Golden Answer - Quality Control Note - Code

Python

```python
# ========================================
# Integrated Churn & Revenue Analysis Script
# ========================================
# Requirements: pandas, numpy, matplotlib, scipy, scikit-learn
# Input files: Product.csv, Region.csv, Sales.csv (in same directory)
# Output files: integrated_dataset.csv, churn_dataset.csv, roc_curve.png,
#               calibration_deciles.png, churn_profitability_overlay.png,
analysis_summary.json
# ========================================

import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy import stats
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import StratifiedKFold, cross_val_predict
from sklearn.metrics import roc_auc_score, roc_curve, confusion_matrix,
precision_recall_fscore_support, brier_score_loss
import json

# ---------- Load CSVs Robustly ----------
def read_csv_auto(path):
    try:
        return pd.read_csv(path, sep=None, engine="python")
    except Exception:
        for sep in [",",";", "|", "\t"]:
            try:
                return pd.read_csv(path, sep=sep)
            except Exception:
                continue
        return pd.read_csv(path, encoding="latin-1")

df_prod = read_csv_auto("Product.csv")
df_reg = read_csv_auto("Region.csv")
```

```
df_sales = read_csv_auto("Sales.csv")

# ---------- Merge ----------
def infer_join_key(left, right, candidates):
    for c in candidates:
        if c in left.columns and c in right.columns:
            return c, c
    commons = [c for c in left.columns if c in right.columns]
    return (commons[0], commons[0]) if commons else (None, None)

lk, rk = infer_join_key(df_sales, df_prod, ["ProductKey", "ProductID",
"productkey", "productid"])
merged = df_sales.merge(df_prod, left_on=lk, right_on=rk, how="left") if lk
else df_sales.copy()

lk2, rk2 = infer_join_key(merged, df_reg, ["SalesTerritoryKey", "RegionID",
"salesterritorykey", "regionid"])
if lk2:
    merged = merged.merge(df_reg, left_on=lk2, right_on=rk2, how="left")

# ---------- Data Cleaning ----------
def coerce_numeric_auto(df):
    for c in df.columns:
        if df[c].dtype == "O" and any(k in c.lower() for k in ["price",
"revenue", "sales", "amount", "total", "charge"]):
            df[c] = pd.to_numeric(df[c].astype(str).str.replace(r"[,$% ]", "",
regex=True), errors="coerce")
    return df

merged = coerce_numeric_auto(merged)
for c in merged.columns:
    if merged[c].dtype.kind in "biufc":
        merged[c] = merged[c].fillna(merged[c].median())
    else:
        merged[c] = merged[c].fillna(merged[c].mode().iloc[0] if not
merged[c].mode().empty else "Unknown")

# ---------- Feature Engineering ----------
date_col = next((c for c in merged.columns if "date" in c.lower()), None)
if date_col: merged[date_col] = pd.to_datetime(merged[date_col],
errors="coerce")

cust_col = next((c for c in merged.columns if "customer" in c.lower() or
"reseller" in c.lower()), None)
```

```
if date_col and cust_col:
    tmp = merged[[cust_col, date_col]].dropna().copy()
    tmp["ym"] = tmp[date_col].dt.to_period("M")
    act_months = tmp.groupby(cust_col)["ym"].nunique().rename("ActiveMonths")
    merged = merged.merge(act_months, on=cust_col, how="left")
else:
    merged["ActiveMonths"] = 1

rev_cols = [c for c in merged.columns if "revenue" in c.lower() or "sales" in
c.lower()]
qty_cols = [c for c in merged.columns if "qty" in c.lower() or "quantity" in
c.lower()]
price_cols = [c for c in merged.columns if "price" in c.lower() or "charge" in
c.lower()]

merged["TotalRevenue"] = merged[rev_cols[0]] if rev_cols else (
    merged[price_cols[0]] * merged[qty_cols[0]] if (price_cols and qty_cols)
else 0.0
)
merged["ARPU"] = np.where(merged["ActiveMonths"] > 0, merged["TotalRevenue"] /
merged["ActiveMonths"], 0.0)
merged["ComplaintRate"] = 0.0

merged["MonthlyCharges"] = merged[price_cols[0]] if price_cols else
merged["TotalRevenue"]
merged["TotalCharges"] = merged["TotalRevenue"]

# ---------- Define Churn ----------
if "Churn" not in merged.columns and date_col and cust_col:
    last_date = merged[date_col].max()
    cutoff = last_date - pd.Timedelta(days=90)
    recent = merged[merged[date_col] >
cutoff].groupby(cust_col).size().rename("recent_orders")
    churn_df = merged[[cust_col]].drop_duplicates().merge(recent, on=cust_col,
how="left")
    churn_df["Churn"] = (churn_df["recent_orders"].fillna(0) == 0).astype(int)
    merged = merged.merge(churn_df[[cust_col, "Churn"]], on=cust_col,
how="left")
elif "Churn" not in merged.columns:
    merged["Churn"] = 0

# ---------- Save Integrated ----------
merged.to_csv("integrated_dataset.csv", index=False)
```

```
# ---------- Customer-level Aggregation ----------
if cust_col:
    agg = {
        "Churn": "max",
        "ActiveMonths": "max",
        "ARPU": "mean",
        "MonthlyCharges": "mean",
        "TotalCharges": "sum",
        "TotalRevenue": "sum",
    }
    cat_cols = [c for c in merged.columns if merged[c].dtype == "O"]
    for c in cat_cols[:5]:
        agg[c] = lambda s: s.mode().iloc[0] if not s.mode().empty else
s.iloc[0]
    customer_df = merged.groupby(cust_col).agg(agg).reset_index()
else:
    customer_df = merged.copy()
customer_df.to_csv("churn_dataset.csv", index=False)

# ---------- Modeling ----------
if customer_df["Churn"].nunique() == 2 and customer_df["Churn"].sum() > 0:
    y = customer_df["Churn"].astype(int)
    num_feats = [c for c in customer_df.columns if customer_df[c].dtype.kind in
"biufc" and c != "Churn"]
    cat_feats = [c for c in customer_df.columns if customer_df[c].dtype ==
"O"][:6]
    pre = ColumnTransformer([
        ("num", Pipeline([("imp", SimpleImputer(strategy="median")), ("sc",
StandardScaler(with_mean=False))]), num_feats),
        ("cat", Pipeline([("imp", SimpleImputer(strategy="most_frequent")),
("oh", OneHotEncoder(handle_unknown="ignore"))]), cat_feats),
    ])
    model = Pipeline([("pre", pre), ("clf", LogisticRegression(max_iter=300,
solver="liblinear"))])
    skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
    X = customer_df[num_feats + cat_feats]
    y_prob = cross_val_predict(model, X, y, cv=skf, method="predict_proba")[:,
1]
    auc = roc_auc_score(y, y_prob)
    fpr, tpr, thr = roc_curve(y, y_prob)
    j = tpr - fpr
    best_thr = thr[np.argmax(j)]
    y_pred = (y_prob >= best_thr).astype(int)
    brier = brier_score_loss(y, y_prob)
```

```
    cm = confusion_matrix(y, y_pred)
    pr, rc, f1, _ = precision_recall_fscore_support(y, y_pred,
average="binary")

    # ---------- ROC Plot ----------
    plt.figure()
    plt.plot(fpr, tpr, label=f"AUC={auc:.3f}")
    plt.plot([0,1],[0,1],"--")
    plt.xlabel("FPR"); plt.ylabel("TPR"); plt.title("ROC Curve"); plt.legend()
    plt.savefig("roc_curve.png", dpi=150, bbox_inches="tight"); plt.close()

    # ---------- Calibration Deciles ----------
    df_cal = pd.DataFrame({"y": y, "p": y_prob})
    df_cal["decile"] = pd.qcut(df_cal["p"], 10, labels=False,
duplicates="drop")
    cal = df_cal.groupby("decile").agg(pred=("p","mean"),
obs=("y","mean")).reset_index()
    plt.figure()
    plt.plot(cal["decile"], cal["pred"], "o-", label="Predicted")
    plt.plot(cal["decile"], cal["obs"], "s-", label="Observed")
    plt.xlabel("Decile"); plt.ylabel("Churn rate"); plt.title("Calibration by
Decile"); plt.legend()
    plt.savefig("calibration_deciles.png", dpi=150, bbox_inches="tight");
plt.close()

# ---------- Log-Scaled Overlay ----------
if date_col:
    cat_col = next((c for c in merged.columns if merged[c].dtype == "O" and c
!= cust_col), None)
    rev_col = next(
        (
            c
            for c in merged.columns
            if any(k in c.lower() for k in ["revenue", "sales", "total"])
            and merged[c].dtype.kind in "biufc"
        ),
        None,
    )
    if cat_col and rev_col:
        df_time = merged[[date_col, rev_col, "Churn", cat_col]].dropna().copy()
        if not df_time.empty:
            df_time["ym"] =
pd.to_datetime(df_time[date_col]).dt.to_period("M").dt.to_timestamp()
```

```
            rev_ts = df_time.groupby(["ym",
cat_col])[rev_col].mean().rename("avg_rev")
            churn_prof = df_time[df_time["Churn"]==1].groupby(["ym",
cat_col])[rev_col].mean().rename("churn_prof")
            overlay = pd.concat([rev_ts, churn_prof], axis=1).reset_index()
            if not overlay.empty:
                top_cat = df_time[cat_col].value_counts().idxmax()
                sub = overlay[overlay[cat_col]==top_cat].sort_values("ym")
                if not sub.empty:
                    plt.figure()
                    plt.plot(sub["ym"], sub["avg_rev"], label="Avg revenue")
                    plt.plot(sub["ym"], sub["churn_prof"], label="Churn
profitability")
                    plt.yscale("log"); plt.xlabel("Month"); plt.ylabel("Value
(log scale)")
                    plt.title(f"Log-scaled Overlay: {top_cat}"); plt.legend()
                    plt.savefig("churn_profitability_overlay.png", dpi=150,
bbox_inches="tight"); plt.close()
                else:
                    print(f"❌ Log-scaled overlay plot cannot be generated:
Subset for top category '{top_cat}' is empty.")
            else:
                print("❌ Log-scaled overlay plot cannot be generated: Overlay
DataFrame is empty.")
        else:
            print("❌ Log-scaled overlay plot cannot be generated: Filtered
DataFrame for plot is empty.")
    else:
        print("❌ Log-scaled overlay plot cannot be generated: Suitable
categorical or revenue column not found or revenue column is not numeric.")


# ---------- Save Summary ----------


summary = {
    "roc_auc": float(auc),
    "youden_threshold": float(best_thr),
    "brier": float(brier),
    "precision": float(pr),
    "recall": float(rc),
    "f1": float(f1),
    "confusion_matrix": cm.tolist(),
    "outputs": {
```

```
        "integrated_dataset": "integrated_dataset.csv",
        "churn_dataset": "churn_dataset.csv",
        "roc_curve": "roc_curve.png",
        "calibration_deciles": "calibration_deciles.png",
        "overlay_chart": "churn_profitability_overlay.png"
    }
}
with open("analysis_summary.json", "w") as f:
    json.dump(summary, f, indent=2)

print("✅ Analysis complete. All outputs saved.")
```