GAM— Piecewise, LOESS, and GAM Splines - Python Version

Note: The original MARS section used the `sklearn-contrib-py-earth` package, which is **no longer actively maintained** and may fail to install on modern Python versions.
The MARS code is left **commented out** for reference — you can un-comment it if you successfully install `py-earth` (try `conda install -c conda-forge sklearn-contrib-py-earth`).
Equivalent flexibility can be achieved through **GAM** or **spline-based models**, which are shown below.

1. Setup and Data

```python
import sys
print("Active Python interpreter:", sys.executable)

import numpy as np, pandas as pd
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.linear_model import LinearRegression, Ridge
import inspect

# --- Fix for SciPy >= 1.13 removing .A from sparse matrices ---
import scipy.sparse as sp

if not hasattr(sp.spmatrix, "A"):
```

```python
    def _toarray(self):
        return self.toarray()
    sp.spmatrix.A = property(_toarray)
# -------------------------------------------------------------

np.random.seed(4321)
```

```
Active Python interpreter: /opt/anaconda3/bin/python
```

```python
from sklearn.datasets import fetch_openml

ames = fetch_openml(name="house_prices", as_frame=True).frame

keep = {
    "SalePrice":"SalePrice",
    "BedroomAbvGr":"Bedroom_AbvGr",
    "YearBuilt":"Year_Built",
    "MoSold":"Mo_Sold",
    "LotArea":"Lot_Area",
    "Street":"Street",
    "CentralAir":"Central_Air",
    "1stFlrSF":"First_Flr_SF",
    "2ndFlrSF":"Second_Flr_SF",
    "FullBath":"Full_Bath",
    "HalfBath":"Half_Bath",
    "Fireplaces":"Fireplaces",
    "GarageArea":"Garage_Area",
    "GrLivArea":"Gr_Liv_Area",
    "TotRmsAbvGrd":"TotRms_AbvGrd"
}

df = ames[list(keep.keys())].rename(columns=keep).dropna().copy()

X = df.drop(columns=["SalePrice"])
y = df["SalePrice"].values

num_cols = X.select_dtypes(include=[np.number]).columns.tolist()
cat_cols = [c for c in X.columns if c not in num_cols]

# Handle OneHotEncoder version changes and ensure dense output
if "sparse_output" in inspect.signature(OneHotEncoder).parameters:
```

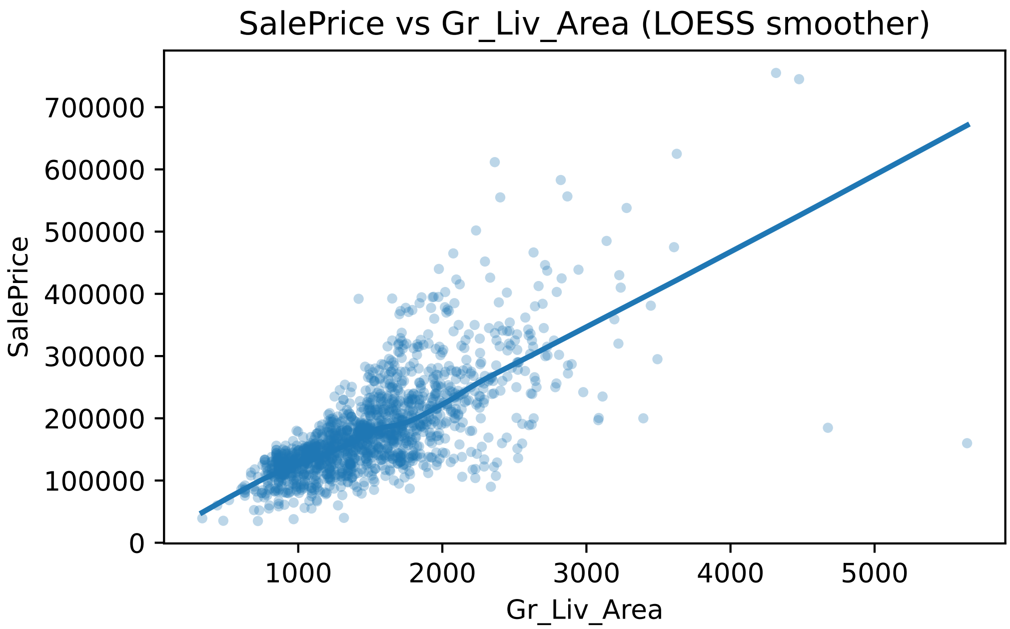
```python
    encoder = OneHotEncoder(drop="first", handle_unknown="ignore", sparse_output=False)
else:
    encoder = OneHotEncoder(drop="first", handle_unknown="ignore", sparse=False)

pre = ColumnTransformer([
    ("num", StandardScaler(), num_cols),
    ("cat", encoder, cat_cols)
], sparse_threshold=0)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=4321)
df.head()
```

	SalePrice	Bedroom_AbvGr	Year_Built	Mo_Sold	Lot_Area	Street	Central_Air	First_Flr_SF
0	208500	3	2003	2	8450	Pave	Y	856
1	181500	3	1976	5	9600	Pave	Y	1262
2	223500	3	2001	9	11250	Pave	Y	920
3	140000	3	1915	2	9550	Pave	Y	961
4	250000	4	2000	12	14260	Pave	Y	1145

---

2. Visual Check for Nonlinearity

```python
import statsmodels.api as sm
fig = plt.figure()
plt.scatter(df["Gr_Liv_Area"], df["SalePrice"], s=8, alpha=0.3)
low = sm.nonparametric.lowess(df["SalePrice"], df["Gr_Liv_Area"], frac=0.3, return_sorted=Tru
plt.plot(low[:,0], low[:,1], linewidth=2)
plt.title("SalePrice vs Gr_Liv_Area (LOESS smoother)")
plt.xlabel("Gr_Liv_Area"); plt.ylabel("SalePrice")
plt.tight_layout()
```

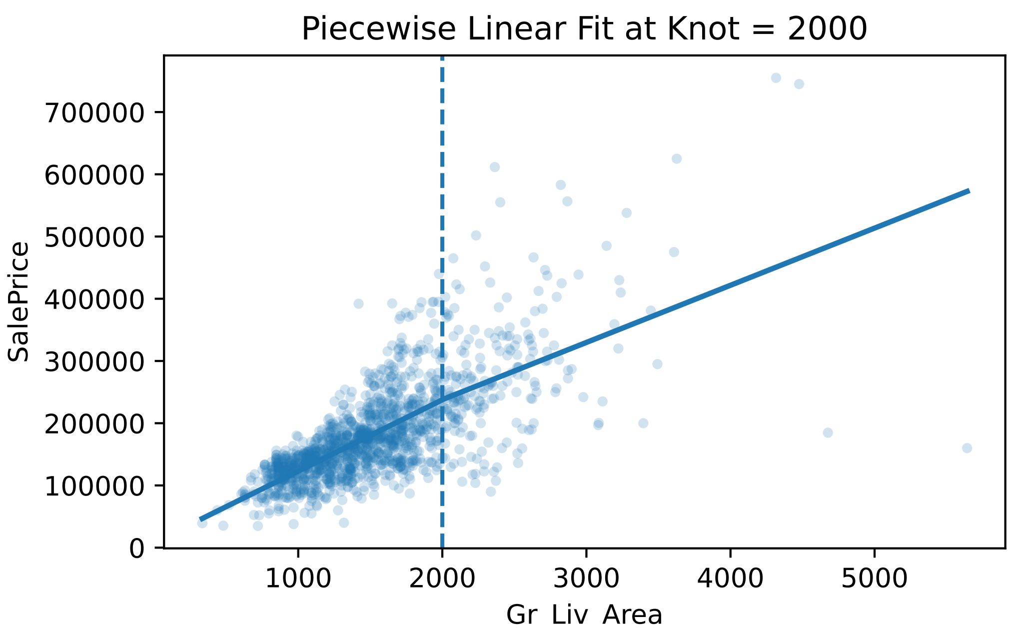


---

3. Piecewise Regression Example

```
class PiecewiseLinear:
    def __init__(self, knot=2000.0):
        self.knot = knot
        self.lin = LinearRegression()

    def _transform(self, x):
        x1 = np.asarray(x).reshape(-1,1)
        hx = np.maximum(0, x1 - self.knot)
        return np.hstack([x1, hx])

    def fit(self, x, y):
        Z = self._transform(x)
        self.lin.fit(Z, y)
        return self

    def predict(self, x):
        Z = self._transform(x)
```

```python
        return self.lin.predict(Z)

pw = PiecewiseLinear(knot=2000).fit(df["Gr_Liv_Area"].values, df["SalePrice"].values)

grid = np.linspace(df["Gr_Liv_Area"].min(), df["Gr_Liv_Area"].max(), 200)
pred = pw.predict(grid)

plt.figure()
plt.scatter(df["Gr_Liv_Area"], df["SalePrice"], s=8, alpha=0.2)
plt.plot(grid, pred, linewidth=2)
plt.axvline(2000, linestyle="--")
plt.title("Piecewise Linear Fit at Knot = 2000")
plt.xlabel("Gr_Liv_Area"); plt.ylabel("SalePrice")
plt.tight_layout()
```



---

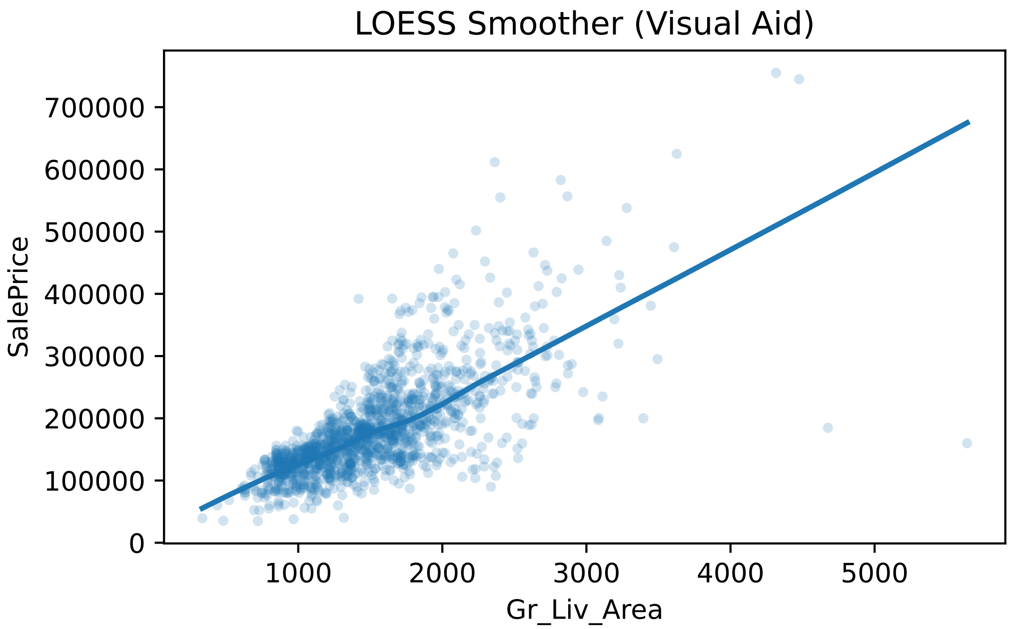
4. MARS (Commented Out — For Reference Only)

```
# The MARS implementation (py-earth) is not actively maintained and may fail to install on mo
# If you wish to try it, install from conda-forge:
#     conda install -c conda-forge sklearn-contrib-py-earth
#
# Example (commented out):
#
# from pyearth import Earth
#
# mars = Pipeline([("prep", pre), ("model", Earth(max_degree=2))]).fit(X_train, y_train)
# print(mars.named_steps["model"].summary())
#
# xs = np.linspace(X["Garage_Area"].min(), X["Garage_Area"].max(), 200)
# yhat = mars.predict(pd.DataFrame({"Garage_Area": xs}))
#
# plt.figure()
# plt.scatter(X_train["Garage_Area"], y_train, s=8, alpha=0.2)
# plt.plot(xs, yhat, linewidth=2)
# plt.title("MARS with Garage_Area (Commented Out Example)")
# plt.xlabel("Garage_Area"); plt.ylabel("SalePrice")
# plt.tight_layout()
```

---

5. LOESS Visualization

```
fig = plt.figure()
plt.scatter(df["Gr_Liv_Area"], df["SalePrice"], s=8, alpha=0.2)
low = sm.nonparametric.lowess(df["SalePrice"], df["Gr_Liv_Area"], frac=0.4, return_sorted=Tru
plt.plot(low[:,0], low[:,1], linewidth=2)
plt.title("LOESS Smoother (Visual Aid)")
plt.xlabel("Gr_Liv_Area"); plt.ylabel("SalePrice")
plt.tight_layout()
```



---

6. GAM and Spline Approaches (Modern Alternatives)

```
from pygam import LinearGAM, s
gam1 = LinearGAM(s(0)).fit(X_train[["Garage_Area"]].values, y_train)
print(gam1.summary())
gam2 = LinearGAM(s(0) + s(1)).fit(X_train[["Garage_Area","Gr_Liv_Area"]].values, y_train)
print(gam2.summary())
num_train = X_train[num_cols].values
gam3 = LinearGAM().fit(num_train, y_train)
print(gam3.summary())
```

```
LinearGAM
=============================================== ==========================================
Distribution:                        NormalDist Effective DoF:
Link Function:                     IdentityLink Log Likelihood:
23263.1874
Number of Samples:                         1022 AIC:
                                                AICc:
                                                GCV:                                      3
```

```
                                                 Scale:                                      30
                                                 Pseudo R-Squared:
==========================================================================================
Feature Function                  Lambda               Rank         EDoF         P > x
================================= ==================== ============ ============ ==========
s(0)                              [0.6]                20           11.7         1.11e-
16      ***
intercept                                              1            0.0          1.11e-
16      ***
==========================================================================================
Significance codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

WARNING: Fitting splines and a linear function to a feature introduces a model identifiabili
         which can cause p-values to appear significant when they are not.

WARNING: p-values calculated in this manner behave correctly for un-penalized models or model
         known smoothing parameters, but when smoothing parameters have been estimated, the p
values
         are typically lower than they should be, meaning that the tests reject the null too
None
LinearGAM
=============================================== ==========================================
Distribution:                        NormalDist Effective DoF:
Link Function:                     IdentityLink Log Likelihood:
22770.8323
Number of Samples:                         1022 AIC:
                                                 AICc:
                                                 GCV:                                        19
                                                 Scale:                                       1
                                                 Pseudo R-Squared:
==========================================================================================
Feature Function                  Lambda               Rank         EDoF         P > x
================================= ==================== ============ ============ ==========
s(0)                              [0.6]                20           12.6         1.11e-
16      ***
s(1)                              [0.6]                20           8.3          1.11e-
16      ***
intercept                                              1            0.0          1.11e-
16      ***
==========================================================================================
Significance codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

WARNING: Fitting splines and a linear function to a feature introduces a model identifiabili
```

```
         which can cause p-values to appear significant when they are not.

WARNING: p-values calculated in this manner behave correctly for un-penalized models or model
         known smoothing parameters, but when smoothing parameters have been estimated, the p
values
         are typically lower than they should be, meaning that the tests reject the null too
None
LinearGAM
=============================================== ==============================================
Distribution:                        NormalDist Effective DoF:
Link Function:                     IdentityLink Log Likelihood:
21995.395
Number of Samples:                         1022 AIC:
                                                AICc:
                                                GCV:                                        10
                                                Scale:                                       8
                                                Pseudo R-Squared:
==============================================================================================
Feature Function                  Lambda               Rank         EDoF         P > x
================================= ==================== ============ ============ ============
s(0)                              [0.6]                20           8.7          6.58e-
07      ***
s(1)                              [0.6]                20           12.8         1.11e-
16      ***
s(2)                              [0.6]                20           11.0         5.15e-
01
s(3)                              [0.6]                20           8.4          1.11e-
16      ***
s(4)                              [0.6]                20           8.6          1.11e-
16      ***
s(5)                              [0.6]                20           10.1         2.17e-
03      **
s(6)                              [0.6]                20           3.2          9.76e-
02      .
s(7)                              [0.6]                20           2.2          4.19e-
01
s(8)                              [0.6]                20           2.9          3.16e-
10      ***
s(9)                              [0.6]                20           9.0          1.11e-
16      ***
s(10)                             [0.6]                20           5.7          1.11e-
16      ***
s(11)                             [0.6]                20           6.5          3.78e-
```

```
01
intercept                                              1            0.0          1.11e-
16    ***
==========================================================================================
Significance codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

WARNING: Fitting splines and a linear function to a feature introduces a model identifiabilit
         which can cause p-values to appear significant when they are not.

WARNING: p-values calculated in this manner behave correctly for un-penalized models or model
         known smoothing parameters, but when smoothing parameters have been estimated, the p
values
         are typically lower than they should be, meaning that the tests reject the null too
None

/var/folders/v0/1fynz4dx4sd1h7nxwmvzrxf00000gn/T/ipykernel_95010/2456984288.py:3: UserWarning

KNOWN BUG: p-values computed in this summary are likely much smaller than they should be.

Please do not make inferences based on these values!

Collaborate on a solution, and stay up to date at:
github.com/dswah/pyGAM/issues/163


/var/folders/v0/1fynz4dx4sd1h7nxwmvzrxf00000gn/T/ipykernel_95010/2456984288.py:5: UserWarning

KNOWN BUG: p-values computed in this summary are likely much smaller than they should be.

Please do not make inferences based on these values!

Collaborate on a solution, and stay up to date at:
github.com/dswah/pyGAM/issues/163


/var/folders/v0/1fynz4dx4sd1h7nxwmvzrxf00000gn/T/ipykernel_95010/2456984288.py:8: UserWarning

KNOWN BUG: p-values computed in this summary are likely much smaller than they should be.

Please do not make inferences based on these values!

Collaborate on a solution, and stay up to date at:
```

github.com/dswah/pyGAM/issues/163

---

7. Regression Spline Example (Fallback)

```
from patsy import dmatrix

design_tr = dmatrix("bs(Garage_Area, df=6) + bs(Gr_Liv_Area, df=6)", data=X_train, return_typ
design_te = dmatrix("bs(Garage_Area, df=6) + bs(Gr_Liv_Area, df=6)", data=X_test, return_type

ridge = Ridge(alpha=1.0).fit(design_tr, y_train)
pred_spline = ridge.predict(design_te)
```

---

8. Model Comparison (Piecewise vs GAM vs Spline)

```
def rmse(y, yhat):
    try:
        return mean_squared_error(y, yhat, squared=False)
    except TypeError:
        return np.sqrt(mean_squared_error(y, yhat))

def mse(y, yhat): return mean_squared_error(y, yhat)
def rsq(y, yhat): return r2_score(y, yhat)

p_piece = pw.predict(X_test["Gr_Liv_Area"].values)

try:
    p_gam = gam3.predict(X_test[num_cols].values)
except Exception:
    p_gam = np.full_like(y_test, np.nan, dtype=float)

p_spline = pred_spline

cmp = pd.DataFrame({
```

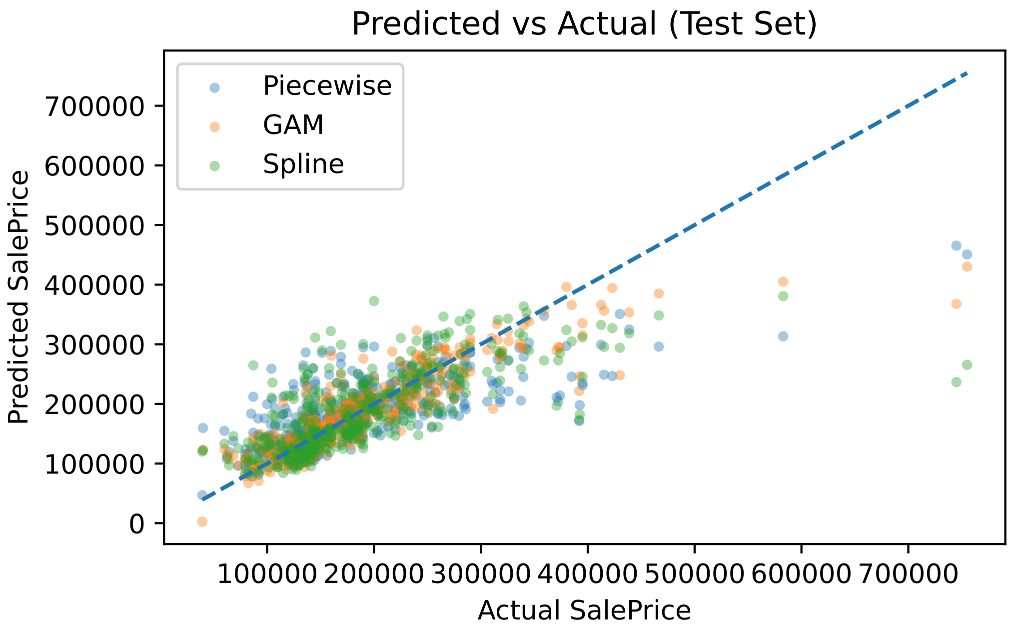
```
    "Model": ["Piecewise", "GAM (pyGAM)" if not np.isnan(p_gam).all() else "GAM (not availabl
    "RMSE": [rmse(y_test, p_piece), rmse(y_test, p_gam) if not np.isnan(p_gam).all() else np
    "MSE": [mse(y_test, p_piece), mse(y_test, p_gam) if not np.isnan(p_gam).all() else np.nan
    "R2": [rsq(y_test, p_piece), rsq(y_test, p_gam) if not np.isnan(p_gam).all() else np.nan
}).sort_values("R2", ascending=False)

cmp
```

	Model	RMSE	MSE	R2
1	GAM (pyGAM)	40805.867162	1.665119e+09	0.766556
0	Piecewise	58108.537553	3.376602e+09	0.526612
2	Regression Spline	60289.000524	3.634764e+09	0.490419

---

9. Predicted vs Actual Comparison

```
plt.figure()
plt.scatter(y_test, p_piece, s=8, alpha=0.4, label="Piecewise")
if not np.isnan(p_gam).all():
    plt.scatter(y_test, p_gam, s=8, alpha=0.4, label="GAM")
plt.scatter(y_test, p_spline, s=8, alpha=0.4, label="Spline")
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], linestyle="--")
plt.legend()
plt.title("Predicted vs Actual (Test Set)")
plt.xlabel("Actual SalePrice"); plt.ylabel("Predicted SalePrice")
plt.tight_layout()
```



---

10. Interpretation Summary

- **Piecewise:** adds simple thresholds, easy to explain.
- **GAMs:** provide smooth, interpretable nonlinearities.
- **Splines:** flexible approximations that behave like local polynomials.
- **MARS (optional):** similar conceptually but less maintained in Python; consider using R's `earth` instead.