RICHFIELD
richfield.ac.za

Name and Surname: Mabrand Mosala

Student ITS No: 402417806

Qualification: BSc IT   Year of Study: 2025   Semester: 2nd semester

Assignment due date: 15 October 2025   Date submitted: 15 October 2025

QUESTION	EXAMINER MARKS	MODERATOR MARKS	REMARKS

ASSIGNMENT INSTRUCTIONS

Please tick each box to confirm completion.

Use Times New Roman font, size 12, with 1.5 line spacing throughout the document.
Apply Harvard Referencing Style for all citations and references.

For essay-style assignments, please include the following sections:

Table of Contents
Introduction
Main Body (with relevant subheadings)
Conclusion
References

Submit the assignment in PDF format on Moodle.
Use the specified cover page provided.
Include a signed declaration of originality.

DECLARATION OF ORIGINALITY:
I hereby declare that this assignment is my own work and has not been copied from any other source except where due acknowledgment is made. I affirm that all sources used have been properly cited and that this submission complies with the institution's policies on academic integrity and plagiarism.

Student Signature: M.Mosala   Date: 15/10/2025

Table of Contents

QUESTION 1

a) P(A and B) = both / total visitors
= 2000/10 000
= 0.20
=20%

b) P(A or B) = P(A) + P(B) – P(A and B)
= (3000 / 10 000) + (4 500 / 10 000) – (2000 / 10 000)
= 0.30 + 0.45 – 0.20
= 0.55
= 55%

c) P(A|B) = P(A and B) / P(B)

= (2000 / 10 000) / (4500 / 10 000)

= 2000 / 4500

= 0.444

= 44.4%

d) Overall return rate: 4500 / 10 000 = 45%
Return rate for Banner-Clickers: 2000 / 3000 = 66%
Return rate for non-clickers: 2500 / 7000 = 35.7%

Yes, the banner clickers are almost twice as likely to return (66.7% vs 35.7%)

e) P(A) x P(B) = 0.30 x 0.45 = 0.135
P(A and B) = 0.20

The events are not independent, because 0.20 is not 0.135
What it means for business: Banner-clicking and returning are related and clicking the banner influences people to come back

QUESTION 2

Part 1:

```
import numpy as np

np.random.seed(42)

# Part 1: Dataset Creation
# 1. Randomly assign 15 participants a fitness level
print("PART 1: DATASET CREATION")
print("=" * 50)
fitness_levels = np.random.choice([0,1,2], size=15, p=[1/3,1/3,1/3])

# 2-4. Generate step counts based on fitness level and store in array
dataOfSteps = np.zeros((15,14))

for x, fitness_level in enumerate(fitness_levels):
    if fitness_level == 0:
        mean, sd = 6000, 600
    elif fitness_level == 1:
        mean, sd = 7500, 500
    else:
        mean, sd = 9000, 700

    # generate 14 days of step counts
    data = np.random.normal(mean, sd, 14)
    data = np.round(data).astype(int)
    data = np.clip(data, 3000, 15000)
    dataOfSteps[x] = data

# 5. Display fitness levels and dataset
print("FITNESS LEVEL ASSIGNMENTS")
print("-----------------------------------------")
print("Participant|Fitness Level| Description")
print("-" * 40)

for x, fitness_level in enumerate(fitness_levels):
    if fitness_level == 0:
        description = "Low"
    elif fitness_level == 1:
        description = "Moderate"
    else:
        description = "High"
    print(f"{x + 1:11}|{fitness_level:13}|{description}")

print("\nDAILY STEP COUNTS (14 days per participant)")
print("=" * 100)

for x in range(15):
    if fitness_levels[x] == 0:
        level1 = "Low"
    elif fitness_levels[x] == 1:
        level1 = "Moderate"
    else:
        level1 = "High"

    steps_str = " | ".join(f"{step:5}" for step in dataOfSteps[x])
    print(f"P{x + 1:2} ({level1:8}) | {steps_str}")
```

Part 2:

```
# Part 2: Data Analysis
print("\n" + "=" * 50)
print("PART 2: DATA ANALYSIS")
print("=" * 50)

# a. Calculate the average daily steps per participant. Sort these averages in descending order
#    and display the top 5 participants along with their averages.
avg_steps = np.mean(dataOfSteps, axis=1)
grouped_participants = list(enumerate(avg_steps))
sorted_participants = sorted(grouped_participants, key=lambda x: x[1], reverse=True)

print("a) AVERAGE DAILY STEPS PER PARTICIPANT (Top 5)")
print("-" * 50)
print("Rank | Participant | Fitness Level | Average Steps")
print("-" * 50)

for rank, (idx, average_steps) in enumerate(sorted_participants[:5], 1):
    fitness_level = fitness_levels[idx]
    level_description = ["Low", "Moderate", "High"][fitness_level]
    print(f"{rank:4} | P{idx + 1:11} | {level_description:12} | {average_steps:13.2f}")

# b. Calculate the overall mean and standard deviation of all step counts combined.
#    Round both to the nearest whole number.
overall_mean = np.mean(dataOfSteps)
overall_std = np.std(dataOfSteps)
overall_mean_round = np.round(overall_mean).astype(int)
overall_std_round = np.round(overall_std).astype(int)

print(f"\nb) OVERALL MEAN AND STANDARD DEVIATION")
print(f"Mean of all steps: {overall_mean_round}")
print(f"Standard Deviation: {overall_std_round}")

# c. Compute the median daily steps for each participant. Identify and display the participant(s)
#    with the highest and lowest median values, including their participant numbers and median values.
median_dailySteps = np.median(dataOfSteps, axis=1)

print(f"\nc) MEDIAN DAILY STEPS PER PARTICIPANT")
print("Participant | Fitness Level | Median Steps")
print("-" * 45)

for p in range(15):
    fitness_level = fitness_levels[p]
    level_description = ["Low", "Moderate", "High"][fitness_level]
    print(f"P{p+1:10} | {level_description:12} | {median_dailySteps[p]:12.2f}")

# Find highest and lowest median
max_median = np.max(median_dailySteps)
min_median = np.min(median_dailySteps)
max_indices = np.where(median_dailySteps == max_median)[0]
min_indices = np.where(median_dailySteps == min_median)[0]

print(f"\nHIGHEST MEDIAN: {max_median:.2f} steps")
```

```python
for idx in max_indices:
    level = ["Low", "Moderate", "High"][fitness_levels[idx]]
    print(f"  - Participant P{idx+1} ({level})")

print(f"LOWEST MEDIAN: {min_median:.2f} steps")
for idx in min_indices:
    level = ["Low", "Moderate", "High"][fitness_levels[idx]]
    print(f"  - Participant P{idx+1} ({level})")

# d. Count and display how many participants have an average daily step
count above 8000 over the 14 days.
over_8000 = avg_steps > 8000
high_activity_indices = np.where(over_8000)[0]
high_activity_count = len(high_activity_indices)

print(f"\nd) PARTICIPANTS WITH AVERAGE > 8000 STEPS: {high_activity_count}
participants")
print("Participant | Fitness Level | Average Steps")
print("-" * 45)

for idx in high_activity_indices:
    fitness_level = fitness_levels[idx]
    level_description = ["Low", "Moderate", "High"][fitness_level]
    print(f"P{idx+1:10} | {level_description:12} | {avg_steps[idx]:13.2f}")

# e. Compute and display the 25th, 50th (median), and 75th percentiles of
all step counts
#    combined across all participants and days.
percentiles = np.percentile(dataOfSteps, [25, 50, 75])

print(f"\ne) PERCENTILES OF ALL STEP COUNTS")
print(f"25th Percentile: {percentiles[0]:.2f} steps")
print(f"50th Percentile (Median): {percentiles[1]:.2f} steps")
print(f"75th Percentile: {percentiles[2]:.2f} steps")

print("\n" + "=" * 50)
print("ANALYSIS COMPLETE!")
print("=" * 50)
```

Figure 2.1:

```
/opt/anaconda3/bin/python /Users/macdee/PycharmProjects/PythonProject6/fitness.py
PART 1: DATASET CREATION
==================================================
FITNESS LEVEL ASSIGNMENTS
----------------------------------------
Participant|Fitness Level| Description
----------------------------------------
          1|            1|Moderate
          2|            2|High
          3|            2|High
          4|            1|Moderate
          5|            0|Low
          6|            0|Low
          7|            0|Low
          8|            2|High
          9|            1|Moderate
         10|            2|High
         11|            0|Low
         12|            2|High
         13|            2|High
         14|            0|Low
         15|            0|Low
```

Figure 2.2

```
DAILY STEP COUNTS (14 days per participant)
====================================================================================================
P 1 (Moderate) | 7214.0 | 7038.0 | 6194.0 | 7975.0 | 7908.0 | 6738.0 | 7286.0 | 7129.0 | 7148.0 | 6430.0 | 7185.0 | 7799.0 | 8780.0 | 7697.0
P 2 (High    ) | 9086.0 | 8639.0 | 8580.0 | 9663.0 | 9204.0 | 8555.0 | 8285.0 | 8887.0 | 8626.0 | 8996.0 | 8839.0 | 9273.0 | 8114.0 | 9764.0
P 3 (High    ) | 10945.0 | 9836.0 | 9153.0 | 9617.0 | 8294.0 | 7892.0 | 9542.0 | 8623.0 | 8057.0 | 8384.0 | 8209.0 | 9094.0 | 9407.0 | 9621.0
P 4 (Moderate) | 7947.0 | 7877.0 | 7396.0 | 7188.0 | 6746.0 | 8050.0 | 7411.0 | 7295.0 | 8090.0 | 7051.0 | 7917.0 | 7648.0 | 6981.0 | 7462.0
P 5 (Low     ) | 6584.0 | 6477.0 | 6897.0 | 6203.0 | 8023.0 | 5448.0 | 5761.0 | 5963.0 | 5149.0 | 6625.0 | 6542.0 | 6011.0 | 5679.0 | 5103.0
P 6 (Low     ) | 5526.0 | 6446.0 | 5873.0 | 5744.0 | 6301.0 | 6695.0 | 6154.0 | 6189.0 | 6823.0 | 6105.0 | 5814.0 | 6404.0 | 5846.0 | 5779.0
P 7 (Low     ) | 6764.0 | 5825.0 | 4407.0 | 6207.0 | 5763.0 | 5827.0 | 6272.0 | 5900.0 | 6129.0 | 4787.0 | 5434.0 | 6842.0 | 5989.0 | 4996.0
P 8 (High    ) | 8249.0 | 8305.0 | 9072.0 | 8697.0 | 8539.0 | 9003.0 | 9334.0 | 8819.0 | 8598.0 | 8705.0 | 9238.0 | 8995.0 | 9537.0 | 8195.0
P 9 (Moderate) | 7112.0 | 7887.0 | 7099.0 | 8192.0 | 8203.0 | 8196.0 | 7060.0 | 7538.0 | 7253.0 | 7962.0 | 8353.0 | 7937.0 | 7505.0 | 7317.0
P10 (High    ) | 9454.0 | 8144.0 | 9375.0 | 8360.0 | 9434.0 | 8887.0 | 8728.0 | 8380.0 | 8750.0 | 9389.0 | 9731.0 | 9369.0 | 9955.0 | 10777.0
P11 (Low     ) | 5805.0 | 5876.0 | 5136.0 | 6714.0 | 6780.0 | 5480.0 | 6371.0 | 6730.0 | 6136.0 | 6508.0 | 6105.0 | 5270.0 | 6630.0 | 6795.0
P12 (High    ) | 9514.0 | 8332.0 | 8474.0 | 8209.0 | 9539.0 | 9888.0 | 9297.0 | 9658.0 | 8393.0 | 9102.0 | 8041.0 | 8460.0 | 9615.0 | 8832.0
P13 (High    ) | 9847.0 | 9377.0 | 10914.0 | 9066.0 | 8016.0 | 8976.0 | 8326.0 | 9684.0 | 9029.0 | 8904.0 | 8913.0 | 9518.0 | 8683.0 | 9544.0
P14 (Low     ) | 6627.0 | 5795.0 | 5444.0 | 5692.0 | 6426.0 | 6055.0 | 6378.0 | 7058.0 | 6139.0 | 5515.0 | 6634.0 | 6031.0 | 6523.0 | 6640.0
P15 (Low     ) | 5425.0 | 6829.0 | 6543.0 | 5638.0 | 6183.0 | 6154.0 | 6014.0 | 6523.0 | 6862.0 | 6004.0 | 6799.0 | 6593.0 | 6139.0 | 6106.0

PythonProject6 > fitness.py                143:16 (3581 chars, 86 line breaks)  LF  UTF-8  4 spaces  /opt/anaconda3
```

Figure 2.3:

Run Analysis-softwareEngineers × fitness ×

```
==================================================
PART 2: DATA ANALYSIS
==================================================
a) AVERAGE DAILY STEPS PER PARTICIPANT (Top 5)
--------------------------------------------------
Rank | Participant | Fitness Level | Average Steps
--------------------------------------------------
   1 | P        13 | High         |       9199.79
   2 | P        10 | High         |       9195.21
   3 | P         3 | High         |       9048.14
   4 | P        12 | High         |       8953.86
   5 | P         2 | High         |       8893.64

b) OVERALL MEAN AND STANDARD DEVIATION
Mean of all steps: 7557
Standard Deviation: 1424
```

```
c) MEDIAN DAILY STEPS PER PARTICIPANT
Participant | Fitness Level | Median Steps
---------------------------------------------
P        1 | Moderate     |      7199.50
P        2 | High         |      8863.00
P        3 | High         |      9123.50
P        4 | Moderate     |      7436.50
P        5 | Low          |      6107.00
P        6 | Low          |      6129.50
P        7 | Low          |      5863.50
P        8 | High         |      8762.00
P        9 | Moderate     |      7712.50
P       10 | High         |      9372.00
P       11 | Low          |      6253.50
P       12 | High         |      8967.00
P       13 | High         |      9047.50
P       14 | Low          |      6258.50
P       15 | Low          |      6168.50
```

```
HIGHEST MEDIAN: 9372.00 steps
  - Participant P10 (High)
LOWEST MEDIAN: 5863.50 steps
  - Participant P7 (Low)

d) PARTICIPANTS WITH AVERAGE > 8000 STEPS: 6 participants
Participant | Fitness Level | Average Steps
---------------------------------------------
P        2 | High         |       8893.64
P        3 | High         |       9048.14
P        8 | High         |       8806.14
P       10 | High         |       9195.21
P       12 | High         |       8953.86
P       13 | High         |       9199.79

e) PERCENTILES OF ALL STEP COUNTS
25th Percentile: 6318.50 steps
50th Percentile (Median): 7483.50 steps
75th Percentile: 8744.50 steps

==================================================
ANALYSIS COMPLETE!
==================================================

Process finished with exit code 0
```

PythonProject6 > fitness.py   143:16 (3581 chars, 86 line breaks)   LF   UTF-8   4 spaces   /opt/anaconda3

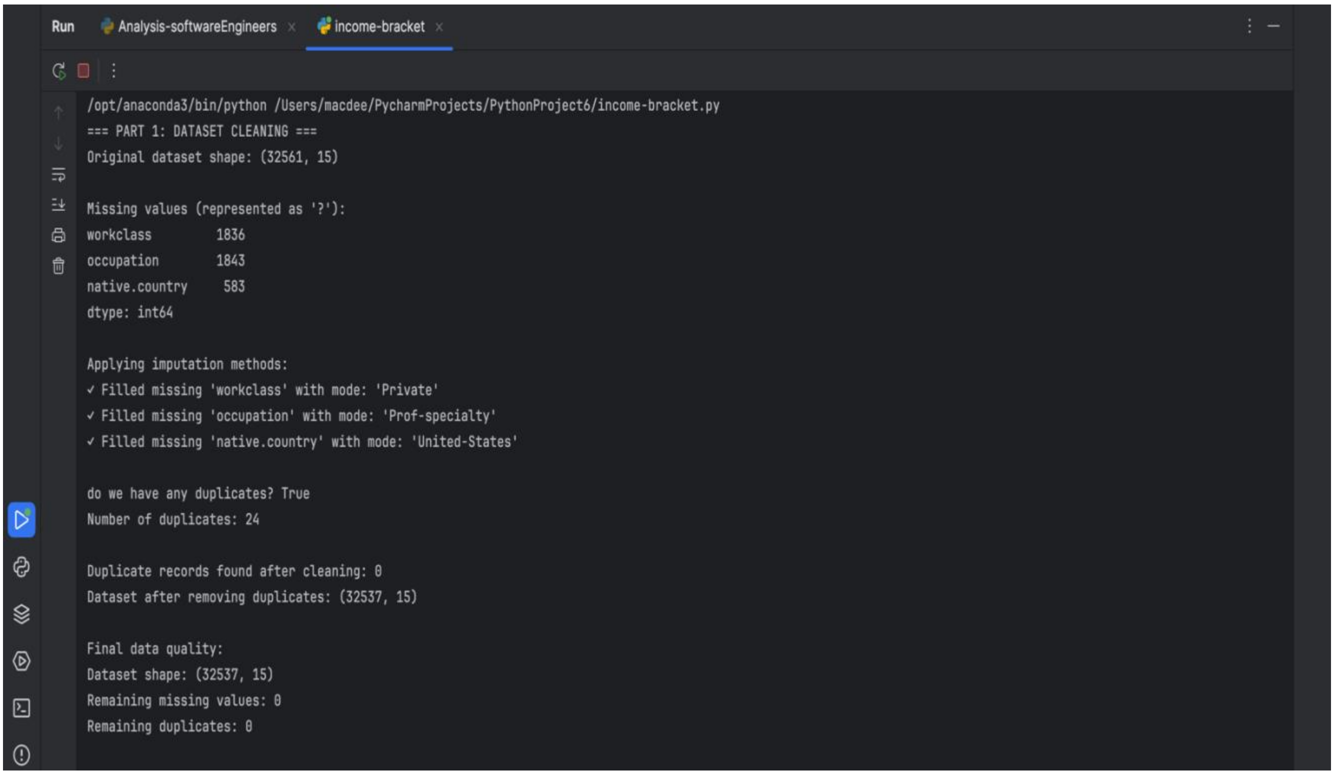
QUESTION 3

Part 1: Dataset preparation and Cleaning

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

# PART 1: DATASET PREPARATION AND CLEANING

# 1. Import the dataset using the Pandas library
Data = pd.read_csv("//Users//macdee//Downloads//adult.csv")
df = pd.DataFrame(Data)

print("=== PART 1: DATASET CLEANING ===")
print(f"Original dataset shape: {df.shape}")

# 2. Locate and address missing data points (indicated by "?" values)
print("\nMissing values (represented as '?'):")
missing = (df == '?').sum()
print(missing[missing > 0])


# Replace '?' with NaN first
df_clean = df.replace('?', np.nan)

# 3. Apply appropriate imputation methods based on data type:

print("\nApplying imputation methods:")

# • Categorical variables: Replace missing values with the most frequently
occurring value (mode)
categorical_cols = ['workclass', 'occupation', 'native.country']
for col in categorical_cols:
    if df_clean[col].isnull().sum() > 0:
        mode_value = df_clean[col].mode()[0]
        df_clean[col] = df_clean[col].fillna(mode_value)
        print(f"✓ Filled missing '{col}' with mode: '{mode_value}'")

# Numerical variables: Replace missing values with the middle value
(median)

numerical_cols = ['age', 'education.num', 'hours.per.week', 'capital.gain',
'capital.loss']
for col in numerical_cols:
    if df_clean[col].isnull().sum() > 0:
        median_value = df_clean[col].median()
        df_clean[col] = df_clean[col].fillna(median_value)
        print(f"✓ Filled missing '{col}' with median: {median_value}")

# 4. Eliminate any irrelevant entries or duplicate records
#check if there's any duplicates
print(f"\ndo we have any duplicates? {df.duplicated().any()}")
print(f"Number of duplicates: {df.duplicated().sum()}")  # ← CHANGED to df
instead of df_clean
df_clean = df_clean.drop_duplicates()
```

```
print(f"\nDuplicate records found after cleaning:
{df_clean.duplicated().sum()}")
df_clean = df_clean.drop_duplicates()
print(f"Dataset after removing duplicates: {df_clean.shape}")

# Final quality check
print(f"\nFinal data quality:")
print(f"Dataset shape: {df_clean.shape}")
print(f"Remaining missing values: {df_clean.isnull().sum().sum()}")
print(f"Remaining duplicates: {df_clean.duplicated().sum()}")
```

Figure 3.1



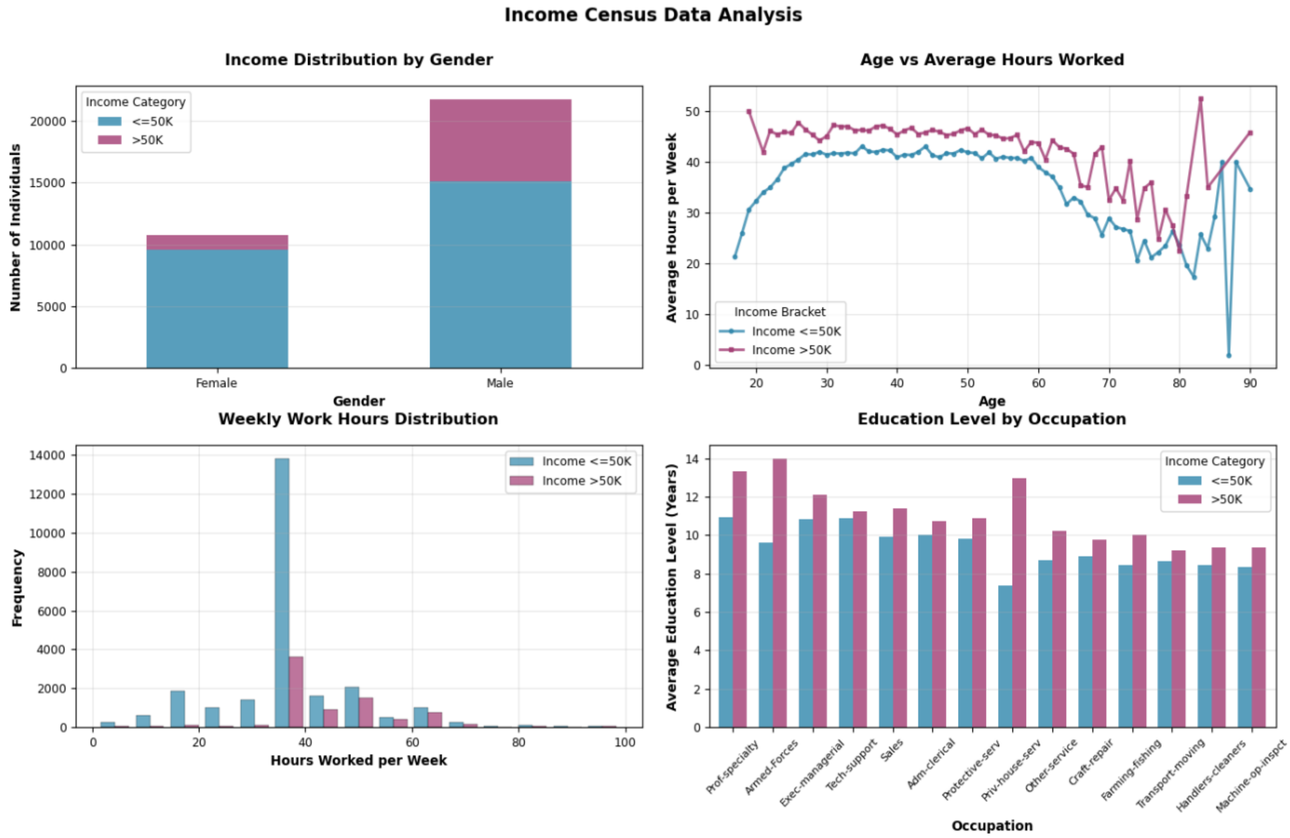
Part 2: Data Virtualisation

```
# Part 2: Data Visualization with SMALLER, CLEANER LAYOUT
print("\n" + "=" * 50)
print("PART 2: DATA VISUALIZATION")
print("=" * 50)

# Define color scheme
LOW_INCOME_COLOR = '#2E86AB'    # Blue for <=50K
```

```
HIGH_INCOME_COLOR = '#A23B72'  # Burgundy for >50K

# Create SMALLER 2x2 subplot grid
fig, axes = plt.subplots(2, 2, figsize=(16, 12))  # Reduced from (20,16)

# 1. Top-Left: Stacked Bar Chart - Income Distribution by Gender
income_gender = df_clean.groupby(['sex', 'income']).size().unstack()

income_gender.plot(kind='bar', stacked=True, ax=axes[0,0],
                   color=[LOW_INCOME_COLOR, HIGH_INCOME_COLOR],
                   alpha=0.8)

axes[0,0].set_title('Income Distribution by Gender', fontsize=12,
fontweight='bold', pad=15)
axes[0,0].set_xlabel('Gender', fontsize=10, fontweight='bold')
axes[0,0].set_ylabel('Number of Individuals', fontsize=10,
fontweight='bold')
axes[0,0].tick_params(axis='x', rotation=0, labelsize=9)
axes[0,0].tick_params(axis='y', labelsize=9)
axes[0,0].legend(title='Income Category', title_fontsize=9, fontsize=9)
axes[0,0].grid(axis='y', alpha=0.3)

# 2. Top-Right: Line Graph - Age vs Average Hours Worked
age_hours = df_clean.groupby(['age',
'income'])['hours.per.week'].mean().reset_index()
low_income_age = age_hours[age_hours['income'] == '<=50K']
high_income_age = age_hours[age_hours['income'] == '>50K']

axes[0,1].plot(low_income_age['age'], low_income_age['hours.per.week'],
               color=LOW_INCOME_COLOR, marker='o', linewidth=2,
               markersize=3, label='Income <=50K', alpha=0.8)
axes[0,1].plot(high_income_age['age'], high_income_age['hours.per.week'],
               color=HIGH_INCOME_COLOR, marker='s', linewidth=2,
               markersize=3, label='Income >50K', alpha=0.8)

axes[0,1].set_title('Age vs Average Hours Worked', fontsize=12,
fontweight='bold', pad=15)
axes[0,1].set_xlabel('Age', fontsize=10, fontweight='bold')
axes[0,1].set_ylabel('Average Hours per Week', fontsize=10,
fontweight='bold')
axes[0,1].tick_params(labelsize=9)
axes[0,1].legend(title='Income Bracket', title_fontsize=9, fontsize=9)
axes[0,1].grid(True, alpha=0.3)

# 3. Bottom-Left: Histogram - Distribution of Weekly Work Hours
low_income_hours = df_clean[df_clean['income'] ==
'<=50K']['hours.per.week']
high_income_hours = df_clean[df_clean['income'] ==
'>50K']['hours.per.week']

axes[1,0].hist([low_income_hours, high_income_hours], bins=15, alpha=0.7,
# Reduced bins
               color=[LOW_INCOME_COLOR, HIGH_INCOME_COLOR],
               label=['Income <=50K', 'Income >50K'],
               edgecolor='black', linewidth=0.3)  # Thinner edges

axes[1,0].set_title('Weekly Work Hours Distribution', fontsize=12,
fontweight='bold', pad=15)
axes[1,0].set_xlabel('Hours Worked per Week', fontsize=10,
fontweight='bold')
axes[1,0].set_ylabel('Frequency', fontsize=10, fontweight='bold')
```

```
axes[1,0].tick_params(labelsize=9)
axes[1,0].legend(fontsize=9)
axes[1,0].grid(True, alpha=0.3)

# 4. Bottom-Right: Grouped Bar Chart - Education Level by Occupation
education_occupation = df_clean.groupby(['occupation',
'income'])['education.num'].mean().unstack()
occupation_sorted =
education_occupation.mean(axis=1).sort_values(ascending=False).index
education_occupation_sorted = education_occupation.loc[occupation_sorted]

# Plot with SMALLER bars and labels
bar_width = 0.7  # Narrower bars
education_occupation_sorted.plot(kind='bar', ax=axes[1,1],
                                 color=[LOW_INCOME_COLOR,
HIGH_INCOME_COLOR],
                                 alpha=0.8, width=bar_width)

axes[1,1].set_title('Education Level by Occupation', fontsize=12,
fontweight='bold', pad=15)
axes[1,1].set_xlabel('Occupation', fontsize=10, fontweight='bold')
axes[1,1].set_ylabel('Average Education Level (Years)', fontsize=10,
fontweight='bold')
axes[1,1].tick_params(axis='x', rotation=45, labelsize=8)  # Smaller font
for x-axis
axes[1,1].tick_params(axis='y', labelsize=9)
axes[1,1].legend(title='Income Category', title_fontsize=9, fontsize=9)
axes[1,1].grid(axis='y', alpha=0.3)

# Final adjustments with MORE SPACING
plt.suptitle('Income Census Data Analysis', fontsize=14, fontweight='bold',
y=0.98)
plt.tight_layout(pad=3.0, h_pad=2.5, w_pad=2.5)  # Reduced padding

print("Visualization completed with optimized sizing!")
plt.show()
```

Figure 3.2



QUESTION 4

```
from cProfile import label

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from more_itertools.recipes import reshape
from sklearn.linear_model import LinearRegression
from sympy.abc import alpha

software_engineer = {
    'Years_Experience': [1, 2, 3, 4, 5, 6, 7, 8, 10 , 12 ,13 ,15 ,16 ,18
,20],
    'Education_Level': [0, 0, 1, 1, 0, 2, 1, 2, 1, 2, 0, 1, 2, 0, 1],
    'Location' : [0, 1, 1, 2, 0, 2, 2, 1, 1, 0, 2, 2, 1, 0, 1],
    'Salary' : [48, 53, 60, 65, 68, 80, 78, 88, 90, 100, 92, 105, 108, 115,
120]
}
df = pd.DataFrame(software_engineer)


#Mapping
education = {0: "Bachelor's", 1:"Masters", 2:"PHD"}
location_mapping = {0:"Remote", 1:"On-site", 2:"Hybrid"}
education_colors = {0: '#2E86AB', 1: '#A23B72', 2: '#F9C80E'}
```

```
markers = ['o', 's', '^']
colours = ['blue', 'green', 'red']

#creating a scatter plot



df['Education'] = df['Education_Level'].map(education)

#years of experience vs salary

plt.figure(figsize=(10,6))
for x, edu_level in enumerate([0,1,2]):
    for y, loc_level in enumerate ([0,1,2]):
        mask = (df['Education_Level'] == edu_level) & (df['Location'] ==
loc_level)
        plt.scatter(df[mask]['Years_Experience'], df[mask]['Salary'],
                    marker = markers[x], c=colours[y], s=80 )
plt.title("Salary vs Years of Experience")
plt.xlabel('Years of Experience')
plt.ylabel('Salary ($ thousands)')

#question 3- create a line fir for each locationprint("Data points by
location:")
for loc in [0,1,2]:
    mask = df['Location'] == loc
    location_data = df[mask]
    print(f"{location_mapping[loc]}: {len(location_data)} data points")
    print(f"  Experience: {list(location_data['Years_Experience'])}")
    print(f"  Salary: {list(location_data['Salary'])}")
    print()

# Now add the trend lines
for loc in [0,1,2]:
    mask = df['Location'] == loc
    x_data = df[mask]['Years_Experience'].values.reshape(-1, 1)
    y_data = df[mask]['Salary'].values

    # Only create line if we have data points
    if len(x_data) > 0:
        # Fit linear regression model
        model = LinearRegression()
        model.fit(x_data, y_data)

        # Create points for the line
        x_line = np.linspace(df['Years_Experience'].min(),
df['Years_Experience'].max(), 100).reshape(-1, 1)
        y_line = model.predict(x_line)

        # Plot the line of best fit
        plt.plot(x_line, y_line,
                 color=colours[loc],
                 linestyle='--',
                 linewidth=3,
                 label=f'{location_mapping[loc]}',
                 alpha=0.8)

# Customize the plot
plt.title("Salary vs Years of Experience with Trend Lines by Location")
plt.xlabel('Years of Experience')
plt.ylabel('Salary ($ thousands)')
```

```
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left')
plt.grid(True, alpha=0.3)
plt.tight_layout()
plt.show()

# Print the regression equations
print("\nRegression Equations for Each Location:")
print("=" * 45)
for loc in [0,1,2]:
    mask = df['Location'] == loc
    if len(df[mask]) > 0:
        x_data = df[mask]['Years_Experience'].values.reshape(-1, 1)
        y_data = df[mask]['Salary'].values

        model = LinearRegression()
        model.fit(x_data, y_data)

        print(f"{location_mapping[loc]}: Salary = {model.intercept_:.1f} +
{model.coef_[0]:.2f}×Experience")
        print(f"  (Each year adds ${model.coef_[0]:.2f}K to salary)")

print("QUESTION 4: Separate Regression Lines for Each Location")
print("=" * 55)
print("Fitting separate lines using ONLY experience as independent
variable\n")

# Fit separate regression lines for each location
for loc in [0, 1, 2]:
    # Filter data for this location only
    mask = df['Location'] == loc
    location_data = df[mask]

    # Extract ONLY experience (independent variable) and salary (dependent
variable)
    X = location_data[['Years_Experience']]  # Independent variable:
Experience only
    y = location_data['Salary']  # Dependent variable: Salary

    print(f"\n{location_mapping[loc]} Location Analysis:")
    print(f"Data points: {len(X)}")
    print(f"Experience values: {list(X['Years_Experience'])}")
    print(f"Salary values: {list(y)}")

    # Fit linear regression using ONLY experience
    model = LinearRegression()
    model.fit(X, y)

    # Make predictions for the line
    x_range = np.linspace(0, 21, 100).reshape(-1, 1)
    y_pred = model.predict(x_range)

    # Display the regression equation
    print(f"Regression Equation: Salary = {model.intercept_:.2f} +
{model.coef_[0]:.2f} × Experience")
    print(f"R-squared: {model.score(X, y):.3f}")
    print(f"Interpretation: Each additional year of experience adds
${model.coef_[0]:.2f}K to salary")

print("\n" + "=" * 55)
print("SUMMARY: All three models use ONLY experience to predict salary")
print("Each location has its own separate regression equation!")
```

```python
#Question 5: Multiple Linear Regression Model
# Question 5: Multiple Linear Regression Model
print('===MULTIPLE LINEAR REGRESSION===')
# PREPARE THE FEATURES (INDEPENDENT VARIABLES)
df['Experience2'] = df['Years_Experience'] ** 2  # Experience squared
print(df[['Years_Experience', 'Experience2', 'Education_Level', 'Location',
'Salary']])
print()

# Define independent variables (X) and target variable (y)
X = df[['Years_Experience', 'Experience2', 'Education_Level', 'Location']]
# input features
y = df['Salary']  # This is what we're predicting - USE LOWERCASE y!

print("Independent Variables (X):")
print(X)
print()
print("Target Variable (y):")
print(y)
print()

# Create and train multiple linear regression model
multiple_model = LinearRegression()
multiple_model.fit(X, y)  # Make sure you're using lowercase y here!

print('====MULTIPLE LINEAR REGRESSION RESULTS:====')
print(f"Model intercept: {multiple_model.intercept_:.2f}")
print('\nCoefficients - How much each variable affects salary:')  # Fixed:
/n to \n
names = ['Years_Experience', 'Experience^2', 'Education_Level', 'Location']

for i, (name, coef) in enumerate(zip(names, multiple_model.coef_)):
    print(f" {name}: {coef:.4f}")

print("\nIMPACT ANALYSIS")  # Added \n for spacing
print("================")

# Compare impact fairly, normalize coefficients
X_normalised = (X - X.mean()) / X.std()
normalised_model = LinearRegression()
normalised_model.fit(X_normalised, y)  # Use lowercase y here too!

normalised_coefs = normalised_model.coef_
print("Normalized Coefficients (for fair comparison):")
for name, norm_coef in zip(names, normalised_coefs):
    print(f"  {name}: {abs(norm_coef):.4f}")

# Finding which variable has the most impact
max_impact_idx = np.argmax(abs(normalised_coefs))
max_impact_feature = names[max_impact_idx]
max_impact_value = abs(normalised_coefs[max_impact_idx])

print(f"\nVARIABLE WITH GREATEST IMPACT: {max_impact_feature}")
print(f"   Impact score: {max_impact_value:.4f}")

# Model performance
r_squared = multiple_model.score(X, y)  # Use lowercase y here!
print(f"\nMODEL PERFORMANCE:")
print(f"R-squared: {r_squared:.4f}")
print(f"This means the model explains {r_squared*100:.1f}% of salary
```

```
variation")

print("\n" + "=" * 70)

# QUESTION 6: Display coefficients and interpret impact
print("=" * 70)
print("QUESTION 6: Model Coefficients and Impact Analysis")
print("=" * 70)

# Displays the intercept and coefficients
print("MODEL COEFFICIENTS AND INTERCEPT:")
print("-" * 50)
print(f"Intercept: ${multiple_model.intercept_:.2f}K")
print("(Base salary when experience=0, education=Bachelor's,
location=Remote)")
print()

# Show each coefficient and what it represents
coefficients = multiple_model.coef_
feature_names = ['Years_Experience', 'Experience^2', 'Education_Level',
'Location']

print("Coefficients (salary change per unit increase):")
for name, coef in zip(feature_names, coefficients):
    print(f"  {name}: {coef:+.4f}")

# Show which variable has the greatest impact
print("\n" + "=" * 50)
print("IMPACT ANALYSIS:")
print("-" * 50)

print("Normalized Coefficients (comparing equal scales):")
normalized_coefs = normalised_model.coef_
for name, norm_coef in zip(feature_names, normalized_coefs):
    print(f"  {name}: {abs(norm_coef):.4f}")

# Display the most impactful variable
print(f"\nVARIABLE WITH GREATEST IMPACT: {max_impact_feature}")
print(f"Impact score: {max_impact_value:.4f}")

# Brief explanation of what this means
print("\nEXPLANATION OF MY FINDINGS:")
print("-" * 50)
print(f"{max_impact_feature} has the strongest influence on salary.")
print(f"This variable explains the most variation in software engineer
salaries.")

print(f"\nModel R-squared: {r_squared:.4f}")
print(f"The model explains {r_squared*100:.1f}% of salary variation.")
print("=" * 70)

# QUESTION 7: Predict salaries for new software engineers
print("=" * 70)
print("QUESTION 7: Salary Predictions")
print("=" * 70)

# Create the first case: 9 years, Master's (1), Remote (0)
print("CASE 1:")
print("-" * 30)
# Master's degree = 1, Remote = 0, Experience squared = 9^2 = 81
case1_features = np.array([[9, 81, 1, 0]])  # [Experience, Experience^2,
```

```
Education, Location]
case1_prediction = multiple_model.predict(case1_features)[0]
print("Software Engineer Profile:")
print("• 9 years of experience")
print("• Master's degree")
print("• Remote work location")
print(f"PREDICTED SALARY: ${case1_prediction:.2f}K")
print()

# Create the second case: 14 years, PhD (2), Hybrid (2)
print("CASE 2:")
print("-" * 30)
# PhD = 2, Hybrid = 2, Experience squared = 14^2 = 196
case2_features = np.array([[14, 196, 2, 2]])  # [Experience, Experience^2,
Education, Location]
case2_prediction = multiple_model.predict(case2_features)[0]
print("Software Engineer Profile:")
print("• 14 years of experience")
print("• PhD degree")
print("• Hybrid work location")
print(f"PREDICTED SALARY: ${case2_prediction:.2f}K")
print()

# Compare the two predictions
print("COMPARISON:")
print("-" * 30)
salary_difference = case2_prediction - case1_prediction
print(f"Salary difference: ${salary_difference:.2f}K")
print("The more experienced engineer with PhD earns significantly more.")
print("=" * 70)

# QUESTION 8: Create combined graph with everything
print("=" * 70)
print("QUESTION 8: Combined Graph with Predictions")
print("=" * 70)

# Create a new figure for the combined graph
plt.figure(figsize=(12, 8))

# PART 1: Plot the original scatter points (from Questions 1-2)
print("Plotting original data points...")
for x, edu_level in enumerate([0, 1, 2]):
    for y, loc_level in enumerate([0, 1, 2]):
        # Find data points with specific education and location
        mask = (df['Education_Level'] == edu_level) & (df['Location'] ==
loc_level)
        plt.scatter(df[mask]['Years_Experience'], df[mask]['Salary'],
                    marker=markers[x],
                    c=colours[y],
                    s=100,
                    alpha=0.7,
                    label=f'{education[edu_level]},
{location_mapping[loc_level]}')

# PART 2: Add lines of best fit for each location (from Question 3)
print("Adding trend lines for each location...")
for loc in [0, 1, 2]:
    # Get data for this location only
    mask = df['Location'] == loc
    x_data = df[mask]['Years_Experience'].values.reshape(-1, 1)
    y_data = df[mask]['Salary'].values
```

```
    # Only create line if we have data points
    if len(x_data) > 0:
        # Fit the trend line
        model = LinearRegression()
        model.fit(x_data, y_data)

        # Create smooth line for plotting
        x_line = np.linspace(df['Years_Experience'].min(),
df['Years_Experience'].max(), 100).reshape(-1, 1)
        y_line = model.predict(x_line)

        # Plot the trend line
        plt.plot(x_line, y_line,
                 color=colours[loc],
                 linestyle='--',
                 linewidth=2,
                 label=f'{location_mapping[loc]} trend',
                 alpha=0.8)

# PART 3: Add the predicted points from Question 7 (as special stars)
print("Adding predicted salary points...")

# Plot prediction 1: 9 years, Master's, Remote
plt.scatter(9, case1_prediction,
            marker='*',  # Star marker
            s=300,  # Large size
            c='gold',  # Gold color
            edgecolors='black',  # Black border
            linewidth=2,
            label='Prediction: 9yrs, Master\'s, Remote',
            zorder=10)  # Make sure it appears on top

# Plot prediction 2: 14 years, PhD, Hybrid
plt.scatter(14, case2_prediction,
            marker='*',  # Star marker
            s=300,  # Large size
            c='purple',  # Purple color
            edgecolors='black',  # Black border
            linewidth=2,
            label='Prediction: 14yrs, PhD, Hybrid',
            zorder=10)  # Make sure it appears on top

# Customize the graph appearance
plt.title("Software Engineer Salary Analysis\n(Experience vs Salary with
Predictions)", fontsize=14)
plt.xlabel('Years of Experience', fontsize=12)
plt.ylabel('Salary ($ thousands)', fontsize=12)
plt.grid(True, alpha=0.3)

# Create a comprehensive legend
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left', fontsize=10)

# Adjust layout and display
plt.tight_layout()
plt.show()

print("Combined graph created successfully!")
print("• Blue/Green/Red circles/squares/triangles = Original data points")
print("• Dashed lines = Trend lines for each location")
print("• Gold star = Prediction for 9 years, Master's, Remote")
```

```python
print("• Purple star = Prediction for 14 years, PhD, Hybrid")
print("=" * 70)
```

OUTPUT OF THE CODE

/opt/anaconda3/bin/python
/Users/macdee/PycharmProjects/PythonProject6/Analysis-softwareEngineers.py

Remote: 4 data points

  Experience: [1, 5, 12, 18]

  Salary: [48, 68, 100, 115]

On-site: 6 data points

  Experience: [2, 3, 8, 10, 16, 20]

  Salary: [53, 60, 88, 90, 108, 120]

Hybrid: 5 data points

  Experience: [4, 6, 7, 13, 15]

  Salary: [65, 80, 78, 92, 105]

Regression Equations for Each Location:

==

Remote: Salary = 46.8 + 3.99×Experience

  (Each year adds $3.99K to salary)

On-site: Salary = 50.9 + 3.62×Experience

  (Each year adds $3.62K to salary)

Hybrid: Salary = 56.3 + 3.08×Experience

  (Each year adds $3.08K to salary)

QUESTION 4: Separate Regression Lines for Each Location

==

Fitting separate lines using ONLY experience as independent variable

Remote Location Analysis:

Data points: 4

Experience values: [1, 5, 12, 18]

Salary values: [48, 68, 100, 115]

Regression Equation: Salary = 46.80 + 3.99 × Experience

R-squared: 0.982

Interpretation: Each additional year of experience adds $3.99K to salary

/opt/anaconda3/lib/python3.12/site-packages/sklearn/base.py:493: UserWarning: X does not have valid feature names, but LinearRegression was fitted with feature names

  warnings.warn(

/opt/anaconda3/lib/python3.12/site-packages/sklearn/base.py:493: UserWarning: X does not have valid feature names, but LinearRegression was fitted with feature names

  warnings.warn(

/opt/anaconda3/lib/python3.12/site-packages/sklearn/base.py:493: UserWarning: X does not have valid feature names, but LinearRegression was fitted with feature names

  warnings.warn(

On-site Location Analysis:

Data points: 6

Experience values: [2, 3, 8, 10, 16, 20]

Salary values: [53, 60, 88, 90, 108, 120]

Regression Equation: Salary = 50.93 + 3.62 × Experience

R-squared: 0.966

Interpretation: Each additional year of experience adds $3.62K to salary

Hybrid Location Analysis:

Data points: 5

Experience values: [4, 6, 7, 13, 15]

Salary values: [65, 80, 78, 92, 105]

Regression Equation: Salary = 56.30 + 3.08 × Experience

R-squared: 0.929

Interpretation: Each additional year of experience adds $3.08K to salary

==

SUMMARY: All three models use ONLY experience to predict salary

Each location has its own separate regression equation!

===MULTIPLE LINEAR REGRESSION===

	Years_Experience	Experience2	Education_Level	Location	Salary
0	1	1	0	0	48
1	2	4	0	1	53
2	3	9	1	1	60
3	4	16	1	2	65
4	5	25	0	0	68
5	6	36	2	2	80
6	7	49	1	2	78
7	8	64	2	1	88
8	10	100	1	1	90
9	12	144	2	0	100
10	13	169	0	2	92
11	15	225	1	2	105
12	16	256	2	1	108
13	18	324	0	0	115

14	20	400	1	1	120

Independent Variables (X):

	Years_Experience	Experience2	Education_Level	Location
0	1	1	0	0
1	2	4	0	1
2	3	9	1	1
3	4	16	1	2
4	5	25	0	0
5	6	36	2	2
6	7	49	1	2
7	8	64	2	1
8	10	100	1	1
9	12	144	2	0
10	13	169	0	2
11	15	225	1	2
12	16	256	2	1
13	18	324	0	0
14	20	400	1	1

Target Variable (y):

0	48
1	53
2	60
3	65
4	68
5	80
6	78

```
7     88
8     90
9    100
10    92
11   105
12   108
13   115
14   120
Name: Salary, dtype: int64

====MULTIPLE LINEAR REGRESSION RESULTS:====
Model intercept: 44.80

Coefficients - How much each variable affects salary:
 Years_Experience: 4.8238
 Experience^2: -0.0621
 Education_Level: 3.7418
 Location: -1.0373

IMPACT ANALYSIS
=================
Normalized Coefficients (for fair comparison):
  Years_Experience: 29.2664
  Experience^2: 7.9168
  Education_Level: 2.9890
  Location: 0.8286

VARIABLE WITH GREATEST IMPACT: Years_Experience
```

Impact score: 29.2664

MODEL PERFORMANCE:

R-squared: 0.9911

This means the model explains 99.1% of salary variation

==

==

QUESTION 6: Model Coefficients and Impact Analysis

==

MODEL COEFFICIENTS AND INTERCEPT:

Intercept: $44.80K

(Base salary when experience=0, education=Bachelor's, location=Remote)

Coefficients (salary change per unit increase):

Years_Experience: +4.8238

Experience^2: -0.0621

Education_Level: +3.7418

Location: -1.0373

==

IMPACT ANALYSIS:

Normalized Coefficients (comparing equal scales):

Years_Experience: 29.2664

Experience^2: 7.9168

Education_Level: 2.9890

Location: 0.8286

VARIABLE WITH GREATEST IMPACT: Years_Experience

Impact score: 29.2664

EXPLANATION OF MY FINDINGS:

Years_Experience has the strongest influence on salary.

This variable explains the most variation in software engineer salaries.

Model R-squared: 0.9911

The model explains 99.1% of salary variation.

==

==

QUESTION 7: Salary Predictions

==

CASE 1:

Software Engineer Profile:

• 9 years of experience

• Master's degree

• Remote work location

PREDICTED SALARY: $86.93K

CASE 2:

Software Engineer Profile:

• 14 years of experience

• PhD degree

• Hybrid work location

PREDICTED SALARY: $105.58K

COMPARISON:

Salary difference: $18.65K

The more experienced engineer with PhD earns significantly more.

==

==

QUESTION 8: Combined Graph with Predictions

==

/opt/anaconda3/lib/python3.12/site-packages/sklearn/base.py:493: UserWarning: X does not have valid feature names, but LinearRegression was fitted with feature names

  warnings.warn(

/opt/anaconda3/lib/python3.12/site-packages/sklearn/base.py:493: UserWarning: X does not have valid feature names, but LinearRegression was fitted with feature names

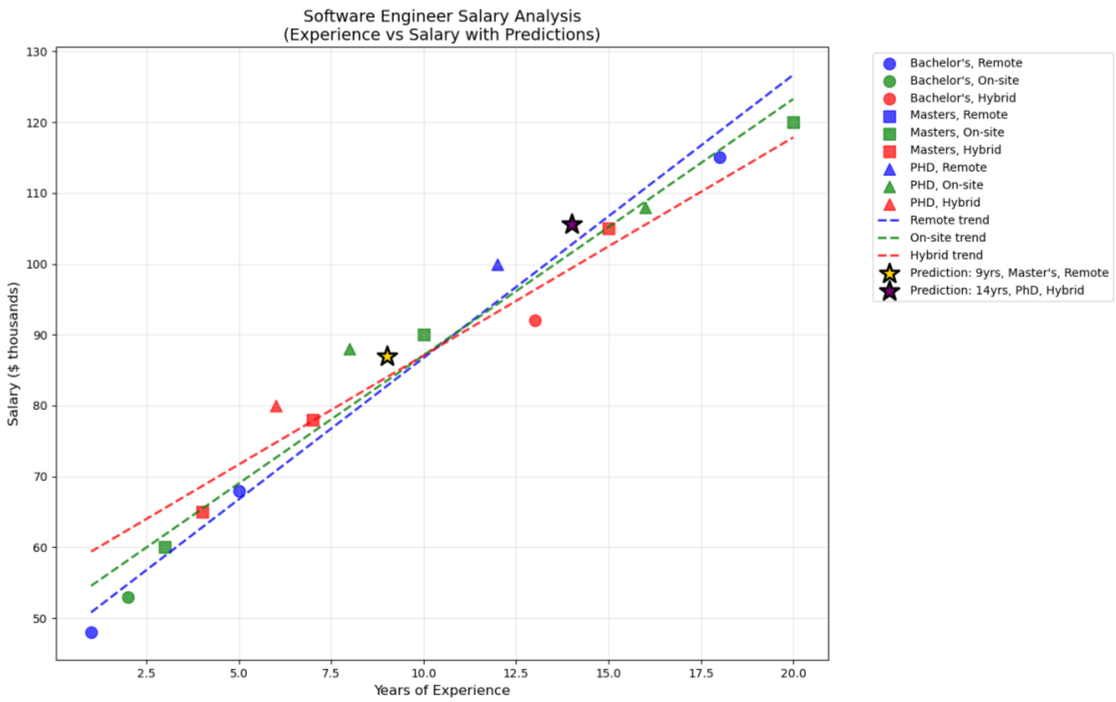
  warnings.warn(

Plotting original data points...

Adding trend lines for each location...

Adding predicted salary points...

Figure 4.1: VISUALISATION: SCATTER PLOT


Software Engineer Salary Analysis
(Experience vs Salary with Predictions)
Salary ($ thousands)
Years of Experience
130
120
110
100
90
80
70
60
50
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
Bachelor's, Remote
Bachelor's, On-site
Bachelor's, Hybrid
Masters, Remote
Masters, On-site
Masters, Hybrid
PHD, Remote
PHD, On-site
PHD, Hybrid
Remote trend
On-site trend
Hybrid trend
Prediction: 9yrs, Master's, Remote
Prediction: 14yrs, PhD, Hybrid