```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""


@author: fran-pellegrino
"""

# ---------- imports ----------
import pandas as pd
import numpy as np
import statsmodels.api as sm
import matplotlib.pyplot as plt
from sklearn.linear_model import Ridge, RidgeCV, Lasso, LassoCV, lasso_path, LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import seaborn as sns
from sklearn.metrics import r2_score, accuracy_score, roc_curve, mean_squared_error, classification_report, roc_auc_score, confusion_matrix
import scipy.stats as stats

#Question 1 below
df = pd.read_csv("techSalaries2017.csv")

cols = df.columns.tolist()

mapped = {
    'company': cols[0],
    'job_title': cols[1],
    'office_location': cols[2],
    'total_annual_compensation': cols[3],
    'base_salary': cols[4],
    'stock_grants': cols[5],
    'bonus': cols[6],
    'years_experience': cols[7],
    'time_with_company': cols[8],
    'gender': cols[9],
    'deg_masters': cols[10],
    'deg_bachelors': cols[11],
    'deg_doctorate': cols[12],
    'deg_highschool': cols[13],
    'deg_somecollege': cols[14],
    'asian': cols[15],
    'white': cols[16],
    'multi_racial': cols[17],
    'black': cols[18],
    'hispanic': cols[19],
    'race_qual': cols[20],
    'education_qual': cols[21],
    'age': cols[22],
    'height_inches': cols[23],
    'zodiac': cols[24],
    'sat': cols[25],
    'gpa': cols[26]
}

numeric_predictors = [
    mapped['years_experience'],
    mapped['time_with_company'],
    mapped['deg_masters'], mapped['deg_bachelors'], mapped['deg_doctorate'],
    mapped['deg_highschool'], mapped['deg_somecollege'],
    mapped['asian'], mapped['white'], mapped['multi_racial'],
    mapped['black'], mapped['hispanic'],
    mapped['age'], mapped['height_inches'],
    mapped['zodiac'], mapped['sat'], mapped['gpa']
]

target = mapped['total_annual_compensation']

# ensure numeric conversion and drop missing
df[target] = pd.to_numeric(df[target], errors='coerce')
for c in numeric_predictors:
    df[c] = pd.to_numeric(df[c], errors='coerce')

data = df[[target] + numeric_predictors].dropna()

# ---------- separate X and y ----------
y = data[target]
X = data[numeric_predictors]

# ---------- multiple regression ----------
X_const = sm.add_constant(X)
model_full = sm.OLS(y, X_const).fit()
print("\n=== Multiple Linear Regression Summary ===")
print(model_full.summary())

# full model R²
r2_full = model_full.rsquared
print(f"\nR² for full model: {r2_full:.4f}")

# ---------- find best single predictor ----------
r2_values = {}

for col in numeric_predictors:
    X_single = sm.add_constant(X[[col]])
    model_single = sm.OLS(y, X_single).fit()
    r2_values[col] = model_single.rsquared

# find best predictor (highest R²)
best_pred = max(r2_values, key=r2_values.get)
best_r2 = r2_values[best_pred]

print(f"\nBest single predictor: {best_pred}")
print(f"R² (variance explained) by {best_pred}: {best_r2:.4f}")
print(f"R² (variance explained) by full model: {r2_full:.4f}")
print(f"Additional variance explained by other predictors: {r2_full - best_r2:.4f}")

# ---------- optional: inspect coefficient for best predictor ----------
best_model = sm.OLS(y, sm.add_constant(X[[best_pred]])).fit()
print(f"\n=== Simple Regression on {best_pred} ===")
print(best_model.summary())

# Visual for predictors ranked by individual R²
r2_sorted = dict(sorted(r2_values.items(), key=lambda item: item[1], reverse=True))

plt.figure(figsize=(10, 6))
plt.barh(list(r2_sorted.keys()), list(r2_sorted.values()), color='cornflowerblue')
plt.gca().invert_yaxis()  # best predictor at top
plt.xlabel('Individual R² (Variance Explained)')
plt.title('Predictive Strength of Individual Variables for Total Compensation')

plt.show()
# visual for showing gap between variance explaiend by experience and variance by full model

# R² values
r2_single = 0.1788
r2_full = 0.2851
r2_additional = r2_full - r2_single

# Plot setup
labels = ['Best Predictor\n(Years Experience)', 'Full Model']
values = [r2_single, r2_full]

plt.figure(figsize=(6, 5))
bars = plt.bar(labels, values, color=['skyblue', 'seagreen'])
plt.ylabel('Variance Explained (R²)')
plt.title('Explained Variance: Best Predictor vs. Full Regression Model')

# Annotate bars with % values
for bar in bars:
    height = bar.get_height()
    plt.text(bar.get_x() + bar.get_width()/2, height + 0.01,
             f"{height:.2f}", ha='center', va='bottom', fontsize=11)

plt.ylim(0, 0.35)
plt.tight_layout()
plt.show()

# Question 2 below

print("\n=== Ridge Regression Analysis ===\n")
# ensure numeric
df[target] = pd.to_numeric(df[target], errors='coerce')
for c in numeric_predictors:
    df[c] = pd.to_numeric(df[c], errors='coerce')

data = df[[target] + numeric_predictors].dropna()

# ---------- X, y split ----------
X = data[numeric_predictors]
y = data[target]

# ---------- train/test split ----------
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# ---------- scale predictors ----------
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# ---------- Ridge regression with CV for alpha ----------
alphas = np.logspace(-3, 3, 100)
ridge_cv = RidgeCV(alphas=alphas, store_cv_results=True)
ridge_cv.fit(X_train_scaled, y_train)

best_alpha = ridge_cv.alpha_
print(f"Best alpha: {best_alpha:.4f}")

# fit ridge with best alpha
ridge_model = Ridge(alpha=best_alpha)
ridge_model.fit(X_train_scaled, y_train)

# evaluate performance
y_pred_train = ridge_model.predict(X_train_scaled)
y_pred_test = ridge_model.predict(X_test_scaled)

r2_train = r2_score(y_train, y_pred_train)
r2_test = r2_score(y_test, y_pred_test)

print(f"R² (train): {r2_train:.4f}")
print(f"R² (test): {r2_test:.4f}")

# ---------- check importance (coefficients) ----------
coefs = pd.Series(ridge_model.coef_, index=numeric_predictors)
coefs_sorted = coefs.abs().sort_values(ascending=False)

best_pred = coefs_sorted.index[0]
print(f"\nBest predictor (highest absolute coefficient): {best_pred}")
print(coefs_sorted.head(5))

# ---------- visualize coefficient magnitudes ----------
plt.figure(figsize=(10,6))
plt.barh(coefs_sorted.index, coefs_sorted.values, color='teal')
plt.gca().invert_yaxis()
plt.title('Ridge Regression: Predictor Importance (Absolute Coefficients)')
plt.xlabel('Coefficient Magnitude')
plt.tight_layout()
plt.show()

# ---------- single predictor model for comparison ----------
# rebuild ridge with only best predictor
X_best_train = X_train[[best_pred]].values
X_best_test = X_test[[best_pred]].values

Xb_train_scaled = scaler.fit_transform(X_best_train)
Xb_test_scaled = scaler.transform(X_best_test)

ridge_single = Ridge(alpha=best_alpha)
ridge_single.fit(Xb_train_scaled, y_train)
y_pred_single = ridge_single.predict(Xb_test_scaled)
r2_single = r2_score(y_test, y_pred_single)

print(f"\nR² (variance explained) by best predictor ({best_pred}): {r2_single:.4f}")
print(f"R² (variance explained) by full Ridge model: {r2_test:.4f}")
print(f"Additional variance explained by other predictors: {r2_test - r2_single:.4f}")

# ---------- bar chart comparing single vs full ----------
labels = [f'Best Predictor\n({best_pred})', 'Full Ridge Model']
values = [r2_single, r2_test]

plt.figure(figsize=(6,5))
bars = plt.bar(labels, values, color=['skyblue','seagreen'])
plt.ylabel('Variance Explained (R²)')
plt.title('Explained Variance: Best Predictor vs. Full Ridge Model')

for bar in bars:
    h = bar.get_height()
    plt.text(bar.get_x() + bar.get_width()/2, h + 0.01, f"{h:.2f}", ha='center', va='bottom', fontsize=11)

plt.ylim(0, 0.35)
plt.tight_layout()
plt.show()

# Question 3 below

print("\n=== Lasso Regression Analysis ===\n")

# --------- Ensure numeric and drop rows with missing target or predictors ----------
df[target] = pd.to_numeric(df[target], errors='coerce')
for c in numeric_predictors:
    df[c] = pd.to_numeric(df[c], errors='coerce')

data = df[[target] + numeric_predictors].dropna().reset_index(drop=True)
print("Rows after dropping NA:", data.shape[0])

X = data[numeric_predictors].values
y = data[target].values

# --------- train/test split (IMPORTANT: split before any scaling/tuning) ----------
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# --------- scale features (Lasso requires scaling for meaningful regularization) ----------
scaler = StandardScaler()
X_train_s = scaler.fit_transform(X_train)
X_test_s = scaler.transform(X_test)

# --------- LassoCV to choose alpha (lambda) via cross-validation ----------
# Use a logspace grid internally; LassoCV will search and return alpha_
alphas_path = np.logspace(-4, 2, 200)   # wide grid from 1e-4 to 1e2
lasso_cv = LassoCV(alphas=alphas, cv=5, max_iter=10000, n_jobs=-1, random_state=42)
lasso_cv.fit(X_train_s, y_train)

best_alpha = lasso_cv.alpha_
print(f"\nOptimal alpha (lambda) found by CV: {best_alpha:.6g}")

# --------- Fit final Lasso model with best alpha ----------
lasso_final = Lasso(alpha=best_alpha, max_iter=10000)
lasso_final.fit(X_train_s, y_train)

# --------- Performance on train/test ----------
y_pred_train = lasso_final.predict(X_train_s)
y_pred_test = lasso_final.predict(X_test_s)

r2_train = r2_score(y_train, y_pred_train)
r2_test = r2_score(y_test, y_pred_test)
rmse_test = np.sqrt(mean_squared_error(y_test, y_pred_test))

print(f"Train R²: {r2_train:.4f}")
print(f"Test  R²: {r2_test:.4f}")
print(f"Test  RMSE: ${rmse_test:,.2f}")

# --------- Coefficients and sparsity info ----------
coef = pd.Series(lasso_final.coef_, index=numeric_predictors)
n_total = coef.shape[0]
n_zero = (coef == 0).sum()
n_nonzero = n_total - n_zero
print(f"\nOut of {n_total} predictors, {n_zero} coefficients were shrunk to EXACTLY zero by Lasso.")
print(f"Number of nonzero coefficients: {n_nonzero}")
print("\nNonzero coefficients (sorted by magnitude):")
print(coef[coef != 0].abs().sort_values(ascending=False))

# --------- OPTIONAL: show the selected best single predictor as in prior tasks
# We'll compute single-predictor R² for the test set to compare (for chosen best single predictor by coefficient magnitude)
if n_nonzero > 0:
    best_pred = coef.abs().idxmax()
    print(f"\nFeature with largest |beta| in Lasso: {best_pred}")
    # simple linear regression on scaled single predictor for comparison
    from sklearn.linear_model import LinearRegression
    lr = LinearRegression().fit(X_train_s[:, numeric_predictors.index(best_pred)].reshape(-1,1), y_train)
    r2_single_test = r2_score(y_test, lr.predict(X_test_s[:, numeric_predictors.index(best_pred)].reshape(-1,1)))
    print(f"R² on test using that single predictor (simple linear reg): {r2_single_test:.4f}")

# --------- VISUAL 1: Coefficient paths across alphas (lasso_path) ----------

# NOTE: We already scaled X_train_s with StandardScaler (mean=0). Do NOT pass fit_intercept to lasso_path.
# Use the same alphas array you fed into LassoCV for consistency (or create a fine grid).

# Compute LASSO path (coefs_path shape: n_features x n_alphas)
alphas_out, coefs_path, _ = lasso_path(X_train_s, y_train, alphas=alphas_path, max_iter=10000)

# Plot coefficient paths
plt.figure(figsize=(12, 8))

# Plot every path and label each (so legend will include all predictors)
for i, feat in enumerate(numeric_predictors):
    plt.plot(np.log10(alphas_out), coefs_path[i, :], label=feat, linewidth=1.25)

# Vertical line at chosen alpha
plt.axvline(np.log10(best_alpha), color='k', linestyle='--', linewidth=1, label=f'Best α={best_alpha:.2g}')

# Annotate only the top-3 predictors (by absolute final coefficient in the fitted Lasso model)
top3 = coef.abs().sort_values(ascending=False).index[:3].tolist()
for feat in top3:
    idx = numeric_predictors.index(feat)
    # find closest column in alphas_out to best_alpha
    idx_alpha = np.argmin(np.abs(alphas_out - best_alpha))
    coef_at_best = coefs_path[idx, idx_alpha]
    # place the text a little to the right of the vertical line to avoid overlap
    x_text = np.log10(alphas_out[idx_alpha]) + 0.03
    plt.text(x_text, coef_at_best, feat, fontsize=10, fontweight='bold',
             bbox=dict(facecolor='white', alpha=0.6, edgecolor='none', pad=1))

# Legend: show full mapping, place it outside the main axes to avoid covering paths
plt.legend(loc='center left', bbox_to_anchor=(1.02, 0.5), fontsize='small', ncol=2)

plt.xlabel('log10(alpha)')
plt.ylabel('Coefficient value')
plt.title('Lasso coefficient paths (all predictors) — top 3 labeled on plot')
plt.grid(alpha=0.3)
plt.tight_layout(rect=[0,0,0.78,1])  # leave space on right for legend
plt.show()

# Question 4

print("\n=== Logistic Regression Analysis ===\n")

df_raw = pd.read_csv("techSalaries2017.csv")
cols = df_raw.columns.tolist()

df = df_raw[df_raw[mapped['gender']].isin(['Male', 'Female'])].copy()
df['gender_encoded'] = df[mapped['gender']].map({'Male': 0, 'Female': 1})

# Ensure numeric conversion for these numeric columns (coerce errors to NaN)
numeric_predictors = [
    mapped['total_annual_compensation'],    # compensation (we keep in original units)
    mapped['years_experience'],
    mapped['time_with_company'],
    mapped['deg_masters'], mapped['deg_bachelors'], mapped['deg_doctorate'],
    mapped['deg_highschool'], mapped['deg_somecollege'],
    mapped['asian'], mapped['white'], mapped['multi_racial'],
    mapped['black'], mapped['hispanic'],
    mapped['age'], mapped['height_inches'],
    mapped['zodiac'], mapped['sat'], mapped['gpa']
]

for c in numeric_predictors:
    df[c] = pd.to_numeric(df[c], errors='coerce')

# Drop rows with NA in any of the numeric predictors or gender (so both models use the same data)
df_clean = df.dropna(subset=numeric_predictors + ['gender_encoded']).copy()
print("Rows used after filtering and dropping NA:", df_clean.shape[0])

# ---------- Train/Test split (stratify by gender) ----------
train_df, test_df = train_test_split(df_clean, test_size=0.2, random_state=42,
                                     stratify=df_clean['gender_encoded'])
print("Train / Test shapes:", train_df.shape, test_df.shape)

# ---------- Model A: Gender ~ total_annual_compensation (uncontrolled) ----------
X_A_train = sm.add_constant(train_df[[mapped['total_annual_compensation']]])
y_train = train_df['gender_encoded']
logit_A = sm.Logit(y_train, X_A_train).fit(disp=False)

# Evaluate on test set
X_A_test = sm.add_constant(test_df[[mapped['total_annual_compensation']]])
y_test = test_df['gender_encoded']
proba_A_test = logit_A.predict(X_A_test)
pred_A_test = (proba_A_test >= 0.5).astype(int)

# Model A metrics
coef_A_comp = logit_A.params[mapped['total_annual_compensation']]
pval_A_comp = logit_A.pvalues[mapped['total_annual_compensation']]
oddsratio_A_comp = np.exp(coef_A_comp)
aic_A = logit_A.aic
bic_A = logit_A.bic
psr_A = logit_A.prsquared  # McFadden's pseudo-R2
auc_A = roc_auc_score(y_test, proba_A_test)
acc_A = accuracy_score(y_test, pred_A_test)

print("\n=== Model A: Gender ~ Total Annual Compensation ===")
print(logit_A.summary())
print(f"Compensation coefficient (beta): {coef_A_comp:.6e}")
print(f"p-value: {pval_A_comp:.4g}")
print(f"Odds ratio (exp(beta)): {oddsratio_A_comp:.6f}")
print(f"AIC: {aic_A:.3f}, BIC: {bic_A:.3f}, McFadden pseudo-R²: {psr_A:.4f}")
print(f"Test AUC: {auc_A:.4f}, Test accuracy (0.5 cutoff): {acc_A:.4f}")

# Confusion / classification report for Model A
print("\nConfusion matrix (Model A, test):")
print(confusion_matrix(y_test, pred_A_test))
print("\nClassification report (Model A, test):")
print(classification_report(y_test, pred_A_test, digits=4))

# ---------- Model B: Gender ~ Compensation + controls ----------
predictors_B = [
    mapped['total_annual_compensation'],
    mapped['years_experience'],
    mapped['time_with_company'],
    mapped['deg_masters'], mapped['deg_bachelors'], mapped['deg_doctorate'],
    mapped['deg_highschool'], mapped['deg_somecollege'],
    mapped['asian'], mapped['white'], mapped['multi_racial'],
    mapped['black'], mapped['hispanic'],
    mapped['age'], mapped['height_inches'],
    mapped['zodiac'], mapped['sat'], mapped['gpa']
]

# Build train/test design matrices for model B (preserve indices)
X_B_train = train_df[predictors_B].copy()
X_B_test = test_df[predictors_B].copy()
y_train_B = train_df['gender_encoded']
y_test_B = test_df['gender_encoded']

# Add constant and fit Logit (no scaling so compensation coefficient is in $ units)
X_B_train_const = sm.add_constant(X_B_train)
logit_B = sm.Logit(y_train_B, X_B_train_const).fit(disp=False)

# Evaluate on test
X_B_test_const = sm.add_constant(X_B_test)
proba_B_test = logit_B.predict(X_B_test_const)
pred_B_test = (proba_B_test >= 0.5).astype(int)

# Model B metrics and compensation coeff
coef_B_comp = logit_B.params[mapped['total_annual_compensation']]
pval_B_comp = logit_B.pvalues[mapped['total_annual_compensation']]
oddsratio_B_comp = np.exp(coef_B_comp)
aic_B = logit_B.aic
bic_B = logit_B.bic
psr_B = logit_B.prsquared
auc_B = roc_auc_score(y_test_B, proba_B_test)
acc_B = accuracy_score(y_test_B, pred_B_test)

print("\n=== Model B: Gender ~ Compensation + Controls ===")
print(logit_B.summary())
print(f"Compensation coefficient (beta, controlled): {coef_B_comp:.6e}")
print(f"p-value (controlled): {pval_B_comp:.4g}")
print(f"Odds ratio (exp(beta), controlled): {oddsratio_B_comp:.6f}")
print(f"AIC: {aic_B:.3f}, BIC: {bic_B:.3f}, McFadden pseudo-R²: {psr_B:.4f}")
print(f"Test AUC: {auc_B:.4f}, Test accuracy (0.5 cutoff): {acc_B:.4f}")

print("\nConfusion matrix (Model B, test):")
print(confusion_matrix(y_test_B, pred_B_test))
print("\nClassification report (Model B, test):")
print(classification_report(y_test_B, pred_B_test, digits=4))

# ---------- Visual 1: Model A — binned empirical proportion female vs compensation with logistic fit ----------
plt.figure(figsize=(8,6))
# create bins for compensation
bins = np.quantile(df_clean[mapped['total_annual_compensation']], np.linspace(0,1,25))
# use test set for visualization of fit
test_comp = X_A_test[mapped['total_annual_compensation']].values
df_test_plot = pd.DataFrame({
    'compensation': test_comp,
    'female': y_test.values,
    'prob_model': proba_A_test
})

# compute bin centers and empirical proportion female
df_test_plot['bin'] = pd.cut(df_test_plot['compensation'], bins=bins, include_lowest=True)
bin_stats = df_test_plot.groupby('bin').agg(
    comp_mean=('compensation','mean'),
    prop_female=('female','mean'),
    n=('female','size')
).dropna()

# scatter empirical proportions (circle size by n)
plt.scatter(bin_stats['comp_mean'], bin_stats['prop_female'], s=bin_stats['n']*2,
            label='Empirical proportion female (binned)', alpha=0.7)

# logistic curve line across compensation range (use model A params)
comp_range = np.linspace(df_clean[mapped['total_annual_compensation']].min(),
                         df_clean[mapped['total_annual_compensation']].max(), 300)
X_curve = sm.add_constant(pd.DataFrame({mapped['total_annual_compensation']: comp_range}))
curve_probs = logit_A.predict(X_curve)

plt.plot(comp_range, curve_probs, color='red', lw=2, label='Fitted logistic curve (Model A)')
plt.xlabel('Total Annual Compensation ($)')
plt.ylabel('Probability of being Female')
plt.title('Model A — Empirical proportion female (binned) with fitted logistic curve')
plt.legend()
plt.grid(alpha=0.3)
plt.tight_layout()
plt.show()

# ---------- Visual 2: Model B — Partial-dependence style curve (compensation effect controlling for others) ----------
# Hold all other predictors at their median (train set medians)
medians = X_B_train.median()

# Build a grid over compensation values and predict probability while holding others fixed
comp_vals = np.linspace(df_clean[mapped['total_annual_compensation']].quantile(0.01),
                        df_clean[mapped['total_annual_compensation']].quantile(0.99), 300)

# Build grid DataFrame having same columns and order as X_B_train
pd_grid = pd.DataFrame(np.tile(medians.values, (comp_vals.shape[0], 1)),
                       columns=medians.index,
                       index=range(comp_vals.shape[0]))

# Overwrite the compensation column with the grid values
pd_grid[mapped['total_annual_compensation']] = comp_vals

# Now add constant and reindex columns to exactly match training design matrix (including const)
pd_grid_const = sm.add_constant(pd_grid)

# Ensure same columns and order as X_B_train_const used to fit logit_B
# X_B_train_const was built earlier as: X_B_train_const = sm.add_constant(X_B_train)
expected_cols = X_B_train_const.columns.tolist()
pd_grid_const = pd_grid_const.reindex(columns=expected_cols, fill_value=0)

# Sanity check: indices/columns should match shape
assert pd_grid_const.shape[1] == len(logit_B.params), \
       f"Column mismatch: pd_grid_const has {pd_grid_const.shape[1]} cols; model expects {len(logit_B.params)} params"

# Predict probabilities on the aligned grid
probs_pd = logit_B.predict(pd_grid_const)

# Plot the partial-dependence style curve
plt.figure(figsize=(8,6))
plt.plot(comp_vals, probs_pd, color='navy', lw=2, label='Predicted P(Female) controlling for other predictors')
# show rug for training observations for context
sns.rugplot(train_df[train_df['gender_encoded']==1][mapped['total_annual_compensation']].values,
            height=0.04, color='magenta', label='Female (train) - rug')
sns.rugplot(train_df[train_df['gender_encoded']==0][mapped['total_annual_compensation']].values,
            height=0.02, color='gray', label='Male (train) - rug')

plt.xlabel('Total Annual Compensation ($)')
plt.ylabel('Predicted Probability of being Female (controlling)')
plt.title('Model B — Predicted P(Female) vs Compensation (Other predictors at median)')
plt.legend(loc='upper right')
plt.grid(alpha=0.3)
plt.tight_layout()
plt.show()

# Extra Credit Part 1

print("\n=== Extra Credit Part 1 ===\n")

variables = ['Height', 'totalyearlycompensation', 'Age']

for var in variables:
    plt.figure(figsize=(12,4))

    # Histogram + KDE
    plt.subplot(1,2,1)
    sns.histplot(df[var].dropna(), kde=True, bins=30, color='skyblue')
    plt.title(f'{var} Distribution (Histogram + KDE)')
    plt.xlabel(var)

    # Q-Q Plot
    plt.subplot(1,2,2)
    stats.probplot(df[var].dropna(), dist="norm", plot=plt)
    plt.title(f'{var} Q-Q Plot')

    plt.tight_layout()
    plt.show()

for var in variables:
    stat, p = stats.shapiro(df[var].dropna())
    print(f"{var}: Shapiro-Wilk p-value = {p:.4f}")

# Extra Credit Part 2

print("\n=== Extra Credit Part 2 ===\n")

companies = ["nvidia", "apple", "amazon", "google"]
titles = ["software engineer", "data scientist", "product manager"]

df_filtered = df[
    df["company"].str.lower().isin(companies) &
    df["title"].str.lower().isin(titles)
].copy()

# Clean up capitalization for readability
df_filtered["company"] = df_filtered["company"].str.title()
df_filtered["title"] = df_filtered["title"].str.title()

# -----------------------------
# 2. Compute mean and std of compensation
# -----------------------------
summary_stats = (
    df_filtered.groupby(["company", "title"])["totalyearlycompensation"]
    .agg(["mean", "std", "count"])
    .reset_index()
    .sort_values(by="mean", ascending=False)
)

print("=== Mean and Standard Deviation of Total Compensation ===")
print(summary_stats.to_string(index=False))

# -----------------------------
# 3. Visualization 1 — Grouped Bar Chart (Mean Comp)
# -----------------------------
plt.figure(figsize=(10,6))
sns.barplot(
    data=summary_stats,
    x="title",
    y="mean",
    hue="company",
    palette="viridis",
    edgecolor="black"
)
plt.title("Mean Total Compensation by Role Across Major Tech Companies", fontsize=14)
plt.xlabel("title")
plt.ylabel("Mean Total Annual Compensation ($)")
plt.xticks(rotation=20)
plt.legend(title="company", bbox_to_anchor=(1.05, 1), loc='upper left')
plt.tight_layout()
plt.show()

# -----------------------------
# 4. Visualization 2 — Heatmap (Job Title × Compan
# -----------------------------
pivot = summary_stats.pivot(index="title", columns="company", values="mean")
plt.figure(figsize=(8,5))
sns.heatmap(
    pivot,
    annot=True,
    fmt=".0f",
    cmap="magma",
    linewidths=0.3,
    cbar_kws={"label": "Mean Total Compensation ($)"}
)
plt.title("Heatmap of Mean Compensation by Role and Company", fontsize=14)
plt.ylabel("title")
plt.xlabel("company")
plt.tight_layout()
plt.show()

# -----------------------------
# 5. Visualization 3 — Faceted Boxplot (Spread of Compensation)
# -----------------------------
g = sns.catplot(
    data=df_filtered,
    x="title",
    y="totalyearlycompensation",
    kind="box",
    col="company",
    palette="Set2",
    col_wrap=2,
    height=4,
    sharey=False
)
g.set_titles("{col_name}")
g.set_axis_labels("title", "Total Annual Compensation ($)")
plt.subplots_adjust(top=0.85)
g.fig.suptitle("Distribution of Compensation by Title, Faceted by Company", fontsize=14)
plt.show()
```