Team Challenge: Pipelines en Machine Learning

Miembros del equipo:

- Luis Tamayo
- Juan Manuel Gómez
- Lander Combarro
- Victor Manuel de Sousa Sánchez

30 noviembre 2024

MUSIC GENRE

Objetivo:

- Clasificar canciones por sus características técnicas
- Usar pipelines para tal fin

	instance_id	artist_name	track_name	popularity	acousticness	danceability	duration_ms	energy	instrumentalness	key	liveness	loudness	mode	speechiness	tempo	obtained_date	valence	music_genre
0	32894.0	Röyksopp	Röyksopp's Night Out	27.0	0.00468	0.652	-1.0	0.941	0.79200	A#	0.115	-5.201	Minor	0.0748	100.889	4-Apr	0.759	Electronic
1	46652.0	Thievery Corporation	The Shining Path	31.0	0.01270	0.622	218293.0	0.890	0.95000	D	0.124	-7.043	Minor	0.0300	115.00200000000001	4-Apr	0.531	Electronic
2	30097.0	Dillon Francis	Hurricane	28.0	0.00306	0.620	215613.0	0.755	0.01180	G#	0.534	-4.617	Major	0.0345	127.994	4-Apr	0.333	Electronic
3	62177.0	Dubloadz	Nitro	34.0	0.02540	0.774	166875.0	0.700	0.00253	C#	0.157	-4.498	Major	0.2390	128.014	4-Apr	0.270	Electronic
4	24907.0	What So Not	Divide & Conquer	32.0	0.00465	0.638	222369.0	0.587	0.90900	F#	0.157	-6.266	Major	0.0413	145.036	4-Apr	0.323	Electronic
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
50000	58878.0	BEXEY	GO GETTA	59.0	0.03340	0.913	-1.0	0.574	0.00000	C#	0.119	-7.022	Major	0.2980	98.02799999999999	4-Apr	0.330	Hip-Hop
50001	43557.0	Roy Woods	Drama (feat. Drake)	72.0	0.15700	0.709	251860.0	0.362	0.00000	B	0.109	-9.814	Major	0.0550	122.04299999999999	4-Apr	0.113	Hip-Hop
50002	39767.0	Berner	Lovin' Me (feat. Smiggz)	51.0	0.00597	0.693	189483.0	0.763	0.00000	D	0.143	-5.443	Major	0.1460	131.079	4-Apr	0.395	Hip-Hop
50003	57944.0	The-Dream	Shawty Is Da Shit	65.0	0.08310	0.782	262773.0	0.472	0.00000	G	0.106	-5.016	Minor	0.0441	75.88600000000001	4-Apr	0.354	Hip-Hop
50004	63470.0	Naughty By Nature	Hip Hop Hooray	67.0	0.10200	0.862	267267.0	0.642	0.00000	F#	0.272	-13.652	Minor	0.1010	99.20100000000001	4-Apr	0.765	Hip-Hop

50005 rows × 18 columns

El data set se compone de:

- 17 features
- 1 target → music_genre

MUSIC GENRE → Clasificador multiclase

Transformaciones NO implementadas en el pipeline:

- **Limitación transformaciones sobre el target → Eliminación instancias (filas) no posible.**

Eliminacion de filas con valor "?" en la columna "tempo":

Durante el desarrollo de este notebook se probo a aplicar esta transformacion del data frame a traves del pipeline.
El problema es que este cambio solo se aplica a X_train, dejando y_train con mas instancias, lo que da error de forma.

Por tal motivo, esta transformacion se va a realizar fuera del pipeline.
Como se vio anteriormente, el target de mantenia homogeneo, no penalizando en exceso ninguna categoria del mismo.

```
filtro_tempo = df_music["tempo"] != "?"

df_music = df_music.loc[filtro_tempo]
df_music["tempo"] = df_music["tempo"].astype(float)
```

[90] ✓ 0.0s

Eliminación de valores negativos en la columna "duration_ms"

Al igual que en el caso anterior, no ha sido posible implementarlo como un paso del pipeline.
Se ejecutara exteriormente.

```
filtro_duration = df_music["duration_ms"] >= 0

df_music = df_music.loc[filtro_duration]
```

[91] ✓ 0.0s

Nota

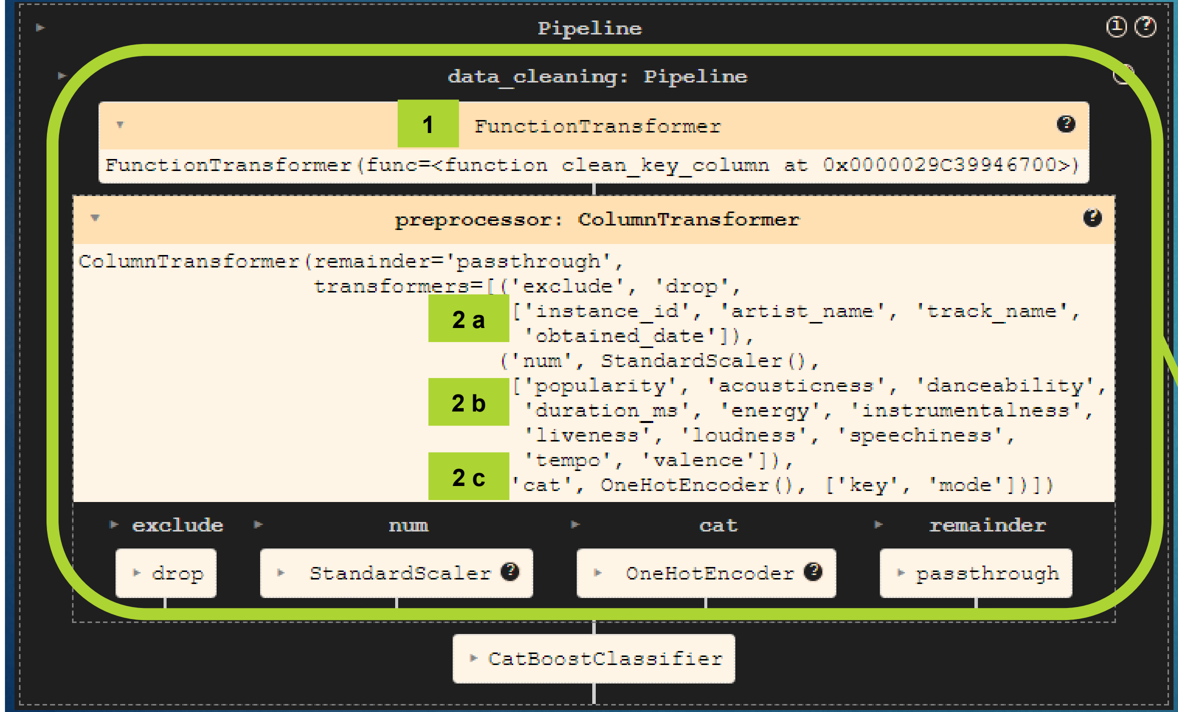
No se opto por imputar la mediana debido a la gran desviación estándar y a que se disponía de 50.000 instancias para entrenar.

Se comprobó que al borrar estas instancias el target seguía balanceado.

MUSIC GENRE → Clasificador multiclase

Estructura del pipeline y sus transformaciones:

1) Elimina el símbolo "#" del string para dejar las categóricas de la columna de forma correcta.
2) a → Elimina las columnas que no van a ser necesarias para el entrenamiento.
2) b → Aplica estandarización a las columnas numéricas para poder ser usado con modelos de sklearnt.
2) c → Aplica One Hot Encoder a las categóricas por el mismo motivo que el punto 2b.




```python
# Función para limpiar la columna "key"
def clean_key_column(X):
    X = X.copy()
    X['key'] = X['key'].apply(lambda value: value.replace("#", "") if "#" in value else value)
    return X

# ColumnTransformer
preprocessor = ColumnTransformer(
    transformers=[                                              2 a
        ('exclude', 'drop', columns_to_drop),
        ('num', StandardScaler(), numeric_columns)              2 b
        ('cat', OneHotEncoder(), cat_columns)
    ],                                                          2 c
    remainder='passthrough'
)

# Pipeline principal
pipeline_X = Pipeline(steps=[
    ('clean_key', FunctionTransformer(clean_key_column, validate=False)),   1
    ('preprocessor', preprocessor)
])
```

MUSIC GENRE → Clasificador multiclase

Data set después de las transformaciones:

	num__popularity	num__acousticness	num__danceability	num__duration_ms	num__energy	num__instrumentalness	num__liveness	num__loudness	num__speechiness	num__tempo
0	-0.402080	-0.101859	1.188409	0.398996	-0.597768	-0.556029	0.781729	0.235331	-0.652215	-0.752957
1	1.013194	-0.769722	0.963710	0.511278	0.305498	-0.555319	0.472040	0.250627	-0.524218	-0.452904
2	1.077524	-0.873110	0.817655	-0.491243	0.638081	-0.556067	-0.364122	0.565484	-0.660092	-0.323030
3	0.369887	-0.866135	1.340081	0.052766	-0.457932	-0.555983	-0.813791	-0.770098	-0.561633	0.143558
4	0.112565	0.284968	-0.918144	-0.436768	-1.017276	-0.554658	-0.603202	0.111178	-0.620708	-0.254398

num__valence	cat__key_A	cat__key_B	cat__key_C	cat__key_D	cat__key_E	cat__key_F	cat__key_G	cat__mode_Major	cat__mode_Minor
0.624614	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0
-0.048400	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	1.0
1.163835	0.0	0.0	0.0	0.0	0.0	0.0	1.0	1.0	0.0
1.666568	0.0	0.0	0.0	0.0	0.0	0.0	1.0	1.0	0.0
-0.656544	0.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0

MUSIC GENRE → Clasificador multiclase

Modelo usados y cross validation para seleccionar el mejor:

```
# Modelos
models = {
    'RandomForest': RandomForestClassifier(class_weight='balanced', random_state=42),
    'XGBoost': XGBClassifier(eval_metric='mlogloss', random_state=42),
    'LightGBM': LGBMClassifier(random_state=42, verbose=-1),
    'CatBoost': CatBoostClassifier(verbose=0, random_state=42),
}

# Validación cruzada
recall_scores = {}

for model_name, model in models.items():
    pipeline_full = Pipeline(steps=[
        ('data_cleaning', pipeline_X),
        ('classifier', model)
    ])

    skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42) # se hace StratifiedKFold para poder hacer shuffle
    cv_scores = cross_val_score(pipeline_full, X, y, cv=skf, scoring="balanced_accuracy", error_score="raise")
    mean_score = np.mean(cv_scores)
    recall_scores[model_name] = mean_score
    print(f"Modelo: {model_name}, Balanced Accuracy: {mean_score:.4f}")
```

Nota

Los 4 modelos han dado un score muy parecido, por lo que no se descartara ninguno de ellos.

⇩

```
Modelo: RandomForest, Balanced Accuracy: 0.5494
Modelo: XGBoost, Balanced Accuracy: 0.5714
Modelo: LightGBM, Balanced Accuracy: 0.5785
Modelo: CatBoost, Balanced Accuracy: 0.5867
```

⇩

Han dado un recall medio muy bajo, luego se analizará la matriz de confusión para ver él porque.

MUSIC GENRE → Clasificador multiclase

Hiperparametros y Gridsearch:

```python
# Hiperparámetros
param_grids = {
    'RandomForest': {
        'classifier__n_estimators': [50, 100, 200],
        'classifier__max_depth': [None, 10, 20, 30]
    },
    'XGBoost': {
        'classifier__n_estimators': [100, 200],
        'classifier__learning_rate': [0.01, 0.1, 0.2],
        'classifier__max_depth': [3, 6, 10]
    },
    'LightGBM': {
        'classifier__n_estimators': [100, 200],
        'classifier__learning_rate': [0.01, 0.1, 0.2],
        'classifier__max_depth': [-1, 10, 20]
    },
    'CatBoost': {
        'classifier__iterations': [100, 200],
        'classifier__learning_rate': [0.01, 0.1, 0.2],
        'classifier__depth': [4, 6, 10]
    }
}
```

```python
grid_search_results = []

# GridSearchCV para los modelos que se quiere evaluar
for model_name, model in models.items():
    pipeline_full = Pipeline(steps=[
        ('data_cleaning', pipeline_X),
        ('classifier', model)
    ])

    grid_search = GridSearchCV(pipeline_full, param_grid=param_grids[model_name],
                               cv=5, scoring="balanced_accuracy", n_jobs=-1)
    grid_search.fit(X, y)

    best_model = grid_search.best_estimator_
    best_score = grid_search.best_score_
    grid_search_results.append((model_name, best_model, best_score))
    print(f"Modelo: {model_name}, Mejor Balanced Accuracy en validación: {best_score:.4f}")
```

```
Modelo: RandomForest, Mejor Balanced Accuracy en validación: 0.5654
Modelo: XGBoost, Mejor Balanced Accuracy en validación: 0.5900
Modelo: LightGBM, Mejor Balanced Accuracy en validación: 0.5807
Modelo: CatBoost, Mejor Balanced Accuracy en validación: 0.5948
```

NOTA

Han mejorado sensiblemente, pero lejos de los esperado

MUSIC GENRE → Clasificador multiclase

Evaluación contra test:

```
Modelo: XGBoost
Balanced Accuracy en el conjunto de prueba: 0.6556
```

NOTA

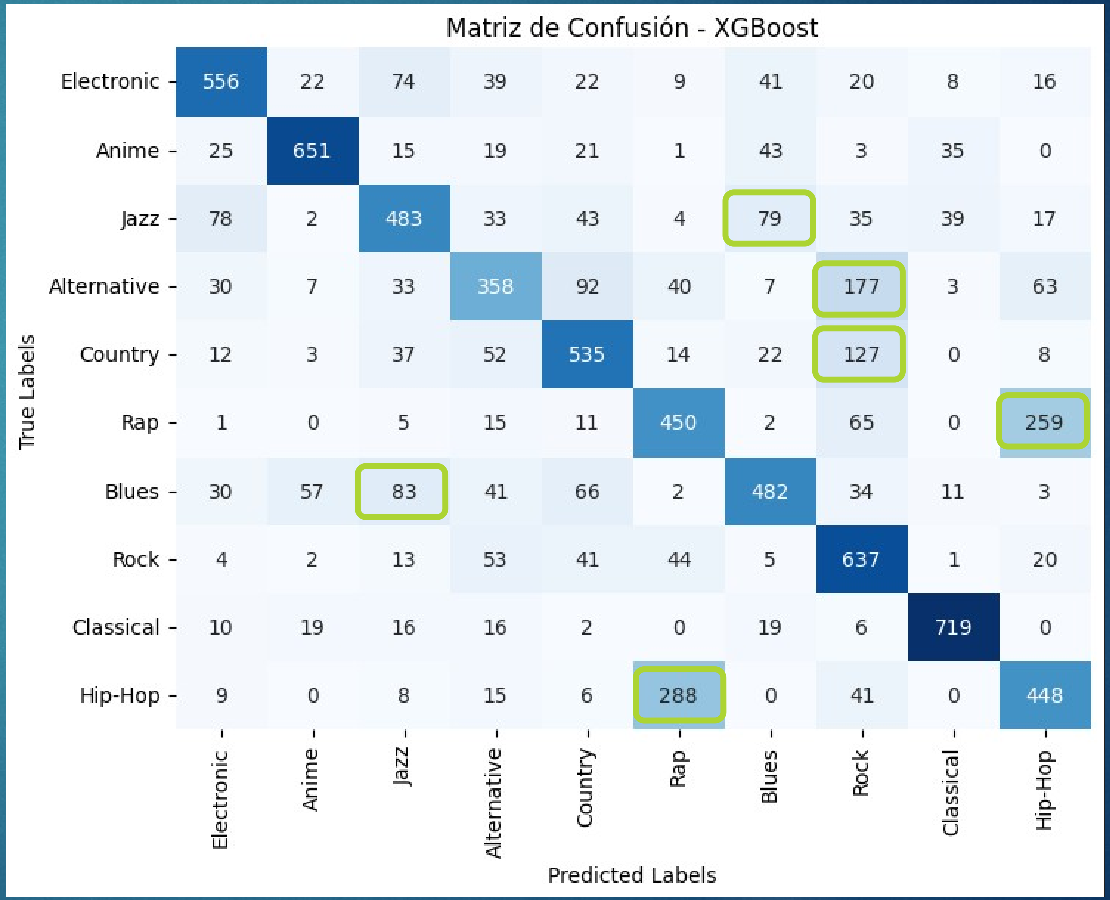
Se muestra el mejor modelo de todos.

Se aprecia como hay géneros musicales que podríamos llamar parecidos, que el modelo los confunde:

- Hip-Hop vs Rap
- Jazz vs Blues
- Country vs Rock

El genero que mejor identifica es:

- Classical



WINE CLASS

Objetivo:

- Clasificador binario

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
3369	5.8	0.32	0.20	2.6	0.027	17.0	123.0	0.98936	3.36	0.78	13.9	7
1935	6.1	0.27	0.43	7.5	0.049	65.0	243.0	0.99570	3.12	0.47	9.0	5
4860	5.6	0.19	0.47	4.5	0.030	19.0	112.0	0.99220	3.56	0.45	11.2	6
1997	7.2	0.62	0.06	2.7	0.077	15.0	85.0	0.99746	3.51	0.54	9.5	5
3258	9.0	0.60	0.29	2.0	0.069	32.0	73.0	0.99654	3.34	0.57	10.0	5

El data set se compone de:

- 12 features
- 5,197 entradas
- *Target: class*

WINE CLASS → CLASIFICADOR

Estructura del pipeline y sus transformaciones:

1. Aplicación de logaritmo
1.bis Generación de Escalado a valores

2. Excluir columnas

3. Generación de modelos

```
#Creamos el pipeline

columns_to_exclude=['total sulfur dioxide']
columnas = list(X_train.columns)
columnas.remove('total sulfur dioxide')
```

```
# Escalado de valores numéricas
num_pipeline = Pipeline(
    [
     ("Logaritmo",FunctionTransformer(func = np.tanh)),
     ("SScaler", StandardScaler())
    ]
)
```

1

```
# Preprocesado completo
preprocessing = ColumnTransformer(
    [
     ("Process Numeric", num_pipeline, columnas),
     ("Exclude", "drop", columns_to_exclude)
    ], remainder = "passthrough")
```

2

```
# Modelos a utilizar
logistic_pipeline = Pipeline(
    [("Preprocesado", preprocessing),
     ("Modelo", LogisticRegression())
    ])

random_pipeline = Pipeline(
    [("Preprocesado", preprocessing),
     ("Modelo", RandomForestClassifier())
    ])

xgb_pipeline = Pipeline(
    [("Preprocesado", preprocessing),
     ("Modelo", XGBClassifier())
    ])
```

3

WINE CLASS → CLASIFICADOR

```python
# Aplicamos los gridsearch sobre los modelos

pipe_reg_log_param = {
                    "Modelo__penalty": [None,"l2"],
                    "Modelo__C": np.logspace(0, 4, 10)
                    }

pipe_rand_forest_param = {
    'Modelo__n_estimators': [10, 100, 200, 400],
    'Modelo__max_depth': [1,2,4,8],
    'Modelo__max_features': [1, 2, 3]
    }

pipe_xgb_param = {
    'Modelo__n_estimators': [10, 100, 200, 400],
    'Modelo__max_depth': [1,2,4,8],
    'Modelo__learning_rate': [0.1,0.2,0.5,1.0]
}

cv = 5

gs_reg_log = GridSearchCV(logistic_pipeline,
                          pipe_reg_log_param,
                          cv=cv,
                          scoring="accuracy",
                          verbose=1,
                          n_jobs=-1)

gs_rand_forest = GridSearchCV(random_pipeline,
                          pipe_rand_forest_param,
                          cv=cv,
                          scoring="accuracy",
                          verbose=1,
                          n_jobs=-1)

gs_xgb = GridSearchCV(xgb_pipeline,
                      pipe_xgb_param,
                      cv=cv,
                      scoring="accuracy",
                      verbose=1,
                      n_jobs=-1)

pipe_grids = {"gs_reg_log":gs_reg_log,
        "gs_rand_forest":gs_rand_forest,
        "gs_xgb":gs_xgb}
```

1.

```python
# Vamos a coger los mejores resultados de cada modelo

best_grids = [(i, j.best_score_) for i, j in pipe_grids.items()]

best_grids = pd.DataFrame(best_grids, columns=["Grid", "Best score"]).sort_values(by="Best score", ascending=False)
best_grids
```

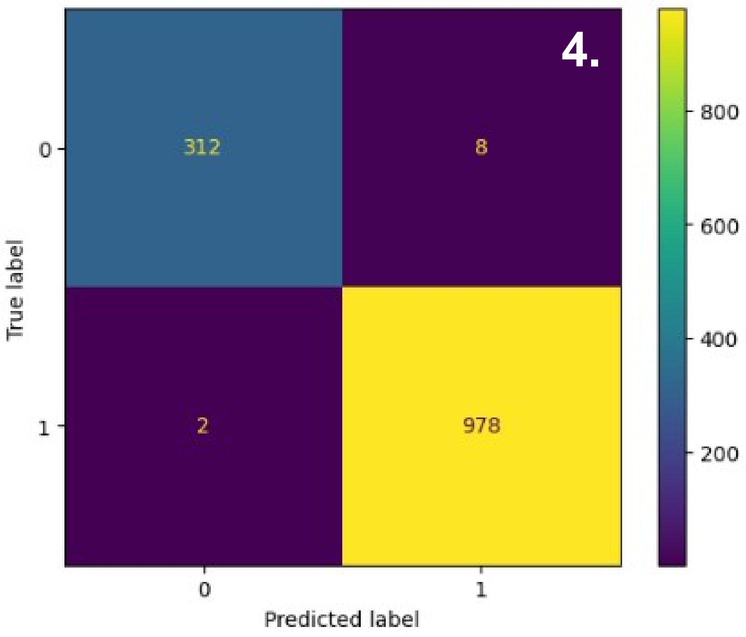
2.a

2.b

	Grid	Best score
2	gs_xgb	0.991341
1	gs_rand_forest	0.989609
0	gs_reg_log	0.952280

```python
# Aplicamos el modelo al X_test entrenado ya para predecir mis valores

y_pred = gs_xgb.predict(X_test)
```

3.



1. Generación GridSearch pipelines

2. Selección best score

3. Predicción

4. Confusion matrix

CALIFORNIA HOUSING

Objetivos:

- Regresor del precio de las casas
- Clustering no supervisado de categorías en las viviendas

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value	ocean_proximity
0	-122.23	37.88	41.0	880.0	129.0	322.0	126.0	8.3252	452600.0	NEAR BAY
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0	1138.0	8.3014	358500.0	NEAR BAY
2	-122.24	37.85	52.0	1467.0	190.0	496.0	177.0	7.2574	352100.0	NEAR BAY
3	-122.25	37.85	52.0	1274.0	235.0	558.0	219.0	5.6431	341300.0	NEAR BAY

El data set se compone de:

- 10 features
- 20,640 entradas
- *Target* (para el regresor): *median_house_value*

CALIFORNIA HOUSING

Estructura del pipeline y sus transformaciones:

1. Crear tres nuevas *features* más descriptivas
1.bis Eliminar features no representativas

2. Imputar valores missings

3. Modificar la distribución de las features numéricas a una más Gaussiana
4. Estandarizar los datos.

5. Ordinal Encoder de la feature "ocean_proximity"

```
# create extra feaatures function
def create_extra_features(X):
    '''
    Create new features for the dataset that offer greater explainability than the original DataFrame.
    '''
    X = X.copy()
    X['rooms_per_house'] = X['total_rooms'] / X['households']
    X['bedrooms_ratio'] = X['total_bedrooms'] / X['total_rooms']
    X['income_cat'] = pd.cut(X['median_income'],
                             bins=[0., 1.5, 3.0, 4.5, 6., np.inf],
                             labels=[0, 1, 2, 3, 4])
    return X

# pipeline to create new features with greater explainability. Returns a DataFrame
pipeline_create_features = Pipeline(steps=[
    ('features_creator', FunctionTransformer(create_extra_features, validate=False))
    ])
```

1

```
# pipeline for numerical features. Imputes the median (for missing values),
# transforms the data to be more Gaussian-like, and standardizes it
pipeline_num = Pipeline([
    ('median_imputer', SimpleImputer(strategy='median')),
    ('power_transformer', PowerTransformer(method='yeo-johnson', standardize=True))
    ])
```

2

3-4

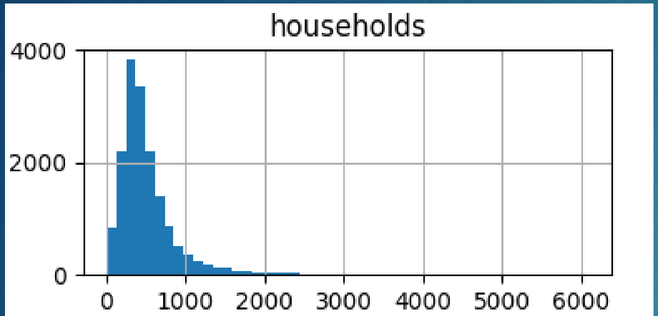
```
# impute the mode for categorical features
pipeline_cat = Pipeline([
    ('mode_imputer', SimpleImputer(strategy='most_frequent'))
    ])

# pipeline para hacer Ordinal Encoding sobre la feature: "ocean_proximity". Con un orden pre-escogido
pipeline_oh = Pipeline([
    ('mode_imputer', SimpleImputer(strategy='most_frequent')),
    ('ordinal_encoder', OrdinalEncoder(categories=[['INLAND', '<1H OCEAN', 'NEAR BAY', 'NEAR OCEAN', 'ISLAND']],
                                       handle_unknown='use_encoded_value',
                                       unknown_value=-1))
    ])
```

5

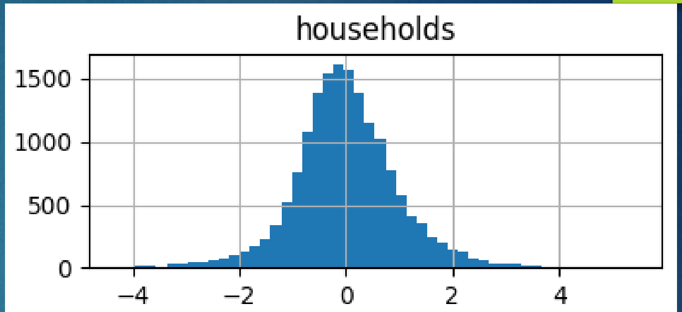
```
# pipeline to perform Ordinal Encoding on the feature: "ocean_proximity" with a predefined order
preprocessor = ColumnTransformer([
    ('exclude', 'drop', feats_to_exclude),
    ('process_num', pipeline_num, feats_num),
    ('process_cat', pipeline_cat, ['income_cat']),
    ('ord_encoder', pipeline_oh, ['ocean_proximity'])
    ],
    remainder='passthrough'
)
```

1-5

CALLIFORNIA HOUSING → REGRESOR



PowerTransformer()



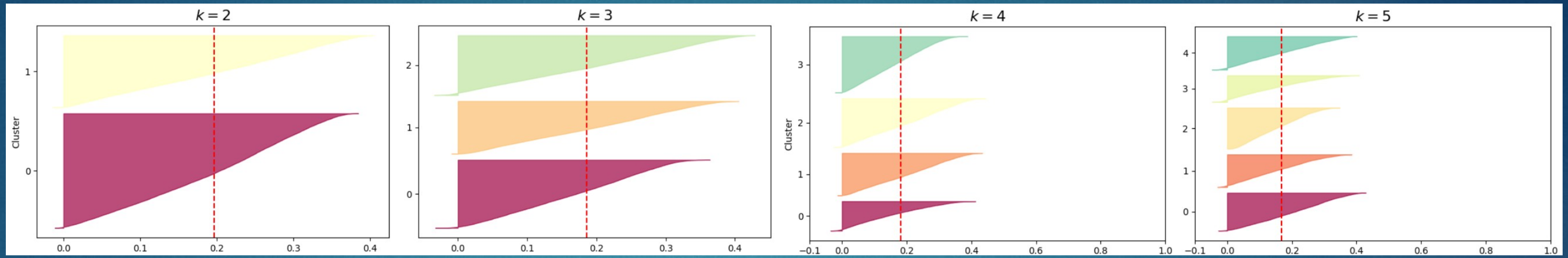

```python
# define models
models = {
    'linear': LinearRegression(),
    'random_forest': RandomForestRegressor(random_state=random_state),
    'XGBoost': XGBRegressor(random_state=random_state),
    'ridge': Ridge(),
    'lasso': Lasso(),
    'elastic_net': ElasticNet(),
    'gradient_boosting': GradientBoostingRegressor(),
    'SVR': SVR(),
    'k_neighbors': KNeighborsRegressor(),
    'catboost': CatBoostRegressor(random_state=random_state, verbose=0)
}

# define pipelines for each model
pipelines = {name: create_pipeline_with_target_transform(model) for name, model in models.items()}

# cross-validation with each pipeline
kwargs = {
    'cv': 10,
    'scoring': 'neg_root_mean_squared_error',
    'n_jobs': -5
}
for name, pipe in pipelines.items():
    result = -cross_val_score(pipe, X_train, y_train, error_score='raise', **kwargs)
    print(f'{name}: {np.mean(result):.4f}')
```

```python
# funtion for pipelines creation
def create_pipeline_with_target_transform(model):
    """
    Create pipelines with TransformedTargetRegressor for the different models.
    This time, there's no need to apply StandardScaler to the target transformation.
    """
    return Pipeline([
        ('create_feats', pipeline_create_features),
        ('preprocessor', preprocessor),
        ('model', TransformedTargetRegressor(
            regressor=model,
            transformer=PowerTransformer(method='box-cox', standardize=False))
        ])
```

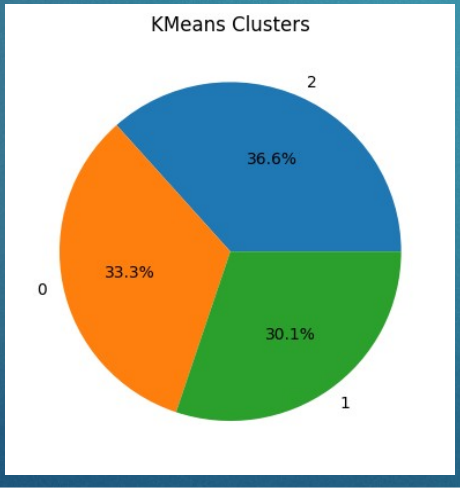
GridSearchCV()

```python
# RandomForest, XGBoost and CatBoost hyperparameters
param_grids = {
    'random_forest': {
        'model__regressor__n_estimators': [100, 200],
        'model__regressor__max_depth': [3, 6, 9],
        'model__regressor__min_samples_split': [5, 10],
        'model__regressor__min_samples_leaf': [2, 4],
        'model__regressor__max_features': ['sqrt', None],
    },

    'XGBoost': {
        'model__regressor__n_estimators': [100, 200],
        'model__regressor__max_depth': [3, 6, 9],
        'model__regressor__learning_rate': [0.01, 0.05, 0.1],
        'model__regressor__subsample': [0.8, 1.0],
        'model__regressor__colsample_bytree': [0.8, 1.0],
        'model__regressor__gamma': [0, 0.3, 0.5],
    },

    'catboost': {
        'model__regressor__iterations': [100, 200],
        'model__regressor__depth': [3, 6, 9],
        'model__regressor__learning_rate': [0.01, 0.05, 0.1],
        'model__regressor__bagging_temperature': [0, 0.5, 1],
        'model__regressor__random_strength': [1, 1.5, 2],
    }
}
```

CALIFORNIA HOUSING → CLUSTERING K-Mean




```python
# fit and predict for the k = 3
n_clusters = 3
pipeline_kmeans = Pipeline([('create_feats', pipeline_create_features),
                            ('preprocessor', preprocessor),
                            ('model', KMeans(n_clusters=n_clusters, random_state=random_state))
                           ])

# predictions addes to the original X. It could also be added to X_transformed instead
X['kmeans_cluster'] = pipeline_kmeans.fit_predict(X)
```



kmeans_cluster		0	1	2
longitude	mean	-119.578177	-120.354875	-118.915373
latitude	mean	35.351218	37.144071	34.641403
housing_median_age	mean	26.613434	28.002572	31.0045
total_rooms	mean	3224.533586	1919.382353	2690.903905
total_bedrooms	mean	531.162039	393.867609	662.655957
population	mean	1433.094419	1012.958695	1758.290404
households	mean	499.719365	353.95982	619.270285
median_income	mean	5.832559	2.806307	2.965051
median_house_value	mean	300422.237797	113257.698811	198943.32773
ocean_proximity	top	<1H OCEAN	INLAND	<1H OCEAN

Gracias

https://github.com/elecomexp/pipelines_team_challenge