3456

```
# -*- coding: utf-8 -*-
# 说明：这段“轻量版”代码一次性完成前面 4 个问题（Q1/Q2/Q5/Q6）
# 依赖：已有 DataFrame 变量 df，且包含下列列名（能缺少部分，会自动跳过）
# 必要列：
#   价格列： "Listing Price (USD)"
#   区域列： "Geographic Region"
#   年份列： "Year"
# 建议的数值列（用于相关性/昂贵船差异）： 尺寸/排水/帆面积/动力（缺了也不影响整体运行）

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

# （可选）如果安装了 scipy，就做一个简单的区域差异显著性检验
try:
    from scipy.stats import kruskal
    _HAS_SCIPY = True
except Exception:
    _HAS_SCIPY = False

PRICE = "Listing Price (USD)"
REG   = "Geographic Region"
YEAR  = "Year"

# —— 一些可能用到的数值列（存在才用，主要用于Q1、Q2）——
NUM_CANDIDATES = [
    "Hull length","Beam (width)","Draft","Deck area",
    "Light displacement (MLC)","Upwind sail area","Downwind sail area",
    "Mainsail area","Engine(s) power","Year"
]
num_cols = [c for c in NUM_CANDIDATES if c in df.columns]

# 清理价格（去除无效/非正）
df = df.copy()
df = df[pd.to_numeric(df[PRICE], errors="coerce") > 0]
df[PRICE] = df[PRICE].astype(float)

# ===================== Q1. 相关性Top-3（数值列两两相关） =====================
print("\n=== Q1) Highest numerical correlations — top 3 ===")
if len(num_cols) >= 2:
    corr = df[num_cols].corr(method="pearson")
    pairs = (corr.where(~np.eye(len(corr), dtype=bool))
                 .stack()
                 .rename("r")
```

```
                .reset_index()
                .rename(columns={"level_0":"feat1","level_1":"feat2"}))
    # A–B 与 B–A 视为同一对；按 |r| 排序取前三
    pairs["key"] = pairs.apply(lambda r: tuple(sorted([r["feat1"], r["feat2"]])), axis=1)
    pairs = pairs.drop_duplicates("key").assign(abs_r=lambda d: d["r"].abs()) \
                 .sort_values("abs_r", ascending=False)
    for i, row in enumerate(pairs.head(3).itertuples(index=False), 1):
        print(f"Top–{i}: {row.feat1} — {row.feat2} (r = {row.r:.3f})")
else:
    print("可用数值特征不足，无法计算。")

# ===================== Q2. 价格分布 + 昂贵船共性 =====================
print("\n=== Q2) Price distribution & traits of expensive boats ===")
prices = df[PRICE].dropna()
# 画两个最基础的直方图（原尺度 + 对数x轴），方便快速观察尾部
plt.figure(figsize=(7,3.2)); plt.hist(prices, bins=40); plt.title("Price (USD)"); plt.xlabel("Price");
plt.ylabel("Count"); plt.show()
plt.figure(figsize=(7,3.2)); plt.hist(prices, bins=40); plt.xscale("log"); plt.title("Price (USD) — log
x"); plt.xlabel("Price (log)"); plt.ylabel("Count"); plt.show()

# 定义“昂贵船”：价格Top 10%
thr = prices.quantile(0.90)
expensive = df[df[PRICE] >= thr]
others    = df[df[PRICE] <  thr]
print(f"定义昂贵船阈值(Top10%): ≈ ${thr:,.0f}；样本量：昂贵 {len(expensive)}，其他
{len(others)}")

# 数值特征均值差（昂贵–其他）→ 看哪些指标在昂贵船上显著更高
if num_cols:
    deltas = (expensive[num_cols].mean() - others[num_cols].mean()) \
             .sort_values(ascending=False).head(8)
    print("\n数值特征（昂贵均值–其他均值）Top差异：")
    for k, v in deltas.items():
        print(f"- {k}: Δmean ≈ {v:.3f}")

# 类别特征简单看两列（有则输出）：船体类型/材料/区域哪类更常见
for cat in [c for c in ["Hull type","Construction",REG] if c in df.columns]:
    s1 = expensive[cat].value_counts(normalize=True)
    s2 = others[cat].value_counts(normalize=True)
    diff = (s1 - s2).dropna().sort_values(ascending=False).head(3)
    print(f"\n类别特征更常见（昂贵 - 其他）—— {cat}:")
    for k, v in diff.items():
        print(f"- {k}: +{v*100:.1f} pp")

# ===================== Q5. 区域对价格的影响 + 一致性 =====================
```

```
print("\n=== Q5) Geographic region effect on pricing (+ consistency) ===")
if REG in df.columns:
    sub = df[[PRICE, REG]].dropna()
    # 分区域中位价与样本量
    tab = (sub.groupby(REG).agg(median=(PRICE,"median"), n=(PRICE,"size"))
                .sort_values("median", ascending=False))
    print(tab.head(10).to_string())
    # 箱线图（对数y轴）快速可视化
    plt.figure(figsize=(8,3.2))
    data = [g[PRICE].values for _, g in sub.groupby(REG)]
    labels = [str(k) for k,_ in sub.groupby(REG)]
    plt.boxplot(data, labels=labels, showfliers=False)
    plt.yscale("log"); plt.title("Price by Region (log y)"); plt.xticks(rotation=20); plt.ylabel("USD
(log)"); plt.show()
    # 简单显著性（非参数Kruskal），有就做
    if _HAS_SCIPY and tab.shape[0] >= 2:
        groups = [g[PRICE].values for _, g in sub.groupby(REG)]
        H, p = kruskal(*groups)
        print(f"Kruskal–Wallis: H={H:.3f}, p={p:.4f}  （p<0.05 视为区域分布有显著差异）")
    else:
        print("显著性检验跳过（缺少scipy或分组不足）。")
else:
    print("缺少区域列，跳过。")

# 一致性“快速检查”：仅双体/单体样本分别看区域中位价排序（有就看，不画图）
for htype in ["Catamaran","Monohull"]:
    if REG in df.columns and "Hull type" in df.columns:
        sub = df[(df["Hull type"]==htype) & df[PRICE].notna() & df[REG].notna()]
        if len(sub) >= 30:
            tab = (sub.groupby(REG)[PRICE].median().sort_values(ascending=False).head(5))
            print(f"\n{htype} 子样本的区域中位价Top5：")
            for k, v in tab.items():
                print(f"- {k}: ${v:,.0f}")

# ====================== Q6. 按年份的均价/中位价 + 趋势 ======================
print("\n=== Q6) Average & median price by year built (trend) ===")
if YEAR in df.columns:
    sub = df[[YEAR, PRICE]].dropna()
    # 合理的年份过滤（避免 0/异常年）
    sub = sub[(sub[YEAR] > 1900) & (sub[YEAR] < 2100)]
    byy = (sub.groupby(YEAR).agg(avg=(PRICE,"mean"), med=(PRICE,"median"),
n=(PRICE,"size"))
                .sort_index())
    print(byy.tail(10).to_string())
    # 画均值/中位数曲线（各一张）
```

```
    plt.figure(figsize=(7,3.0)); plt.plot(byy.index, byy["avg"], marker="o"); plt.title("Average price by year"); plt.xlabel("Year"); plt.ylabel("USD"); plt.show()
    plt.figure(figsize=(7,3.0)); plt.plot(byy.index, byy["med"], marker="o"); plt.title("Median price by year"); plt.xlabel("Year"); plt.ylabel("USD"); plt.show()
    # 简易“平滑中位数”看趋势
    med_roll = byy["med"].rolling(3, center=True, min_periods=1).median()
    plt.figure(figsize=(7,3.0));
    plt.plot(byy.index, byy["med"], alpha=0.4)
    plt.plot(byy.index, med_roll, linewidth=2)
    plt.title("Median price (3-year rolling)"); plt.xlabel("Year"); plt.ylabel("USD"); plt.show()
else:
    print(“缺少年份列，跳过。")
```

Part2

```
# -*- coding: utf-8 -*-
# 目的：用最“轻量”的代码完成题目 Part 2（a~e）
# 思路非常简单：选一批可泛化的特征 → 训练一个常见模型 → 打分 → 看特征重要性 → 预测无价样本并算均/中位数
# 注释尽量写中文，代码不追求严谨（比如不过度调参、不过度清洗）

import numpy as np
import pandas as pd
from pathlib import Path

# 模型与预处理（最常用的一套即可）
from sklearn.model_selection import train_test_split
```

```
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.ensemble import RandomForestRegressor  # 简单、稳、好用
from sklearn.metrics import r2_score, mean_absolute_error
from sklearn.inspection import permutation_importance

# ====== a) 我们要用的“可泛化特征”（不用品牌/型号这类强识别字段）======
TARGET = "Listing Price (USD)"
# 数值：尺寸/排水/动力/帆面积/年份 —— 对未见船也有解释力
NUM_FEATS = [
    "Year","Hull length","Beam (width)","Draft","Deck area",
    "Light displacement (MLC)","Upwind sail area","Downwind sail area",
    "Mainsail area","Engine(s) power"
]
# 类别：地区/船体类型/材料/索具 等 —— 对价格也会有系统性影响
CAT_FEATS = [
    "Geographic Region","Hull type","Construction","Rigging type","Category"
]

# 为了能直接跑，这里做个“存在即用”过滤（防止你数据里缺少某些列）
NUM_FEATS = [c for c in NUM_FEATS if c in df.columns]
CAT_FEATS = [c for c in CAT_FEATS if c in df.columns]

# 只保留参与建模的列
data = df[[TARGET] + NUM_FEATS + CAT_FEATS].copy()
data = data[pd.to_numeric(data[TARGET], errors="coerce") > 0]  # 过滤掉目标的无效值

# ====== b) 一个最常见的端到端管道：缺失值填补 + 类别独热 + 随机森林 ======
num_pipe = Pipeline([
    ("imputer", SimpleImputer(strategy="median")),  # 数值缺失 → 用中位数补
])
cat_pipe = Pipeline([
    ("imputer", SimpleImputer(strategy="most_frequent")),  # 类别缺失 → 用众数补
    ("onehot", OneHotEncoder(handle_unknown="ignore"))     # 未见新类别也能预测
])

prep = ColumnTransformer([
    ("num", num_pipe, NUM_FEATS),
    ("cat", cat_pipe, CAT_FEATS),
])

# 选择 RandomForest 的原因（一句话）：少调参也能给出“还不错”的基线 + 自带非线性能力
model = RandomForestRegressor(
```

```
    n_estimators=400, random_state=42, n_jobs=-1
)

pipe = Pipeline([
    ("prep", prep),
    ("model", model),
])

# ====== c) 切分/训练/评估（只做一次留出法，够用）======
X = data[NUM_FEATS + CAT_FEATS]
y = data[TARGET].astype(float)

X_tr, X_te, y_tr, y_te = train_test_split(X, y, test_size=0.25, random_state=42)
pipe.fit(X_tr, y_tr)

pred = pipe.predict(X_te)
print("\n[Evaluation]")
print(f"R2  : {r2_score(y_te, pred):.3f}")                    # 解释度
print(f"MAE : ${mean_absolute_error(y_te, pred):,.0f}")       # 绝对误差（直观看价格误差有多大）

# ====== d) 特征重要性（用置换重要性，更直白；不严谨但好理解）======
# 解释：把某个特征随机打乱，观察模型分数下降多少 → 越下降说明越“重要”
r = permutation_importance(pipe, X_te, y_te, scoring="r2", n_repeats=8, random_state=42)
feat_names = pipe.named_steps["prep"].get_feature_names_out()  # 展开后的所有特征名（含one-hot）
imp = (pd.DataFrame({"feature": feat_names,
                     "imp_mean": r.importances_mean})
       .sort_values("imp_mean", ascending=False))
print("\n[Permutation importance — top 15]")
print(imp.head(15).to_string(index=False))

# （可选）把独热后的类别重要性“汇总回字段级别”，看哪个字段整体更重要
field_imp = (imp.assign(field=lambda d: d["feature"].str.split("=").str[0])
               .groupby("field", as_index=False)["imp_mean"].sum()
               .sort_values("imp_mean", ascending=False))
print("\n[Field-level importance — top 10]")
print(field_imp.head(10).to_string(index=False))

# ====== e) 读取“无价格的10条样本”并预测；输出均值与中位数 ======
PRED_PATH = Path("BoatData_NoPrice_V1.csv")  # 改成你的文件名/路径
if PRED_PATH.exists():
    new_df = pd.read_csv(PRED_PATH)
    new_X = new_df.reindex(columns=NUM_FEATS + CAT_FEATS)  # 只取我们训练用的列
    # 最后用全部数据重新训练一下再预测（提高稳定性）
```

```
    pipe.fit(X, y)
    new_pred = pipe.predict(new_X)
    new_df["Predicted Price (USD)"] = new_pred

    print("\n[Predict 10 boats — summary]")
    print(f"Average : ${np.mean(new_pred):,.0f}")
    print(f"Median  : ${np.median(new_pred):,.0f}")

    # 保存一个带预测的文件（交作业可用）
    out_file = PRED_PATH.with_name(PRED_PATH.stem + "_with_predictions.csv")
    new_df.to_csv(out_file, index=False)
    print(f"Saved: {out_file}")
else:
    print("\n[Notice] Cannot find 'BoatData_NoPrice_V1.csv'. Put the file next to this
notebook/script.")
```

特征分析

```
# -*- coding: utf-8 -*-
# 目的：对“挂牌价（Listing Price (USD)）”做特征分析（feature analysis）
# 产出：相关性/互信息、模型重要性（permutation importance）、部分依赖（PDP 可选）
# 说明：注释全中文；方法名/参数保持英文；你可按需删改列名和模型。

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

from sklearn.model_selection import train_test_split, KFold, cross_val_score
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.metrics import r2_score, mean_absolute_error
from sklearn.feature_selection import mutual_info_regression
from sklearn.inspection import permutation_importance, PartialDependenceDisplay
from sklearn.ensemble import HistGradientBoostingRegressor
```

```
# ========= 0) 基础设定 =========
# 目标列（价格）；若你的列名不同，改这里
TARGET_COL = "Listing Price (USD)"

# 可能的“标识/明显泄露列”（不用于建模）
ID_LIKE_COLS = {
    "Make","Model","Variant","Sailboat designer","Last built hull",
    "Country","Region","State","Geographic Region","Category"
}
# ↑ 注意：其中一些“类别信息”可以纳入特征，如果你希望将“地区”等作为解释变量，
#   就不要把它们加入 ID_LIKE_COLS，或在后面将其从过滤集中移出。

# ========= 1) 数据准备与列类型自动识别 =========
assert TARGET_COL in df.columns, f"找不到目标列：{TARGET_COL}"
data = df.copy()

# 去除明显无效或非正的价格（回归目标通常需 >0）
data = data[pd.to_numeric(data[TARGET_COL], errors="coerce") > 0].copy()
data[TARGET_COL] = data[TARGET_COL].astype(float)

# 自动识别数值列/类别列
num_cols = data.select_dtypes(include=["number"]).columns.tolist()
cat_cols = data.select_dtypes(include=["object","category"]).columns.tolist()

# 目标列不算特征
num_cols = [c for c in num_cols if c != TARGET_COL]

# 可选择剔除 ID 类列（如不想剔除“Geographic Region”，从集合里删掉即可）
drop_cols = []
for c in list(ID_LIKE_COLS):
    if c in cat_cols:
        # 你可以根据分析目的选择是否保留。例如：希望评估“地区”对价格的影响 → 不剔除
        # 这里演示：把“Geographic Region”保留下来，其余去掉
        if c != "Geographic Region":
            drop_cols.append(c)

cat_cols = [c for c in cat_cols if c not in drop_cols]

# 若全部是数值或全部是类别也没关系；ColumnTransformer 会自动跳过空子管道
print(f"数值特征数：{len(num_cols)}，类别特征数：{len(cat_cols)}")

# ========= 2) 一些“单变量”层面的特征评分 =========
# 2.1 与目标的“相关性”（数值特征）：Pearson / Spearman
#     含义：线性相关（Pearson）和秩相关（Spearman），快速粗筛
def corr_with_target(df, y, cols):
```

```
    out = []
    for c in cols:
        s = pd.to_numeric(df[c], errors="coerce")
        mask = s.notna() & y.notna()
        if mask.sum() < 20:  # 防止样本过少导致不稳定
            continue
        pear = s[mask].corr(y[mask], method="pearson")
        spear = s[mask].corr(y[mask], method="spearman")
        out.append((c, pear, spear, mask.sum()))
    return (pd.DataFrame(out, columns=["feature","pearson","spearman","n"])
              .sort_values("pearson", key=lambda x: x.abs(), ascending=False))

y = data[TARGET_COL]
corr_table = corr_with_target(data, y, num_cols)
print("\n[单变量评分] 与价格的相关性（数值特征）TOP 10：")
print(corr_table.head(10).to_string(index=False))

# 2.2“互信息”（类别/混合特征）：Mutual Information（对非线性/任意单调关系更敏感）
#     做法：对每个候选特征单独计算与 y 的 MI；类别将先做简单编码（频数目标化）
def simple_encode_for_mi(s):
    # 将类别列转换为整数编码（频数高的值更靠前）；数值列原样返回
    if s.dtype == "O" or str(s.dtype).startswith("category"):
        codes = s.astype("category").cat.codes
        return codes.replace(-1, np.nan)  # 缺失为 -1 → NaN
    return pd.to_numeric(s, errors="coerce")

mi_candidates = num_cols + cat_cols
mi_frame = []
for c in mi_candidates:
    x = simple_encode_for_mi(data[c])
    mask = x.notna() & y.notna()
    if mask.sum() < 50:
        continue
    # 互信息对尺度不敏感；离散/连续混合也可用
    # random_state 固定以提高复现性；n_neighbors 可调
    mi = mutual_info_regression(
        X = x[mask].to_frame(),
        y = y[mask].values,
        discrete_features = False,
        random_state = 42
    )[0]
    mi_frame.append((c, mi, mask.sum()))
mi_table = (pd.DataFrame(mi_frame, columns=["feature","MI","n"])
              .sort_values("MI", ascending=False))
print("\n[单变量评分] 互信息（数值+类别混合）TOP 10：")
```

```
print(mi_table.head(10).to_string(index=False))

# ========= 3) 建模与“模型解释型”特征重要性 =========
# 方法选择：HistGradientBoostingRegressor（树提升，鲁棒、速度快、可处理非线性）
# 预处理：
#   – 数值：缺失填补+标准化（提升学习稳定性）
#   – 类别：缺失填补+OneHotEncoder（handle_unknown='ignore' 以防新类别）
pre_num = Pipeline(steps=[
    ("imputer", SimpleImputer(strategy="median")),
    ("scaler", StandardScaler())
])

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

ct = ColumnTransformer(
    transformers=[
        ("num", pre_num, num_cols),
        ("cat", pre_cat, cat_cols)
    ],
    remainder="drop",
    verbose_feature_names_out=False
)

model = HistGradientBoostingRegressor(
    learning_rate=0.07,
    max_depth=None,            # 自动生长；也可设如 6~10
    max_iter=500,              # 迭代轮数
    l2_regularization=0.0,
    early_stopping=True,
    random_state=42
)

pipe = Pipeline(steps=[
    ("prep", ct),
    ("model", model)
])

# 切分训练/验证集（留出法评估+后续做 permutation importance）
X = data[num_cols + cat_cols]
y = data[TARGET_COL]

X_train, X_valid, y_train, y_valid = train_test_split(
```

```
    X, y, test_size=0.25, random_state=42
)

# 训练
pipe.fit(X_train, y_train)

# 验证集表现
pred = pipe.predict(X_valid)
print("\n[模型评估] 验证集性能：")
print(f"R2  = {r2_score(y_valid, pred):.3f}")
print(f"MAE = {mean_absolute_error(y_valid, pred):,.0f}")

# ========= 4) Permutation Importance（置换重要性）=========
# 含义：打乱某一特征，观察验证集得分下降幅度 → 越大说明该特征越重要（模型不可替代性）
r = permutation_importance(
    estimator=pipe,
    X=X_valid,
    y=y_valid,
    scoring="r2",
    n_repeats=10,
    random_state=42
)

# 从 ColumnTransformer 导出展开后的特征名（含 one-hot 后的稀疏名）
#   说明：get_feature_names_out 可输出数值列原名 + 类别列的独热名（如 Region=Europe）
feature_names = pipe.named_steps["prep"].get_feature_names_out()
imp_df = (pd.DataFrame({
            "feature": feature_names,
            "importance_mean": r.importances_mean,
            "importance_std": r.importances_std
          })
          .sort_values("importance_mean", ascending=False))

print("\n[Permutation Importance] TOP 20：")
print(imp_df.head(20).to_string(index=False))

# ========= 5)（可选）部分依赖图 PDP =========
# 含义：在“其他特征取样本分布”的条件下，考察“单个特征变化”对预测的平均影响趋势
# 适合连续数值特征（如 Hull length / Beam / Engine(s) power）；类别特征请看 one-hot 名称
top_numeric_for_pdp = [c for c in ["Hull length","Beam (width)","Deck area","Light
displacement (MLC)","Engine(s) power"]
                       if c in num_cols][:2] # 例示最多画2个，防止输出过多

if top_numeric_for_pdp:
    plt.figure()
```

```
    PartialDependenceDisplay.from_estimator(
        estimator=pipe,
        X=X_valid,
        features=top_numeric_for_pdp,
        kind="average"
    )
    plt.suptitle("Partial Dependence (selected numeric features)")
    plt.tight_layout()
    plt.show()

# ========= 6) 小结打印（英文正文，便于粘贴；你可改为中文正文）=========
print("\n" + "="*78)
print("FEATURE ANALYSIS — SHORT SUMMARY (fill with your actual top names)")
print("="*78)

# 6.1 单变量：相关性 + 互信息（展示前5）
print("\n[Univariate scores]")
if not corr_table.empty:
    print("- Pearson/Spearman top correlations with price:")
    for _, row in corr_table.head(5).iterrows():
        print(f"  • {row['feature']}: pearson={row['pearson']:.3f}, spearman={row['spearman']:.3f}
(n={int(row['n'])})")
if not mi_table.empty:
    print("- Mutual information (mixed features):")
    for _, row in mi_table.head(5).iterrows():
        print(f"  • {row['feature']}: MI={row['MI']:.4f} (n={int(row['n'])})")

# 6.2 模型置换重要性（展示前10）
print("\n[Model-based importance — permutation]")
for _, row in imp_df.head(10).iterrows():
    print(f"  • {row['feature']}: meanΔR2={row['importance_mean']:.4f} (±
{row['importance_std']:.4f})")

print("\nNotes:")
print("- Pearson/Spearman capture linear/monotonic links; MI captures generic
dependence.")
print("- Permutation importance reflects model reliance; higher mean drop in R2 ⇒ more
important.")
print("- For categorical features, check one-hot names (e.g., Geographic Region=Europe).")
print("- Consider log-transforming price or using robust metrics if heavy tails are present.”)
```

3456加长版

```
# -*- coding: utf-8 -*-
# 说明：把下列代码粘贴到你的 notebook / .py 文件即可运行。
# 要求：DataFrame 变量名为 df；列名需与注释一致（可按需改名）。
# 目标：一次性计算 Q1/Q2/Q5/Q6 的核心结果，并打印为“框内英文摘要”。
# 约定：注释全部中文；打印输出全部英文；你可自行分块取用。

import numpy as np
import pandas as pd

# 可选的统计检验（地理区域价格差异）；若未安装 scipy，会自动降级跳过检验
try:
    from scipy import stats
    _HAS_SCIPY = True
except Exception:
    _HAS_SCIPY = False

def _safe_cols_exists(df, cols):
    """检查列是否存在；不存在则返回已存在的交集并给出提示用以容错。"""
    ok = [c for c in cols if c in df.columns]
    return ok

def _top3_correlations(num_df):
    """计算数值列两两皮尔逊相关，返回绝对值最高的前三对（去重）。"""
    if num_df.shape[1] < 2:
        return []
    corr = num_df.corr(method="pearson")
    # 展平为 (feat1, feat2, corr)
    s = (
        corr.where(~np.eye(len(corr), dtype=bool))
            .stack()
            .rename("corr")
            .reset_index()
            .rename(columns={"level_0":"feat1","level_1":"feat2"})
    )
    # 去重（A-B 与 B-A 视为同一对）
    s["pair"] = s.apply(lambda r: tuple(sorted([r["feat1"], r["feat2"]])), axis=1)
    s = s.drop_duplicates("pair", keep="first")
    # 绝对值排序取前三
```

```
    s = s.assign(abs_corr=lambda x: x["corr"].abs()).sort_values("abs_corr", ascending=False)
    return s.head(3)[["feat1","feat2","corr"]].to_records(index=False)

def _expensive_traits(df, price_col="Listing Price (USD)", q=0.90):
    """以价格分位数阈值定义昂贵样本，返回阈值、数值特征均值差Top3、类别特征占比差Top3。
"""
    if price_col not in df.columns:
        return np.nan, [], []
    sub = df.dropna(subset=[price_col]).copy()
    if sub.empty:
        return np.nan, [], []
    thr = sub[price_col].quantile(q)
    expensive = sub[sub[price_col] >= thr]
    others    = sub[sub[price_col] <  thr]
    # 数值列差异（昂贵均值 - 其他均值）
    num_cols = sub.select_dtypes(include=["number"]).columns.tolist()
    num_cols = [c for c in num_cols if c != price_col]
    num_deltas = []
    if num_cols and not expensive.empty and not others.empty:
        exp_mean = expensive[num_cols].mean()
        oth_mean = others[num_cols].mean()
        dd = (exp_mean - oth_mean).sort_values(ascending=False)
        num_deltas = list(dd.head(3).items())
    # 类别列占比差（昂贵占比 - 其他占比）
    cat_cols = sub.select_dtypes(include=["object","category"]).columns.tolist()
    cat_diffs = []
    for col in cat_cols:
        s1 = expensive[col].value_counts(normalize=True)
        s2 = others[col].value_counts(normalize=True)
        # 对齐后求差
        dd = (s1 - s2).dropna().sort_values(ascending=False)
        # 选择该列中最大的正差一条作为代表
        if not dd.empty:
            cat_diffs.append((col, dd.index[0], float(dd.iloc[0])))
    # 取前三个“最显著提升的类别-取值”
    cat_diffs = sorted(cat_diffs, key=lambda x: abs(x[2]), reverse=True)[:3]
    return float(thr), num_deltas, cat_diffs

def _region_effect(df, price_col="Listing Price (USD)", region_col="Geographic Region"):
    """区域价格差异：返回中位价排名、样本量、以及Kruskal/ANOVA结果（如可用）。"""
    out = {
        "rank_table": [],
        "test_name": None,
        "stat": None,
        "pvalue": None
```

```
    }
    if not {price_col, region_col}.issubset(df.columns):
        return out
    sub = df[[price_col, region_col]].dropna()
    if sub.empty:
        return out
    # y取对数更稳健
    sub = sub.assign(log_price=np.log(sub[price_col].astype(float)+1e-9))
    # 区域中位价排名（名义价）
    rk = (sub.groupby(region_col)
            .agg(median_price=(price_col,"median"), n=(price_col,"size"))
            .sort_values("median_price", ascending=False))
    out["rank_table"] = list(rk.reset_index().itertuples(index=False, name=None))
    # 统计检验
    if _HAS_SCIPY:
        groups = [g["log_price"].values for _, g in sub.groupby(region_col)]
        try:
            H, p = stats.kruskal(*groups)
            out.update({"test_name":"Kruskal–Wallis (on log price)", "stat":float(H),
"pvalue":float(p)})
        except Exception:
            pass
    return out

def _year_trend(df, price_col="Listing Price (USD)", year_col="Year"):
    """按年份的均价/中位价与滚动中位数（不画图，直接给关键数字）。"""
    out = {
        "table": [],
        "first_year": None,
        "last_year": None,
        "avg_first": None,
        "avg_last": None,
        "med_first": None,
        "med_last": None
    }
    if not {price_col, year_col}.issubset(df.columns):
        return out
    sub = df[[price_col, year_col]].dropna()
    if sub.empty:
        return out
    # 合理年份过滤
    sub = sub[(sub[year_col].astype(float) > 1900) & (sub[year_col].astype(float) <
2100)].copy()
    if sub.empty:
        return out
```

```
    g = (sub.groupby(year_col)
            .agg(avg_price=(price_col,"mean"),
                 med_price=(price_col,"median"),
                 n=(price_col,"size"))
            .sort_index())
    out["table"] = list(g.reset_index().itertuples(index=False, name=None))
    # 取首尾年份与均值/中位数
    years = g.index.to_list()
    out["first_year"], out["last_year"] = int(years[0]), int(years[-1])
    out["avg_first"], out["avg_last"]   = float(g["avg_price"].iloc[0]), float(g["avg_price"].iloc[-1])
    out["med_first"], out["med_last"]   = float(g["med_price"].iloc[0]),
float(g["med_price"].iloc[-1])
    return out

def make_report_box(df):
    """主函数：汇总四个问题并打印为英文框。"""
    # —— Q1/Q2 依赖的列推断 ——
    price_col = "Listing Price (USD)"
    region_col = "Geographic Region"
    year_col   = "Year"

    # 数值列（用于相关性；排除明显ID/重复性弱的列可自行补充）
    num_df = df.select_dtypes(include=["number"]).copy()
    # 计算Top-3相关
    top3 = _top3_correlations(num_df)

    # 昂贵样本共性（Top10%）
    thr, num_deltas, cat_diffs = _expensive_traits(df, price_col=price_col, q=0.90)

    # 区域效应
    reg_res = _region_effect(df, price_col=price_col, region_col=region_col)

    # 年份趋势
    year_res = _year_trend(df, price_col=price_col, year_col=year_col)

    # —— 组装英文输出（你可按需分块复制） ——
    lines = []
    push = lines.append

    # 顶部框线
    push("="*86)
    push("COMPACT ANSWERS BLOCK (Q1, Q2, Q5, Q6) — FILL WITH YOUR DATA AS
NEEDED")
    push("="*86)
```

```
    # Q1: Top-3 correlations
    push("\n[1) Highest numerical correlations — top 3]")
    if len(top3) == 0:
        push("- Not enough numerical columns to compute pairwise correlations.")
    else:
        for i, (a, b, r) in enumerate(top3, start=1):
            push(f"- Top-{i}: {a} — {b} (r = {r:.3f}).")

    # Q2: Price distribution & expensive traits
    push("\n[2) Listing-price distribution & traits of expensive boats]")
    push("- Shape: right-skewed with a long upper tail; a log-scaled x-axis typically clarifies
the tail.")
    if not np.isnan(thr):
        push(f"- Definition of \"expensive\": top 10% by price (threshold ≈ ${thr:,.0f}).")
    else:
        push("- Definition of \"expensive\": top 10% by price (threshold not available).")
    # 数值特征差异
    if num_deltas:
        push("- Numerical traits (expensive minus others; top differences):")
        for name, delta in num_deltas:
            push(f"  • {name}: Δmean ≈ {delta:.3f}")
    else:
        push("- Numerical traits: insufficient data to compute mean differences.")
    # 类别特征差异
    if cat_diffs:
        push("- Categorical skews (more frequent categories among expensive boats):")
        for col, level, diff in cat_diffs:
            pct = diff * 100
            sign = "+" if diff >= 0 else "-"
            push(f"  • {col} = {level}: Δshare ≈ {sign}{abs(pct):.1f} pp")
    else:
        push("- Categorical skews: no clear category uplift detected or insufficient data.")

    # Q5: Geographic region effect + consistency
    push("\n[5) Geographic region effect on pricing + consistency]")
    if reg_res["rank_table"]:
        push("- Median-price ranking by region (name, median, n):")
        for region, med, n in reg_res["rank_table"][:6]:
            try:
                med_fmt = f"${float(med):,.0f}"
            except Exception:
                med_fmt = str(med)
            push(f"  • {region}: {med_fmt} (n={int(n)})")
    else:
        push("- Region ranking: not available (missing columns or insufficient data).")
```

```
    if reg_res["test_name"] is not None:
        push(f"- Significance: {reg_res['test_name']}: stat = {reg_res['stat']:.3f}, p =
{reg_res['pvalue']:.4f}.")
        push("  Interpretation: p<0.05 indicates statistically significant cross-region
differences.")
    else:
        push("- Significance: skipped (scipy not available). Consider Kruskal–Wallis/ANOVA on
log price.")
    push("- Consistency checklist: verify ordering within hull-type slices, recent-year cohorts,
and size bands;")
    push("  if rankings persist, the regional premium/discount is likely stable rather than
composition-driven.")

    # Q6: Year built — average & median trend
    push("\n[6) Average & median price by year built — observed trends]")
    if year_res["table"]:
        fy, ly = year_res["first_year"], year_res["last_year"]
        push(f"- Coverage: {fy} → {ly}.")
        push(f"- Average price: {fy} ≈ ${year_res['avg_first']:,.0f}; {ly} ≈ $
{year_res['avg_last']:,.0f}.")
        push(f"- Median price:  {fy} ≈ ${year_res['med_first']:,.0f}; {ly} ≈ $
{year_res['med_last']:,.0f}.")
        push("- Read with caution: the mean–median gap indicates tail thickness; consider
median/log-median for robustness.")
        push("- Optional: control for size/region/type with a log-price model to isolate a pure
year (vintage) effect.")
    else:
        push("- Year trend: not available (missing columns or insufficient data).")

    # 底部框线
    push("\n" + "="*86)

    # 打印整体框
    print("\n".join(lines))

# === 调用入口（直接运行） ===
# 注意：确保 df 已在当前环境中，且包含至少以下列：
#   - "Listing Price (USD)"
#   - "Geographic Region"
#   - "Year"
# 其他可选用于解释的数值列：Hull length / Beam (width) / Draft / Deck area / Light
displacement / sail areas / Engine(s) power 等
make_report_box(df)
```

part2加长版

```
# -*- coding: utf-8 -*-
# 任务：为“帆船定价”建立可泛化的模型，并输出特征重要性与对无价样本的预测
# 说明：注释全中文；打印/列名英文保持一致；把 df 换成你已加载的训练DataFrame 变量
# 假设：训练集 df 含目标列 "Listing Price (USD)"；预测集 CSV 文件路径见 PRED_PATH

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

from pathlib import Path
from sklearn.model_selection import train_test_split, KFold, cross_val_score
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import OneHotEncoder, StandardScaler, FunctionTransformer
from sklearn.impute import SimpleImputer
from sklearn.metrics import r2_score, mean_absolute_error
from sklearn.inspection import permutation_importance
from sklearn.ensemble import HistGradientBoostingRegressor

# =============== A) 特征选择（尽量可泛化到未见的品牌/型号）================
# 逻辑：
# 1) 不使用 Make/Model/Variant/Designer 这类“识别性”强、对未见样本泛化差的列；
# 2) 保留结构与性能可解释的列（尺寸/排水量/帆面积/动力/年份/区域/船体类型/材料/索具）；
# 3) 对价格做 log1p 变换（右偏长尾 → 稳定方差与残差）。
TARGET = "Listing Price (USD)"
assert TARGET in df.columns, "训练集缺少目标列 Listing Price (USD)"

# 可能存在的候选列（按你 df.info() 的字段整理；缺哪个会自动忽略）
NUM_CANDIDATES = [
    "Year","Hull length","Beam (width)","Draft","Deck area",
    "Light displacement (MLC)","Upwind sail area","Downwind sail area",
    "Mainsail area","Engine(s) power"
]
CAT_CANDIDATES = [
    "Geographic Region","Country/Region/State","Hull type",
    "Construction","Rigging type","Category","Appendages"
]
```

```
# 实际存在的列（容错）
num_cols = [c for c in NUM_CANDIDATES if c in df.columns]
cat_cols = [c for c in CAT_CANDIDATES if c in df.columns]

# =============== 预处理：数值/类别 各自管道 ==================
# 数值：中位数填补 + 标准化（对树模型不是必须，但可稳定训练）
num_pipe = Pipeline([
    ("imputer", SimpleImputer(strategy="median")),
    ("scaler", StandardScaler())
])

# 类别：众数填补 + OneHot（忽略未见新类别，保证预测集能跑）
cat_pipe = Pipeline([
    ("imputer", SimpleImputer(strategy="most_frequent")),
    ("onehot", OneHotEncoder(handle_unknown="ignore", sparse=True))
])

# 组合列转换器
prep = ColumnTransformer(
    transformers=[
        ("num", num_pipe, num_cols),
        ("cat", cat_pipe, cat_cols)
    ],
    remainder="drop",
    verbose_feature_names_out=False
)

# =============== B) 模型选择 ==================
# 选择 HistGradientBoostingRegressor（基于树的梯度提升）：
# 理由：非线性表达力强、对特征缩放不敏感、处理缺失表现好、推理快、鲁棒性高。
gbr = HistGradientBoostingRegressor(
    learning_rate=0.07,
    max_iter=600,
    max_depth=None,          # 由算法自适应
    early_stopping=True,
    random_state=42
)

# 目标变换：log1p（稳定长尾）；预测后再 expm1 还原
def to_log1p(y):     # y -> log1p(y)
    return np.log1p(y)

def from_log1p(yh): # 还原到原价空间
    return np.expm1(yh)
```

```
# 包装一个 FunctionTransformer 只用于评价时反变换；管道里直接拟合 log1p(y)
# （注：sklearn 的 TransformedTargetRegressor 也可，但此处直接手动处理更透明）
pipe = Pipeline([
    ("prep", prep),
    ("model", gbr)
])

# =============== C) 训练/验证与评估 ==================
# 留出验证集 + K 折交叉验证（在 log 空间评估R2；在原价空间评估MAE更直观）
data = df[[TARGET] + num_cols + cat_cols].dropna(subset=[TARGET]).copy()
y = data[TARGET].astype(float)
X = data.drop(columns=[TARGET])

X_tr, X_va, y_tr, y_va = train_test_split(
    X, y, test_size=0.25, random_state=42
)

# 拟合时用 log1p(y)
pipe.fit(X_tr, to_log1p(y_tr))

# 验证集预测（先得到log，再还原到原价）
yhat_va_log = pipe.predict(X_va)
yhat_va = from_log1p(yhat_va_log)

# 评估：R2（在原价空间偏乐观，仍展示），MAE（更稳健），以及 log-R2（反映相对误差）
r2_lin = r2_score(y_va, yhat_va)
mae = mean_absolute_error(y_va, yhat_va)
r2_log = r2_score(to_log1p(y_va), yhat_va_log)

print("\n[Validation metrics]")
print(f"R2 (linear price)     : {r2_lin:.3f}")
print(f"MAE (USD)             : {mae:,.0f}")
print(f"R2 (log-price space)  : {r2_log:.3f}")

# 简单 K 折（log 空间R2更稳定）
cv = KFold(n_splits=5, shuffle=True, random_state=42)
cv_scores = cross_val_score(pipe, X, to_log1p(y), cv=cv, scoring="r2")
print(f"CV R2 (log space) mean±std : {cv_scores.mean():.3f} ± {cv_scores.std():.3f}")

# =============== D) 特征重要性（Permutation Importance）==================
# 解释：打乱某一特征的列，观察验证集分数下降幅度（在log空间R2）
perm = permutation_importance(
    estimator=pipe,
    X=X_va, y=to_log1p(y_va),
```

```
    n_repeats=10, scoring="r2", random_state=42
)

# 展开独热后的特征名（便于定位具体类别贡献）
feat_names = pipe.named_steps["prep"].get_feature_names_out()
imp_df = (pd.DataFrame({
            "feature": feat_names,
            "imp_mean": perm.importances_mean,
            "imp_std":  perm.importances_std
        })
        .sort_values("imp_mean", ascending=False))

print("\n[Permutation importance — top 20 (log-R2 drop)]")
print(imp_df.head(20).to_string(index=False))

# =============== E) 对“无价格样本”进行预测 ==================
# 修改为你的实际文件路径；要求列结构与训练时使用的特征一致（缺失会自动填补/新类别自动忽略）
PRED_PATH = Path("BoatData_NoPrice_V1.csv") # TODO: 替换为真实路径
if PRED_PATH.exists():
    new_df = pd.read_csv(PRED_PATH)
    # 仅取建模所需列；缺失将由预处理管道填补
    new_X = new_df.reindex(columns=num_cols + cat_cols)

    # 用全量训练（更稳健）：把训练+验证合并重新拟合一次
    pipe.fit(X, to_log1p(y))
    new_pred_log = pipe.predict(new_X)
    new_pred = from_log1p(new_pred_log)

    # 输出基本统计
    avg_price = float(np.mean(new_pred))
    med_price = float(np.median(new_pred))
    print("\n[Predictions for BoatData_NoPrice_V1.csv]")
    print(f"Average predicted price : ${avg_price:,.0f}")
    print(f"Median predicted price  : ${med_price:,.0f}")

    # 保存带预测的CSV
    out_path = PRED_PATH.with_name(PRED_PATH.stem + "_with_predictions.csv")
    new_out = new_df.copy()
    new_out["Predicted Price (USD)"] = new_pred
    new_out.to_csv(out_path, index=False)
    print(f"Saved predictions to: {out_path}")
else:
    print("\n[Notice] Prediction file not found. Set PRED_PATH to your
'BoatData_NoPrice_V1.csv'.")
```

```
# ================ 可选：把“聚合级别的重要性”进一步汇总 ==================
# 对于独热后的类别特征，可以将同一原始字段的所有 one-hot 列重要性求和，得到字段级重要性
field_imp = (
    imp_df.assign(field=lambda d: d["feature"].str.split("=").str[0]) # 例如 Geographic
Region=Europe → Geographic Region
        .groupby("field", as_index=False)["imp_mean"].sum()
        .sort_values("imp_mean", ascending=False)
)
print("\n[Field-level importance (sum over one-hot levels) — top 10]")
print(field_imp.head(10).to_string(index=False))
```