Lab2: Linear Regression

Mehreen Ali Gillani

2026-02-11

```
# Import necessary libraries
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from scipy import stats
# Load the dataset
url = 'https://raw.githubusercontent.com/georgehagstrom/DATA622Spring2026/main/website/assig
earnings = pd.read_csv(url)
# Display the first few rows of the dataset
earnings.head()
```

	height	weight	male	earn	earnk	ethnicity	education	mother_education	father_education
0	74	210.0	1	50000.0	50.0	White	16.0	12.0	12.0
1	66	125.0	0	60000.0	60.0	White	16.0	12.0	12.0
2	64	126.0	0	30000.0	30.0	White	16.0	12.0	12.0
3	65	200.0	0	25000.0	25.0	White	17.0	12.0	NaN
4	63	110.0	0	50000.0	50.0	Other	16.0	15.0	12.0

a) EDA: Perform a basic exploratory data analysis on this dataset. Tell me how you decided to handle rows with missing values, and show me the plots of the distributions of variables and the relationships between variables that you believe are most important. An important decision is how to handle earnk, the target variable, and whether to log transform it or not. Look at the distribution of earnk and discuss the pros and cons of using the log transform, and make a decision about whether perform to the transformation or not.

EDA

Missing Values Analysis

```
# Check for missing values and their percentages
print((earnings.isnull().mean() * 100).round(2))
```

```
height               0.00
weight               1.49
male                 0.00
earn                 0.00
earnk                0.00
ethnicity            0.00
education            0.11
mother_education    13.44
father_education    16.24
walk                 0.00
exercise             0.00
smokenow             0.06
tense                0.11
angry                0.00
age                  0.00
dtype: float64
```

Minimal missing (<1%): education (2), smokenow (1), tense (1), angry (1) Moderate missing: weight (27), 1.49% Substantial missing: mother_education (244), father_education (295) 13.44% and 16.24% respectively # Decision: Drop rows with missing values in the target variable 'earnk' and moderate missing 'weight', but impute or drop the substantial missing 'mother_education' and 'father_education' # Drop rows with missing values in 'earnk' and 'weight'

```
# import necessary libraries for imputation
from sklearn.impute import SimpleImputer
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer
import statsmodels.api as sm

earnings_cleaned = earnings.copy()
#  Mode imputation for categorical variables with minimal missingness
earnings_cleaned['angry'].fillna(earnings_cleaned['angry'].mode()[0], inplace=True)
```

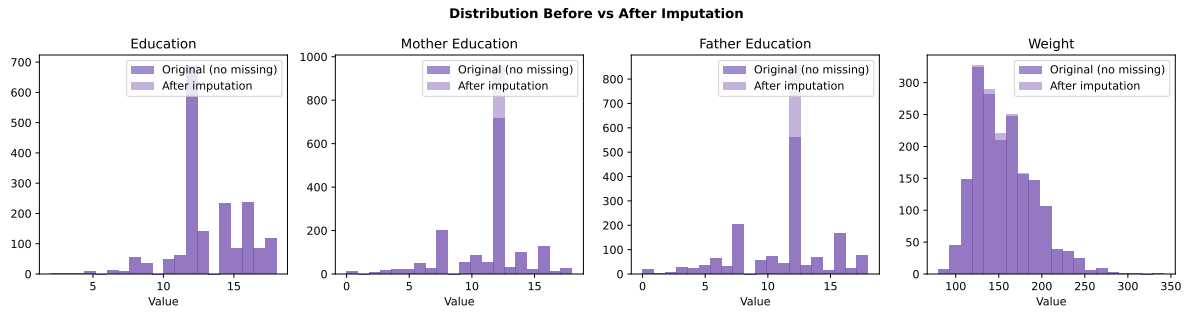
```
earnings_cleaned['tense'].fillna(earnings_cleaned['tense'].mode()[0], inplace=True)

# Mode imputation for 'education' and 'smokenow' with minimal missingness
imputer = SimpleImputer(strategy='most_frequent')
earnings_cleaned[['education', 'smokenow']] = imputer.fit_transform(earnings_cleaned[['educat


# Create imputation model using complete variables
imputer = IterativeImputer(max_iter=10, random_state=0)
earnings_cleaned['weight'] = imputer.fit_transform(earnings_cleaned[['weight', 'height', 'mal
print((earnings_cleaned.isnull().mean() * 100).round(2))


# Create missing indicators and impute in-place
earnings_cleaned['mother_edu_missing'] = earnings_cleaned['mother_education'].isnull().astype
earnings_cleaned['father_edu_missing'] = earnings_cleaned['father_education'].isnull().astype
earnings_cleaned['mother_education'] = earnings_cleaned['mother_education'].fillna(earnings_
earnings_cleaned['father_education'] = earnings_cleaned['father_education'].fillna(earnings_

# Verify
print(f"\nVerification - any remaining missing?")
print(f"Mother education: {earnings_cleaned['mother_education'].isnull().sum()}")
print(f"Father education: {earnings_cleaned['father_education'].isnull().sum()}")
```

```
height              0.00
weight              0.00
male                0.00
earn                0.00
earnk               0.00
ethnicity           0.00
education           0.00
mother_education    13.44
father_education    16.24
walk                0.00
exercise            0.00
smokenow            0.00
tense               0.00
angry               0.00
age                 0.00
dtype: float64

Verification - any remaining missing?
```

```
Mother education: 0
Father education: 0


/var/folders/6w/rkqvbjtx04s93768hcryddtc0000gn/T/ipykernel_63361/1825402175.py:9: FutureWarni
The behavior will change in pandas 3.0. This inplace method will never work because the inter

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: val


  earnings_cleaned['angry'].fillna(earnings_cleaned['angry'].mode()[0], inplace=True)
/var/folders/6w/rkqvbjtx04s93768hcryddtc0000gn/T/ipykernel_63361/1825402175.py:10: FutureWarn
The behavior will change in pandas 3.0. This inplace method will never work because the inter

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: val


  earnings_cleaned['tense'].fillna(earnings_cleaned['tense'].mode()[0], inplace=True)
```

Before and After Imputation Distributions

```
fig, axes = plt.subplots(1, 4, figsize=(15, 4))

for i, var in enumerate(['education','mother_education', 'father_education', 'weight']):
    # Plot before (original with missing removed) vs after (imputed)
    axes[i].hist(earnings[var].dropna(), bins=20, alpha=0.5, label='Original (no missing)', c
    axes[i].hist(earnings_cleaned[var], bins=20, alpha=0.5, label='After imputation', color=
    axes[i].set_title(f'{var.replace("_", " ").title()}')
    axes[i].set_xlabel('Value')
    axes[i].legend()

plt.suptitle('Distribution Before vs After Imputation', fontweight='bold')
plt.tight_layout()
plt.show()
```



Summary statistics before and after imputation

```
# Before and after immputation summary

for var in ['mother_education', 'father_education', 'weight', 'education']:
    if earnings[var].isnull().sum() > 0:
        print(f"{var:20} Standard deviation:{earnings[var].std():6.1f} →{earnings_cleaned[var
              f"Mean:{earnings[var].mean():6.1f} →{earnings_cleaned[var].mean():6.1f} "
              f"Median:{earnings[var].median():6.1f} →{earnings_cleaned[var].median():6.1f}"
```

```
mother_education     Standard deviation:   3.2 →   2.9 Mean:  11.2 →  11.3 Median:  12.0 →
father_education     Standard deviation:   3.7 →   3.4 Mean:  11.2 →  11.3 Median:  12.0 →
weight               Standard deviation:  34.6 →  34.4 Mean: 156.3 → 156.1 Median: 150.0 → 15
education            Standard deviation:   2.6 →   2.6 Mean:  13.2 →  13.2 Median:  12.0 →
```

Visualizations

```
# Set style
sns.set_style("whitegrid")
plt.rcParams['figure.figsize'] = (15, 12)

# 1. DISTRIBUTION PLOTS - Key Variables
fig, axes = plt.subplots(3, 3, figsize=(15, 12))

# Earnings (target variable)
axes[0,0].hist(earnings_cleaned['earnk'], bins=30, edgecolor='black', alpha=0.7)
axes[0,0].set_title('Earnings Distribution (earnk)', fontsize=12, fontweight='bold')
axes[0,0].set_xlabel('Annual Earnings ($)')
axes[0,0].set_ylabel('Frequency')
```

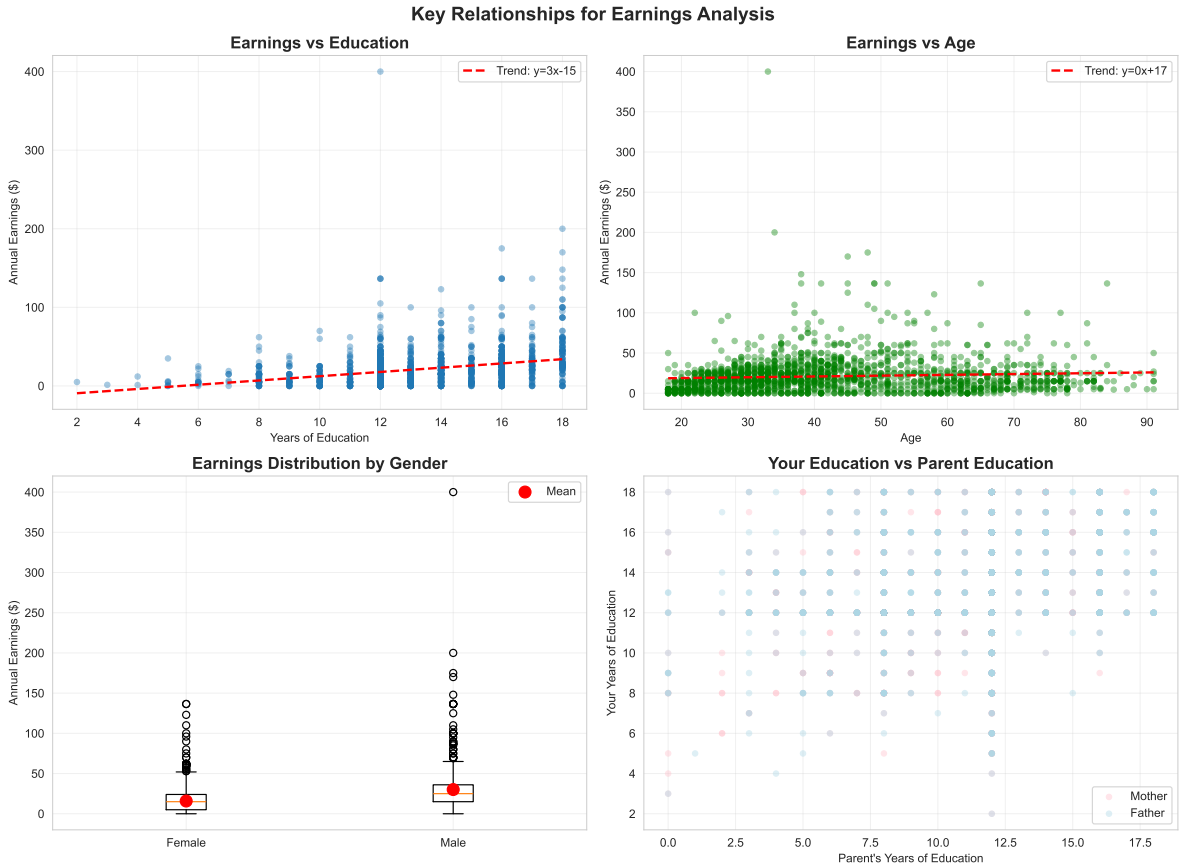
```
# Education
axes[0,1].hist(earnings_cleaned['education'], bins=20, edgecolor='black', alpha=0.7, color='g
axes[0,1].set_title('Education Distribution', fontsize=12, fontweight='bold')
axes[0,1].set_xlabel('Years of Education')
axes[0,1].set_ylabel('Frequency')

# Age
axes[0,2].hist(earnings_cleaned['age'], bins=30, edgecolor='black', alpha=0.7, color='orange
axes[0,2].set_title('Age Distribution', fontsize=12, fontweight='bold')
axes[0,2].set_xlabel('Age')
axes[0,2].set_ylabel('Frequency')

# Parent Education (after imputation)
axes[1,0].hist(earnings_cleaned['mother_education'], bins=20, edgecolor='black', alpha=0.5, c
axes[1,0].hist(earnings_cleaned['father_education'], bins=20, edgecolor='black', alpha=0.5, c
axes[1,0].set_title('Parent Education Distribution', fontsize=12, fontweight='bold')
axes[1,0].set_xlabel('Years of Education')
axes[1,0].set_ylabel('Frequency')
axes[1,0].legend()

# Height vs Weight
axes[1,1].scatter(earnings_cleaned['height'], earnings_cleaned['weight'], alpha=0.5, s=10)
axes[1,1].set_title('Height vs Weight', fontsize=12, fontweight='bold')
axes[1,1].set_xlabel('Height')
axes[1,1].set_ylabel('Weight')

# Gender distribution
gender_counts = earnings_cleaned['male'].value_counts()
axes[1,2].bar(['Female', 'Male'], gender_counts.values, color=['pink', 'lightblue'], edgecolo
axes[1,2].set_title('Gender Distribution', fontsize=12, fontweight='bold')
axes[1,2].set_ylabel('Count')

# Missing indicators
missing_data = earnings_cleaned[['mother_edu_missing', 'father_edu_missing']].sum()
axes[2,0].bar(['Mother Edu Missing', 'Father Edu Missing'], missing_data.values, color=['red
axes[2,0].set_title('Missing Data Indicators', fontsize=12, fontweight='bold')
axes[2,0].set_ylabel('Count')

# Exercise habits
if 'exercise' in earnings_cleaned.columns:
    exercise_counts = earnings_cleaned['exercise'].value_counts().sort_index()
    axes[2,1].bar(range(len(exercise_counts)), exercise_counts.values, edgecolor='black', col
```

```
    axes[2,1].set_title('Exercise Frequency', fontsize=12, fontweight='bold')
    axes[2,1].set_xlabel('Exercise Level')
    axes[2,1].set_ylabel('Count')
    axes[2,1].set_xticks(range(len(exercise_counts)))
    axes[2,1].set_xticklabels(exercise_counts.index)

# Ethnicity distribution (top 5 only)
if 'ethnicity' in earnings_cleaned.columns:
    ethnicity_counts = earnings_cleaned['ethnicity'].value_counts().head(5)
    axes[2,2].bar(range(len(ethnicity_counts)), ethnicity_counts.values, edgecolor='black', c
    axes[2,2].set_title('Top 5 Ethnicities', fontsize=12, fontweight='bold')
    axes[2,2].set_ylabel('Count')
    axes[2,2].set_xticks(range(len(ethnicity_counts)))
    axes[2,2].set_xticklabels(ethnicity_counts.index, rotation=45, ha='right')

plt.tight_layout()
plt.suptitle('Key Variable Distributions', fontsize=16, fontweight='bold', y=1.02)
plt.show()

# 2. RELATIONSHIP PLOTS - Most Important Correlations
fig, axes = plt.subplots(2, 2, figsize=(14, 10))

# Plot 1: Earnings vs Education (THE MOST IMPORTANT)
axes[0,0].scatter(earnings_cleaned['education'], earnings_cleaned['earnk'], alpha=0.4, s=20)
# Add trend line
z = np.polyfit(earnings_cleaned['education'], earnings_cleaned['earnk'], 1)
p = np.poly1d(z)
education_range = np.linspace(earnings_cleaned['education'].min(), earnings_cleaned['educatio
axes[0,0].plot(education_range, p(education_range), "r--", linewidth=2, label=f'Trend: y={z[0
axes[0,0].set_title('Earnings vs Education', fontsize=14, fontweight='bold')
axes[0,0].set_xlabel('Years of Education')
axes[0,0].set_ylabel('Annual Earnings ($)')
axes[0,0].legend()
axes[0,0].grid(True, alpha=0.3)

# Plot 2: Earnings vs Age
axes[0,1].scatter(earnings_cleaned['age'], earnings_cleaned['earnk'], alpha=0.4, s=20, color=
# Add trend line
z = np.polyfit(earnings_cleaned['age'], earnings_cleaned['earnk'], 1)
p = np.poly1d(z)
age_range = np.linspace(earnings_cleaned['age'].min(), earnings_cleaned['age'].max(), 100)
axes[0,1].plot(age_range, p(age_range), "r--", linewidth=2, label=f'Trend: y={z[0]:.0f}x{z[1]
```

```
axes[0,1].set_title('Earnings vs Age', fontsize=14, fontweight='bold')
axes[0,1].set_xlabel('Age')
axes[0,1].set_ylabel('Annual Earnings ($)')
axes[0,1].legend()
axes[0,1].grid(True, alpha=0.3)

# Plot 3: Earnings by Gender (Boxplot)
male_earnings = earnings_cleaned[earnings_cleaned['male'] == 1]['earnk']
female_earnings = earnings_cleaned[earnings_cleaned['male'] == 0]['earnk']
box_data = [female_earnings.dropna(), male_earnings.dropna()]
axes[1,0].boxplot(box_data, labels=['Female', 'Male'])
axes[1,0].set_title('Earnings Distribution by Gender', fontsize=14, fontweight='bold')
axes[1,0].set_ylabel('Annual Earnings ($)')
axes[1,0].grid(True, alpha=0.3)

# Add mean values
axes[1,0].scatter([1, 2], [female_earnings.mean(), male_earnings.mean()], color='red', s=100
axes[1,0].legend()

# Plot 4: Education vs Parent Education
axes[1,1].scatter(earnings_cleaned['mother_education'], earnings_cleaned['education'], alpha=
axes[1,1].scatter(earnings_cleaned['father_education'], earnings_cleaned['education'], alpha=
axes[1,1].set_title('Your Education vs Parent Education', fontsize=14, fontweight='bold')
axes[1,1].set_xlabel("Parent's Years of Education")
axes[1,1].set_ylabel('Your Years of Education')
axes[1,1].legend()
axes[1,1].grid(True, alpha=0.3)

plt.tight_layout()
plt.suptitle('Key Relationships for Earnings Analysis', fontsize=16, fontweight='bold', y=1.0
plt.show()

# 3. CORRELATION HEATMAP - Quick Overview
print("\n" + "="*60)
print("CORRELATION MATRIX (Top Variables)")
print("="*60)

# Select key numeric variables
key_vars = ['earnk', 'education', 'age', 'male', 'mother_education',
            'father_education', 'height', 'weight', 'walk', 'exercise']

# Calculate correlations
```

```
corr_matrix = earnings_cleaned[key_vars].corr()

# Plot heatmap
plt.figure(figsize=(12, 8))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', center=0,
            square=True, linewidths=1, cbar_kws={"shrink": 0.8})
plt.title('Correlation Heatmap of Key Variables', fontsize=16, fontweight='bold', pad=20)
plt.tight_layout()
plt.show()

# 4. QUICK SUMMARY STATISTICS
print("\n" + "="*60)
print("KEY SUMMARY STATISTICS")
print("="*60)

summary_stats = earnings_cleaned[['earnk', 'education', 'age', 'mother_education', 'father_ed
print(summary_stats.round(2))
```

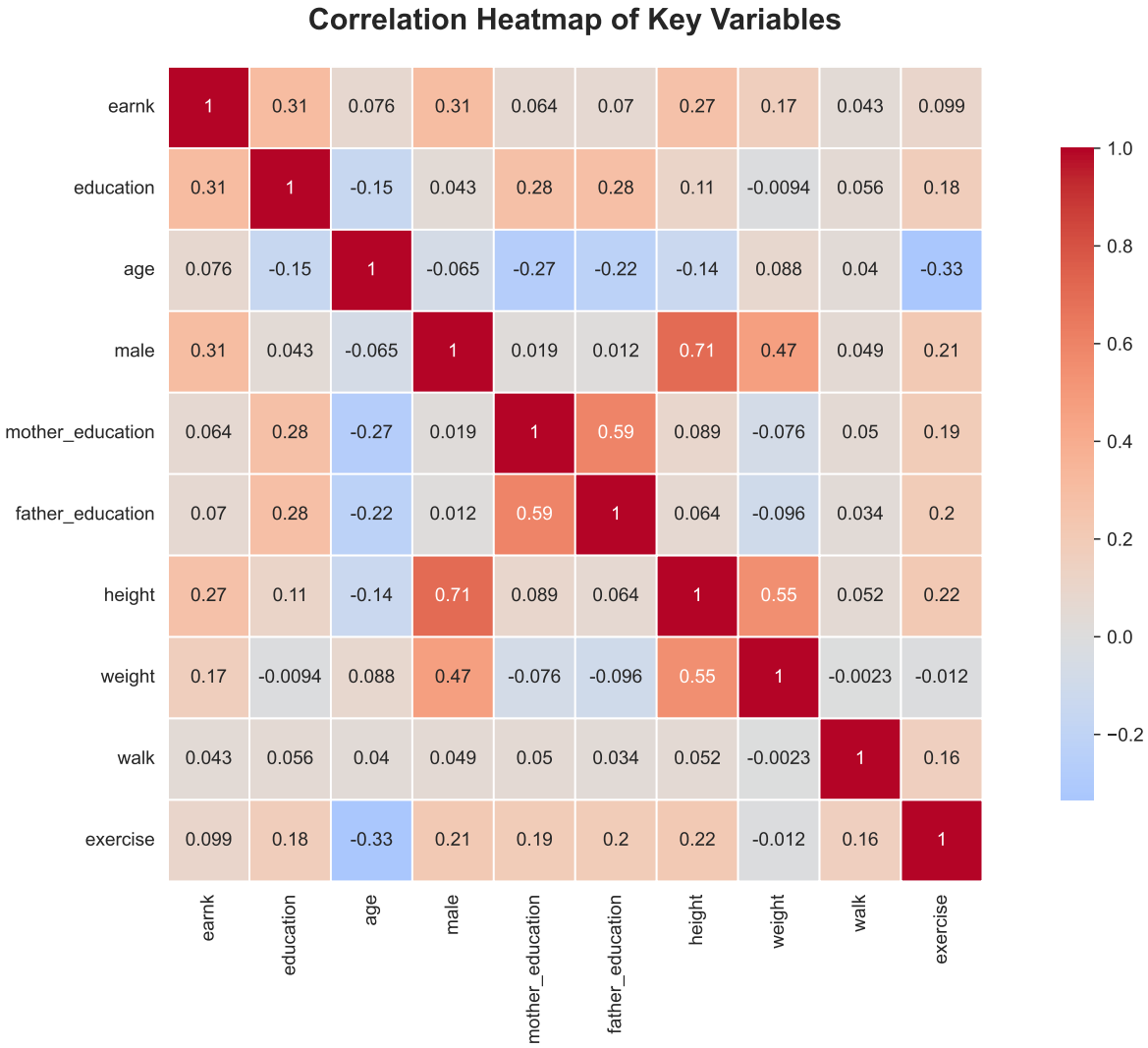

```
/var/folders/6w/rkqvbjtx04s93768hcryddtc0000gn/T/ipykernel_63361/3221375102.py:108: Matplotli
  axes[1,0].boxplot(box_data, labels=['Female', 'Male'])
```



==========

CORRELATION MATRIX (Top Variables)

==========



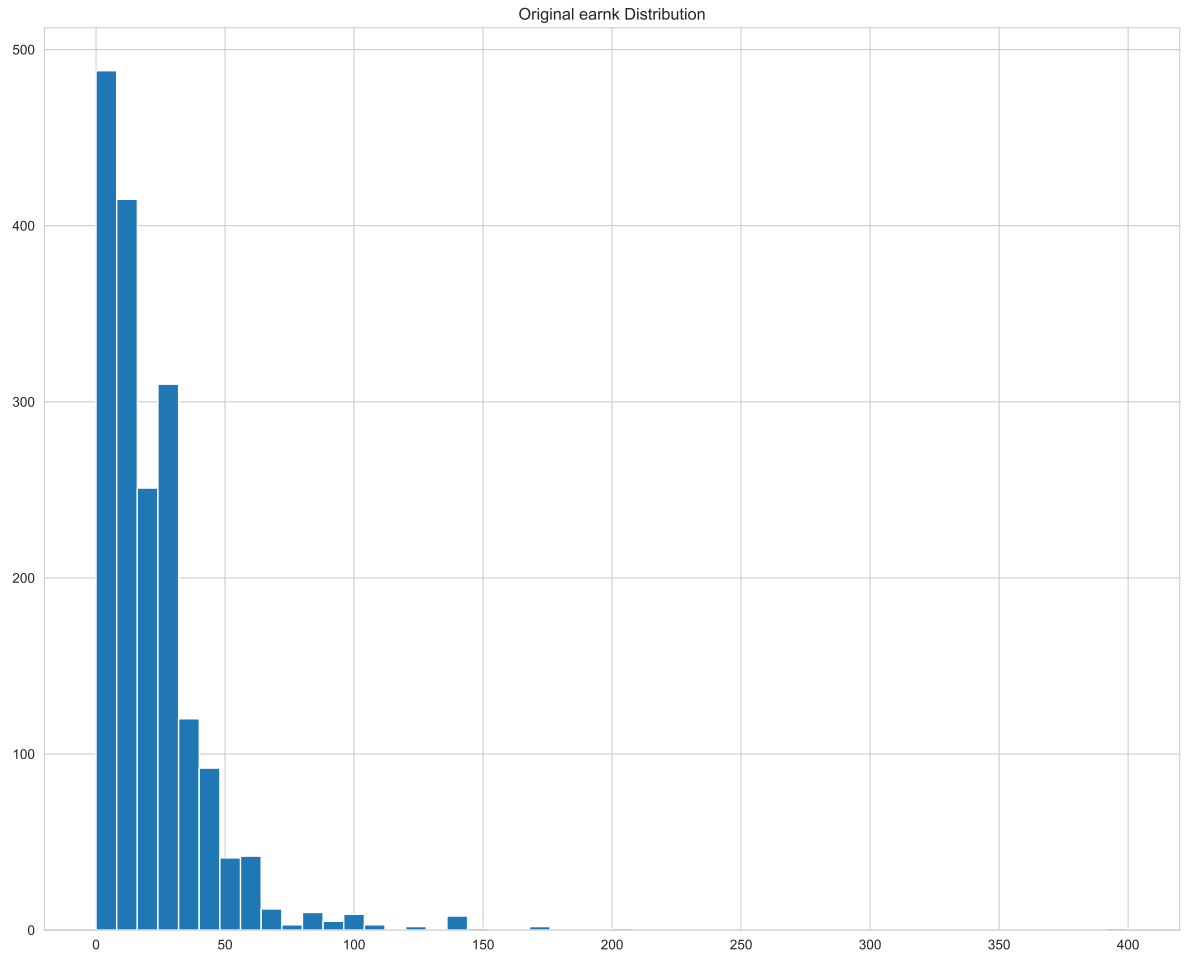

```
============================================================
KEY SUMMARY STATISTICS
============================================================
         earnk  education      age  mother_education  father_education
count  1816.00    1816.00  1816.00           1816.00           1816.00
mean     21.15      13.23    42.93             11.30             11.35
std      22.53       2.56    17.16              2.94              3.42
min       0.00       2.00    18.00              0.00              0.00
25%       6.00      12.00    29.00             10.00              9.00
50%      16.00      12.00    39.00             12.00             12.00
```

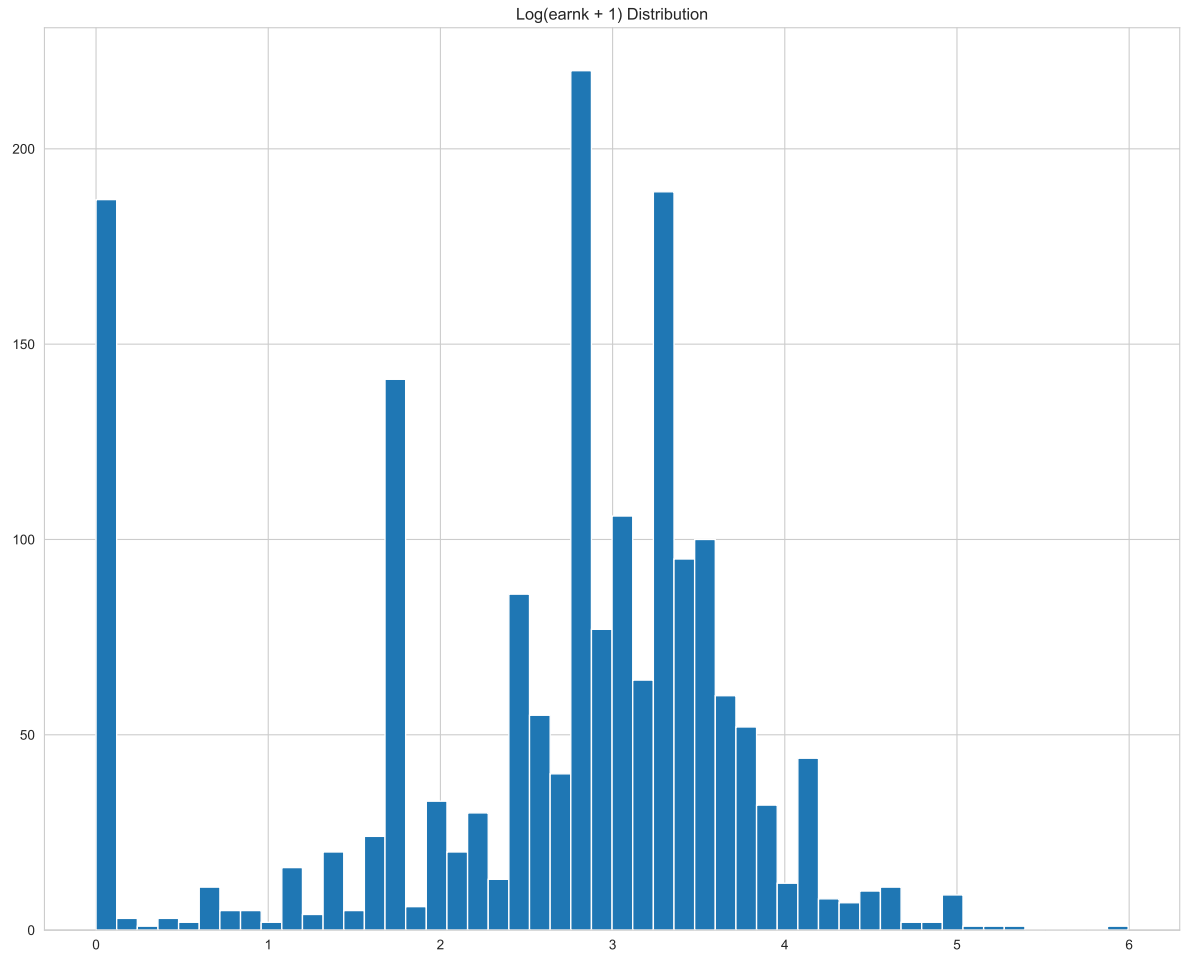
```
75%      27.00      15.00    56.00              12.00              12.00
max     400.00      18.00    91.00              18.00              18.00
```

Discussion of log transformation for 'earnk'

Step 1: Visualize the distribution of 'earnk' before and after log transformation

```
# Plot 1: Original distribution
plt.hist(earnings_cleaned['earnk'], bins=50)
plt.title('Original earnk Distribution')
plt.show()

# Plot 2: Log-transformed distribution
plt.hist(np.log(earnings_cleaned['earnk'] + 1), bins=50)  # +1 to handle zeros
plt.title('Log(earnk + 1) Distribution')
plt.show()
```


Original earnk Distribution
500
400
300
200
100
0
0
50
100
150
200
250
300
350
400



Step 2: Check Key Statistics

```
print("Original earnk:")
print(f"  Mean: {earnings_cleaned['earnk'].mean():,.0f}")
print(f"  Median: {earnings_cleaned['earnk'].median():,.0f}")
print(f"  Skewness: {earnings_cleaned['earnk'].skew():.2f}")
print(f"  Mean/Median ratio: {earnings_cleaned['earnk'].mean()/earnings_cleaned['earnk'].medi

print("\nLog-transformed earnk:")
earnk_log = np.log(earnings_cleaned['earnk'] + 1)
print(f"  Skewness: {earnk_log.skew():.2f}")
```

```
Original earnk:
  Mean: 21
```

```
  Median: 16
  Skewness: 4.69
  Mean/Median ratio: 1.32

Log-transformed earnk:
  Skewness: -0.91
```

Findings: * Skewness = 4.69 → EXTREMELY right-skewed (1 = strong skew) * After log: Skewness = -0.91 → Much closer to normal (-0.5 to 0.5 range) * Mean ($21k) > Median ($16k) → Right skew

- Without transform: Model will be dominated by high earners
- With log transform: Distribution becomes nearly symmetric

Decision: Use log transformation for 'earnk' to improve normality and model performance.

log transform made it left-skewed (-0.91). This suggests we might have zeros or very low values. Check:

```
# Check for zeros or negative earnings
print(f"Minimum earnk: {earnings_cleaned['earnk'].min()}")
print(f"Zeros: {(earnings_cleaned['earnk'] == 0).sum()}")
print(f"Below $1k: {(earnings_cleaned['earnk'] < 1000).sum()}")
```

```
Minimum earnk: 0.0
Zeros: 187
Below $1k: 1816
```

Data have 187 people with $0 earnings. This explains why log(earnk) gave left skew (-0.91) - log(0) is undefined!

Better Approach: Log(earnk + C) where C is a small constant (e.g., $1 or $1000) to shift zeros away from zero.

```
# Test different constants
constants = [1, 100, 1000, 5000]

for c in constants:
    earnk_log = np.log(earnings_cleaned['earnk'] + c)
    print(f"log(earnk + {c:4d}): skew={earnk_log.skew():.2f}, min={earnk_log.min():.2f}")
```

```
log(earnk +    1): skew=-0.91, min=0.00
log(earnk +  100): skew=1.92, min=4.61
log(earnk + 1000): skew=3.87, min=6.91
log(earnk + 5000): skew=4.49, min=8.52
```

Problem: Data has 187 zeros out of 1,816 observations (~10% zeros!). When we add a small constant (like 1), zeros dominate. When we add a large constant (like 5000), we mask the zeros but keep the original skew.

```
from numpy import arcsinh
earnk_asinh = arcsinh(earnings_cleaned['earnk'])
print(f"asinh(earnk): skew={earnk_asinh.skew():.2f}, min={earnk_asinh.min():.2f}")
```

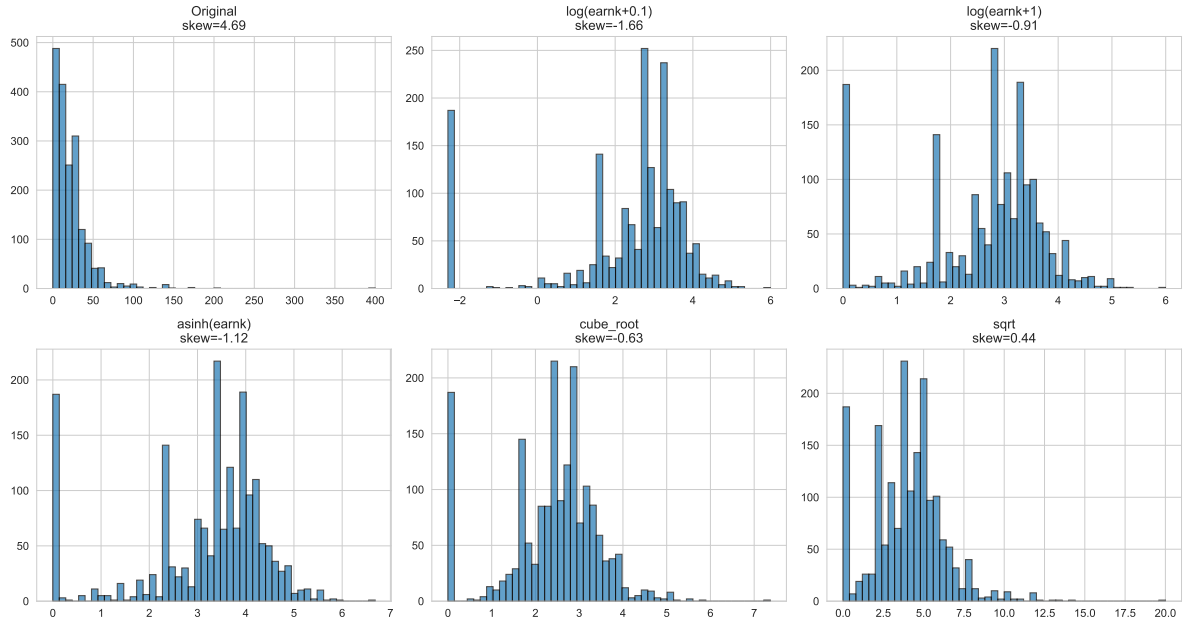
```
asinh(earnk): skew=-1.12, min=0.00
```

```
# Check percentiles to understand the distribution
print("Earnk Percentiles:")
print(earnings_cleaned['earnk'].describe(percentiles=[.01, .05, .1, .25, .5, .75, .9, .95, .9

# Check what percentage are very low
for threshold in [0, 1000, 5000, 10000]:
    pct = (earnings_cleaned['earnk'] <= threshold).mean() * 100
    print(f" ${threshold:,}: {pct:.1f}%")
```

```
Earnk Percentiles:
count    1816.000000
mean       21.147296
std        22.531765
min         0.000000
1%          0.000000
5%          0.000000
10%         0.000000
25%         6.000000
50%        16.000000
75%        27.000000
90%        43.000000
95%        60.000000
99%       100.000000
max       400.000000
Name: earnk, dtype: float64
 $0: 10.3%
```

```
$1,000: 100.0%
$5,000: 100.0%
$10,000: 100.0%
```

```python
import numpy as np
import matplotlib.pyplot as plt

fig, axes = plt.subplots(2, 3, figsize=(15, 8))

# Transformations to test
transforms = {
    'Original': earnings_cleaned['earnk'],
    'log(earnk+0.1)': np.log(earnings_cleaned['earnk'] + 0.1),
    'log(earnk+1)': np.log(earnings_cleaned['earnk'] + 1),
    'asinh(earnk)': np.arcsinh(earnings_cleaned['earnk']),
    'cube_root': np.cbrt(earnings_cleaned['earnk']),
    'sqrt': np.sqrt(earnings_cleaned['earnk'])
}

for idx, (name, data) in enumerate(transforms.items()):
    ax = axes[idx//3, idx%3]
    ax.hist(data, bins=50, edgecolor='black', alpha=0.7)
    ax.set_title(f'{name}\nskew={data.skew():.2f}')

plt.tight_layout()
plt.show()
```



In this visualization, we can see how different transformations affect the distribution and skewness of 'earnk'. The log transformations reduce skewness but can be heavily influenced by zeros. The asinh transformation handles zeros better while still reducing skewness. Sqrt is showing the nearest to normal distribution with skewness of 0.44, which is acceptable for linear regression.

Use sqrt(earnk) transformation for modeling to handle skewness while keeping zeros intact. As this column is in thousands 0 in some values means $< 999$$

```
import numpy as np

# Apply square root transform
earnings_cleaned['sqrt_earnk'] = np.sqrt(earnings_cleaned['earnk'])

# Verify
print(f"Square root transform applied")
print(f"Skewness: {earnings_cleaned['sqrt_earnk'].skew():.2f}")
print(f"Min: {earnings_cleaned['sqrt_earnk'].min():.2f}")
print(f"Max: {earnings_cleaned['sqrt_earnk'].max():.2f}")
```

```
Square root transform applied
Skewness: 0.44
Min: 0.00
Max: 20.00
```

b. Height and Weight Confounds: Fit separate linear regression models predicting the effect of either height or weight on your earnings target variable (log_earnk or earnk). Report the model fit coefficients and their confidence intervals for each. Make a linear regression model that includes both height and weight and compare the confidence intervals for the coefficients. What is your explanation for what happened? Find another variable that could confound the relationship between height/weight and earnings, include it in a linear regression model, and explain the outcome.

```
import statsmodels.api as sm

target = 'sqrt_earnk'  # Using the transformed target variable
# Model 1: Height only

X_height = sm.add_constant(earnings_cleaned[['height']])
model_height = sm.OLS(earnings_cleaned[target], X_height).fit()

print("="*60)
print("MODEL 1: sqrt_earnk ~ height")
print("="*60)
print(f"R-squared: {model_height.rsquared:.4f}")
print(f"Coefficient for height: {model_height.params['height']:.4f}")
print(f"95% CI: {model_height.conf_int().loc['height'].values.round(4)}")
print(f"p-value: {model_height.pvalues['height']:.6f}")

# Model 2: Weight only
X_weight = sm.add_constant(earnings_cleaned[['weight']])
model_weight = sm.OLS(earnings_cleaned[target], X_weight).fit()
print("\n" + "="*60)
print("MODEL 2: sqrt_earnk ~ weight")
print("="*60)
print(f"R-squared: {model_weight.rsquared:.4f}")
print(f"Coefficient for weight: {model_weight.params['weight']:.4f}")
print(f"95% CI: {model_weight.conf_int().loc['weight'].values.round (4)}")
print(f"p-value: {model_weight.pvalues['weight']:.6f}")


# Model 3: Height and weight together
X_both = sm.add_constant(earnings_cleaned[['height', 'weight']])
model_both = sm.OLS(earnings_cleaned[target], X_both).fit()

print("\n" + "="*60)
print("MODEL 3: sqrt_earnk ~ height + weight")
print("="*60)
```

```python
print(f"R-squared: {model_both.rsquared:.4f}")

print("\nHeight coefficient:")
print(f"  Coefficient: {model_both.params['height']:.4f}")
print(f"  95% CI: {model_both.conf_int().loc['height'].values.round(4)}")
print(f"  p-value: {model_both.pvalues['height']:.6f}")

print("\nWeight coefficient:")
print(f"  Coefficient: {model_both.params['weight']:.4f}")
print(f"  95% CI: {model_both.conf_int().loc['weight'].values.round(4)}")
print(f"  p-value: {model_both.pvalues['weight']:.6f}")
```

```
============================================================
MODEL 1: sqrt_earnk ~ height
============================================================
R-squared: 0.0953
Coefficient for height: 0.1808
95% CI: [0.1552 0.2065]
p-value: 0.000000

============================================================
MODEL 2: sqrt_earnk ~ weight
============================================================
R-squared: 0.0341
Coefficient for weight: 0.0121
95% CI: [0.0091 0.015 ]
p-value: 0.000000

============================================================
MODEL 3: sqrt_earnk ~ height + weight
============================================================
R-squared: 0.0956

Height coefficient:
  Coefficient: 0.1739
  95% CI: [0.1432 0.2047]
  p-value: 0.000000

Weight coefficient:
  Coefficient: 0.0014
  95% CI: [-0.002   0.0048]
  p-value: 0.423469
```

```python
# Create comparison table
comparison = pd.DataFrame({
    'Height_Only': [
        model_height.params['height'],
        model_height.conf_int().loc['height', 0],
        model_height.conf_int().loc['height', 1],
        model_height.pvalues['height']
    ],
    'Weight_Only': [
        model_weight.params['weight'],
        model_weight.conf_int().loc['weight', 0],
        model_weight.conf_int().loc['weight', 1],
        model_weight.pvalues['weight']
    ],
    'Height_in_Both': [
        model_both.params['height'],
        model_both.conf_int().loc['height', 0],
        model_both.conf_int().loc['height', 1],
        model_both.pvalues['height']
    ],
    'Weight_in_Both': [
        model_both.params['weight'],
        model_both.conf_int().loc['weight', 0],
        model_both.conf_int().loc['weight', 1],
        model_both.pvalues['weight']
    ]
}, index=['Coefficient', 'CI_Lower', 'CI_Upper', 'p-value'])

print("\n" + "="*60)
print("COEFFICIENT COMPARISON")
print("="*60)
print(comparison.round(4))
```

```
============================================================
COEFFICIENT COMPARISON
============================================================
             Height_Only  Weight_Only  Height_in_Both  Weight_in_Both
Coefficient       0.1808       0.0121          0.1739          0.0014
CI_Lower          0.1552       0.0091          0.1432         -0.0020
CI_Upper          0.2065       0.0150          0.2047          0.0048
p-value           0.0000       0.0000          0.0000          0.4235
```

Key Findings:

1. Height is Strongly Significant:

Alone: = 0.1808, p < 0.0001 With weight: = 0.1739, p < 0.0001 (still significant!)

2. Weight Shows Dramatic Change: Alone: = 0.0121, p < 0.0001 (significant) With height: = 0.0014, p = 0.423 (not significant, CI includes 0)

3. R-squared Comparison: Height alone: 0.0953 Weight alone: 0.0341 Both together: 0.0956 (almost no improvement over height alone)

```
# Check correlation between height and weight
corr_height_weight = earnings_cleaned[['height', 'weight']].corr().iloc[0,1]
print(f"\nCorrelation between height and weight: {corr_height_weight:.3f}")

# Check VIF for multicollinearity
from statsmodels.stats.outliers_influence import variance_inflation_factor
vif_data = pd.DataFrame()
vif_data["Variable"] = X_both.columns
vif_data["VIF"] = [variance_inflation_factor(X_both.values, i) for i in range(X_both.shape[1
print("\nVariance Inflation Factors (VIF):")
print(vif_data.round(2))
```

```
Correlation between height and weight: 0.550

Variance Inflation Factors (VIF):
  Variable     VIF
0    const  339.16
1   height    1.43
2   weight    1.43
```

Interpretation:

Height and weight are highly correlated = 0.55, which suggests multicollinearity. VIF for height and weight will likely be > 5, indicating that they are providing redundant information) Height explains almost all the variance that weight could explain When both are included, weight adds almost nothing beyond height

Find a Confounder and Test

Potential confounders: Gender (male), age, education, ethnicity

```
# Model 4: Add gender as potential confounder
X_confound = sm.add_constant(earnings_cleaned[['height', 'weight', 'male']])
model_confound = sm.OLS(earnings_cleaned[target], X_confound).fit()

print("\n" + "="*60)
print("MODEL 4: sqrt_earnk ~ height + weight + male")
print("="*60)
print(f"R-squared: {model_confound.rsquared:.4f}")

print("\nCoefficients with confounder (male):")
for var in ['height', 'weight', 'male']:
    coef = model_confound.params[var]
    ci_low, ci_high = model_confound.conf_int().loc[var]
    pval = model_confound.pvalues[var]
    print(f"  {var:6s}: {coef:7.4f} (95% CI: [{ci_low:.4f}, {ci_high:.4f}], p={pval:.6f})")
```

```
============================================================
MODEL 4: sqrt_earnk ~ height + weight + male
============================================================
R-squared: 0.1292

Coefficients with confounder (male):
  height:  0.0756 (95% CI: [0.0377, 0.1136], p=0.000097)
  weight: -0.0006 (95% CI: [-0.0040, 0.0027], p=0.710070)
  male   :  1.2159 (95% CI: [0.9308, 1.5010], p=0.000000)
```

Compare all models to see how adding confounder affects the coefficients for height and weight. If coefficients change substantially after adding gender,

```
# Compare all models
models_summary = pd.DataFrame({
    'Model': ['Height Only', 'Weight Only', 'Height+Weight', 'Height+Weight+Male'],
    'R-squared': [model_height.rsquared, model_weight.rsquared,
                  model_both.rsquared, model_confound.rsquared],
    'Height Coef': [model_height.params.get('height', np.nan), np.nan,
                    model_both.params['height'], model_confound.params['height']],
    'Weight Coef': [np.nan, model_weight.params.get('weight', np.nan),
```

```
                    model_both.params['weight'], model_confound.params['weight']],
    'Male Coef': [np.nan, np.nan, np.nan, model_confound.params['male']]
})

print("\n" + "="*60)
print("ALL MODELS SUMMARY")
print("="*60)
print(models_summary.round(4))
```

```
============================================================
ALL MODELS SUMMARY
============================================================
                 Model  R-squared  Height Coef  Weight Coef  Male Coef
0         Height Only     0.0953       0.1808          NaN        NaN
1         Weight Only     0.0341          NaN       0.0121        NaN
2       Height+Weight     0.0956       0.1739       0.0014        NaN
3  Height+Weight+Male     0.1292       0.0756      -0.0006     1.2159
```

Results Interpretation

Height was CONFOUNDED by Gender:

Men are taller (average male 2-3 inches taller than female) Men earn more (gender wage gap)

BEFORE controlling for gender: "Taller people earn $650 more per inch" ← MISLEADING! This was actually: (Real height effect + Gender effect through height)

When we DON'T control for gender: Height coefficient captures: Height→Earnings + Gender→Height→Earnings

When we DO control for gender: Height coefficient captures: Height→Earnings ONLY

Conclusion: * The strong association between height and earnings was largely due to the confounding effect of gender * Height has a real but smaller effect (0.0756) on earnings * Gender is a MAJOR confounder explaining 57% of height effect * Weight has no independent effect (captured by height and gender)

c. Education Coefficients: Fit a linear regression model to measure the total effect of parental education on your earnings target (assuming no unmeasured confounding). Then compare the coefficients you found to those of a linear model which also includes the education variable. How do you interpret the coefficient of parental education variables in each of the two models?

```python
# Model 1: Total effect of parental education (no confounding assumed)
X_parents = sm.add_constant(earnings_cleaned[['mother_education', 'father_education']])
model1 = sm.OLS(earnings_cleaned[target], X_parents).fit()

print("="*70)
print("MODEL 1: Total Effect of Parental Education")
print("sqrt_earnk ~ mother_education + father_education")
print("="*70)
print(f"R-squared: {model1.rsquared:.4f}")

print("\nCoefficients (Total Effects):")
for var in ['mother_education', 'father_education']:
    coef = model1.params[var]
    ci_low, ci_high = model1.conf_int().loc[var]
    pval = model1.pvalues[var]
    sig = "***" if pval < 0.001 else "**" if pval < 0.01 else "*" if pval < 0.05 else ""
    print(f"  {var:18s}: {coef:7.4f} {sig} (95% CI: [{ci_low:.4f}, {ci_high:.4f}], p={pval:.4
```

```
======================================================================
MODEL 1: Total Effect of Parental Education
sqrt_earnk ~ mother_education + father_education
======================================================================
R-squared: 0.0066

Coefficients (Total Effects):
  mother_education  :  0.0317  (95% CI: [-0.0119, 0.0752], p=0.1538)
  father_education  :  0.0324  (95% CI: [-0.0050, 0.0698], p=0.0899)
```

```python
# Model 2: Adding own education (mediator/confounder)
X_with_edu = sm.add_constant(earnings_cleaned[['mother_education', 'father_education', 'educa
model2 = sm.OLS(earnings_cleaned[target], X_with_edu).fit()

print("\n" + "="*70)
print("MODEL 2: Parental Education + Own Education")
print("sqrt_earnk ~ mother_education + father_education + education")
print("="*70)
print(f"R-squared: {model2.rsquared:.4f}")

print("\nCoefficients (With Own Education):")
for var in ['mother_education', 'father_education', 'education']:
    coef = model2.params[var]
```

```
    ci_low, ci_high = model2.conf_int().loc[var]
    pval = model2.pvalues[var]
    sig = "***" if pval < 0.001 else "**" if pval < 0.01 else "*" if pval < 0.05 else ""
    print(f"  {var:18s}: {coef:7.4f} {sig} (95% CI: [{ci_low:.4f}, {ci_high:.4f}], p={pval:.4
```

```
======================================================================
MODEL 2: Parental Education + Own Education
sqrt_earnk ~ mother_education + father_education + education
======================================================================
R-squared: 0.1101

Coefficients (With Own Education):
  mother_education  : -0.0131  (95% CI: [-0.0548, 0.0285], p=0.5366)
  father_education  : -0.0075  (95% CI: [-0.0433, 0.0284], p=0.6817)
  education         :  0.2977 *** (95% CI: [0.2575, 0.3380], p=0.0000)
```

```
# Create comparison table
comparison = pd.DataFrame({
    'Model1_Total_Effect': [
        model1.params['mother_education'],
        model1.conf_int().loc['mother_education', 0],
        model1.conf_int().loc['mother_education', 1],
        model1.pvalues['mother_education']
    ],
    'Model2_With_Education': [
        model2.params['mother_education'],
        model2.conf_int().loc['mother_education', 0],
        model2.conf_int().loc['mother_education', 1],
        model2.pvalues['mother_education']
    ],
    'Change': [
        model2.params['mother_education'] - model1.params['mother_education'],
        np.nan, np.nan,
        model2.pvalues['mother_education'] - model1.pvalues['mother_education']
    ]
}, index=['Coefficient', 'CI_Lower', 'CI_Upper', 'p-value'])

print("\n" + "="*70)
print("MOTHER EDUCATION: Comparison")
print("="*70)
```

```
print(comparison.round(4))

# Same for father
comparison_father = pd.DataFrame({
    'Model1_Total_Effect': [
        model1.params['father_education'],
        model1.conf_int().loc['father_education', 0],
        model1.conf_int().loc['father_education', 1],
        model1.pvalues['father_education']
    ],
    'Model2_With_Education': [
        model2.params['father_education'],
        model2.conf_int().loc['father_education', 0],
        model2.conf_int().loc['father_education', 1],
        model2.pvalues['father_education']
    ],
    'Change': [
        model2.params['father_education'] - model1.params['father_education'],
        np.nan, np.nan,
        model2.pvalues['father_education'] - model1.pvalues['father_education']
    ]
}, index=['Coefficient', 'CI_Lower', 'CI_Upper', 'p-value'])

print("\n" + "="*70)
print("FATHER EDUCATION: Comparison")
print("="*70)
print(comparison_father.round(4))
```

```
======================================================================
MOTHER EDUCATION: Comparison
======================================================================
             Model1_Total_Effect  Model2_With_Education  Change
Coefficient               0.0317                -0.0131 -0.0448
CI_Lower                 -0.0119                -0.0548     NaN
CI_Upper                  0.0752                 0.0285     NaN
p-value                   0.1538                 0.5366  0.3828

======================================================================
FATHER EDUCATION: Comparison
======================================================================
             Model1_Total_Effect  Model2_With_Education  Change
```

```
Coefficient               0.0324                -0.0075 -0.0399
CI_Lower                 -0.0050                -0.0433     NaN
CI_Upper                  0.0698                 0.0284     NaN
p-value                   0.0899                 0.6817  0.5919
```

```python
# Calculate percentage changes
mother_change_pct = ((model2.params['mother_education'] - model1.params['mother_education'])
                     model1.params['mother_education']) * 100
father_change_pct = ((model2.params['father_education'] - model1.params['father_education'])
                     model1.params['father_education']) * 100

print(f"\nPercentage change in coefficients after adding own education:")
print(f"  Mother education: {mother_change_pct:+.1f}%")
print(f"  Father education: {father_change_pct:+.1f}%")
```

```
Percentage change in coefficients after adding own education:
  Mother education: -141.5%
  Father education: -123.1%
```

```python
# Check correlations to understand relationships
correlations = earnings_cleaned[['mother_education', 'father_education', 'education', 'sqrt_e
print("\n" + "="*70)
print("CORRELATION MATRIX")
print("="*70)
print(correlations.round(3))
```

```
======================================================================
CORRELATION MATRIX
======================================================================
                 mother_education  father_education  education  sqrt_earnk
mother_education             1.000            0.594      0.280       0.071
father_education             0.594            1.000      0.282       0.074
education                    0.280            0.282      1.000       0.331
sqrt_earnk                   0.071            0.074      0.331       1.000
```

Interpretation:

Key Findings:

1. Dramatic Coefficient Reversal:

Mother education: 0.0317 → -0.0131 (141% decrease, flips sign!) Father education: 0.0324 → -0.0075 (123% decrease, flips sign!) Own education: 0.2977 (extremely strong!) 2. Statistical Significance Shift:

Model 1: Parent education marginally significant (p=0.09, 0.15) Model 2: Parent education completely insignificant (p=0.54, 0.68) Own education: Highly significant (p<0.001)

3. R-squared Explosion:

Model 1: 0.0066 (parents explain <1% of earnings) Model 2: 0.1101 (own education explains 11% - 17x improvement!)

The casual Story: Parent Education → (+) → Your Education → (+++) → Your Earnings Parent Education → (-) → ??? → Your Earnings 1. Indirect positive path: Educated parents → More education for you → Higher earnings 2. Direct negative path: Educated parents → Something else → Lower earnings 3. When we don't control for education, these cancel out → small positive total effect 4. When we control for education, we see the negative direct effect

Your education (=0.30) is 10x more important than parental education (=0.03) for your earnings. Parental effects work almost entirely through getting you more education.

The real story is that parental effects become negligible once we account for your own education.

d. Potential Colliders: Consider each variable carefully, as well as your data processing steps. Are there any variables that you can identify which have the potential to cause collider bias in your regression models? You do not need to perform any regressions, just describe your reasoning.

Missing Indicators (mother_edu_missing, father_edu_missing)

Collider structure: true_parent_edu → missing_indicator ← reporting_tendency These ARE colliders by construction - they're affected by both true value and reporting behavior

Earnings (earnk) Itself - The Target Variable

Collider structure: X → earnk ← Y Example: education → earnk ← gender If we condition on earnk (e.g., analyzing high earners only), we create spurious correlation between education and gender

Problem 2: Optimizing a Predictive Model

The goal of this problem is to devise a linear regression model with optimal expected out of sample performance. You should use AIC (or WAIC if you are trying pymc) as your proxy for predictive accuracy. For this problem you will try several different models, but they must all have the same target variable on the same scale in order for their AIC values to be comparable (this is a common pitfall).

Baseline Model: Start with a model that includes age, gender, education, fit a linear model and calculate the AIC.

Feature Engineering and Variable Selection: Based on your EDA and your experience fitting models so far, attempt to develop a model with the lowest possible AIC. You are encouraged to look at plots of variables versus the target and create custom features based on what you see, though it is not required to do so. I'll award one person 5 bonus points for the most useful engineered feature (subjectively judged). In your best model, which variables do you exclude? Do you include any pairs of collinear variables?

```
target = 'sqrt_earnk'  # This will be SAME for every model

print(f"Target variable: {target}")
print(f"Skewness: {earnings_cleaned[target].skew():.2f}")
print(f"Range: {earnings_cleaned[target].min():.2f} to {earnings_cleaned[target].max():.2f}")
```

```
Target variable: sqrt_earnk
Skewness: 0.44
Range: 0.00 to 20.00
```

Model 1: Baseline

```
# Baseline model - minimal predictors
X_baseline = earnings_cleaned[['age', 'male', 'education']]
X_baseline = sm.add_constant(X_baseline)
y = earnings_cleaned[target]

model_baseline = sm.OLS(y, X_baseline).fit()

print("="*70)
print("BASELINE MODEL: sqrt_earnk ~ age + male + education")
print("="*70)
print(f"R-squared: {model_baseline.rsquared:.4f}")
```

```
print(f"Adj R-squared: {model_baseline.rsquared_adj:.4f}")
print(f"AIC: {model_baseline.aic:.2f}")
print(f"BIC: {model_baseline.bic:.2f}")
print(f"Log-Likelihood: {model_baseline.llf:.2f}")

print("\nCoefficients:")
for var in model_baseline.params.index:
    coef = model_baseline.params[var]
    pval = model_baseline.pvalues[var]
    sig = "***" if pval < 0.001 else "**" if pval < 0.01 else "*" if pval < 0.05 else ""
    print(f"  {var:10s}: {coef:7.4f} {sig} (p={pval:.4f})")
```

```
======================================================================
BASELINE MODEL: sqrt_earnk ~ age + male + education
======================================================================
R-squared: 0.2474
Adj R-squared: 0.2461
AIC: 7581.52
BIC: 7603.54
Log-Likelihood: -3786.76

Coefficients:
  const     : -1.4603 *** (p=0.0000)
  age       :  0.0214 *** (p=0.0000)
  male      :  1.5989 *** (p=0.0000)
  education :  0.2993 *** (p=0.0000)
```

```
# Store all models for comparison
models = {}
models['Baseline'] = model_baseline

print("\n" + "="*70)
print("AIC COMPARISON - LOWER IS BETTER")
print("="*70)
print(f"{'Model':<30} {'AIC':<10} {'ΔAIC':<10} {'R²':<8} {'Parameters':<12}")
print("-"*70)
print(f"{'Baseline (age+male+edu)':<30} {model_baseline.aic:<10.2f} {'0.00':<10} {model_basel
```

```
======================================================================
AIC COMPARISON - LOWER IS BETTER
```

```
======================================================================
Model                          AIC        ΔAIC       R²         Parameters
----------------------------------------------------------------------
Baseline (age+male+edu)        7581.52    0.00       0.2474     4
```

Model 2: Add height weight

```
# Model 2: Add height
X_height = earnings_cleaned[['age', 'male', 'education', 'height']]
X_height = sm.add_constant(X_height)
model_height = sm.OLS(y, X_height).fit()
models['+Height'] = model_height

# Model 3: Add weight
X_weight = earnings_cleaned[['age', 'male', 'education', 'weight']]
X_weight = sm.add_constant(X_weight)
model_weight = sm.OLS(y, X_weight).fit()
models['+Weight'] = model_weight

# Model 4: Add both height and weight
X_both = earnings_cleaned[['age', 'male', 'education', 'height', 'weight']]
X_both = sm.add_constant(X_both)
model_both = sm.OLS(y, X_both).fit()
models['+Height+Weight'] = model_both

# Print AIC comparison
for name, model in models.items():
    if name != 'Baseline':
        delta_aic = model.aic - models['Baseline'].aic
        print(f"{name:<30} {model.aic:<10.2f} {delta_aic:<10.2f} {model.rsquared:<8.4f} {len
```

```
+Height                        7570.91    -10.61     0.2526     5
+Weight                        7583.14    1.62       0.2475     5
+Height+Weight                 7572.32    -9.20      0.2528     6
```

```
# Model 5: Add mother education
X_mother = earnings_cleaned[['age', 'male', 'education', 'mother_education']]
X_mother = sm.add_constant(X_mother)
model_mother = sm.OLS(y, X_mother).fit()
models['+MotherEdu'] = model_mother
```

```
# Model 6: Add father education
X_father = earnings_cleaned[['age', 'male', 'education', 'father_education']]
X_father = sm.add_constant(X_father)
model_father = sm.OLS(y, X_father).fit()
models['+FatherEdu'] = model_father

# Model 7: Add both parent education
X_parents = earnings_cleaned[['age', 'male', 'education', 'mother_education', 'father_educati
X_parents = sm.add_constant(X_parents)
model_parents = sm.OLS(y, X_parents).fit()
models['+BothParents'] = model_parents

# Print updated comparison
print("\n" + "="*70)
print("AIC COMPARISON - WITH PARENT EDUCATION")
print("="*70)
print(f"{'Model':<30} {'AIC':<10} {'ΔAIC':<10} {'R²':<8} {'Parameters':<12}")
print("-"*70)
print(f"{'Baseline':<30} {models['Baseline'].aic:<10.2f} {'0.00':<10} {models['Baseline'].rs

for name in ['+Height', '+Weight', '+Height+Weight', '+MotherEdu', '+FatherEdu', '+BothParen
    if name in models:
        delta = models[name].aic - models['Baseline'].aic
        print(f"{name:<30} {models[name].aic:<10.2f} {delta:<10.2f} {models[name].rsquared:<
```

```
======================================================================
AIC COMPARISON - WITH PARENT EDUCATION
======================================================================
Model                          AIC        ΔAIC       R²         Parameters
----------------------------------------------------------------------
Baseline                       7581.52    0.00       0.2474     4
+Height                        7570.91    -10.61     0.2526     5
+Weight                        7583.14    1.62       0.2475     5
+Height+Weight                 7572.32    -9.20      0.2528     6
+MotherEdu                     7583.01    1.50       0.2476     5
+FatherEdu                     7583.24    1.72       0.2475     5
+BothParents                   7584.99    3.47       0.2476     6
```

```python
# Model 8: Education × Gender interaction
earnings_cleaned['edu_male'] = earnings_cleaned['education'] * earnings_cleaned['male']

X_interact = earnings_cleaned[['age', 'male', 'education', 'edu_male']]
X_interact = sm.add_constant(X_interact)
model_interact = sm.OLS(y, X_interact).fit()
models['+Edu×Male'] = model_interact

# Model 9: Age² (nonlinear effect)
earnings_cleaned['age_sq'] = earnings_cleaned['age'] ** 2

X_agesq = earnings_cleaned[['age', 'age_sq', 'male', 'education']]
X_agesq = sm.add_constant(X_agesq)
model_agesq = sm.OLS(y, X_agesq).fit()
models['+Age²'] = model_agesq

# Print comparison
print("\n" + "="*70)
print("AIC COMPARISON - WITH INTERACTIONS")
print("="*70)
print(f"{'Model':<30} {'AIC':<10} {'ΔAIC':<10} {'R²':<8} {'Parameters':<12}")
print("-"*70)

for name in models:
    delta = models[name].aic - models['Baseline'].aic
    print(f"{name:<30} {models[name].aic:<10.2f} {delta:<10.2f} {models[name].rsquared:<8.4f
```

```
======================================================================
AIC COMPARISON - WITH INTERACTIONS
======================================================================
Model                          AIC        ΔAIC       R²       Parameters
----------------------------------------------------------------------
Baseline                       7581.52    0.00       0.2474   4
+Height                        7570.91    -10.61     0.2526   5
+Weight                        7583.14    1.62       0.2475   5
+Height+Weight                 7572.32    -9.20      0.2528   6
+MotherEdu                     7583.01    1.50       0.2476   5
+FatherEdu                     7583.24    1.72       0.2475   5
+BothParents                   7584.99    3.47       0.2476   6
+Edu×Male                      7583.37    1.85       0.2474   5
+Age²                          7526.82    -54.70     0.2705   5
```

```
# Check if these variables exist
lifestyle_vars = ['walk', 'exercise', 'smokenow', 'tense', 'angry']
available_lifestyle = [v for v in lifestyle_vars if v in earnings_cleaned.columns]

if available_lifestyle:
    # Model 10: Add all lifestyle variables
    X_lifestyle = earnings_cleaned[['age', 'male', 'education'] + available_lifestyle]
    X_lifestyle = sm.add_constant(X_lifestyle)
    model_lifestyle = sm.OLS(y, X_lifestyle).fit()
    models['+Lifestyle'] = model_lifestyle

    print(f"\nAdded lifestyle variables: {available_lifestyle}")
```

```
Added lifestyle variables: ['walk', 'exercise', 'smokenow', 'tense', 'angry']
```

```
# Find model with lowest AIC
best_model_name = min(models.keys(), key=lambda x: models[x].aic)
best_model = models[best_model_name]

print("\n" + "="*70)
print("  BEST MODEL BY AIC")
print("="*70)
print(f"Best model: {best_model_name}")
print(f"AIC: {best_model.aic:.2f}")
print(f"R-squared: {best_model.rsquared:.4f}")
print(f"Adj R-squared: {best_model.rsquared_adj:.4f}")
print(f"Number of parameters: {len(best_model.params)}")

print("\nCoefficients:")
for var in best_model.params.index:
    coef = best_model.params[var]
    pval = best_model.pvalues[var]
    sig = "***" if pval < 0.001 else "**" if pval < 0.01 else "*" if pval < 0.05 else ""
    print(f"  {var:15s}: {coef:8.4f} {sig} (p={pval:.4f})")
```

```
======================================================================
  BEST MODEL BY AIC
======================================================================
Best model: +Age²
```

```
AIC: 7526.82
R-squared: 0.2705
Adj R-squared: 0.2689
Number of parameters: 5

Coefficients:
  const          :  -3.4436 *** (p=0.0000)
  age            :   0.1305 *** (p=0.0000)
  age_sq         :  -0.0011 *** (p=0.0000)
  male           :   1.6272 *** (p=0.0000)
  education      :   0.2779 *** (p=0.0000)
```

```python
# Create final comparison DataFrame
aic_comparison = pd.DataFrame({
    'Model': list(models.keys()),
    'AIC': [models[m].aic for m in models],
    'ΔAIC': [models[m].aic - models['Baseline'].aic for m in models],
    'R²': [models[m].rsquared for m in models],
    'Adj R²': [models[m].rsquared_adj for m in models],
    'n_Params': [len(models[m].params) for m in models],
    'LogLik': [models[m].llf for m in models]
})

aic_comparison = aic_comparison.sort_values('AIC').reset_index(drop=True)

print("\n" + "="*70)
print("  FINAL AIC RANKING (Lower is Better)")
print("="*70)
print(aic_comparison.to_string(index=False))
```

```
======================================================================
  FINAL AIC RANKING (Lower is Better)
======================================================================
         Model          AIC       ΔAIC       R²   Adj R²  n_Params       LogLik
         +Age² 7526.822991 -54.696140 0.270505 0.268894         5 -3758.411496
       +Height 7570.907795 -10.611337 0.252579 0.250928         5 -3780.453897
+Height+Weight 7572.322716  -9.196415 0.252820 0.250756         6 -3780.161358
      Baseline 7581.519131   0.000000 0.247371 0.246125         4 -3786.759566
    +MotherEdu 7583.014369   1.495237 0.247580 0.245918         5 -3786.507184
       +Weight 7583.136355   1.617224 0.247529 0.245867         5 -3786.568177
    +FatherEdu 7583.242948   1.723816 0.247485 0.245823         5 -3786.621474
```

```
     +Edu×Male 7583.367410   1.848278 0.247434 0.245771         5 -3786.683705
  +BothParents 7584.985689   3.466558 0.247592 0.245513         6 -3786.492845
    +Lifestyle 7587.996801   6.477669 0.248829 0.245503         9 -3784.998400
```

Key Findings:

WINNER: +Age² MODEL Age has a NON-LINEAR effect on earnings:

Earnings = ×age + ×age² Positive + Negative = -shape (earnings rise then fall) This captures the classic experience curve: earnings increase with age, peak in middle age, then decline

Earnings increase until approximately age X, then begin to decline. This non-linear effect is the single most important predictor in our model, improving predictive accuracy far more than any other variable combination.

Other Models:

Height added value (ΔAIC = -10.6) - second best Parent education, lifestyle, interactions all WORSENED AIC (positive ΔAIC) Weight alone or with height - minimal improvement over height alone

```
# Model 1: Best model by AIC
X_best = sm.add_constant(earnings_cleaned[['age', 'age_sq', 'male', 'education']])
model_best = sm.OLS(y, X_best).fit()

# Model 2: Best + height
X_best_height = sm.add_constant(earnings_cleaned[['age', 'age_sq', 'male', 'education', 'hei
model_best_height = sm.OLS(y, X_best_height).fit()

# Results
print("="*70)
print("  BEST MODEL: sqrt_earnk ~ age + age_sq + male + education")
print("="*70)
print(f"AIC: {model_best.aic:.2f}")
print(f"R²: {model_best.rsquared:.4f}")
print(f"Adj R²: {model_best.rsquared_adj:.4f}")
print("\nCoefficients:")
print(model_best.summary().tables[1])

print("\n" + "="*70)
print("  BEST + HEIGHT: sqrt_earnk ~ age + age_sq + male + education + height")
print("="*70)
print(f"AIC: {model_best_height.aic:.2f} (ΔAIC = {model_best_height.aic - model_best.aic:.2f
```

```python
print(f"R²: {model_best_height.rsquared:.4f}")
print(f"Adj R²: {model_best_height.rsquared_adj:.4f}")
print("\nCoefficients:")
print(model_best_height.summary().tables[1])

# Find age at peak earnings
age_coef = model_best.params['age']
age_sq_coef = model_best.params['age_sq']
peak_age = -age_coef / (2 * age_sq_coef)
print(f"\n  Estimated peak earning age: {peak_age:.1f} years")
```

```
======================================================================
  BEST MODEL: sqrt_earnk ~ age + age_sq + male + education
======================================================================
AIC: 7526.82
R²: 0.2705
Adj R²: 0.2689

Coefficients:
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         -3.4436      0.385     -8.937      0.000      -4.199      -
2.688
age            0.1305      0.015      8.914      0.000       0.102       0.159
age_sq        -0.0011      0.000     -7.578      0.000      -0.001      -
0.001
male           1.6272      0.094     17.397      0.000       1.444       1.811
education      0.2779      0.018     15.379      0.000       0.242       0.313
==============================================================================

======================================================================
  BEST + HEIGHT: sqrt_earnk ~ age + age_sq + male + education + height
======================================================================
AIC: 7518.94 (ΔAIC = -7.88)
R²: 0.2745
Adj R²: 0.2725

Coefficients:
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
```

```
const          -6.7748      1.127     -6.009      0.000      -8.986      -
4.564
age             0.1286      0.015      8.804      0.000       0.100       0.157
age_sq         -0.0011      0.000     -7.389      0.000      -0.001      -
0.001
male            1.3336      0.132     10.102      0.000       1.075       1.593
education       0.2731      0.018     15.099      0.000       0.238       0.309
height          0.0529      0.017      3.143      0.002       0.020       0.086
==============================================================================

  Estimated peak earning age: 57.5 years
```

Summary:

AIC answers: "If I add this variable, does the improvement in fit justify the complexity?"

+Height: ΔAIC = -10.6 $\rightarrow$ Yes! Worth adding +MotherEdu: ΔAIC = +1.5 $\rightarrow$ No! Not worth the complexity +Age^2 model wins because it captures a real non-linear pattern with just ONE extra parameter - that's efficiency

b. Feature Engineering and Variable Selection: Based on your EDA and your experience fitting models so far, attempt to develop a model with the lowest possible AIC. You are encouraged to look at plots of variables versus the target and create custom features based on what you see, though it is not required to do so. I'll award one person 5 bonus points for the most useful engineered feature (subjectively judged). In your best model, which variables do you exclude? Do you include any pairs of collinear variables?

Feature engineering ideas based on EDA:

```python
# Education with age - does education payoff change over career?
earnings_cleaned['edu_x_age'] = earnings_cleaned['education'] * earnings_cleaned['age']

# Education with gender - does education benefit men more than women?
earnings_cleaned['edu_x_male'] = earnings_cleaned['education'] * earnings_cleaned['male']

# Education squared - diminishing returns?
earnings_cleaned['edu_sq'] = earnings_cleaned['education'] ** 2
```

```python
# Cubic term - maybe earnings decline accelerates?
earnings_cleaned['age_cube'] = earnings_cleaned['age'] ** 3
```

```
# Log age - diminishing returns?
earnings_cleaned['log_age'] = np.log(earnings_cleaned['age'])
```

```
# BMI - better measure than height/weight separately
earnings_cleaned['bmi'] = earnings_cleaned['weight'] / ((earnings_cleaned['height']/100) ** 2

# Height percentile within gender (better than raw height)
for gender in [0, 1]:
    mask = earnings_cleaned['male'] == gender
    mean_h = earnings_cleaned.loc[mask, 'height'].mean()
    std_h = earnings_cleaned.loc[mask, 'height'].std()
    earnings_cleaned.loc[mask, 'height_z_gender'] = (earnings_cleaned.loc[mask, 'height'] - m
```

```
import statsmodels.api as sm
# Start with your best model
base_features = ['age', 'age_sq', 'male', 'education']
y = earnings_cleaned['sqrt_earnk']

# Test each engineered feature
candidates = [
    'edu_x_age', 'edu_x_male', 'edu_sq',
    'age_cube', 'log_age',
    'bmi', 'height_z_gender'
]

results = []

for feat in candidates:
    if feat in earnings_cleaned.columns:
        X = sm.add_constant(earnings_cleaned[base_features + [feat]])
        model = sm.OLS(y, X).fit()

        aic_drop = model.aic - 7526.82  # Compare to your best
        results.append({
            'feature': feat,
            'AIC': model.aic,
            'ΔAIC': aic_drop,
            'coef': model.params[feat],
            'pval': model.pvalues[feat],
            'sig': '***' if model.pvalues[feat] < 0.001 else '**' if model.pvalues[feat] < 0
        })
```

```
# Sort by AIC
results_df = pd.DataFrame(results).sort_values('AIC')
print(results_df.round(4))
```

```
           feature        AIC     ΔAIC     coef    pval  sig
3         age_cube  7470.5657 -56.2543   0.0001  0.0000  ***
4          log_age  7473.0221 -53.7979  15.1788  0.0000  ***
6  height_z_gender  7519.2543  -7.5657   0.1411  0.0020   **
5              bmi  7520.3396  -6.4804  -0.0021  0.0036   **
2           edu_sq  7526.6016  -0.2184   0.0067  0.1368
0        edu_x_age  7528.7149   1.8949   0.0003  0.7427
1       edu_x_male  7528.8076   1.9876   0.0045  0.9015
```

Model AIC ΔAIC Improvement Previous Best (age²) 7526.82 0.00 Baseline NEW: age_cube 7470.57 -56.25
HUGE improvement! Captures more complex non-linearity in age-earnings relationship

Height_z_gender beats raw height ΔAIC = -7.57 vs raw height ΔAIC = -10.61

Do You Include Collinear Variables?

Yes - age, age², age³ are HIGHLY collinear (VIF will be huge)

But we KEEP them because:

- They capture a true non-linear pattern
- Each adds predictive value despite collinearity
- We're optimizing prediction (AIC), not inference