我这部分为大家演示一个使用sklearn预测房价的小案例。

整个工作流包括加载原始数据 → 清洗数据 → 特征工程 → 模型训练 → 评估 → 调优，最后得到一个最佳模型，并使用模型进行预测。

我们先来看下原始数据。加州住房数据集包含 ****2w+条记录****，覆盖了1990年加州所有街区的统计信息。每个街区有 ****8个关键特征****，比如这里的经度纬度定位了地理位置，房龄中位数反映了房屋新旧程度，而收入中位数则是预测房价的黄金指标。

然后我们先执行一下脚本，先得到一些信息，再来进行解释

可以看到这边能够得到一些直方图的图片，首先是分布直方图。

通过分布直方图（点击展示），我们发现了几个信息：  
第一，****收入中位数****呈现明显的 ****三峰分布****（手势指向峰值）。这可能对应加州的低收入、中产和高收入群体——而高收入群体的街区往往集中在硅谷等科技中心。  
第二，****总房间数****的分布（手势向右拖尾）暴露了严重右偏——绝大部分的街区房间数低于2,000间，但最大值竟达到 39320****间****！这显然是大学城或军事基地等特殊区域。  
第三，****房价中位数****的分布（点击切换图表）显示，95%的房价集中在0-50万美元之间，但存在少量超过50万美元的极端值。这些异常值可能是海滨豪宅或数据错误，需要进一步验证。

加载完数据之后，我们就要做到机器学习中至关重要的一步 - 数据预处理，我们给模型 “投喂” 数据前，也得精心处理一番。​

X = df.drop('medianHouseValue', axis=1)，这里我们把数据中的medianHouseValue房价中位数这一列剔除，单独拎出来，因为它是我们要预测的目标变量，好比考试要解答的题目，而剩下的数据集X就是题目里的已知条件。

y = df['medianHouseValue']，把目标变量存到y里，这样就完成了自变量和因变量的分离。​

接下来，X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)，这一步是把数据集拆分成训练集和测试集。训练集就像学生的练习题，用来让模型学习规律；测试集则是考试卷，检验模型学得怎么样。这里我们拿出 20% 的数据作为测试集，random\_state=42就像给拆分过程设定了一个固定的 “配方”，保证每次拆分结果都一样，方便我们重复实验。​

再看处理缺失值的部分，total\_bedrooms\_median = X\_train['totalBedrooms'].median()，我们算出训练集里totalBedrooms列的中位数。然后用这个中位数填补训练集和测试集里的缺失值，

X\_train['totalBedrooms'].fillna(total\_bedrooms\_median, inplace=True)和X\_test['totalBedrooms'].fillna(total\_bedrooms\_median, inplace=True)，让数据完整起来，避免缺失值影响模型学习。​

最后，scaler = StandardScaler()创建了一个标准化工具，

X\_train\_scaled = scaler.fit\_transform(X\_train)和X\_test\_scaled = scaler.transform(X\_test)对数据进行标准化处理。这一步就像给不同单位的测量数据统一成米、千克这样的标准单位，让模型能更公平地处理每个特征，提升预测的准确性。​

数据预处理完成后，我们就来到了模型搭建的环节。就像建筑师面对一块空地，会尝试用不同的设计方案盖房子，我们要用多种模型去挖掘数据里的规律，解决回归预测问题。​

首先是线性回归模型。lin\_reg = LinearRegression()创建了一个线性回归对象，lin\_reg.fit(X\_train\_scaled, y\_train)让它在标准化后的训练数据里 “学习”。

线性回归就像在数据的海洋里寻找一条最能代表数据趋势的直线，它假设自变量和因变量之间存在线性关系，简单直接，就像用一把直尺在散点图上比划，试图找到一条能串起大多数数据点的线。​

第二个是随机森林模型。rf\_reg = RandomForestRegressor(n\_estimators=100,max\_depth=10, random\_state=42)创建了一个由 100 棵决策树组成、树最大深度为 10 的随机森林模型，rf\_reg.fit(X\_train\_scaled, y\_train)开启训练。决策树就像一个 “数据侦探”，通过不断问问题拆分数据，找到规律。而随机森林就是一群这样的 “侦探”，每棵树都从不同角度分析数据，最后把大家的结论汇总投票，这样得出的结果往往比单个侦探更可靠，能捕捉到数据里复杂的非线性关系。​

接着是 XGBoost 模型。xgb\_reg = XGBRegressor(n\_estimators=300, learning\_rate=0.05, max\_depth=6)设定了 300 个基学习器、学习率 0.05 和最大深度 6 的 XGBoost 模型，xgb\_reg.fit(X\_train\_scaled, y\_train)开始训练。XGBoost 同样基于决策树，但它更像是一个 “学霸团队”，每棵新树都专注于纠正前面树的错误，学习率就像团队学习的进度条，控制每次改进的幅度。它通过高效的算法和正则化手段，在保证学习能力的同时避免过拟合。​

最后是 Stacking 集成模型。estimators = [('rf', RandomForestRegressor()), ('xgb', XGBRegressor())]定义了两个基础模型，随机森林和 XGBoost，stack\_reg = StackingRegressor(estimators, final\_estimator=LinearRegression())把它们组合起来，再用线性回归作为最终模型进行整合，stack\_reg.fit(X\_train\_scaled, y\_train)完成训练。这就好比组建一支队伍，每个基础模型都是有特长的队员，有的捕捉非线性关系，有的防止过拟合，而最终的线性回归模型就像教练，综合队员们的表现，给出最合理的预测，实现 “1 + 1> 2” 的效果 。​

前面我们搭建了多个回归模型，那我们搭建好模型之后需要对模型进行评估来判断模型的好坏。​

y\_pred = model.predict(X\_test)这一步，让模型用之前没见过的测试数据X\_test进行预测。​

随后，我们用三个关键指标来评判模型表现。第一个是 MAE（平均绝对误差），print(f"MAE: {mean\_absolute\_error(y\_test, y\_pred):.3f}")，它计算的是预测值和真实值之间误差的平均绝对值。打个比方，如果预测房价，MAE 就是每次预测误差的平均大小，数值越小，说明模型预测得越准，就像射箭离靶心越近。​

第二个指标是 MSE（均方误差），print(f"MSE: {mean\_squared\_error(y\_test, y\_pred):.3f}")。它和 MAE 类似，但对较大的误差 “惩罚” 更重，因为计算时误差先平方再平均。这就好比在考试中，大的错误会被扣分更多，通过 MSE 我们能更敏锐地发现模型的严重失误。​

第三个指标 R² Score（决定系数），print(f"R² Score: {r2\_score(y\_test, y\_pred):.3f}")。它的取值范围在 0 到 1 之间，1 代表模型完美预测，0 则表示模型预测和瞎猜差不多。R² Score 就像模型的 “实力认证”，数值越高，说明模型解释数据变化的能力越强，越值得信赖。​

除了这些数值指标，我们还通过可视化来直观感受模型效果。

我们把真实值和预测值绘制成散点，每个点代表一次预测，生成一个直方图来观察。​

接下来，对每个模型 进行评估。线性回归模型的MAE,MSE和R平方的值分别是

如果我们想要高效产出准确的模型，需要一套精密的流水线。下面这部分就是搭建数据预处理和模型优化的 “超级流水线”。​

首先看preprocessor = ColumnTransformer(...)这部分。ColumnTransformer 可以把他想象成 “数据大管家”，能同时处理不同类型的数据任务。这里面有两个 “助手”：第一个是('num', StandardScaler(), ['medianIncome', 'housingMedianAge', 'totalRooms'])，它用StandardScaler对指定的数值列进行标准化处理，就好比把不同单位的测量数据统一成标准单位，让模型更好 “消化”；第二个('poly', PolynomialFeatures(degree=2), ['medianIncome', 'totalRooms'])，它像一个 “数据变形大师”，通过PolynomialFeatures对指定列进行多项式转换，生成二次项特征，挖掘数据间更复杂的非线性关系。​

接着，pipeline = Pipeline([...])把数据预处理和模型训练组装成一条完整的流水线。('preprocessor', preprocessor)是流水线的 “预处理车间”，先对数据进行清洗、转换；('regressor', GradientBoostingRegressor())则是 “模型生产车间”，用梯度提升回归模型对处理好的数据进行学习和预测。这样一来，数据从流水线一端进入，就能直接在另一端产出训练好的模型，避免了中间繁琐的数据衔接问题。​

但光有流水线还不够，我们还得调试参数，让它达到最佳性能。param\_dist = {...}定义了参数搜索空间，这里列举了GradientBoostingRegressor模型的几个关键参数的取值范围，比如n\_estimators（树的数量）、learning\_rate（学习率）和max\_depth（树的最大深度），就像给模型设置了不同的 “工作模式”。​

search = RandomizedSearchCV(...)接下来就是进行搜索。RandomizedSearchCV会在我们设定的参数空间里随机抽样，通过交叉验证（cv=3）的方式，用neg\_mean\_squared\_error作为评分标准，找到能让模型误差最小的参数组合。这就好比不断调整工厂机器的参数，找到生产效率最高的设置。​

当搜索结束，best\_model = search.best\_estimator\_帮我们找到了 “最佳流水线配置”，也就是最佳模型。print(f"Best Params: {search.best\_params\_}")和print(f"Best RMSE: {np.sqrt(-search.best\_score\_):.2f}")输出了最优参数和对应的均方根误差，让我们直观了解模型的性能。​

最后，dump(best\_model, 'best\_house\_price\_model.joblib')把训练好的最佳模型保存下来，以后遇到类似任务，直接调用快速产出预测结果。​