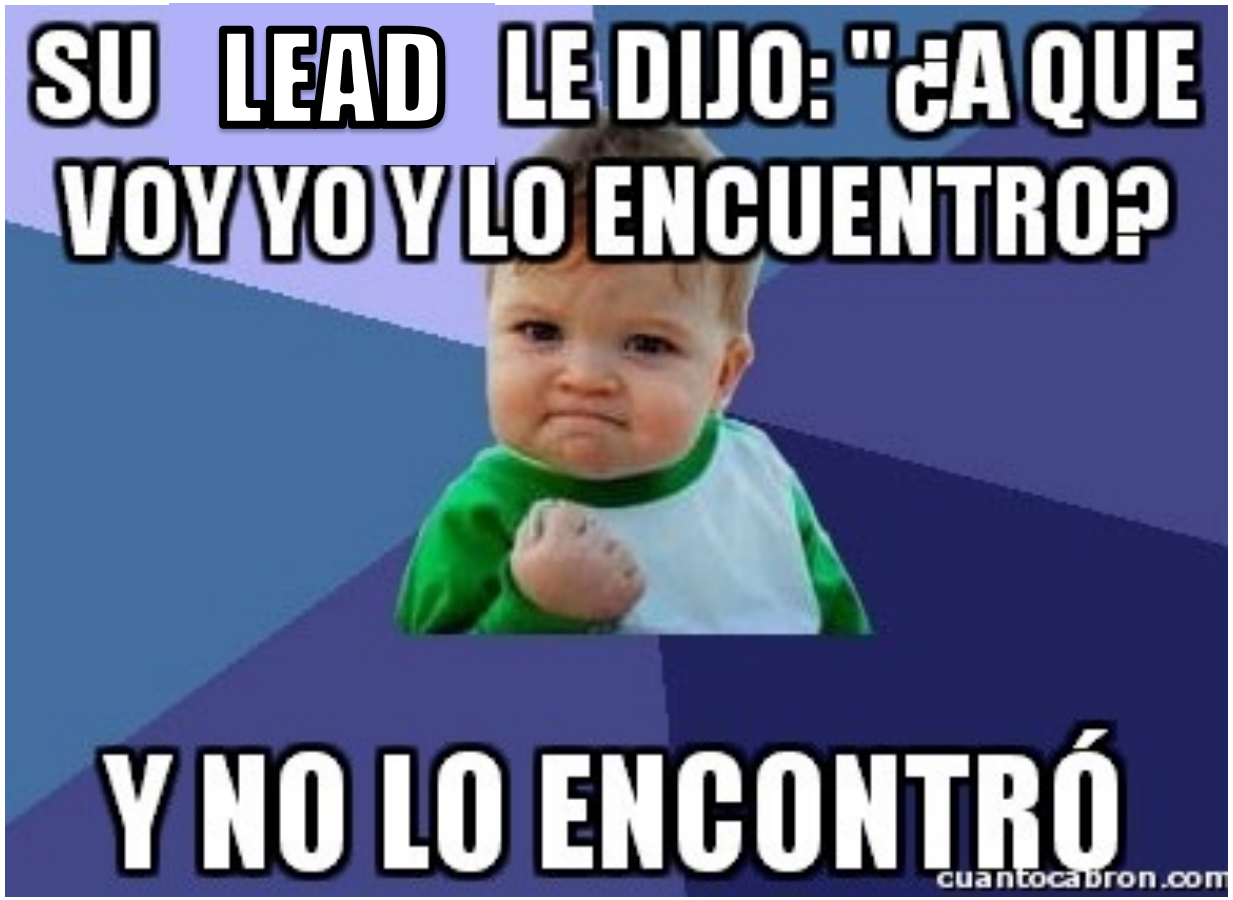
www.linkedin.com/in/irenerodluj

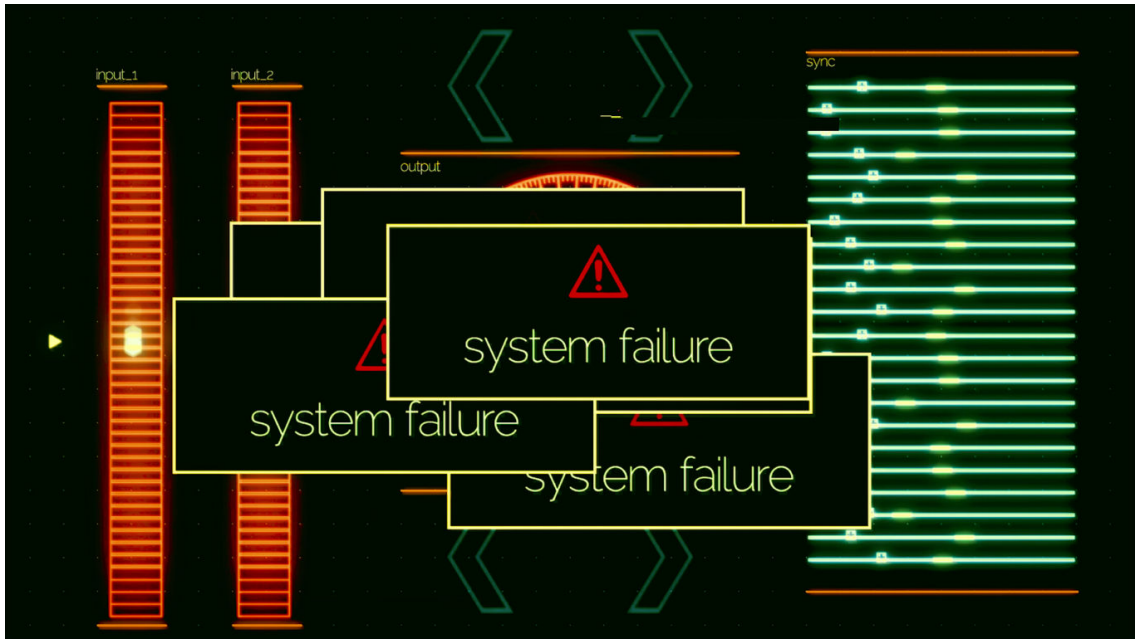
[Coming soon] https://github.com/irenerodriguez/pydata-pipelines

Hola!

AXA Openbank affirm

verti Afi Escuela


SU LEAD LE DIJO: "¿A QUE VOY YO Y LO ENCUENTRO?
Y NO LO ENCONTRÓ
cuantocabron.com


input_1
input_2
output
sync
system failure
system failure
system failure

DANGEROUS CODE

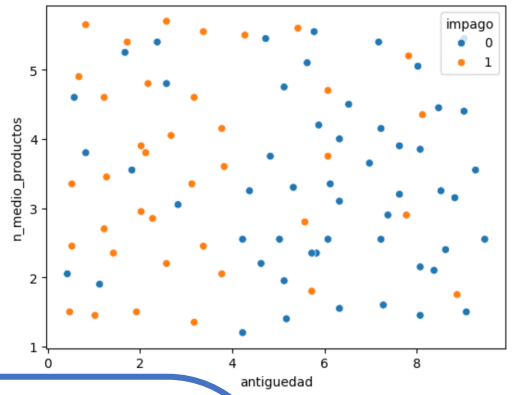
ENTER AT YOUR OWN RISK

Scikit-learn 1.0.2

Round 1

Necesito clasificar a mis clientes en función a su riesgo de impago y en base a su antigüedad como cliente y el número medio de productos al año.

	antiguedad	n_medio_productos	impago
0	4.225	1.20	0
1	5.175	1.40	0
2	6.325	1.55	0
3	7.275	1.60	0
4	8.075	1.45	0



¡Por supuesto! Voy a usar k-NN que el problema parece sencillito.
Yo ya sé que en el caso de k-NN tengo que **normalizar los datos** e **imputar missing values**.
También voy a dividir mis datos en **training y test** y sé que el conjunto de test solo lo puedo utilizar para evaluar el resultado final de mi modelo.
¡Manos a la obra!

https://guoguibing.github.io/librec/datagen.html

Scikit-learn 1.0.2

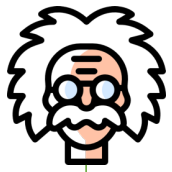
Round 1

```python
from sklearn.model_selection import train_test_split

X = data.drop(columns='impago').copy()
y = data['impago'].copy()

# SET a random_state to make your pipeline reproducible!
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

ML Engineer

```python
## TRAIN
from sklearn.impute import SimpleImputer
from plot_decision_boundary import plot_decision_boundary
from sklearn.metrics import accuracy_score

ss = StandardScaler(with_mean=True, with_std=True)
X_train_transform_ss = ss.fit_transform(X_train)

si = SimpleImputer(strategy='mean')
X_train_transform_ss_si = si.fit_transform(X_train_transform_ss)

clf = KNeighborsClassifier(n_neighbors=1)

clf.fit(X_train_transform_ss_si, y_train)

plot_decision_boundary(clf, X_train_transform_ss_si, y_train)

print("Accuracy in train: ", accuracy_score(y_train, clf.predict(X_train_transform_ss_si)))


# DISCLAIMER: model registry
with open(os.path.join('pkl','round_1','standard_scaler.pkl'),'wb') as f:
    pickle.dump(ss,f)

with open(os.path.join('pkl','round_1','simple_imputer.pkl'),'wb') as f:
    pickle.dump(si,f)

with open(os.path.join('pkl','round_1','knn.pkl'),'wb') as f:
    pickle.dump(clf,f)
```

```
Accuracy in train:  1.0
```

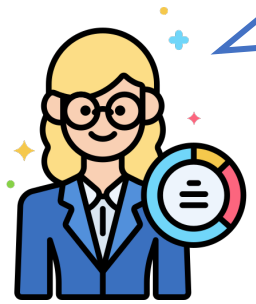


```python
# load
with open(os.path.join('pkl','round_1','standard_scaler.pkl'), 'rb') as f:
    ss = pickle.load(f)

with open(os.path.join('pkl','round_1','simple_imputer.pkl'), 'rb') as f:
    si = pickle.load(f)

with open(os.path.join('pkl','round_1','knn.pkl'), 'rb') as f:
    clf = pickle.load(f)

X_test_transf_ss = ss.transform(X_test)
X_test_transf_ss_si = si.transform(X_test_transf_ss)
prediction = clf.predict(X_test_transf_ss_si)

print("Accuracy in test: ", accuracy_score(y_test, prediction))
```

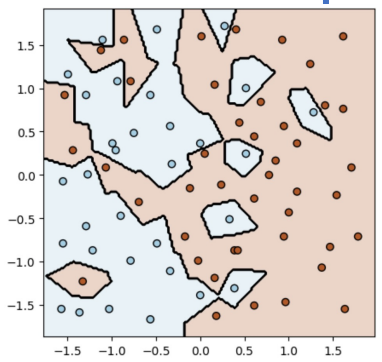
```
Accuracy in test:  0.6
```

Scikit-learn 1.0.2

Round 2



Está claro que poner en número de vecinos a 1 no ha sido buena idea… Voy a probar a poner 3 vecinos para **reducir el overfitting**…




```python
ss = StandardScaler(with_mean=True, with_std=True)
X_train_transform_ss = ss.fit_transform(X_train)

si = SimpleImputer(strategy='mean')
X_train_transform_ss_si = si.fit_transform(X_train_transform_ss)

clf_1 = KNeighborsClassifier(n_neighbors=1)
clf_1.fit(X_train_transform_ss_si, y_train)

clf_3 = KNeighborsClassifier(n_neighbors=3)
clf_3.fit(X_train_transform_ss_si, y_train)
```

```python
with open(os.path.join('pkl','round_2','standard_scaler.pkl'), 'rb') as f:
    ss = pickle.load(f)

with open(os.path.join('pkl','round_2','simple_imputer.pkl'), 'rb') as f:
    si = pickle.load(f)

with open(os.path.join('pkl','round_2','knn_1.pkl'), 'rb') as f:
    clf_1 = pickle.load(f)

with open(os.path.join('pkl','round_2','knn_3.pkl'), 'rb') as f:
    clf_3 = pickle.load(f)

X_test_transf_ss = ss.transform(X_test)
X_test_transf_ss_si = si.transform(X_test_transf_ss)

print("Accuracy in test: ", accuracy_score(y_test, clf_1.predict(X_test_transf_ss_si)))
print("Accuracy in test: ", accuracy_score(y_test, clf_3.predict(X_test_transf_ss_si)))
```

```
Accuracy in test:  0.6
Accuracy in test:  0.7
```

¡Me quedo con 3 vecinos!

Recuerda que no puedes usar el conjunto de test para tomar decisiones, solo para evaluar el modelo final…

Round 3

Scikit-learn 1.0.2

¡Cierto! Estaba el tema este de la **validación cruzada**… puedo decidir el número de vecinos sin mirar al conjunto de test…

	param_n_neighbors	mean_train_score	mean_test_score	rank_test_score
0	1	1.000000	0.566154	3
1	3	0.815556	0.631282	2
2	5	0.775948	0.723077	1

```
{'algorithm': 'auto',
 'leaf_size': 30,
 'metric': 'minkowski',
 'metric_params': None,
 'n_jobs': None,
 'n_neighbors': 5,
 'p': 2,
 'weights': 'uniform'}
```

```python
from sklearn.model_selection import GridSearchCV

ss = StandardScaler(with_mean=True, with_std=True)
X_train_transform_ss = ss.fit_transform(X_train)

si = SimpleImputer(strategy='mean')
X_train_transform_ss_si = si.fit_transform(X_train_transform_ss)

clf = KNeighborsClassifier()

# Puedes poner tu propia métrica
cv = GridSearchCV(clf,
                  param_grid = {'n_neighbors': [1, 3, 5]},
                  scoring = 'accuracy', refit = True,
                  cv = 3,
                  return_train_score=True)

cv.fit(X_train_transform_ss_si, y_train)


cv_results = pd.DataFrame(cv.cv_results_)
display(cv_results[['param_n_neighbors', 'mean_train_score', 'mean_test_score', 'rank_test_score']])

print(cv.best_estimator_.get_params())

# save: DISCLAIMER: model registry
with open(os.path.join('pkl','round_3','standard_scaler.pkl'),'wb') as f:
    pickle.dump(ss,f)

with open(os.path.join('pkl','round_3','simple_imputer.pkl'),'wb') as f:
    pickle.dump(si,f)

# WATCH OUT!
with open(os.path.join('pkl','round_3','knn.pkl'),'wb') as f:
    pickle.dump(cv.best_estimator_,f)
```

Round 3

Scikit-learn 1.0.2

¡Código listo para ir a **producción**!

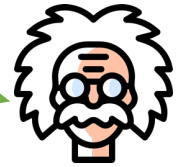
```python
with open(os.path.join('pkl','round_3','standard_scaler.pkl'), 'rb') as f:
    ss = pickle.load(f)

with open(os.path.join('pkl','round_3','simple_imputer.pkl'), 'rb') as f:
    si = pickle.load(f)

with open(os.path.join('pkl','round_3','knn.pkl'), 'rb') as f:
    clf = pickle.load(f)


X_test_transf_ss = ss.transform(X_test)
X_test_transf_ss_si = si.transform(X_test_transf_ss)

print("Accuracy in test: ", accuracy_score(y_test, clf.predict(X_test_transf_ss_si)))
```

```
Accuracy in test:  0.55
```

ML Engineer

¡OJO! Puedes tener ***data leakage*** en el preprocesado. La validación cruzada hay que hacerla también sobre los pasos de preprocesado (escalado, imputación, feature selection, etc) para:

1. Asegurar que los datos de los folds de CV son vírgenes.
2. Ajustar hiperparámetros en los algoritmos de preprocesado.

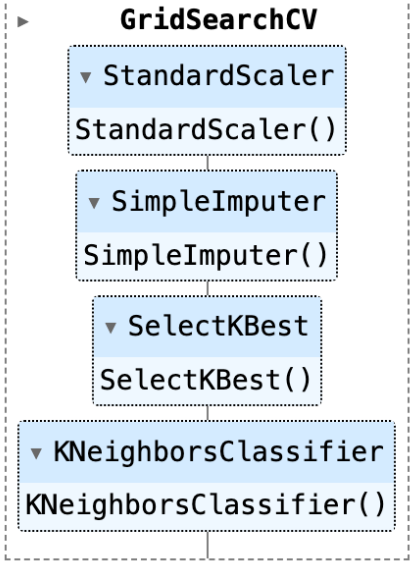


Round 4

Scikit-learn 1.0.2

¡PIPELINES DE SCIKIT-LEARN!

Fit + transform
Fit + predict/predict_proba

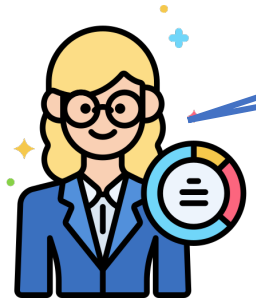



```python
from sklearn.pipeline import Pipeline
from sklearn.feature_selection import SelectKBest, f_classif
from sklearn import set_config

set_config(display="diagram")

pl=Pipeline([
    ('scaler', StandardScaler()),
    ('imputer', SimpleImputer()),
    ('feature_selection', SelectKBest(score_func=f_classif)),
    ('classifier',KNeighborsClassifier())
])

my_param_grid = [
    {
    'imputer__strategy': ['mean', 'median'],
    'feature_selection__k': [1,2],
    'classifier__n_neighbors': [1, 3, 5],
    }
]

cv = GridSearchCV(pl,
                  param_grid = my_param_grid,
                  scoring = 'accuracy', refit = True,
                  cv = 3,
                 return_train_score=True)

cv.fit(X_train,y_train)
```

Round 4

Scikit-learn 1.0.2

¡Pipelines de Scikit-learn nos vamos a **producción**!



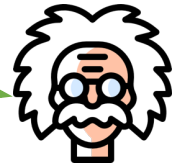
	param_imputer__strategy	param_feature_selection__k	param_classifier__n_neighbors	mean_train_score	mean_test_score	rank_test_score
9	median	1	5	0.802353	0.749744	1
5	median	1	3	0.842092	0.736923	2
0	mean	1	1	0.980261	0.736410	3

```
with open(os.path.join('pkl','round_4','pipeline.pkl'), 'rb') as f:
    pipe = pickle.load(f)

print("Accuracy in test: ", accuracy_score(y_test, pipe.predict(X_test)))
```

```
Accuracy in test:  0.55
```

ML Engineer

¡OJO! Parece que el modelo no generaliza bien del todo…. ¿podrías probar algún modelo más simple? ¿una **regresión logística** y ver cuál sale mejor?



Round 5

¡PIPELINES DE SCIKIT-LEARN!

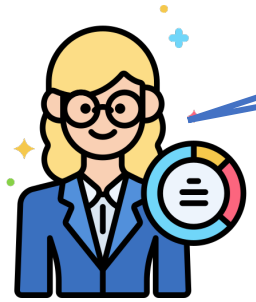
Scikit-learn 1.0.2

```python
from sklearn.linear_model import LogisticRegression

pl=Pipeline([
    ('scaler', StandardScaler()),
    ('imputer', SimpleImputer()),
    ('feature_selection', SelectKBest(score_func=f_classif)),
    ('classifier',KNeighborsClassifier())
])

my_param_grid = [
    {
     'imputer__strategy': ['mean', 'median'],
     'feature_selection__k': [1,2],
     'classifier': [KNeighborsClassifier()],
     'classifier__n_neighbors': [1, 3, 5],
    },
    {
     'imputer__strategy': ['mean', 'median'],
     'feature_selection__k': [1,2],
     'classifier': [LogisticRegression(random_state=42)],
     'classifier__C': [0.01, 0.1, 1.0, 10.0, 100.0]
    }
]

cv = GridSearchCV(pl,
                  param_grid = my_param_grid,
                  scoring = 'accuracy', refit = True,
                  cv = 3,
                 return_train_score=True)

cv.fit(X_train,y_train)

cv_results = pd.DataFrame(cv.cv_results_).sort_values(by="rank_test_score")
display(cv_results[['param_imputer__strategy','param_feature_selection__k',
                    'param_classifier','param_classifier__n_neighbors', 'param_classifier__C',
                    'mean_train_score', 'mean_test_score', 'rank_test_score']])
print(cv.best_estimator_.get_params())

# WATCH OUT!
with open(os.path.join('pkl','round_5','pipeline.pkl'),'wb') as f:
    pickle.dump(cv.best_estimator_,f)
```

Round 5

Scikit-learn 1.0.2

¡Pipelines de Scikit-learn nos vamos a **producción**!



param_imputer__strategy	param_feature_selection__k	param_classifier	param_classifier__n_neighbors	param_classifier__C	mean_train_score	mean_test_score	rank_test_score
median	1	LogisticRegression(random_state=42)	NaN	100.0	0.763007	0.763077	1
mean	1	LogisticRegression(random_state=42)	NaN	100.0	0.763007	0.763077	1
median	1	LogisticRegression(random_state=42)	NaN	10.0	0.763007	0.763077	1
mean	1	LogisticRegression(random_state=42)	NaN	10.0	0.763007	0.763077	1

```
with open(os.path.join('pkl','round_5','pipeline.pkl'), 'rb') as f:
    pipe = pickle.load(f)

print("Accuracy in test: ", accuracy_score(y_test, pipe.predict(X_test)))
```

```
Accuracy in test:  0.75
```

Scikit-learn 1.0.2

Round 6

	n_medio_productos	impago	tipo_antiguedad
0	1.20	0	senior
1	1.40	0	senior
2	1.55	0	master
3	1.60	0	master
4	1.45	0	master

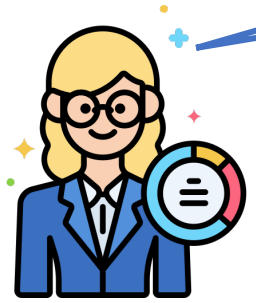
Tiene Buena pinta el modelo, pero verás… para mejorar la interpretabilidad desde negocio en realidad nos gustaría dividir la variable antigüedad en los segmentos first_year, junior, senior, y master.

¡De acuerdo!

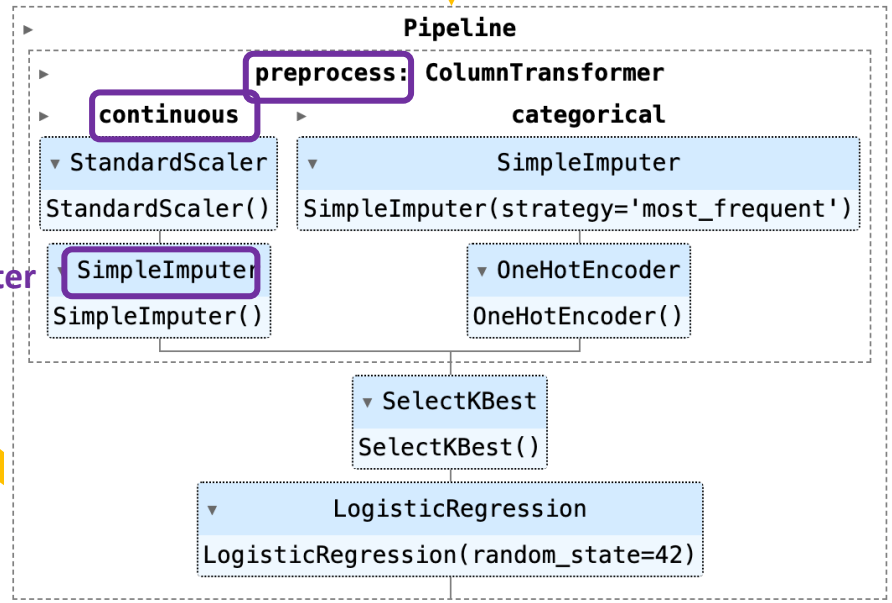
A ver cómo lo hago ahora… tengo que **codificar** la variable categórica por una parte además de que no puedo tratar de igual forma los **missing values**…

Round 6

¡COLUMN TRANSFORMERS!



imputer



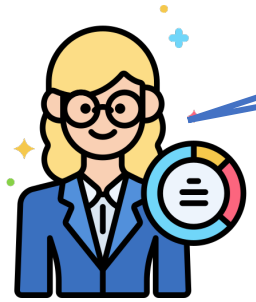
Scikit-learn 1.0.2

```
continuous_pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('imputer', SimpleImputer(strategy='mean')),
])

categorical_pipeline = Pipeline([
    ('imputer', SimpleImputer(strategy='most_frequent')),
    ('codification', OneHotEncoder(sparse_output=False))
])

preprocessing_pipeline = ColumnTransformer([
        ('continuous', continuous_pipeline, ['n_medio_productos']),
        ('categorical', categorical_pipeline, ['tipo_antiguedad'])
    ],
    remainder = 'drop'
)

pipeline = Pipeline([
    ('preprocess', preprocessing_pipeline),
    ('feature_selection', SelectKBest(score_func=f_classif)),
    ('classifier', LogisticRegression(random_state=42))])

pipeline.set_output(transform="pandas")

my_param_grid = [
    {
     'preprocess__continuous__imputer__strategy': ['mean','median'],
     'feature_selection__k': [1,2],
     'classifier': [RandomForestClassifier()],
     'classifier__n_estimators': [10, 50, 100],
    },
    {
     'preprocess__continuous__imputer__strategy': ['mean','median'],
     'feature_selection__k': [1,2],
     'classifier': [LogisticRegression(random_state=42)],
     'classifier__C': [0.01, 0.1, 1.0, 10.0, 100.0]
    }
]

cv = GridSearchCV(pipeline,
                  param_grid = my_param_grid,
                  scoring = 'accuracy', refit = True,
                  cv = 3,
                 return_train_score=True)

cv.fit(X_train,y_train)
```

Round 6

Scikit-learn 1.0.2

¡Pipelines de Scikit-learn nos vamos a **producción**!



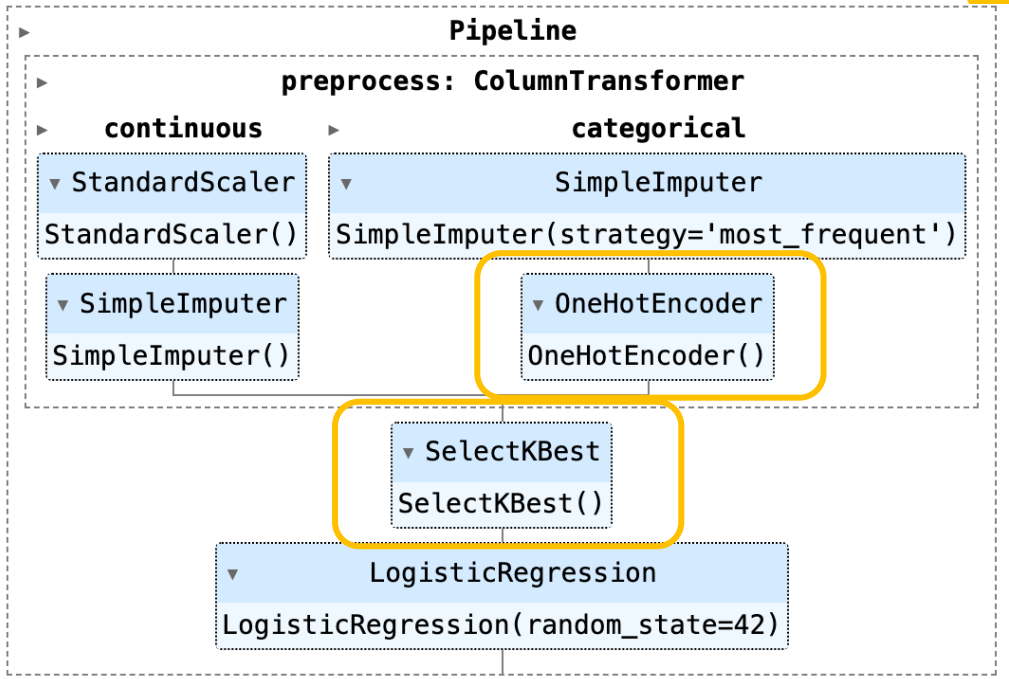
	param_preprocess__continuous__imputer__strategy	param_feature_selection__k	param_classifier	param_classifier__n_estimators	param_classifier__C	mean_train_score	mean_test_score	rank_test_score
22	mean	2	LogisticRegression(random_state=42)	NaN	1.0	0.67098	0.670769	1
23	median	2	LogisticRegression(random_state=42)	NaN	1.0	0.67098	0.670769	1
0	mean	1	RandomForestClassifier()	10	NaN	0.67085	0.656410	3

```python
from sklearn.metrics import accuracy_score

with open(os.path.join('pkl','round_categorical','pipeline.pkl'), 'rb') as f:
    pipe = pickle.load(f)

print("Accuracy in test: ", accuracy_score(y_test, pipe.predict(X_test)))
```

```
Accuracy in test:  0.7
```

Round 7

Scikit-learn 1.0.2



¡Fenomenal! ¿Me podrías decir qué variables son las más relevantes para tu modelo?

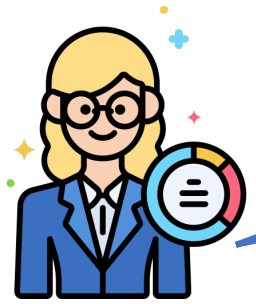
¡En seguida!

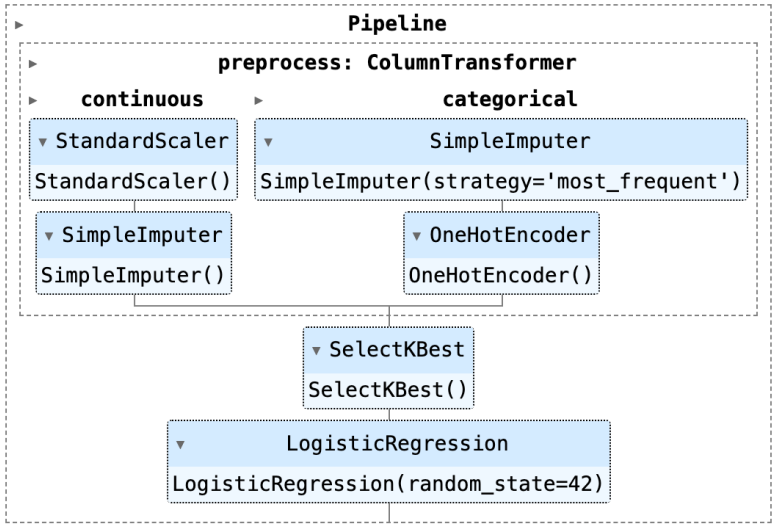
Solo necesito tener en cuenta las variables generadas por el one-hot-encoder y las variables seleccionadas por SelectKBest

Round 7

Scikit-learn 1.0.2

Solo necesito recordar los atributos y funciones más importantes de **Pipeline** y **ColumnTransformer**…

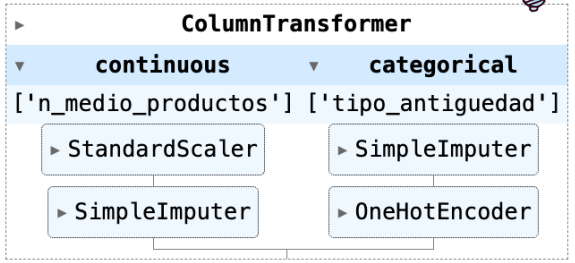





```
display(pipe.named_steps)
```

```
{'preprocess': ColumnTransformer(transformers=[('continuous',
                                                Pipeline(steps=[('scaler', StandardScaler()),
                                                                ('imputer', SimpleImputer())]),
                                                 ['n_medio_productos']),
                                                ('categorical',
                                                 Pipeline(steps=[('imputer',
                                                                  SimpleImputer(strategy='most_frequent')),
                                                                 ('codification',
                                                                  OneHotEncoder())]),
                                                 ['tipo_antiguedad'])]),
 'feature_selection': SelectKBest(k=2),
 'classifier': KNeighborsClassifier(n_neighbors=1)}
```

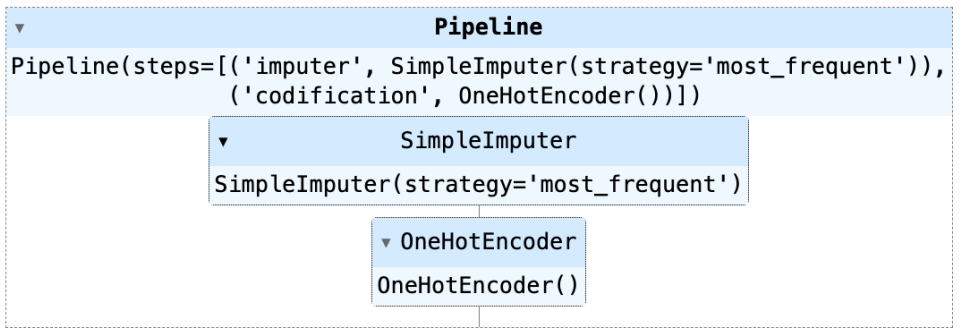
```
display(pipe.named_steps['preprocess'])
```




```
display(pipe.named_steps['preprocess'].named_transformers_)
```

```
{'continuous': Pipeline(steps=[('scaler', StandardScaler()), ('imputer', SimpleImputer())]),
 'categorical': Pipeline(steps=[('imputer', SimpleImputer(strategy='most_frequent')),
                                ('codification', OneHotEncoder())])}
```

```
display(pipe.named_steps['preprocess'].named_transformers_['categorical'])
```



Scikit-learn 1.0.2

Round 7

Voy a revisar las variables de entrada y salida de one-hot-encoder…

```
Pipeline(steps=[('imputer', SimpleImputer(strategy='most_frequent')),
                ('codification', OneHotEncoder())])
```

SimpleImputer(strategy='most_frequent')

OneHotEncoder()

```
print("Variables salida one hot encoder: ", pipe.named_steps['preprocess'].named_transformers_['categorical'].named_steps['codification'].get_feature_names_out())
```

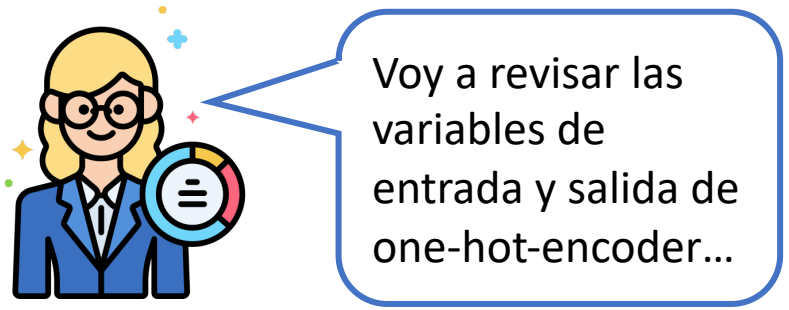
```
Variables salida one hot encoder:  ['x0_first_year' 'x0_junior' 'x0_master' 'x0_senior']
```

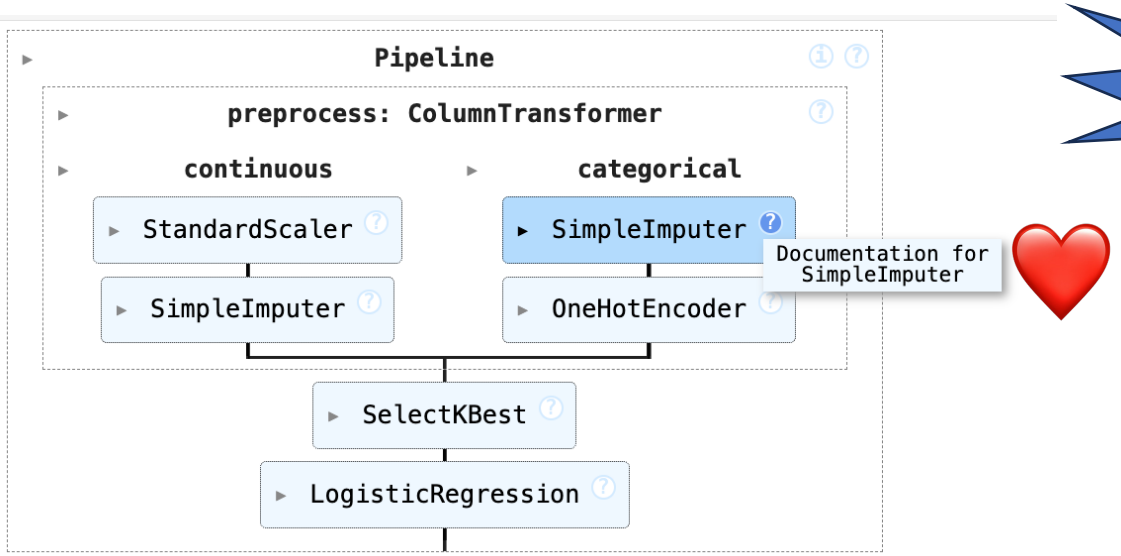
¿De dónde salió ese x_0? Voy a mirar la salida del SimpleImputer….

```
print("Variables salida categorical-imputer: ", pipe.named_steps['preprocess'].named_transformers_['categorical'].named_steps['imputer'].get_feature_names_out())
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[23], line 1
----> 1 print("Variables salida categorical-imputer: ", pipe.named_steps['preprocess'].named_transformers_['categorical'].named_steps['imputer'].get_feature_names_out())

AttributeError: 'SimpleImputer' object has no attribute 'get_feature_names_out'
```

Round 7





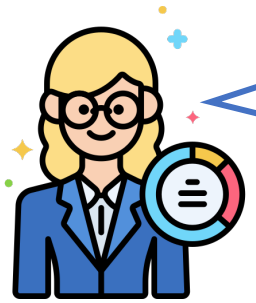
Scikit-learn 1.4.0

```
print("Variables entrada categorical-imputer: ",
      pipe.named_steps['preprocess'].named_transformers_['categorical'].named_steps['imputer'].feature_names_in_)
print("Variables salida categorical-imputer: ",
      pipe.named_steps['preprocess'].named_transformers_['categorical'].named_steps['imputer'].get_feature_names_out())

print("Variables entrada one hot encoder: ",
      pipe.named_steps['preprocess'].named_transformers_['categorical'].named_steps['codification'].feature_names_in_)
print("Variables salida one hot encoder: ",
      pipe.named_steps['preprocess'].named_transformers_['categorical'].named_steps['codification'].get_feature_names_out())
```

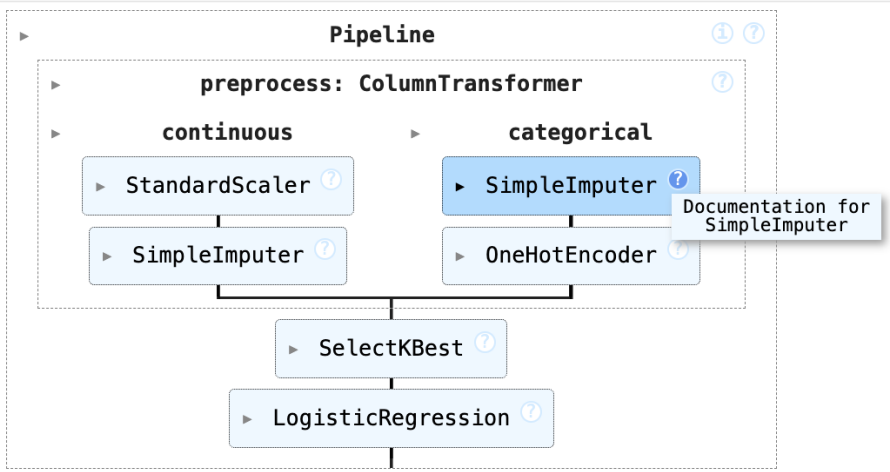
```
Variables entrada categorical-imputer:  ['tipo_antiguedad']
Variables salida categorical-imputer:  ['tipo_antiguedad']
Variables entrada one hot encoder:  ['tipo_antiguedad']
Variables salida one hot encoder:  ['tipo_antiguedad_first_year' 'tipo_antiguedad_junior'
 'tipo_antiguedad_master' 'tipo_antiguedad_senior']
```

```
continuous_pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('imputer', SimpleImputer(strategy='mean')),
])

categorical_pipeline = Pipeline([
    ('imputer', SimpleImputer(strategy='most_frequent')),
    ('codification', OneHotEncoder(sparse_output=False))
])

preprocessing_pipeline = ColumnTransformer([
        ('continuous', continuous_pipeline, ['n_medio_productos']),
        ('categorical', categorical_pipeline, ['tipo_antiguedad'])
    ],
    remainder = 'drop'
)

pipeline = Pipeline([
    ('preprocess', preprocessing_pipeline),
    ('feature_selection', myFeatureSelector(n_features=3)),
    ('classifier', LogisticRegression(random_state=42))])

pipeline.set_output(transform="pandas")
```

Round 7



Continuemos con la importancia de variables…



Scikit-learn 1.4.0

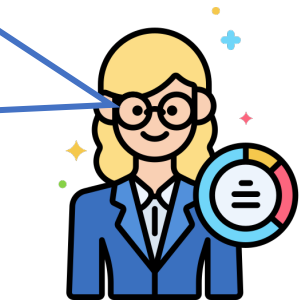
```
fimp = pd.DataFrame(pipe.named_steps['classifier'].coef_.T, columns=['coef'])
fimp['feature'] = pipe.named_steps['classifier'].feature_names_in_
fimp
```

	coef	feature
0	0.729020	categorical__tipo_antiguedad_junior
1	-1.143261	categorical__tipo_antiguedad_master

```
subpl = Pipeline(pipe.steps[:-2])
x_transf = subpl.transform(X_train)
x_transf.head(2)
```

	continuous__n_medio_productos	categorical__tipo_antiguedad_first_year	categorical__tipo_antiguedad_junior	categorical__tipo_antiguedad_master	categorical__tipo_antiguedad_ser
40	0.364954	0.0	0.0	1.0	
67	0.006896	0.0	1.0	0.0	

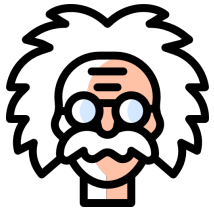
¿Y qué entró exactamente al algoritmo de selección de variables?



Round 8

Scikit-learn 1.4.0

¿Has visto el ultimo método de selección de variables? Una pena que no esté en scikit-learn y no podamos probarlo para el proyecto….



Mi querido Lead… las pipelines ya no tienen secretos para mí! puedo desarrollar mis propios transformadores/estimadores siempre y cuando respete la API definida.

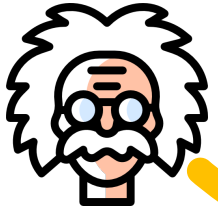
```python
from sklearn.base import BaseEstimator, TransformerMixin

class myFeatureSelector(BaseEstimator, TransformerMixin):
    def __init__(self, n_features):
        self.n_features=n_features
        self.selected_features=None

    def fit(self, X, y = None):
        corr = pd.concat([X,y], axis=1).corr().iloc[:-1,-1].abs()
        self.selected_features = list(corr.sort_values(ascending=False).head(self.n_features).index)
        return self

    def transform(self, X):
        return X.loc[:,self.selected_features]

    def get_feature_names_out(self):
        return self.selected_features
```


```python
pipeline = Pipeline([
    ('preprocess', preprocessing_pipeline),
    ('feature_selection', myFeatureSelector(n_features=3)),
    ('classifier', LogisticRegression(random_state=42))])
```

Final Round

Te contaría sobre FeatureUnion, pero eso mejor lo dejamos para otro día… [spoiler]

¿¿¿???



Final Round


LOS ANGELES
LAKERS
SPALDING
NBA
2+
Loto
30pcs
Dalco
PIPERLAB

¡GRACIAS!