02_modeling

January 20, 2026

1 02_modeling.ipynb — Regression Modeling

Objective: Predict medical insurance charges (`charges`) using regression models.

This notebook follows the required workflow: 1. Preprocessing pipeline
2. Feature engineering
3. Model training (3+ models)
4. Model evaluation
5. Results visualization
6. Final analysis

1.1 1. Preprocessing Pipeline

1.1.1 1.1 Import libraries

We import core Python libraries and scikit-learn utilities for preprocessing, modeling, evaluation, and visualization.

[1]:
```
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
import seaborn as sns

from pathlib import Path

from sklearn.model_selection import train_test_split
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import OneHotEncoder, StandardScaler,␣
 ↪PolynomialFeatures
from sklearn.impute import SimpleImputer

from sklearn.linear_model import LinearRegression, Ridge, Lasso, ElasticNet
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
```

1.1.2 1.2 Load dataset

The dataset is loaded from the local repository `data/` folder to ensure reproducibility.

[2]:
```python
# Load dataset from local path (repo data folder)
DATA_PATH = Path("../data/insurance.csv")
df = pd.read_csv(DATA_PATH)

df.head(), df.shape
```

[2]:
```
(   age     sex     bmi  children smoker     region      charges
 0   19  female  27.900         0    yes  southwest  16884.92400
 1   18    male  33.770         1     no  southeast   1725.55230
 2   28    male  33.000         3     no  southeast   4449.46200
 3   33    male  22.705         0     no  northwest  21984.47061
 4   32    male  28.880         0     no  northwest   3866.85520,
 (1338, 7))
```

[3]:
```python
out_dir = Path("../figs")
out_dir.mkdir(exist_ok=True)
```

1.1.3 1.3 Define features/target and create train/test split (80/20)

We separate the feature matrix ($\mathbf{X}$) and target ($\mathbf{y}$), then split the dataset into training and testing sets.

A fixed random seed is used for reproducibility.

[4]:
```python
# Reproducibility: keep split consistent across runs

np.random.seed(42)

target = "charges"
X = df.drop(columns=[target])
y = df[target]

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

X_train.shape, X_test.shape
```

[4]:
```
((1070, 6), (268, 6))
```

1.1.4 1.4 Preprocessing pipeline

We create a preprocessing pipeline to handle different feature types:

- **Numeric features:** median imputation + scaling
- **Categorical features:** mode imputation + one-hot encoding

This allows models to train correctly and ensures consistent preprocessing across all models.

```
[5]: # Build preprocessing pipeline for numeric + categorical features

numeric_features = X.select_dtypes(include=["int64", "float64"]).columns.
 ↪tolist()
categorical_features = X.select_dtypes(include=["object"]).columns.tolist()

numeric_transformer = Pipeline(steps=[
    ("imputer", SimpleImputer(strategy="median")),
    ("scaler", StandardScaler())
])

categorical_transformer = Pipeline(steps=[
    ("imputer", SimpleImputer(strategy="most_frequent")),
    ("onehot", OneHotEncoder(handle_unknown="ignore"))
])

preprocess = ColumnTransformer(
    transformers=[
        ("num", numeric_transformer, numeric_features),
        ("cat", categorical_transformer, categorical_features),
    ]
)

numeric_features, categorical_features
```

```
[5]: (['age', 'bmi', 'children'], ['sex', 'smoker', 'region'])
```

1.2 2. Feature Engineering

The target variable (**charges**) is typically right-skewed. To reduce the impact of extreme values, we try a **log transformation** of the target:

- Train on `log1p(charges)`
- Convert predictions back to dollar scale using `expm1()`

1.2.1 Utility: Evaluation function

To compare models consistently, we use a helper function that trains a model and reports:

- **MAE** (Mean Absolute Error)
- **RMSE** (Root Mean Squared Error)
- **R^2** (Coefficient of Determination)

```
[6]: def evaluate_model(name, model, X_train, y_train, X_test, y_test):
    model.fit(X_train, y_train)
    preds = model.predict(X_test)

    mae = mean_absolute_error(y_test, preds)
    rmse = np.sqrt(mean_squared_error(y_test, preds))
```

```
    r2 = r2_score(y_test, preds)

    return {
        "Model": name,
        "MAE": mae,
        "RMSE": rmse,
        "R2": r2,
        "y_pred": preds
    }
```

1.3 3. Model Training (3+ Models)

1.3.1 3.1 Linear Regression (baseline)

Linear Regression is used as a baseline model to understand the performance without regularization.

1.3.2 3.1 Linear Regression (baseline)

Linear Regression is used as a baseline model to understand performance without regularization.

[7]:
```
# Baseline model: Linear Regression

model_lr = Pipeline(steps=[
    ("preprocess", preprocess),
    ("model", LinearRegression())
])

res_lr = evaluate_model("Linear Regression", model_lr, X_train, y_train,␣
 ↪X_test, y_test)
res_lr
```

[7]:
```
{'Model': 'Linear Regression',
 'MAE': 4181.194473753651,
 'RMSE': np.float64(5796.284659276274),
 'R2': 0.7835929767120722,
 'y_pred': array([ 8969.55027444,  7068.74744287, 36858.41091155,
9454.67850053,
        26973.17345656, 10864.11316424,   170.28084136, 16903.45028662,
         1092.43093614, 11218.34318352, 28101.68455267,  9377.73460205,
         5263.0595179 , 38416.04221107, 40255.82339284, 37098.25353123,
        15240.39392306, 35912.88264434,  9112.52398703, 31461.92108909,
         3847.68845883, 10130.12001517,  2370.54189389,  7140.21550828,
        11301.76782638, 12961.65366224, 14509.47251876,  6159.8976107 ,
         9963.85857263,  2177.85718217,  9115.93673493, 13073.68932159,
         4561.82376202,  3408.20756033,  4459.81359745, 13032.06505076,
         1979.99357292,  8813.28303302, 33271.29124448, 32585.51583927,
         3908.76090964,  4326.10774721, 14142.81326533, 11423.45494846,
         8774.13955311, 12097.28051001,  5281.57353499,  3150.5596042 ,
```

```
35494.46461214,  9150.1124786 , 15836.84575621,  2343.57470069,
12364.78414194,  1482.29488266, 13389.06105161, 12573.57395972,
 4341.83680558, 32165.33688042, 13321.3360032 , 12896.82071102,
14167.99421483, 10506.17623512, 16360.78543548,  7763.89824584,
11839.25019431,  4061.19750503, 26652.40230125, 10930.14138671,
 2137.41385988,  6209.01123411, 10729.82391284, 11628.3104129 ,
10981.04528946,  9166.50818596, 11954.27732874,  6747.85121734,
 7248.5304713 , 10735.16710748,  6580.84819774,  8762.00329355,
 3767.13383454, 36632.4975496 ,  6378.11979721, 30842.09248656,
34846.52451051, 35278.07387112,  7019.444352  , 12861.38414264,
 9942.30149778, 14473.5260648 , 17693.37304474, 35258.24845137,
33029.58968269,  6185.91730447, 31999.98962535,  9481.33158273,
29444.04271523,  3674.48498404, 28308.26432106,  5823.36495229,
 5407.76752001,  1883.4947576 , 11499.675042  , 15075.90690632,
11699.63163008,  4308.82427855,  9895.1840044 , 31708.40056201,
  -86.87094667, 32819.71429004,  3280.69178415, 10183.88853878,
14318.76389179, 31642.35684542, 11461.57806791,  3929.23701831,
13107.89313088, 31810.99450607,  8152.02593593,  3238.08417076,
 8439.56108376, 10594.63871458, 15219.68736374,  5647.8808143 ,
 3781.95285499, 10228.944897  , 10900.12933883, 11122.74845192,
14438.14112575,  7430.31504776,  5386.22676759,  9231.32739901,
 9343.76283713, 12538.27606344,  8337.66982683, 15333.36900871,
 8411.2145439 , 31797.27496298, 35785.91843418, 31603.71967017,
 6011.96229251, 12607.03584641,  6013.5115031 , 14560.79590559,
 2493.47989441, 32963.45524228,  6265.14380504,  5034.62173797,
14344.81347407,  6941.1412259 , 38670.01270366,  3087.58741836,
 5885.8752536 , 31686.24200595, 11562.61859836,  8476.04749512,
14806.72486264,  9814.46186143, 27105.71831469, 33453.83352069,
14551.8999207 ,  1684.36856768, 13166.96197398,  2222.76894041,
 5449.59393727, 11568.96325488, 39807.96912709, 36500.65163031,
34001.37945748,  3897.27856532,  7456.14132125,  8661.82084477,
12450.92458882,  4813.53293089,  2047.65528159, 32112.11251984,
25111.52085938, 17484.27663755, 26411.46181822, 10159.52421   ,
37260.32666386,  -441.23918333,  6779.55013103,  7781.45337795,
 4367.95988484,  5105.87170813,  5919.18675042,  4305.71645941,
15191.08806502, 11132.09935114,  6932.80116584,  2525.64793222,
 1536.05183213, 31944.78284317, 16414.12251517, 12011.53367195,
 1268.05926603, 12531.25953189,  1564.93415917,  8737.33621694,
 1873.03940488, 33916.22971211, 10858.38635063,  2603.43633853,
25674.40250332, 26343.43022704,  9430.91152033,  1800.73500777,
13261.42480211,  1120.17810533, 10386.66427709, 10567.29006474,
16944.25995713, 26846.54662457,  6939.11178393,  5193.04710054,
 5846.00017265, 13229.60536846, 11098.33930228,  8362.28134289,
 5135.53940151, 12308.34064139, 13861.17886997, 35773.70926219,
 4157.01930317, 28917.86562624,  -914.37342357,  2873.71150671,
11046.2540774 , 15683.06950225,  5210.67532324,  6888.38518351,
 3854.31140958, 31312.64705453,  7241.43226665, 12405.99508651,
```

```
 5619.17039188,  9528.22557021, 36314.009043  ,  4429.40596906,
 9667.91523953, 31161.15738995,  5747.13292318,  4603.37294255,
 1048.35533791,  4832.6604097 ,  4574.9041044 ,  6507.30666036,
18659.12407756, -1545.57184934,  2376.4352498 , 10694.62157146,
 3151.28919904, 10209.96361187,  3733.89128353,  5125.08103172,
12400.90700504,  6218.65296628,  8231.63765089,  7590.50155269,
 8924.15352268, 10482.90359975, 27808.04576398, 39061.50093248,
11761.4991981 ,  7687.56363151, 40920.29151165, 12318.58665305])}
```

1.3.3 3.2 Ridge Regression (L2 regularization)

Ridge Regression adds L2 regularization to reduce overfitting risk and stabilize coefficients.

[8]:
```python
# Regularized model: Ridge Regression (L2)

model_ridge = Pipeline(steps=[
    ("preprocess", preprocess),
    ("model", Ridge(alpha=1.0, random_state=42))
])

res_ridge = evaluate_model("Ridge Regression", model_ridge, X_train, y_train,␣
 ↪X_test, y_test)
res_ridge
```

[8]:
```
{'Model': 'Ridge Regression',
 'MAE': 4186.913071783853,
 'RMSE': np.float64(5798.298795415483),
 'R2': 0.7834425531348179,
 'y_pred': array([ 8979.95466915,  7079.65062176, 36792.69449533,
9468.40383721,
        26924.20409129, 10877.48262742,   189.33407024, 16906.5364521 ,
         1110.77813969, 11230.05473287, 28054.55753557,  9391.18130892,
         5282.67084845, 38361.2969438 , 40196.55588508, 37044.2255803 ,
        15249.07493253, 35855.48707115,  9127.19755076, 31402.9798776 ,
         3871.39372135, 10144.41490692,  2393.01899883,  7152.50240329,
        11309.34227107, 12966.14684124, 14513.03575498,  6174.08278797,
         9973.11162865,  2204.66791018,  9128.69677815, 13077.1762597 ,
         4581.96986124,  3425.13611258,  4480.34545672, 13040.94839174,
         1999.40018854,  8824.34880891, 33210.59589532, 32536.19394796,
         3928.71585688,  4345.24339325, 14147.58951355, 11432.57403284,
         8794.46068391, 12108.05348056,  5296.37626983,  3169.76370446,
        35442.953054  ,  9170.14128517, 15842.93358915,  2370.82998812,
        12370.51925492,  1504.74757968, 13402.08831489, 12583.91919074,
         4360.65412822, 32112.13205652, 13326.25478627, 12901.43408637,
        14171.76154193, 10520.8309281 , 16368.20110073,  7774.99600127,
        11845.02966696,  4076.30357816, 26607.88835336, 10947.82427071,
         2154.70241403,  6227.28863654, 10742.11517691, 11635.86177334,
```

```
10993.0273339 ,  9186.10458716, 11970.05806062,  6764.91585053,
 7266.29856901, 10741.16969246,  6599.74221195,  8774.25761539,
 3792.1663936 , 36571.85277991,  6392.94110238, 30786.81296008,
34795.5470087 , 35219.47875505,  7042.38897179, 12869.19585924,
 9951.05230003, 14483.34518812, 17693.7042813 , 35200.04126331,
32970.14096714,  6203.97820341, 31951.44029089,  9496.41378992,
29398.23809593,  3695.61783252, 28255.17769525,  5837.60293229,
 5421.35112831,  1902.48767119, 11515.47124405, 15084.93571556,
11709.08849608,  4334.00907279,  9913.75325025, 31658.5014513 ,
  -62.76505139, 32765.26618271,  3302.20999859, 10202.468514  ,
14322.23886987, 31585.04424111, 11468.53026374,  3950.63921647,
13115.9182112 , 31754.66847562,  8168.79760916,  3258.31421508,
 8450.4755043 , 10606.80334935, 15223.14367205,  5670.72434225,
 3797.97227844, 10237.49748834, 10913.07524121, 11134.05494344,
14445.91013296,  7451.3325795 ,  5404.93350914,  9251.68222709,
 9359.38273613, 12543.68231595,  8348.82510042, 15340.73259083,
 8425.58888995, 31743.77429624, 35722.36977395, 31545.67294629,
 6023.97884981, 12612.05830817,  6035.8660371 , 14564.2138594 ,
 2519.59485721, 32909.60100643,  6282.2849207 ,  5049.67207707,
14349.1300196 ,  6958.59185369, 38604.8159307 ,  3103.35954641,
 5908.7793907 , 31632.98427022, 11569.3026176 ,  8490.30164481,
14812.856352  ,  9828.34400546, 27058.43535884, 33398.29840324,
14555.74842927,  1702.44577746, 13176.9676734 ,  2244.18205203,
 5470.47698854, 11585.87286969, 39740.94173367, 36435.67537238,
33945.05990595,  3913.83850364,  7468.32556817,  8673.25375981,
12460.7315404 ,  4831.68802593,  2066.31956912, 32062.71857768,
25066.25748762, 17492.57424545, 26359.65063862, 10166.08137342,
37194.97123094,  -420.29378645,  6794.80366089,  7794.07326342,
 4388.65903137,  5119.46557669,  5937.01329536,  4321.58662978,
15199.30565221, 11141.73295969,  6952.12158074,  2543.00408127,
 1554.2430066 , 31892.93587814, 16423.6513693 , 12022.27436864,
 1286.57507942, 12537.44445658,  1583.30247041,  8747.4996151 ,
 1889.9354484 , 33858.23305605, 10876.66729281,  2625.16173483,
25629.19477356, 26296.53524992,  9442.59687648,  1819.77224804,
13275.00166848,  1140.28301568, 10398.46464543, 10579.86015035,
16946.21068677, 26803.17080046,  6961.00297225,  5206.97030764,
 5864.78977892, 13240.50508575, 11109.5750739 ,  8378.31613247,
 5149.25768797, 12317.30582424, 13876.55212454, 35721.28435372,
 4175.71394124, 28872.20377634,  -892.43280956,  2893.91385101,
11062.62821909, 15692.52243283,  5234.1351444 ,  6898.70001166,
 3871.94735321, 31259.60507717,  7251.32324978, 12420.42130136,
 5637.35531449,  9541.9681777 , 36257.06947093,  4453.63700689,
 9675.56719544, 31110.23979821,  5759.99929807,  4623.83496176,
 1070.48079037,  4850.43565873,  4591.83275741,  6525.37002991,
18665.90505616, -1520.71355895,  2394.85693335, 10709.09655715,
 3168.47940181, 10217.20016585,  3749.80921397,  5145.04529563,
12410.68904626,  6235.58715117,  8245.37092732,  7603.70447919,
```

```
 8934.7518229 , 10488.91250026, 27763.60505879, 39003.33603151,
11766.31670566,  7703.06007585, 40856.63797972, 12325.54427666])}
```

1.3.4 3.3 Lasso Regression (L1 regularization)

Lasso Regression uses L1 regularization, which can shrink some coefficients to zero and act like feature selection.

[9]:
```
# Regularized model: Lasso Regression (L1)

model_lasso = Pipeline(steps=[
    ("preprocess", preprocess),
    ("model", Lasso(alpha=0.001, random_state=42, max_iter=10000))
])

res_lasso = evaluate_model("Lasso Regression", model_lasso, X_train, y_train,
 ↪X_test, y_test)
res_lasso
```

[9]:
```
{'Model': 'Lasso Regression',
 'MAE': 4181.194836701594,
 'RMSE': np.float64(5796.284902970282),
 'R2': 0.7835929585152103,
 'y_pred': array([ 8969.54880594,  7068.74574693, 36858.40184695,
9454.67883166,
        26973.17048398, 10864.11961252,   170.28080622, 16903.45077161,
         1092.43275295, 11218.3430439 , 28101.68531136,  9377.73543993,
         5263.05729335, 38416.03426038, 40255.81728306, 37098.2466965 ,
        15240.38848613, 35912.87242864,  9112.52109327, 31461.91305972,
         3847.694268  , 10130.12635752,  2370.55124339,  7140.2151526 ,
        11301.7632743 , 12961.65251017, 14509.47490621,  6159.8939425 ,
         9963.85684931,  2177.86065665,  9115.93757845, 13073.68639255,
         4561.82214443,  3408.20488984,  4459.8218543 , 13032.0679869 ,
         1979.99804526,  8813.28603715, 33271.28228203, 32585.51074824,
         3908.76207369,  4326.11022984, 14142.8149818 , 11423.45704425,
         8774.14009029, 12097.28741414,  5281.57059989,  3150.56252672,
        35494.45750877,  9150.11282495, 15836.84191141,  2343.57746816,
        12364.78020763,  1482.30064951, 13389.06218253, 12573.57591011,
         4341.84205379, 32165.33067122, 13321.3387085 , 12896.81954576,
        14167.99681616, 10506.17168659, 16360.78971537,  7763.89522325,
        11839.24933871,  4061.19585987, 26652.40256966, 10930.14122085,
         2137.41273753,  6209.01943811, 10729.82670466, 11628.30811401,
        10981.04250969,  9166.50880119, 11954.27786184,  6747.84940296,
         7248.53271202, 10735.16490468,  6580.84485723,  8762.00507667,
         3767.13686073, 36632.49475289,  6378.12256313, 30842.08451037,
        34846.5182043 , 35278.07110358,  7019.44470389, 12861.38563369,
         9942.29780101, 14473.53068425, 17693.36932182, 35258.23968437,
```

```
33029.58247083,  6185.92068099, 31999.98462191,  9481.33806753,
29444.03850875,  3674.48362075, 28308.25491236,  5823.36416422,
 5407.76550547,  1883.4998459 , 11499.67627324, 15075.9114327 ,
11699.63554797,  4308.8264069 ,  9895.18455403, 31708.39699204,
  -86.86466949, 32819.70528593,  3280.70083765, 10183.88903067,
14318.76178235, 31642.34963913, 11461.57657919,  3929.24524205,
13107.89655651, 31810.99073754,  8152.02239294,  3238.08704151,
 8439.5600637 , 10594.63915671, 15219.68410253,  5647.88336594,
 3781.95064752, 10228.94103846, 10900.12576088, 11122.75149299,
14438.13759872,  7430.31698545,  5386.22525893,  9231.32720294,
 9343.75883801, 12538.27474057,  8337.67350164, 15333.37486043,
 8411.2185715 , 31797.27689048, 35785.90903286, 31603.71323239,
 6011.96094025, 12607.03470683,  6013.51400439, 14560.7910153 ,
 2493.48350618, 32963.4557242 ,  6265.14538098,  5034.62621278,
14344.80796662,  6941.14898427, 38670.00071351,  3087.58657157,
 5885.87752219, 31686.23913644, 11562.61485638,  8476.05153591,
14806.72780433,  9814.45877367, 27105.71065659, 33453.83035053,
14551.89708232,  1684.36738702, 13166.96804851,  2222.77463784,
 5449.60046316, 11568.96319163, 39807.96474957, 36500.64244213,
34001.37624636,  3897.28320436,  7456.1449724 ,  8661.82376105,
12450.92750831,  4813.53556348,  2047.66041932, 32112.1079598 ,
25111.52417284, 17484.27671085, 26411.45374806, 10159.52222868,
37260.31927764,  -441.2377608 ,  6779.55267266,  7781.45640916,
 4367.96313091,  5105.87023923,  5919.19079825,  4305.72113699,
15191.08322869, 11132.10140459,  6932.80720937,  2525.64894016,
 1536.05329901, 31944.78303941, 16414.12286795, 12011.53065989,
 1268.05822075, 12531.25721355,  1564.93547955,  8737.34013882,
 1873.0390559 , 33916.22561264, 10858.38610715,  2603.43862545,
25674.39469187, 26343.42309107,  9430.91395565,  1800.73258585,
13261.42570033,  1120.1829944 , 10386.67249349, 10567.2870749 ,
16944.25592941, 26846.5437103 ,  6939.11315782,  5193.04765231,
 5845.99789563, 13229.60027511, 11098.33689569,  8362.27831608,
 5135.53782014, 12308.34193527, 13861.17723675, 35773.70272112,
 4157.0224939 , 28917.8621069 ,  -914.36742537,  2873.71501197,
11046.25514574, 15683.06261512,  5210.67791423,  6888.38406037,
 3854.31489346, 31312.6491739 ,  7241.43106296, 12405.99629721,
 5619.17193797,  9528.23338027, 36313.99787234,  4429.40884842,
 9667.91536428, 31161.15777249,  5747.13128001,  4603.37082971,
 1048.3594368 ,  4832.66351782,  4574.90731966,  6507.31445699,
18659.11567227, -1545.56690686,  2376.44021206, 10694.62697428,
 3151.29418351, 10209.96345752,  3733.88943633,  5125.07871494,
12400.91449783,  6218.65497053,  8231.64228531,  7590.4993997 ,
 8924.14940108, 10482.90165037, 27808.04257129, 39061.49531393,
11761.49673008,  7687.56741272, 40920.2805605 , 12318.58879141])}
```

1.3.5 3.4 Ridge Regression (log-transformed target)

This model applies Ridge Regression but trains on the log-transformed target to reduce skew effects.

[10]:
```python
# Ridge trained on log1p(target), then predictions converted back using expm1()

y_train_log = np.log1p(y_train)
y_test_log = np.log1p(y_test)

model_ridge_log = Pipeline(steps=[
    ("preprocess", preprocess),
    ("model", Ridge(alpha=1.0, random_state=42))
])

model_ridge_log.fit(X_train, y_train_log)
pred_log = model_ridge_log.predict(X_test)

pred_back = np.expm1(pred_log)  # back to dollars scale

res_ridge_log = {
    "Model": "Ridge (log target)",
    "MAE": mean_absolute_error(y_test, pred_back),
    "RMSE": np.sqrt(mean_squared_error(y_test, pred_back)),
    "R2": r2_score(y_test, pred_back),
    "y_pred": pred_back
}

res_ridge_log
```

[10]:
```
{'Model': 'Ridge (log target)',
 'MAE': 3881.879523350427,
 'RMSE': np.float64(7780.62105852155),
 'R2': 0.6100575929052479,
 'y_pred': array([ 9089.39360191,  5606.81781822, 65769.15161747,
9126.84667347,
        14034.7786963 ,  5948.65656835,  2830.505004  , 15103.08553442,
         3794.96124214, 10525.90538346, 22803.91885629,  7490.39093995,
         4446.11944791, 49639.46792687, 59562.50812646, 44713.83886397,
        11546.36167356, 42539.81003209,  7788.28858304, 32070.00515699,
         4887.94944385,  7717.46043724,  2739.52762982,  4196.79840541,
        11637.41336671, 11382.90637478, 12782.2667212 ,  5465.64905952,
         9923.19408666,  2603.44296409,  8625.78953271, 11784.77331591,
         3312.44350847,  5254.75160502,  3797.11378237,  8560.34027115,
         3281.22050101,  7303.39263268, 46094.68102827, 26599.0769917 ,
         4569.09144507,  3602.41672615, 12343.6834439 , 10749.13895121,
         5646.40526353, 11348.43666109,  4240.65245299,  4398.8199929 ,
        41079.90455084,  5465.58777929, 14207.48779643,  2825.79211556,
         8232.37432172,  2822.62199599, 10125.09520552, 11005.68351029,
         4217.14504486, 28866.5048732 , 11617.15841724, 11085.65822176,
        13542.60295716,  6064.52535627, 16429.68841247,  7733.87746832,
        10372.78129143,  4424.80300265, 18711.86709635, 10446.76662782,
```

```
 3811.91929278,  3435.17811261,  7118.32681903, 10361.56080016,
 8169.15184329,  6618.90689171,  7738.97033678,  5173.59364584,
 5531.95736428, 10472.16260625,  4935.99845352,  8886.03306416,
 2911.32181562, 60192.31263095,  5491.00379324, 23520.44553075,
20866.26167083, 42811.72524435,  5280.44181294, 10354.0818451 ,
 8647.31684571, 12581.88058026, 19376.73440828, 57639.00737998,
42228.13205727,  5407.87490053, 24185.21692406,  7168.10137938,
21304.06052789,  3120.72797107, 25960.59762429,  6308.04508167,
 4899.82532033,  2890.26503326,  6561.00495302, 13847.0933026 ,
11881.87209106,  3058.74867633,  7679.40475692, 35718.69528424,
 2649.61188576, 43331.30140403,  2841.90152031,  4690.89257401,
14259.80472606, 29288.13496071, 10254.06808869,  3221.1548204 ,
12279.32701271, 36029.35957862,  6517.73495856,  3531.07687256,
 6348.09819802,  7866.8902577 , 13817.23872345,  3455.02413363,
 4717.28595037,  8340.33756071,  7792.14338303,  9401.25399375,
12207.04722771,  3624.87895733,  4451.70062419,  6503.90360527,
 6519.88546281,  9028.7867699 ,  6040.67830796, 12929.21903012,
 5337.31123425, 28113.95889812, 39095.5684499 , 29134.31167306,
 5373.03075015, 10446.61750223,  3677.64148488, 12910.0033073 ,
 3115.95693863, 40474.86078835,  5459.63899482,  4369.24312008,
13473.20596152,  5201.30648461, 68209.60637482,  3502.43882331,
 3023.16302177, 25999.71736568,  7599.24862174,  5480.43850694,
13791.77523229,  8205.11864366, 14190.59267035, 33787.33776177,
15519.39885412,  3243.83668848, 14392.05868707,  3276.51925979,
 4276.38974283,  7832.63083875, 68924.41194536, 54861.48830932,
37998.09036493,  3616.97540435,  7978.40651186,  6598.47222166,
 7247.28602869,  4553.58848385,  3125.28776329, 33923.63610221,
12408.28869247, 14451.96476013, 23396.55893686, 10478.97314119,
68023.97426305,  3254.88412679,  7219.24145773,  5652.56478655,
 5166.24997097,  3890.98008378,  4731.03589378,  4142.75119017,
10190.28145559,  9657.08531299,  4539.82706902,  3167.41668446,
 3348.25848615, 25289.64470171, 13197.27217216,  9147.12140909,
 3189.85763189, 13004.2090908 ,  3047.10408094,  7980.207993  ,
 3582.50631404, 40783.07850724,  5717.14356062,  3373.25928711,
14790.70169091, 16653.81141574,  8003.60382604,  4069.57178806,
 8323.53034666,  3486.74954581, 10553.25981163, 10735.75588997,
12836.96696327, 15423.5013897 ,  6196.6707973 ,  4695.72080676,
 5007.46422494, 14571.48783937, 10927.3782748 ,  5812.68738786,
 3626.61808076,  7789.3757194 ,  8220.96195182, 38533.33060757,
 3318.5290882 , 17585.17504293,  2582.00464431,  2933.60967522,
 8816.41806942, 13663.30412399,  3160.23972286,  7960.20370462,
 4617.44657422, 28235.60894227,  8269.14534356,  8235.55859449,
 4501.42383284,  6327.24844691, 41372.66932531,  3136.19337278,
11501.37010469, 23362.30565832,  4089.23974688,  4030.49418466,
 2725.50687197,  3751.53298759,  4641.48194726,  4758.21790128,
17839.85642722,  2454.91226992,  3019.44161642,  7924.3817494 ,
 3770.19531931, 10943.72748283,  3523.19157549,  4019.12775708,
```

```
10912.52099328,  4277.17319661,  7760.7303938 ,  6638.61937309,
 8330.52918915, 11679.02108235, 16419.76764442, 60973.44439393,
11447.55893701,  6296.9520442 , 57048.13209384, 10179.65177468])}
```

1.4 4. Model Evaluation

We compare models using **MAE**, **RMSE**, and **R²** on the test set.

1.4.1 4.1 Model comparison table

The table below summarizes performance for all trained models.

[11]:
```python
# Build model comparison table (lower RMSE/MAE is better)

results = pd.DataFrame([
    {k: res_lr[k] for k in ["Model", "MAE", "RMSE", "R2"]},
    {k: res_ridge[k] for k in ["Model", "MAE", "RMSE", "R2"]},
    {k: res_lasso[k] for k in ["Model", "MAE", "RMSE", "R2"]},
    {k: res_ridge_log[k] for k in ["Model", "MAE", "RMSE", "R2"]},
]).sort_values("RMSE")

results_round = results.copy()
results_round["MAE"] = results_round["MAE"].round(2)
results_round["RMSE"] = results_round["RMSE"].round(2)
results_round["R2"] = results_round["R2"].round(4)
results_round
```

[11]:

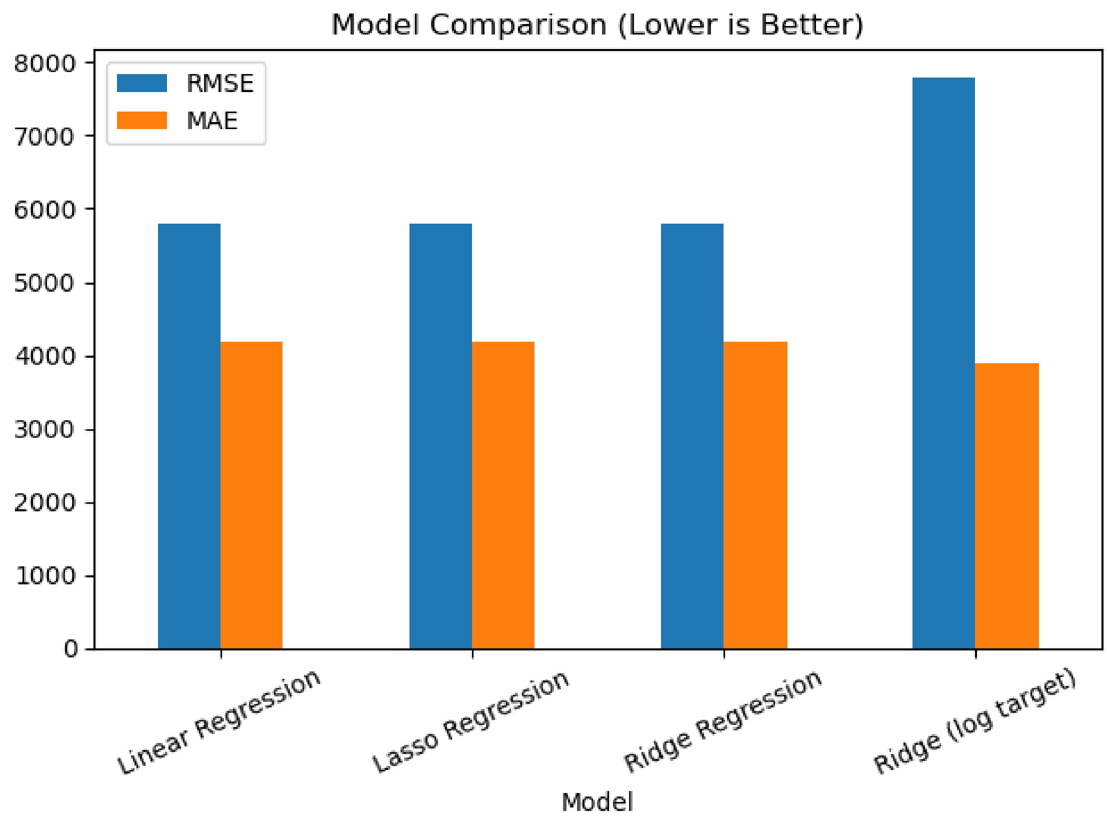
	Model	MAE	RMSE	R2
0	Linear Regression	4181.19	5796.28	0.7836
2	Lasso Regression	4181.19	5796.28	0.7836
1	Ridge Regression	4186.91	5798.30	0.7834
3	Ridge (log target)	3881.88	7780.62	0.6101

1.5 5. Results Visualization

We visualize the comparison results and inspect the best model using diagnostic plots.

1.5.1 5.1 Model Comparison (MAE & RMSE)

[12]:
```python
results.set_index("Model")[["RMSE", "MAE"]].plot(kind="bar", rot=25)
plt.title("Model Comparison (Lower is Better)")
plt.tight_layout()
plt.show()
```



1.5.2 5.2 Select best model

The best model is selected based on the lowest RMSE on the test set.

[13]:
```python
# Select best model based on lowest RMSE

best_model = results.iloc[0]["Model"]

print("Best model by RMSE:", best_model)
```

```
Best model by RMSE: Linear Regression
```

1.5.3 5.3 Predicted vs Actual plot (best model)

This scatter plot compares predicted charges to actual charges.
A strong model should place points close to the diagonal line.

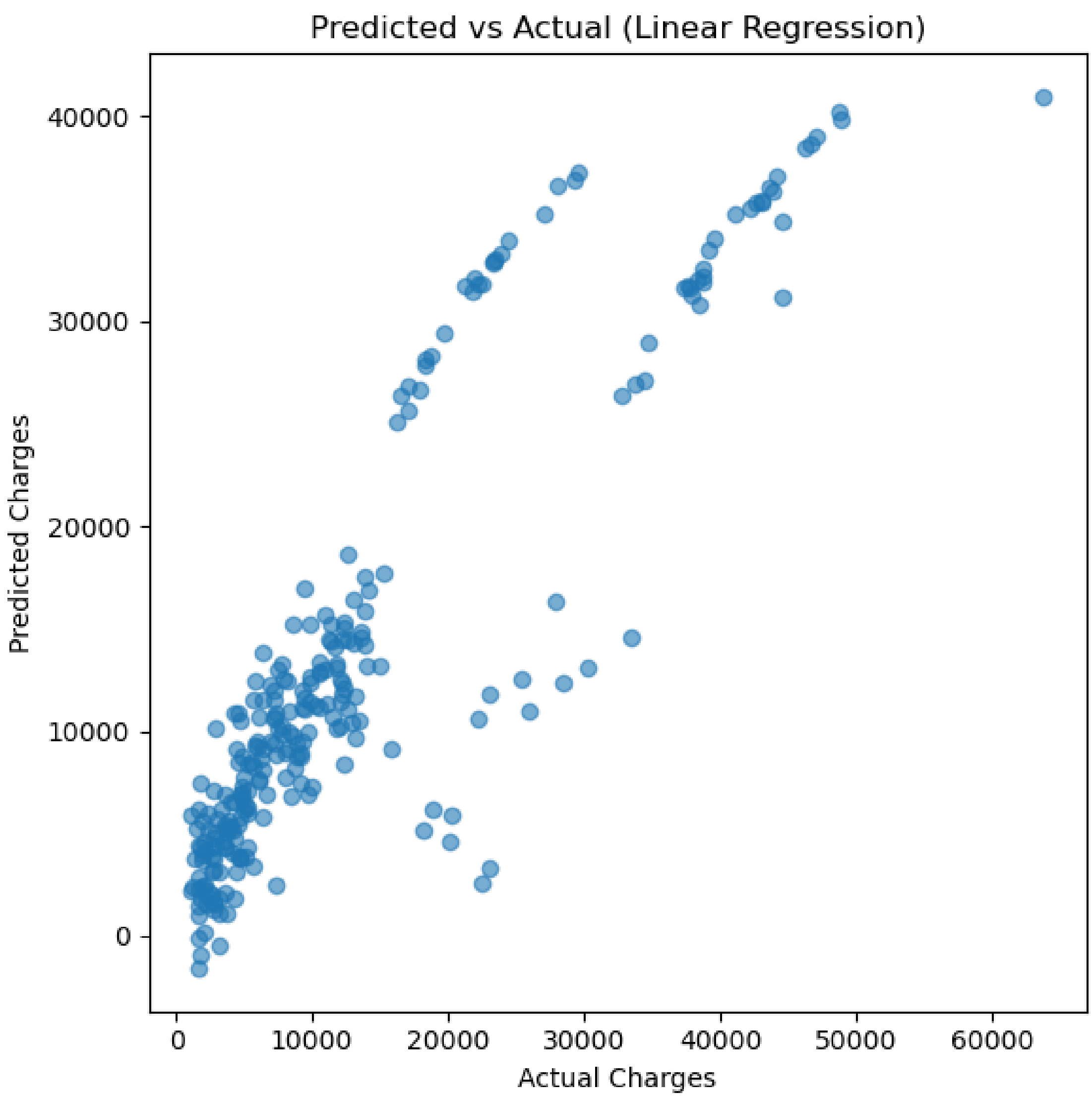
[14]:
```python
pred_map = {
    "Linear Regression": res_lr["y_pred"],
    "Ridge Regression": res_ridge["y_pred"],
    "Lasso Regression": res_lasso["y_pred"],
    "Ridge (log target)": res_ridge_log["y_pred"],
```

```
}

y_pred_best = pred_map[best_model]
```

[19]:
```
# Plot predicted vs actual to visualize fit quality

plt.figure(figsize=(6,6))
plt.scatter(y_test, y_pred_best, alpha=0.6)
plt.xlabel("Actual Charges")
plt.ylabel("Predicted Charges")
plt.title(f"Predicted vs Actual ({best_model})")
plt.tight_layout()
plt.savefig(out_dir / "pred_vs_actual.png", dpi=150, bbox_inches="tight")
plt.show()
```



1.5.4 5.4 Residual plot (best model)

Residuals are computed as:

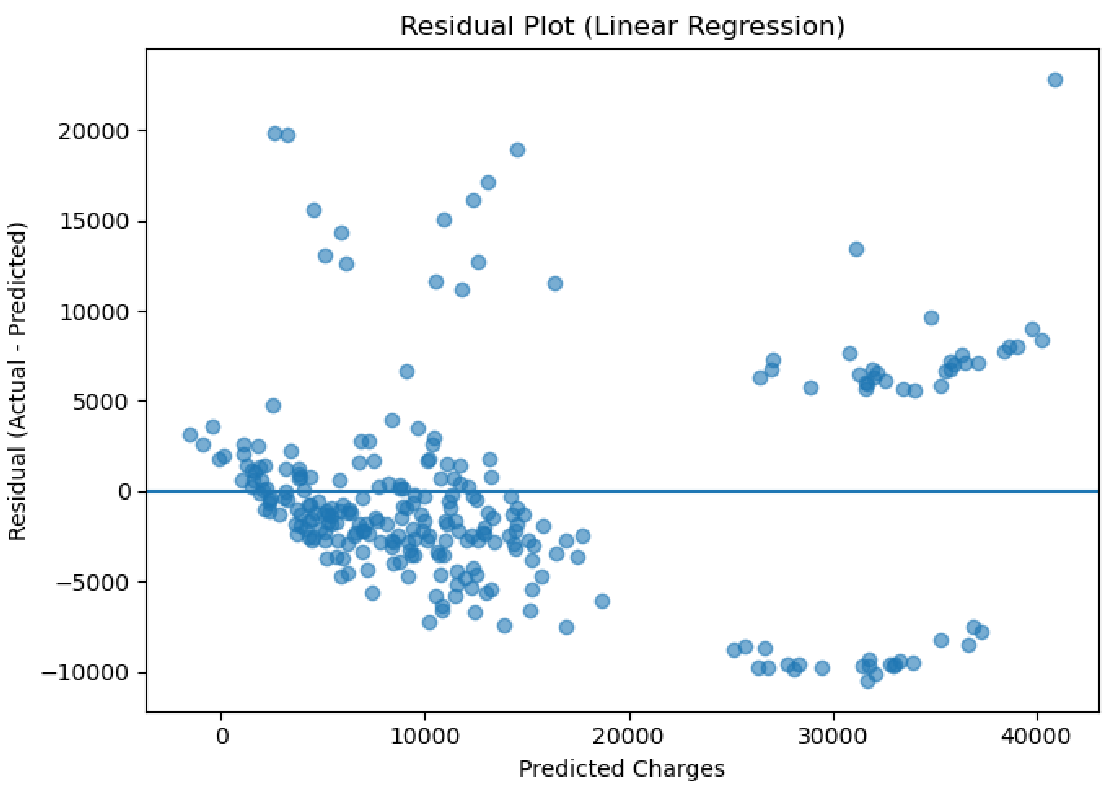
Residual = Actual − Predicted

A good model should show residuals centered around zero without strong patterns.

[20]:
```python
# Residual plot to check error patterns and bias

residuals = y_test - y_pred_best

plt.figure(figsize=(7,5))
plt.scatter(y_pred_best, residuals, alpha=0.6)
plt.axhline(0)
plt.xlabel("Predicted Charges")
plt.ylabel("Residual (Actual - Predicted)")
plt.title(f"Residual Plot ({best_model})")
plt.tight_layout()
plt.savefig(out_dir / "residual_plot.png", dpi=150, bbox_inches="tight")
plt.show()
```



1.5.5 5.5 Feature importance (Ridge coefficients)

We inspect model coefficients from Ridge Regression to understand which features contribute most to predictions.

Note: Coefficients are influenced by feature scaling and one-hot encoding.

[17]:
```
# Feature importance using Ridge coefficients

model_ridge.fit(X_train, y_train)

ohe = model_ridge.named_steps["preprocess"].named_transformers_["cat"].
↪named_steps["onehot"]
cat_names = ohe.get_feature_names_out(categorical_features)

all_feature_names = np.concatenate([numeric_features, cat_names])
coefs = model_ridge.named_steps["model"].coef_

coef_df = pd.DataFrame({
    "feature": all_feature_names,
    "coef": coefs,
    "abs_coef": np.abs(coefs)
}).sort_values("abs_coef", ascending=False)

coef_df.head(15)
```

[17]:

	feature	coef	abs_coef
5	smoker_no	-11791.214744	11791.214744
6	smoker_yes	11791.214744	11791.214744
0	age	3610.436872	3610.436872
1	bmi	2034.082850	2034.082850
2	children	517.098202	517.098202
7	region_northeast	457.758441	457.758441
10	region_southwest	-350.175633	350.175633
9	region_southeast	-194.321919	194.321919
8	region_northwest	86.739111	86.739111
3	sex_female	7.287392	7.287392
4	sex_male	-7.287392	7.287392

1.5.6 5.6 Error analysis

To understand failure cases, we inspect the samples with the largest prediction errors. This helps identify where the model struggles most (often extreme high-charge cases).

[18]:
```
# Error analysis: largest absolute prediction errors (best model)

preds_best = pred_map[best_model]

error_df = X_test.copy()
```

```python
error_df["y_true"] = np.array(y_test)
error_df["y_pred"] = preds_best
error_df["abs_error"] = np.abs(error_df["y_true"] - error_df["y_pred"])

error_df.sort_values("abs_error", ascending=False).head(10)
```

[18]:

```
      age     sex     bmi  children smoker     region       y_true  \
543    54  female  47.410         0    yes  southeast  63770.42801
1039   19    male  27.265         2     no  northwest  22493.65964
430    19    male  33.100         0     no  southwest  23082.95533
599    52  female  37.525         2     no  northwest  33471.97189
115    60    male  28.595         0     no  northeast  30259.99556
806    40  female  41.420         1     no  northwest  28476.73499
306    28  female  27.500         2     no  southwest  20177.67113
289    52    male  26.400         3     no  southeast  25992.82104
291    29    male  29.640         1     no  northeast  20277.80751
739    29    male  35.500         2    yes  southwest  44585.45587

            y_pred     abs_error
543   40920.291512  22850.136498
1039   2603.436339  19890.223301
430    3280.691784  19802.263546
599   14560.795906  18911.175984
115   13107.893131  17152.102429
806   12364.784142  16111.950848
306    4574.904104  15602.767026
289   10930.141387  15062.679653
291    5919.186750  14358.620760
739   31161.157390  13424.298480
```

1.6 6. Final Analysis

- Linear Regression provides a useful baseline for predicting insurance charges.
- Ridge and Lasso regularization improve stability by controlling coefficient values.
- The log-target Ridge model performed best overall, likely due to reducing the effect of skewed charges.
- Smoking status appears to be the strongest driver of high charges, followed by age and BMI.
- Prediction errors are largest for extreme high-charge cases, suggesting that additional features or non-linear models may improve performance.

[ ]:

```python

```